Applied Simulation

Modeling and Analysis using FlexSim

Fifth Edition

Compatible with FlexSim 2017 LTS

Malcolm Beaverstock, PhD

Allen Greenwood, PhD, PE

William Nordgren, MS CIM

FlexSim is chosen by the authors because of its comprehensive functionality and its ease of use. It is 3D simulation software that facilitates modeling, analysis, and visualization of systems in manufacturing, material handling, healthcare, warehousing, mining, transportation, logistics, etc. It effectively provides the means to: (1) address the physical and logical aspects inherent in simulation modeling and (2) experiment with, and analyze, simulation models. *FlexSim* closely aligns with the purpose of this text—namely to understand simulation technology and to use it to solve problems.

Exercises in the book require either an academic license of *FlexSim* or a commercial license of *FlexSim*. However, the free trial version of *FlexSim* (*FlexSim Express*) can be used for exercises in Chapters 3 and 4. Special files for the book are available in the [BookDownLoadFiles.zip] file on the FlexSim Education/ Student website. The first two chapters and the last two chapters of this textbook do not require any software, including *FlexSim*. This book is not intended as a software manual; therefore, refer to the *FlexSim* Help files for more details on software operation.

www.flexsim.com

© 2017 FlexSim Software Products, Inc.

Canyon Park Technology Center

Building A – Suite 2300

Orem, Utah 84097 USA

Phone (801) 224-6914 • Fax (801) 224-6984

sales@flexsim.com

Applied Simulation: Modeling and Analysis using FlexSim

by Malcolm Beaverstock, PhD
Allen Greenwood, PhD, PE
William Nordgren, MS CIM

Copyright © 2017 FlexSim Software Products, Inc. All rights reserved.
Printed in the United States of America

Published by FlexSim Software Products, Inc., Canyon Park Technology Center, Building A Suite 2300, Orem, UT 84097 USA.

FlexSim software and books may be purchased for educational, business, or sales promotional use. For more information, please contact our sales department: +1-801-224-6914 or *sales@flexsim.com*.

Editing: Katherine Bobo, Markus Cueva

Managing Editor: Markus Cueva

Cover Design: Stacy Geisberger, Kris Geisberger, and Ben Wilson, with special thanks to Olivier Pellegrin, Brenton King, TALUMIS, and others who provided screen captures of actual *FlexSim* simulation models.

Printing History:

January 2011: First Edition, version 1 (pre-release)
August 2011: First Edition, version 2 (pre-release)
September 2011: First Edition, version 3
January 2012: Second Edition
July 2012: Third Edition
October 2014: Fourth Edition
June 2017: Fifth Edition

The *FlexSim* logo is a registered trademark of FlexSim Software Products, Inc. Other registered trademarks belonging to third parties are used within this work. Where those designations appear in this book, and FlexSim Software Products, Inc. was aware of a claim, the designations have been printed in italics, caps, or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained here.

ISBN: 978-0-9832319-5-0

To Janet and Jane – for your patience and support

Acknowledgements

Many people have contributed their knowledge and experience to the material in this book. The authors wish to thank everyone in the FlexSim U.S. office for their support and contributions. We especially extend our gratitude to those individuals at General Mills, Inc. and the Center for Advanced Vehicular Systems–Extension at Mississippi State University who encouraged and supported the use of simulation.

Preface

Simulation is an applied technology that is especially useful for analyzing and solving problems. Applying simulation begins by being clear on the problem definition, the reasons for simulating, and the expected outcomes. Simulation with no objective is counterproductive.

A person using simulation then must balance their understanding of the problem with their knowledge of the details of simulation: the underlying simulation concepts, application software, and the analysis methodologies that are employed. The most effective way to learn how to successfully use simulation is through learning how to apply it.

A major challenge in teaching applied simulation is the question of how to effectively blend and balance an understanding of fundamental principles and concepts with the practical side of building simulations. The intent of this book is to help bridge that gap and improve the effectiveness of simulation courses. Not all readers of this book will become simulation experts, but hopefully they will want to utilize the technology to help them or others make better decisions. Consequently the material takes the reader through three levels of users: Occasional, Intermediate, and Advanced.

Modern simulation applications allow the user to accurately represent an environment both as a visualization of the physical elements as well as the underlying, often complex, logical structure. Using both aspects of a simulation greatly increases its effectiveness in gaining acceptance and solving problems. *FlexSim* was chosen for use with this text because of its ease-of-use and rich visual and logical functionality that allows users to focus on simulation concepts and methods. This is not intended to be a *FlexSim* manual as the *FlexSim* Help files and tutorials are more than adequate for that purpose.

The first three chapters provide a background for the practice and use of simulation and provides a base for all user levels. Chapters 4 through 6 focus on the needs of an Occasional User, who may not want to be a simulation professional but wants to try using simulation. These chapters focus on the basics of simulation software, statistics, and model development. They show the reader how to use the standard functionality of the software to build simple models that can incorporate a realistic level of complex logic.

Chapters 7 thru 11 introduces the Intermediate user to the additional theory and background needed to address the very practical issues of reliability, randomness, and output analysis that are critical to a successful simulation.

The Advanced user will find in Chapters 12 through 15 concepts and examples that go beyond the standard software functionality, with a focus on how to easily add robustness to model logic. This section also adds the ability to integrate aspects that exhibit fluid flow characteristics into the simulation.

The final section, Chapters 16 and 17, addresses professional practice - topics that focus on successfully implementing a simulation project. This guide is usually not covered in other texts but is critical for successfully carrying out simulations in a corporate environment. This section can be covered anytime in a course, preferably just prior to starting a course project.

Contents

Preface . . . . . **VII**

Chapter 1: Simulation as a Tool for Understanding **13**

Section 1-1 What is simulation? 13

Section 1-2 The problem with waiting time. 15

Section 1-3 Working in a global, dynamic environment 24

Section 1-4 Simulation in everyday life 29

Section 1-5 Where is the money? 33

Section 1-6 When and where to use simulation 36

Section 1-7 Simulation users 42

Section 1-8 Summary 47

Chapter 2: Introduction to Modeling and Analysis of Queueing Systems **48**

Section 2-1 Prevalence of Queueing 50

Section 2-2 Basic Elements of Queueing 51

Section 2-3 Characteristics of Queueing System Elements. 53

Section 2-4 Measures of Queueing System Performance 60

Section 2-5 Means to Study of Queueing Systems 62

Section 2-6 Analytic Models of Queueing Systems 63

Section 2-7 Analyzing the Behavior of Queueing Systems 67

Section 2-8 Simulating Queueing Systems 74

Section 2-9 Summary 84

Chapter 3: Using Simulation to Solve Problems **93**

Section 3-1 Using a simulation 93

Section 3-2 Simulations with random events 94

Section 3-3 Examples and exercises. 96

Exercise 3-1 Coasting around 98

Exercise 3-2 Martian Transfer Station 100

Exercise 3-3 Slime Inc. 102

Exercise 3-4 Danson Electronics. 104

Exercise 3-5 Maritime Ltd.. 106

Exercise 3-6 Zombie Batteries, Inc.. 108

Chapter 4: Building Basic Simulation Models **111**

Section 4-1 Essential features of discrete-event simulation models. 112

Section 4-2 Simulation environment. 114

Section 4-3 Simulation components 117

Section 4-4 Fixed resources 119
Section 4-5 Transporting items 120
Section 4-6 Editing objects 122
Section 4-7 Making connections 125
Section 4-8 Moving flowitems. 127
Section 4-9 Creating simple learning models 132
Section 4-10 Single-run statistics 133
Exercise 4-1 Johnson Pharmaceutical 136
Exercise 4-2 Lucky Air 138

Chapter 5: Adding Model Logic and Managing Data 141
Section 5-1 Assigning attributes to objects 141
Section 5-2 Adding logic. 142
Section 5-3 Managing data through tables 146
Section 5-4 Grouping and ungrouping flowitems 147
Section 5-5 Reusing custom objects 150
Exercise 5-1 More Lucky Air 152
Exercise 5-2 Even More Lucky Air. 154
Exercise 5-3 Hampton International. 156

Chapter 6: Improving System Representation 159
Section 6-1 Mobile resource objects (task executers). 159
Section 6-2 Using Lists for decision making 165
Section 6-3 Displaying information on the screen. 171
Exercise 6-1 Steve's Stone Cutting 176
Exercise 6-2 Treasures-4-U 178

Chapter 7: Incorporating Resource Availability and Reliability. 181
Section 7-1 Establishing Time Tables 182
Section 7-2 Performance 184
Section 7-3 Reliability 185
Section 7-4 Estimating MTBF and MTTR 187
Section 7-5 Simulating machine failures 188
Section 7-6 Setting MTBF and MTTR in a simulation. 190
Section 7-7 Using personnel for repairs. 192
Section 7-8 Surge 193
Exercise 7-1 Keggler's Brew 194
Exercise 7-2 Chairs for Tots 197

Chapter 8: Modeling Randomness 203
Section 8-1 Importance of probability distributions in simulation 204

Section 8-2 Selecting probability distributions based on sample data . . . 206
Section 8-3 Selecting probability distributions in the absence of sample data. 216
Section 8-4 Sampling from Continuous Probability Distributions 218
Section 8-5 Sampling from Discrete Probability Distributions 222
Section 8-6 Generating Random Numbers 223
Section 8-7 Putting It All Together— Obtaining Random Samples from Probability Distributions 226

Chapter 9: Simulation Mechanics— How Monte Carlo and Discrete-Event Simulations Work 235
Section 9-1 Monte Carlo simulation 236
Section 9-2 Discrete-event simulation. 243

Chapter 10: Fundamentals of Output Analysis 265
Section 10-1 Statistical inference from simulation models 266
Section 10-2 Key tactical experimental design decisions 273
Section 10-3 Creating experiments using ***FlexSim's*** Experimenter 285

Chapter 11: Multi-scenario Experimentation and Optimization . . . 295
Section 11-1 Definition of case study and preliminary analysis. 296
Section 11-2 Comparison of two alternatives 302
Section 11-3 Reducing variability in the samples using Common Random Numbers (CRN). 306
Section 11-4 Comparison of multiple alternatives using Bonferroni's method 310
Section 11-5 Introduction to Optimization. 314

Chapter 12: Logical Relationships Among Objects 333
Section 12-1 Hierarchical software architecture 334
Section 12-2 Understanding how objects work 337
Section 12-3 Scripting in ***FlexSim*** 339
Section 12-3 Scripting tips 345
Exercise 12-1 Hilltop Steel Works 358
Exercise 12-2 The Crafty Framer 360

Chapter 13: Customizing Models 363
Section 13-1 Communicating between objects 363
Section 13-2 Label Tables 368
Section 13-3 Customizing object placements 370
Exercise 13-1 Fister's Express 374
Exercise 13-2 Peoples Surgery Center. 376

Chapter 14: Advanced Logic Functionality and Representation 381
Section 14-1 Basic concepts of Process Flow elements 382
Section 14-2 Defining characteristics of Process Flow elements 383
Section 14-3 Building a Process Flow 383
Section 14-4 Creating zones with Process Flow elements 388
Section 14-5 Synchronizing Process Flow elements with the 3D layer . . 391
Exercise 14-1 Greyson Operations 398
Chapter 15: Simulating Fluid Flow 403
Section 15-1 Basics of fluid-flow simulation 403
Section 15-2 Fluid objects 405
Section 15-3 Specifying Fluid Operations. 407
Exercise 15-1 James Peanuts 412
Exercise 15-2 Western Grain 415
Chapter 16: Applied Simulation Environment 419
Section 16-1 Confidence 419
Section 16-2 The Simulation Modeling and Analysis (SMA) Life Cycle 421
Section 16-3 The modeling and analysis process 422
Section 16-4 Roles in a SMA project 425
Section 16-5 Model-based decision-support systems 428
Section 16-6 Simulation and other tools 429
Section 16-7 The simulation software marketplace 432
Section 16-8 Simulation modeling and analysis success factors. 436
Chapter 17: Managing a Simulation Project 441
Section 17-1 The starting point 441
Section 17-2 Define the system and the reason to simulate. 443
Section 17-3 Prepare a conceptual model 448
Section 17-4 Initial sizing of resources 464
Section 17-5 Building the simulation 465
Section 17-6 Results and analysis 466
Section 17-7 Example of a project template 469
About the Authors. 483
Index 486

Chapter 1 Simulation as a Tool for Understanding

This first chapter lays the foundation for studying simulation as an applied technology and for analyzing and solving problems that exist in real systems. It provides the context for the book by defining what simulation is and by specifying the characteristics of the systems in which simulation is applied. These systems are widespread and diverse; they include all areas of manufacturing and mining, as well as service operations such as healthcare, security, transportation, retail, and distribution, to name a few. The chapter also introduces why simulation is used and what value it adds to all facets of decision making. This chapter also introduces the three types of simulation users (Occasional, Intermediate, and Advanced). The book is organized into three main sections that address the needs, capabilities, and skills of these three types of simulation users.

Objective

Using examples, explain (1) how and why simulation is an exceptional tool to analyze and solve real problems, (2) the economic benefits of simulation, and (3) the role of simulation in decision making.

Section 1-1 What is simulation?

As the title indicates, this book is about simulation, with an emphasis on applied, real life situations. The primary focus is on modeling and analysis. The first step in understanding these concepts is to define the term *simulation* as it is used in the text.

Generally, the verb to *simulate* means to mimic or imitate. This book takes a more precise view of the verb as it relates to actual applications. Here, to *simulate* means "to mimic or imitate through experimentation with a model (or representation) of some real system." The model must encompass both the physical environment (through dynamic visualizations), as well as the logical aspects that direct the actions that occur in the simulated environment.

Simulation involves more than just mimicking or experimenting. It involves defining, designing, and constructing a model or representation; defining the experiments to be conducted; collecting and analyzing the data that will drive the model; and analyzing and interpreting the results obtained from the experiments. Simulation, therefore, as referred to in this book, is the process that encompasses all of the above activities.

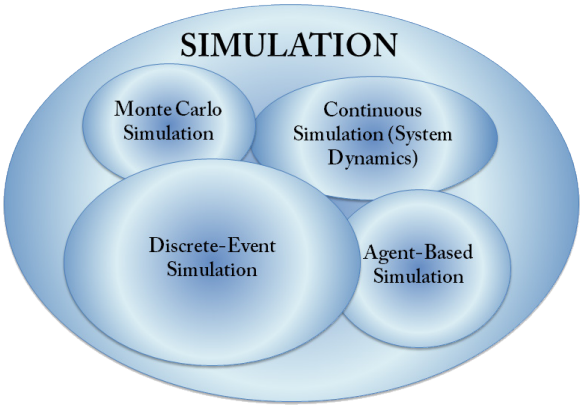


Figure 1.1: Types of simulation for analyzing operations.

There are many types of simulation models that are used for many purposes. Four of these are especially useful for analyzing operations or production-system problems, as shown in Figure 1.1. The focus of this book is on discrete-event simulation, but Monte Carlo simulation is described in Chapter 9 and continuous simulation (also referred to as System Dynamics and fluid simulation) is introduced in Chapter 15.

All simulations are models or representations of a system. Models in general are widely accepted, although two rather paradoxical comments point out the strength and weakness of trying to describe actual processes with models:

> Models are tools to support and extend the power of thinking. [3]
>
> Essentially, all models are wrong, but some are useful. [1]

As this paradox points out, models can be great thinking tools, they foster discussion and understanding of how systems operate and encourage exploring and generating new ideas for improvement. Models are not intended to replace thinking, to the contrary, they are used to enable and enhance thinking processes. Of course, as the paradox also points out, since models are simply representations or abstractions of real systems, they do not, and cannot, capture all of the behaviors and complexities of real systems. However, replicating every detail of a system is not necessary for it to be very useful. In fact, trying to capture too much system detail in a model can make it unwieldy and complex. A model heavy in system detail becomes difficult to understand and maintain and thus is not useful. One of the arts of model building is deciding on the right level of detail to effectively deal with the problem being addressed. Simulations start with models then use and expand them to describe operations systems with complex interactions over a continuum of time. As an analogy, a model is to a picture, as a simulation is to a movie.

For the type of modeling referred to throughout this book, we adopt Pidd's definition that a model is

> an external and explicit representation of part of reality as seen by the people who wish to use the model to understand, to change, to manage and to control that part of reality. [3]

Section 1-2 The problem with waiting time

Over the years there have been many management techniques introduced to reduce and manage the waste in a process. These include: just-in-time, Kanban, business process re-engineering, lean manufacturing, and six-sigma. There is no doubt that these techniques and processes are useful when it comes to efficiency and decisiveness. However, the most important factor, profitability, which should be considered first, is often lost in the process because most engineers are not directly responsible for company profits.

These management techniques have one goal: to reduce the waiting time (waste) in a process. Waiting time in a process can be grouped into three categories: customer waiting time, product waiting time, and machine waiting time. The problem with these waiting times is that they result in a tug of war that pits customers, products, and machines against each other. The difficult question to answer is what kind of waiting time reduction saves the most money. To get an understanding of the difficulty to solve the question at hand you have to understand the dynamics of the process itself and how the three types of waiting time impact the total cost of the process.

Where does waiting time or queuing begin?

Between each transformation step in any process is a period in which the item being produced must wait. This time will be defined as the product waiting time. There are cases when waiting is the objective, such as the money supply of a bank: money held overnight or for a period of days pays interest. But in most cases, waiting time and queuing is a recurring evil that can suck away the profits in a process. The fact is that waiting time can comprise 90 percent of the total throughput time of a process. If this seems high, take a walk through your production facility and measure the time product sits around versus the time it is in actual production or being worked on. You will see very quickly the seriousness of the problem.

Waiting time occurs in every industry, including manufacturing, warehousing, supply chain, services, and healthcare. When was the last time you were annoyed by waiting in the line at the supermarket or delays at the doctor's office? Although difficult to quantify in terms of money, these waiting times create irritation and inconvenience for customers that can be qualitatively high.

In many cases the translation from the physical world into a queuing model is made easy by analyzing the constraints (processes with high utilization) of a system. In other cases the translation is not so evident. For example it may be easy to calculate the cost of floor space used to queue parts in production, but it is another matter to calculate the cost of customers who have to wait for service that might never return because of the frustration of waiting.

The cost of waiting time

Every process system contains the three types of waiting time. If you understand and are able to weight the cost of each type of waiting time, it is possible to determine the optimum cost of the total waiting time. Let's take a look at each type and weight the cost of each.

Customer waiting time

The customer is king. Without customers you have no business, and in every business the goal is to service the customer as quickly as possible. If a customer has to wait for products or services they will always look for another source that can service them in a timely manner. As a result, the cost of customer waiting time is the most expensive. If you have no customers the amount of waiting time for products or machines is moot.

Product waiting time

Product waiting time includes all the waiting time a product (raw material, work-in-process, and finished goods) spends in all its different states. This waiting time will be weighted as the second most expensive waiting time of the three. When you consider the cost of raw material storage and finished good storage (warehouses) you quickly see the cost involved. This cost or waste as seen by the lean engineer is the main target for kaizen events.

Machine waiting time

Machine waiting time is the time a resource (machine or human) is idle waiting to work on a product. This cost is ranked number 3 because it relates only to the cost of an idle resource. It can be costly if the resource is very expensive to purchase. The ROI of equipment is always scrutinized by the financial side of the business to insure equipment has the highest possible utilization. Because the financial side of a business does not understand the effects high resource utilization (95 percent load factor) has on the system as a whole, process efficiency may be doomed even before the first product rolls off the line.

Let the tug of war begin

Every process or logistics problem can be viewed as a distribution problem of the cost of waiting times. Factors that impact these waiting times include but are not limited to variability, quality, equipment reliability, staff reliability, and system load factors. When each type of waiting time increases or decreases it has an impact on the other; this creates a tug of war between the costs of waiting (see Figure 1.2). For example, if you wanted to satisfy the demand of every customer immediately you would have to maintain a finished goods supply of every product you make. The cost of product waiting would sky rocket. On the other hand, if you wanted to reduce your work in process you run the risk of not having product at the right place at the right time; your machines may be

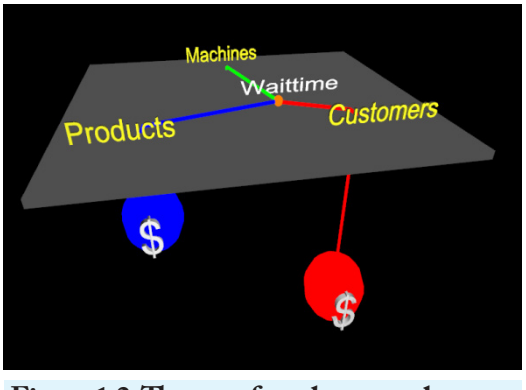


Figure 1.2: The tug of war between the costs of waiting.

idle waiting for product, thus increasing the cost of waiting for machines. This tug of war is not just between two sides---it is a 3-way war. Each can affect the other two.

For example, a machine that is loaded too high will cause a large queue to build in front of the machine. It may also starve those processes downstream. Thus, the product waiting cost increases and deliveries to the customer may fall behind, increasing the cost of customer waiting time. This complex dance of waiting times can be difficult to understand and control unless you know how to analyze and experiment with the correct variables. Before we look at the tools used for such analysis and experimentation, we need to understand the dynamic behavior of what causes waiting time.

How does a queue start?

Queues start the moment the supply of products or customers is greater than the processing capacity. As a result, an unstable situation develops and, theoretically, the queue will increase indefinitely as the utilization of the system approaches 100 percent. Queues build when supply exceeds capacity and decrease when supply is less than capacity. Another factor that can influence the queue is variation in the processing time. We all know what it is like to get behind the person in the checkout line who has all kinds of special requests that cause you to wait endlessly. When viewed over a long period of time the average utilization of a machine or resource may be less than 100 percent, but variation can cause queues to build temporarily or you could have a situation where machines are starved for a period of time.

The equation for the calculation of average waiting time ($\overline{W}_t$) is shown below; the relationship between utilization (ρ) and variability is expressed in terms of the variance coefficient (c_v) and waiting time (W_t). Average processing time is shown as ($\overline{P}_t$).

$$\bar{W}_t = 0.5(1 + c_v)(\frac{\rho}{1-\rho})(\bar{P}_t)$$

When the utilization of a machine increases to more than 70 percent and there is a high degree of variation in arrivals, a situation can develop where a queue can build. The queue can only be eliminated if arrivals are limited or there is added capacity. The relationship between utilization, variability, and average waiting of the product or customer was shown in Figure 1.2.

A constant arrival pattern has a variance coefficient of zero. An independent arrival pattern (Poisson distribution) has a variance coefficient of one. Figure 1.3 shows that there is only a small area where long waiting times occur. Waiting time is only a problem when there is high utilization and a large degree of variability in arrivals or process time. By reducing both factors, the waiting time of the product or customer can be reduced. However, when you only reduce machine utilization, machine waiting time can become a factor. In a system where you have a large degree of variability, increasing capacity is the only way to reduce the utilization of a machine and the number in the queue.

Many processes show a lot more variation than you might think. Breakdowns, pauses, re-sets, waste, operator error, and restocking make a seemingly constant process extremely

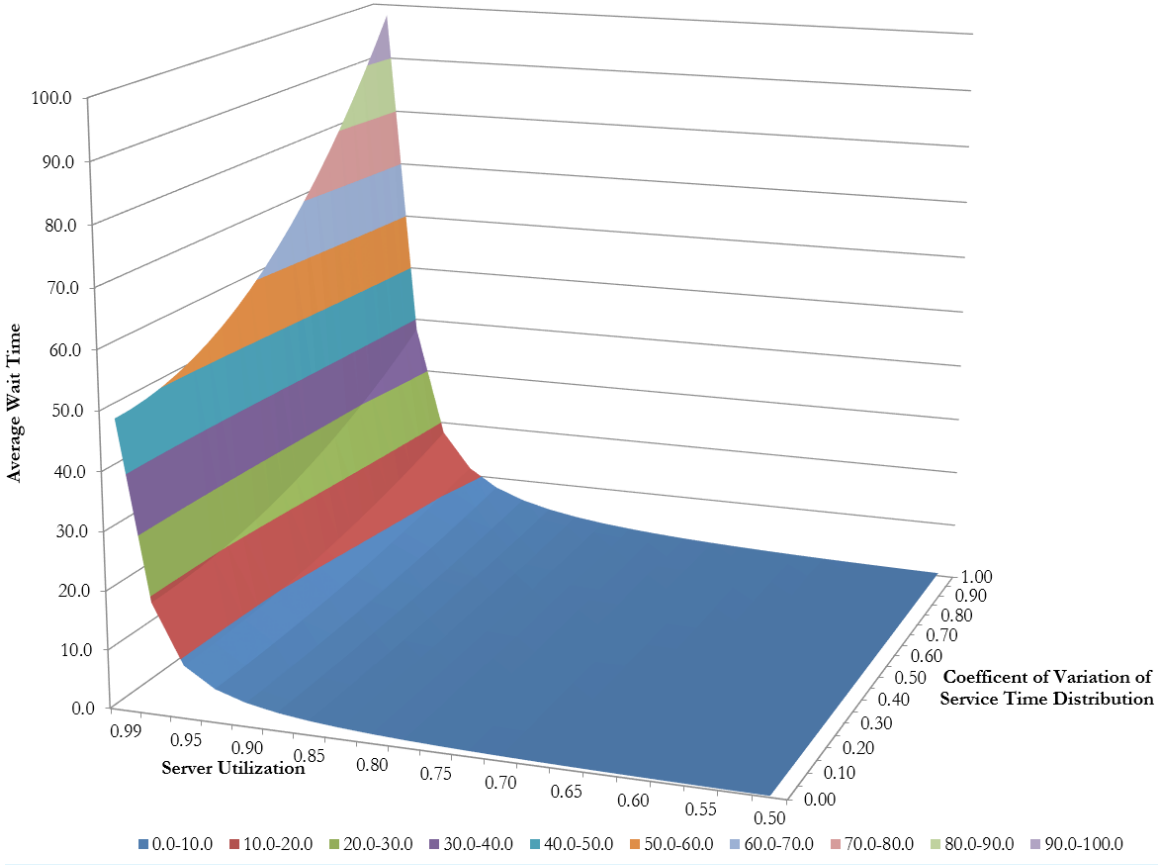


Figure 1.3: The relationship between utilization and variability.

variable. Over time the process might seem constant, but on a weekly, daily, or hourly basis production has a very stochastic nature. With the numerous uncertainties, one can assume your system is a Poisson system and will have a variance coefficient of one. Figure 1.4 shows the relationship of utilization and waiting time of products and machines.

By adding the waiting time of the products and machine you can find the optimum utilization for

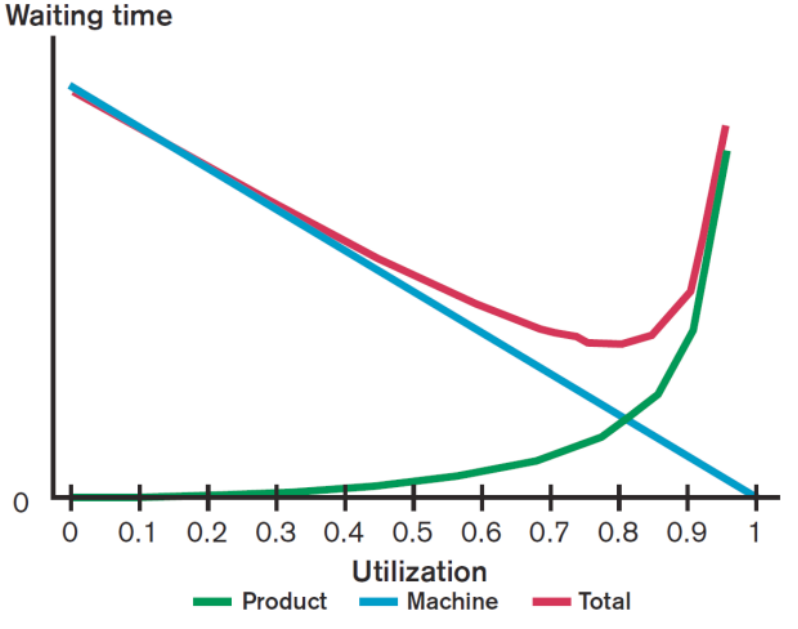


Figure 1.4: The relationship among utilization and waiting times with a Poisson process.

the machine. The minimum cost is derived by calculating the weighted cost factors related to each waiting time. The exponential form of the waiting time graph often confines the optimum utilization of a machine between 70 and 90 percent.

When you consider the dynamics of utilization, waiting time, and the cost of waiting, you can see the inherent problems designed into systems where equipment is purchased with an ROI basis of 90 percent (or higher) load factor. The process of buying equipment already has built in bottlenecks and waste that cost real money and time on the production floor.

Minimizing queues

Queuing problems develop from a combination of high utilization and variability in the arrival pattern and the process time of a machine. Another factor that can cause queuing is batch processing, which increases the process time because the batch cannot move until the last piece in the batch is completed. Batch processing is used to minimize the movement steps between processes and setup times at the process. Using techniques to minimize the setup time when changing part types has reduced the variation in batching processes. Optimization of scheduling batches that reduce setup based on similar part makeup is vital to prevent unnecessary delays in processing.

There are three ways to add capacity to a process that will help reduce the utilization of the process and the product waiting time. The first is to reduce the stochastic nature of the arrival rate of products, the second would be to add a machine, and the third is to reduce the process time on the machine. It is not easy to fix the stochastic nature of arrivals; adding expensive machines or resources may not be cost effective. There is a persistent notion that a minor reduction in process time has no significant influence on waiting time and throughput. The reasoning is that process time only accounts for 5 to 10 percent of the total throughput time. However, this reasoning ignores the relationship between the utilization of a machine and the wait time in the queue associated with the machine.

Suppose the processing time of a production step is five minutes and the utilization of the machine is 90 percent. Using the Pollaczek-Kyntchin equation for M/G/1 queuing systems, one can calculate that the average waiting time is 45 minutes. The total throughput time is 50 minutes, resulting in 88 percent waiting time and 12 percent processing time.

$$\bar{W}_t = 0.5\,(1+1)\left(\frac{0.9}{1-0.9}\right)(5) = 45 \text{ minutes}$$

If you could optimize the working process and reduce the time by 10 percent (30 seconds), would the average waiting time be reduced significantly? At first look, it doesn't appear that the waiting time would be reduced by much. However, as a result of the decrease in processing time, the utilization would drop to 81 percent. Since waiting time increases exponentially with utilization, the waiting time could be reduced significantly. With this value, the waiting time is 19 minutes. The total throughput decreases from 50 minutes to 23.5 minutes, a reduction of approximately 50 percent. This seemingly small change in processing time results in a significant reduction of waiting time.

$$\bar{W}_t = 0.5\,(1+1)\left(\frac{0.81}{1-0.81}\right)(4.5) = 19 \text{ minutes}$$

Using math to calculate waiting time is easy for simple M/G/1 systems. What about more complex systems that are commonplace in today's manufacturing and logistic environment? Mathematical analysis and spreadsheet calculations fall far short of measuring and understanding the behavior of complex stochastic processing and logistic systems. Understanding and managing the competing costs of waiting is critical to the profits of every company, yet very few understand the relationship and effect that waiting has on costs and customer satisfaction, let alone have any idea how to measure the effects so they can manage the process.

Discrete Event Simulation

The only tool ever developed to analyze the stochastic nature and tug of war impact, relationship, and cost of customer, machine, and product waiting time is discrete event simulation. With the use of simulation you don't have to guess how your system is going to behave before or after change. Simulation allows you to scientifically experiment with your system before you make real changes so you know with certainty the changes you make will save real money. How would you like to know with 95 percent confidence that you will save 22 percent of the cost of work-in-process storage costs? Or, that you have 95 percent confidence that you can reduce the delivery time of a product to your customer by 65 percent? You can with the use of simulation.

Proof waiting time exponentially increases with machine utilization

Simulation can be used to prove waiting time exponentially increases as machine utilization increases. We will experiment with an M/G/3. A source will generate the arrival of a box every 10 seconds exponentially distributed (to provide some stochastic behavior to the model). Boxes then move to a queue before moving to one of 3 machines for processing. The processing time at each machine will be set to 30 seconds to balance the system. Boxes are sent to a sink after processing and leave the model. A picture of the model is in Figure 1.5.

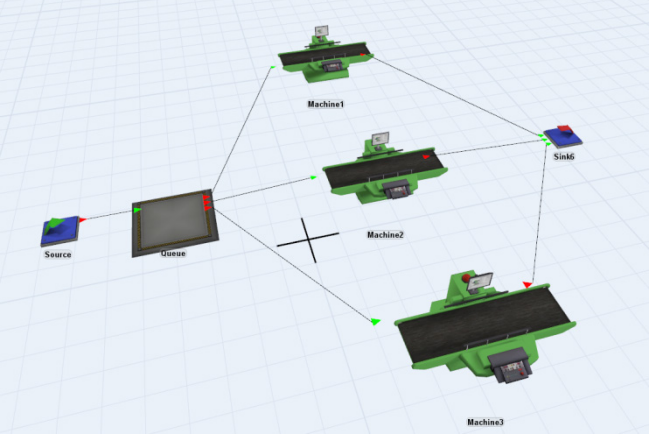


Figure 1.5: M/G/3 queuing system.

The experiment

To run the experiment we will look at two scenarios (see Figure 1.6).

	Variable	Scenario 1	Scenario 2
cycletime machine1	/Machine1>labels/cycletime	30	27
cycletime machine2	/Machine2>labels/cycletime	30	27
cycletime machine3	/Machine3>labels/cycletime	30	27

Figure 1.6: Simulation experiment conguration for two scenarios.

Scenario 1: Baseline

The baseline will establish the utilization percentage of the machine and the average wait time of boxes in the queue with an arrival rate of 1 box every 10 seconds exponentially distributed (Poisson process, $c_v = 1$) and a cycletime of 30 seconds at each machine. We will run the model for 8 hours of production and we will run the model 20 times and average the results. Results will be based on a 95% confidence interval. Performance measures will be the average utilization for all machines, the average wait time of boxes in the queue, and the maximum content in the queue. Results of the baseline experiment are shown in Figure 1.7.

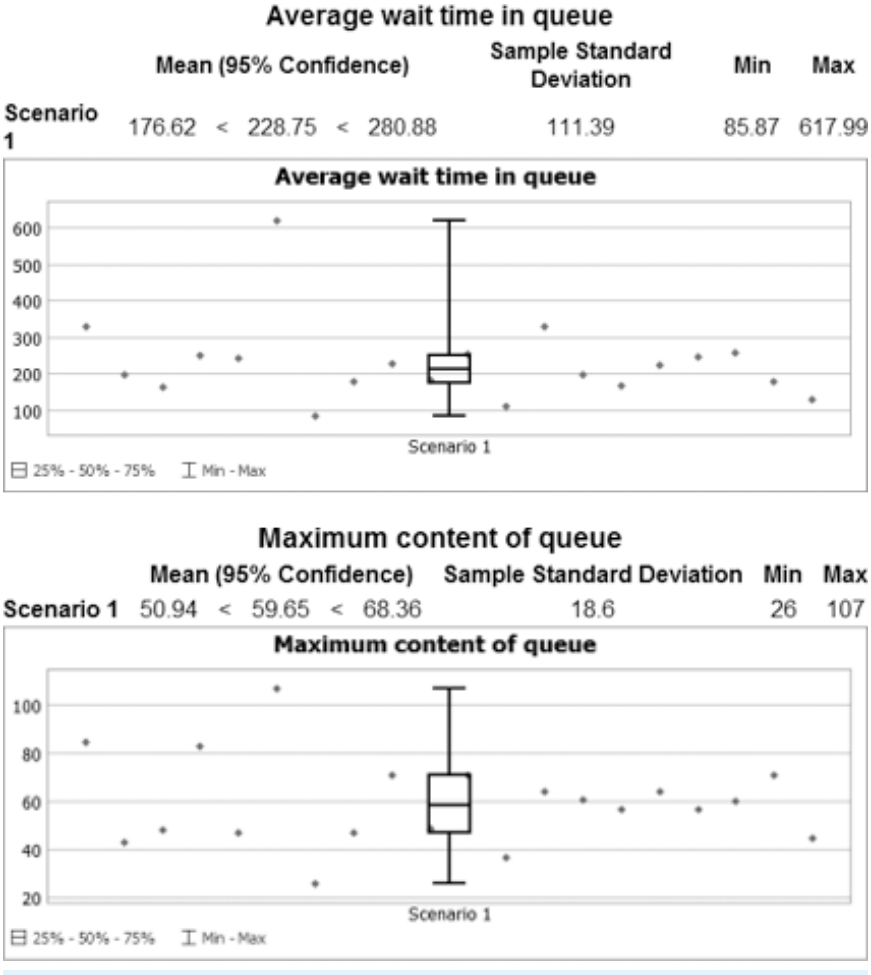


Figure 1.7: Results of the baseline run.

Results show an average of 98.72% utilization at each machine, an average wait time of 228.75 seconds for each box in the queue, and the average maximum content of the queue as 59.65. All averages are based on a 95% percent confidence interval (the min and max range and standard deviation is shown for each average).

Scenario 2: Proposed change

In Scenario 2, the utilization percent of the machines will be lowered by a 10% reduction in cycletime. This will result in a 27 second process time at each machine. The model will be run again with the same settings as the baseline with the only change being the cycletime. The results are displayed and compared to the baseline run in Figure 1.8.

Results show a 10% reduction of utilization of each machine as a result of the 10% reduction in cycletime. Baseline model showed a utilization percent of 98.72% and the proposed model shows an 89.8% average machine utilization. The average wait time of each box fell sharply from 228.75 in the baseline model to 32.05 in the proposed model. The

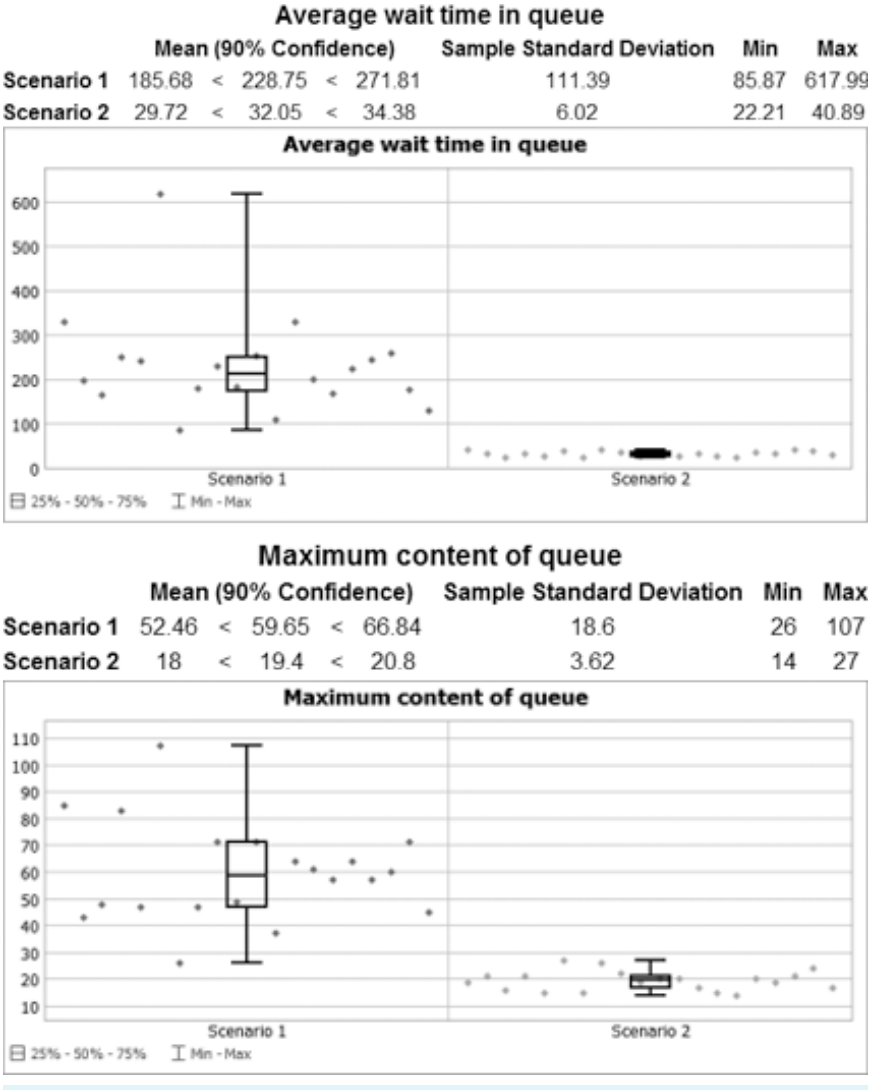


Figure 1.8: Results of the proposed run compared to the baseline run.

maximum content of the queue also fell sharply from 59.65 in the baseline model to 19.4 in the proposed model.

This data confirms the hypothesis that waiting time rises exponentially as machine utilization increases. The baseline model shows a 90% waiting time versus a 10% processing time. The proposed system reduced the waiting time percent to about a 50-50 percentage.

The real story is to look at the average waiting time for the boxes in the queue for each scenario. For the baseline model, where machine utilization was approaching 99%, the average waiting time was almost ten times the processing time for each box. The proposed model, with the cycletime reduced by 10%, reduced the machine utilization by 10% and also reduced the average waiting time for each box to only 32.05 seconds. In Figure 1.9 you can see that the relationship between machine utilization and waiting time in the queue is exponential.

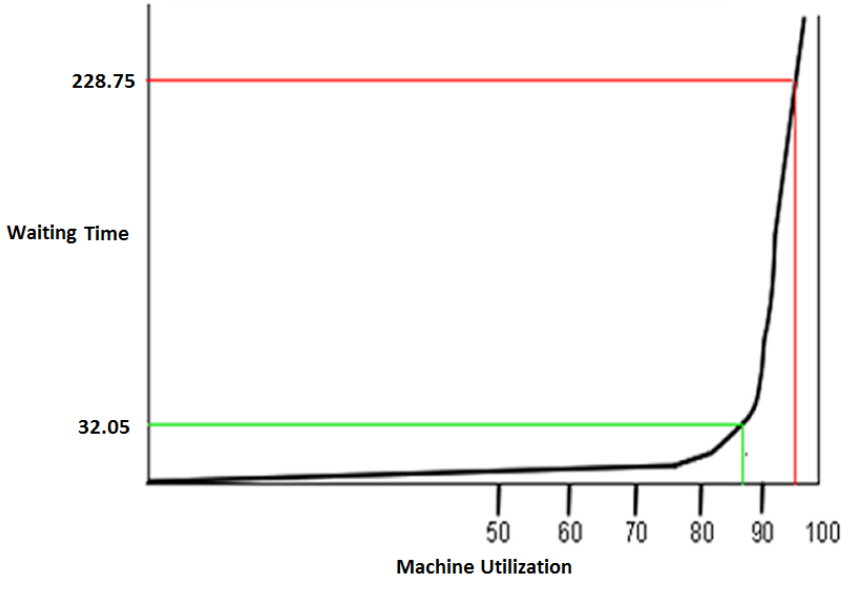


Figure 1.9: The exponential relationship between machine utilization and waiting time.

Results

As you can see, the use of simulation made it easy to understand the relationship of machine utilization and product waiting. Simulation provides an array of data and charting options to easily identify and experiment with bottlenecks (areas of high utilization) to understand the best way to solve problems. What we have looked at so far is very simplistic but would have taken hours to calculate by hand or using spreadsheets. The real benefit of simulation is the ability to build models of real life complexities and relationships that are impossible to analyze any other way. For example, how would the system change if people were introduced to move the boxes and process them at the machines? Add more uncertainty and the waiting time will increase. Simulation is the only tool that allows you to build a model, experiment, and make informed decisions before you implement. How do you know your lean initiatives will work or save as much money as presented? Build a model of the current and proposed system and find out.

Section 1-3 Working in a global, dynamic environment

Economics, in the form of money, land, power, or control, is the primary motivator that influences the way people live. All life is influenced by the shared basic elements of air, water, and land. Communications and computer technology have brought our thought processes and actions even closer together. Actions in one area now impact others as complex, and often non-intuitive, ripple effects. These ripple effects can be short or long term in nature. Even if everyone in the world believed they were working for the same objectives of peace and justice, actions by one segment could unintentionally contradict what another was doing.

Competition, risk, and innovation

Organizations, businesses, and individuals worldwide search for ways to lower costs and optimize their use of critical resources; however, achieving this ideal in a changing, integrated, and dynamic worldwide environment becomes a major challenge. In recent history, organizations brought people together in quality circles to discover ways to avoid waste and improve their products. Today, organizations search for *lean* systems that simplify their operations. They prepare value stream maps to identify wasted time and effort. But in a world that is ever more integrated and dynamic, even these efforts can quickly become out of date or fall prey to unintended consequences.

Success in the world economy is often viewed in terms of competition, risk, and innovation. In a world where time literally is money, it is critical to plan, execute, and work efficiently the first time, every time. For instance, a business must be able to make timely decisions in order to keep up with the changing demand for goods and services. Consider the implications of competition and risk for consumer based operations. Providing goods and services without direct demand information is not economically viable. The result is goods sitting in warehouses dragging down corporate profits. Electronic goods sit idly while technology passes them by, food reaches its expiration date sitting on the shelves, and highly paid service providers sit with no income stream waiting for clients.

Just as manufacturers can't afford to keep large reserves of product on hand, they also can't afford to run out of product and lose customers to their competitors. Sellers of goods, such as large retailers, refuse to accept warehousing costs for products that they might have to sell at "clearance" prices. In response to the need for speed and accuracy in meeting consumer demands, a philosophy of pull demand emerged. In a pull demand scenario, goods and services are produced just as they are needed. While this sounds good in theory, it is difficult to successfully implement in reality.

Now consider some similar scenarios in other areas of commerce and services. Banks want to optimize their staffing and have the correct number of lines open as customer demand fluctuates. Grocery stores have a similar problem balancing the number of workers at check-out lines with personnel loading shelves. Military services have to keep spare parts for equipment and send them to where they're needed. Hospitals need to have the correct

number of rooms, equipment, and staff available for various patient levels and be able to minimize their operating costs at the same time.

Innovation is another important key to success because it provides a competitive or business advantage for one company over another. Innovation comes in many forms. It can be a great new idea for a product, a new machine that improves the manufacturing cost structure, or a collection of small improvements that are built along the way. People in organizations are learning concepts such as Lean, Six Sigma, and Continuous Improvement and using them to analyze and improve their operations and the way they interact while doing business. Many of the concepts involve people at all levels of an organization studying and improving their operations using simple analytical tools.

Unfortunately, a company or organization generates more innovative ideas than it can handle. The company must consider priorities for funds as well as the impact of the innovations before any of those ideas can be implemented. With highly interactive systems, what might be a great idea for one area of a business may actually harm another area. For example, a lean team on the production floor gave a recommendation for how they could increase the capacity of their assembly line without the need for additional equipment or personnel. They suggested that by making a minor change in the resource flow path, they could share part of the path with a nearby line that had excess capacity. There were also proposals from lean teams on another line claiming efficiency improvement, including the line sharing part production. Someone has to decide if the proposed efficiency gain would actually be achieved and be sustainable through production forecasts for both lines.

While the previous example may not have involved a lot of capital cost to implement, the results of making the right choice were significant. When a major capital project is being considered (new plant, new line, major expansion, etc.), time, money, and business impact are all critical elements. The major question is for all that's being invested, will we get what we think we'll get, and when will we get it. The worst possible outcome is an "end of project mismatch discovery," in which people suddenly realize that the project outcome doesn't match what was planned.

The impact of decisions is often based on when they are made. Major projects normally consist of various phases with at least two major decision points for commitment of funds (Figure 1.10). The cumulative amount of cash expended rises exponentially as the project continues, so the earlier a serious problem in concept or design can be detected, the lower the sunk cost will be. The cost of correcting or mitigating any problem is also lower during early project phases; thus, there is a definite need to be correct the first time through a project.

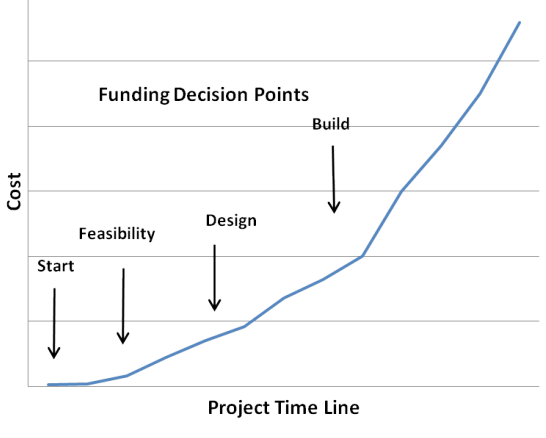


Figure 1.10: Capital project costs.

For example, a large paint company thought of a major expansion to produce a new, environmentally friendly, long-lasting

paint. The line would use a new manufacturing concept for producing the paint. Projected demand for the various paint grades and colors appeared strong, so flexibility in the line was important. Before proceeding, they had to study various proposals to find the most cost-efficient one that produced the results they needed.

In another case, an emergency response organization for a major city worried about its ability to handle various situations. At issue were the number of trained personnel required, the amount and positioning of vehicles and supplies, and the city resources available for immediate as well as long-term operations.

Systems

Being able to understand and analyze the situations just described requires what is known as *systems analysis*. The basic definition of a system is

a set of related elements within some stated boundary.

In practice, a system can be as big as an entire plant or organization or as small as a work cell, hospital emergency room, or bank teller line. What is of interest is the behavior of a system as a whole both internally and in conjunction with its environment. That behavior is distinct from the behavior of any of the individual system components. Four general types of systems have been identified by Checkland [2] and described by Robinson [4]:

- *Natural systems:* Weather and galaxies, etc.
- *Human activity systems:* Call centers, train stations, etc.
- *Designed physical systems:* Buildings, equipment, transportation vehicles, etc.
- *Designed abstract systems:* Mathematics, music, literature, etc.

This book focuses on the interaction of human activity systems and designed physical systems, often referred to as operations systems. These systems are normally involved with the production of goods and services, such as manufacturing processes, service delivery, and transportation and logistics operations.

All decision makers throughout an organization have to try to figure out what is happening—or what is going to happen—in their systems and how to take the appropriate action. In essence, they try to understand the consequences of possible actions prior to taking action. Decision making is easy when the systems are simple and when there are no options to choose from; however, operations systems are not easy to understand and offer a large number of possible options or actions.

Trying to analyze and make the right decision is difficult for any single person because each system has one or more of the following behavioral characteristics:

1. Components within a system are subject to their own random events
2. Random events in the environment impact the system

3. System behavior is time dependent
4. System components have complex interactions, and thus there are many connecting paths within a system

Random events are common in operations systems. Some examples of random events include orders for production based on real-time sales, customer arrival times and those customers' particular needs, machine breakdowns, response time of operators to answer an event, arrival time of material, and material losses. When multiple pieces of equipment are involved and operations are integrated, these random events all compete with each other to impact overall performance.

Complexity

Even the most lean and simplified operations may be considered complex if they are influenced by external decisions, such as production schedule changes; random events, such as breakdowns; or their own flexibility that allows them to be changed frequently.

Oftentimes manufacturing and other operations depend on the characteristics of the product or service being provided. This can add operational complexity, especially if the arrival sequence to the system is random, or at least appears to be random, and has a high degree of variability. For instance, a machine may be able to produce a variety of parts, but its setup and processing times depend on the type of part being produced. Of course, the processing can vary from part to part within a type. Another example is a shopping checkout line where the time to tally the goods depends on how many people there are and if additional steps (e.g., weighing vegetables or checking coupons) need to be performed. The method of payment (e.g., check, cash, credit card, food stamps) also induces variability.

Even a lean manufacturing design can prove to have complex dynamics if impacted by random events. Calculating the performance of a simplified processing line is more difficult if it is subject to part shortages at random times, or if it is dependent on operating personnel to clean the equipment periodically before subsequent steps can be completed—especially if the personnel are shared and have other competing tasks to do. Lean operations that have any degree of batch processing may use simulation to help optimize the batch size under various conditions.

Decision making

In today's business and economic environment, the only certainty we know is that nothing is certain. Change will happen at an even faster rate; therefore, when a decision maker attempts to analyze a system and formulate a plan for optimizing its performance, that person may well find such an endeavor extremely difficult. Simply thinking through such issues using simple calculations is no longer a viable option; the systems involved are too dynamic and random.

As a result, methods have been developed to help a decision maker analyze systems. These methods are known as decision support systems. A decision support system acts as

a "window" or analysis tool to help the decision maker formulate action plans. Simulation is one such application used as a decision support tool. A simple and concise definition of using simulation in a decision support role is

> experimentation with a simplified imitation (on a computer) of an operations system as it progresses through time, for the purpose of better understanding and/or improving the system. [1]

Simulation in a decision support role is illustrated in Figure 1.11. In any type of system, decision makers need to understand, analyze, design, and manage the system; all of these activities involve making decisions. These decisions typically involve real systems that are comprised of complex processes; those processes in turn have highly interdependent components, which exhibit variability and change over time. Without the use of a model, decision makers will use a variety of methods to gather information from a real system, process the information, and then choose an action that causes a change in the system. Model-based decision making in general, and simulation-based decision making in particular, provide a means for decision makers to investigate a representation of the system, experiment with alternatives, and predict the effect of proposed changes, all external to the real system. This approach greatly increases the decision space (i.e. allows many more alternatives to be evaluated), is not intrusive on the real system, and enables risk of the actions to be assessed. In order for the decision maker to interact effectively with simulation models directly, models need to be embedded in decision–support systems that 1) facilitate entering data into the model, and 2) enhance the presentation of output from the model.

Simulation aids in decision making by directly identifying areas where cost or efficiency can be improved; providing insight into risk analysis by increasing knowledge of the system's dynamics and interactions; also by identifying the possible scenarios resulting from various actions. For example, while simulation may not be able to predict specific demands for future goods and services, it can be used to assess the impact of demand variability on a system's ability to react.

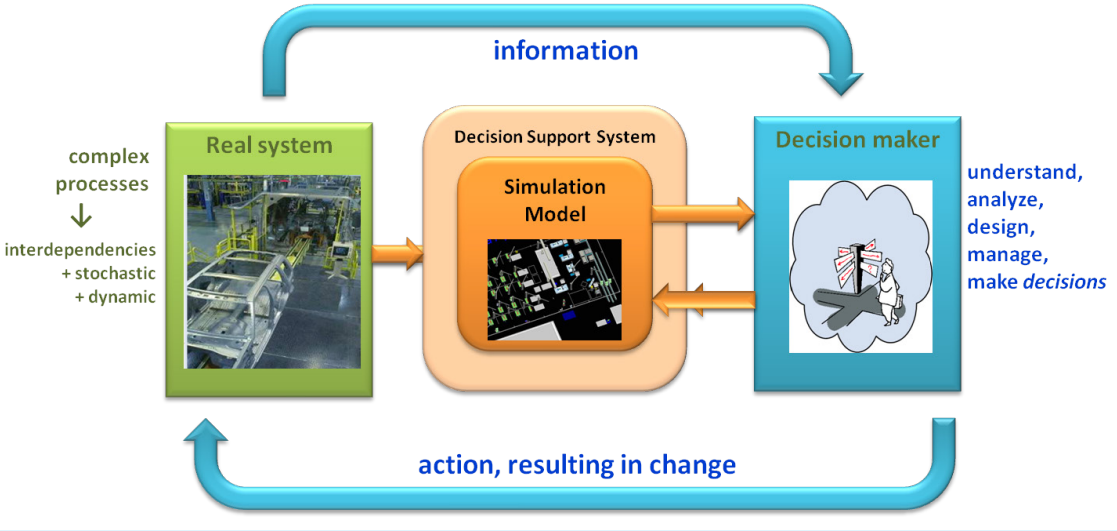


Figure 1.11: Model-based decision support system.

Because it is now critical for businesses and organizations to understand dynamic systems, a new emphasis on simulation technologies and applications has emerged. People across an entire organization can use simulation to help make good decisions.

Simulation has become a common analysis tool, used to

- analyze systems for improvement;
- match project scope with business requirements;
- share information across organizations;
- focus on the impact of complex dynamics.

To accommodate these goals, simulation has undergone a change—a change made possible by computer and software technologies. Traditionally, simulation has been associated with "experts" in the field who are entrusted not only with providing answers, but also with being the repository for dynamic knowledge. Now, however, the reach of the traditional paradigm is much too narrow; the knowledge and analytical tools to be available for everyone to use as a basic tool.

The result is a change in the paradigm for those who use simulation:

- Traditional paradigm (limited base of users)
    - Complex tools have no commonality
    - Experts receive training and use the simulation application
    - Experts solve problems for others
    - Simulation service is expanded by adding experts
- New paradigm
    - User friendly tools empower people
    - Users with varying levels of simulation knowledge can be accommodated
    - Users create common databases to share information
    - Service is expanded by increasing the user base

Because of this change in paradigm, simulation is currently at work in a wide variety of applications for systems that are part of everyday life.

Section 1-4 Simulation in everyday life

Everything experiences change in some way, either in time, space, or interactions. The ability to understand, analyze, and even predict these dynamics has been the goal of humanity since the beginning of time. Through simulation we can replicate historical dynamics, study them, and project possible future results.

Simulation is commonplace today and is often taken for granted. It has been the mainstay of training systems for many years. Commercial pilots work countless hours on flight simulators before transporting the public. Every branch of the military depends on simulators to train personnel and keep them ready for any eventuality. Equipment operators, from supertankers to subway train conductors, use simulation to hone their skills for handling dynamic situations that may occur.

Manufacturing has become a prime area for simulation efforts. When a single artisan was working in a shop, the output was very predictable. The pace was that of the individual. The only issues may have been the procurement of raw materials and the condition of the tools. With the start of the industrial revolution, and especially the assembly line, manufacturing changed into a complex entity involving multiple pieces of equipment, operations, and people—all of which interacted with each other. Today, manufacturing depends on simulation to design and test new manufacturing concepts, prioritize changes to existing operations, and ensure that the outcome of a proposed project will meet expectations. An example simulation model used to design a manufacturing system is shown in Figure 1.12.

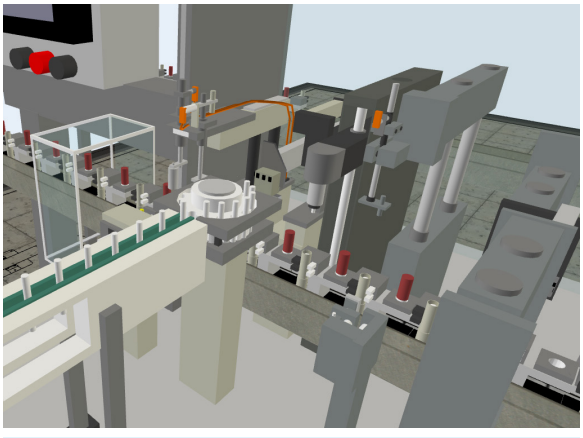


Figure 1.12: Manufacturing system design.

Consider a company's effort to successfully execute a major project. In many cases, simulation has become an integral part of design and an effective means to manage projects. For example, during the early initial or screening phase of a design or project, a simple simulation provides a "sanity" check on the concept as a whole. It also helps to settle on the most applicable approach given the current information. In addition, the simulation provides a means to communicate and document assumptions made during that phase. Once funds are committed to develop a more detailed plan, design, and cost estimate, more detail is added to the simulation. The simulation oftentimes becomes a common ground for discussions with the sales and marketing groups to get a better definition on the proposed demand.

Once a project is approved, simulation can be used to monitor and follow design decisions and identify potential problems. Again, the simulation provides a common ground for business units and project teams to negotiate changes. Staffing details are also simulated, and operational details are developed with plant personnel. During the execution phase, more detail is added to the simulation in order to address scheduling and to fine tune operations, all based on the latest marketing input.

Simulation beyond manufacturing

Simulation is also used in applications beyond manufacturing processes. The trucks or trains that transport raw materials or finished products are planned and scheduled based on simulations, as are the warehouses that they interact with. Traffic patterns, including intersection light sequencing, are now simulated in order to ease congestion. Large retailers simulate just-in-time scenarios to keep their shelves stocked while minimizing inventory. Railroads have long used simulations to optimize freight train length against schedule, terrain, and engine operating cost. Most recently, container terminal operations are being simulated to optimize the loading and unloading of large ships. An example simulation model for optimizing container port operations is shown in Figure 1.13.

Figure 1.13: Simulations optimize container port operations.

Simulations are being used to benefit service operations as well. Banks, ticket counters, amusement parks, cruise line check-in stations, and even airport security areas use simulation to anticipate the impact of equipment problems, sudden surges of people, or the number of open lanes. Simulation is helping to design more efficient health care facilities. Clinics and operating areas are prime targets to improve service quality and lower costs. An example simulation model for improving health care facilities is shown in Figure 1.14.

Figure 1.14: Healthcare is an emerging area for simulation.

Weather patterns, including hurricane paths, are generated by simulations based on atmospheric conditions and other meteorological data. Some re-creations of accident scenes and other events are merely animations while others are actually simulations based on the physics involved.

The most obvious use of simulation is in the ever growing field of computer games. In the early versions of the "pong" or "tennis" games, a simulated ball went back and forth between players with only the basic interaction parameters included. The ball would

bounce from the paddle based on the angle of impact. At the time it was considered unique. Since then, game simulations have advanced to the stage where players are totally immersed in a virtual environment. An important side effect of the game simulation technology is that dedicated players expect to pick up any new game and start playing without having to go through extensive documentation or training.

With the significant change in computer hardware and software, it is now possible to represent operations systems in a similarly realistic manner. *FlexSim*, the software used in this book to teach simulation concepts and methods, is an industry leader in 3D representation of manufacturing, logistics, and healthcare systems. The software is also extremely easy to use so that model builders can focus on creating a true representation of a physical environment.

These are but a few of the many areas where simulation technology is being used, either in a very obvious way or behind the scenes. The common thread is that simulations are created based on some level of knowledge. That knowledge may have been obtained from experience, physical laws, historical data, sequence-of-event tables, cause-effect relationships, or, in the case of the games, a created set of rules.

Basic characteristics of simulations

The notion that simulations are knowledge based is the first and most important tenet:

> Simulations require diverse knowledge of the system being considered. That knowledge includes understanding historical, current, and/or anticipated characteristics of the system being considered. The simulation then becomes a repository of that knowledge and the resulting dynamics; and, it makes these available to others.

The value of a simulation, based on current or historical characteristics, is that it represents the dynamics of the simulated system and therefore allows study, investigation, experimentation, and modification without disturbing the actual system. Alternatively, a simulation based on anticipated or planned characteristics represents the dynamic behavior of a system that doesn't presently exist; users can therefore study the anticipated behavior before it actually exists.

The second tenet is:

> Simulation users gain knowledge and value from the simulation.

This simple statement means that those who look to others to simulate, analyze, and provide answers learn very little about the simulated operation and hence their own environment. To use and to help build a simulation brings a person in direct contact with the simulated environment. In doing so, a person develops a deeper understanding of how the real world system operates and behaves.

Section 1-5 Where is the money?

Simulation is used for a reason: it provides a value-added return for the people and the financial resources it uses. Some of that payback is in hard cash, while the rest is in soft, intangible benefits. Defined savings helps everyone—especially accountants—while the intangible benefits are appreciated by management. To illustrate these benefits, this section presents examples that are based on more than a thousand simulation projects over a wide variety of applications used in various business and service sectors.

As shown in Figure 1.15, benefits of simulation normally fall into four categories: feasibility assessment, cost avoidance, detail design, and operations. Each category has both tangible and intangible benefits associated with it. As with most characterizations, there is always some overlap between the categories. The categories also line up with normal procedures for evaluating capital projects or any change to existing operations.

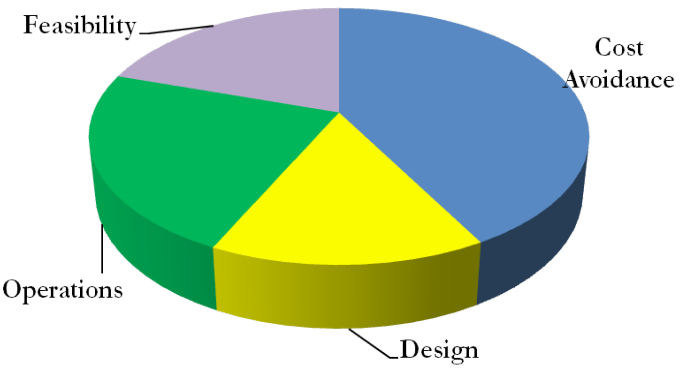


Figure 1.15: Simulation benefit categories.

Feasibility assessment

In the feasibility phase of a project, new concepts are thought of and plans are made to execute them. This is the optimal time to make changes and identify possible problems since only a small amount of effort has been expended thus far. Typical savings during this phase have ranged from 20% to 40% of the total cost incurred during this phase. Figure 1.16 provides some example benefits that may be realized by simulation during the feasibility phase of a project. Direct savings have accrued from reducing the time to obtain an accurate estimate of the cost and benefits of a proposed project. With simulation, the time to consider and prioritize alternative recommendations is also reduced. Time savings can

- Reduced time for accurate estimates
- Faster Lean analysis
- Improved decision making before project is started
- Easier consensus building
- Early identification of possible problems and their solutions
- Sounding board for innovative ideas

Figure 1.16: Feasibility phase savings (20-40% of project cost).

be directly calculated from employee cost/hr and from the value of a faster time to market from business projections.

During the feasibility phase, the value of having a more accurate cost and benefit analysis carries a soft return by allowing those involved to quickly focus on those alternatives with the highest return on investment at the lowest capital costs. Such improved decision making before a project is committed is known to minimize delays and false starts later.

Simulation also makes it easier to involve more people early in the project cycle, thus increasing the amount of buy-in from various stakeholders. It also provides a sounding board for innovative concepts and modifications that can change a project from one that provides a slight improvement to one that makes a significant contribution.

Cost avoidance

Cost Avoidance is a topic that accountants hate but management appreciates. This phase normally occurs during the time between feasibility and final design. It involves funds that were originally allocated to the project but are no longer needed and thus are available for use elsewhere. The dollars are real and often major. They can also be calculated directly. Figure 1.17 provides some example benefits that may be realized by simulation during the cost avoid-ance phase of a project.

- Eliminating non-productive equipment
- Optimizing equipment size
- Minimizing surge and floor space
- Balancing asset utilization and flexibility

Figure 1.17: Cost avoidance (2-40 % of project costs).

A major consumer products manufacturing company was able to save nearly 2 million dollars when simulation showed that a plant layout could provide the necessary production with one less packing line. A drink provider was able to reduce a project by $500,000 by not having to provide additional raw material storage for a planned expansion. Over $300,000 was saved in equipment and floor space when simulation showed smaller sized equipment to be adequate. Such savings have been 2% to 40% of the original project cost estimate.

Simulation also has less tangible benefits at this phase, including the ability to more accurately match equipment and operations with business needs. Using simulation at this phase also forces marketing, engineering, and operations to agree on the precise scope and outcome of a project. It requires the operations side to address issues such as balancing asset utilization with flexibility. For example, high flexibility can be achieved by either having multiple pieces of diagnostic equipment in a hospital, each doing a specific analysis, or a few multipurpose machines capable of performing many procedures. The former case carries higher costs while the latter may result in a long line of customers/patients waiting for its use. Simulation provides the platform to effectively address these types of issues.

Detail design

- Faster resolution of design issues or changes
- Consistent, reusable engineering design and documentation
- Reduced cost caused by design errors
- Faster testing and startup

Figure 1.18: Design/Execution savings (3-30% of project costs).

Simulation provides tangible benefits during the detailed design of any proposed operation or change to an existing operation. Savings of 3% to 30% are typical. For example, a major equipment manufacturing company used simulations to quickly resolve design issues and saw a 15% reduction in engineering costs. The company also accrued savings because of simulation's ability to provide consistent, reusable engineering design and documentation. Since this company guaranteed performance of their equipment, the simulations meant faster testing and startup periods. Figure 1.18 provides some example the benefits that may be realized by simulation during the detailed design phase of a project.

Simulation allows those involved to detect design errors early in the final design phase, resulting in less tangible, but still quantifiable, benefits in the form of more savings. A soft drink manufacturer decided to add another bottling line. Cases from the line would be sent to a palletizer that six other lines were already sharing. Based on historical data, the palletizer would have the capacity to handle the new line; however, a simulation taking line dynamics into account showed that the single palletizer would actually be a bottleneck and reduce the plant's total output. A net savings of nearly $600,000 was attributed to finding and rectifying the problem. Savings were based on estimates of the cost of modifying the packing lines after the incorrect design was completed, loss of production time due to the modifications, and loss of profit from being bottlenecked by the palletizer. The net savings were those costs minus the cost of the additional palletizer.

Simulating operations

- Reduced analysis time
- Dynamic representation of operations
- Simple and quick method for trying changes or new ideas
- Accurate assessment of schedule impact
- Quickly analyzing the impact of shared equipment and staffing issues

Figure 1.19: Operations (5-25% of project cost).

Using simulation to model actual operations typically returns savings of 5% to 25%, based on reduced operating costs or increased throughput. As shown in Figure 1.19, much of the savings are predicated based on a simulation being able to reduce analysis time, thus providing a quick method for trying out changes or new ideas. Such analysis is especially important when shared equipment and staffing issues are involved.

Most importantly, a simulation can show the impact of an operating schedule on performance and then guide management in obtaining optimal operations. For example, simulation showed a

consumer products company that if they would simply change the transfer logic between their batch operations and filling line, they could gain a needed 15% increase in production without any capital outlay. Similarly, a shipbuilder gained 10% to 20% in throughput in a bottleneck shop by re-sequencing work.

Section 1-6 When and where to use simulation

Before learning how to build simulation models and analyze their results, it is important to understand what models are and how they can be used. This section illustrates many types of operational problems that simulation can, and has addressed. The "Baker's Dozen" of examples are certainly not exhaustive, but are based on actual experience and demonstrate the versatility and power of simulation. Each issue includes a screenshot of a simulation model used to address the issue.

Issue 1: Understanding the bottlenecks that limit an operations system's performance.

Every system has a bottleneck that restricts how well the system operates. Some companies base their performance calculations on the operation downstream of the bottleneck. Bottlenecks in processing operations are often identified and known; however, they are often transient and change as the system dynamics change.

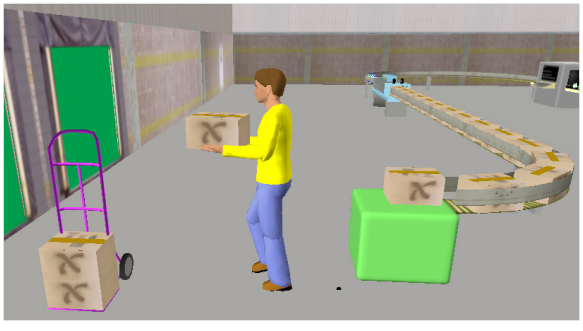


Figure 1.20: Bottlenecks can appear anywhere.

There are also many ways to alleviate a bottleneck. In manufacturing, equipment from another production line can be used. In a supermarket, a new checkout line can be opened. At times there is no single bottleneck to focus on. Whenever a step in a process stops, it becomes the bottleneck. For a restaurant, the bottlenecks may change according to the seasons. Dynamically changing bottlenecks makes a system very difficult, if not impossible, to analyze with simple spreadsheet methods. The dynamic focus of simulation makes it an excellent tool for bottleneck analysis.

Issue 2: Addressing the accumulation of material in a manufacturing process.

Figure 1.21: Accumulation surge before a filler.

Modern manufacturing theory focuses on the elimination of any unnecessary steps or accumulation of work materials in a process. Surge areas, which are used to store materials ahead of equipment or workstations, are targets of Lean thinking; however, Lean operations do not necessarily mean the eliminations of all surge areas. Point-of-use storage is still an integral part of modern, efficient operations.

The sizing of surge areas and buffers are not intuitively obvious as they are based on the dynamic behavior of the operation. Simulation is used to identify where the surges are needed and what size they should be so that a minimal amount of inventory is accumulated in a production area.

Issue 3: Analyzing new concepts.

Simulation is often used to analyze the impact of modified or new operating practices or policies. Testing new concepts with simulation is much faster and more economical than trying to do it on the actual system. The im-pact of reliability assumptions on equipment can be tested, as well as the new methods. If there are absolutely no historical data to use in the simulation, then the simulation can be used to conduct a sensitivity analysis in which variables are modified in order to study that impact on system performance. The results are used to specify the operational characteristics of the equipment or used to estimate changes in operation.

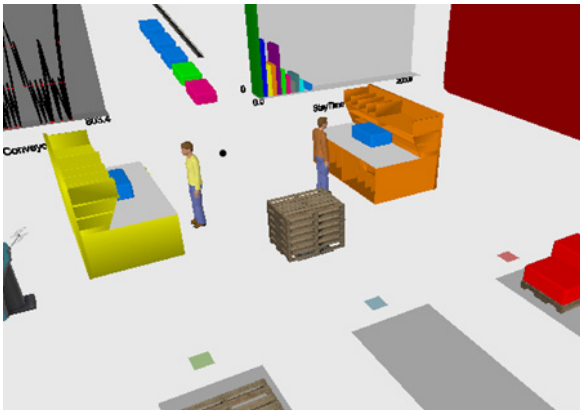


Figure 1.22: Evaluating alternatives before implementation.

Issue 4: Studying combined processing and packaging systems.

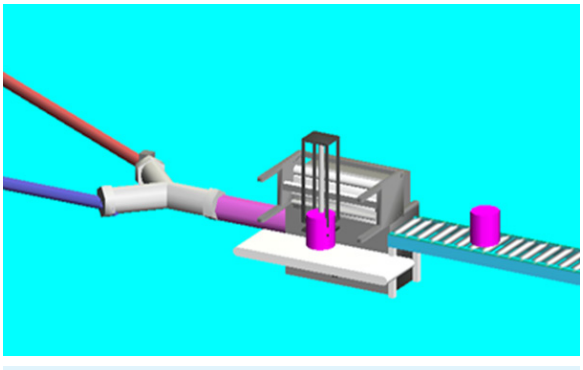


Figure 1.23: Paint blending and filling operations are closely coupled.

Systems that both process and package goods are prime candidates for simulation analysis. A major issue is that the process and packaging areas are closely coupled, making them highly dependent on each other. The complexity is increased further if the processing is fluid or batch in nature and the packaging line is discrete. A simple continuous blending system for paint that goes to a packaging operation is an example of a closely coupled system. Once the operation commences, disruptions at any point can cause significant production disruptions.

Issue 5: Balancing cost and service level.

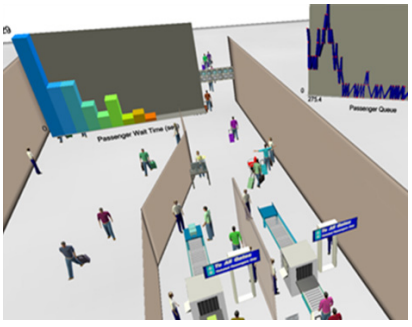


Figure 1.24: Improving airport security system operations.

At an airport security check point, travelers who have their tickets and identification verified may be sent to one of many inspection lines depending on several factors: ticket status, line length, and need for special screening. Simulation is usually used to design and analyze these systems. The simulations help determine the number of lines, amount of equipment, and staffing levels that should be used throughout the hours of operation.

This determination must balance customer service with cost of operations. Simulation is used to evaluate the efficiency of such proposed concepts as frequent flyer lines and pre-screening background checks for certain types of travelers.

Issue 6: Validating complex material transfer procedures.

In an effort to make operations very flexible, material is often routed from one operation to another based on various rules. In a cheese plant, material is passed from fermentation units to processing areas based on the history of the receiving unit. These rules have to take into account the nature and age of the material in each location and the cleaning record of the receiving vessel. Simulation of such operating rules is used for training as well as ensuring that the rules do not become unintentional bottlenecks.

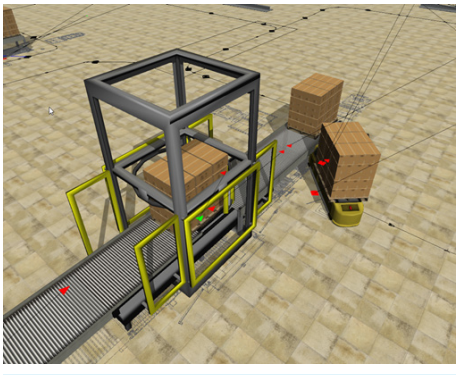


Figure 1.25: Multiple transfer paths.

Issue 7: Checking batch operating sequences.

Batch operations contain a sequence of events that often change with the product. In fluid operations, the batch is usually described in a way prescribed by industry standards. The description starts with a recipe table listing the amount of each material to be added and a separate table lists the steps in the sequence. The use of a recipe and step table has become the standard for specifying batch operations. For multiple products, separate recipe and step tables have to be maintained. Logic is employed to handle any additional cleaning required and to direct the output of the batch to its destination. Simulation is needed to analyze the number of batches that can be made especially if some of the steps can be subject to random events, such as the requirement of an operator or delays from test results.

Step Table				Recipe Table		
Step #	Description	Delay		Ingredient	Amount	Step#
1	Add Water	0		Water	2500	1
2	Add Citrate	10		Citrate	150	2
3	Mix	30		Premix	225	4
4	Add Premix	0		Water	1000	5
5	Add Water	65				
6	Mix	65				

Figure 1.26: Batch specification tables.

Issue 8: Analyzing the impact of production schedules on operations.

Experience has shown that the greatest impact on resources and performance is the scheduling of products or people through the system. In manufacturing, a production facility is often at the mercy of a marketing or sales organization that dictates the order of material flow. Simulations that only focus on producing a single item or a set sequence of items are not robust enough to analyze performance over an extended period of time.

Additionally, when simulating a schedule, operations usually have to be adjusted for each item going through the system, and monitoring of the line is needed in order to know when the scheduled task has been completed.

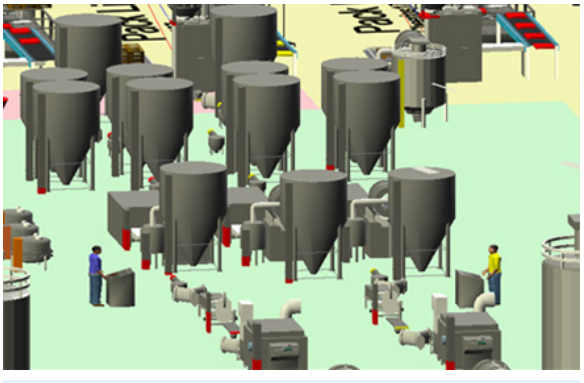


Figure 1.27: Two independent production lines sharing storage and packing equipment.

Operations where more than one area that can be scheduled and the separate production areas can interact can be complex to analyze. Similarly, complexity occurs if the operations must be carried out in a particular sequence without exception and shared equipment is involved. In both these cases simulation provides a robust method that takes complexities into account.

At times the steps required to clean or set up an operation can be more complex than the operation itself. Food processing is an example of this as it has extensive requirements for sanitation and allergy cleaning that depend on the particular product as well as the elapsed time of operation. In addition to food processing, hospital requirements for sanitation and cleaning are also extensive. In all these cases, the time required to perform the activities varies. Simulation takes these dynamics into account and allows performance metrics to be accurately calculated.

Issue 9: Improving raw material distribution within a facility.

The beginning or end of a system containing many operations is often a shared resource. A common example is the use of a shared raw material facility or shared warehouse area.

While bottlenecks are known for the individual operations, the impact of the shared facilities is often neglected until it actually causes a loss in performance.

The speed of getting material from an incoming dock to its destination is often dependent on the routing and availability of transportation resources. The same is true of moving finished goods to their correct warehouse or loading dock destination. In medical simulations, shared resources of X-Ray or lab areas cause similar problems. Because of the interrelated nature of these operations, simulation is required.

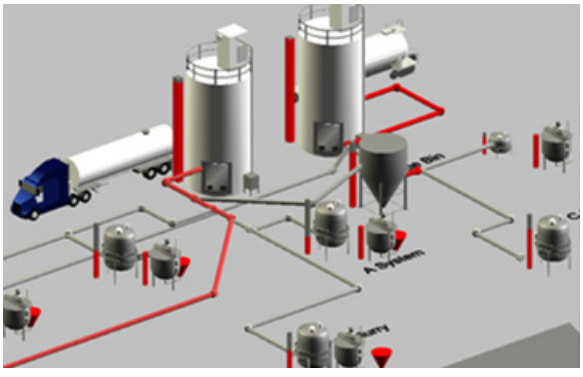


Figure 1.28: Distributing raw material to multiple plant sections.

Issue 10: Effectively coordinating a production line.

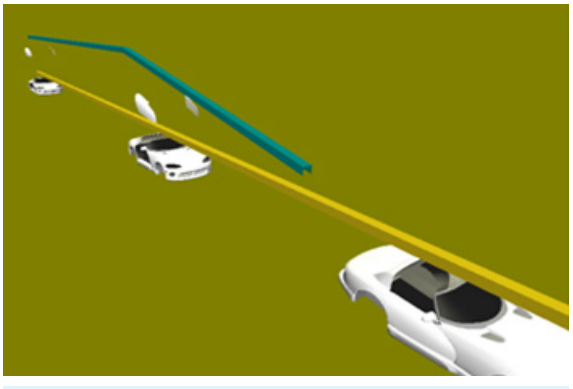


Figure 1.29: Automotive assembly lines require coordinated operations.

Coordination with other areas in a system is similar to the problems caused by shared facilities as discussed in Issue 9. In that case, however, each processing operation is still independent of the shared equipment. With coordination, two or more independent operations must interact at the exact same time. For example, in an automobile production line, components manufactured on independent lines must arrive at an assembly point at the same time in order to fill a particular order.

In the food industry, various components must join to form "variety packs" of a finished product. The traditional way of dealing with such coordination is to use large surge systems. In a lean environment, this process is not considered efficient, and simulation is used to develop line configurations, operating procedures, and production rate strategies, as an alternative to surge systems.

Coordination of production areas is further complicated by the stopping and starting of parts of the production lines. Equipment downtime is generally considered to be the result of some failure; however, predicting downtimes is complex as there are often competing failures in different sections of a machine. Downtimes can also be caused by outside influences, such as operator break times, scheduled meeting times, and even preventive maintenance. Taking competing downtimes into account and correctly entering them into a simulation is important for representing system behavior and conducting a comprehensive analysis. Simulation is a valuable tool for studying the impact of downtimes and how to overcome them.

Issue 11: Working with operations that involve people.

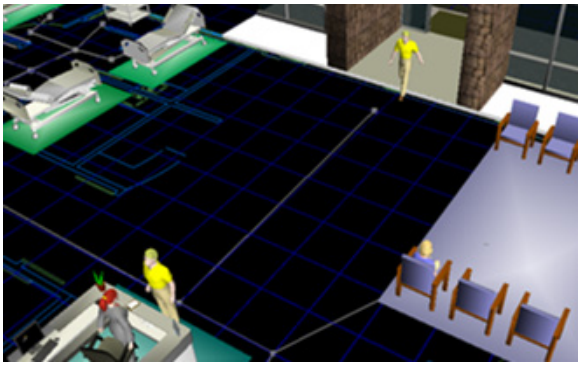


Figure 1.30: Healthcare simulations track people and their medical needs.

The behavior of people taking part in a simulated system is the most difficult aspect to characterize or model. Individual performance rates and logical choices vary widely. Simulation can take many of these characteristics into account when operators, fork truck drivers, bank tellers, healthcare personnel, ticket takers, airport security, or other people are involved. The improvement of healthcare facilities is of particular importance today. Simulation provides a method to analyze how patient needs can be both effectively and efficiently handled in facilities such as hospitals, emergency rooms and clinics.

Issue 12: Developing logic for material transfer.

Making significant improvements to operations can involve specialized logic normally applied through control systems. Such logic includes dynamically changing rates that depend on operating conditions, manipulating flows to maintain coordinated operations, or using photo eyes to control movement of material. Such logic can be complex, and accurately predicting its impact can be difficult without simulating the system.

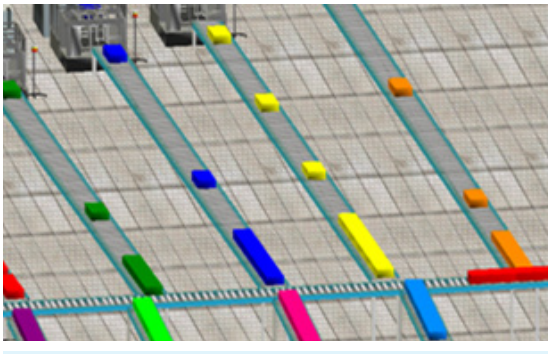


Figure 1.31: Conveyor photoeyes and logic control material traffic.

Issue 13: Designing for just-in-time manufacturing.

Providing goods and services for a just-in-time production system means producing materials at a rate that matches the market demand. Such an operation is also called a "pull" system and requires precise control of the manufacturing process. Whenever demand changes, the processing systems must adapt to the demand in both quantity and quality. Significant logic may be employed to assure that the desired product is correctly managed. Additional logic may require that one operation be suspended to produce another product and then return to the prior operation. A pull manufacturing environment is best designed and analyzed with simulation.

Figure 1.32: Pies are made to order.

Degree of complexity for various operations

The previous examples indicate complexity in operations and should raise red flags signaling that simulation should be used for design, analysis, or validation. Table 1.1 summarizes a number of conditions which can increase the degree of operational complexity. Simulation is especially useful when the degree of complexity is moderate or greater.

System characteristic	Degree of complexity
Straight through operations--Known rates--known upsets	slight
Variables dependent on service, product, or operating conditions	slight-moderate
Lean design with random events or external changes	moderate
Bottlenecks with by-pass or alternative logic	moderate
Bottlenecks that change by conditions	moderate
Surge or Kanban areas needed	moderate
Introduction of new operations or services	moderate
Reliability important--failure modes, interactions	moderate
Interaction between fluid and discrete manufacturing areas	moderate-high
Multiple possible material transitions or transit paths	moderate-high
Batching operations	moderate-high
Scheduling influence on performance	moderate-high
products or services must be executed in a particular sequence	moderate-high
Specialized setup or cleaning requirement	moderate-high
Shared equipment or services	moderate-high
Coordination required with other areas or people	high
Competing downtimes (e.g., reliability, break or lunch times, shut downs)	high
Interaction of services, equipment, and people	high
Specialized logic (e.g., dynamically adjusting variables based on condition)	high
Pull manufacturing (make to order) operations	high

Table 1.1: Degree of complexity of typical operations.

Section 1-7 Simulation users

Not everyone who uses simulation will have the time or inclination to become proficient in all of its aspects. As discussed in this book, there are many requirements for a simulation project to be successful. A wide variety of skills are needed; however, in a simulation project, everyone does not need to know all aspects. Users should be able to find parts of a simulation project that match their needs as well as their abilities. Users should also be able to increase their simulation knowledge at their own pace. The remainder of this book follows the progression of a user from having a casual relationship with simulation to one

who is deeply involved in advanced topics. The benefits, however, are available to all levels of users.

Basic background for every user

Everyone who wants to use simulation to help analyze and solve problems must possess the same basic knowledge of the theoretical background; they must also understand what simulations can and cannot accomplish. Often managers, business unit leaders, marketing personnel, engineers, or operations personnel such as team leaders may request that a simulation help them solve a problem.

These individuals are most likely the owner of the problem or issue to be analyzed by simulation. They typically have much more knowledge about the purpose and scope of the simulation than of the simulation application itself.

Making an informed decision based on a simulation and having confidence in the results requires a general knowledge of where simulations have been used and what sort of problems they have solved. Users should understand the economics of simulation, especially where simulations have benefitted their organizations. It is also important to know how simulations interact with other corporate systems such as Lean Teams or productions planning programs.

Even though someone may not be involved in the simulation process, they may still want to open and run pre-built simulations. The user interfaces they use are usually custom built to make controls intuitive to the user. By running the simulation, even a non-simulation builder can visually validate the simulation against historical data or check the operating characteristics of the simulation. This interaction takes place through custom interfaces, data tables, and separate applications, such as *MS Excel*. Custom built reports are usually created for the person running the simulation.

Individuals who will go on to directly build simulations should also start with this background before starting the simulation building process.

The Occasional User

The Occasional User is a person who occasionally uses simulation to solve specific issues or to analyze an operation. At this level, simulation is often used only once or twice a year. Training must be short, the interfaces intuitive, and re-training (when simulation is again used after an extended lapse of time) minimal. To be effective, these users need a large library of standard functionality that allows them to build realistic simulations with minimal effort.

The Occasional User begins with a basic understanding of simulation and its use. This user must be able to discuss the system to be simulated with people directly involved with the physical process. The Occasional User should also be familiar with modifying data in tables and MS Excel for both input and report functions. The Occasional User must have sufficient understanding of the system being simulated to make rational decisions

about changes to the input variables and be able to understand the output and performance measures.

To get the most out of a simulation, the Occasional User may define or help specify a simulation that a more advanced user can build. For that purpose, the Occasional User constructs what is known as an Object Flow Diagram (OFD) that defines material flow, operating parameters, and logic. A value stream map may act as a starting point, but the OFD is required in order to add information that is not included in the standard value stream map. The method for building an OFD is detailed in a later chapter.

The capabilities and required skills of an Occasional User are summarized below:

Occasional User capabilities

- Have a basic understanding of the elements in simulation software
- Be able to visualize how various simulation objects can represent the physical environment and the activities being simulated
- Validate system operating characteristics by observing the simulation as it runs
- Understand when the complexity of the simulation requires more advanced skills

Occasional User skills

- Use standard logic and functionality of the software, including data for objects and operational scenarios
- Use reports to analyze performance measures
- Build basic discrete models, including operators and transportation
- Define and analyze simulation output and object statistics
- Understand the steps in a simulation project

The Intermediate User

The Intermediate User is an active simulation practitioner and typically operates as a problem solver as his or her primary focus, such as an engineer in a corporate or operations group. Intermediate Users also perform in logistics, research, and business unit functions. For this user, simulation is a tool that is available to help with his or her main responsibility.

The Intermediate User has all the skills of the Occasional User and works closely with him or her to analyze systems and resolve issues. Like the Occasional User, the Intermediate User builds basic simulations. The Intermediate User, however, brings a better, more realistic understanding of simulating reliability and random events—issues that are critical

for any model to accurately represent a process. This user also has more insight into how to gather data for use in the simulation as well as how to analyze and interpret results.

This user may utilize a tool such as an Object Flow Diagram (OFD) (which is described in a later chapter) to better understand and define the logic required for the simulation, including the sequence of events for operations and the logic that determines material flow paths.

Lastly, while this level of usage still doesn't require a programming background, the user must be willing and able to understand the basic software application structure and architecture.

The required skills and capabilities of an Intermediate User are listed below:

Intermediate User capabilities

- All capabilities and skills of the Occasional User
- Understand the theory and practical aspects of reliability, randomness, and data analysis
- Expand/modify existing models

Intermediate User skills

- Accurately include primary and competing failure paths for reliability
- Correctly establish random variable parameters
- Set up design of experiments and analyze the corresponding data

The Advanced User

Building on the skills of both Occasional and Intermediate Users, the Advanced User can build complex models with extensive custom logic. This user's primary responsibility isn't usually simulation alone, but some internal consulting function. He or she will be called on to help simulate and analyze conditions or systems that are beyond the scope of other users. This person is most likely the primary technical contact when an external simulation provider is involved.

In larger organizations, the Advanced User may be responsible for maintaining copies of the simulation software and for developing custom libraries. This user can modify not only the basic characteristics of three dimensional simulation objects but also the image itself. As a teacher/consultant, the Advanced User works with other users to continually develop simulation skills.

By knowing the underlying software architecture and having proficiency in basic programming, the Advanced User can add additional logic to objects, such as photo eyes on conveyors and complex maneuvers of transportation vehicles. The logic may also

include dynamic messaging between areas in the simulation. These capabilities become an important skill when totally new logic—logic not covered by standard choices in the simulation software—is required for an operation. The Advanced User will find the most benefit when he or she is able to utilize logic and concepts that fall outside the range of normal thought processes. This user is also able to visualize how basic discrete simulation can be used for non-typical applications, such as for fluid flow and human interactions.

The capabilities and skills of an Advanced User are listed below:

Advanced User capabilities

- All capabilities and skills of Occasional and Intermediate Users
- Understand when more complexity is required in a simulation.
- Visualize how basic simulation software can be used for new applications
- Assist the Occasional and Intermediate Users in building and analyzing simulations

Advanced User skills

- Advanced application functions
- Programming skills that include a basic level of scripting functions such as setting up equations and logic such as If statements.
- Expanded use of simulation application tools

Ideally, the vast majority of users would have both Occasional and Intermediate user skills. Modern simulation software makes obtaining this level of skill an attainable goal. Since all skill levels are often needed at some point in order to obtain the most benefit from simulation, organizations with only Occasional Users will often outsource the advanced skills to the software vendor, third party consultants, or engineering firms.

Large organizations may have a distribution of users like the one shown in Figure 1.33; however, the value of simulation will only be realized if an organization maintains at least one person at, or above, the level of Occasional User.

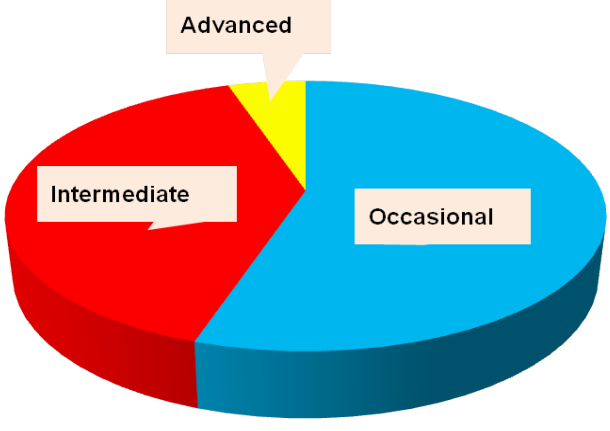


Figure 1.33: Typical distribution of simulation users.

Section 1-8 Summary

The best way to wrap up this introductory chapter and to begin a journey through simulation modeling and analysis is through a quote from one of the pioneers of discrete-event simulation, A. Alan B. Pritsker. The following is from Dr. Pritsker's keynote address (Pritsker, 1989) that he delivered at the 1989 Winter Simulation Conference.

> "Simulation works because it deals with reality. We simulate real systems. We get closer to the system than any other type of modeler. We study the old system, collect data, understand first principles about the system, check out procedures in use before we start modeling, and we test proposed solutions against current operations or baseline designs. We do not force a system into a preconceived normative model. We strive to have our models used and our best alternatives implemented. We stay with a problem until a solution is implemented. We recognize that the model upon which we make our recommendations contains additional information and insights that are useful during implementation." (p.1)
>
> "It is ironic that simulation, a word that connotes a sense of unrealness, works because of the ability to include realism in the models that underlie a simulation analysis." (p. 2)

References

1. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, Chichester, England: John Wiley & Sons, Ltd.

2. Checkland,P. (1981) *Systems Thinking, Systems Practice*, Chichester, England: John Wiley & Sons, Ltd.

3. Pidd, M. (2003). *Tools for Thinking, Modeling in Management Science*, 2nd Edition, Chichester, England: John Wiley & Sons, Ltd.

4. Pritsker, A. A. B. 1989. "Keynote Address, Why Simulation Works," in *Proceedings of the 1989 Winter Simulation Conference*, edited by E. A. MacNair, K. J. Musselman, and P. Heidelberger, 1-6. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chapter 1 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important—in the chapter.
2. Define *operations system* and provide some examples.
3. Identify some behavioral characteristics of operations systems that make them difficult to analyze.
4. Define simulation in terms of decision support.
5. Describe how model-based decision support systems enhance the decision-making process.
6. Describe the paradigm shift that has occurred for how simulation is used.
7. Identify some ways in which simulation is used in everyday life.
8. Describe some areas besides manufacturing that use simulation.
9. Describe the two basic tenets of simulation.
10. Provide examples of benefits that result from using simulation; consider each of the four benefit categories.
11. Describe the skills that are desirable in each of the three types of simulation users.
12. Describe the capabilities commonly found in each of the three types of simulation users.
13. Provide specific examples of operations problems that simulation can address.
14. Describe some system characteristics that affect the degree of operational complexity and where simulation may be useful.
15. Provide several reasons why simulation works.

Chapter 2 Introduction to Modeling and Analysis of Queueing Systems

Objectives

To understand and evaluate queueing systems by 1) defining the structure, operation, modeling, and analysis of stochastic queuing systems; 2) describing the use of analytic models of queuing systems to assess system performance; 3) introducing discrete-event simulation models of queueing systems and describes their use to assess system performance; 4) explaining the benefits of using simulation models compared to analytic models.

Queueing systems are so prevalent in our society that we dedicate an entire chapter to the definition and description of basic queueing systems. Many production systems (those that produce goods and those that produce services) are based on the notion of queueing. Understanding how queueing systems function and behave is critical to solving problems and making decisions about such systems, and of course, building and analyzing models of these systems.

This chapter begins by describing the prevalence of queueing systems in our everyday world. Subsequently, we describe the elements of queueing systems, the characteristics of those elements, and typical measures that are used to express the behavior of such systems. Other than studying the actual system (e.g., through observation), there are two approaches to studying queueing systems through modeling—analysis and simulation. We briefly discuss both approaches so the reader understands the benefits and limitations of each approach. Of course, our focus is on the simulation approach since it is the subject of this book; however, later in this chapter we use analytical models to illustrate some of the basic behaviors of simple queueing systems. These simple systems are typically building blocks or components of larger, complex systems. The concepts and behaviors described in this chapter form the basis of discussions of simulation methodologies and applications and are used throughout the book.

Section 2-1 Prevalence of queueing

Queues are prevalent throughout our work and personal lives. Think about it—waiting lines, buffers, storage locations, etc.—they are everywhere! They are a part of what is generally referred to as a queueing system. While such systems are also commonly referred to as waiting-line systems, in this book, we use the term queueing system. Once you understand what a queueing system is, it is not difficult to see how much of what we, and our businesses, do revolves around queueing systems. Table 2.1 provides a few examples of situations where queueing systems exist and some of the issues that these systems generate. As you will see illustrated throughout this book, simulation is a great means to analyze these types of problems.

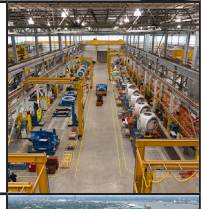
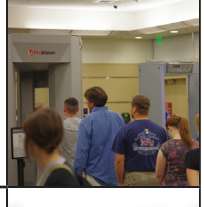
Queuing situation	Problems to be addressed	
Aircraft waiting to be serviced	What maintenance policy improves throughput?	
Keeping the bottleneck resource busy	How much inventory (buffer) is needed to keep the bottleneck resource from being idle (starving)?	
Ship waiting on a berth at a port	What is the effect of adding additional resources (e.g., loading/unloading crane), or a modified shift schedule?	
Insurance claims waiting to be processed	How can case workers be assigned to reduce the time the claim is in the system?	
Passengers waiting to check in at an airport	How can the system be designed to expedite the checking of both priority and basic passengers?	
Truck waiting to be loaded at a cross-docking facility	How should fork trucks be assigned to better serve waiting trucks? How can truck arrivals be better scheduled to reduce wait time?	

Table 2.1: Examples of queuing systems and problems that need analysis.

Queueing systems are the focus of many process improvement projects, including such methodologies as Lean and Six Sigma [1]. Lean focuses on reducing waste, in any type of business system, not just in manufacturing. Waste can take many forms, not just in terms of defects, but in waiting time, inventory, transport, overproduction, etc. All these forms of waste are prevalent in queueing systems. Six Sigma focuses on reducing variability in any type of business system, also not just in manufacturing.

Section 2-2 Basic elements of queueing

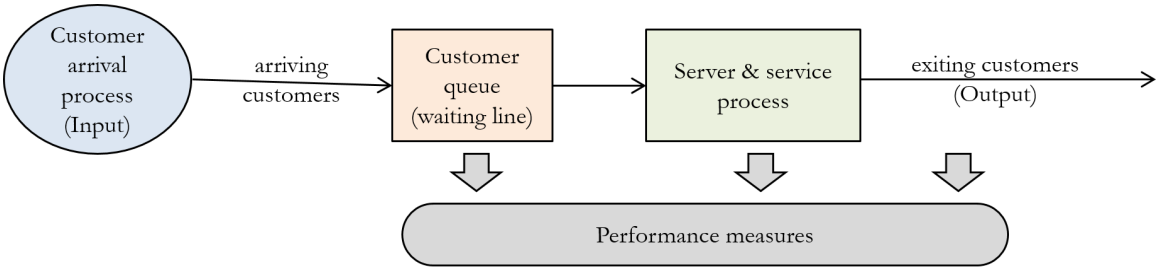
In order to define and describe queueing systems, we first need to define a system. According to INCOSE [5], a system is a "collection of different elements that together produce results not obtainable by the elements alone." Elements, or parts of a system, include all of the things required to produce systems-level results, such as people, hardware, software, facilities, policies, documents, etc. These elements are organized or arranged into relationships. The value added by a system is often created by these relationships. Interactions among elements through these relationships create properties and behaviors that are different than the individual elements.

All of the systems described in this book are socio-technical systems. Emery and Trist [3] originally coined this term in 1960 to describe systems that involve a complex interaction between humans, machines and the environmental aspects of the work system. Of course, this interaction is commonplace in most of today's enterprise systems [2]; therefore, all of these factors—people, machines and context—need to be considered when developing models of queueing systems.

When defining a system, it is important to explicitly define the system boundaries that delineate what is inside and what is outside (i.e., the environment) of a system. Without clearly defining system boundaries, the analysis of a system could become unwieldy. For example, to address a problem in a production system, it must be clear whether the analysis is of a work cell, line, facility, or supply chain. Boundaries decide the project's scope. All stakeholders, all those involved in a project, must understand what is being considered, and just as importantly, what is not being considered.

Queueing systems are composed of three main elements: customers, servers, and queues. The interactions of these elements result in the behavior of a queueing system. The behavior is usually characterized in terms of performance measures, also referred to as key performance indices (KPIs). A simple queueing system is illustrated in Figure 2.1. Queueing systems are usually much more complex than the one illustrated in Figure 2.1; however, the more complex systems are typically an interconnected network of simpler systems.

Customer is the general term for anything that requires a service, such as a material needing to be machined or inspected, an order needing to be picked from a warehouse, packaged, and shipped, a patient needing a medical examination, etc. Customers can be people (e.g., patients, passengers), things (e.g., parts, transporters), or something more conceptual or abstract (e.g., orders, phone calls). Customers move through a system; their movement over time is a main contributor to the system's dynamic behavior.



Idle time → Waiting time on server (waiting for customers)

Figure 2.1: Basic elements of a simple queuing system.

Server is the general term for anything that provides a service, such as a machine, inspector, order handler, fork truck, medical imaging device, nurse, etc. Oftentimes systems are composed of multiple possible servers (e.g., multiple places to check out in a grocery store or to serve customers). If these servers are identical, or at least assumed so, they are referred to as parallel servers and the system is referred to as a multi-server system.

Queues are holding areas for customers awaiting access to a server, such as passengers awaiting check in, material awaiting machining or inspection, an order awaiting picking and packing, a patient awaiting a medical examination, etc.

The interactions between customers and servers result in a fundamental tradeoff between the cost of customers waiting for service and the cost of providing the service. This tradeoff is the basis of most queueing system analyses; the goal of the analyses is to find the best balance between the two costs since they are inversely related (i.e., decreasing the cost of waiting for the customer usually involves increasing the cost of providing the service, and vice versa). As will be discussed later in the chapter, we can also view the analysis of queueing systems as a tradeoff of wait times both for the customer (waiting for service) and wait time for the server (idle time while waiting for customers).

Lead time is typically defined as the time between when a request for service is made and when the request is fulfilled. Two key components of lead time are process times and wait times. The wait and process times of individual processes accumulate and result in lead time. This is illustrated by the Gantt chart in Figure 2.2, which includes a sample of six customers that proceed through three sequential processes. While the lead times appear similar in duration across this small sample, individual customer perceptions of the system would likely be quite different. Since Customer 2's wait time is very large relative its three very short process times, it would likely view the system unfavorably. In contrast, Customer 5's

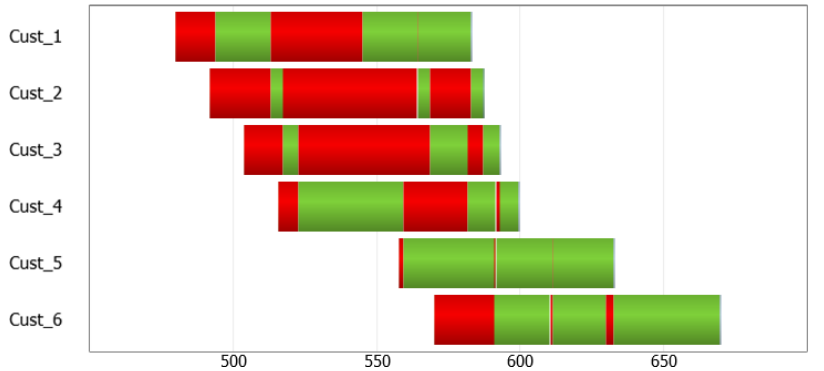


Figure 2.2: Lead time is composed of process time and wait time.

wait time is extremely small relative to its process time; therefore, it would likely view the system performance very favorably.

Wait time is considered both in absolute and relative terms. An example of an absolute concern is where the quality of item is time sensitive (e.g., the amount of time before a product will spoil). On the other hand, many wait times are considered not in an absolute sense, but relatively, commonly relative to process time. The perceived value of the service also is a key consideration. For example, most people expect a low ratio of wait time to process time when purchasing a common product, but will accept a much higher ratio when purchasing tickets to a highly desired concert or sporting event.

Section 2-3 Characteristics of queueing system elements

This section defines and describes the characteristics of customers, servers, and queues, the three main elements of queueing systems.

Customer Characteristics

When modeling and analyzing queueing systems, performance is oftentimes determined mainly by the characteristics of the customers, including the customer arrival process, type, queueing behavior, selection of server, and calling population. Each is discussed below:

Customer arrival process

The customer arrival process is defined in terms of the method of arrival and the stationarity of the process.

The method of arrival (i.e., how customers arrive into a system) is specified in one of two primary ways based on either (1) the time between customer arrivals (referred to as *interarrival time*) or (2) a schedule. Interarrival times are typically random variables and are specified by probability distributions.

Oftentimes, the arrival process is expressed in terms of the arrival rate and not the actual interarrival times. For example, if the arrival process is defined as customers arriving at an average rate of 20 per hour, then the average interarrival time is three minutes (the reciprocal of the mean rate). The relationship between time between arrivals and arrival rate is illustrated in Figure 2.3. For this small example, only four one-hour time periods, the average of the arrival rates per hour (2, 0, 1, 3) is 1.5; alternatively, six total arrivals in four hours results in an average rate of 6/4 = 1.5 customers per hour. This is the reciprocal of the average of the five times between the six arrivals (0.32, 1.46, 0.82, 0.25, 0.49), 0.67 hour or 40 minutes.

The arrival rate is typically assumed constant and is not dependent on the number of customers currently in the system. An arrival process that does not change over time is considered to be stationary. Stationarity is often assumed, but in many cases, the arrival process is non-stationary (e.g., the number of arrivals to a restaurant is not stationary over the entire day; the rate at meal times is different from the rate between meal times).

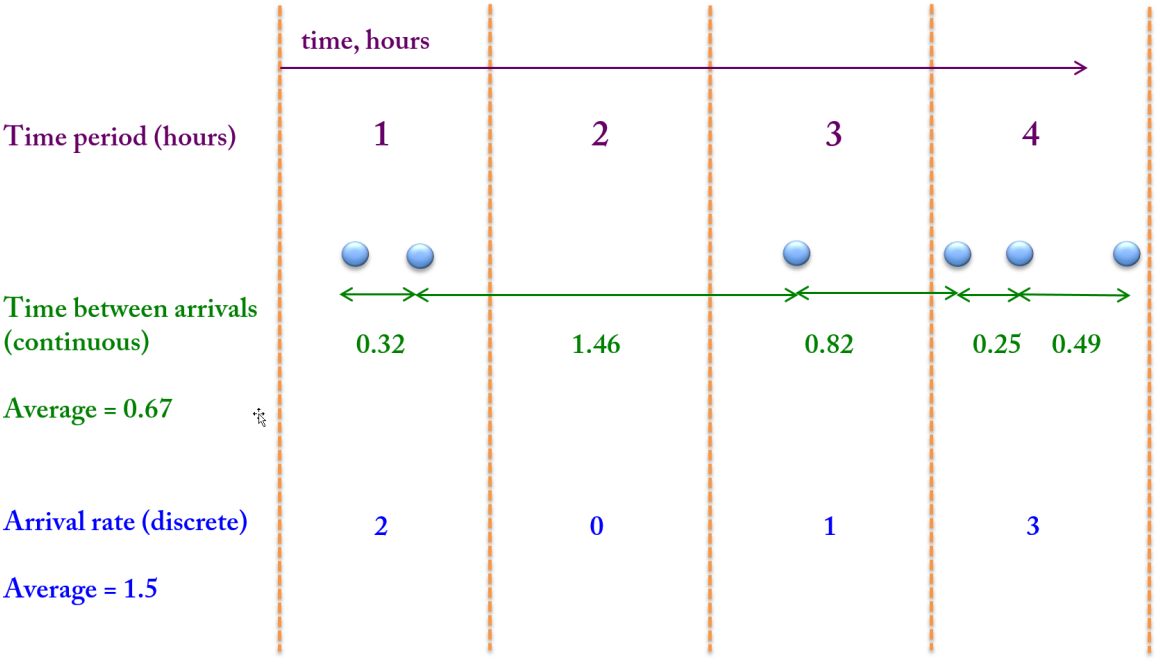


Figure 2.3: Time between arrivals versus arrival rate.

Type

If the basic service process is different for different customers, then we categorize the customers into types. One such difference in type can result from a difference in expected process times. For example, parts being supplied to the military may require more inspection, and thus a longer process time, than those provided to commercial customers.

Different customers may be routed differently depending on their types. For example, frequent flyers are usually processed differently than other types of passengers when checking in for their flight.

Priority systems process customers differently in terms of process speed, number of resources, type of waiting, etc. For example, emergency rooms use a priority system to triage and process patients differently depending on their medical problems.

Queueing behavior

Most of the time arriving customers join a waiting line and wait there until service is performed; however, three alternative customer behaviors are possible:

1. *Balking.* This occurs when a customer arrives to a system, evaluates that the wait time is too long, and leaves the system without joining the queue or obtaining service. The decision is typically based on the number in queue awaiting service. The balking decision can be based on personal preferences or queue capacity. The balking decision can be based on a tradeoff of the perceived wait time versus the value added by the service. For example, the perceived wait time at a checkout station might be considered too long for the purchase of a single item.

Balking can also occur due to the lack of queue capacity. For example, there may be no place for arriving vehicles to park at an auto repair shop.

2. *Reneging.* This occurs when a customer arrives, joins a waiting line, and leaves before receiving service because of the length of the wait. This is almost exclusively based on personal preference and may vary from situation to situation; we all have different thresholds for how long we are willing to wait for a given service.
3. *Jockeying.* This only occurs in multi-queue systems (e.g., when there is a separate queue for each server). Grocery store checkout is a good example—one queue for each checkout station. In this case we often move to a different waiting line when other lines are moving forward faster than the one we are in.

Selection of server

If multiple servers are available, customers may have a preference for which server provides the service. Oftentimes, this is not the case since servers are typically assumed to be of equal capability, and the customer chooses the first available. In queueing system terminology, like servers are referred to as parallel servers.

Calling population

The population of potential customers is either finite or assumed infinite. If assumed infinite, then the arrival rate is not affected by the number of customers currently in the system. This is the most common situation; however, if a system only contains a small number of customers, then the calling population impacts the behavior of the system. For example, if a system is concerned with the repair of ten units of equipment when the units randomly fail (the units of equipment are the customers and the maintenance workers are the servers), then the limited number of customers must be considered.

Server Characteristics

The characteristics of servers can also greatly impact the performance of queueing systems, including, service or process time, process steps, availability, number, resource needs, preemption, and selection of customer.

Service time or process time

Typically service times are random variables and are specified by probability distributions. As noted earlier, the times may vary by customer type. An alternative way of defining the service process is in terms of rate, as described above for the arrival process and illustrated in Figure 2.3 (the relationship between arrival rate and interarrival time). For example, if the average service time is three minutes, then the average service rate is 0.33 customers per minute or 20 customers per hour.

The service rate is typically assumed constant and is not dependent on the number of customers currently in the system. If the server speeds up or slows down based on the number of customers waiting, then the system is said to have a state-dependent service process.

Process steps

Related to the previous item, providing a customer the desired service may involve a set of processes. There are two different types of process steps:

1. *Common.* A set of process steps is applied to all customers (e.g., machine, clean, and inspect a part at a workstation). In this case, the process steps are often combined into one process or service time.
2. *Variable.* Process steps may vary by customer type (e.g., in an emergency room a patient's treatment steps depend on their classification from triage) or be intermittent (e.g., perform an inspection on every fifth part produced on a machine).

Different customers may require different types of service, and thus the set of process steps may be different from customer to customer.

Oftentimes, depending on the type of customer, there is a setup operation that needs to occur before the planned process time. For example, when painting parts, the paint time may be the same for each separate color, but there may be a changeover time when the color changes in order to accommodate a cleaning operation between the applications of different colors. In this case, the setup time and process time need to be separated and applied individually.

Availability

Servers are not always available to provide service; when this happens, servers are considered down. In many cases, models assume 100% server availability (no downtime), at least initially. There are two basic types of downtime—planned and unplanned.

1. *Planned.* In this category, the frequency and duration of the downtimes are known. For example, people take breaks, machines must be resupplied or changed to support a different type of customer, and equipment fails and needs to be repaired. Shift schedules also affect availability, typically as a group of resources.
2. *Unplanned.* Here, the frequency and duration of the downtimes are unknown and are usually specified as probability distributions. For example, equipment failures are typically specified through their reliability.

Number

The number of servers required to provide a satisfactory level of service to customers is often the subject of analyses and a key aspect of process design. The number of servers can change over time (e.g., based on a shift schedule or to better track the level of customer demand).

Resource needs

The server may need other resources to complete the service operation. For example, a process step may require an additional material or a shared resource (one that is shared among servers).

Preemption

A server is preempted if the customer being served is interrupted so that the server can handle another activity. Oftentimes in an emergency room, an arriving patient interrupts the service of the patient currently being treated due to the criticality of arrival's illness. Also, an item in a production process may be preempted to handle a rush job, one that is either late or for an important client.

Selection of customer

Most commonly in multi-server systems, customers are pushed to servers, (i.e., customers go to the first available server); however, in some systems servers pull customers, (i.e., the server chooses the customer). The process of pushing and pulling between customers and servers is further discussed in the next section.

Server-Customer Characteristics

In more complex systems, customers and servers may not act independently; these interactions affect the way a system behaves.

Types

The operation of a system may depend on a relationship between a specific type of customer and a specific type of server. For example, a server's primary capability may be for one type of customer; it can handle other types, just not as effectively.

Transportation

The basic assumption in queueing theory, and in simple simulation models, is that customers transition from queues to servers in zero time; however, in many queueing systems a transport resource is involved. The transport resource is oftentimes shared (i.e., it serves multiple queues or multiple systems). Examples of a shared transport resource include a material handler that moves material between operations in a manufacturing system and an orderly that moves patients in a healthcare system.

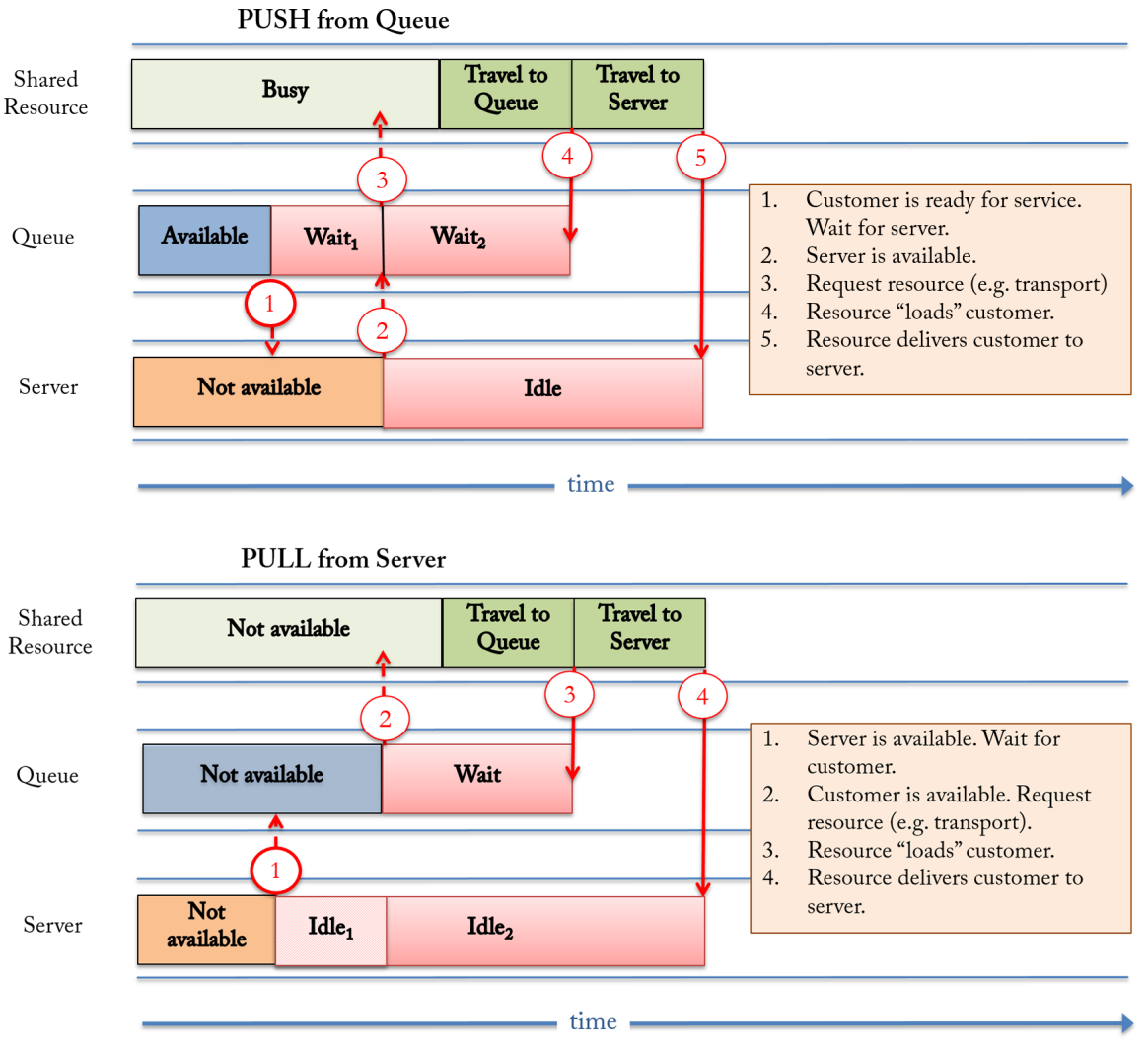


Figure 2.4: Server-Queue interactions in Push and Pull systems

A system behaves quite differently, and thus exhibits significantly different performance, when shared resources are involved. A system's performance depends on the characteristics of the shared resources that are involved, including their availability and utilization, speed, the times to load a customer at a queue and unload a customer at a server, and the distances that shared resources travel. The use of a shared resource that has limited availability will cause additional wait times both for customers and for servers.

Shared resources are commonly considered in simulation models but are difficult to consider in analytic models.

Push versus Pull

In general, it is assumed that customers are pushed from a queue to a server and, as noted above, that the transition time is negligible. Generally, this is a good assumption, at least initially, when defining and building simulation models; however, customers may be pulled from queues by servers. Pull systems are common in Lean applications. It is important to

understand which approach is being used in the system being modeled since the different approaches can cause the system to behave quite differently and thus exhibit significantly different performance.

Figure 2.4 shows two system behaviors that illustrate the differences resulting from pushing versus pulling between a queue and a server and the effect of using a shared resource between a queue and server.

The top portion of Figure 2.4 illustrates an example of pushing from a queue to a server. In a push system the impetus comes from the customer in the queue. In the example above, the customer is ready for service, but the server is unavailable and thus the customer must wait for the server to become available; this is shown as Wait1 on the customer in the queue. If no shared resource is involved, only the customer waits. Since the customer is moved to the server as soon as the server is available, there is no idle time on the server; however, if a shared resource is used to transport the customer to the server, the resource is requested when both the server and the customer are available. If the resource is not available, the customer must wait for the resource to become available and travel to the queue's location and possibly load the customer; this is denoted as Wait on the customer in the queue. If the resource is not available, the server must also wait or become idle—it must wait for the resource to become available, travel to the queue's location, possibly load the customer, travel to the server, and possibly unload the customer; this is denoted as Idle on the server.

The bottom portion of Figure 2.4 provides an illustration of a server pulling a customer from a queue. In a pull system the impetus is from the server. In the example the server is ready to serve, but there is no customer in the selected queue; therefore, the server must wait, or become idle, until a customer is available; this is shown as Idle1 on the server. If no shared resource is involved, only the server waits. Since the customer is moved to the server as soon as the customer is available, there is no wait time on the customer; however, if a shared resource is used to transport the customer to the server, the resource is requested when both the server and the customer are available. If the resource is not available, the customer must wait for the resource to become available, travel to the queue's location, and possibly load the customer; this is denoted as Wait. If the resource is not available, the server must also wait or continue to be idle—it must wait for the resource to become available, travel to the queue's location, possibly load the customer, travel to the server, and possibly unload the customer; this is denoted as Idle.

The examples in Figure 2.4 again illustrate the two types of waiting that must be balanced in queueing systems: customer wait time and server wait time (usually referred to as idle time). Pushing and pulling between queues and servers are commonly considered in simulation models, but are difficult to consider in analytic models.

Queue Characteristics

The characteristics of the queue can also greatly impact the performance of queueing systems, including capacity, queue discipline or ordering, and release quantity.

Capacity

The capacity of a queue is the maximum number of customers that can be in a queue at any one time.

Queue discipline or ordering

The way that customers are arranged in the queue is known as queue discipline, or ordering. A common order is first in, first out (FIFO) (i.e., customers are arranged in the order in which they entered the system). The opposite queue discipline is last in, first out (LIFO) (e.g., desktop file stacking often works this way—the last item placed on a stack is the first one processed). There are also many means of arranging by priority (e.g., triaging patients based on the severity of their illness, processing orders that are late first, and selecting tasks based on shortest expected process time).

Release quantity

Release quantity is the number of customers released to the server when the server is available to perform a service activity. Typically, customers are released one at a time to the server; however, batching may occur at the queue, and in this case multiple customers are released to the server.

Section 2-4 Measures of queueing system performance

Queueing systems almost always involve a design tradeoff between the cost of providing the service and the cost of waiting. In order to perform the tradeoff, various performance measures must be considered.

A measure is typically the average of some state of the system. Common system states are listed below. Depending on the situation, the number and time measures are either expressed in terms of the queue or of the system (queue time + service time).

- Number of customers in a queue/system
- Wait time (time in the queue), time in the system, time between two points in a process, lead time
- Server busy or idle. Other server states are scheduled down (e.g., for an operator break), unscheduled down (e.g., for an equipment failure), waiting for another resource, blocked (customer cannot move to next operation), etc.
- Number of customers that balked
- Number of customers that reneged

Measures can also be in terms of a threshold (e.g., the percentage of customers that waited longer than a certain value). For example, customers may become agitated if they must wait more than 15 minutes to clear airport security, or products may spoil if they are in the system more than eight hours.

Figure 2.2 illustrated the measure of lead time—time between when a customer's request for a service is made and when the request is fulfilled. Lead time is typically composed of the time it takes the system to compete multiple service processes. Within each service process there is a tradeoff of opposing wait times: wait by the customer for service and wait by the server for customers (often referred to as starving, or idleness, the opposite of utilization). How the balance is achieved depends on system characteristics and costs associated with waiting and providing the service.

Figure 2.5 illustrates the interaction between server and customer waiting, including a simple three-step sequential process. In this example, customer X waits for Process 1 and Process 3 because Server 1 and Server 3 are busy serving other customers when Customer X is ready for service. In the time snippet illustrated in Figure 2.5, only Server 2 waits for a customer. Since the customer waits more than the servers, it appears that for this system there is a higher cost associated with a server waiting, or being idle (at least Servers 1 and 3), than the cost of customer waiting. Of course, the time period shown is much too short to draw any meaningful inference.

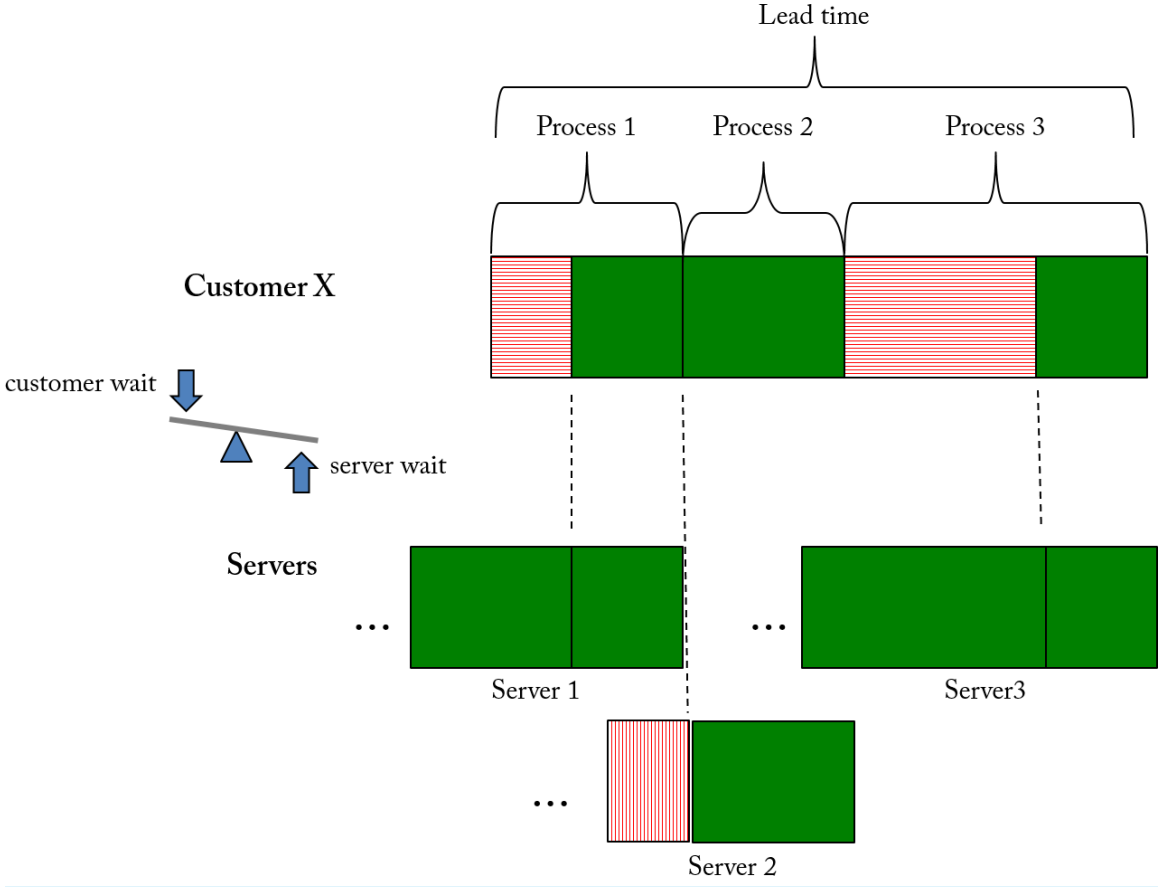
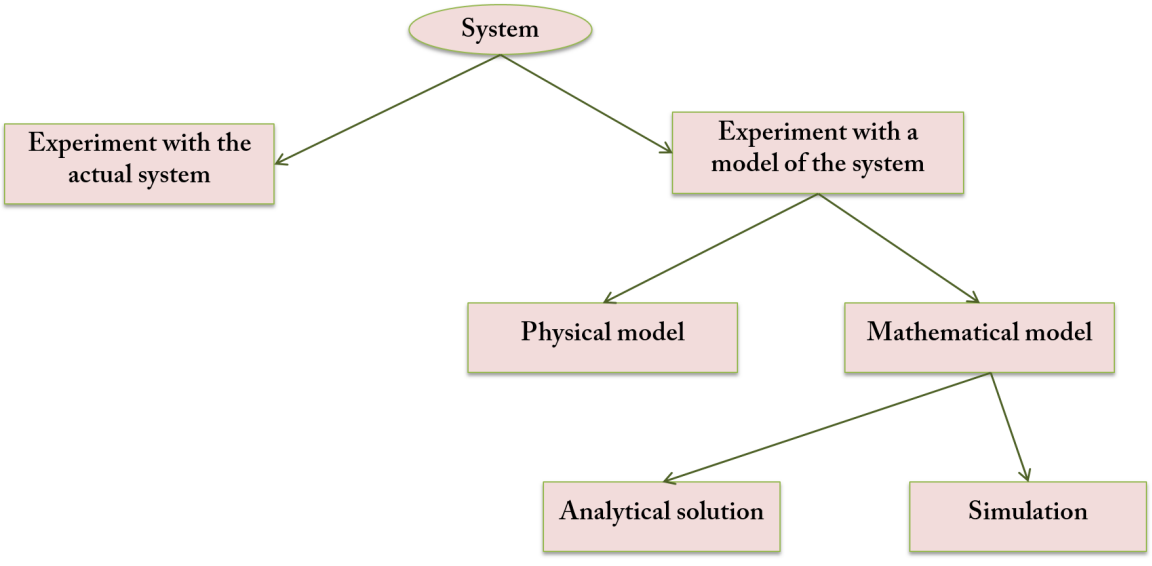


Figure 2.5: Interaction of customer wait time and server wait time.

Section 2-5 Means to study of queueing systems

As shown in Figure 2.6, there are many ways to study or analyze the behavior of a system. The primary delineation is between studying the real system or studying the system via a model (an abstraction of the system). Studying the real system is typically very time consuming, intrusive, and not feasible. Since most work with systems is focused on improvement or change, the proposed system does not even exist. This makes assessing the performance of system alternatives very difficult. It is rarely effective to make a change and then assess the results. For example, if we want to consider the impact of adding an additional production line, it would not be effective to acquire and install the equipment, hire operators, use the new line for a while, then assess its impact; therefore, most studies use models. Models are abstractions of the real system that represent the salient features needed to assess performance.



From: A. Law, *Simulation Modeling and Analysis, 4th Ed,* 2007, p. 5

Figure 2.6: Approaches for studying systems. [6]

Experimenting using a model can either be done using a physical model or a mathematical model. Physical models include mockups of a system, such as models of buildings that are used by architects and wind-tunnel models used by aerospace engineers. Since most system improvement projects focus on operations, however, mathematical models are primarily used.

Mathematical models can take two basic forms: those providing analytic solutions and those involving simulation. In the subsequent section, we will introduce the notion of analytic queueing models—those that can be solved mathematically in closed form. In order to be able to solve the mathematical models, you must apply simplifying assumptions to the models. As you will see in the models in the next section, using the mathematical models is quite easy; however, their assumptions may be too limiting to effectively analyze an operational system. All models have assumptions; otherwise, they would just be clones

of the real system. Simulation models have far fewer restrictions on assumptions but must be built and verified, which takes time.

As will be discussed later in the book, some of the key challenges for simulation modelers are deciding what to model, choosing the model assumptions, and determining how much detail to include in the model. The basic premise is to include only enough detail to answer the questions being posed and no more. Of course, this is easy to say, but hard to do. This is a skill that is developed through professional practice. The main downside of adding more detail than is needed to a model is that it adds unnecessary complexity, which can negatively affect model maintainability.

Section 2-6 Analytic models of queueing systems

Analytic models of queueing systems are available through a widely studied area of operations research called queueing theory. The development of queueing theory began in the early twentieth century through the work of Danish engineer Agner Erlang. A common notation used to classify queueing-system models, referred to as Kendall's notation, is defined as A/B/s/m/p/q where:

A is the arrival pattern, typically a distribution of the arrival rate or interarrival times.

B is the service pattern, typically a distribution of the service rate or service times.

s is the number of servers (assumed to be identical).

m is the maximum number of customers allowed in the system (if not specified, assumed infinite).

p is the size of the calling population (if not specified, assumed infinite).

q is the queue discipline (if not specified, assumed to be FIFO).

The following symbols are used to represent A and B:

M indicates a Poisson or Markovian process where the rate is Poisson distributed and the times are exponentially distributed. The mean arrival rate is denoted as λ (with the mean time between arrivals as $1/\lambda$). The mean service rate is denoted as μ (with the mean service time as $1/\mu$).

E indicates an Erlang distribution (sum of independent, identical exponential distributions).

D indicates deterministic, implying the interarrival or service times are known and constant.

G indicates a general, meaning any distribution.

The simplest queueing-system model is M/M/1. If the arrival rate is assumed to be Poisson distributed (time between arrivals is exponentially distributed), the service rate is Poisson distributed (service time is exponentially distributed), there is one server, the queue has infinite capacity, and the queue discipline is FIFO, then the formulas for key system performance measures are as listed in Figure 2.7. These measures are long-run steady-state averages (i.e., the current state of the system is independent of the initial state).

This model also assumes one type of customer, an infinite calling population of customers, no balking, no reneging, and that the server is available 100% of the time.

$$Server\ utilization,\ \rho = \frac{\lambda}{\mu}$$

$$Mean\ number\ in\ queue,\ L_q = \frac{\rho^2}{(1-\rho)}$$

$$Mean\ time\ in\ queue,\ W_q = \frac{\rho}{\mu(1-\rho)}$$

$$Mean\ number\ in\ system,\ L = \frac{\rho}{(1-\rho)}$$

$$Mean\ time\ in\ system,\ W = \frac{1}{(\mu-\lambda)}$$

Figure 2.7: Measures of performance for the M/M/1 model.

Consider a simple example to illustrate the application of the M/M/1 model. Times between arrivals are exponentially distributed with a mean of one minute. (Alternatively, the arrival rate is Poisson distributed with a mean of one customer per minute). The service times are exponentially distributed with a mean of 0.9 minute. (Alternatively, the service rate is Poisson distributed with a mean of 1/0.9 = 1.11 customers per minute.) The average server utilization, ρ, is therefore 0.9 or 90%. Based on the formulas in Figure 2.7, the average customer wait time, Wq, is 8.1 minutes. Depending on the situation, this would probably be considered a significant, and possibly unacceptable, wait time—waiting on the average 8.1 minutes for 0.9 minutes of service. Another way of expressing this is as a ratio of wait time to service time; in this case, the average wait time is nine times the average service time.

Most of the performance measures obtained from analytic models of queueing systems are averages, or mean values. While the measures are useful, averages can be misleading. For the M/M/1 model, the distribution function of the wait times is given by the following:

$$P\left(W_q > t\right) = \rho e^{-\mu(1-\rho)t}$$

The wait-time distribution, which is exponentially distributed, is plotted in Figure 2.8 for the example above (M/M/1 with λ=1.0, μ=1.11) that results in an average wait time, Wq, of 8.1 minutes. Note that for this system there is a 40% chance the wait time will be greater than the mean (8.1), 20% greater than 13.4 minutes, 3% greater than 30 minutes, 0.1% greater than one hour, etc. On the more positive side, 20% of the wait times will be below 1 minute, 50% below 5 minutes, 70% below 10 minutes, etc.

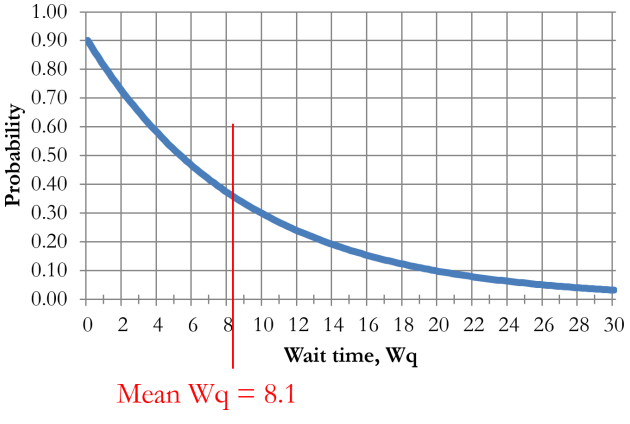
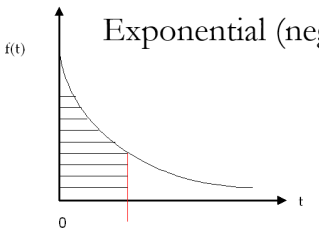


Figure 2.8: P(Wq > t) for M/M/1 with λ=1.0, μ=1.11.

As noted above, assuming exponentially distributed interarrival and service times are fundamental to using the formulas for determining system performance measures; therefore, we provide a quick review of the exponential distribution. A more detailed explanation is available in any basic statistics textbook. The density and distribution functions are



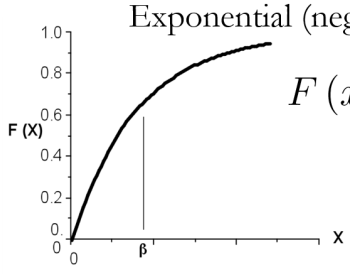
Exponential (negative) density function

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

Note (recall) for the exponential

$$\mu = \beta$$
$$\sigma = \beta$$

Exponential (negative) cumulative or distribution function



$$F(x) = P(X \le x) = \int_{l.l.}^{x} f(w)\,dw = \int_{0}^{x} \frac{1}{\beta} e^{\frac{-w}{\beta}} dw = -e^{\frac{-w}{\beta}} \Big|_{0}^{x} = 1 - e^{\frac{-x}{\beta}}$$

Figure 2.9: Characteristics of the exponential distribution.

shown in Figure 2.9 along with a plot of the functions. The exponential distribution is very variable. Its mean and standard deviation are equal resulting in a Cv = 1.0. Cv is a common standardized measure of variability and is the ratio of the standard deviation to the mean.

Take note of the shape of the density function. It is not symmetric like the normal distribution and is unbounded at the upper end, which means it is possible, though unlikely, to have very high values. The highest frequency of occurrences (interarrival or service times) is at the low end; in other words, most interarrival and service times will be small, but occasionally the system will experience very high values. For example, an exponential distribution with a mean of ten minutes, using the formula for the distribution function, has a 63.2% chance an individual time will be less than 10 minutes, 39.3% chance less than 5 minutes, and 9.5% chance less than one minute. Also, there is a 0.2% chance an individual time will be greater than 60 minutes. Many systems exhibit such behavior, but many do not; therefore, it is always vitally important to understand the system's actual distributions because, as will become evident later in the book, choice of the distributions can dramatically affect results and decisions. Also, unbounded distributions should be used cautiously in simulations. These issues are explored in detail in Chapter 8.

Figure 2.10 provides formulas for comparable performance measures for the case where the service-time distribution is not limited to the exponential. The general-case formulas are provided on the left side of the figure. The measures are a function of the variability of the service-time distribution, via the distribu-

$$\rho = \frac{\lambda}{\mu}$$

M/G/1	**M/D/1**
$L = \rho + \frac{\rho^2(1+\sigma^2\mu^2)}{2(1-\rho)}$	$L = \rho + \frac{\rho^2}{2(1-\rho)}$
$L_q = L - \rho$	$L_q = \frac{\rho^2}{2(1-\rho)}$
$W = \frac{L}{\lambda}$	$W = \frac{1}{\mu} + \frac{\rho}{2\mu(1-\rho)}$
$W_q = W - \frac{1}{\mu}$	$W_q = \frac{\rho}{2\mu(1-\rho)}$

Figure 2.10: Measures of performance for the M/G/1 and M/D/1 models.

tion's standard deviation. The formulas on the right side of the figure are for the extreme case where the standard deviation is zero (i.e., the service time is a constant). As with the M/M/1 model, these formulas are for a single-server system with Poisson-distributed arrival rates (exponentially distributed time between arrivals), infinite capacity queue, FIFO queue discipline, one type of customer, infinite calling population of customers, no balking, no reneging, and 100% server availability. Again, these measures are long-run steady-state averages (i.e., the current state of the system is independent of the initial state).

Formulas for comparable long-run steady-state average performance measures are provided in Figure 2.11 for the case where the number of servers is not limited to one. All other assumptions are the same as the M/M/1 model. Formulas on the left part of the figure are general, for any number of identical servers *s*; formulas on the right part of the figure are for the two-server case.

M/M/s	**M/M/2**
$P_0 = \left[\sum_{r=0}^{s-1} \frac{(\lambda/\mu)^r}{r!} + \frac{(\lambda/\mu)^s}{s!(1-\rho)}\right]^{-1}$	$P_0 = \left[1 + \lambda/\mu + \frac{(\lambda/\mu)^2}{2(1-\rho)}\right]^{-1}$
$L_q = \frac{\rho(\lambda/\mu)^s P_0}{s!(1-\rho)^2}$	$L_q = \frac{\rho(\lambda/\mu)^2 P_0}{2(1-\rho)^2}$
$W_q = \frac{(\lambda/\mu)^s P_0}{s!s\mu(1-\rho)^2}$	$W_q = \frac{(\lambda/\mu)^2 P_0}{4\mu(1-\rho)^2}$

Figure 2.11: Measures of performance for the M/M/s and M/M/2 models.

There is a general relationship between the average number in the system, L, and the average wait time, W, referred to as Little's Law [7, 8]. Under steady-state conditions, the average number of customers in a queueing system is equal to the average rate at which they arrive multiplied by the average time a customer spends in the system, (i.e., L = λW. Also, similarly Lq = λWq).

In their seminal book *Factory Physics* [4], Hopp and Spearman pose Little's Law as a basic law of manufacturing. The relationship is expressed thus: work in process (WIP); inventory is the product of throughput (TP) and cycle time CT). Throughput is the average quantity of goods and items produced per unit of time.

WIP = TP * CT [units = units/time * time]

This fundamental law describes the underlying behavior of production systems and defines a basic relationship among key performance measures. For example, if the average throughput is 10 units per day and the average cycle time is 2 days, then the WIP is 20 units.

The formulas presented in this section are just a small sampling of the many other cases that are described in books dedicated to, or with more emphasis on, queueing theory. These

formulas are presented to illustrate the differences between analytic models and simulation models and to provide a rough-cut approximation to more general systems.

It should be apparent that obtaining system performance measures with analytic models is fast and straightforward—all that is involved is populating formulas with a few system parameters. Of course, developing these formulas is difficult and requires high-level skills in mathematics, but using them is quite easy; however, there are two major shortcomings of analytic models: they (1) are limited by assumptions that are oftentimes unrealistic and (2) provide only long-run, steady-state average measures. Simulations, on the other hand, require significant time and expertise to develop but have far fewer restrictive assumptions and are not limited to steady-state measures.

Section 2-7 Analyzing the behavior of queueing systems

We can use the simple queueing system formulas to illustrate tradeoffs and performance assessments of queueing systems and to study system behavior under differing conditions.

Basic Economic Tradeoff

As indicated earlier, queueing systems involve a fundamental tradeoff between the cost of the customer waiting for service and the cost of providing the service.

This tradeoff is illustrated through an example of a simple parts facility, operated by Bob (the single server), in a manufacturing operation. The parts facility is being evaluated for possible improvements. The customers are maintenance workers that need parts in order to repair failed machines in the manufacturing area. Maintenance workers arrive at the parts facility at an average arrival rate (λ) of 15 per hour, and the rate is Poisson distributed. The service times, or parts fulfillment times, are exponentially distributed with a mean of three minutes ($1/\mu$). The system could alternatively be defined as one with an average time between arrivals of maintenance workers ($1/\lambda$) of four minutes and an average service/fulfillment rate (μ) of 20 customers per hour.

There are two proposed improvement options being considered:

A. Reducing Bob's average process time to 2.5 minutes at a cost of $2,000 per month.

B. Reducing the variability in Bob's process time. The mean time would be the same, three minutes, but the standard deviation would be reduced from 3 minutes (standard deviation equals the mean for the exponential distribution) to one minute. The service time is assumed to be normally distributed.

Downtime on the machines that the maintenance workers are repairing costs $15 per hour in lost profit. Since the facility operates one shift, five days per week, assume, considering holidays, it operates 160 hours per month, on the average.

Maintenance workers are served in the order in which they arrive (i.e., FIFO queue discipline), and maintenance workers do not balk or renege. The calling population is assumed infinite. (This might be an issue in this case, but assume that the maintenance workers are servicing a very large number of machines.)

The arrival process is assumed stationary, and Bob, the server, is always available. Bob himself will most certainly take breaks and not serve customers during that time, but if someone else takes his place while he is on break, then we can assume the server is always available. The evaluation only considers the impact of the parts facility on operations; in other words, machine breakdowns are not considered per se, only the maintenance workers' demand for parts based on machine failures. Given these assumptions, the M/M/1 model and formulas in Figure 2.7 can be used to analyze the Base Case and Option A. Similarly, the M/G/1 model and formulas in Figure 2.10 can be used to assess Option B.

For the Base Case, the average time in the system, W, is 12 minutes, or $W = 1/(\mu-\lambda) = 1/(20 - 15) = 0.2$ hour). This seems like a lot—12 minutes lead time for 3 minutes of service (9 minutes of wait time), on the average, especially when Bob's average utilization, ρ, is 75%. It is important to note that in this example, time in the system refers to time in the parts facility. The parts facility is just a subset of the overall machine failure and repair problem; the problem is scoped to only address the effect of the parts facility operation.

For Option A, the proposed change decreases the mean service time from 3.0 to 2.5 minutes; therefore, the service rate increases from 20 to 24 customers per hour. As a result, the average time in the system, W, is reduced to 6.67 minutes (2.5 minutes of service time + 4.17 minutes of waiting time). In other words, a 16.7% reduction in service time results in a 53.7% reduction in wait time. Since the average service time is less and the arrivals remain the same, the server utilization, ρ, will decrease to 62.5%. Note that even though Bob is only busy 62.5% of the time, the average wait time is still nearly double the average service time. This is the nature of highly-variable queueing systems.

Option A results in a savings of 0.089 hours per maintenance action [(12.0 – 6.67)/60]. That means for each maintenance action there would be 0.089 hours less downtime or 0.089 hours more production. At a profit of $15 per hour, Option A would generate $1.34 in additional profit per maintenance action. That is great, but the improvement will cost $2,000/month; is it worth it? There are 15 maintenance actions per hour on the average (λ), and if the facility operates 160 hours per month, that results in an average of 2,400 maintenance actions per month. Therefore, by implementing Option A, monthly profit would be expected to increase by $3,204. Therefore, the change would involve a net increase in profit of $1,204 per month; not bad, for a $2,000 per month investment.

But what about Option B? The suggested improvement in this proposal is expected to reduce the variability in the service process—the mean stays the same, but the standard deviation is reduced from three minutes to one. Applying the M/G/1 model and using the formulas provided in Figure 2.10, the average time in the system, W, is reduced from 12.0 minutes (0.20 hours) in the base case to 8 minutes (0.133 hours). At a profit of $15 per hour, the change would generate $1.01 in additional profit per maintenance action or $2,412 in additional profit per month. Therefore, no more than $2,412 should be spent to implement the change. Of course, the maximum cost will be less when the company's discount rate is considered.

The results of the example are summarized in Table 2.2.

	Option	Base Case	A	B
	Situation	Current	Speed up service time	Less variable service time
		Model		
Parameters	**Units**	**M/M/1**	**M/M/1**	**M/G/1**
λ	cust/hr	15	15	15
μ	cust/hr	20	24	20
ST = $1/\mu$	min/cust	3.0	2.5	3.0
Std. Dev. ST	min/cust	3.0	2.5	1.0
Measures				
ρ	% util.	75.0	62.5	75.0
W	minutes	12.0	6.7	8.0
Wq	minutes	9	4.2	5.0
Savings per month	$	--	3,204	2,412
Cost per month	$	--	2,000	unknown

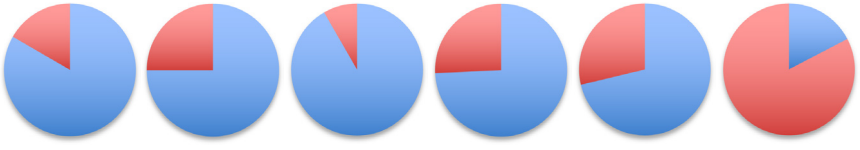
Table 2.2: Summary of the options and their expected performance for Bob's Repair Center.

System Performance Sensitivities

Again, we must consider the crucial tradeoff between the cost of customers waiting for service and the cost of providing the service. A simple example from the medical industry illustrates the sensitivity of a system's performance to its operating parameters.

The example considers a small medical testing facility in a local clinic. Request for tests arrive at an average rate five per hour and are completed at an average rate of six per hour. Assuming there is a single tester, rates are Poisson distributed (times are exponentially distributed), and the requests are processed in a first-in, first-out manner, then we can approximate the system with the M/M/1 model.

As shown in the first case in Table 2.3, and based on the M/M/1 formulas in Figure 2.7, the tester is busy 83.3% of the time and requests wait 50 minutes on the average after arrival for the test to begin. The ratio of wait time to process time is 5:1.

		Model					
Parameters	**units**	**M/M/1**	**M/M/1**	**M/M/1**	**M/G/1**	**M/D/1**	**M/M/2**
λ	cust/hr	5	5	5.5	5	5	5
μ	cust/hr	6	6.67	6	6 *	6 **	6
ST = 1/μ	min/cust	10	9	10	10	10	10
ρ	% util.	83.3	75.0	91.7	83.3	83.3	41.7
Measures							
Lq	cust.	4.2	2.3	10.1	2.4	2.1	0.2
Wq	minutes	50	27	110	28.9	24.8	2.1
Wq/ST	---	5.0	3.0	11.0	2.9	2.5	0.2
Wait Time Process Time							

* Std. dev. = 4 versus 10 for exponential service time

** Std. dev. = 0 versus 10 for exponential service time

Table 2.3: Performance measures for alternative simple queueing systems.

Variations in the system are shown in the remaining columns of Table 2.3. In the second case when the average service rate is increased by about ten percent, the average wait time is cut nearly in half (27 minutes), the server utilization drops to 75%, and the ratio of wait time to process time drops to 3:1 when compared to the base case. Conversely, when the arrival rate is increased by ten percent, as shown in column three, the average wait time more than doubles (110 minutes), the server utilization increases to nearly 92%, and the ratio of wait time to process time more than doubles to 11:1 when compared to the base case. We can see that system performance is heavily dependent upon the average arrival and service rates and the server utilization (ρ). In both of these cases the M/M/1 formulas are used.

The fourth and fifth columns in Table 2.3 show the sensitivity to variability in the service process using the M/G/1 and M/D/1 models. The coefficient of variability Cv is the ratio of the distribution's standard deviation to its mean. For the service-time distribution in the base case, the Cv is 1.0 (for the exponential distribution the mean and standard deviation

are equal). If the Cv is reduced to 0.4, then average wait time drops to 28.9 minutes, compared to 50.0 for the base case. Note, for the M/G/1 case, the average service time is the same as the base case, but its variability is greatly reduced. In the M/D/1 case, there is no variability in the service time; it is a constant 10 minutes. As a result, average wait time drops to 24.8 minutes, about one half of the base case. In both the M/G/1 and M/M/1 cases the expected server utilization is the same as the base case, 83.3%; therefore, system performance is heavily dependent upon the variability in the service, and arrival, processes. As a result, the specification of the arrival and service processes are critical to having useful models, whether analytic or simulation. How to determine the distributions of the interarrival and service times is discussed later in the book.

The last case in Table 2.3 is for the two-server case (i.e., a second lab tester is used and customers can be served by either of the two servers). In this case, average wait time is reduced considerably, to 2.1 minutes (versus 50 minutes for the single server), but the expected server utilization is halved, from 83.3% to 41.7%. The ratio of wait time to process time also drops considerably to 0.2:1, compared to 5:1 for the base case.

More System Parameter Sensitivities

The discussion of the effect of assumptions and system parameters on system performance is continued in this section, but in a bit more detail. The M/G/1 model is used with an average arrival rate of 1.0 customer per minute; arrival rate is Poisson distributed. (Conversely, the average time between arrivals is exponentially distributed with a mean of one minute per customer.)

The service-time distribution is varied, both in terms of average service rate and variability in the service time. The average service rate varies from 1.01 to 2.0 customers per minute; (i.e., the service times vary between 0.99 and 0.50 minutes). This results in the server utilization being between 0.99 and 0.50. At the same time the Coefficient of Variable (Cv) of the service-time distribution varies from 0.00 to 1.0; the cases range from the M/D/1 model with deterministic or constant service times to the M/M/1 model with exponentially distributed service times.

Figure 2.12 illustrates the joint effect of server utilization and service-time distribution variability on average wait time. Over the ranges defined above, average wait time ranges between 0.3 minutes for the (Cv=0.0, ρ=0.50) case and 98.0 minutes for the (Cv=1.0, ρ=0.99) case. Obviously, the system parameters greatly affect their performance.

Notice in Figure 2.12 that there is a wide area where the surface is rather flat, or where the average waiting time is rather low. This occurs when server utilization and variability are low. Of course, the value is relative—even when the server utilization is only 80% (average service time is 0.8 minutes), the average wait time ranges from 1.6 to 3.2 minutes, or the average wait time is double or quadruple the average service time, for a Cv of service time from 0.0 to 1.0, respectively. Even when server utilization is 50% (average service time is 0.5 minutes), average wait time is between 0.3 and 0.5 minutes for Cv between 0.0 and 1.0, respectively.

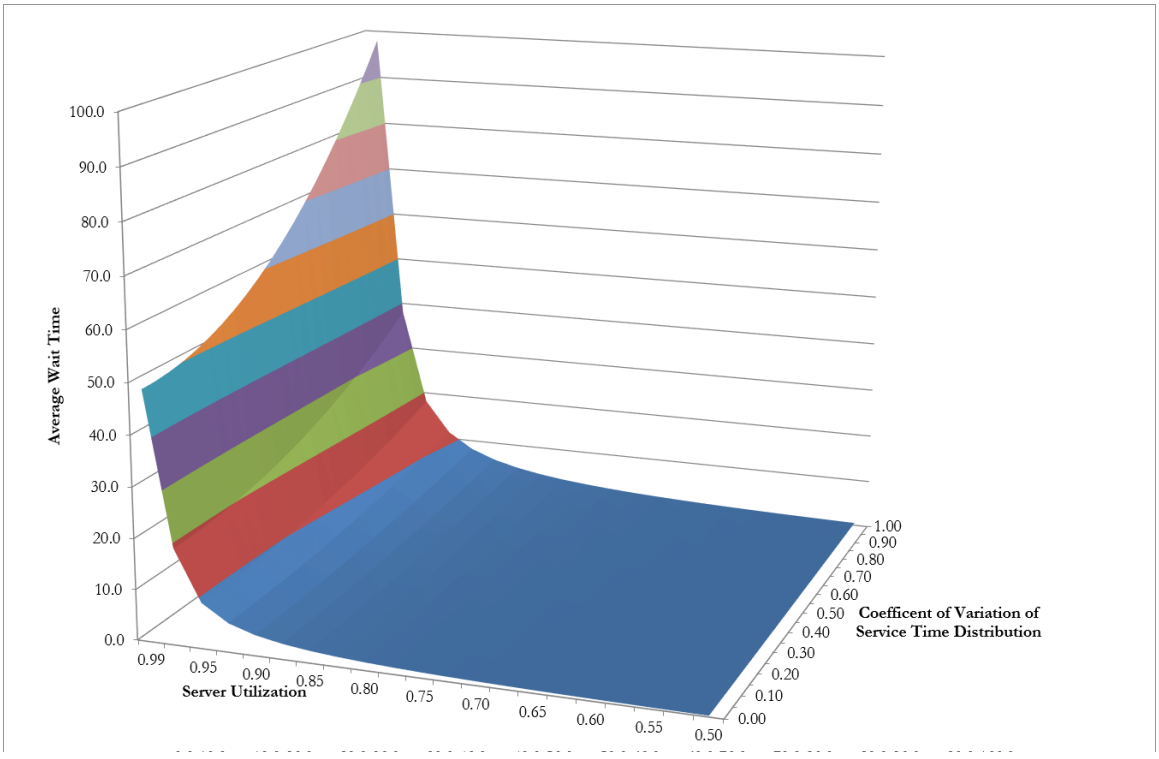


Figure 2.12: Joint effect of server utilization ($0.50 \le \rho \le 0.99$) and service-time variability ($0.0 \le Cv \le 1.0$) on average wait time (Wq) using the M/G/1 model.

Similarly, notice in Figure 2.12 that there is an area where the surface is very steep (i.e., the average waiting time is very high). This occurs when server utilization and variability are high. When the server utilization is 95% (average service time is 0.95 minutes), the average wait time ranges from 9.0 to 18.1 minutes, or the average wait time is nearly 10 to 20 times the service time, for a Cv of service time from 0.0 to 1.0, respectively. When the server utilization is 99% (average service time is 0.99 minutes), the average wait time is 98.0 minutes, nearly 100 times the service time for the Cv = 1.0 case.

Figure 2.13 takes a more detailed look at average wait times for an M/G/1 system via a 3D plot (left) and a contour plot (right). In this case, server utilizations only range between 80% and 95%. Again, the joint effect of server utilization and the variability in the service times (characterized by the Cv of the service-time distribution) significantly affect the performance of the queueing system.

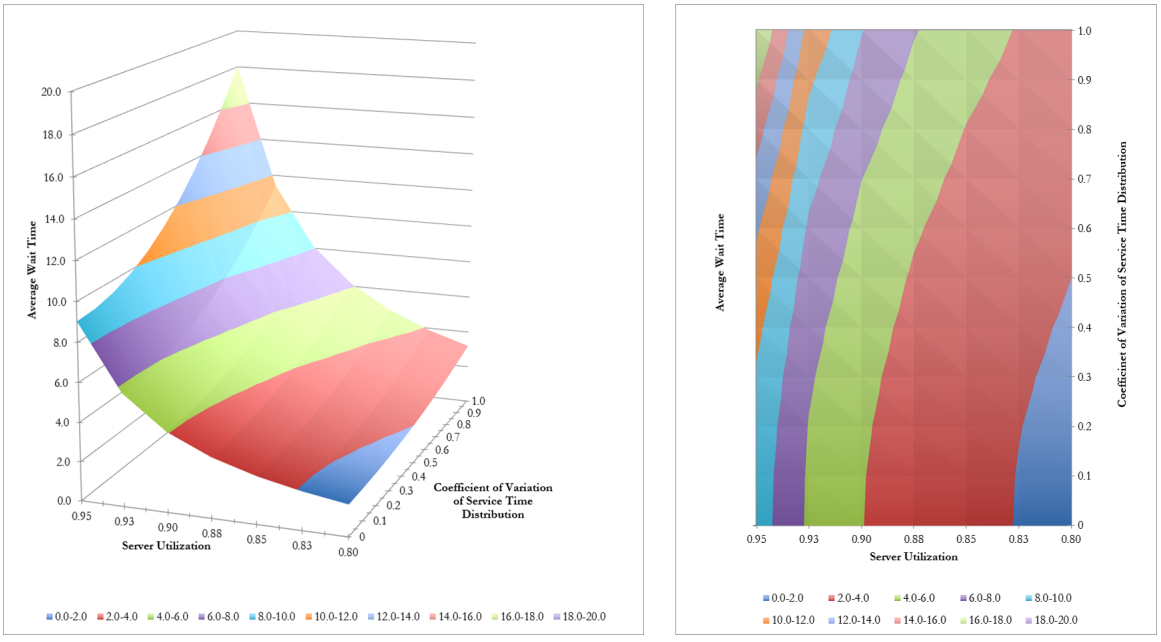


Figure 2.13: Joint effect of server utilization ($0.80 \le \rho \le 0.95$) and service-time variability ($0.0 \le Cv \le 1.0$) on average wait time (Wq) using the M/G/1 model.

The sensitivities of performance to system parameters are also illustrated through the marginal plots shown in Figure 2.14 and Figure 2.15. The marginal plot in Figure 2.14 shows the effect of server utilization on average wait time for the high-variability case (Cv = 1.0, M/M/1 model). Figure 2.15 shows the effect of variability (Cv of service-time distribution) on average wait time when the server utilization is 90% (ρ = 0.90, M/G/1 model).

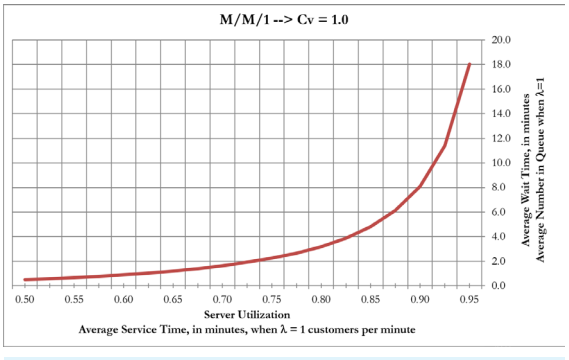


Figure 2.14: The effect of server utilization on average wait time (M/M/1 model, Cv = 1.0).

As discussed earlier, queueing systems involve a tradeoff of customer and server wait times. Wq explicitly measures the wait time for the customer, and the server utilization, ρ, implicitly measures the wait time of the server. For example, from the plot in Figure 2.14 (M/M/1 case), a server utilization of 50% means that on the average, the server spends half of its time waiting for customers (50% idle time) and customers spend half of their time in the system waiting (Wq = 0.5 minutes when service time $1/\mu$ = 0.5 minutes). Conversely, a server utilization of 90% means the server spends 10% of its

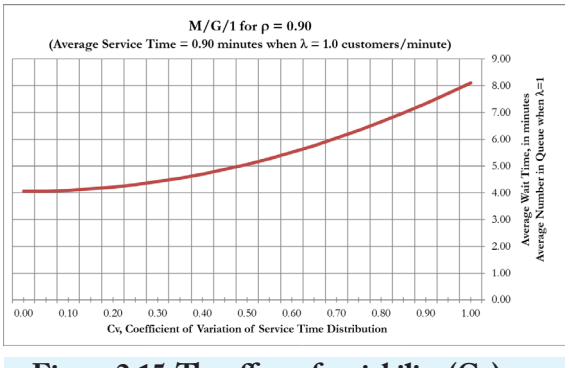


Figure 2.15: The effect of variability (Cv) on average wait time (M/G/1 model, ρ = 0.90).

time waiting for customers (10% idle time) and customers spend 90% of their time in the system waiting for service (Wq = 8.1 minutes when service time $1/\mu$ = 0.9 minutes).

Section 2-8 Simulating queueing systems

Queueing theory formulas provide good insight into the behavior of simple systems; however, as mentioned earlier, they are oftentimes limited by assumptions and their inability to handle complex interactions among queueing-system components. In those cases, simulation is necessary to understand and analyze the system's behavior and estimate its performance.

The basic difference between the simulation and queueing theory approaches is that simulation is logic based and not formula based. The logic is as detailed and complex as is needed to adequately represent the system to answer the questions being posed. This is a fundamental premise of modeling—models should be as simple as they can be in order to address the questions being considered. That is why it is so critical to articulate the objective of the analysis as clearly and as early as possible. Overmodeling, building a model more complex than it needs to be, not only wastes time and resources, but can make the model unstable, hard to document, and difficult to maintain. These issues are discussed further in Chapters 15 and 16.

In this section, we briefly introduce how queueing systems are modeled via simulation. A more detailed discussion of how simulations work is provided in Chapter 8. This section defines the notions of events and states and describes how states are used to characterize system performance. Two examples are used to illustrate how simulation provides insight into system behavior. The first example considers the effect of reneging and balking on performance of a simple queueing system, both of which are assumed not to occur in basic queueing theory models. The second example considers the effect of linked processes and the effect of downtime on queueing-system behavior. The simulation models are discussed only in terms of basic queueing-system behavior and the effect of relaxing constraints and assumptions. How the simulation models are developed and operate are not considered here; these issues are discussed in detail throughout the book.

Discrete-Event Simulation

There are many types of simulations, but the focus of much of this book is on *discrete-event simulation*. As the name indicates discrete-event simulation is used to represent systems in which states change at discrete points in time as a result of events that occur.

Queueing systems are examples of discrete-event systems. Customer arrivals are one type of event that causes states to change in a system; for example, a customer arriving in the system can result in a server being made busy or an increase in the number of customers waiting in the queue. Customer arrivals occur at discrete points in time, as you can see illustrated for the server-status state in Figure 2.16. In this case the server is idle at the start of the simulation (state value is 0 for idle) until about time 1.5 when a customer arrives. The server is busy (state changes to a value of 1 for busy) during the service activity, which has

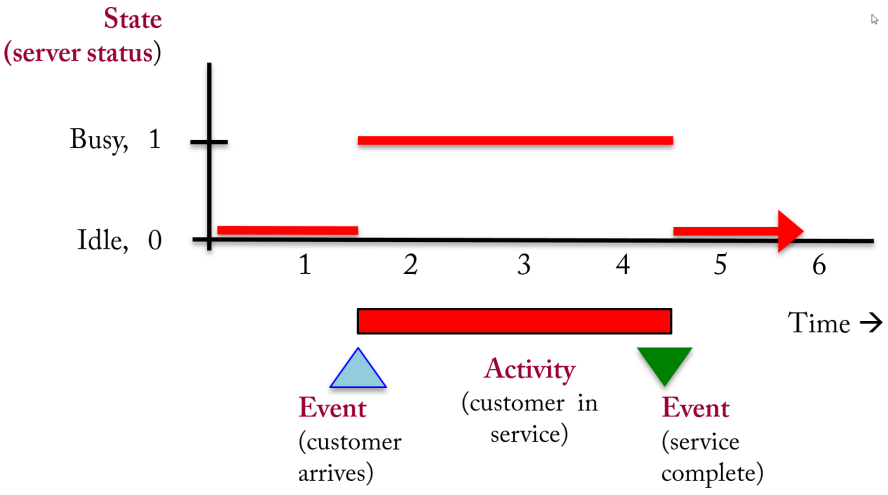


Figure 2.16: State of the system changes when events occur.

a duration of three minutes. The server becomes idle again (state change to 0) at time 4.5 if no customers arrive during the service activity. In this simple case, over the duration of 6 minutes, the server utilization is 50%, calculated as follows: [0(1.5 + 1.5) + 1(3)] ÷ 6.0 = 0.50 or 50%.

The server utilization is an example of a *time-weighted average* performance measure; that is, it is a weighted average of state values. In this case the server-status state can have only two values 0 (idle) and 1 (busy). The server spent 3.0 minutes idle (state=0) over the 6.0-minute simulation, 1.5 at the beginning and 1.5 at the end. The only busy time is during the one 3.0-minute service activity.

Another queueing-system example is provided in Figure 2.17. In this case, the performance measure is the average number in queue. The state is the number in queue, where the minimum value is 0 and there is theoretically no maximum. A set of randomly generated times between customer arrivals is listed on the left part of the figure. The interarrival times are used for the customer arrival event times, which are denoted on the time axis as upward pointing triangles. A set of randomly generated service times is listed on the right part of the figure. Chapter 8 discusses how the interarrival and service times are randomly generated. The first customer arrives at time 0 and has a service time of 6 minutes. A departure event occurs when a customer completes service; therefore, the first departure event occurs at time 6 when the first customer completes service. As will be illustrated in the next figure, departure events depend on the interaction of the status of the server and the service time—thus, they depend on the system dynamics and logic and cannot be determined in advance.

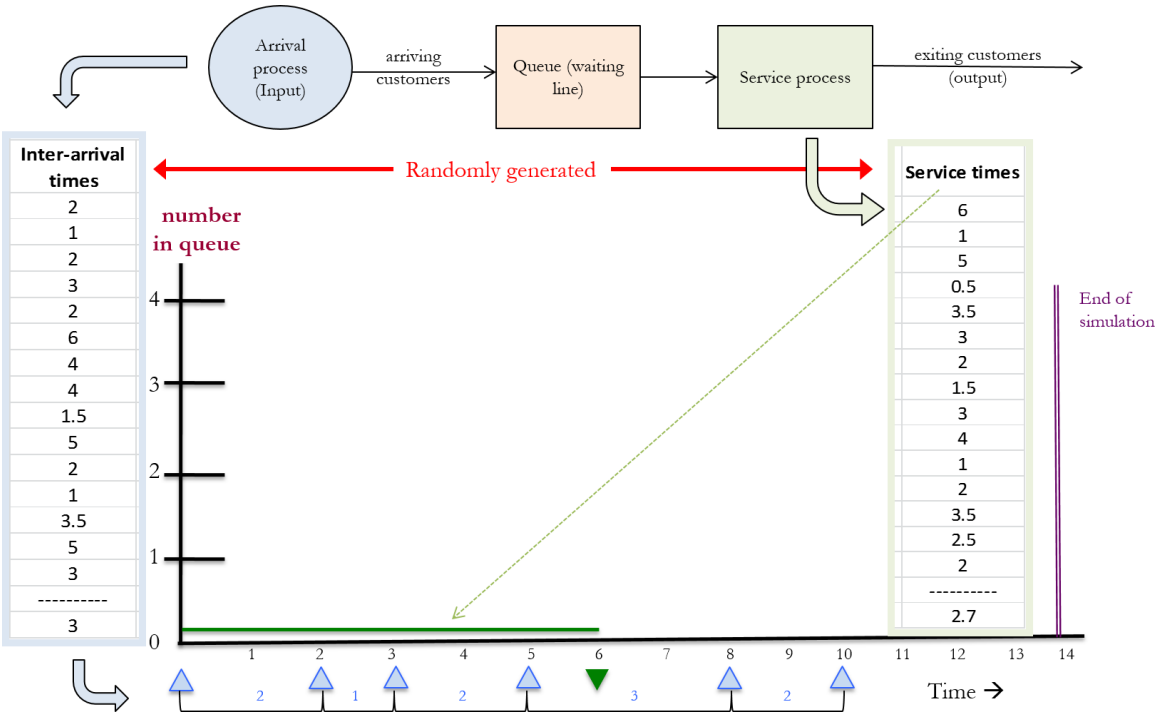


Figure 2.17: Simulation of queuing system based on occurrence of random events.

Figure 2.18 provides a manual simulation of a simple single-server system using the random interarrival and service times from Figure 2.17. The details of how the simulation works are explained in Chapter 8, but the basic logic is like this: if a customer arrives and the server is busy, the customer joins the queue and moves toward service over time in a FIFO manner. For example, Customer 2 arrives at time 2, assumes the first position in the queue since the server is busy with the first customer, and enters service at time 6 when the first customer departs; therefore, the wait time for Customer 2 is 4. (The wait time for

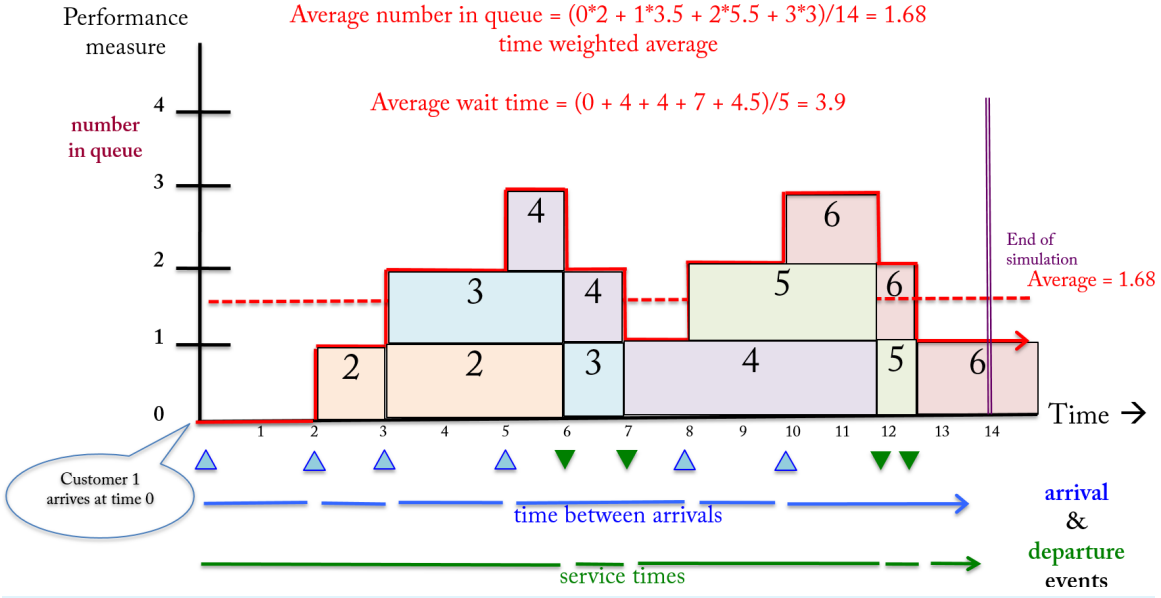


Figure 2.18: Example manual discrete-event simulation.

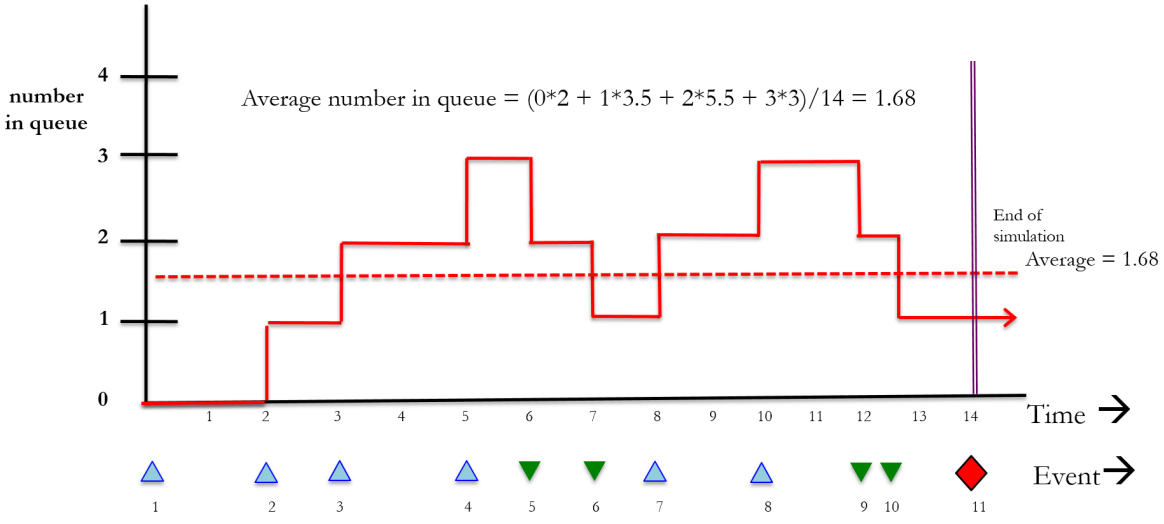
Customer 1 is 0 since the customer arrives at the beginning of the simulation when the server is idle.) Similarly, Customer 3 arrives at time 3, assumes the second position in the queue since Customer 1 is in the first position, moves to the first position at time 6 when Customer 1 enters service, and enters service at time 7 when the second customer departs; therefore, the wait time for Customer 3 is 4. This logic continues for the duration of the simulation. The performance measure average waiting time is calculated as an *observation average*—the individual wait times are summed and the sum is divided by the total number of customers entering service.

The simulation in Figure 2.18 is carried out for just 14 time units, but it still illustrates the logic involved in a simulation of a simple queueing system and how random events drive system performance. In addition to customer arrival and departure events, the simulation includes an end-of-simulation event that stops the simulation. In terms of system performance, in this case, the server is busy all of the 14 simulated minutes (100% utilization). The queue size varies from zero to three customers and the time-weighted average number in queue is 1.68. The wait times for the five customers that entered service range between zero and seven and the observational average wait time is 3.9 minutes. The performance measure calculations are shown at the top of Figure 2.18.

Of course the simulated time considered in Figure 2.18 is much too short to obtain meaningful results, but that is not its intent here—this example is only used to illustrate the process of performing a discrete-event simulation and to show how performance measures are determined using simulation. It also illustrates the difference between using queueing theory formulas to estimate system performance and using simulation. As is evident from this small example, simulations are much too tedious to be performed manually; therefore, simulations are performed by computers. Specialized simulation software, such as *FlexSim*, is imperative for developing and executing simulation models of even simple systems and for analyzing the results obtained from the simulations.

As discussed above, the average number in queue, L_q (using queueing theory notation) is calculated as a time-weighted average. Typically, and as illustrated in Figure 2.18, the time-weighted average is the sum of the products of the state value (e.g., 0, 1, 2, …) and the total time the system was in that state during the simulation; however, in simulation the average is obtained in a different manner that is more efficient from a computing standpoint. The result is the same. The alternative approach is defined below; it basically requires maintaining a running total (weighted sum) and not the entire history.

The number in the queue awaiting service at the any time T is denoted as $N_q(T)$, however, in event-driven discrete-event simulation models, it is more convenient to track $N_q(J)$, the number in queue at event J instead of $N_q(T)$. Event J occurs when the Jth customer is available to the server for processing. The average number in queue, L_q, is calculated as the time-weighted average number in queue up to simulated time T (or event J which occurs at time T). It is the area under the number-in-queue versus time curve, or as shown Figure 2.18 and more generally in Figure 2.19. It is the weighted average of the number in queue between two subsequent events (J-1 and J), $N_q(J\text{-}1)$, with the time between subsequent events, $\Delta t(J)$, as the weight. The relationship is defined as



Lq = average number in queue at time T or event J(T) = [TotNq(J-1) +Nq(J-1)×Δt(J)] ÷ t(J)
Lq at time 14 or event J(14)=11:
[TotNq(10) +Nq(10)×Δt(11)] ÷ t(11) = [22 + 1×1.5] ÷ 14 = 1.68

Figure 2.19: Alternative approach for calculating time-weighted measure average number in queue.

$$\widehat{L_q} = \overline{N_q(J)} = \frac{\sum_{j=1}^{J} N_q(j-1) * [t(j) - t(j-1)]}{t(J)} = \frac{TotN_q(J-1) + N_q(J-1}{t(J)}$$

where

$$TotN_q(J-1) = \sum_{j=1}^{J-2} N_q(j-1) * [t(j) - t(j-1)]$$

$$\Delta t(J) = t(J) - t(J-1)$$

Thus, as shown in the equation above, the simulation only needs to maintain the running total, $TotN_q(J-1)$, and the time of the last event, $t(J-1)$. The total is updated at each event J based on $N_q(J)$ and $t(J)$.

Balking and Reneging

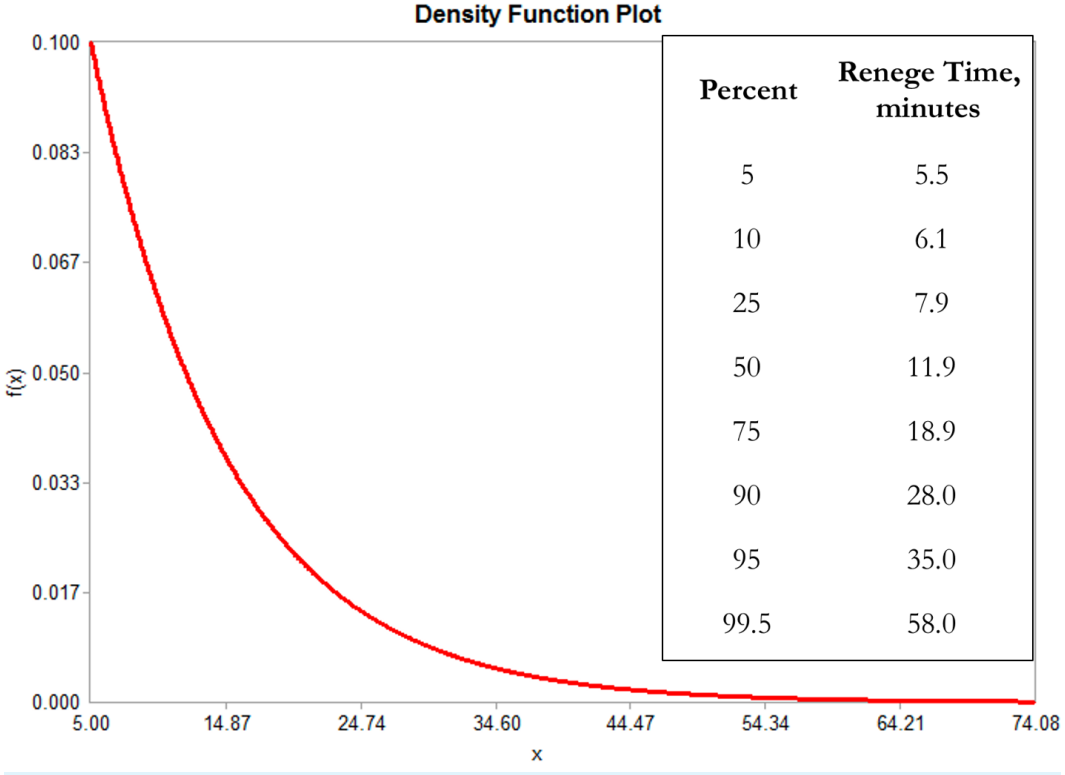
As indicated earlier, customers may balk or renege in queueing systems. Recall, customers balk if when they arrive to a system they immediately leave without joining the queue based on their evaluation that the wait time is too long; customers renege when after joining a waiting line, they leave before receiving service due to length of the wait.

The effect of balking and reneging is illustrated through the simulation of a small fast-food restaurant. There is only a single server. Customers arrive at an average rate of six per hour and are served at an average rate of 6.67 per hour. Alternatively, the average time between customer arrivals is 10 minutes and the average service time is nine minutes, both exponentially distributed. Customers are processed in a first-in, first-out manner.

Balking and reneging are personal decisions and context sensitive. To model the reneging behavior, the following distribution is used to determine each customer's balking value; for example, there is a 10% chance the arriving customer will balk if there are as few as two customers in the queue (alternatively, two or more customers in the queue when the customer arrives), a 10% chance the arriving customer will balk if there are as few as three customers in the queue, etc. No customers balk if there are zero or one customer in the queue.

Percent	Balk Value
10.00	2.00
10.00	3.00
15.00	4.00
30.00	5.00
15.00	6.00
10.00	7.00
10.00	8.00

To model the reneging behavior, the renege time is assumed to be exponentially distributed with a mean, R_{mean}, of 15 minutes, but with a minimum of 5 minutes. In other words, everyone will wait 5 minutes, and then the time after 5 minutes is exponentially distributed with a mean of 10 minutes. This is illustrated in Figure 2.20. Notice that five



Percent	Renege Time, minutes
5	5.5
10	6.1
25	7.9
50	11.9
75	18.9
90	28.0
95	35.0
99.5	58.0

Figure 2.20: Distribution of renege time for (R_{min}, R_{spread}) = (5, 10).

percent of the customers renege after being in the line 5.5 minutes, 50% after 11.9 minutes, etc. In general the renege time can be expressed as exponential (R_{min}, R_{spread}) where R_{min} = minimum balk time and $R_{spread} = (R_{mean} - R_{min})$. In the case below, balk time is exponential (5, 10).

Of course in a simulation model, the parameters and distributions are not restricted and are easy to change; this is a major advantage of simulation.

A screenshot of the simulation model is shown in Figure 2.21. In this case, the parameters are set to include balking and reneging. The results displayed in the dashboard are for 6.5 hours of simulated time; also, coincidently, 6.5 hours is the time of the arrival of the 50th customer. As the dashboard shows, through 6.5 hours, four percent of the customers balked (two out of the 50) and 24 percent have reneged (12 out of the 50 arrivals). Those customers that balked and reneged are shown in the simulation model toward the bottom of screen. The server has been busy 76.9% of the time. The State Gantt chart shows the distribution of idle and busy times over the 6.5-hour simulation. On the average, customers waited 5.0 minutes; this excludes those who balked and reneged. The average number in queue is 0.6 and, as shown by time plot, the queue length varies between zero and four.

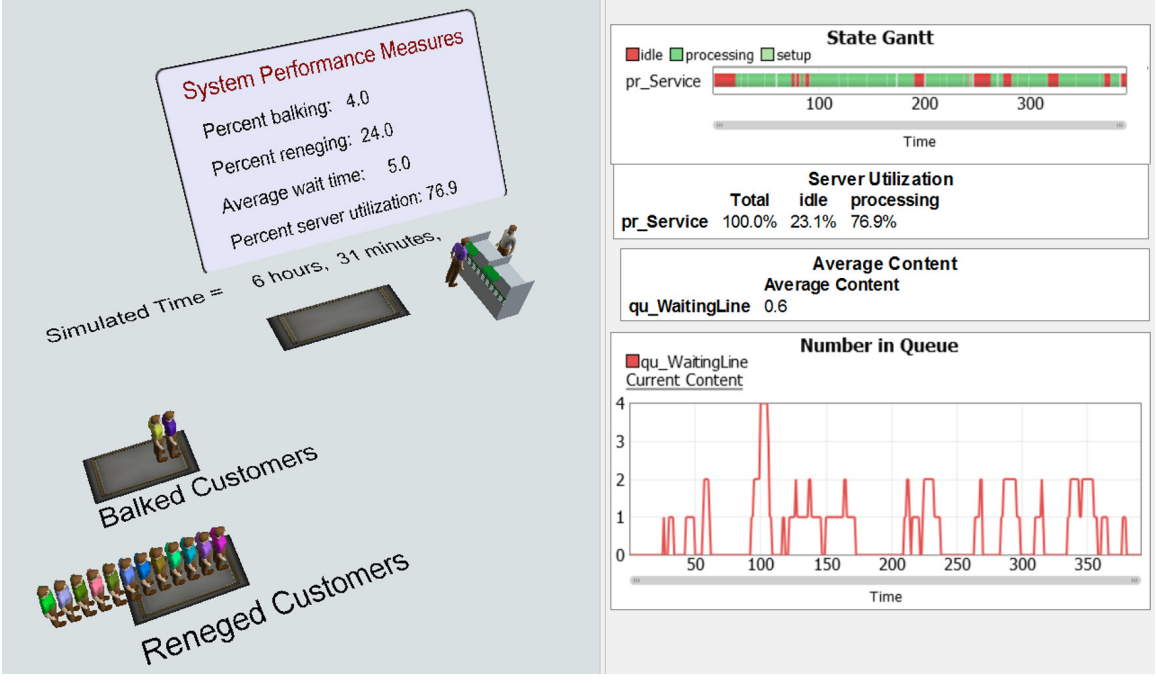


Figure 2.21: Simulation model to assess the impact of balking and reneging.

If no balking and reneging are considered, then the M/M/1 queueing theory model could be used. The results are shown in the first column of Table 2.4. It shows the average wait time is over an hour (63.3 minutes) and the server is busy nearly 85% of the time. It is doubtful customers would wait this long—many would leave because of the wait time (renege) or not even get in the queue because of the perceived high wait time (balk).

	Scenarios/Cases						
	1 MM1	2 Balk only	3 B+R (5,10)	4 B+R (10,10)	5 B+R (10,20)	6 B+R (20,20)	7 B+R (30,60)
Balking	No	Yes	Yes	Yes	Yes	Yes	Yes
Reneging	No	No	Yes	Yes	Yes	Yes	Yes
Renege minimum, minutes	---	---	5	10	10	20	30
Renege scale, minutes	---	---	10	10	20	20	60
Avg. Percent Balk	0.0	12.1	2.2	3.4	4.6	6.7	9.7
Avg. Percent Renege	0.0	0.0	25.6	20.2	16.9	10.6	3.6
Avg. Wait Time, minutes	63.3	18.5	5.4	7.3	9.1	11.9	15.9
Avg. Number in Queue	6.5	1.6	0.5	0.7	0.9	1.1	1.4
Server Utilization, percent	84.5	77.0	63.3	67.0	68.9	72.6	76.0

Based on 25 replications of 40 hours each.

Interarrival and service times are exponentially distributed with means 10 and 9 minutes, respectively

Table 2.4. Effect on system performance for various balking and reneging scenarios.

Scenario 2 in Table 2.4 shows the effect of balking only. In this system, it is estimated that about 12% of the customers balk; this means 12% of the customers would not purchase food. The reduced number of customers results in a lower average wait time (18.5 minutes) and lower server utilization (63.3%).

Scenario 3 in Table 2.4 includes balking and reneging. The reneging is as defined above—minimum time before reneging is five minutes and the average time to renege is 15 minutes. The simulated results in the screen shot Figure 2.21 differ from Scenario/Case 3 in Table 2.4 because the information in the table is based on a longer simulated time. Later in the book we will discuss how to determine how long to run simulation models. In this case only 2.2% balk, but 25.6% renege. The average wait time drops to 5.4 minutes and the server is busy only 63.3% of the time. While the wait time is greatly reduced, so is the number of customers served.

Scenario 4 through 7 in Table 2.4 show the effect that different values of the reneging parameters (R_{min}, R_{spread}) have on performance. You can see that system performance varies greatly depending on the reneging parameters.

This example again demonstrates the impact of a system's parameters on estimated performance. An essential part of modeling is ensuring that the parameters used in the model, and the assumptions underlying the model, best represent the system being analyzed.

Downtime

As mentioned earlier, most analytical models assume servers are available 100% of the time, which in most cases is not realistic; however, as will be discussed later in the book, it is not difficult to consider downtime in simulation models.

Typically, there are two types of downtime: planned and unplanned. Examples of planned downtime are operator breaks and equipment resupply (e.g., replenishing paper in a copy machine). Unplanned downtimes are expected, but when they occur is unknown (e.g., paper jams in a copy machine).

To illustrate the effect of downtime, we return to the model used in Figure 2.2—the one that was used to illustrate overall lead time and the contributions of three sequential process steps to overall lead time.

The underlying system is defined as follows: the mean time between arrivals of items to be produced is exponentially distributed with a mean of 10 minutes; each item goes through three sequential process steps with buffer space prior to the first workstation and between each workstation; each item varies in size; an item's length is uniformly distributed between 0.25 and 2.75 feet (expected or average length is 1.5 feet); an item's width is uniformly distributed between 0.5 and 4.5 feet (expected or average width is 2.5 feet); process times depend on the size of the item, in this case its area; therefore the process time for each process step, in minutes, is

$$\begin{aligned} \text{ProcessTime} &= 2.0 \times \text{Length} \times \text{Width} + \text{exponential}(2.5) \\ &= 2.0 \times \text{uniform}(0.25, 2.75) \times \text{uniform}(0.5, 4.5) + \text{exponential}(2.5) \end{aligned}$$

The average processing time of each of the three process steps is 10 minutes. There is, however, considerable variability. For example, based on the process time formula above, the expected processing time of a half-foot square item is 3.0 minutes; the expected processing time of an eight square foot item (two-feet long by four-feet wide) is 18.5 minutes.

In terms of planned downtime, each workstation and the upstream department that provides the material to be processed, take two 10-minute breaks per shift and a 30-minute lunch break. The breaks occur at about two hours and six hours into the shift (±10 minutes); the lunch break occurs about four hours into the shift (±10 minutes).

Unplanned downtimes occur at each workstation and in the upstream department. The times between downtimes are exponentially distributed with a mean of three hours; the downtime is uniformly distributed between 5 and 15 minutes.

A screenshot of the simulation model is provided in Figure 2.22. The dashboards in the screen show the system performance after five 8-hour days of simulation. The time plot on the left shows the large variability in the number of items in the system (work in process). The pie chart on the right shows the composition of the first server's time over the five days, including idle, processing, breakdown (unplanned downtime), and scheduled down (planned downtime for personal and lunch breaks). The average lead time (time through the three process steps) over the five days is 87.8 minutes; the minimum and maximum lead times are 13.4 and 168.3 minutes, respectively.

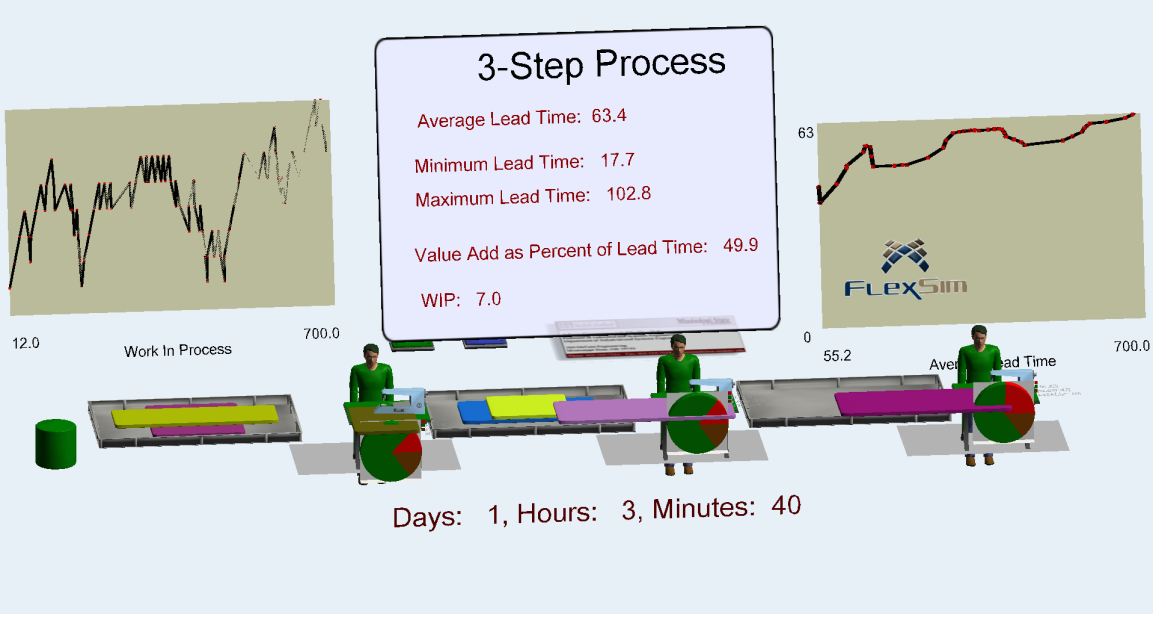


Figure 2.22: Screenshot of a model of a three-step production system.

Table 2.5 provides some of the data obtained the simulation model, namely the average time in the system. According to the data, the no-variability case would have constant arrival and service rates, and thus there would be no waiting; the lead time would simply be the sum of the three 10-minute processing times. Adding variability, as expected, increases the average lead time as well as the variability in the lead time.

Scenario	System Characteristic	Average Lead Time	Std. Dev. of Lead Time
1	No Variability	30.0	---
2	Variability in IAT & PT	58.3	5.4
3	Scenario 2 + Breaks	74.8	6.1
4	Scenario 3 + Downtime	105.9	17.8

Table 2.5: Lead time for various scenarios of a three-process production system.

The values provided in Table 2.5 are based on ten replications of 40 hours each. A replication is one run of the simulation model. As discussed in Chapter 9, simulation models must be executed multiple times to account for the variability from one simulation run to another. This is the same as would be done in any statistical analysis—we would not use one data point for our estimate, especially when there is a lot of variability in the data. In this case, using one 40-hour run might represent an extreme case, a very good or very bad week; we do not want to make decisions on extremes, so we average multiple results. Determining how many simulation runs to make is discussed in Chapter 9.

The variation that can occur from simulation run to simulation run is illustrated in Figure 2.23. The time plots at the top of the figure signify the number of customers in the buffer prior to the second server in the sequence over the duration of the simulation; each plot shows the behavior of the system during two different forty-hour weeks. The Gantt charts at the bottom of the figure also the represent the behavior of the system during two different forty-hour weeks, but for the items (customers) that are in the system 1000 minutes into the simulation. Obviously the lead times are quite different between the two runs of the model. We can see from this example how dangerous it would be to base a decision on only one run. This figure also illustrates the types of information that can be easily gleaned from simulation models.

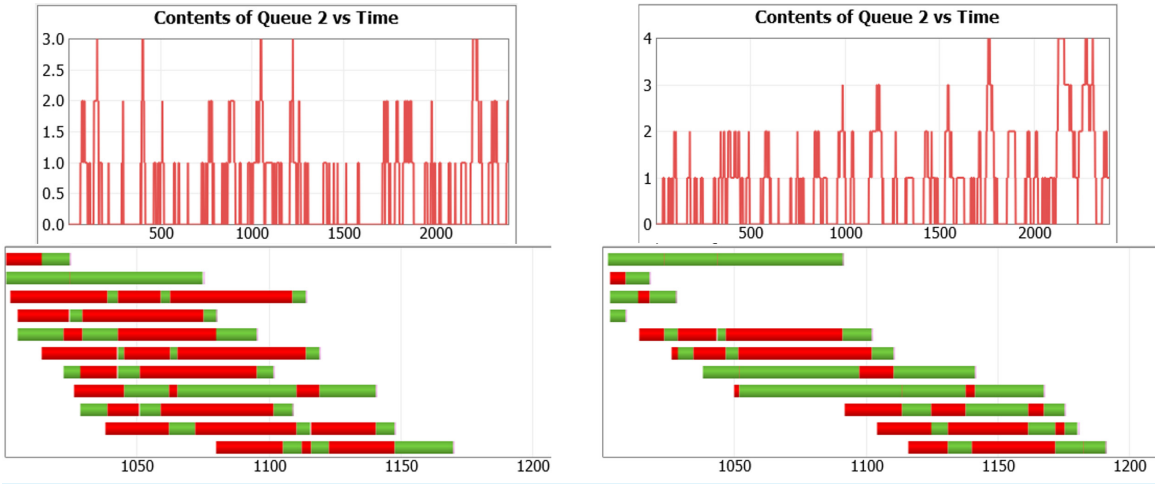


Figure 2.23: Varying system behavior across model replications.

Section 2-9 Summary

Queueing systems are everywhere in our personal and professional lives. They are key building blocks of almost all business systems. Understanding how queueing systems function and behave is critical to solving problems and making decisions; therefore, queueing systems are an important topic for study and analysis.

Queueing systems are composed of three primary elements: customers, servers, and queues. In every modeling effort, it is important to consider the system in terms of the characteristics of these basic elements. The characteristics are defined in detail in the chapter and are summarized in Table 2.6. A model may assume away some of the characteristics, but it should be done knowingly and the assumptions should be documented. A good practice is to invoke many assumptions initially so the modeling effort is not overwhelming, and then relax the assumptions in steps. Of course, the evolving model should be tested and validated at each step.

The characteristics of customers, servers, and queues all affect the overall performance of a system. There are many measures of system performance for queueing systems; the main ones deal with the number of customers in a queue, wait time of customers, and the utilization of servers. A common measure is customer lead time, which is the time span

between the time a request for service is made and the time it is fulfilled. Lead time may span multiple service process steps and is composed of two main elements: process or service times and wait times at each step. All queueing systems involve a fundamental tradeoff between the cost of customers waiting for service and the cost of providing the service. A portion of the service cost is waiting as well, where the server is waiting for customers. This is often reflected through the server's utilization or its complement, idleness.

The interaction of the characteristics of customers, servers, and queues generate the dynamics of queueing systems. Understanding, capturing, and analyzing the system dynamics are key to decision making. Ignoring these effects typically leads to bad decisions with unexpected results. As a result, queueing system models, especially simulation models, become valuable tools for problem solving and decision making.

Studying and changing a real system is typically too disruptive and expensive; therefore, models—representations or abstractions of a real system—are used for studying and analyses. With queueing systems there are two basic modeling approaches: analytic and simulation. For simple systems and under specific assumptions, the analytic approach (using queueing theory) works well; however, oftentimes the assumptions you have to make in order to use the performance measure formulas from queueing theory are too restrictive, and simulation must be used. It takes time to build, validate, and analyze simulation models, and special software is required, but simulations are only limited by the assumptions needed to scope the problem and make the modeling effort feasible. Simulation models are not limited to specific probability distributions or queue disciplines; servers do not always have to be available; customer arrivals can be scheduled, among other things. Both approaches are discussed in this chapter so the reader can understand the benefits and limitations of each; however, the remainder of the book focuses on the simulation approach.

- Customer
  - Arrival Process
    - Scheduled or interarrival times
    - Stationarity
  - Type
  - Queue behavior
    - Balking
    - Reneging
    - Jockeying
  - Selection of server
  - Calling population
- Server
  - Service/process time
  - Process steps
  - Availability/Downtime
    - Planned
    - Unplanned
  - Number
  - Resource needs
  - Preemption
  - Selection of customer
- Customer-Server interactions
  - Type
  - Transportation
  - Push/Pull
- Queue
  - Capacity
  - Queue discipline or ordering
  - Release quantity

Table 2.6: Summary of the characteristics of queueing-system elements.

Since models are abstractions of reality, they must have some assumptions. In simulation modeling, the modeler chooses the assumptions. While having this freedom is good, since the model is not bound to unrealistic assumptions, it does present a challenge—what to

assume and what not to assume. The basic premise is to include only enough detail and enough realism to answer the questions being posed and no more.

The concepts and behaviors described in this chapter are fundamental and are used throughout the book; they form the basis of discussions of simulation methodologies and applications. Key points include the following:

1. Queueing systems are
    a. everywhere and form the basis of business systems that produce both goods and services.
    b. dynamic, change over time, and typically contain a high degree of variability.
    c. composed of three basic elements: customers, servers, queues. Characteristics of these elements are the key contributors to system performance.
    d. measured by customer lead time, customer wait time, server utilization, number in queue, etc.
    e. analyzed considering the fundamental tradeoff between the cost of customer waiting and the cost of providing the service; alternatively, a balance between customer wait time for service and server wait time for customers (idleness).
    f. analyzed either analytically using formulas from queueing theory or using simulation.
2. Typically, the arrival and services processes are random variables.
3. Representing the variability in systems and system dynamics is a key challenge, but is essential for making good decisions.
4. Customer wait times increase with
    a. increasing server utilization.
    b. increasing variability in the arrival process or service process.
5. A basic relationship from queueing theory, known as Little's Law, that was restated by Hopp & Spearman is Work in Process = Throughput × Cycle Time.
6. For the arrival and service processes, times and rates are inverses (e.g. if the average time between customer arrivals is 10 minutes, then the average arrival rate is 1/10 customer per minute or 6 customers per hour).
7. Discrete-event simulation is
    a. used to model queueing systems, and other types of systems.
    b. used to represent systems in which states change at discrete points in time.
8. Both time-weighted averages and observational averages are used to compute performances measures from simulation models.

References

1. Babin, P. and Greenwood, A. "Discretely Evaluating Complex Systems," *Industrial Engineer*. Vol. 43, No. 2, 2011, pp. 34-38.

2. Baxter, G. and Sommerville, I. "Socio-technical Systems: From Design Methods to Systems Engineering." *Interacting with Computers*. Vol. 23. No. 1. 2011. pp. 4-17.

3. Emery F. and Trist E. "Socio-technical systems." In: Churchman C., Verhulst M., editors. *Management Science Models and Techniques*. Vol. 2. Oxford, UK: Pergamon. 1960. pp. 83-97.

4. Hopp, W. and Spearman, M. "Factory Physics." *Foundations of Manufacturing Management*. 2nd Ed. Boston: Irwin McGraw-Hill, 2001.

5. incose.com, accessed 20 Jan 2015.

6. Law, A. *Simulation Modeling and Analysis*, 4th edition. New York: McGraw-Hill. 2007. p. 5.

7. Little, D. and Graves, S. "Chapter 5 – Little's Law" in Chhajed, D. and Lowe, T. editors. *Building Intuition: Insights From Basic Operations Management Models and Principles*. Springer Science + Business Media. 2008.

8. Little, D. "Little's Law as Viewed on its 50th Anniversary." *Operations Research*. Vol. 59. No. 4 May – June 2011, pp. 536-549.

Chapter 2 Review Questions

1. What is another name for a queue?
2. What are the three main elements of queueing systems and identify some characteristics of each element.
3. Why are queueing systems dynamic?
4. What are the two components of lead time?
5. In basic queueing theory and simulation models, what is the assumed order in which a server processes customers?
6. What is meant by a *stationary* arrival process?
7. What is the difference between the actions of balking and reneging?
8. Explain what happens when a customer "jockeys".
9. Identify reasons why a server is not available 100% of the time.
10. Describe the two different behaviors that occur between a server and a queue when a shared resource is used to transport items between the queue and server.
11. Describe the systems that are defined by the notations:
    a. M/E/2/10//SPT
    b. D/G/1
    c. M/G/3
12. Define Little's Law.
13. Explain the relationship between work-in-process inventory, throughput, and cycle time.
14. Define the basic tradeoff that is typically considered in the analysis of queueing systems.
15. Describe how variability affects queueing system performance. For example, if the variability in process times is increased, then how are the system's waiting times, number of customers waiting, etc. affected?
16. Explain how the study of queueing systems differs between the queueing theory approach and the discrete-event simulation approach.
17. Explain the difference between an observational average and a time-weighted average.

18. Identify some of the possible states in a queueing system.
19. Describe what is meant by a *discrete-event* system.
20. What is a replication?
21. Provide examples of queueing systems and define for each system the
    a. customers in terms of their types and their queueing behavior
    b. server(s) and their availability
    c. problems that need to be solved in those systems
    d. key measures of system performance
22. If the mean time between arrivals is 12 minutes, what is the mean arrival rate per hour?
23. What is the assumed distribution of service times in basic queueing theory models?
24. If the arrival and service rates for a M/M/1 queueing system are 18 and 20 per hour, respectively, then the server is expected to be busy what percentage of the time?
25. If the arrival rates to a system are Poisson distributed, then how are the time between arrivals distributed?
26. A single maintenance crew takes care of all repairs to Bully's giant server farm, which runs all the time. The average breakdown rate of the servers is 1 per hour; on the average it takes 50 minutes to repair a server. Use basic queueing theory to answer the following questions.
    a. What is the assumed distribution of the time between breakdowns?
    b. What is the average time between breakdowns, in minutes?
    c. What percentage of the time is the maintenance crew busy?
    d. On the average, how many servers are waiting to be repaired?
    e. What is the annual cost of downtime, if Bully loses $50 for every hour a server is down?
27. Calls come into a single help desk at an average rate of 5 per hour. On the average, the operator spends 10 minutes with each caller. Use basic queueing theory to answer the questions.
    a. What is the assumed distribution of the time between calls?

b. What is the average time between calls, in minutes?
c. What percentage of the time is the operator busy?
d. On the average, how many callers are waiting to be served?
e. On the average, how long are callers on the line?

28. Maintenance workers arrive at a parts center operated by Bob at an average rate of 15 per hour. Bob requires 3 minutes to process each maintenance worker request, on the average.
    a. On the average, what percentage of the time is Bob busy?
    b. On the average, how many maintenance workers are in line waiting to see Bob?
    c. On the average, how long do maintenance workers need to wait in line in order to see Bob, and how long are they in the system?
    d. Should a proposed improvement be implemented if it reduces Bob's average process time by 0.25 minutes? Assume the improvement costs $10,000 per year and downtime on the machines (that the maintenance workers are servicing) costs $100 per hour in lost profit. Assume the parts center operates 2,000 hours per year.

29. Consider a single-server FIFO queueing system where the time between arrivals is exponentially distributed with a mean of 10 minutes. On the average, 40% of the arrivals are Red items and the remainder are Blue. The processing times for Red and Blue are exponentially distributed with means 12 and 6 minutes, respectively.
    a. What is the average service **rate per hour** for Red items?
    b. If the system is simulated for 40 hours, how many Red items would you expect to arrive?

30. Consider the following clinic operations in order to answer the following questions. The time between arrivals of patients to the imaging facility (e.g., X-ray) is exponentially distributed with a mean of 30 minutes, and the time to process a patient is exponentially distributed with a mean of 25 minutes.
    a. On the average, how many patients would the clinic expect in 8 hours?
    b. On the average, how many minutes do patients have to wait from the time they arrive until they begin the imagining process?
    c. On the average, how many patients are waiting for imagining?

d. On the average, for what percentage of the time is the imagining facility being utilized?

31. If λ = 5/hour and μ = 6/hour what is the effect of the following on average wait time. Assume exponential times between arrivals (mean 12 minute), exponential service times (mean 10 minutes), and FIFO queue discipline unless otherwise noted.
    a. Single server
    b. Two parallel servers
    c. Constant 10-minute service times
32. Re-consider Question #30. Answer all of the questions assuming there are two identical imagining facilities for the patients to use.
33. Re-consider Question #30. Answer all of the questions assuming the standard deviation of the process times is one-half of the mean and not the same as the mean, as is the case for the exponential distribution.
34. Re-consider Question #30. Answer all of the questions assuming the process times is a constant 25 minutes.
35. Compare the performance of the systems in Question 30, 32, 33, and 34.

Chapter 3 Using Simulation to Solve Problems

Objective

Experience how simulation is used by interacting with pre-built simulations, analyzing results, and recommending solutions to specific operational questions.

It is important for all stakeholders (i.e., those who own the problem being simulated, those supporting the simulation, those building the simulation, and those who will make decisions based on the simulation) to understand the power of simulation and how it can be used to solve problems. This chapter uses six diverse models to illustrate how simulation can be applied to improve system performance. All models employ a custom interface so that the users play the role of decision maker and Occasional User. The focus is on analysis and decision making and not on software and model building—that comes later in the book.

Section 3-1 Using a simulation

Simulation is an applied technology. There is little value to a simulation built without a reason or a problem to be solved; therefore, simulation must be experienced to be appreciated and understood. This chapter is an introduction to simulation applications through the eyes of an Occasional User.

An Occasional User, as defined by this book, is a user who doesn't use simulation on a regular basis but appreciates the value of simulation. Usually this person's main responsibility is something other than simulation—a manager, lead engineer, team leader, etc. The Occasional User is not proficient in building simulation models but, with a little review, can make use of a simulation built by others, especially if the simulation is intuitive. It is assumed, however, that the user is familiar with some of the basic concepts of simulation.

The Occasional User knows enough about simulation to create a functional specification for others based on his or her knowledge of the system being simulated. As the "owner" of

the problem or issue to be studied by the simulation, the Occasional User can detail the simulation requirements, define the scope, and establish the metrics for the analysis.

This chapter demonstrates how simulation can be used by the Occasional User. Most simulators on the market today are set up for Occasional Users to run simulations and even provide "run-time" versions of the software. While *FlexSim* software is used in this book, the methods for doing the exercises in this section should be typical of other products as well.

The simulations used in the following examples represent models built by Intermediate and Advanced Users. They are usually prepared for Occasional Users to use and therefore can be opened and run without any knowledge of the underlying software application. Details of the application environment for *FlexSim* are contained in the help files.

Section 3-2 Simulations with random events

The challenge for the Occasional User is to understand the simulation he or she is viewing and to interpret the results. The simulation interface will normally allow the user to change certain variables and view previously defined metrics. For a simulation that contains random events, this can be confusing.

Almost every event in the world will happen differently every time it occurs. Each event is unique and influenced by its environment. This phenomenon is characterized in simulation models by using random events defined by probability distributions. For example, distributions are used to represent arrival processes, such as patients coming to an emergency room; process times, such as the time to assemble products; transportation times, such as the time to move material from one location to another; interruptions, such as machine breakdowns, etc. These statistical distributions can be developed based on historical data or by professional estimate.

As shown in Figure 3.1, and as described above, input to a simulation model involves random variables; therefore, the model is considered stochastic since it uses random variables. These stochastic inputs are a subset of the uncontrollable variables in the model (i.e., they are not under the control of the decision maker).

Controllable factors, or decision variables, are those that are manipulated in the model to improve a system's performance (e.g., number of workstations, number of transporters to move products, processing rules). Since the inputs to a model are stochastic, the output or performance measures are also stochastic. Care must be taken to evaluate stochastic output. One run of a model represents just one way a system may behave; therefore, a decision should not be made on whether the model is valid or if one alternative is better than another based on a single run since one run could represent an extreme case.

The bottom line is that a simulation that utilizes random events must be either run for a long period of time or many times (i.e., for many repetitions or replications). This is the only way to obtain data that can be used to make an accurate decision. Statistical methods can be used to determine the length of a simulation run or the number of times a simulation

model needs to be run to ensure a specific level of confidence in the estimate obtained from the simulation model.

Figure 3.1 illustrates how simulation models are used. Once the values of the decision variables are set, the model is run and output is obtained. The user then changes the decision variables (based on the output from the initial run as well as the specified system performance criteria) and re-runs the model. The user continues this iterative process until a satisfactory system configuration is realized.

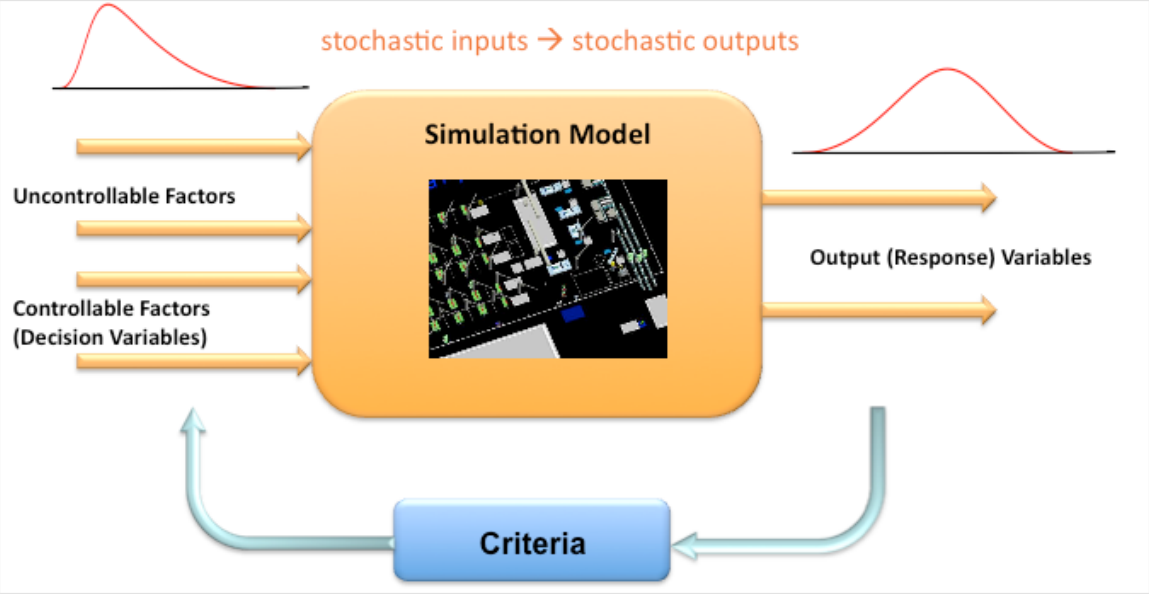


Figure 3.1: Stochastic simulation.

Almost all the simulation models in this book include some degree of randomness. An option to use the same random number streams was selected for each example so that variable changes could be more easily compared; however, to maintain focus on the use of simulation by the Occasion User, only a single or a few runs will be requested to answer the questions. It needs to be stressed that this is not meant to downplay the importance of the statistical analysis, which is a critical part of any simulation project; however, it is common for Occasional Users to simply run the simulation and collect data without regard to the underlying statistics. This is because he or she typically uses a model developed by someone else, and the experimental conditions are pre-set within the model execution environment. Simulation models are commonly embedded in decision support systems that free the user from connecting to required data sets, formatting results, and setting up experiments.

Three chapters in this book deal with statistical analysis—one for input data analysis (Chapter 8) and two for analyzing simulation output (Chapters 10 and 11). Chapter 11 includes a discussion on the use of the Experimenter functionality included in *FlexSim*. The Experimenter facilitates carrying out multiple simulation runs and performing statistical analyses.

Section 3-3 Examples and exercises

This section provides six simulation models that illustrate a wide range of problems that are typically addressed by using simulation modeling and analysis. Using models, the user can adjust

- capital investment amounts and resource staffing levels to balance customer service level and cost (operators and cars on a roller coaster);
- arrival sequence, number, and timing of transport units in a distribution system to reduce work-in-process inventory (space station);
- production rate and batch size to improve throughput (bottling plant);
- investment alternatives to improve productivity (electronics assembly);
- scheduling of transportation resources to more efficiently carry out a project (marine system);
- procurement of equipment to dynamically meet customer needs (Zombie).

The examples focus on system operation and improvement issues and not on model building. The intent is for the user to play the role of the Occasional User. All the models use a custom interface, like the ones shown in Figure 3.2, to minimize the user's interaction with modeling details or the simulation software. The user interacts with the model in a decision support role.

The models are all developed in *FlexSim*, but they could have been developed in any simulation software. They do not require any knowledge of the *FlexSim* software other than to open *FlexSim* first and then select to open a saved model. The simulation model and a user interface are displayed when the file is opened.

The custom interface varies for each example but has a similar structure throughout. The interface has tabs that relate to specific actions the user may want

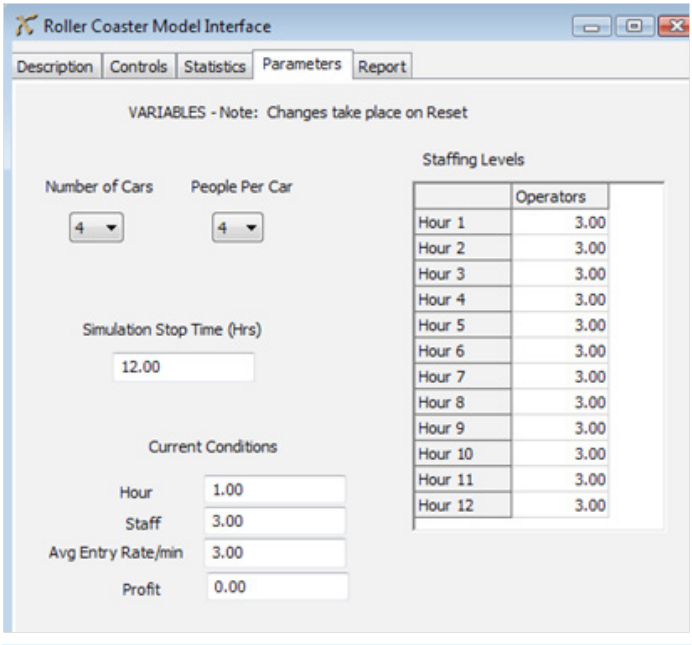


Figure 3.2: Examples of custom interface for example models.

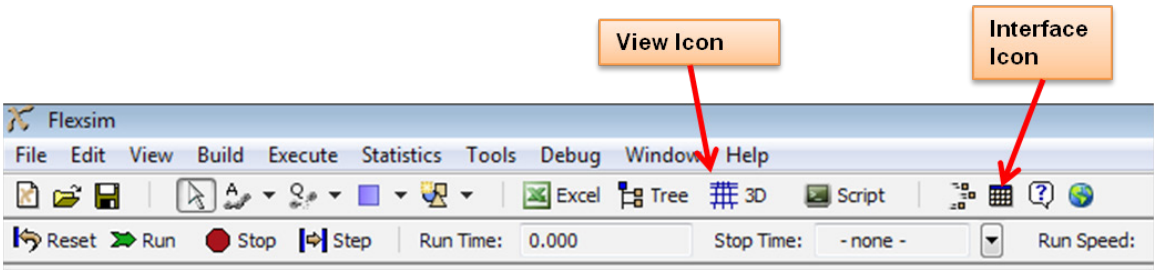


Figure 3.3: FlexSim general interface row.

to perform. For example, the Controls tab, as shown in Figure 3.2, provides a convenient way to reset, start, and stop the simulation run, as well as to change speed and view the model running from different perspectives or views.

Other tabs contain the decision variables that can be changed by the user. The Statistics tab contains statistical summaries of key metrics, while the Report tab contains specific metrics collected during the run, usually in a table format. Table values can be copied and pasted into Excel for additional analysis. The total elapsed time is given at the bottom of the interface.

While the simulations can be run from the custom interface, the user may also use the upper control bar of the *FlexSim* application as shown in Figure 3.3. For example, if the custom user interface or model view is inadvertently closed, it may be opened by going to the View drop down menu and selecting *Model Control GUI*. The model may also be stopped, started, or reset from the control bar, and the speed may also be changed.

Additional details of the *FlexSim* application environment, including moving around the 3D screen, are in the help files.

Exercise 3-1 Coasting around

Background

As the manager of the "Super Rocket" roller coaster ride at the Grand Bay amusement park, you are responsible for the day-to-day operation and financial performance of the coaster (see Figure 3.4). Your parent company owns the coaster and leases the space from the park. The coaster at this park is only three years old and is the lowest in financial performance of all the parent company's assets.

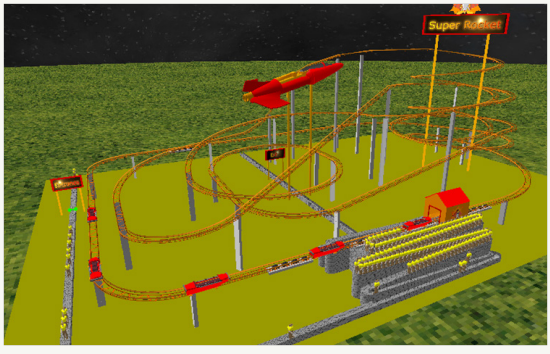


Figure 3.4: Super Rocket roller coaster.

This year you, the manager, must show improved financials for the ride or find another job. There are many trade-offs to consider. You are in competition with all the other rides at the park. You get paid by the park based on the number of riders served. Customers may love the coaster, but if they must wait in line too long, they will walk away or not bother to ride again that day. Potential customers who see long lines will opt for other rides with shorter lines. Most choices to increase riders involve increased costs of equipment or staffing.

Your brother-in-law happens to be familiar with simulation methods and offered to help. You explain your problem and provide base data about the operation. You are appreciative of the results he gave you and decide to run the simulation yourself.

Problem statement

How can the profitability of the coaster be improved during the operating season?

Operating data

Riders enter the coaster line at different rates during the 12-hour operating day. Operating data show the rates per minute to have a Poisson distribution; their means are provided in the table in Figure 3.5.

	Riders/Min
Hour 1	3
Hour 2	4
Hour 3	5.5
Hour 4	7.5
Hour 5	8
Hour 6	9
Hour 7	10
Hour 8	10
Hour 9	8
Hour 10	5
Hour 11	4
Hour 12	2.5

Figure 3.5: Rider average per hour.

A minimum of three workers are needed to operate the coaster. One person takes tickets, one

supervises loading the cars, and one supervises unloading the cars. Extra people can be added.

The first additional person helps to unload the cars, the next helps to load the cars. In each case the unloading or loading times are reduced.

The coaster is currently operating with a base of four cars, each with a capacity of four riders. New cars can be added to the base with the depreciation costs added to the operating expense.

Rider capacity of the cars can also be increased with a retrofit paid for through a depreciation charge. Other operating costs include depreciation costs for the present cars, hourly wages, and general maintenance costs. A summary of the operating costs is provided in Figure 3.6.

	Financials	
CreditPerPerson	$	1.00
Wage/Hr	$	(6.50)
OldCarDep/Hr	$	(3.00)
NewCarDep/Hr	$	(7.50)
Maint/car/hr	$	(25.00)
CarUpgradeDep/Hr	$	(1.10)

Figure 3.6: Coaster operating costs.

The following options are available to increase financial performance:

- Increase number of cars
- Increase car capacity (upgrade)
- Increase staffing level for an hour

Expected results

Provide an operating plan that includes staffing levels. Determine if there is sufficient return to request additional cars.

Modeling and analysis issues

- How long should the simulation run? The simulation stop time is set for 12 hours; selecting the run button will continue the simulation from the stopped point.
- What variables should have the most impact?
- Is there a plan for trying different changes?
- What is most important about the financial report?
- While financial results are the primary metrics, do you feel long wait times will reduce future riders?

Exercise 3-2 Martian Transfer Station

Background

The space exploration task force on Mars has seven laboratory stations (see Figure 3.7). Supplies are brought in on cargo rockets and then transferred to their destination by surface vehicles. Pallets are recycled and brought to the transfer station where two robotic cranes separate the incoming supplies. When a pallet is complete for a station, a guided vehicle picks it up and carries it to the station. The transfer station is operated by two locals.

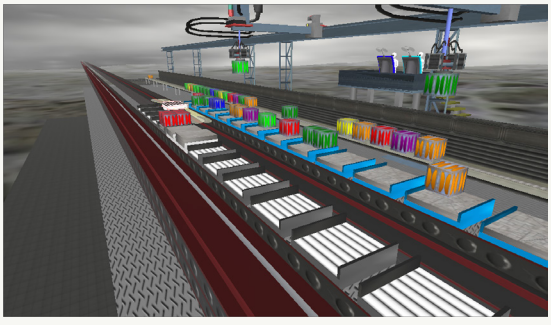


Figure 3.7: Martian transfer station.

Problem statement

The stations are receiving increasing amounts of supplies—some of them with time-critical contents. Can the work-in-process inventory be reduced by changing operational parameters?

Operating data

The transfer station has two working cranes. Empty pallets are delivered by a transporter vehicle and transferred to the loading station with a shuttle. A crane will only start loading the next box of supplies if there is a pallet in the appropriate transfer station and the correct load is available.

Each pallet contains four crates. When a pallet is filled, it is moved to the distribution side by the shuttle, and a call is made for pickup. When the transporter that was called stops, the full pallet is loaded and taken to the experiment station. Transporters already en-route will not stop for the pallet.

Only a certain number of transporters are available on the planet. They are sent out at specific time intervals. After going through the transfer station, they continue to one of the seven lab stations. After depositing any load, they continue to a transportation center, where they are brought back as a group to the sending station to once again be sent to the transfer station.

The supply rocket is normally packed randomly on earth as supplies are brought to the spaceport. Requests have been made to stage the rocket load first, and then group them on the rocket by sequence, with several boxes grouped by their destination.

One time unit is one Martian01 second. As on Earth, there are 60 Martian seconds per Martian minute and 60 Martian minutes per Martian hour.

The operating rules can be expressed as follows:

- Transporters leave the sending station at a fixed interval.
- If empty pallets are needed or a pallet is in place and ready for pickup, a request is sent to the sending station and the next transporter is given the assignment. A transporter can have only one assignment at a time.
- Only one transporter can be at the transfer station at a time—others will bypass the station.
- Crates from the supply ship arrive either in a random sequence or grouped for each destination.

Expected results

- Run the simulation for a period of time and observe the system dynamics. How do the cranes work to position crates on pallets?
- Devise a plan to maximize the distribution of materials to the laboratory stations. What are the tradeoffs with cost per crate?

Modeling and analysis issues

- What is the impact of the arrival sequence?
- Can you tell by observation what logic is employed in moving material from the space ship to the pallet layout?

Exercise 3-3 Slime Inc.

Background

As the new engineer working for Slime, Inc., you've been asked to familiarize yourself with the company's operations (see Figure 3.8). This plant only bottles the green variety of slime and ranks in the middle of all plants in performance. Overall plant performance across package sizes is a key metric, but performance of packing the popular 12-pack of slime is critical from a market perspective. You've found a simulation of the plant that was built last year and decided to try it as a way of understanding the operation.

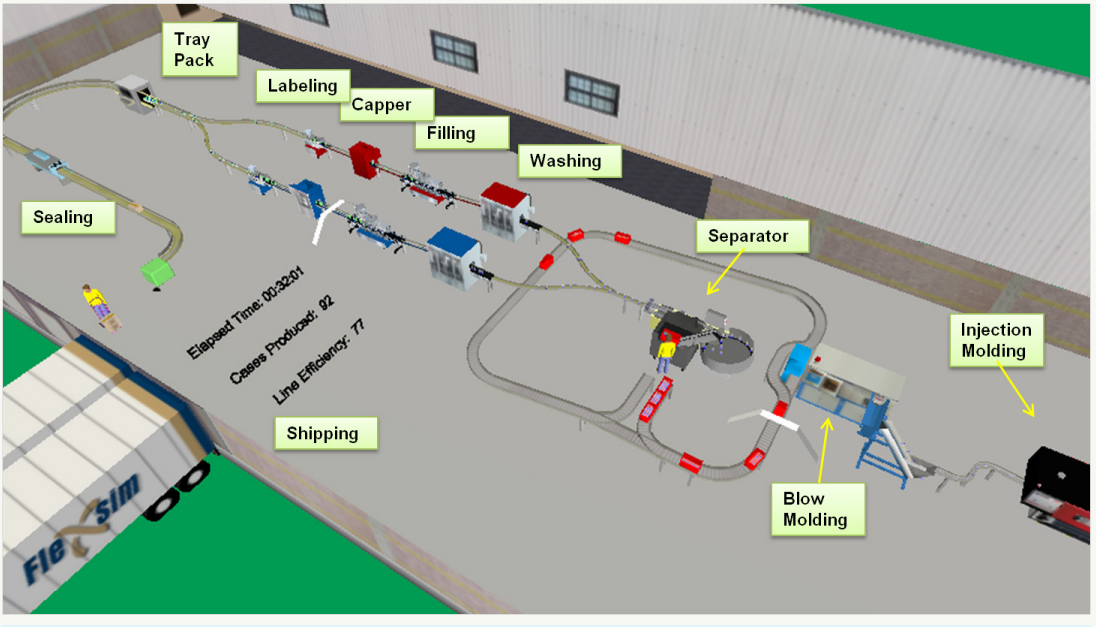


Figure 3.8: Slime Factory.

Marketing is pushing the plant to agree to higher speeds and smaller tray pack units because people can only handle so much slime at a time.

As an incentive, bonus points are awarded based on meeting production targets and result in additional pay at the end of the fiscal year.

Problem statement

What are the best operating conditions to meet marketing and plant needs?

Operating data

Starting from raw plastic, the bottles are created with an injection molding machine. The plastic forms are puffed into the bottle shape by a blow molder. The empty bottles are put into baskets that transfer them to an un-scrambler, which in turn lines them

up to go into the filler lines. There are two filler lines that clean, fill, cap, and label the bottles. The two lines join at the single tray packer. Trays are conveyed to a case closer and brought to an operator who loads a hand truck to put them into waiting trucks.

One operator is responsible for the molding and unscrambling operations. This operator unloads the baskets of clean bottles and places the basket back on the return conveyor. The operator also must attend to any downtime on the equipment in the area. The second operator carries cases to the trucks and fixes any problems with the filling and case packing operations. Operators are concerned about their workloads under the higher production rates.

Once a truck is filled, the facility's doors will close and re-open only when another truck has arrived.

The plant can operate at molding speeds between 45 and 85 bottles per minute. At higher speeds the machines become less reliable and jam more often. This requires the operators to stop their production-related work and fix the equipment.

The tray packer can handle 12, 16, and 24 count trays. Production standards for the plant were set based on the 24 count trays when the plant was built. At a given bottle rate, more 12 count trays are produced, thus requiring the second operator to make more trips into the trucks carrying 5 cartons per load.

Plant efficiency is rated at better than 97% based on their standard 24 count tray and 55 bottles per minute. Lower efficiencies will result in lower bonus points.

Expected results

Considering changes in bottles per case and the production rate, prepare a recommendation for the next engineering group meeting indicating what you've found out about the operations and what might be done to improve performance and meet marketing needs.

Modeling and analysis issues

- For what period of time should the simulation run?
- Does the production or utilization report give any indication of a bottleneck?
- Note what happens to operations as line speed increases.
- How busy do the operators appear to be?
- Is there a tradeoff between the plant's operating performance metric and the need for smaller sizes?

Exercise 3-4 Danson Electronics

Background

The Danson Electronics plant (see Figure 3.9) in Pembroke has been assembling high quality DVD players for two years. To improve their operations, they instituted a continuous improvement program with teams active on the plant floor.

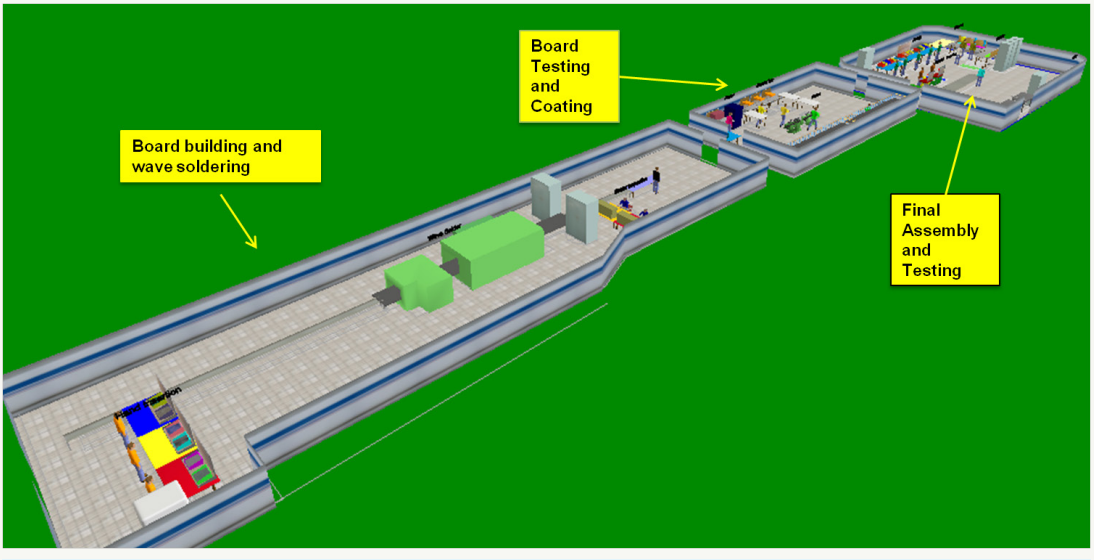


Figure 3.9: Danson Electronics.

Problem statement

Which proposal should be adopted, or should all proposals be considered with a priority established. What else should be done?

Operating data

The plant is divided into three buildings. In the first, double circuit boards are populated with components and then sent to a wave soldering machine. After inspection they are grouped onto a tray and manually transferred to the next building.

In the second building, the double boards are separated using a router, and the individual boards are then put through a series of tests. After testing, the boards are coated and put in racks to dry. Dry boards are manually transported to the third building for final assembly.

In the assembly building the boards go through a series of assembly steps and tests. The assembled units are placed on a table and moved to an aging box. After sitting in the aging box, the units are tested again before moving on to shipping.

Four proposals have been put forth to improve plant productivity:

- The first building team notices that boards are backed up through the wave soldering machine. They attribute the backup to the fact that the router operator only has another batch of boards brought over when the current tray was empty. They want a surge added before the router so that boards can be moved more effectively.
- The second building team wants to purchase a new router that would decrease the routing time by 20%.
- The third building team wants new assembly machines to increase their throughput by 20%.
- The union wants another person hired for the second building to help move boards between buildings 2 and 3.

Expected results

Suggest how the improvement team recommendations should be handled.

Modeling and analysis issues

- What should the simulation run time be?
- Where do backups occur?
- What priority should be given to the suggestions?
- What improvement level would be considered good?
- What is most important: total production, WIP, or quality numbers?

Exercise 3-5 Maritime Ltd.

Background

The town of SeaSide is situated on the coast and enjoys a vibrant tourist trade. Last year a hurricane badly eroded a significant part of their shoreline and exposed their main beaches to further damage. The town council has hired Maritime Ltd. to build a new breakwater using large rocks from an excavation further up the coast (see Figure 3.10). Maritime has quoted the job for $11 million with significant penalties if the job exceeds the amount and a reward based on coming in under the price.

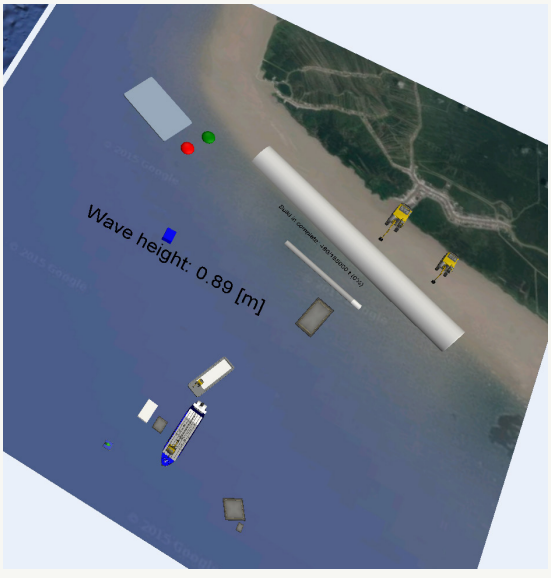


Figure 3.10: Breakwater construction.

To plan the project, Maritime has engaged a consulting firm to build a simulation to help plan their options. Since Maritime doesn't want to give the consulting firm access to their cost information, they have decided to have their engineers input the cost data and run the simulation themselves. The consulting firm acquired tidal data to use in the simulation.

Problem statement

Use the simulation to determine the best approach to finish the job under the quoted price.

Operating data

The new breakwater will require 165,000 tons of rock carried by coastal barges (coasters) for 833 nautical miles from the quarry to the town. The quarry is open for the loading coasters 12 hours a day on weekdays only. The loaded coasters sail to a position just outside the coastline where the rocks are transferred to a barge. The barge then sails to the coastline and discharges the stones directly in front of the location. Transfers can't be made if the wave action is too high.

It is possible to use a pontoon to buffer rock material off shore. The coaster can discharge to the pontoon while the barge is making a transfer so that the barge can then load from the pontoon if the coaster has left. The pontoons are available in two sizes.

The choices and costs associated with renting equipment are shown in Figure 3.11. Site costs are $37,250 per day.

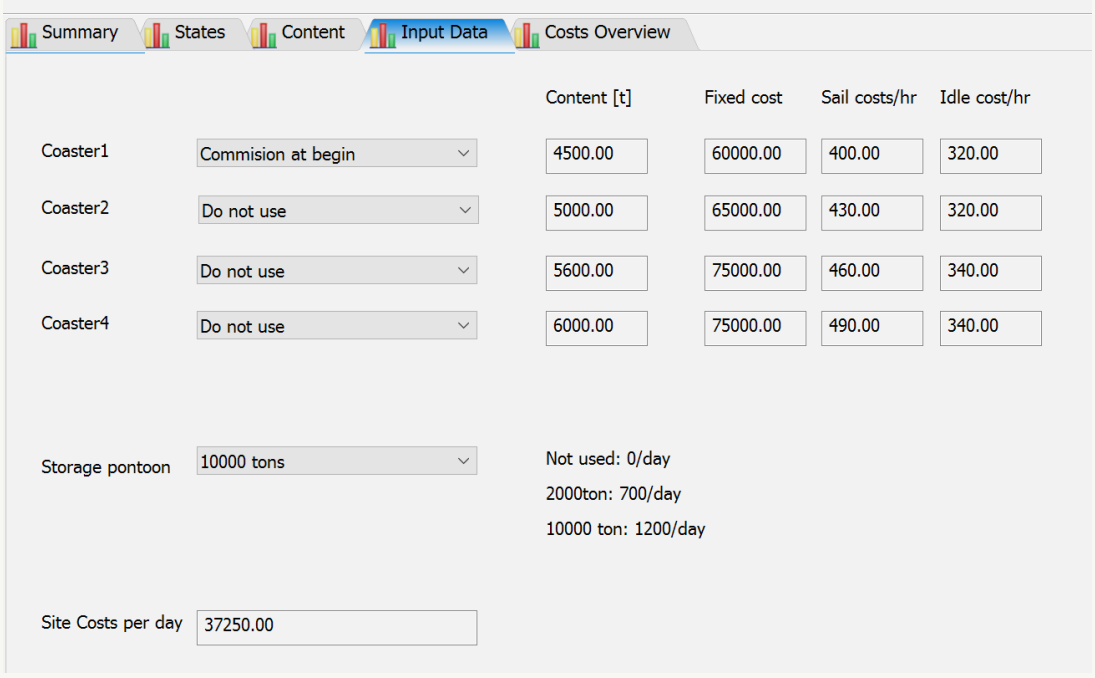


Figure 3.11: Marine project costs and choices.

The simulation has three views. One for the quarry, one for the coastal travel, and one for the project site. Charts accumulate dynamically and the costs are calculated at the end of the simulation when the transfer of rocks is completed.

Exercise 3-6 Zombie Batteries, Inc.

Background

It finally happened—a virus outbreak changed 80% of the earth's population into the living dead. But in a twist from mythology, the zombies don't need to feed on brains. They want power—battery power! With a new battery, they go back to their caves, hook up, and enjoy the charge.

The human survivors found the technology to produce energy cubes, containers that seem to satiate the undead for days. In your quest to be a plant manager at a Zombie Batteries plant in one of the last human cities, you must pass a test that shows your ability to make quick decisions under pressure (see Figure 3.12).

The test is a simulation of the plant where your job is to fabricate enough energy cubes to keep the advancing undead horde away from the citizens of the town, buying the town precious time to continue to sustain the local economy and provide you with the vital resources to improve the manufacturing plant.

Problem statement

Your problem is to come up with a dynamic operating strategy to satisfy an army of the living dead. The simulation will determine your ability to make decisions to keep the company profitable. Your job is to use the simulation as battery demand varies to achieve the best profit. The test period is short so you have only 2.5 minutes of simulation time to profitably run the plant.

Operating data

There isn't a lot of competition for selling batteries to the zombies since they are… well… a little difficult to deal with. They will pay full price for batteries that are fully charged and haven't been drained from sitting on the shelf. Also, if there are no batteries available when zombies arrive, they get very mad.

The energy cell production facility has three processes that must be completed to produce a fully-charged energy cell from an empty cell; they must be:

1. filled with a radioactive material;
2. taken underwater for a high-energy charge;
3. run through a centrifuge to render the cube stable.

Once production has completed, energy cells are placed in a queue for the zombies to pick up – which is good, because any zombie that makes it to the queue and DOESN'T see a viable energy cell will get angry and damage your equipment. Try and keep the queue stocked to keep your machines in working order, and to allow the local economy to thrive in a zombie-free environment.

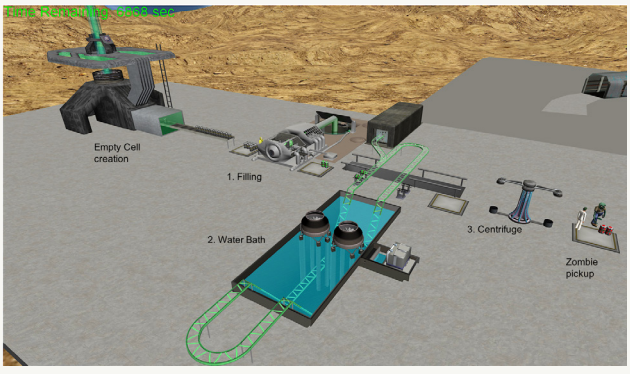


Figure 3.12: Zombie Batteries operation.

But there's a problem: energy cells have a life span and the volatile energy charge inside will rapidly degrade. If a zombie arrives at the queue to find a weakened energy cell, it won't be satiated for nearly as long and they will be coming back faster and faster. Find the right balance of cells in the queue, repair the equipment the zombies destroy, and survive for 2.5 minutes.

When the model starts, you will have a working plant with one radiation filling machine, one radiation operator, one tram (to transport the energy cell through the charging pool), one centrifuge machine, and one centrifuge operator. You will have $1000 to purchase machines, trams, or operators.

Each time an empty energy cell canister arrives it will cost you $15. You can adjust how often they arrive with the slider at the top of the control panel shown in Figure 13.13. It is critical that you do not spend all your money by having too many empty canisters arrive before they are needed.

You earn money each time a zombie appears and takes a charged energy cell back to the hideout. A fully charged energy cell will get you $50. As energy cells decay, you will get less money for each energy cell. You can only buy production equipment and operators with your cash supply. This supply will fluctuate depending on the cost of canisters and the payments from the town, which vary depending on how effective you've been at keeping zombies away (i.e., how full the energy containers are when they're taken). The cost of each piece of equipment and the cost of each operator are shown in the control panel.

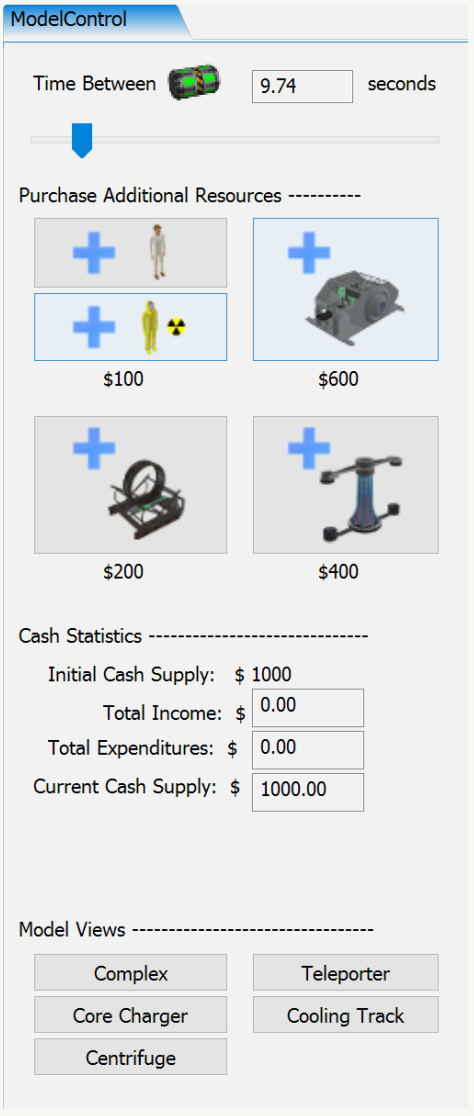


Figure 3.13: Battery controls.

Chapter 3 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important—in the chapter.
2. Discuss why simulation is considered an applied technology.
3. Select an operations system and identify some controllable and uncontrollable variables; identify which of those might be considered random.
4. Discuss why stochastic simulations need to be run more than one time.
5. Describe what knowledge is needed for an Occasional User to use simulation as an effective decision-making tool.
6. Describe the three main steps involved in developing and using simulation to support decision making.
7. Provide some examples of random variables that are found in simulation models.
8. Provide examples of controllable factors in a simulation model.

Chapter 4 Building Basic Simulation Models

The Occasional User uses simulation as part of his or her job description, but it is not usually his or her main focus. Typical job titles of the Occasional User include operations analyst or industrial, process, packaging, or manufacturing engineer. Individuals engaged in manufacturing research and development may also be Occasional Users. Typically, the Occasional User will be involved with simulation five or six times per year. Occasional Users can build relatively straightforward simulation models using drop-down menus or wizards. When more detailed simulations are required, the Occasional User can call on technical support or third-party consultants.

Objectives

To start building simulations that represent the key physical and logical elements of operational systems using the basic components and standard logic that are part of every modern simulation application. Utilize the simulation output to analyze system performance.

This chapter defines the essential features needed to represent a system as a discrete-event model and introduces the basic structure and components found in most simulation software modeling and analysis packages. It includes a discussion of the modeling environment, introduces the basic structure and functionality of modeling objects, defines relationships among objects, describes how to move items through a model, and discusses obtaining output statistics from a single run of a model. This chapter will illustrate the object functionality as used in the *FlexSim* application. The remainder of the book builds the reader's capability to create and analyze simulation models through the *FlexSim* software. While this book exclusively uses *FlexSim*, the functionality is similar to other applications (though it may be implemented differently).

This chapter, and those that follow, contains exercises that will develop those simulation skills used by Occasional Users. The discussions in this chapter are meant to be a quick-start introduction to simulation modeling. Some of the topics will be explored in greater detail

in later chapters. Consult the User Manual and tutorials in the Help section of *FlexSim* for further information about using the software.

The authors strongly believe that any simulation course should include a discussion of the professional practice of simulation. As is stressed in this book, simulation is an applied methodology; thus it is important that simulation is applied correctly and effectively. This is discussed in the last two chapters.

When the topic of professional practice is covered in a course depends on the course objectives, content, and background of the students. We include the topic at the end of the book so that it gives the course designer the most flexibility.

If the course involves a project, we suggest covering the topic when the project is introduced. Also, we have found it is better to discuss the topic once students have some experience developing and analyzing simulation models.

Section 4-1 Essential features of discrete-event simulation models

It is important for all simulation users to understand that a simulation model is a simplified representation of a system that has a problem that needs to be solved. That representation, as shown in Figure 4.1, addresses three main features of any operational system: interactions, variability, and time.

The interactions occur among the many components that define a system and encompass relationships among system resources, such as people, equipment, material, and information. Building a simulation model is much like solving a puzzle—the modeler must decide what objects are needed to represent a system and how those objects are related to one another, organizing what may seem to be a jumble of information into a coherent picture. The resulting model is executed to give a likeness of system behavior and performance.

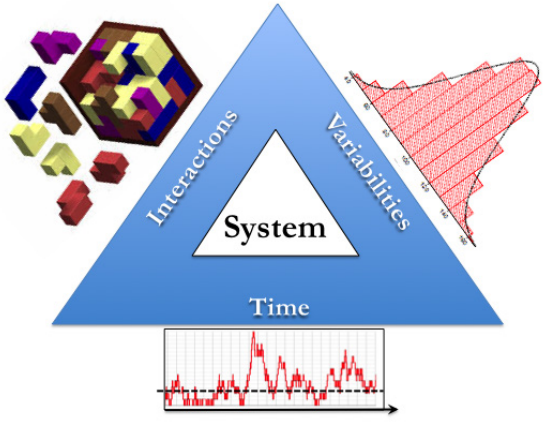


Figure 4.1: Main features of any operational system.

Simulation models also represent variabilities inherent in the system. The variabilities are both planned and unplanned. Planned variabilities include systematic variations in task times, operators taking breaks, resource changes due to shift schedules, etc. Unplanned variabilities include equipment failures, absenteeism, changes in the demand for products and services, quality problems, etc.

The third key feature that simulation models address is time. All operational systems are dynamic; that is, they change over time. One of the main reasons that operational systems are so complex to analyze is that the system interactions and variabilities combine over time

and result in system behavior. A simulation must also capture a system over an appropriate period of time that allows all important interactions to take place.

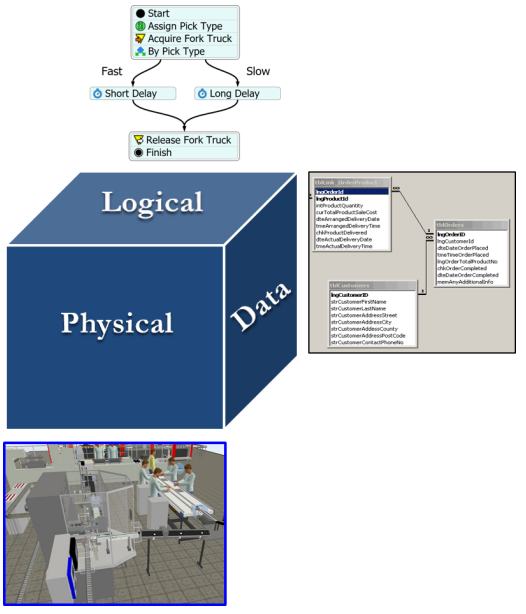


Figure 4.2: Simulation software concepts.

The interactions inherent in simulation models occur between objects that represent the actual physical resources (people, equipment, etc.) in the system. The objects are where system activities take place, such as waiting, storing, processing, transporting, etc. Objects are defined primarily in terms of their physical and logical characteristics, but they also include data. The basic notion of a simulation object is illustrated in Figure 4.2. The physical aspects of objects include their shape and appearance, size, and location in a three-dimensional space. Logical aspects of objects include what, when, and where to perform system activities. Data associated with objects include specific information about the physical and logical aspects of the object, such as dimensions, speeds, graphical image, next operation, process time, etc. Data may be specified in a variety of ways, such as as a constant, array or table, or probability distribution.

Objects in most simulation software represent common functionalities found in operational systems. These objects are the building blocks of a model; they encapsulate the physical, logical, and data aspects of such functions as processing, queueing, transporting, etc. Figure 4.3 defines some of the basic functionalities and the name of the object(s) used in *FlexSim* to implement the functionality in the model. This is just a partial list and is only used to introduce basic functionalities. As shown in the table, some functions can be performed by different objects; for example, an item may be moved by a fixed object such as a conveyor or a mobile object such as a person. The choice of the object depends on the context in the system.

Function	Object
Create	Source
Destroy	Sink
Delay – planned duration	Processor, Multiprocessor
Delay – unplanned duration	Queue, Conveyor
Group	Combiner
Ungroup	Separator
Convey	Conveyor, Task Executer

Figure 4.3: Relationship between function and *FlexSim* object.

Modeling objects in *FlexSim* are very powerful and yet very user friendly. The objects are defined in terms of a well-structured, consistent user interface. The objects enable the user to specify varied levels of physical, logical, and data characteristics depending on the complexity of what is being modeled. Characteristics may be (1) entered directly with numerical

values or through check boxes, (2) chosen from a comprehensive list of options, known as picklists, (3) selected from tables or external entities such as spreadsheets and databases, or (4) specified through flow charts or custom computer code snippets. Objects are linked through a hierarchical data structure. As a simulation executes, objects track the states that they are in (e.g., busy/idle, number in the object) over time and summarize the state data into statistics such as utilization, average number of items in an object, and average time items spend in the object (stay time).

Section 4-2 Simulation environment

FlexSim, like other professional simulation software applications, provides extensive support for building and analyzing simulation models. Figure 4.4 outlines the basic components found in such software. Most of the components will be discussed later in the book.

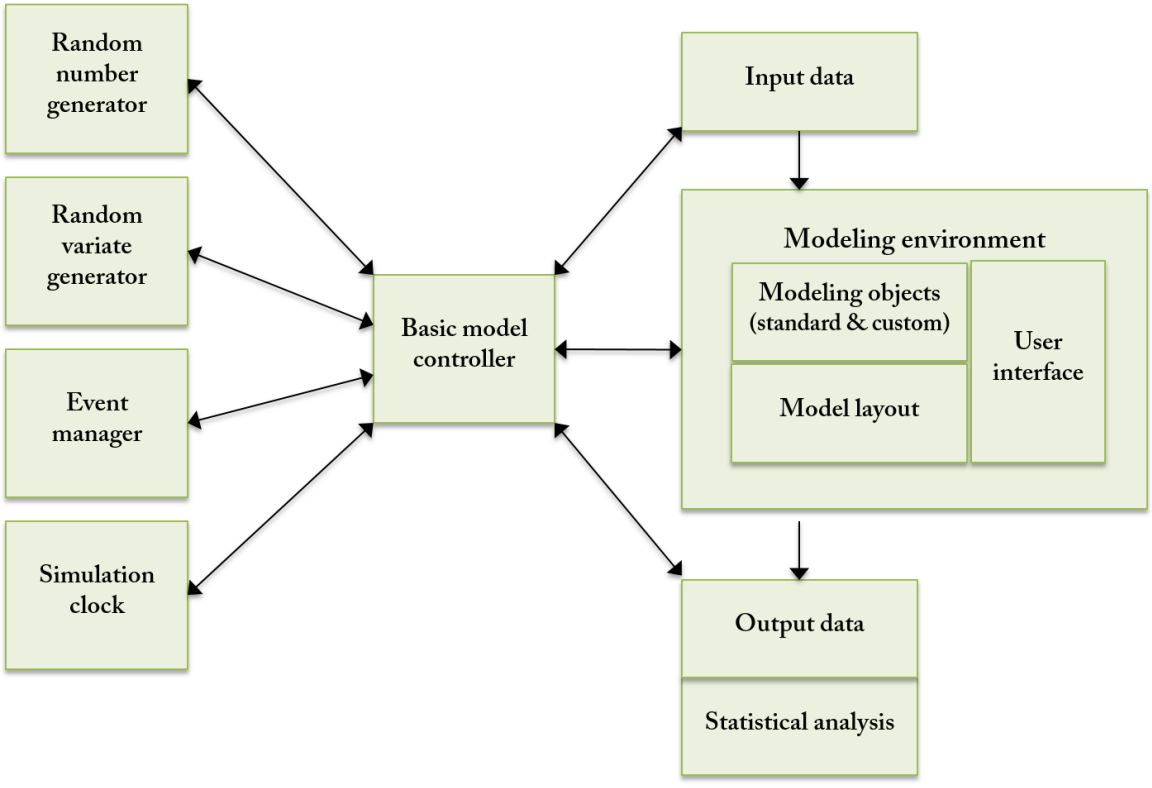


Figure 4.4: Software simulation concepts.

This chapter concerns the modeling environment and focuses on the components in the right portion of Figure 4.4. As introduced above, all simulation software provides a set of pre-defined modeling objects that facilitates model building. Such objects include queues that store items awaiting processing (due to unplanned delays), processing objects that modify or create planned delays for items as they flow through the model (service operations), transporting objects that move items through the model, etc. The number and capability of such objects vary widely in the various simulation software products that are

on the market today. Some software, like *FlexSim*, allows users to easily change the behavior of objects and Advanced Users to create their own objects.

The modeling objects are typically placed and arranged on a layout, or simulation surface, that resembles the phys.cal layout of the system being modeled. The objects are connected in a way that represents the flow of items through the model and permits communication among objects.

All simulation software products have their own user interface that enables users to effectively and efficiently build models, import data, analyze results, etc. While much of the underlying logic is the same across the various simulation software products, the user interface and accessibility to the underlying logic varies extensively. *FlexSim's* interface is quite effective for all levels of users; however, much of its power lies in its openness and ability to modify the objects and their behavior to best match the modeling requirements.

As mentioned above and shown in Figure 4.4, a simulation model is data driven; that is, without data, the model is useless and cannot function. Simulation software, therefore, needs to provide mechanisms to easily import data. Similarly, the reason we build and execute models is to obtain information and to understand the consequences of actions; therefore, simulation software needs to provide mechanisms that easily present or export data and that facilitate the analysis of that data.

Simulation environment terminology

When describing a simulation, certain terms are used to talk about the component pieces. While the specific names may vary in each software application, the concepts and functionality are basically the same.

Time and space

Time and space in the simulation environment are dimensionless until the user decides what the dimensions should be. To change the defaults go to the File / Global Preferences / Environment and select the desired units. *FlexSim* then has to be closed and reopened for the change to take effect; however, it is best to decide on the units prior to building a model and set them based on the following means.

When starting a new model, a screen appears, as shown in Figure 4.5, showing the definition of units. The values are used during some reporting and internal calculations. All data input for the model should be consistent with these units. For example, if the time unit is seconds, the time between arrivals and cycle time for a processor would both have to be expressed in seconds. As a result, statistics that are generated by

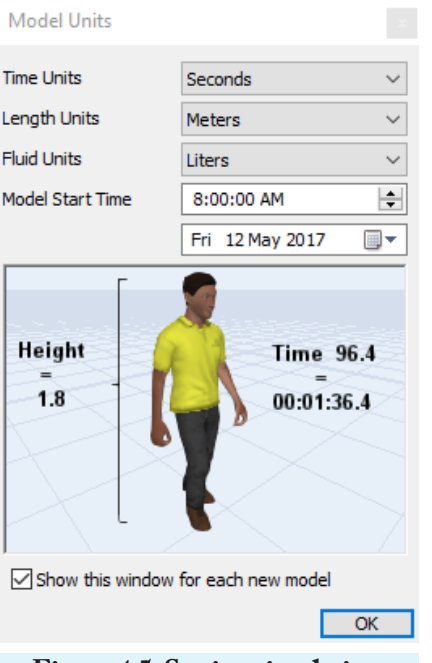


Figure 4.5: Setting simulation units.

the simulation, such as the amount of time a processor is operating, will be interpreted in seconds. Once selected, the units may not be changed until a new model is opened. The units for a model are saved with the model file.

Space is also dimensionless. The default unit of space on the simulation layout screen is one grid unit of size 1x1x1. The user has to decide the actual units that the default grid space represents. Equipment can be scaled appropriately. Movable equipment, such as fork trucks, have speed values assigned to them as the number of grid units traveled per clock unit. Consistent data values are critical to the success of a simulation.

The main screen

Chapter 3 dealt with models that were already built along with the simulation environment to run them. This section starts with a blank page or layout known as the main screen, as shown in Figure 4.6.

As discussed in the previous section, all simulation applications have an environment in which a simulation model is built. That environment contains the surface for building the model (sometimes referred to as a layout), menus for controlling files and functions, and basic units or objects that are used to build the simulation model. *FlexSim*, as with most other simulation software, uses the conventions associated with Microsoft applications (file management, drop-down menus, tool bars, etc.). As a side note, in the student version of *FlexSim*, the watermark "Educational Version" may appear on the layout.

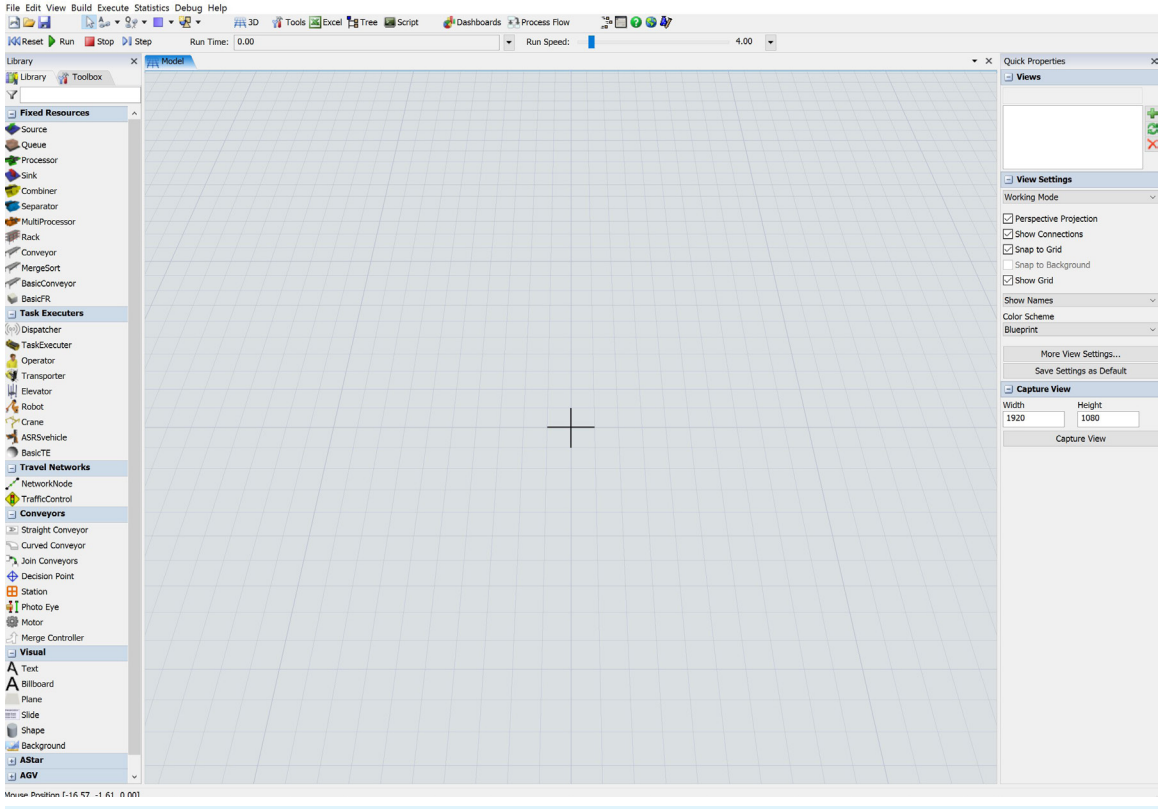


Figure 4.6: Example model with key FlexSim objects.

The main screen is made up of five major areas:

Area 1: Main toolbars

- Top row: Set of Microsoft Windows style drop-down menu options for file control as well as *FlexSim* operating controls
- Lower row: Shortcut buttons that allow quick access to some common *FlexSim* interfaces, operating elements, and model views

Area 2: Simulation control panel

- *Buttons*
  - Reset: Initializes the model
  - Run: Begins execution of the simulation
  - Stop: Stops the model at the end of the current clock cycle (a simulation can be re-started from the stop point)
  - Step: Moves simulation ahead to the next scheduled model event
- *Time controls*
  - Run Time: Displays the current model time in simulation time units
  - Stop Time (drop down arrow): Sets a specific time at which the simulation will be stopped (once stopped, the value can be changed to set a new future stop time)
  - Speed Slider: Defines the number of simulation time units per second of real time; the drop down arrow can set a specific rate

Area 3: Library and Toolbox icon grid

- Library: Objects that can be brought into the simulation model by clicking and holding down the mouse button while dragging the object onto the modeling surface or layout; other libraries can be added; the Toolbox allows data resources such as tables and other functions to be created

Area 4: Model view window

- Surface where the simulation model is built

Area 5: Quick Properties panel

- Displays that can be used to monitor and change data for various simulation objects; the displays will change based on the particular context.

Most applications allow modifications to the basic screen. For the most current details about modifying the simulation surface, check the Help files.

Section 4-3 Simulation components

Remember that graphically-oriented simulation software applications make use of elements, called objects, that are placed on the simulation surface and can be viewed. In *FlexSim*, an

object is the most basic building element block of a simulation and usually represents a physical piece of equipment that performs a specific function.

In *FlexSim* there are two basic types of objects: discrete and fluid (continuous). The discrete objects are used to develop discrete-event simulation models where model behavior results from events that occur at discrete points in time, such as an item arriving into the system or a machine stopping due to an internal failure. Fluid or continuous objects are used to model behavior that results from changes that occur continuously over time, such as the filling of a tank with a liquid. Most of the focus of this book is on discrete-event models and the use of discrete objects. Fluid objects are discussed in a later chapter.

Simulations normally involve actual discrete entities that physically move around in the simulated environment. Examples include a box that is painted by a machine, a case of product that is transported to a warehouse, or a passenger moving through an airport. In *FlexSim*, these entities are called flowitems. Depending on the simulation, these could be boxes, products, customers, paperwork, and so on. Without flowitems, there isn't a need for simulation. Some simulation software packages refer to flowitems as entities or transactions.

In *FlexSim*, flowitems are listed in the Flowitem Bin which is accessed through the Tools menu at the top of the screen. Normally, flowitems are brought into a simulation through the source object where the choice of using a 3D image is listed in a drop-down menu. See the User Manual for more details.

Flowitems interact with objects in a simulation. Objects might perform an operation, create a delay, or move the items. In *FlexSim*, there are two general categories of such discrete objects: fixed resources and mobile resources (task executers)

A simulation model is simply a collection of these objects and flowitems put together in such a manner as to simulate the behavior of a system. The *FlexSim* simulation model shown in Figure 4.7 is one such combination of objects.

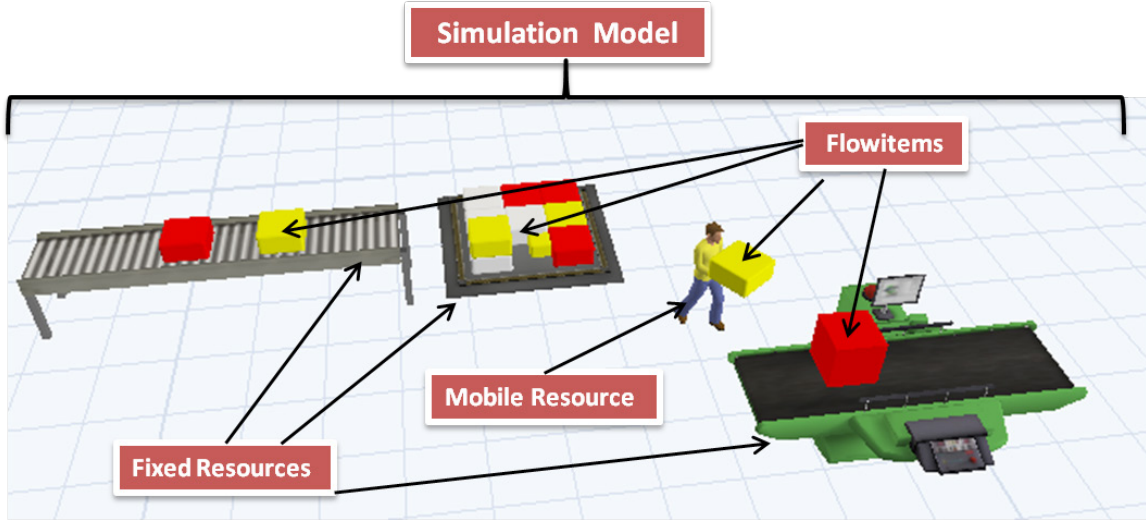


Figure 4.7: Example model.

Section 4-4 Fixed resources

Fixed resources are the objects that send, receive, and perform activities/operations on flowitems. They are also the most common object type. They are referred to as fixed because they are largely stationary. Once they are placed on the model surface, they tend to stay in that place unless manipulated later by the modeler.

Examples of fixed resources include the following:

Source

- Creates and releases flowitems
- Modes of arrival: interarrival time, arrival schedule, arrival sequence

Sink

- Receives and removes flowitems from the simulation model

Queue

- Temporarily stores flowitems when downstream objects cannot accept them
- Can receive multiple flowitems at a time
- Can batch process flowitems
- Receives flowitems until its specified maximum content is reached

Processor

- Processes or forces a delay on a flowitem
- May call operators for setup or process operations
- May incur scheduled or unscheduled downtimes
- Handles one flowitem at a time or multiple flowitems independently (if its maximum content > 1)

MultiProcessor

- Performs a set of operations or processes in sequence
- Operations/processes may have separate times and call separate resources
- Handles one flowitem at a time

Conveyor

- Moves flowitems over a fixed path (not necessarily linear) at a specified speed
- Flowitems enter and leave the conveyor one at a time
- Modes: accumulating, non-accumulating; for accumulating conveyors, if the lead item on a conveyor cannot exit, the items following it will queue; conversely, for non-accumulating conveyors, if the lead item on a conveyor cannot exit, the conveyor and all items on the conveyor stop
- Capacity is limited by number of flowitems or available space on the conveyor (total length of flowitems); spacing between flowitems may be specified

- The library may contain many types of conveyors that allow advanced functions, such as defining when and how flow items leave the object

Combiner

- Groups multiple flowitems
- One input from Port 1, which may be a container to hold flowitems from other ports
  - Can have process times, use resources, and incur scheduled or unscheduled downtimes
  - Modes: join, cannot be separated; pack, container from port 1 holds others, can be separated; and batch, all objects travel as a group

Separator

- Accepts one input at a time
- Can have process times, use resources, and incur scheduled or unscheduled downtimes
- Modes: unpack from packed container; or split, makes copies of a flowitem

Section 4-5 Transporting items

In a *FlexSim* model, the transportation of flowitems is the physical movement from one resource to another. That movement may occur in a number of different ways. By default, fixed resource objects in a model pass flowitems to each other instantly (in zero simulated time). In addition, flowitems may move from one fixed resource to another through the use of an intermediary fixed resource object, such as a conveyor.

Flowitems may also be moved by mobile objects or task executers. These are special objects that have mobility within a model. As the name suggests, they are free to move about within the model executing tasks that have been assigned to them. They may transport flowitems or be used as a resource for a fixed resource.

Examples of task executers that move flowitems are fork trucks, cranes, ambulances, wheelchairs, or people. Task executers that move to fixed resources and perform other tasks include an operator required for some setup, processing, or maintenance operation.

All of the types of fixed resources, task executers, and other objects are shown in the library grid in Figure 4.8. The grid is located on the left side of the main *FlexSim* interface.

Flowitems, as well as all *FlexSim* objects, may carry user-defined data on them while the model runs. The information includes fundamental identity information of the flowitem or operational characteristics of the object as well as dynamic behavioral statistics. That data can be queried and/or set during different events. Future exercises will make use of this capability. This information is as versatile as the modeler desires and can be used to help fixed resources make decisions in the model regarding process times and routing choices. These data storage characteristics are referred to as parameters or labels in *FlexSim* and as attributes in other simulation software.

Bringing objects to the screen

Objects that are used in a simulation model are usually moved from some repository accessed by a tool bar or drop-down menu. Various common conventions are utilized to move objects to the simulation window.

Objects are brought into the model from the *FlexSim* standard libraries or from custom-built libraries. The 3D objects in the library are generic and represent the functionality of the object; the images can be changed, as described in the Help file.

Objects are brought onto the simulation surface (layout) in different ways. The most common way is to click on the object in the library grid then hold the mouse button down and drag it onto to layout surface. You may also click on the object in the library and then just click on the layout when you want it placed. Each click will place another copy of the object. The Escape key ends the placement process for that object. Finally, double clicking on the modeling surface will bring up a menu of objects that can be selected.

Once objects are on the simulation surface they can be moved into position and changed visually in a number of ways. First select an object by left clicking on it. A yellow box will appear around the object indicating that it is highlighted. To move an object around on the surface of the model, click on it with the left mouse button, and drag it to the desired position. You can also move the object up and down in the Z direction using the mouse wheel while holding either the left or right mouse buttons down on the object.

To change the object's size and rotation, click on it. If three colored arrows do not appear as shown in Figure 4.9, go to the edit drop-down menu and click on Resize and Rotate Objects.

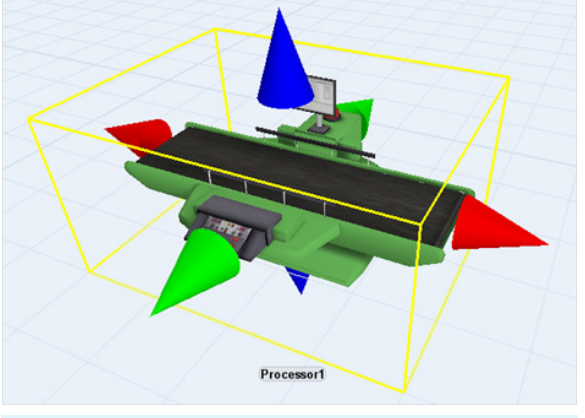


Figure 4.9: Selected Object.

To resize the object, left-click on the desired axis and drag the mouse up or down. To edit the object's rotation, right-click on the arrow corresponding to the desired axis and drag the mouse forward or backward.

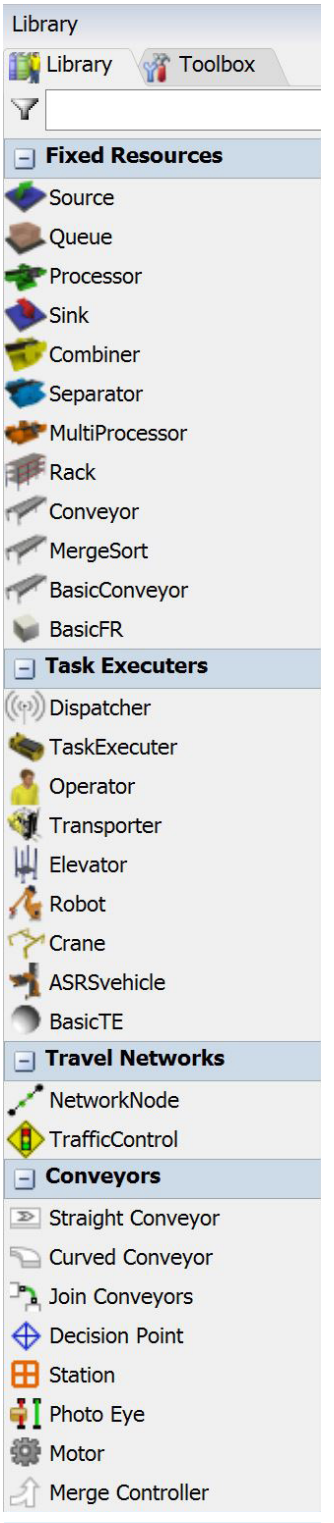


Figure 4.8: Library Grid.

To change the object's position on the axis, move the mouse scroll wheel. To reset the object, right-click on it and under Edit, select Reset Object.

Holding the shift key when left clicking on an object creates a red box around the object. The object is then said to be selected. Advanced operations can be performed on selected objects as indicated in the help file.

Section 4-6 Editing objects

All objects in a simulation application have operational variables that can be modified to characterize their operation. Most graphical applications utilize a double click to open the object. The user interfaces and definition of the variables depend on the particular application. In *FlexSim* all objects have a similar template organization. Open *FlexSim*, select to build a new model, and drag out a processor object to follow along with the text.

The object properties are used to define what the object does and how it behaves (e.g., how long flowitems will stay inside the object, where they will go when they are released, and how the object will appear). Single clicking on the object brings up a summary window called Quick Properties on the right side of the screen. Double left clicking brings up the complete object information in its own window. This view will be used for the following discussion.

Properties window

The first tab, as shown in Figure 4.10, is the default view when an object is selected by double clicking on it. It contains the name of the type of object. At the top of this window is the object's name. As objects are created, they are given default names such as Processor #, where # is the number of objects created since the *FlexSim* application was opened. It is important that each object have a unique name. Objects will have both common tabs and those specific to the particular object.

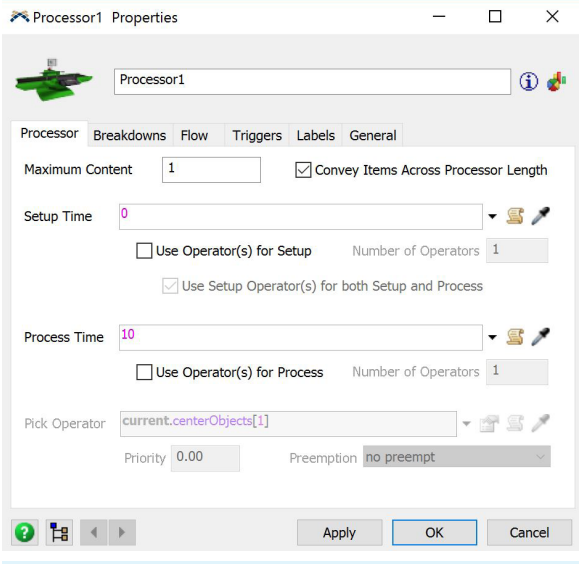


Figure 4.10: Properties Window.

Having a naming convention can help as the number of objects in a simulation grows. One type of naming convention is indicating the type of object as part of the name; for example, pr_Machine1, qu_PartSurge, cv_OutputPath—where pr_, qu_, and cv_ indicate that the associated object is a processor, queue, or conveyor, respectively.

Each object class (i.e., fixed resource, flowitem, task executer) has its own unique set of properties, although all objects will share a similar tabbed layout. The tabs

open other windows that relate to a particular function. All objects have a General and Labels tab and most have Flow, Triggers, and Breakdowns tabs. The common tabs contain the following information:

- General: Attributes that affect the look, sizing, position, and rotation of an object; it also shows a listing of the port connections for the object
- Labels: Table or array of user-defined data or attributes associated with the object
- Flow: Logic for how flowitems move into or out of the object
- Triggers: Optional object functionality to enhance the object's behavior; create and respond to events involving an object
- Breakdowns: Information about the reliability of the object; data are used to set up time between failures and time to repair tables

The Quick Properties Pane on the right side contains one important set of additional information at the top of the window, called Statistics. When opened during a simulation run, the window displays current information about the operation of the objects, including the number of items flowing into and out of the object.

After making changes in the Properties window, or any tab window, clicking Apply saves the changes to the object, but keeps the current window open. Clicking OK saves the changes and closes the window. Clicking Cancel or the X box in the upper right hand corner of the window closes it without saving the changes. Changes can also be made using the Quick Properties Pane; however, there is no Apply button. Just clicking outside the box should set the variable. If both the main object window and Quick Properties Pane are open be sure to check that any changes were set in both places.

Within the object windows, some attributes are represented by a simple text editing field or check boxes while other attributes are represented by a drop-down list that *FlexSim* refers to as picklist options. The picklist itself is a drop-down list of common options associated with this parameter. Details for using the picklist options are contained in the Help files.

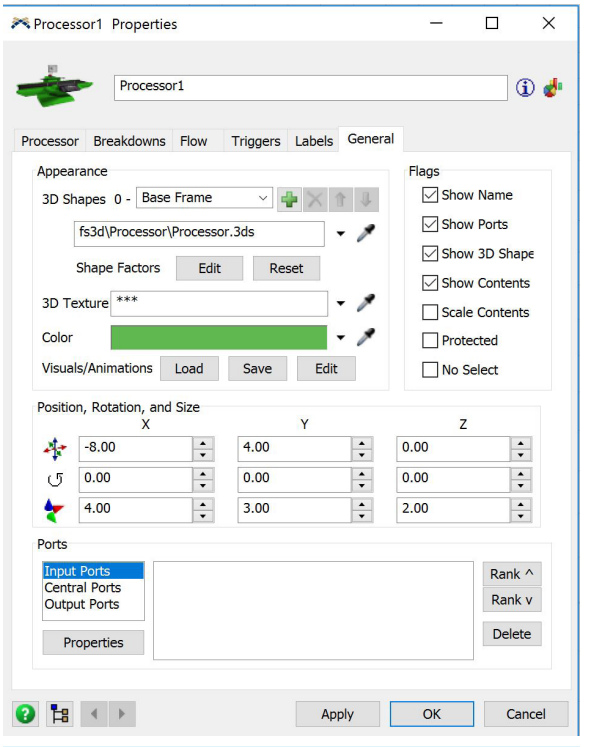


Figure 4.11: General tab in Properties.

General tab

The General tab window contains properties that impact the object's visual appearance, as shown in Figure 4.11.

Items in the top half of the window modify the basic 3D shape of the object (details in the User Manual). Just below

the shape information is the selection window for the object's color. Objects that have been designed with user defined textures can change color. Clicking on the browse icon brings up a color selection window. Colors can be selected directly or by specifying the RGB (Red-Green-Blue) values. When a new color is selected, click OK.

The lower half of the window sets the size, rotation, and position of the object with more precision than using the mouse movements described earlier. The values can be typed in directly or can be changed by using the up and down arrows. Clicking outside the cell after typing applies the change. Moving the windows around so that the object is seen at the same as the General tab window can make it easier to see the effect of any changes.

At the bottom of the General tab window is information about the port connections of the object (port connections are discussed in the next section). Selecting the Input, Output, or Center Ports causes the objects connected to those ports to be listed. Port connections can be deleted by selecting them and then clicking on the Delete button. For some objects the sequence or order of the connections is important. That sequence can be changed by clicking on the connection and using the Rank up and down arrows to change the port order without having to redraw them on the simulation layout.

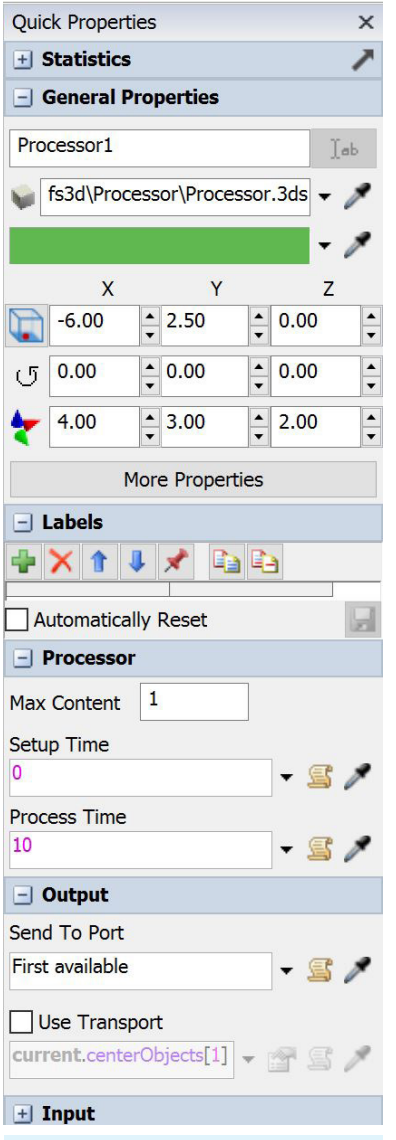


Figure 4.12: Quick Properties pane

Statistics

Operating statistics for an object are shown in the Quick Properties Pane which appears on the right side of the screen and is shown in Figure 4.12. It can also be opened by clicking the icon on the right side of the name field at the top of the parameters tab.

The window provides information about the object that is collected during a simulation. These statistics include the throughput of the object as well as general statistics appropriate for the object.

The Statistics pane also includes information about the state of the object. All operating objects are in some particular state at any point in time. That state is dependent on the type of object. As shown in Figure 4.13, *FlexSim* tracks a large number of possible states during the course of a simulation. Various statistics are developed based on those states. Only a subset of the total number of states may apply to any one object.

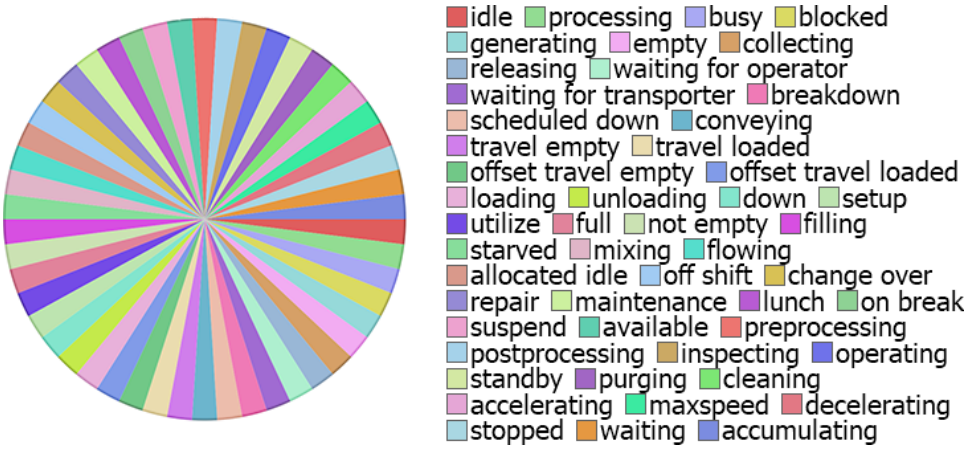


Figure 4.13: FlexSim object states.

The statistics pane gives the current state of the object. Clicking on the Pin icon can place the selected statistic in a separate window called the Dashboard. The pie chart is shown in Figure 4.14.

Depending on the statistics, more than one representation may be shown. The Dashboard can be sized and moved around the window. The statistics can be moved and positioned within the window. The Dashboard can be customized with other information as well, as discussed in the *FlexSim* User Manual.

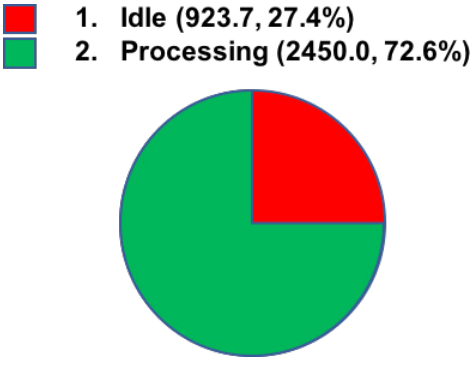


Figure 4.14: State pie chart.

Section 4-7 Making connections

All simulation applications contain a means of indicating how flowitems move through the simulation. The methodology may utilize drop-down menus or a drag- and-connect convention. The connections may, or may not, be visible points on an object. In all cases, correctly following connection protocols is critical to getting the model to function properly.

In *FlexSim*, fixed resources move or route flowitems through models using port connections. These port connections establish the relationships needed between objects to define a flow, as shown in Figure 4.15. Each output-to-input relationship defines a possible route between fixed resources. Every possible routing choice in the model has to have a connection that defines it.

Flow relationships in a model are defined in the direction of flow; that is, from output to input. To create an output port to input port relationship in a model, connect the output port on the sending object to the input port of a receiving object. This connection is accomplished in the direction of flow by

- holding down the A key,
- clicking on the origin or sending object,

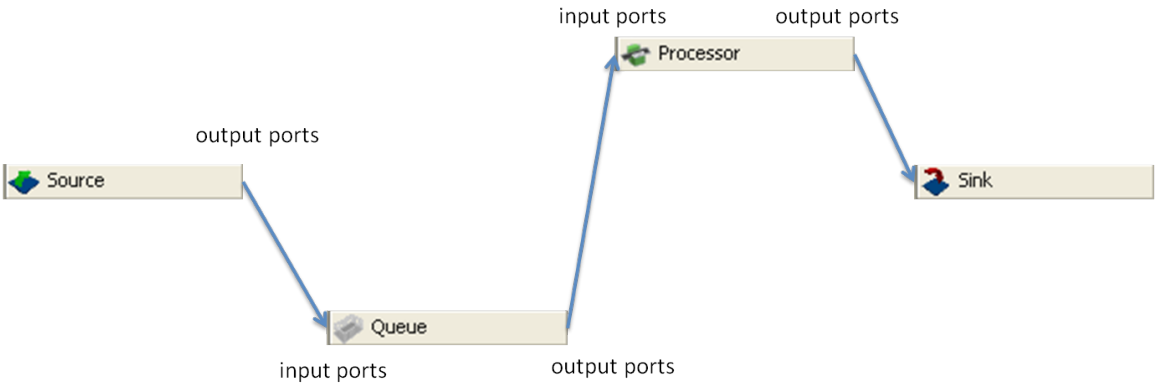


Figure 4.15: Flow connections between objects.

- dragging the curser to the downstream or receiving object.

When the mouse button is released a connection will appear. By default (unless otherwise specified) Connections can also be made using the icons at the top of the screen. Also note that the movement from one object to another requires zero simulated time and that the flowitem goes to the first available downstream object in the numerical order of the ports based on the order of connection.

Another option is available when the object is selected and the A key is held down. If the desired downstream object is not in the model, just click on the modeling surface and release. A palette of fixed resources that can connected to the object will appear. Select the desired object and the object will be added to the model and the connection will be made.

Figure 4.16 shows the input and output ports for a queue object. As a result of the connections made between ports, flowitems will move from the conveyor object to the queue object and then on to one of the two processor objects (to Machine2 via Port 1 or to Machine1 via Port 2).

Find, in Figure 4.16, the connection from the center of the queue object to a task executer, in this case an operator. This connection is called a center-port connection. A center-port relationship allows one object to reference and communicate with another object. A center-port connection is most commonly used to specify a task executer for transporting flowitems between objects or to assist in an object's operation. Center ports are not directional and thus may be connected in any order.

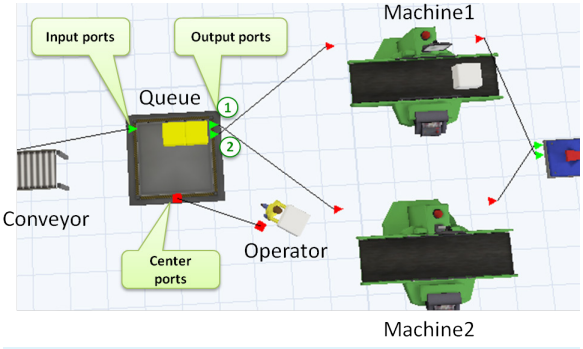


Figure 4.16: Types of ports for fixed resource objects.

In summary, input and output ports direct the path of flowitems through a model; center ports facilitate communication among objects.

The red and green indicators at the ports signify their current condition as open or closed. For example, a processor will have a red input port indicator if it is busy and can't receive a flow item.

Input and output connections are deleted by holding down the Q key and dragging the cursor from the upstream object to the downstream object. Center connections are deleted by dragging between objects while holding down the W key. Connections may also be deleted by going to the General tab on one of the objects involved and deleting it, as discussed earlier.

Objects can have an unlimited number of port connections.

Section 4-8 Moving flowitems

Once connections are created to indicate how flowitems are to move through the simulation, the exact way they move may be specified. In some applications simply making the connection defines the movement. In *FlexSim*, flowitems can move from one fixed resource object to another in many ways: immediately, via conveyor, or by a task executer.

Immediate

When two fixed resources, such as a processor and a queue, are connected together, the flowitems "jump" immediately from one object to the other regardless of the distance between the objects on the modeling surface. The only visualization is the flowitem ready to leave one object and then suddenly appearing in the other.

Conveyors

Conveyors are one of the most common methods for moving items on a manufacturing floor. They can take on many configurations but mainly have two modes of operation: accumulating or non-accumulating. The best example of an accumulating conveyor is one that has rollers. Items enter the conveyor and freely move along its length. If one item stops, the remaining ones continue moving right up to each other.

An example of a non-accumulating conveyor is one that uses a moving belt. Items enter and are carried along the belt, maintaining the same spatial relationship. If the belt stops, all items stop. Conveyors that use buckets or lugs to space out items are also non-accumulating. A moving sidewalk is a combination conveyor where people may stand in a non-accumulating way or walk.

There are many ways of controlling the flow of items on a conveyor, and logic is often added using photo eyes or contacts to direct items along a maze of conveyor connections. *FlexSim* handles the complexity involved with conveyor systems through a series of objects. These objects are optimized for simulations that contain large numbers of conveyors with extensive logical decision points. This very robust functionality can seem confusing at first, especially if the simulation contains only one or two conveyors. Once understood, however, any conveyor can be created easily.

The conveyor objects in *FlexSim* are located in a section within the drag-drop library. All conveyors are made up of a combination of straight or curved sections. A straight section, dragged out of the library is shown in Figure 4.17. When first brought out, the section is just a rectangle with an arrow showing the direction of flow.

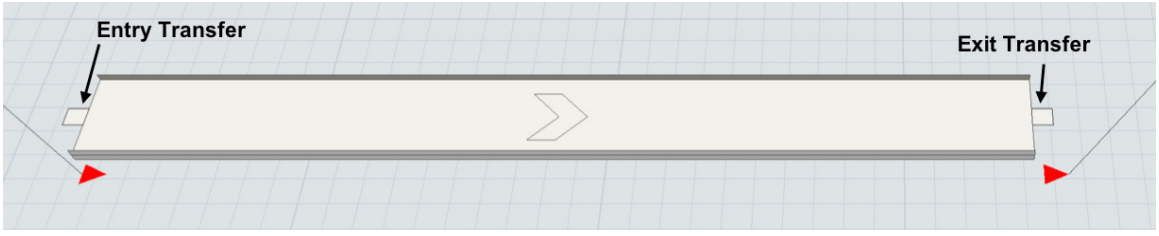


Figure 4.17: Straight conveyor section.

When a connection is made to flow into, or out of the conveyor, transfer points are automatically created. The points are individual objects associated with the conveyor and can be clicked on to open, but contain specialized logic. The entry transfer can be used to pull flowitems or change their orientation on the conveyor. The exit transfer has logic to use a transport or send items to certain objects.

Unlike other objects the conveyor itself only has triggers for reset and messages. The rest of the choices have to do with the physical operation of the conveyor. Opening the conveyor section results in the dialog shown in Figure 4.18.

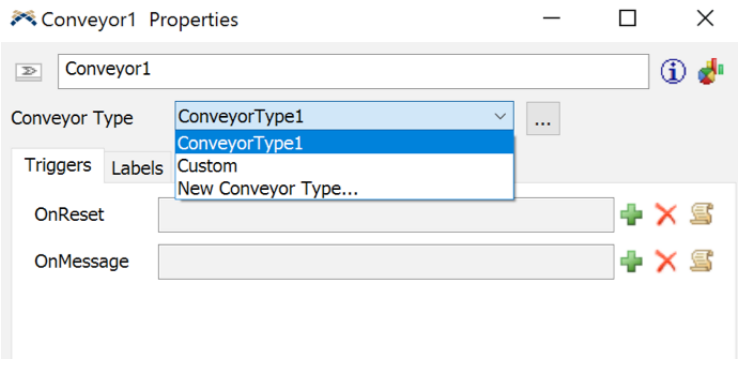


Figure 4.18: Conveyor properties.

In large simulations where many conveyor sections are connected or many similar conveyors are used, being able to configure and make changes to them all at the same time is convenient. Consequently, *FlexSim* conveyor sections are based on defining types. By default, each time a section is brought into the simulation it starts out as *Type 1*. This can be changed as new types are defined by using the dropdown menu. If changes are only desired on a single section, then selecting the type *Custom* makes sense as changes will only affect the selected conveyor section.

Consequently, if a conveyor is made up of a number of sections then they should be of the same type. If there are only a couple of individual conveyor sections being used in separate areas of the simulation they can all be of the same type but modified using the *Custom* option.

To define the type or custom properties of the conveyor click on the box to the right of the Conveyor Type dialog. The first dialog showing the Behavior tab is shown in Figure 4.19.

These properties will be imposed on all conveyors of the same type unless the Custom option is used. In the Behavior tab, the box for an accumulating conveyor is checked by default. The speed and other properties can be set. Notice that units are provided; by default, they reflect the choices made when the simulation was first created but can be changed. However, keeping them consistent with the rest of the simulation is a good idea. The visual

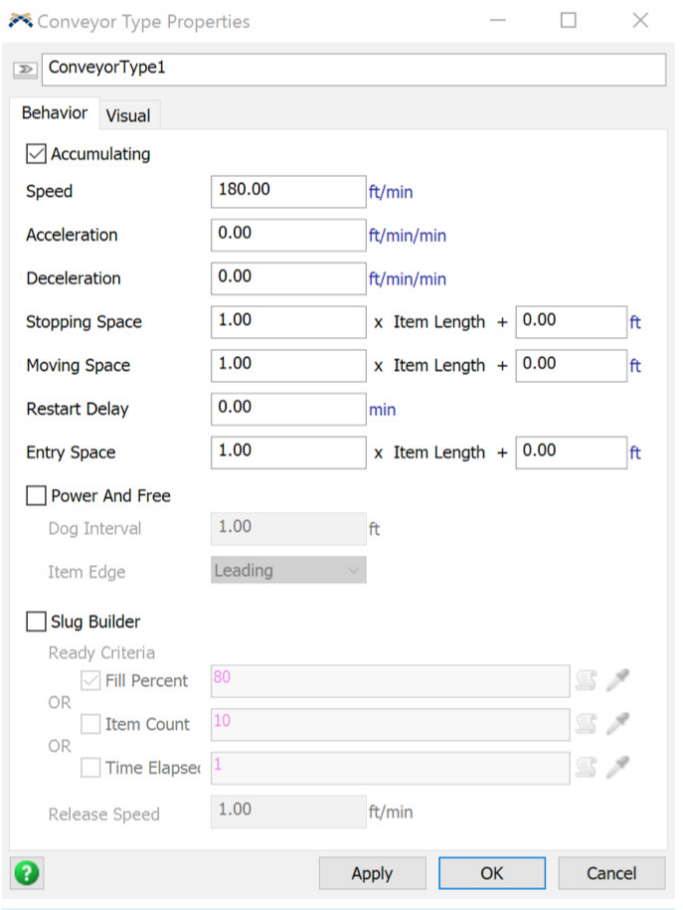


Figure 4.19: Conveyor behavior details.

tab will be discussed shortly. At this time, the only conveyor visual properties that can be changed are the width and color. The *Power And Free* and *Slug Builder* options are used for specialized conveyor systems. Click the **(?)** icon for more details.

While the properties discussed above apply to all the named types, there are some attributes that will only apply to the selected conveyor. These are found in the quick properties pane that appears on the right side of the screen as shown in Figure 4.20. The most important attributes are the Horizontal Length and the Virtual Length. The Virtual Length provides another way to change the time objects are on a conveyor without changing the speed of an entire type. It is also useful when the conveyor is very long and would distort the entire simulation if drawn to scale. When the Virtual Length box is checked, the conveyor will act as if its length was as specified in the box below the Virtual Length check box. When using this function, it's a good idea to check the time on the conveyor visually as the simulation runs to check the values used.

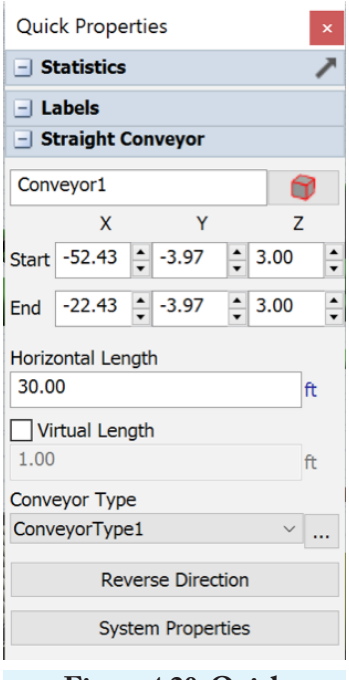


Figure 4.20: Quick properties

Clicking on the Reverse Direction box changes the direction of the conveyor – the arrow on the conveyor will change to indicate the current direction. Clicking on the System Properties will bring up a properties window with two tabs. The Default Types tab indicates the types for each conveyor object when it first appears in the simulation. The General tab has one important check box: Draw Render Mode. This is unchecked by default, but when checked it allows the conveyor visual options to be used as shown in Figure 4.21.

The transition elements are far more powerful than simple entry and exit points. Additional exits from the conveyor along its length can be created by dragging an A connection

line from the point on the conveyor to an object such as a queue.

By default, items on the conveyor will flow from the input to the exit (i.e. the last exit transition) of the conveyor unless directed to by use of decision points described later. Unless decision points are used, a flowitem will go from the input of the conveyor to the last exit transition.

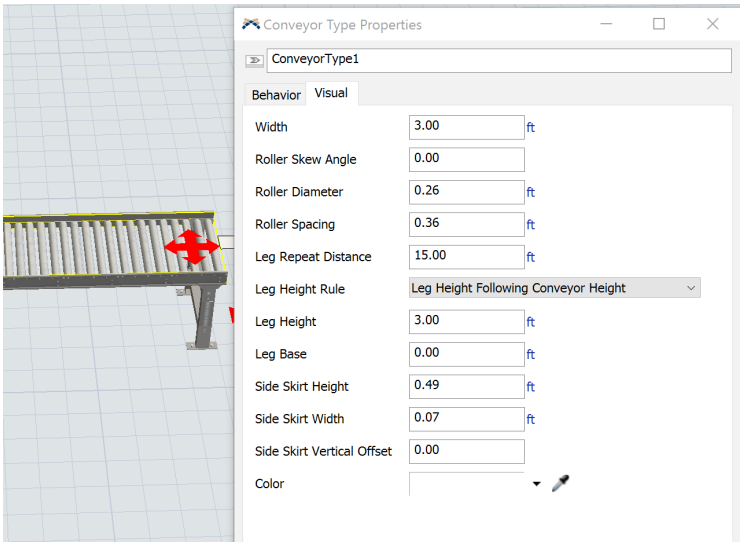


Figure 4.21: Conveyor rendered.

If the flowitem is to go to multiple objects based on the *send to* logic of the last exit, all objects must be connected to that last exit transition.

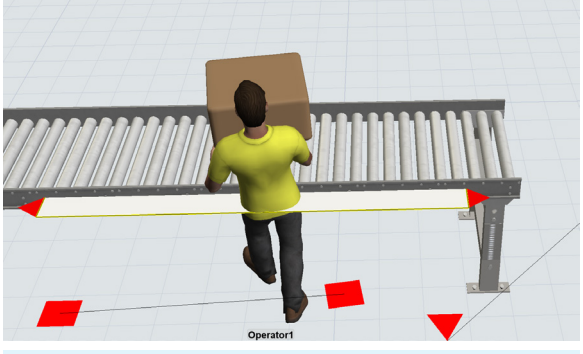


Figure 4.22: Extended exit transfer point.

The transition points can be moved along the conveyor length and even increased in size to cover a range of the conveyor. When using transport to move a flowitem from the conveyor to a downstream object, the transporter will pick the flowitem up at the nearest upstream point on the exit transition as shown in Figure 4.22. Note that the center connection for the transporter is connected to the transfer point, not the conveyor.

Conveyor segments may be connected to form a conveyor system in many ways. Figure 4.23 shows a number of segments connected in various ways. The sections are not rendered so that the connections will be clearer.

When two sections are butted against each other a transition block is automatically inserted. In Figure 4.23, section 2 is placed up against section 1. Section 3 and 4 are placed against sections 1 and 2 at a 90 angle. The curved section 5 is placed at the end of section 3.

Section 6 in the Figure is at an angle against section 1. This was easily accomplished by placing the straight section 6 near section 1 and then first clicking on the Join Conveyors icon in the Conveyor section of the library and then clicking on section 1 and section 6. A transition block with a small curved section is automatically created.

As noted previously, flowitems will want to travel from the beginning of Section 1 straight through to the end of section 2. To make the red (item type 1) box move onto section 3, two decision points are added. They are then connected with an A connection.

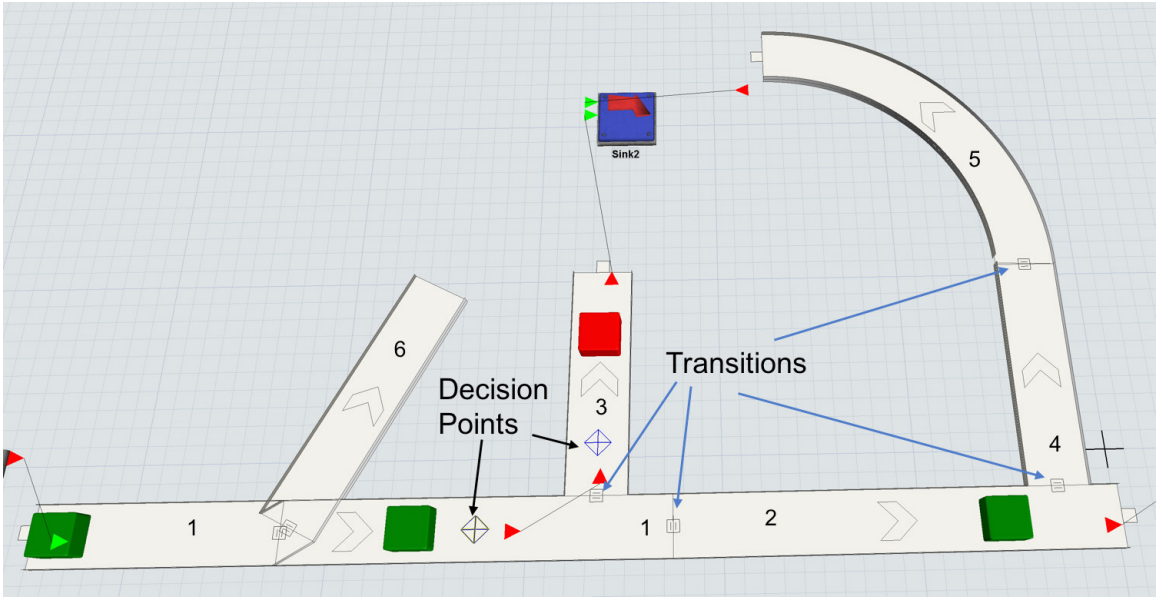


Figure 4.23: Extended exit transfer point.

Double-click on the decision point in section 1, use the On Arrival trigger choice called Send Item, and choose the condition to see if the item type is 1; this will divert type 1 flowitems to conveyor section 3. The Photo eye and Station objects work in a similar manner but contain more advanced choices of what happens when a flowitem comes to them. The dropdown choices for the decision and photo eye objects are somewhat limited and some scripting may be needed for other logical options.

Note that when a conveyor system is put together as in Figure 4.23 the pieces, while logically tied together, are not physically tied together and moving one section will remove it from the system. To move the entire system, hold down the shift key and drag the cursor so that all the conveyor elements are included. They will have a red border on them. When they are selected, they will move as a group. To deselect them, hold down the shift key and click on a blank area of the layout.

Another aspect of conveyors that can cause some confusion is the concept of the maximum content of the conveyor. In the conveyor object the maximum content is determined by a combination of the item size and the settings for the stopping, moving, and entry space shown in Figure 4.19. This is the way conveyors work in actual practice, but for building a simple model it can be confusing.

One way to limit the number of items on a conveyor and still adjust the speed is to use a simple virtual distance that is larger than the number of items. Then set the item length multiplier in the space choices to zero and the additional space to a little more than the virtual length divided by the desired maximum number of items. The speed can be adjusted to make the total travel time correct. Some adjustment in the numbers may be needed.

Any standard logic used by an upstream object such as a queue that depends on the content of a conveyor (e.g. send to conveyor with least number of items) will not work as it is based on finding the next object downstream of the queue. This is because the upstream object is connected to the entry point and not the conveyor. The entry point, as a separate

object, doesn't know anything about items on the conveyor. Such logic can be handled, however, with some customizing of the flow logic.

These building elements in *FlexSim* allow for almost any activities being carried out by conveyors in actual practice to be correctly and accurately simulated. However, to keep things simple, the models needed for the exercises in the first section of the book should only use individual conveyor segments—which should not need customization.

Section 4-9 Creating simple learning models

A good way to become familiar with the operating characteristics of any simulation software application and its objects is to create simple models or learning models. Such a model usually involves a source to create flowitems, conveyors to visualize the movement of flowitems, and any other objects as desired. Keeping the number of objects to a minimum is a good idea, as is using values that make observing results easier. For example, to experiment with the operation of a processor:

1. Drag out a source, a conveyor, a processor, another conveyor, and a sink as shown in Figure 4.24. Open the source object and check the box marked Arrival at time 0. Also, select the By Expression option under the drop-down menu for the inter-arrival times. Use a simple constant value, such as 2, that will make looking at results easier.

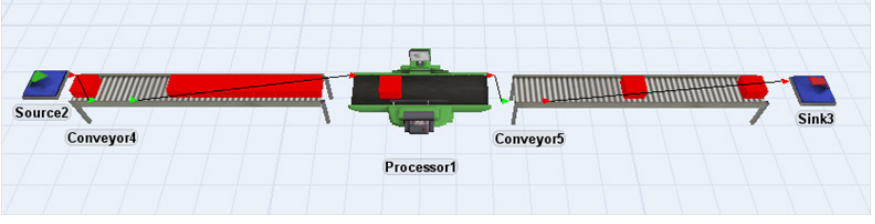


Figure 4.24: Simple model.

2. Open the processor and choose By Expression in the process time drop- down, giving it a value of 1. Again, if the units are minutes, then the processing time is a constant 1 minute.
3. Open the conveyors and set Speed to 1.
4. Run the model and try changing variables.

Create other such simple models that include a minimum number of components and examine their behavior and responses to changes in logic.

Section 4-10 Single-run statistics

Some object statistics are included in the Quick Properties pane and report basic statistical measures as the simulation runs. These basic statistics include:

- Throughput: The number of items entering and leaving an object
- State: Current operating condition (e.g., idle, processing, waiting for transport). Graphical representations are available through the pin icon and dashboard.
- Content: Average, minimum, and maximum number of items in an object
- Staytime: Average, minimum, and maximum time that items spent in an object

Other statistics are obtained through use of the Dashboard. Individual statistics are selected from the Dashboard library and placed in the Dashboard window, as shown in Figure 4.25. See the User Manual for details.

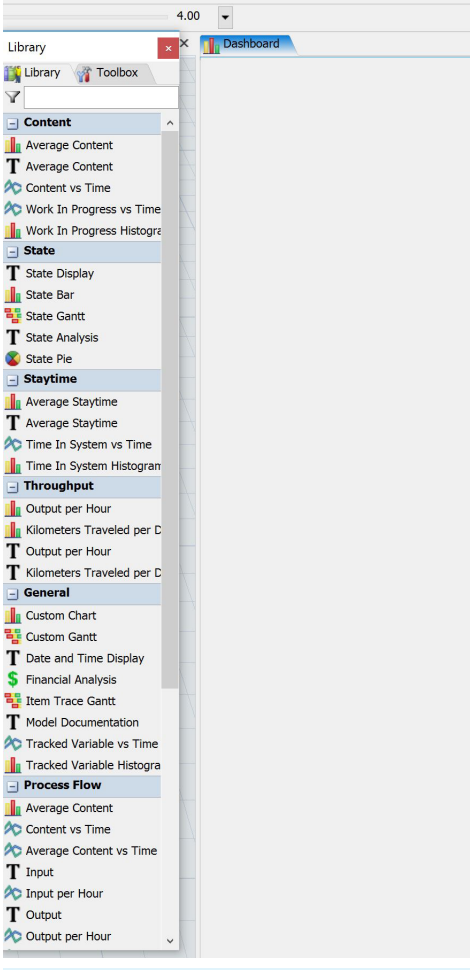


Figure 4.25: Dashboard window.

These general measures are helpful for verifying that a model is performing as expected; however, simulation projects should have specific objectives along with a corresponding set of definitive performance measures that are used to assess a system's performance. In other words, as a system is being designed and analyzed, there are a few key measures that are used to determine whether one configuration is better than another.

These key measures (also referred to as response variables), as shown in Figure 4.26, are the outputs from a simulation model. The outputs are compared to system performance criteria and drive the definition of alternative configurations and the corresponding values of the controllable input variables (also referred to as decision variables).

Figure 4.26 also indicates that if any input to a simulation model is stochastic or probabilistic, then the output will also be stochastic. Most of the time there are several random inputs to a model (e.g., time between arrivals, service times, time between failures, quality indicators). As a result, it is very dangerous to draw any conclusions from a single run of a model. Drawing conclusions from a single run would be analogous to, in basic statistics, making inferences based on a sample size of one! Careful statistical analysis is therefore paramount when using output from a discrete-event simulation model.

In order to draw conclusions and make valid inferences, simulation models must be replicated; that is, the models must be run multiple times. This involves independent random samples from the various statistical distributions that can be used in a model. The infor-

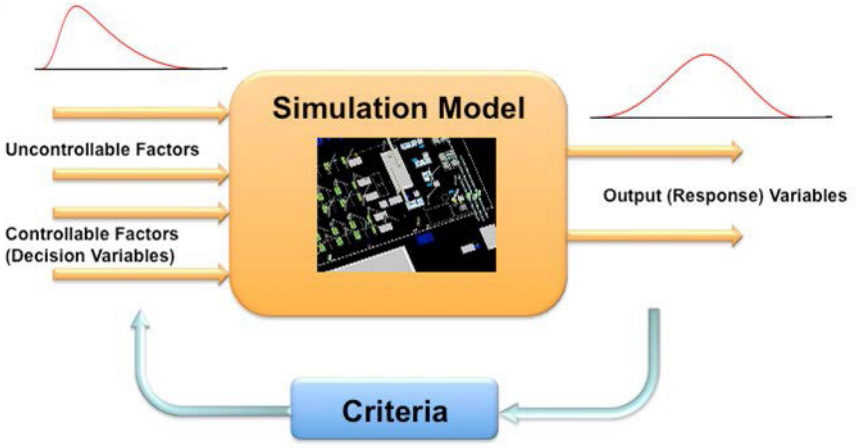


Figure 4.20: Probabilistic model inputs create need for statistical analysis of model outputs.

mation gathered from the multiple runs is combined to provide a basis for conclusions and inference.

Using Simulation Exercises

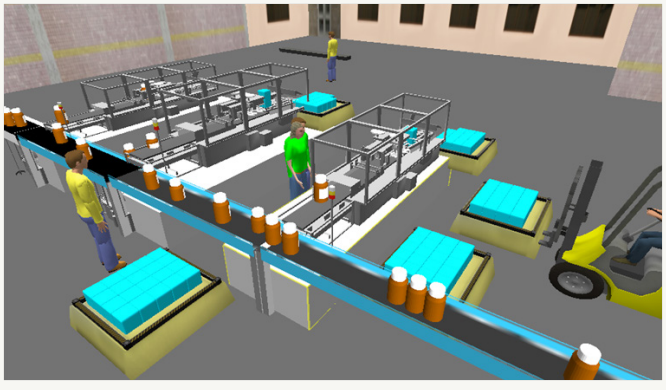
Simulation is an applied technology and has little meaning when used to simply create models without an objective in mind. The belief that if you just build a simulation something good will happen doesn't hold true in practice.

Lessons learned by actually solving problems using simulation are more likely to be remembered. All the exercises in this book focus on the use of simulation to address operational issues or problems. Most of the examples have been taken from actual experiences; however, to maintain anonymity the names have been changed. Each exercise has a similar structure and includes the following sections:

- Background: Provides information about the general nature of the company and the situation that has brought about the particular problem or issue. It also indicates the relationship between the person using simulation as a tool and the owners of the problem.
- Problem statement: Poses a question about operations that needs to be addressed.
- Expected results: States what form the response to the problem statement should take (e.g., a report, analysis, presentation, etc.); the audience expecting the results may also be provided.
- Operating data or additional information: Contains specific information about the system in question; data may be from historical information or previous analysis; as in real life, it may provide more information than is needed for successful completion of the exercise.
- Notes: Provides information on procedures or concepts needed for the exercise; there are always many ways to solve a problem and these steps show one possible way.

Exercise 4-1 Johnson Pharmaceutical

Background



As the largest supplier of over-the-counter medications, Johnson is redesigning its cold remedy line for higher capacity in time for the upcoming flu season. The plan is for the packing line to be completely re-designed. The new plan, conceived by the corporate packaging group, is based on the greedy concept. In that design, boxes of individual doses are taken off the main conveyor line by the first test/wrap machine.

A second test/wrap machine takes off boxes further down the conveyor. The boxes arrive in the packing area at an average rate of 60/min. The wrap machines are rated at an average of 40/min. Based on average rates, the first two machines can handle 133% of the incoming boxes rate; however, a third will also be added as a backup. The plant engineering manager has asked you to validate the design before the plant commits to the proposed production rate.

Problem statement

Validate the new packaging line design.

Operating data

- Time between arrivals of medical boxes: exponentially distributed (0,1,1) with mean of 1 second. The first parameter is the location parameter and is usually 0; the second parameter is the mean; the third parameter identifies the random number stream to use (more on setting this value later).
- Test/Wrap machine cycle times: exponentially distributed (0,1.5,2) with a mean of 1.5 sec.
- Conveyor speed: 1 ft/sec – there is a 4 foot conveyor section between packing machines

Expected results

Comment on the design by running a simulation for an 8-hour shift (28,800 sec). What would you recommend to the plant manager? What rate should the plant commit to?

Modeling and analysis issues

- How should the conveyors be shown? What is the flow logic for them?
- Based just on the average rates what would you expect the capacity of the line to be and why?
- Use single conveyor sections with exit transition at the end going to a processor first then the entry transition for the next section. The logic for the exit transition is to send to next available object.

Exercise 4-2 Lucky Air

Background

Lucky Air, a start-up airline, is committed to providing a shuttle service between the Orange County airport and the Las Vegas airport. With their fleet of regional jets, the company feels that the time is right to bring casino patrons for quick daily visits to Las Vegas. Their promise is to fly as long as there are people who want to travel. If a scheduled plane is full, another will be brought out; their motto is Always a Winner. They expect an increase in business as people try to find money from the slot machines during the downturn in the economy.

The owner wants to set up operations as quickly as possible, so he decides to operate their check-in counter with three ticket agents: one for passengers with e-tickets, another for passengers with paper tickets, and a third for passengers purchasing a ticket. As the only engineer in the new airline, you don't think the level of service will be good and the agents will be working inefficiently. Since you don't want to verbally confront the owner, you decide on showing what might happen with a simulation.

Problem statement

You believe the proposed operation will be inefficient and you want to illustrate the system's behavior to the owner.

Operating data

Estimates of passenger demand, in terms of times between arrivals to the ticketing area and time for an agent to service each passenger type are provided in the table below. Assume the time between arrivals is exponentially distributed while agent service time is normally distributed—all times are in minutes.

Passenger Type	Time Between Arrivals	Service Time
E-ticket mean	exponential(0, 5)	normal(3, 1)
Paper ticket	exponential(0, 10)	normal(8, 3)
Purchase	exponential(0, 15)	normal(12, 3)

Expected results

- Provide the following metrics based on a simulation run of 168 hours:
  - The average and maximum length of the lines for each agent's station
  - The average wait time for each type of customer
  - The average number of customers each station services per hour (throughput)
  - The average utilization for each agent
- Prepare an executive summary based on conclusions drawn from the simulation.

Modeling and analysis issues

- How could the following objects be used to represent the waiting line?
  - a queue—what options would make it look like a line?
  - a conveyor
- Could a processor be used as a conveyor? How? Under what assumptions?
- What are the differences in the four constructs identified above for getting passengers to the agents? Does the choice significantly impact the results?

Chapter 4 Review Questions

1. Identify three nuggets—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the components included in the modeling environment of most simulation software applications.
3. Identify and define the basic elements that comprise a simulation model developed in *FlexSim*.
4. Describe how the movements of flowitems between fixed resources occur within a *FlexSim* model.
5. Describe the difference between fixed resources and task executers (the two basic categories of resources used in *FlexSim*).
6. Discuss how the *FlexSim* object structure enhances model development.
7. Select several *FlexSim* objects; for each one, identify the various states that the object can be in during a simulation.
8. Compare the behavior of accumulating and non-accumulating conveyors.
9. Discuss how a model's behavior differs when a queue or a conveyor are used to move flowitems between two fixed resources.

Chapter 5 Adding Model Logic and Managing Data

The last chapter provided the basics for building discrete-event simulation models. It defined the functionality of some common objects, as well as some of the main properties of those objects. It also illustrated how to bring objects into the model and connect them so that the model represents a system of interest. This chapter explores some of the object properties in more detail and presents means to add logic to a model to enhance its representation of a system's behavior.

Objective

To choose the model elements and logic that best represent the operation of the system and make the appropriate decisions about where and how to populate the simulation with operating data.

In the last chapter, information about an object's behavior was straightforward since each object was designed from the start to work in a particular way (e.g., a processor invoking a planned delay to represent service time). But what if the object's behavior changes based on the circumstances at the time, or if it changes when the value of certain parameters reaches a particular level? The person building the simulation must decide where to best store object information so that it can be used appropriately to modify behavior or logic.

The choices and methods vary by simulation application, although the basic functionality is usually similar. In some cases the application itself may limit the choice; and in other cases, many options may be available, and the decision is left to the model developer. At times the choice can be based on preference or design style, while at other times the choice is dictated by the system being simulated.

Section 5-1 Assigning attributes to objects

Most applications allow attribute information to be maintained on the objects themselves. This construct in *FlexSim* is called a **label**. Attributes, or labels, provide a means to maintain

information about an object or flowitem. Labels are created on objects by opening the Labels tab on the properties window of an object.

Labels can be either numeric or string (they can be arrays as well, but that is an advanced topic). Labels can be referenced from anywhere in a simulation model or at any time during a model's execution. The values can be set manually or changed dynamically. Oftentimes, labels are used to store numeric flags, variables that change dynamically, or indicators of the current state or position of an object or flowitem.

Labels can be used to denote different types of flowitems within the same flowitem class. For example, if the flowitem class represents products, then a label can be used to differentiate small, medium, and large products. Similarly, if the flowitem class represents patients, then a label can be used to denote patients with appointments, walk-ins, emergency cases, etc. A label can be used to set the value of other attributes, such as color and size, enable routings, and affect other types of logic and behavior. A label can be created for a single flowitem when it enters an object or through the Flowitem Bin as detailed in the *FlexSim* User Manual. While any object can have labels, flowitems have a special default label called its **itemtype**. It is a numerical variable that also can be changed dynamically.

Section 5-2 Adding logic

A simulation model is primarily a collection of various types of logic that, when combined, appropriately and reasonably represents the behavior of the system of interest. A large amount of default logic is automatically included in a model when basic objects are added to the simulation view surface and connected to each other. The inclusion of default logic is one of the main features of specialized simulation software. This predefined logic represents common system behaviors and is quickly and easily available to the modeler. Of course, since every system is different, simulation software also provides the means to easily modify the logic to more fully represent the system being modeled.

Most of the remainder of this book presents ways to change and enhance model logic. This section begins that discussion by introducing the use of **triggers** found on an object's Trigger tab. In general, triggers are procedures that cause a specified behavior. They are invoked in response to a certain event or when a condition is met. For example, they are used to take a specified action (e.g., change an object's color or set a label value) when a flowitem enters an object, set a process time when a flowitem is ready to be processed, direct a flowitem to another object based on certain conditions (e.g., the value of the flowitem's itemtype or which downstream object is available), etc. Triggers also may cause events to occur. For example, an on-entry trigger may set the process time to a particular value while a message trigger causes an object to respond to information contained in the message.

Triggers are prevalent throughout *FlexSim*, which provides many built-in trigger options such as set color or set a label value. *FlexSim* also provides the capability to modify pre-built triggers or create completely custom triggers.

Triggers tab

Triggers are action points that execute a set of logic when a particular event occurs, such as a flowitem arriving at, or exiting from, an object. All objects have triggers, but some have more than others, depending on the object's functionality. When a trigger is activated, the logic associated with it fires and is executed.

Most of the triggers associated with an object are contained on that object's Triggers tab. Figure 5.1 shows an example of a Triggers tab. The names, such as OnReset, describe the event when the trigger procedure is executed.

Common trigger points that are available on most objects include the following:

- OnReset: Occurs whenever the simulation model is reset; this is a good place to establish initial conditions for variables and labels.
- OnMessage: Occurs whenever a message is received by the object; messages may come from other objects to coordinate logic or from the same object to create a delay time or initiate logic at some future time.

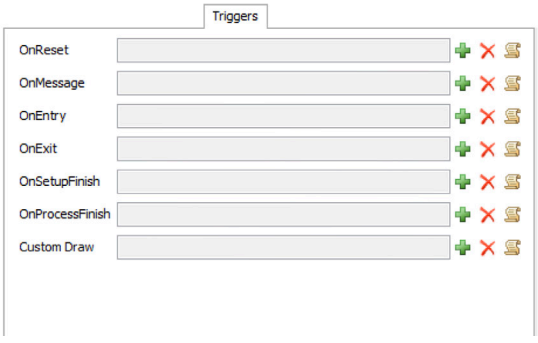


Figure 5.1: Triggers tab.

- OnEntry: Occurs whenever a flowitem enters the object; this is often used to change flowitem label values, color, or itemtype.
- OnExit: Occurs whenever a flowitem leaves the object; this is oftentimes used to update data or take other action before the next flowitem enters the object.
- Custom Draw Code: Occurs each simulation time cycle and allows custom 3D shapes to be added (e.g., changing an object's appearance depending on its current state).

Triggers that relate to processing, collecting, loading, or other functions appear on the appropriate objects. For example, in the processor, there are triggers where logic can be added after the setup action is performed or when the processing time has finished. The desired actions that occur at a trigger event are set by clicking the plus icon (+) and choosing options for each trigger type from the drop-down men.

Processing (or planned delay) time triggers

Processing time, setup time, or a planned delay, is actually set by a trigger that fires when a flowitem first enters a processor. These triggers, however, are found on the Properties window of the object. As shown in Figure 5.2, processing times can be set by expression (e.g., a constant 5 minutes or a calculated value based on object properties), by sampling from a probability distribution, by basing it on label value, etc.

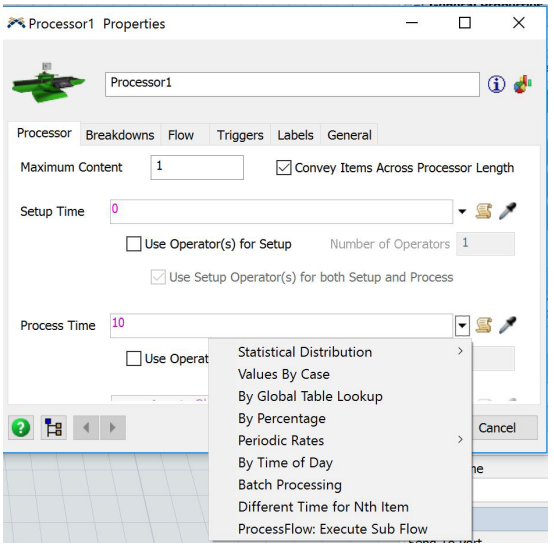


Figure 5.2: Setting Processing times.

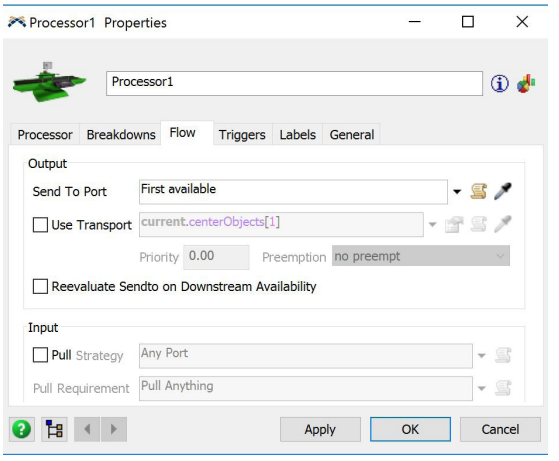


Figure 5.3: Flow tab sections.

Flow triggers

Other triggers are located on the object's Flow tab. As the name indicates, the Flow tab focuses on the movement of the flowitem from the current object to the next object. As shown in Figure 5.3, there are two sections on the Flow tab: Output (which is always active), and Input (which is only active when Pull is selected).

- Output
  - Send to Port: Specifies the logic that determines through which port a flowitem can travel after it leaves the object
  - Use Transport: When selected, calls a task executer to transport the flowitem to the next object; the drop-down menu is used to specify which task executer should be used
- Input: The Pull option is used when items should be pulled into the current object rather than being pushed from the previous object (default method)
  - Pull from Port: Specifies which port or ports should be used for the pull logic
  - Pull requirement: Specifies the logic for pulling a flowitem by some criteria, such as by itemtype or a label value

The example Flow tab in Figure 5.3 shows a processor object that pulls flowitems from preceding objects and, after processing is complete, sends flowitems to the first available downstream object; a transport or mobile resource is used to move the flowitem to the selected object.

The default settings for the Flow tab for objects are as follows:

- Send To Port picklist: First available, open all ports
- Use Transport check box: not selected
- Pull check box: not selected

- Input: pushed from the previous object

For the Output, selecting First Available Port as the Send To Port trigger means that a flowitem will normally exit via Port 1, but if Port 1 is not available—possibly because the downstream object is busy—the flowitem will exit via any other ports that are connected in sequence. By default, no transport is used to move the flowitem to the next object, so the output is pushed directly to the next object. If the downstream object is using a pull logic, it will take precedence. By default, items pulled are selected by the port connections; that is, the object tries to take in a flowitem from Port 1 first, then if none are available, it tries to take in a flowitem through Port 2, etc. Custom logic can also be specified.

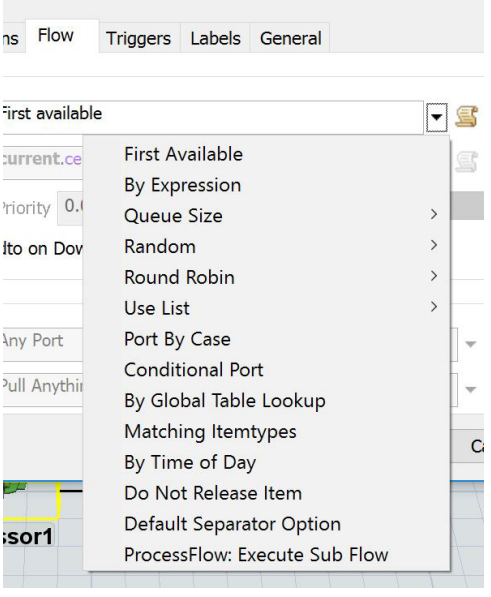


Figure 5.4: Send to Port and Pull flow logic.

The way objects move is related to the logic used in the flow triggers. As shown in Figure 5.4, there are various pre-built options for both Send to Port and Pull Flow logic. For example, some of the ways that flowitems can be sent to downstream objects are based on the shortest queue, in a round robin fashion, randomly, according to a specified probability distribution, based on a flowitem's label or itemtype, etc.

Similarly, flowitems can be pulled from preceding objects based on the longest incoming queue, in a round robin fashion, randomly, according to a specified probability distribution, etc. In addition, custom logic can be developed using the code icon as discussed in a later chapter.

Figure 5.5 summarizes the trigger actions discussed so far, which are available in many objects. In this figure, once an object completes the processing of a flowitem, it decides which port to send the flowitem through by executing the Send To Port trigger. If the Use Transport box is checked, it then executes the logic required to obtain a transporter to move the flowitem to the downstream object. As the flowitem leaves the object, any logic in the OnExit trigger is executed.

The object is then available to process another flowitem from an upstream object.

If the object pulls from an upstream object, the Pull From Port logic determines which port to use and thus which upstream object to pull from. As the flowitem enters the current object, logic in the OnEntry trigger is executed. The object then executes the logic in the Process Time trigger to determine how long to spend processing the flowitem. Finally, an object may get a

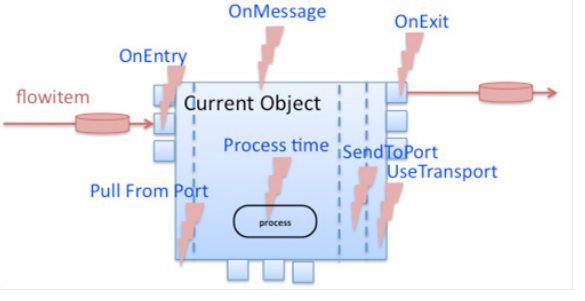


Figure 5.5: Summary of trigger actions.

message from another object at any time; when that occurs, the logic in the OnMessage trigger executes.

Other triggers

There are other types of triggers that can be used to invoke logic. Similar to the processing time trigger discussed above, the time between arrivals in a source is also a trigger. Arrival times are based on simple logic, like a constant (by expression), a sample from a specified probability distribution, etc. An example of modifying the trigger logic on a source object is detailed in the Help files.

Also, events can be created by triggers that change the state of an object when there is a planned downtime on the object (e.g., preventative maintenance, breaks and lunches for operators, shift schedules) or an unplanned downtime on the object (e.g., machine failure or defective parts). These types of triggers and the resulting events are discussed in later sections.

Triggers are also used to create events that are more global in nature. For example, triggers can invoke logic when a model is opened or executed, at different times during experimentation (this type of trigger will be discussed in a later chapter on the topic of experimentation), and through general-purpose triggers called User Events.

Section 5-3 Managing data through tables

Since simulation models are data-driven, managing the model's data is an important issue. One important practice is to remove the data from the internal logic of the model so that it can be easily changed for sensitivity analyses and manipulation by less experienced users. In *FlexSim*, one convenient way to store data, both input and output, is via a Global Table object. These tables can be read from, and written to, during the execution of the simulation model.

Since global tables are needed in the next exercise, one will be setup here as an example. See the User Manual for details.

Global tables are added and edited by selecting the Toolbox option under the View dropdown menu. Choosing the Add option creates a simple single-row, single-column table that can be modified by specifying the number of rows and columns in the appropriate boxes. The table for the next exercise is shown in Figure 5.6 with the basic table structure shown first, followed by the completed table.

Note that editing the header cells (Row 1, Row 2, Column 1, Column 2, etc.) has no impact on the data; these cells are headers for information purposes only. Although optional, they enhance the readability of the model. Copy and paste functions and movement between cells is similar to Microsoft Excel. The table can be moved to Excel by highlighting the table and using the copy and paste commands. Data can be moved into the table from Excel in the same way. Data can also be automatically exchanged between *FlexSim* tables and Excel worksheets.

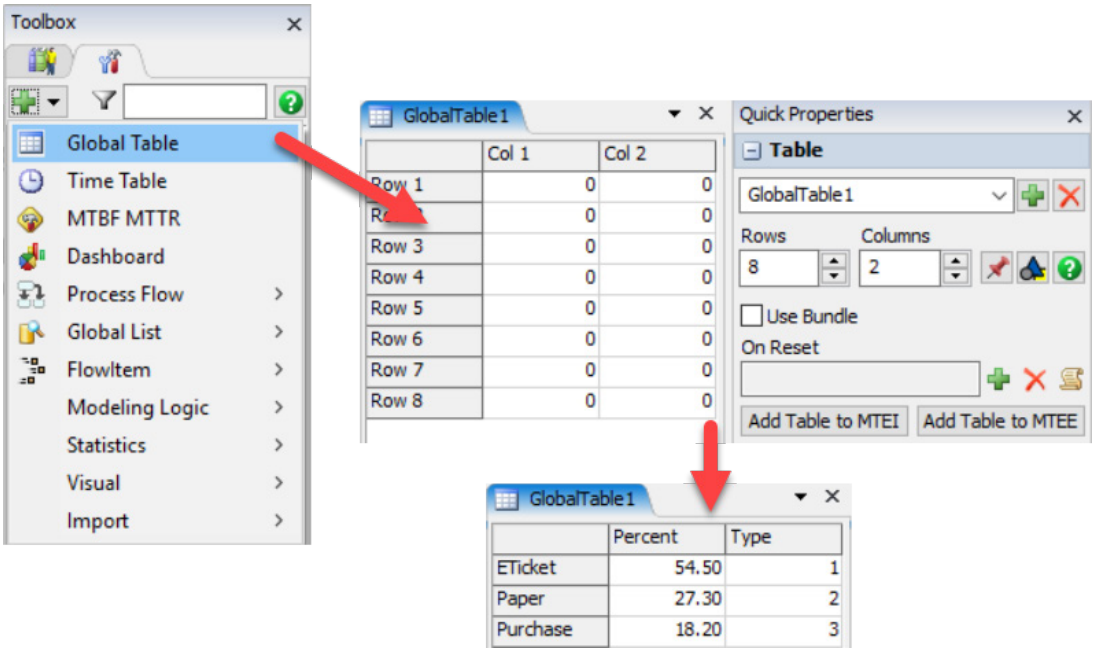


Figure 5.6: Table for Exercise 5-2.

In the example in Figure 5.6, the table is named using the object-naming convention discussed in the previous chapter: the prefix *gt* for Global Table followed by a descriptive name, in this case, PassengerTypes. As with other objects, the table name should be unique for the model.

Section 5-4 Grouping and ungrouping flowitems

There are many situations in operations systems in which the items flowing through the system are grouped or ungrouped. Examples include packing items in a tote for handling or in a box for shipping, unpacking parts from suppliers for assembly, unpacking totes after movement, making multiple copies of an item and distributing them in the system, etc. In *FlexSim*, these operations are handled by the fixed resources known as combiners and separators. Combiners and separators share many of the standard logical attributes of other fixed resources, especially processors, but have some specialized features of their own.

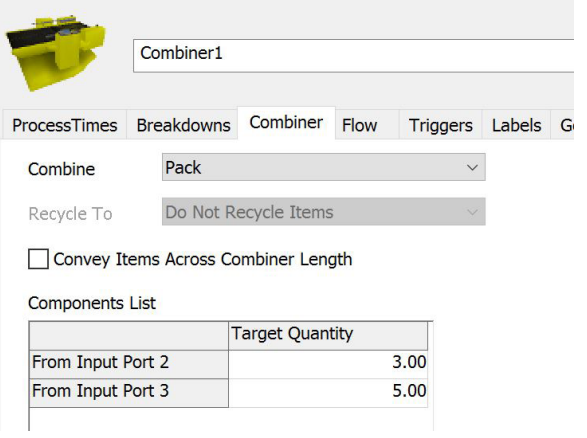


Figure 5.7: Combiner tab.

Combiner

The combiner object, as the name suggests, combines flowitems. A unique tab on the object is the Combiner tab, as shown in Figure 5.7.

The input connected to the first port of the combiner takes on special meaning

depending on the type of combine function that is selected. The item entering from the first port is referred to as the container object and oftentimes represents a pallet, tote, or box. Other flowitems are placed on or in the container and enter through the other input ports.

The following are general characteristics of the combiner:

- A flowitem from Port 1 is required for the combiner to function.
- The target quantity to be combined from Port 1 is always 1.
- Quantities from other ports are specified in the Components List table on the Combiner tab.
- Flowitems from Ports 2 and higher will not be collected until a flowitem enters through Port 1.
- Processing time starts when all flowitems are collected.

Items may be either permanently or temporarily combined. The example in Figure 5.7 will pack (temporarily group) three flowitems from Port 2 and four items from Port 3 in the container that enters the combiner through Port 1.

The picklist options for combining flowitems found on the Combiner tab of the object are summarized below:

Pack

- The flowitem from the first port is designated as the container.
- Flowitems from other ports are physically placed in the container.
- The flowitems and container can be separated by the separator object.
- The process time for the combiner starts as soon as all components are collected.

Join

- The flowitem from the first port is designated as the container.
- Flowitems are collected from other ports and then destroyed.
- The flowitem from Port 1 exits the combiner after the specified number of flowitems is collected and the process time is completed.
- A flowitem that leaves under the Join option cannot be separated.

Batch

- Once a flowitem enters through Port 1, the collection of flowitems from other ports begins.
- When the specified number of flowitems is collected, the process time starts.
- After the process time is complete, all flowitems collected from all ports are released in a single stream.

The various options or modes for combining flowitems are illustrated in Figure 5.8.

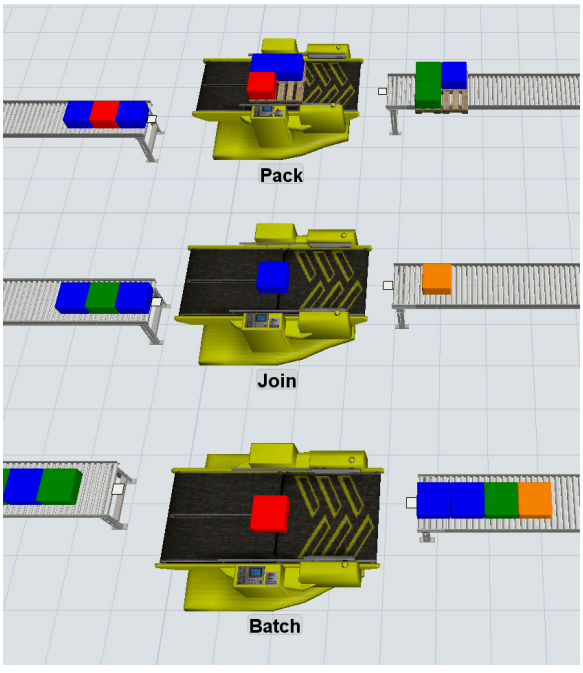


Figure 5.8: Combiner options.

Separator

The complement to the combiner is the separator object. It is primarily used to unpack flowitems that have been packed in a combiner. The separator can also be used to split or make copies of flowitems. The ProcessTimes tab for the separator contains picklist options that specify a Setup Time and a Processing Time. The processing time starts as soon as a flowitem enters the separator.

There are two separating modes that are available on the Separator tab of the object.

- Unpack: The container flowitem is separated from the other flowitems that make up the contents.
  - The default flow logic has the container exiting through the first port of the separator.
  - The Split Quantity has no impact when in Unpack mode.
- Split: The flowitem that enters the separator is split (duplicated) based on the picklist menu entry on the Separator tab. The two separator modes are illustrated in Figure 5.9.

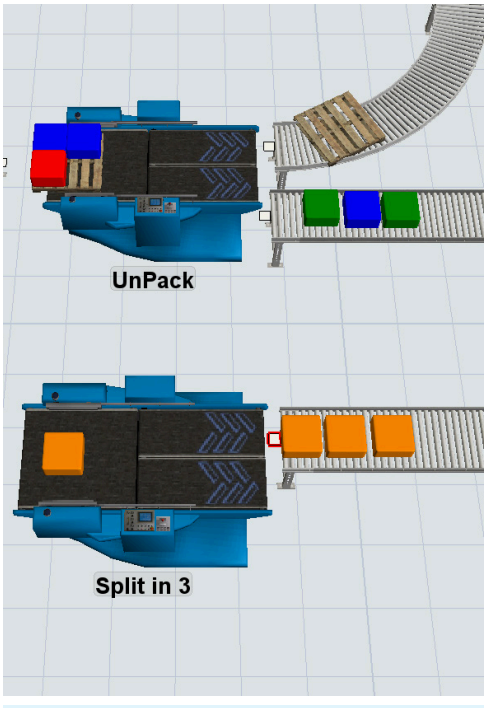


Figure 5.9: Separator modes.

Section 5-5 Reusing custom objects

Users will often create objects with new images or shapes attached to them or include custom logic on the object. Such objects may be desired in other simulations, and significant time can be saved if custom work can be reused. In *FlexSim*, user libraries can be created and used to share many types of objects. Groups of objects can also be saved to a library.

An object can be added to the library by first right clicking on the object and selecting Edit. At the bottom of the secondary window that opens is an option to add the object to the library. There will be a prompt to create a new user library if one hasn't been created. The procedure is shown in Figure 5.10. Note that objects can't be added to the standard *FlexSim* libraries. Right clicking on the library name in the library list allows the library to be saved or renamed. Objects in a user library are brought onto the simulation surface by clicking and dragging.

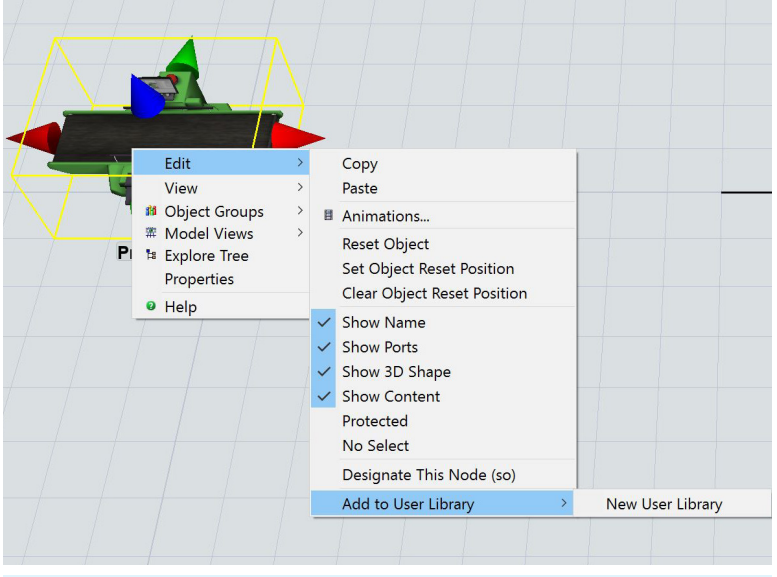


Figure 5.10: Saving in a new library.

Exercise 5-1 More Lucky Air

Background

The president of Lucky Air was not pleased with the information and analysis that was obtained from the last simulation and wanted to try to level the workload for the check-in agents as well as help the customers. He liked what he saw at his local bank, where there was one line of customers that then split to any of the available tellers.

If implemented at Lucky Air, this will mean that the check-in agents all will have to be trained to handle each type of ticket; he wants to know the impact on service if this change is implemented.

Problem statement

Considering a single customer line with three agents who can handle all ticket types, will performance improve for the customers and balance the agents' workload?

Operating data

The operating data are the same as presented in the Lucky Air exercise in the previous chapter but is repeated for convenience.

Passenger Type	Time Between Arrivals	Service Time
E-ticket mean	exponential(0, 5)	normal(3, 1)
Paper ticket	exponential(0, 10)	normal(8, 3)
Purchase	exponential(0, 15)	normal(12, 3)

Expected results

- Use a conveyor to represent the single waiting line; you can change the length of the conveyor legs to make it look like a moving sidewalk.
- Use the simulation model to provide the following metrics based on a simulation run of 168 hours:
  - Average and maximum length of the waiting line
  - Average and maximum passenger wait time in line
  - Average number of customers each station services
  - Average utilization of each agent

Modeling and analysis issues

- How will you identify the passenger types?
- Where should the service time for each type be located? Consider the following options:

    - Service time stored with each passenger
    - Service time stored on the agent
- What triggers on the source should be used?
- Where should passenger data be kept? How would the logic change depending on where the data is located?
- How will connecting the walkway to the agents impact the agents' workload?
- Once the passenger is done being serviced by an agent, route the passenger to one of three sinks to keep track of the number of each type of passenger.

Notes: The passenger item type and a color can be assigned on a source trigger replacing the default value with a number (e.g. 1, 2, 3). The item type can also be used to send passengers to a particular sink.

If the service time will be put on a label, create it on the flow item in the Flow item bin on the Tools library. Then set its value on a source trigger.

If the service time for the agent will be obtained from the flow item use the command: item.name where name is the name for the label in the Processing Time field. If the agent will have all the service times use the drop down choice for Process Time of "Values by Case." Replace the default values for each item type with the expression: *normal(min, std_dev, 1)*.

Exercise 5-2 Even More Lucky Air

Background

Lucky Air has lived up to its name, and many of its patrons have become high rollers at the casinos. The casino operators will provide Lucky Air with a subsidy for each high roller that they bring in.

Consequently, there's a big push to keep the high rollers happy so that they will keep flying on Lucky Air. The head of the airline wants special services for high rollers, especially when they check in.

Problem statement

Prepare a simulation model that will show how the high rollers will be treated if there is a separate priority line for them to be served by the agents.

Operating data

The data are the same as in the previous Lucky Air exercises with the following exceptions:

- It is estimated that 10% of all passengers will be high rollers regardless of the type of ticket they have. Agents will give priority to any passenger in the high-roller waiting line.
- Assume that you didn't know the individual passenger arrival rates but only knew the average total rate (22 per hour) and the percent of each type of customer (54.5% E-Ticket, 27.3% Paper, 18.2% Purchase).

The following arrival schedule table can be set up based on the data provided above. For example, if on the average, E-ticket passengers arrive every 5 minutes, then their mean arrival rate is 12 per hour. Performing this calculation for the remaining customer types results in a total arrival rate to the system of 22 per hour; therefore, the average time between arrivals is 1/22 = 0.045 hr. or 2.7 minutes. On the average, E-ticket passengers comprise 12 of the 22 total passengers per hour, or 54.5%.

Passenger Type	Time between arrivals, minutes	Arrival rate, passengers per hour	Percentage of total
E-ticket	5	12	54.5
Paper	10	6	27.3
Purchase	15	4	18.2
Total	---	22	100.0

Assume the time between arrivals is exponentially distributed. Technically, this is not correct since the variable that is the sum of three different exponentially distributed

random variables is not exponentially distributed; however, for now, this is a reasonable assumption.

Use a single source for the customers and determine the passenger type using the dempircal function.

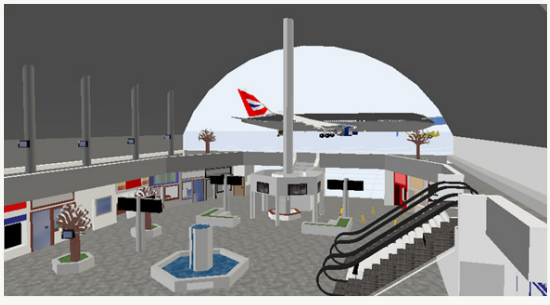
Expected results

Use the simulation to provide the following metrics based on a simulation run of 168 hours:

- Average and maximum length of the waiting lines for regular and high-roller passengers.
- Average passenger wait time.
- Average number of customers each station services.
- Average utilization of each agent.
- What trigger and which drop-down command should be used to set the itemtype?
    - With the itemtype placed on the item, where is the service time logic?

Exercise 5-3 Hampton International

Background



In an effort to become the most advanced airport in the world, Hampton International decided to automate its luggage handling facilities. After lengthy delays, the system was put into operation but was immediately unable to provide the anticipated throughput. Passengers still had to wait for a long time for their luggage, and some aircraft were kept waiting. The problem appears to be in the transportation of luggage from the aircraft to the distribution network.

Once in the distribution system, the bags are conveyed according to information on the luggage tags. The design consultant did not believe in simulating the system during the design phase and has subsequently been fired. The facility manager wants to simulate the operation to identify the problems.

Problem statement

Prepare and use a simulation of the luggage transfer system from the aircraft to the distribution center to identify problems in the system and test options to remedy the problems. Explore the 8-hour period of high volume.

Operating data

As shown in Figure 5.11, under the new system, luggage is brought from the aircraft to a loading station where a series of transporters convey the luggage to the distribution point. Before reaching the distribution station, each transporter is inspected by Homeland Security. Once at the distribution point, the luggage is sent to its destination while the transporter is returned to the loading station.

The basic operating parameters for the system are as follows:

- On the average, luggage arrives at the transporter loading area from the aircraft about every 5 seconds (distributed exponentially).
- The loading area has space enough for 100 bags to wait to be loaded onto transporters.
- Eight pieces of luggage are put on a transporter. It takes about 20 seconds to fill the transporter as soon as all the pieces of luggage are present.
- The transporter brings the luggage to the inspection station; it covers the distance of 1200 feet in 1.5 minutes.

- The inspection station has 4 inspectors. Inspecting a transporter takes an average of 120 seconds; assume the times are exponentially distributed.
- After inspection, the transporter takes 1 minute to move the bags to the unloading station.

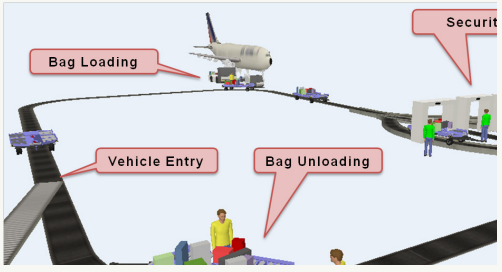


Figure 5.11: New baggage system.

- The unloading process takes 45 seconds, after which the transporter returns 1500 feet to the loading station travelling at a speed of 12.5 feet/ second.
- Transporters enter the luggage loop at the start of each day on an entry track from their storage facility. The initial plan was for 15 transporters.

Use a pallet to represent the transporters on the route. AGV objects could be used with more advanced logic.

Expected results

- How often are bags backed up outside the loading area?
- What should be the expected capacity of the bag retrieval system, in delivered bags per minute?
- Recommend what can be done to improve operations (e.g., more transports, more inspectors, faster transfers).

Modeling and analysis issues

- How detailed do you need to make the simulation? Can you represent all inspection stations with one processor?
- What time unit best serves this simulation?
- Think of how to easily set the travel time on the conveyors.
- What flowitem can be used to represent the transporters?
- How can you load a set of transporters into the model at the start?
- What combiner mode should be used to load bags on the transporter?
- How should the output ports of the separator be connected?
- What command can you use to give the bags more visibility?

Chapter 5 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important-in the chapter.
2. Describe the different ways that attributes can be assigned and stored on a flowitem.
3. Define trigger. Select four triggers, each on a different object. For each trigger, explain when it will occur, describe one action that it can perform, and provide an example of how it would be used in a simulation.
4. Explain the action that occurs when the default Send to Port flow trigger on an object is executed.
5. Indicate the actions taken by the flow trigger options "round robin" and "round robin, if available."
6. Describe a situation in which each of the following options could be used; they are options available on the Send to Port flow trigger: Shortest queue, Random, Round robin, By Percentage, Conditional.
7. Describe a situation where the "pull" option on the Flow tab may be used.
8. At what point in a Processor object's activities is the decision made to send a flow item out through a particular port?
9. Define the steps that are required to build a global table. Describe how data might be exchanged with Microsoft Excel.
10. Describe the advantages and disadvantages of storing information on an object or flowitem or in a data table.
11. Explain the different functions that a combiner object can perform. Provide three operations system examples that would be modeled by a combiner object.
12. Describe the function of the flowitem that enters through the first input port of a combiner object.
13. Explain the different functions that a separator object can perform. Provide three operations system examples that would be modeled by a Separator object.
14. In the Unpack option of the separator, in what order do flow items leave the object?
15. In what ways can a user library be useful?

Chapter 6 Improving System Representation

This chapter extends modeling capability in several key areas. Mobile resources are one of the main classes of objects in any simulation software. In *FlexSim*, mobile resources are referred to as task executers. These are especially versatile and powerful objects. The first section of the chapter describes the various types of task executers, including their basic operation and the main properties that drive their behavior.

Objective

To utilize additional functionality available in simulation applications that enhance modeling and enable help others to better visualize, understand, and have confidence in the simulation results.

Often simulations may focus on simulating activities that have no direct visualization. While standard objects contain a wide variety of logical choices in the drop down menus, trying to make use of them can be difficult. *FlexSim*, in their 2016 version of the software, has added new capabilities to handle logical expressions without compromising the value of good visualizations. In this chapter, the use of Lists is introduced. Additional use of the advanced logic capability is discussed in Chapter 14.

In order to enhance visualization, additional information oftentimes needs to be displayed on the simulation surface. This information may include statistics, state status, descriptions, etc. Sections 6-3 and 6-4 describe how this information can be displayed in *FlexSim*.

Section 6-1 Mobile resource objects (task executers)

In addition to the objects previously described to move flowitems, simulations often require mobile resources to move or carry items. These resources can include forklifts, automated guided vehicles (AGVs), or operators. Simulations may also require mobile resources to carry out certain tasks, such as setting up or repairing a piece of equipment. The mobile resource objects, also referred to as task executers in *FlexSim*, can perform various functions:

- Physically move over the simulation surface

- Carry or transport flowitems between objects
- Execute a series of tasks
- Be assigned to an object in association with a particular operation (e.g., process setup, processing, maintenance)
- Distribute tasks to other task executers
- Follow a path created by network nodes

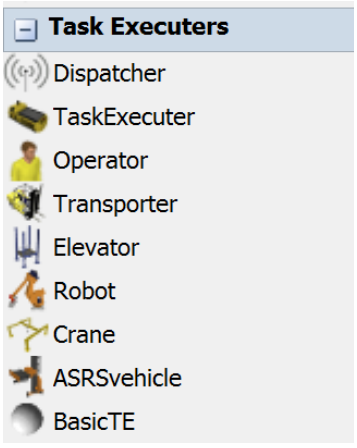


Figure 6.1: Task executers.

As shown in Figure 6.1, *FlexSim* offers a variety of types of task executers in the standard Discrete Objects library. In terms of their basic functionality, the objects are very similar. Note that there is a task executer object within the general class of task executers.

The task executer object and operator objects are very similar. The main difference is that the operator can be animated to represent walking and carrying an item. The transporter object, analogous to a fork truck, can raise flowitems vertically using its forks, as well as travel the surface. An elevator object only moves flowitems vertically, or in the Z direction. A robot moves flowitems between two points on the model surface from a fixed position using its arm (the base of the robot object does not actually move). Cranes and ASRS vehicles are more complex task executers. Most beginners use the basic task executers such as operators and transporters; however, it is helpful to be aware of what is available for more complex modeling projects. All task executers can travel along the model in three dimensions if attached to a path network.

As their name suggests, the primary function of task executer objects is to execute task sequences. A task sequence contains a series of specified tasks that are executed in sequence, has a priority value that indicates its importance to other task sequences, and can preempt or halt a task sequence that is being performed. Task sequences can be created automatically (e.g., when a fixed resource "uses transport"). In the case of Use Transport, the following tasks are executed:

1. Travel to the object that contains the item to be moved
2. Load the item from the calling fixed object to the task executer
3. Break; that is, check to see if there is some other task waiting (e.g., load another available item if the task executer has capacity)
4. Travel to the destination object
5. Unload the item from the task executer to the receiving fixed object

Another automatically generated task sequence occurs when assistance is needed to perform a process. In that case the task sequence is

1. Travel to the object that needs assistance with a process;
2. Utilize the task executer for the duration of the process.

One powerful feature of *FlexSim* is that the modeler can construct his or her own custom task sequences. This is an advanced topic; for more information, refer to the *FlexSim* User Manual. Task executer objects execute a task sequence if there is currently no active task sequence or if the incoming task sequence has a higher priority. Otherwise, the task sequence is dispatched to an associated task executer or it is queued for execution until the current task sequence is completed.

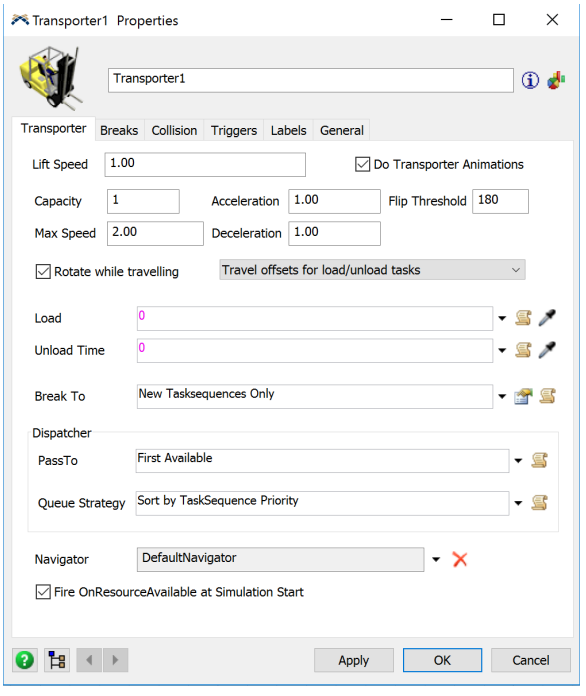


Figure 6.2: Task executer's parameters.

As shown in Figure 6.2, the main interface window for task executers is very different from other objects. The one shown is for a transporter type of task executer. All task executer objects have two main sections, one pertaining to operating properties of the object and the other addressing task executer behavior.

The first section of the interface is used to set values concerning the operation of the task executer; the variables vary somewhat depending on the role that the task executer performs.

Task executers that are being used to convey flowitems have the following built-in attributes:

- Capacity: This indicates the maximum number of flowitems that can be carried or transported at a time.
- Load/Unload time: This attribute gives the time, in simulation units, to load a flowitem at the originating object and the time to unload it at the destination object.
- Lift Speed: This indicates the speed in grid units per time unit.
- Travel offsets for load/unload tasks: If this option is not selected, the task executer will go to the origin of each object to load or unload flowitems. If selected, the task executer will go to the origin first and then move to the exact point where the flowitem is to be loaded or unloaded.
- Flip Threshold: When the angle between the transporter and the destination meets or exceeds this value, the transporter will "flip" to a mirror image in order to be facing the correct direction. This is for advanced visualization effects only.
- Attributes when being used for any task are as follows:
- Rotate while Traveling: If checked, the transporter will rotate as needed in order to orient itself in the direction of travel. This is for visualization effects only.
- Speeds (Max speed, Acceleration, Deceleration): The travel speed is in grid units per time unit and grid units per time unit squared, respectively
    - The acceleration controls how fast a task executer reaches its maximum speed or how fast it slows down when it reaches its destination. Unless a very precise

simulation is being conducted, it is advised to set the accelerations to 0 so that the task executer immediately achieves maximum speed or immediately stops at the destination. Alternatively, the acceleration and deceleration can be set to about one-half of the speed; this is, again, so that the object doesn't spend a lot of time accelerating or decelerating.

- The speed should be set to a reasonable value. For example, the typical walking speed of humans is 3 miles per hour (264 feet per minute, 4.4 feet per second, 75 meters per minute, etc.). Similarly, fork trucks travel 6 to 15 mph loaded and about 12 mph unloaded. Of course, these values should reflect the real system's operating environment and the grid size of the model surface and time unit of the model.
- Example: Assume a task executer has a maximum speed of 250 feet per minute and starts and ends at rest (speed = 0). Consider accelerations of 0, 75, 125, and 250 ft/min2. Using the basic laws of uniformly accelerated linear motion,

$$v = a\Delta t$$

$$d = \tfrac{1}{2}a\,(\Delta t)^2$$

where v = speed at the end of Δt, a = acceleration, and d = distance traveled.

The following table shows the effect of different acceleration/deceleration values on the time, in minutes, for the task executer to travel two distances, 100 feet and 1,000 feet. Note: setting the acceleration/deceleration to 0 really means infinite acceleration/deceleration; that is, the object instantly achieves maximum speed from rest or instantly comes to rest from maximum speed.

acceleration/ deceleration	d=100 ft.	d=1,000 ft.
0	0.40	4.00
75	1.63	7.33
125	1.26	6.00
250	0.89	5.00

The second section of the interface deals with task behavior and involves advanced topics. Its options are mentioned here but are not explained (consult the *FlexSim* User Manual for more information).

- Break To: Logic that describes tasks that can interrupt the current task
- Dispatcher Pass to: Strategy for passing tasks along to other task executers. If more tasks are required than available task executers, the tasks are queued up.
- Queue Strategy: Logic for prioritizing tasks in the object's task sequence queue

Task executers can share tasks; for example, operators can be assigned to respond to the needs of multiple pieces of equipment. To share tasks, the executers can be connected together. They can also be connected to a dispatcher object that assigns tasks based on priorities or other logic. Details for calling and connecting task executers can be found in the *FlexSim* User Manual.

Network nodes

Task executers often are required to travel complex paths, such as the paths that may be found when fork lift trucks travel between rows of a warehouse. In *FlexSim*, network node objects are used to provide such specific paths for task executers to follow as they move around the model surface. The primary interface tab for a network node is shown in Figure 6.3.

Establishing a network for task executers consists of three steps:

1. Connect network nodes to each other to form travel paths.
2. Connect network nodes to the objects that request services from task executers.
3. Connect task executers to the network.

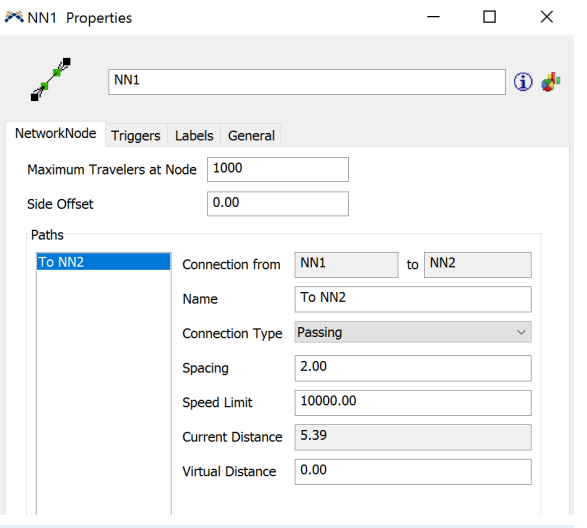


Figure 6.3: Network node interface tab.

The time required for a task executer to travel between two network nodes is the distance between the nodes divided by the speed of the task executer (assuming no acceleration or deceleration). The distance between the nodes is given in terms of the grid units. In Figure 6.3, network nodes 1 and 2 (NN1 and NN2) are 8.54 grid units apart. If the task executer travels at a speed of 20 grid units per minute, then the task executer will take 0.427 minutes to travel between the two points. A convenient feature of network nodes in *FlexSim* is that a virtual distance between nodes can be specified. For example, if the virtual distance from NN1 to NN2 is set to 100 (grid units), then the task executer would take 5 minutes to traverse the path between the nodes (100 grid units / 20 grid units per minute). Since network nodes are bidirectional, virtual distances need to be specified in both directions, from 1 to 2 and from 2 to 1.

A* Algorithm for controlling mobile resources

FlexSim offers an alternative to the use of network nodes and paths for controlling the travel of mobile resources. This alternative, implemented through the AStar module, uses an algorithm called A* that finds a good path between two points while avoiding obstacles. To illustrate the use of A*, consider the simple model in Figure 6.4. In this case, an operator

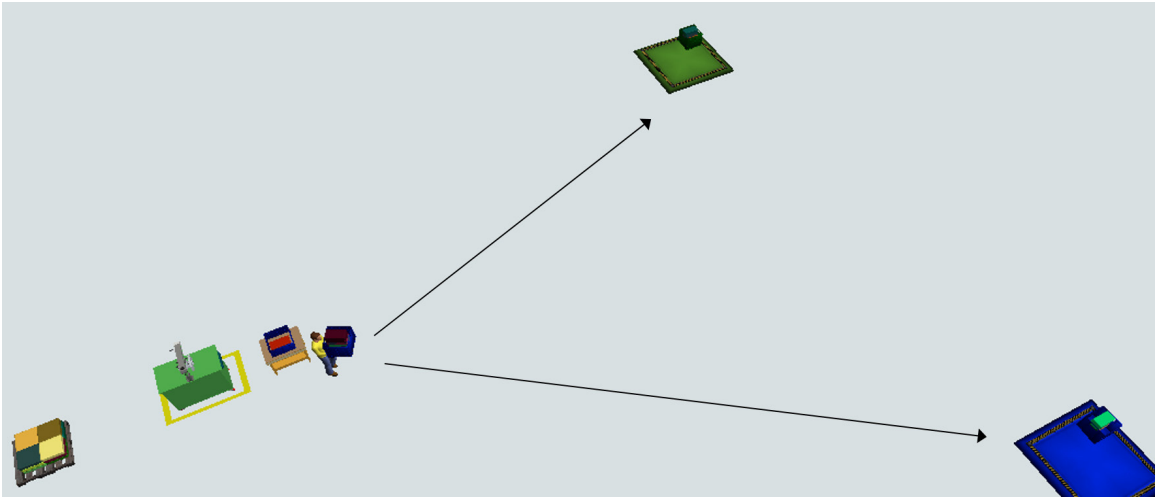


Figure 6.4: Simple path for operator transporting items.

cuts incoming material, places it in a tote, and delivers a full tote to one of two locations (the green and blue queues). By default, the operator takes the shortest path from the cutting area to the delivery points, as shown by the arrows in the figure.

However, there are typically obstructions in these straight-line paths that the operator would need to avoid. By default, the operator would ignore any obstacles and proceed through any objects in the straight-line path. A common way to control the operator movements is through a path network that is defined by placing and connecting network nodes in the model that define the desired path and requiring the operator to follow the path.

Now consider the same model, but with some obstructions, in Figure 6.5. The cutting area is bounded above by a wall and storage room that impedes travel to the green tote area; it

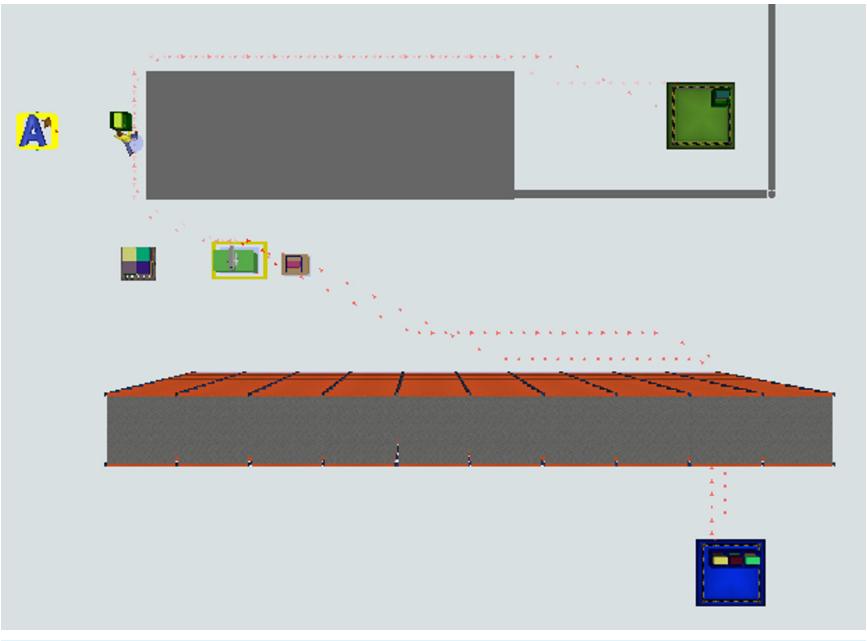


Figure 6.5: Travel paths defined through the A* algorithm.

is bounded below by a rack that impedes travel to the blue tote area. Network nodes could be used to define a path around these obstacles. An alternative approach would be to use the A* algorithm. The operator travel paths defined by A* are indicated by the small red arrows in Figure 6.5.

A* is a module in *FlexSim* that is found in the basic library (in a section named "AStar"), as shown in Figure 6.6. To utilize the A* algorithm, drag out the AStarNavigator anywhere in the model. In Figure 6.5, the navigator is highlighted by the yellow box on the left side of the figure. Use A connections between the navigator and the task executers to be controlled by A*. Also use A connections between the navigator and any modeling objects (e.g., the rack in Figure 6.5) that the mobile resources are to avoid. The connections are not indicated by lines but by a blue area at the base of the object (see the base of the operator in Figure 6.5). The gray box (representing a storage area to be avoided) and gray wall in in Figure 6.5 are A* objects – Barrier and Divider, respectively. The A* objects do not need to be connected to the navigator.

While, the use of the AStar objects requires more processing than network nodes, they are useful in large models with many barriers that require many nodes to define complex paths. The use of AStar is also useful when objects are to be moved around in a model, e.g., when considering alternative layouts, thus not requiring the replacement of network nodes and redefining paths.

For a more information of using the AStar module in *FlexSim*, see the FlexSim User Manual. For a detailed discussion of pathfinding and the A* algorithm in general, see Patel [1].

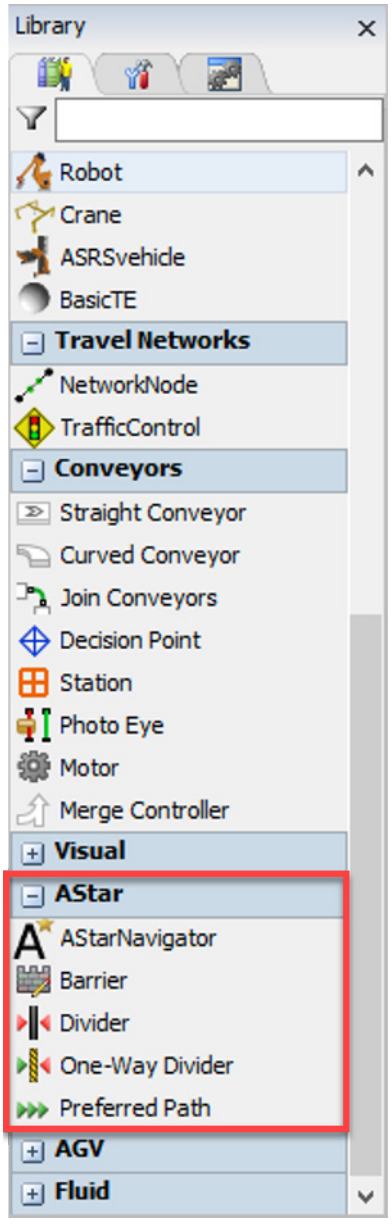


Figure 6.6: AStar objects in the basic library.

Section 6-2 Using Lists for decision making

FlexSim already has an extensive variety of standard logical functions that can be selected using drop down menus to direct the movement of flowitems from one object to another. The FlexScript language (similar to C++) is used to customize the standard logic choices even further. While the combination of standard functions and scripting can handle almost any situation, the use of scripting can be difficult and time consuming. To address this, *FlexSim* has introduced a new programming paradigm that makes it easier and more efficient to include advanced logic. The method is discussed in more detail in Chapter 14.

This section introduces one element of the new functionality called **Lists**.

Lists

A List is a virtual repository for items that can be sorted and individually chosen using Structured Query Language (SQL) statements. SQL is commonly used in relational database management systems. A good example is the problem of trying to decide where flowitems should be sent. Single objective decisions such as "first available" or "by item type" can easily be used with the standard flow drop-down menu if the decisions are simple. With complex decision criteria, however, custom coding may be required and several connections made between objects to assure that all possibilities are accounted for.

Instead of being pushed from one object to another along a fixed connector, an item is pushed to a List which keeps track of the item, along with a set of attributes. The action is instantaneous and not visible on the simulation surface. A receiving object will immediately search the List, based on a set of to pull an item.

The following example not only shows the use of the List function but contains other methods for handling data and carrying out the simulation.

Case Study: Dough vs Cheese

Each year, at the State Fair in the Midwest, Doug set up his stand and served fried dough. Doug's Dough was extremely popular and successful. Last year he faced stiff competition when a food truck selling fried cheese curds pulled up next to him. He lost customers when they found they could get their food faster at the other truck. Doug is sure he needs to increase his frying capacity but not if the investment is worth it. A simulation might help.

When the business turned down, Doug started watching his customers and determined a number of things about their actions. Since there is so much to do at the State Fair, customers did not like to wait and would turn away if the line was too long or they waited too long. They would also jump to the cheese line if that one seemed shorter.

Here is what Doug found about the customers (all times in seconds):

- Customer arrivals: every 22 seconds, distributed exponentially; 60% percent prefer dough and 40% prefer cheese
- Maximum line size customers would tolerate: duniform function from 4 to 8
- Maximum wait time in line: duniform function from 95 to 180
- IF in line > 90 sec: 40% would switch and 60% would stay
- Number of pieces purchased by a customer: 1 piece, 50%; 2 pieces, 35%; 3 pieces, 5%

Information about the operation:

- Fryer: 3 pieces per batch: setup time 20 sec; fry time 90 sec
- Time to complete a sale: 30 sec
- Cheese sale transaction time: 60 sec

From the description of the problem, it is evident that this simulation would require many logical statements to represent the number of random choices that the people being simulated would have to make. While such logic could be carried out using the standard

choices discussed previously in this book, the List function would simplify the effort. The following approach to the simulation is only one of many possible methods.

The customers are the obvious choice to be the flowitems in the simulation. Since most of the decision elements are based on an individual customer, each customer will have a set of choice attributes. Labels on the customers is the best place to store the attributes. The values of the labels are assigned when the customer enters the model. The following labels are on each customer:

- MaxLine – The size of the line the customer will not join
- MaxWait – The most time a customer will stand in a line
- WillSwitch – A flag (1 or 0) indicating their willingness to switch lines

The basic physical elements include queues for the dough and cheese lines, processors for the cheese and dough trucks, a source for the fried dough, a queue for the basket of dough pieces, and a combiner that will match the customer with the correct number of dough pieces. Conveyors provide walkways to allow dough customers to exit or change lines. They conveyors, as well as the cheese line have queues before them to hold the decision logic. These basic elements are shown in Figure 6.7.

Begin creating the decision logic by setting up a List table. To build the List, use the Tools dropdown, click on the (+) icon, and select to add a Global List/ item list. A list properties window will appear with some preset entries. Name the list "customers." There are two types of fields in the list. An expression field that calculates some value. If it is a value that may change while a flowitem/element is in the list, then the box labeled Dynamic should be checked. A Label field contains the value of the indicated label of the element in

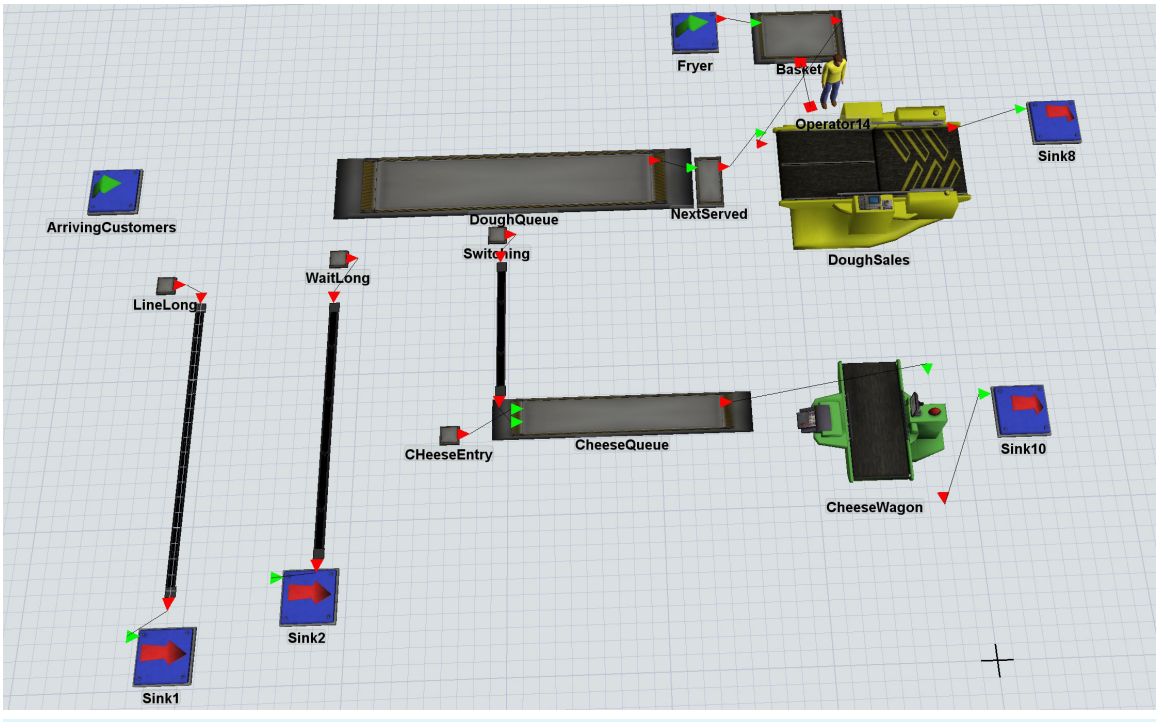


Figure 6.7: Basic elements for the Dough example.

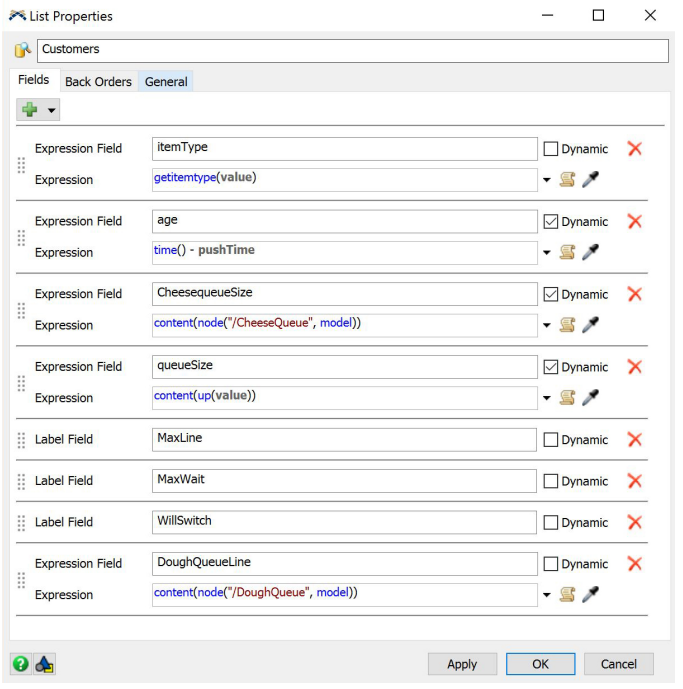


Figure 6.8: Label fields for customers.

the list. Extraneous entries can be deleted. The list needed is shown in Figure 6.8.

The decision fields include how long the item has been in the list as well as the sizes of the cheese and dough queue lines at any point in time. The queuesize field is the size of The queue that the table refers to. The code to get these values will be clear in the chapter on scripting.

The logic to simulate the movement of customers starts by opening the customer source and the Flow tab. From the dropdown menu for Send To Port, choose Use List > Push to Item List. The name of the List table will appear in the dialog box. This will put all customers generated by the source into a virtual List table from which other objects can call them.

Open the dough queue, and in the Flow tab, check the box to pull. Select Use List > Pull from Item List in the Pull Strategy drop down menu. A dialog box appears with the name of the List. In the Query drop down menu select the + icon then Where/itemtype. An SQL statement appears to create the statement WHERE itemtype = 1. Clicking on the + icon followed by Where/DoughQueueLine adds another SQL statement. Changing that to read AND DoughQueueLine < MaxLine creates the full query that will result in a customer of itemtype being pulled into the dough queue as long as the queue size is less than the customers maximum allowed. The Remove from List on Success box should be checked as the person is no longer in the List but in the dough queue. All queries are handled in a similar way. This dialog box is shown in Figure 6.9. See the *FlexSim* User Manual for details and a list of the SQL commands.

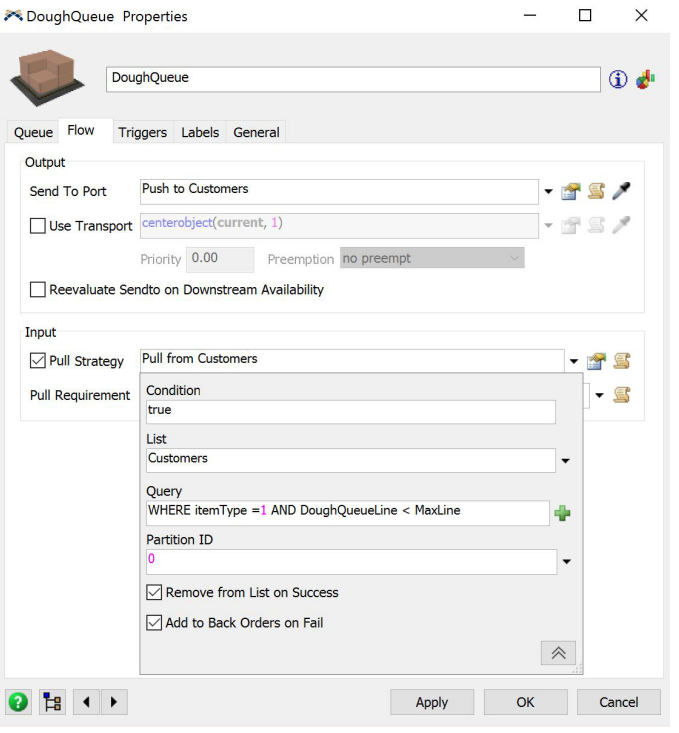


Figure 6.9: Query example.

The query for the cheese line is entered in the pull area of the Flow trigger of the small queue added before the cheese queue. In this case the SQL command is simply

```
WHERE itemType = 2
```

Customers who leave because the dough line is too long are identified by the query in the pull statement on the Line2Long walkway conveyor that contains the SQL command

```
WHERE itemType = 1 AND DoughQueueLine >= MaxLine
```

The previous queries were used to select customers who had just entered the simulation. The customers item list is the repository for all customers who enter the model but it doesn't exist physically. Since customers immediately go to one of the pulling objects, there should never be any customers in the base customer list.

The remaining logic involves those who are in the dough queue itself. A new list table is needed to represent these customers. While customers in this queue are physically shown in the Dough queue, they and their information actually exists in a label table. Rather than creating a new list with the same attributes as the exiting one, *FlexSim* allows such a new list to exist as a separate partition of the currently defined list.

To create this new list, the Send to Port of the dough queue Flow tab is set to push to Customers list. In this case, however, the Partition field at the bottom is changed from the default of 0 to 1. Logically, when an item enters the Dough queue, the goto port trigger files and the customer data is entered into the same customer list but in partition 1. All the attributes in the list now refer to items in the Dough queue.

Customers who get tired of waiting in line will leave through the query in waiting walkway with this query—set to partition 1:

WHERE age >= MaxWait

Customers who are next to be served in the dough line are called by the logic on the pull query for the small queue added just before the combiner using the command that takes the person waiting longest first:

ORDER BY age DESC

Finally, customers who are willing to switch lines are found by using the pull logic in the walkway to the cheese queue with the command

WHERE WillSwitch =1 AND CheesequeueSize < queueSize AND age > 90

Note that all the queries for the last three objects have the partition set to 1.

With the logic in place, the model can run showing customers leaving the dough line because it is too long or their wait is too long, or switching to the cheese line as shown in Figure 6.10. In the figure, 3D objects were obtained from Google 3D Warehouse in place of the cheese processor, the dough combiner and the fryer equipment.

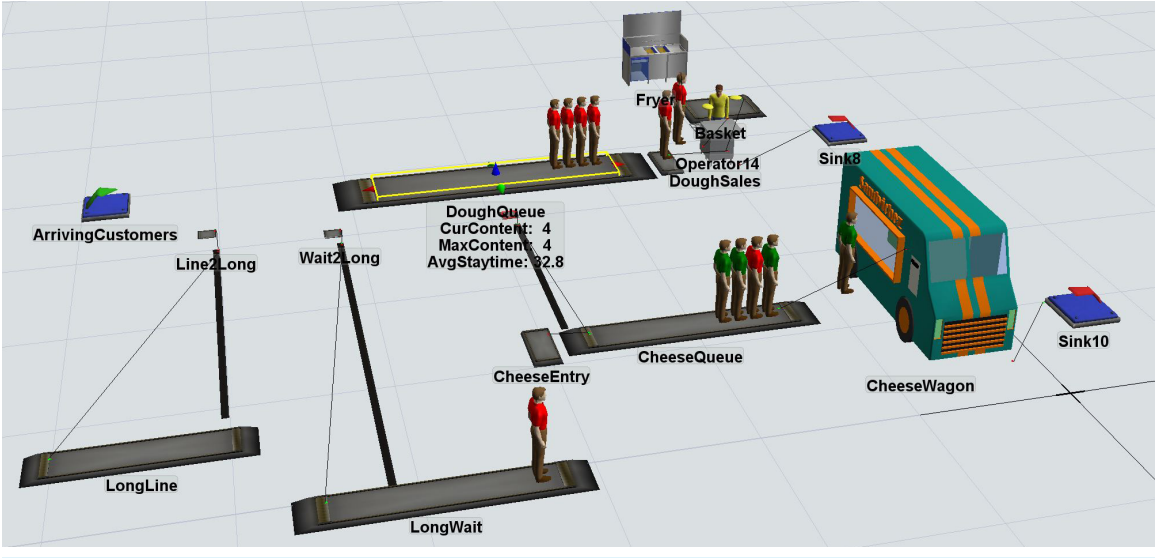


Figure 6.10: Dough and cheese operations.

During a run, the dynamic content of the List can be viewed by going to the Tool display in the Library pane and double clicking on the created list. In the General tab of the list, selecting the view entries button brings up the current entries in the list. Both partitions of the list are shown, but there aren't entries for Partition 0 since these customers came from the source and are immediately taken by one of the queries. An example of the Customer list for the simulation is shown in Figure 6.11.

Lists can also be used to manage mobile elements such as operators. In this case a new Global List/Task Sequence List is used in the fixed resource in need of a transport; check the Use Transport box in the flow trigger and from the dropdown menu select List/Push

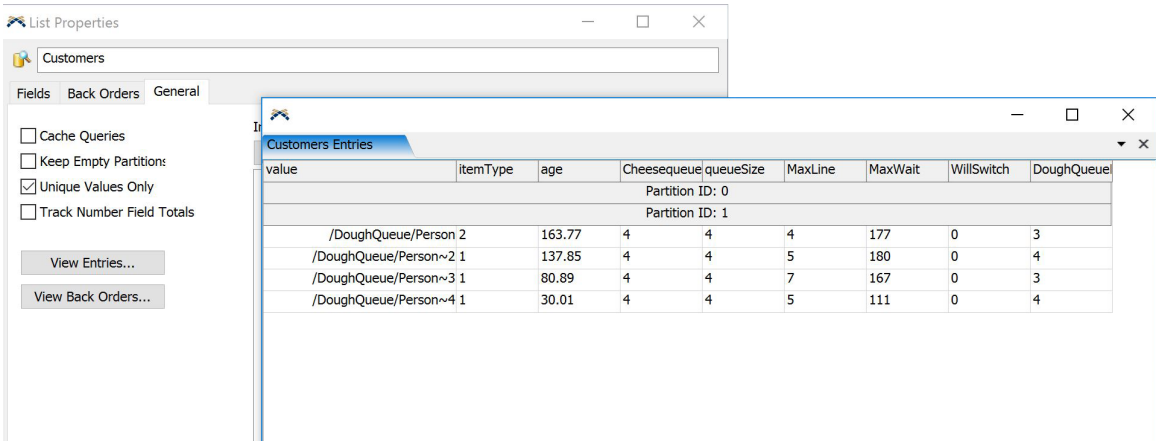


Figure 6.11: Customer list dynamic view.

to List. On the mobile resource go to the OnResourceAvailable trigger and choose to use List/Pull from List.

Section 6-3 Displaying information on the screen

While all the objects and flowitems add visualization to a simulation, there is often the need to add more content to the screen. In *FlexSim*, adding screen elements is accomplished by the visual tool object and the dashboard. These objects perform no direct actions in the simulation but provide a means of reporting information and decorating the model space.

Visual Tool

The main window for the visual tool is used for configuration. Possible representations, selected using the Visual Display picklist menu found on the Display tab, include the following:

- Plane, Cube, Column, Sphere: Basic shapes that can be decorated with imported picture files as texture
- Imported Shape: A 3D graphic that is added to the simulation surface
- Text: Displays a single line of text including
  - basic information such as text or simulation clock time;
  - information obtained from an object as selected in the visual tool
- Presentation Slide: Displays multiple lines of text

The visual tool in Figure 6.12 displays the output of a source, thus denoting the number of items entering the model. The information desired is selected through the visual tool dropdown menu.

Visual characteristics such as size and color are modified on the General tab of the visual tool.

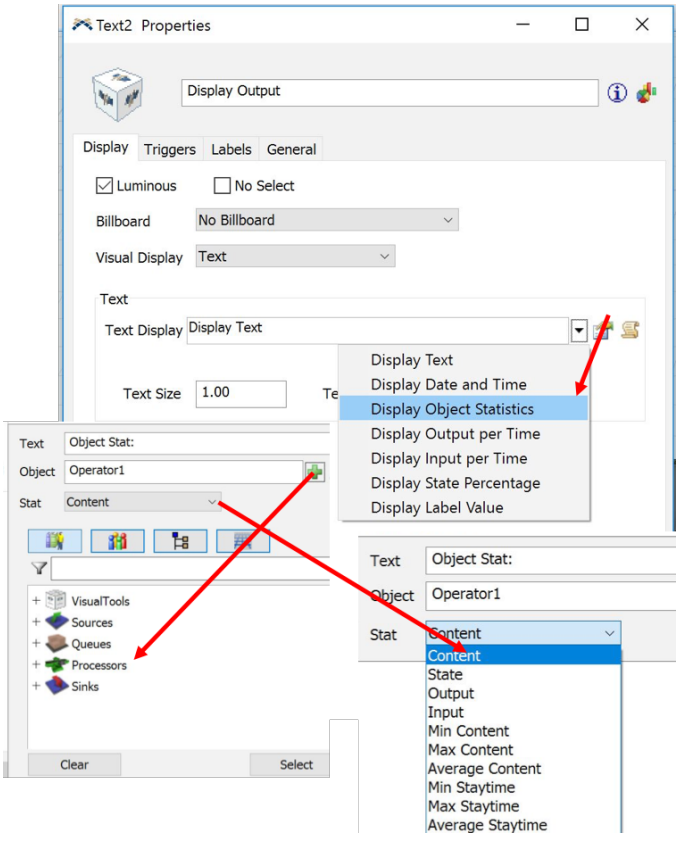


Figure 6.12: Visual Tool example.

Visual tools can also act as a container for other objects in the model. When used as a container, the visual tool becomes a method for hierarchically organizing a model. The container can be saved in a User Library and act as a basic building block in the development of future models.

Dashboard

The dashboard object allows dynamic model information to be grouped and displayed. The dashboard is created by selecting the menu option Statistics > Dashboard. Once opened, various text and graph displays can be added by choosing from the menus on the left of the screen. While the choices on the left indicate the type of output—text, histogram, chart, etc.,—the actual objects and statistics are available through the tabs. The object of interest is selected using the Universal Selection dialog that automatically opens. This dialog shows all the objects in the model organized by object type. More than one object can be selected to allow for comparisons. In the statistics tab, various statistics can be selected. The All the displays are updated as the model runs.

The displays can be moved and resized within the dashboard. Additionally, multiple dashboards can be created and organized by adding tabs using the icon in the upper right corner of the dashboard. Figure 6.13 shows a dashboard displaying dynamic information about the processer in a model.

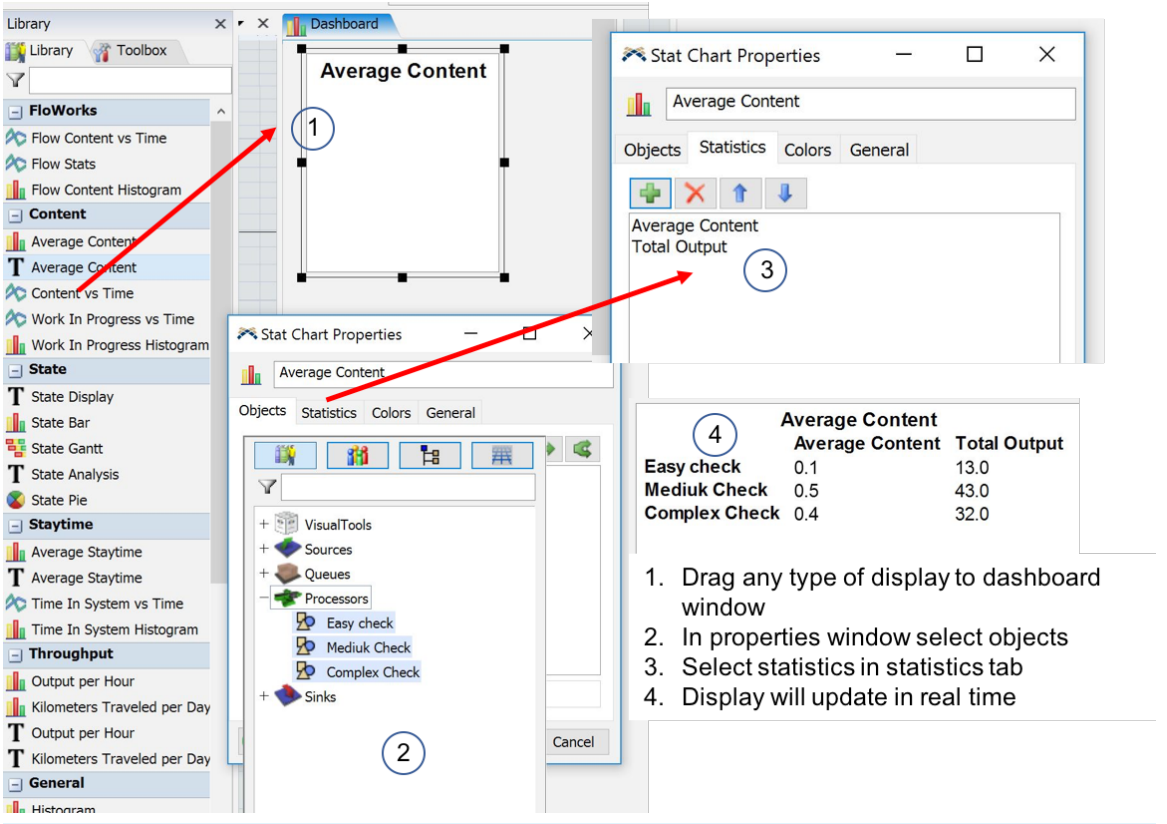


Figure 6.13: Dashboard example.

Creating a Flypath

Manually changing the view positions in the 3D simulation to view the objects from a different perspective is interesting. However, creating a fly-through of the simulation while it is running can provide a much more dynamic view. Building the flypath is easy.

In the toolbox, select to add a *Visual* > *FlyPath* element. As shown in Figure 6.14, a window appears at the bottom of the screen. The units along the timeline are in real time seconds. If necessary, click on the **(+)** icon on the left side of the timeline to start a new flypath. You can change the flypath name.

To start defining the flypath, use the mouse to maneuver the 3D model view to the point where the fly-through should start. Click on the **(+)** icon near the top-center of the timeline. This will create a diamond icon at time 0 and the curser will move to a point 2 seconds ahead of the starting point.

Continue to use the mouse to change the simulation view position and click the **(+)** icon. The default time between points is 2 seconds. The fly-through will move from one point to another during the time indicated between points on the time line. You can move points along the time line and add new ones at any time by clicking the **(+)** icon. The timeline scale can be changed by positioning the curser over the gray bar at the bottom and using the scroll wheel to expand or contract the timeline.

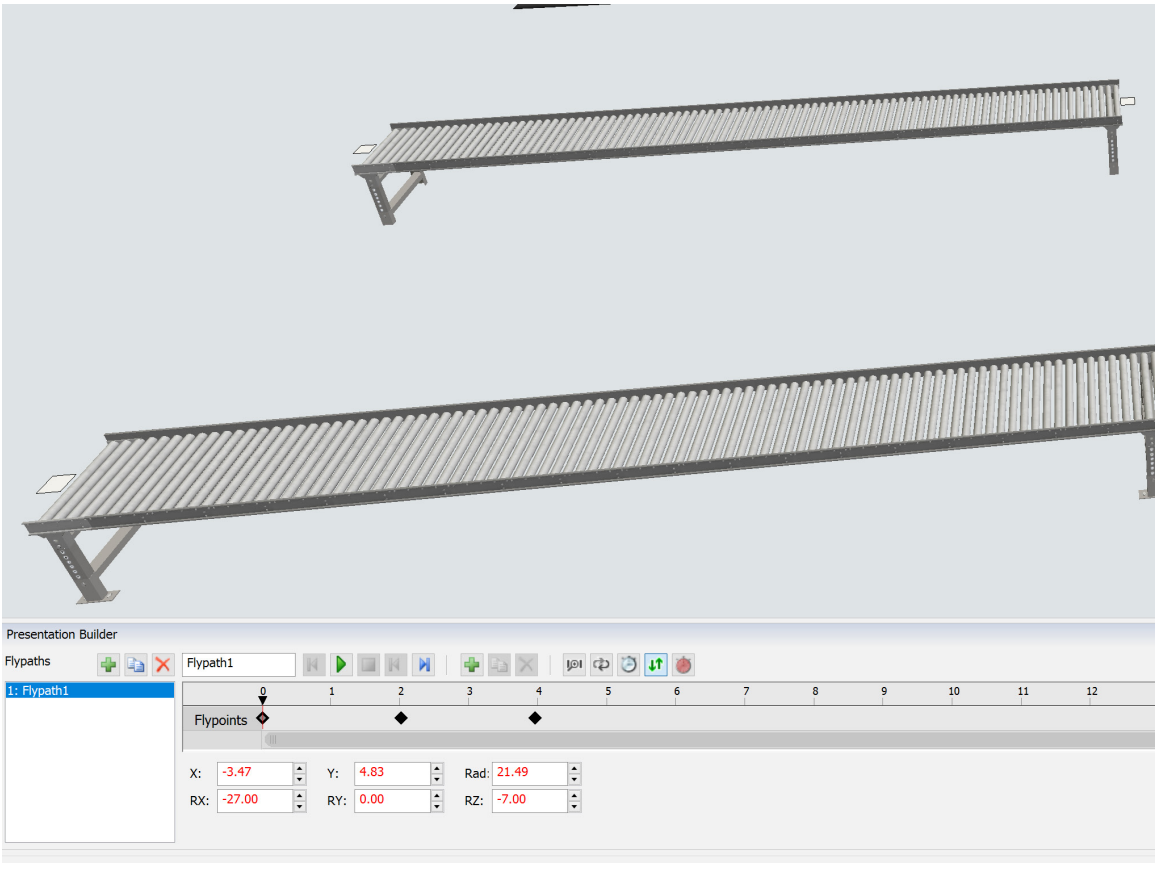


Figure 6.14: Using the flypath window.

The fly-through is viewed dynamically by clicking on the forward icon to the right of the flypath name. Other icons will stop or move the flypath to the start or end. When building the flypath, consider how long you want the motion to be. Going longer distances or greatly changing views may require a longer time between points or additional points. The flypath can be started and run when the simulation is stopped or running. For more details, consult the *FlexSim* help files.

FlexSim also provides the ability to create a video of the simulation along with the fly-through. To build a video, go to the Toolbox and select to add *Visual* > *Video Recorder*. A pop up window is used to configure the parameters of the video. Selecting the right combination of flypath time, simulation time, and frame rate can be confusing. Check the *FlexSim* help files. Third party applications that record the screen can also be used.

References

1. Patel, A. "Amit's A* Pages," http://theory.stanford.edu/~amitp/GameProgramming/index.html, accessed 31 May 2017.

Exercise 6-1 Steve's Stone Cutting

Background

Steve Alpert transformed a large covered area, as shown in Figure 6.15, into a custom stone cutting operation. In the shop, large stones are cut to a specific size and a basic shape for various landscaping uses. As business grew, Steve gave priority to his wholesale customers.

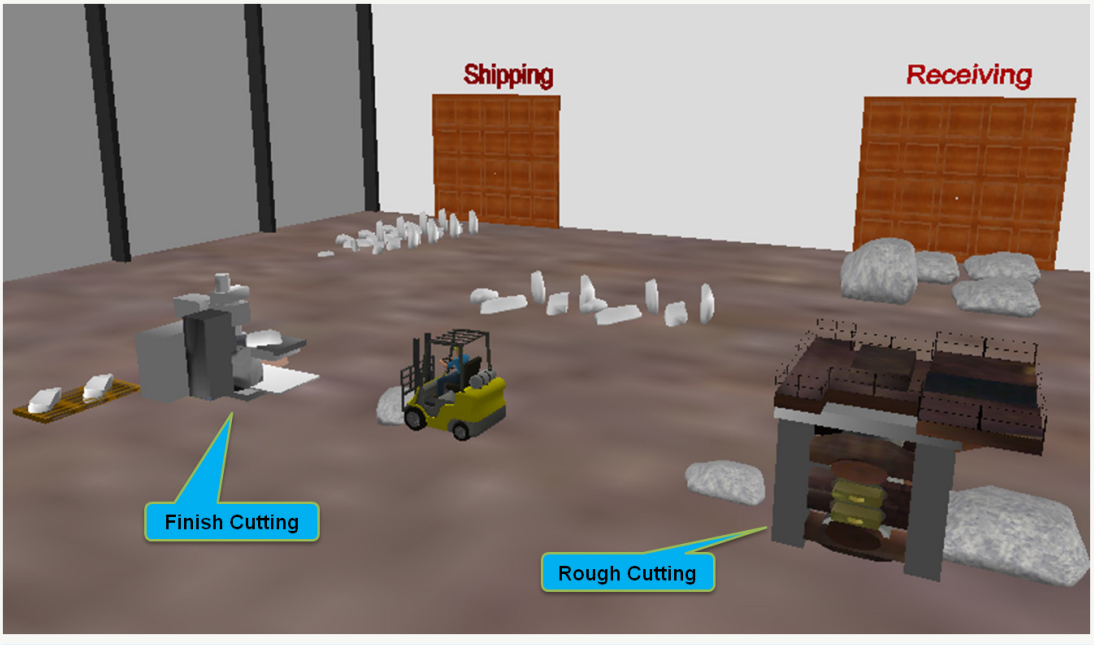


Figure 6.15: Steve's Stone Cutting operation.

Now that the cusrom business is nearly as large as the wholesale side, he's worried that by giving certain customers priority he will start losing new customers. Wholesale projects are usually larger and take more time in rough cutting but less in the finish cutting as the custom pieces require more work. His margins are larger on the custom projects.

Problem statement

Provide a simulation that shows the current wait times for orders and the possibility for improving production for both types of customers.

Operating data

Custom project arrive at the shop with an interarrival time of 50 minutes distributed exponentially. Wholesale jobs each arrive at the shop with an interarrival time that is exponentially distributed with a mean of 60 minutes. Steve has enough rough stones to fill all orders.

The distances (feet) between the equipment and the entry and exit bays are:

From	To	Distance (ft)
Entry Bays	Rough Cutting	200
Rough Cutting	Finish Cutting	400
Finish Cutting	Exit Bay	200
Exit Bay	Entry Bay	400

All stones first go through a rough cutting process and then are taken for final cutting. The stones are transported by a forklift truck between the entry bays (one for wholesale projects, one for custom projects).

Process times in minutes to cut the stones are

- Rough Cutting
  - Wholesale: Triangular distribution: min. 15, max. 25, mode 19;
  - Custom: Triangular distribution: min. 10, max. 18, mode 12;
- Finish Cutting
  - Wholesale: Triangular distribution: min. 10, max. 15, mode 12
  - Custom: Triangular distribution: min. 15, max. 25, mode 19.

The forklift travels a set circular route around the area between the equipment and entry and shipping areas at a speed of 50 ft. per minute. It takes 15 seconds to load and 20 seconds to unload.

Expected results

- Determine the number of each type of order that is received and processed during a 40-hour period.
- Note the average wait time for each type of order before process starts.
- What is the bottleneck? Would a buffer between the cutters help?
- Estimate the benefits of reducing the travel time for the truck by clearing a path through the center of the shop area or having another forklift.

Modeling and analysis issues

- How will wholesale orders get priority? Is there an easy way in the simulation to change from priority to FIFO? Remember that the order the input queues are attached to the Rough cutting machine indicates their priority.
- Are network nodes needed? If so, how should they be set up to represent the correct distances?
- What visual changes can you make to better observe the material flow?
- Consider how send the projects to to their respective shipping locations.

Exercise 6-2 Treasures-4-U

Background

Treasures-4-U is a new company that offers a wide variety of items primarily sold through airline catalogs and the Treasures-4-U website. The last step in their mail order business is a checking station where all packages are inspected and sealed. They are then taken to shipping areas based on weight and delivery type. As business has increased, this department has been falling behind in getting the work through. Staying on schedule is especially important for their ability to guarantee one-day shipping for a premium price.

There are three workstations in the department as shown in Figure 6.16.

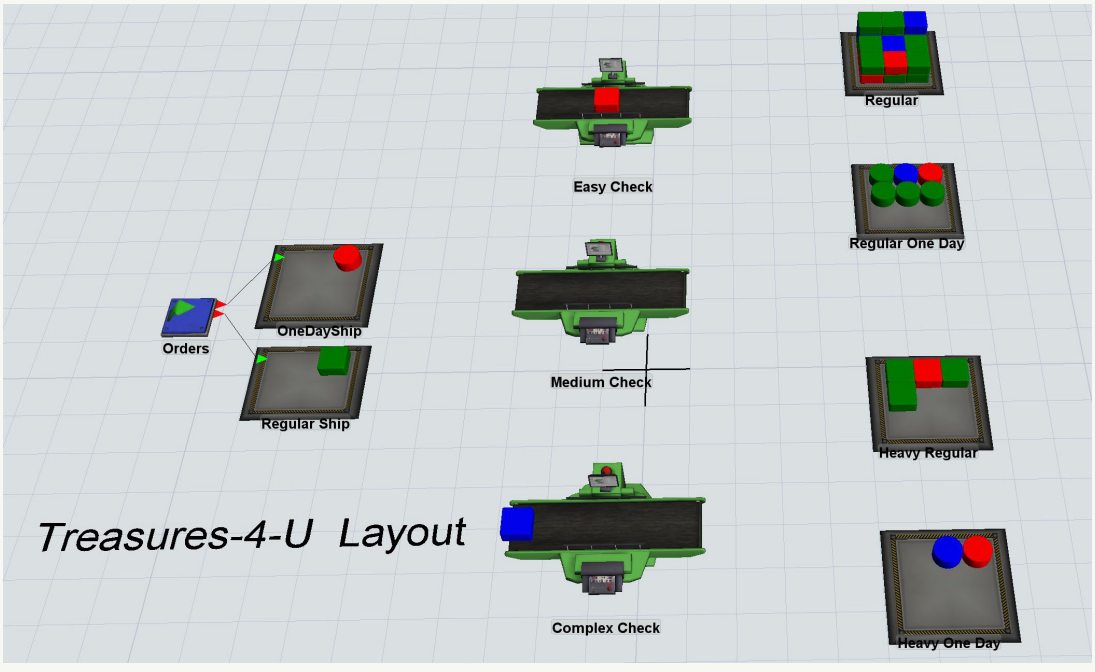


Figure 6.16: Treasures-4-U operations.

Each workstation focuses on a particular classification of order: Easy (there are few items to check); Medium (there are more items); and Complex (there are multiple items or parts).

Orders are also identified as heavy (greater than 15 lbs.) or standard. The heavy orders are handled on the largest workstation regardless of their classification.

A number of ideas that are being studied to improve the throughput of the department, including changing the assignments for the workstations, getting larger workstations to handle the heavy orders, and adding another workstation or more people. Management is looking for help from a simulation of the operation, especially if the order rate continues to increase.

Operating data

- Current inter-arrival time to the department: distributed as exponential(0,15) seconds
- Order weight: distributed as exponential(0,10) lbs.
- Order classification: Easy 30%; Medium 45%; Complex 25%
- One day delivery request 15%; Standard delivery 85%
- Package inspection times (seconds): Easy 15; Medium 30; Complex 60

Expected results

- Determine the number of packages that are shipped in a day.
- How many more shipments are gained from various configurations?
- How many of each type of shipment order are left at the end of the day (regular and 1 day)?

Modeling and analysis issues

- How many Lists are required?
- Can you use a task sequence list to manage the operators carrying the boxes to the shipping queues?

Chapter 6 Review Questions

1. Explain some of the actions performed by mobile resource objects.
2. What connection must be made between a transporter, such as a fork truck and the object that is calling it to perform a task?
3. What tab on the calling object is used to call the transporter?
4. Assume a task executer is used to transport a flowitem between two objects. Complete the list below of basic or default tasks that the mobile resource would perform.
   a. Travel to calling object.
   b.
   c.
   d.
5. Give an illustration of an operations system where a path network might be used.
6. Identify four functions that can be performed with the Visual Tool object and give an example of each.
7. What are the three main reasons for using a dashboard. Identify how the dashboard can be used to supplement the dynamics on the screen and the information provided through a Visual Tool object.
8. Provide a different example than that shown in the book of how the List function can be used to add functionality and simplicity to the flow of item in a simulation.

Chapter 7 Incorporating Resource Availability and Reliability

Objective

To define the concept of downtime, which is inherent in all operations; identify and apply the simulation parameters of both planned/scheduled stoppages and random failure.

As systems become more complex, additional functionality is often required for a more accurate representation of operations. In practice, operations do not operate continuously and simulations must be able to mimic this dynamic starting and stopping. People who interact with equipment are not always available—they have breaks during the day or may be called away for other activities. Many such stoppages are planned but others are random. Consequently, reliability has a significant impact on the performance of a system. It also plays an important role in determining the location and size of any material accumulation points, which is critical for establishing an efficient production system.

There are basically two types of downtime: planned and unplanned. Examples of these are illustrated in Figure 7.1. As will be discussed below, planned downtimes are typically implemented in *FlexSim* via Time Tables, and unplanned downtimes are implemented via MTBF (Mean Time Between Failure) Tables. In both cases, adding downtime to resources in a model is quite easy.

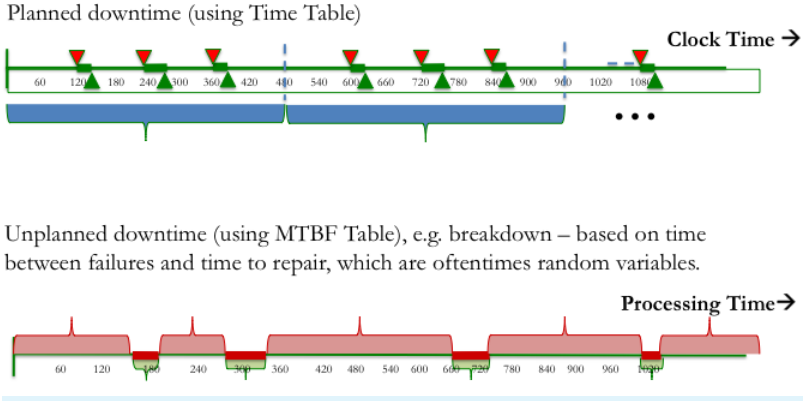


Figure 7.1: Planned and unplanned downtime.

Planned downtimes are anticipated or scheduled (e.g. operator breaks or resource changes for different shifts). Both the start

time and the duration of the downtime are known. As shown in the timeline at the top of Figure 7.1, patterns of planned downtime usually repeat. For example, a 15-minute break occurs two hours into a shift, no matter what time the shift starts; similarly, a 30-minute lunch break occurs four hours into a shift, no matter what time the shift starts. Planned downtimes, when implemented via a Time Table, are based on clock time (i.e., a break occurs two hours into the simulation no matter how much the operator has been utilized).

Unplanned downtimes typically do not occur according to a plan or schedule; in fact, the time between downtimes (failures) is usually a random variable. Likewise, the amount of time a resource is down is usually unknown and oftentimes represented as a random variable. Unplanned downtimes are usually not based on clock time but depend on the states of the resource. The time until a failure occurs on a machine usually depends on how much the machine has run and not the clock time. For example, if a failure is expected after two hours of run time and the equipment is only busy 50% of the time, the failure will occur in four hours, not two. There may be many sources of unplanned downtime, mechanical issues (e.g., material wear), electrical issues (e.g., power interruptions), material consumption (e.g., paper in a copier), etc. Managing all of these different downtimes gets to be quite complex. Fortunately, *FlexSim* does an excellent job of managing multiple downtimes.

Section 7-1 Establishing Time Tables

During the course of a simulation, many objects may start or stop depending on the time of day. For example, workers may not be available during break times, or equipment may only be used on certain days. Starting and stopping times for equipment are commonly determined by a particular work schedule. Creating and managing work schedules usually requires custom programming. However, in *FlexSim*, such time tables are established by building a time table in the Tools tab located on the main menu bar at the top of the screen.

Each time table can control many objects, and each object can be controlled by multiple time tables. The mode dropdown choices control how the data is viewed and repeated in the simulation. Note that filling in data under one mode selection does not automatically transfer to the other modes so only use one mode for each table. Complete information about time tables is included in the Help files.

Consider an example of a small job shop that operates 8 hours (480 minutes) per day. At the end of the day everyone leaves, and then on the next day they pick up where they left off. There are three break times when no orders are accepted and all work stops:

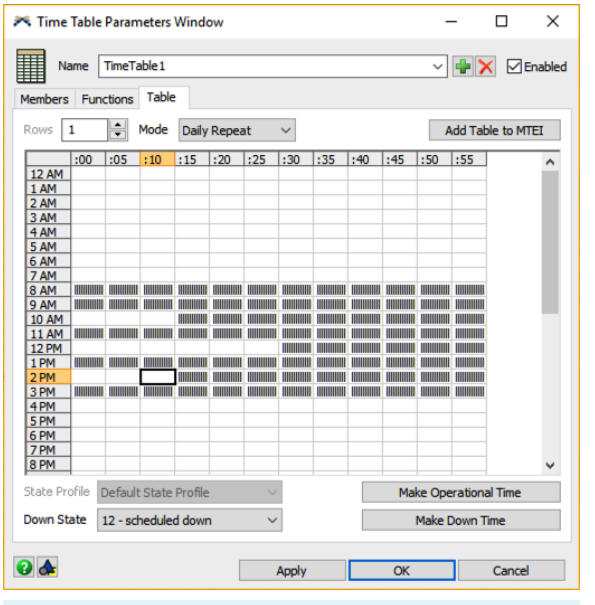


Figure 7.2: Fixed downtime described graphically.

- A 15-minute break after 2 hours (120 minutes)
- A 30-minute break after 4 hours (240 minutes)
- A 15-minute break after 6 hours (360 minutes)

Figure 7.2 shows the graphical approach for setting the times when resources are available (shaded cells) and unavailable (blank cells). In this case the resource is available between 8 a.m. and 4 p.m., except during two 15-minute breaks at 10 a.m. and 2 p.m. and a lunch break between 12:00 and 12:30. As the mode indicates, this schedule is set to repeat daily.

An alternative approach is to enter the downtime information explicitly via the Table form of the time table. There would be 4 rows. One for the morning, lunch, and afternoon break and then the downtime overnight. In each row, the following are specified: the time when the downtime begins, the state (e.g. state 12 is "scheduled down"), and the duration of the downtime. The mode setting has this pattern repeat every 1440 minutes (one day).

However, since the simulation is only interested in the daytime hours when the shop if open, only three entries are needed and the pattern repeated every 480 minutes (working hours) as shown in Figure 7.3.

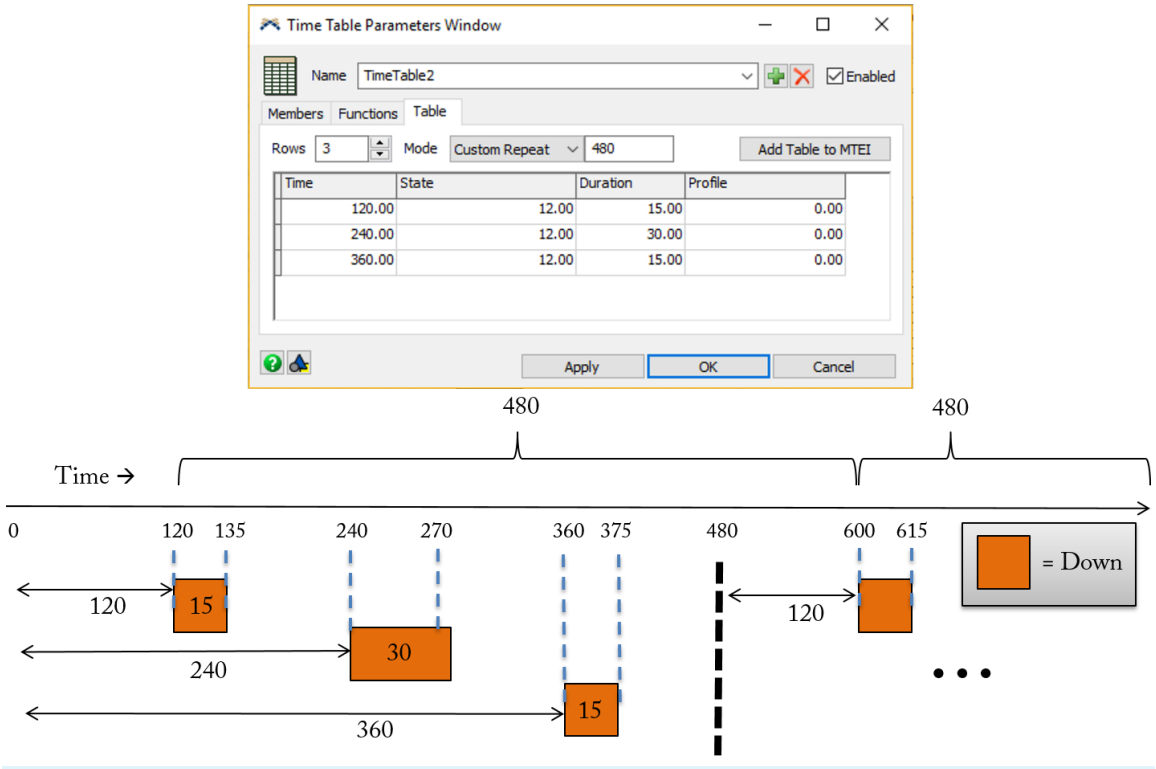


Figure 7.3: Fixed downtime described in a table.

Figure 7.4 illustrates a different approach for specifying a break schedule. Here, a 15-minute break is taken every 120 minutes. In this case the pattern repeats every 135 minutes. This is illustrated in the time line below the *FlexSim* time table entry window in Figure 7.4.

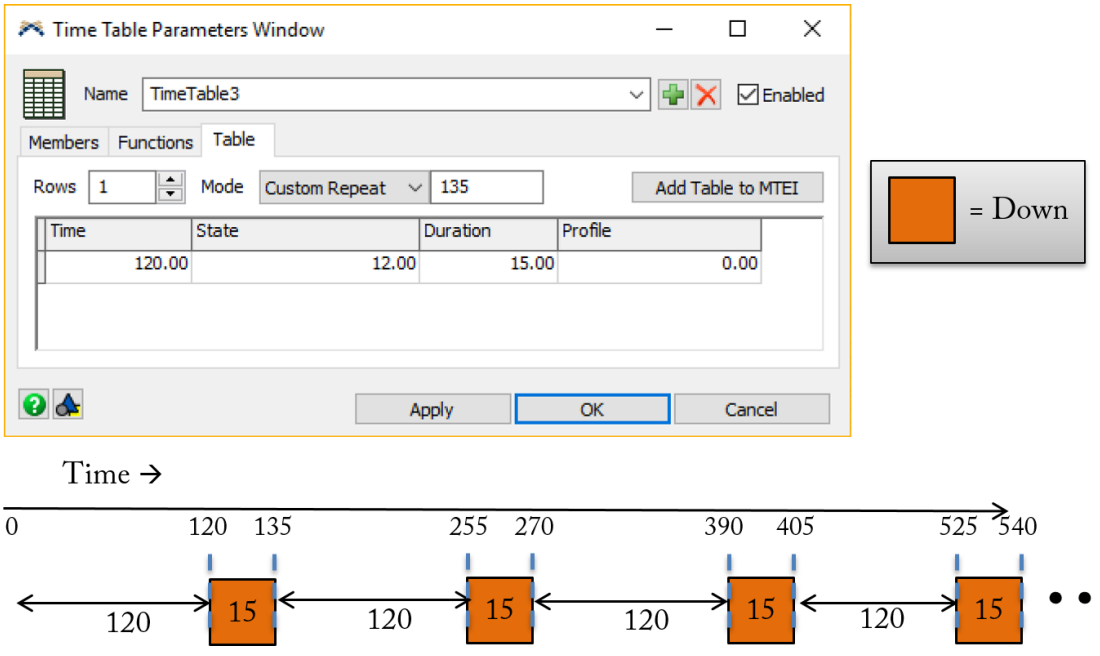


Figure 7.4: An alternative approach to scheduling downtime.

Section 7-2 Performance

Analyzing a system's performance in the face of random events is often the goal of a simulation. Performance can be a confusing term since the parameters that define it may change based on historical practice or other factors. The following are commonly used performance measures; they are often used interchangeably:

The first section of the interface is used to set values concerning the operation of the task executer; the variables vary somewhat depending on the role that the task executer performs.

Task executers that are being used to convey flowitems have the following built-in attributes:

- Maximum rate: Best possible output rate based on design
- Ideal rate: Maximum output rate based on theory
- Target rate: Output rate that is considered consistently achievable
- Planning rate: Output rate used for planning and scheduling purposes
- Efficiency: Actual output rate measured against some specified metric
- (usually target output rate)
- Performance: Some metric indicating how well a system is operating
    - System Performance: Based on an entire operating line
    - Unit Performance: Based on a single piece of equipment
    - Plant Performance: Based on multiple systems

Understanding how performance is measured and calculated is important because economic decisions will be made based on the defined measures. In practice, performance measures are often made as part of an assessment for an operation and can impact a financial bonus or result in a plant being closed. It is therefore essential that performance metrics in a simulation reflect the assumptions and calculation methods of the target system.

Simulation software provides many standard performance measures, which are assessed at the object level. These include:

- Throughput: Number of items moving in and out of an object
- Content: Minimum, maximum, and average number of items in an object over the duration of a simulation
- Staytime: Minimum, maximum, and average time items spend in an object over the duration of a simulation

It is sometimes necessary to consider measures that describe system performance in ways other than the standard or pre-defined measures provided by the software. In those cases, custom code must be used in order to include the performance measures in the model.

Performance does not always have to be a measure of output or time. For example, in some industries, sequence is an important measure. In such cases, sequence measures how closely the individual products in a set maintain their order from when they enter a system to when they exit. This measure is important in just-in-time systems.

Typically in an analysis, multiple measures of performance are considered and have to be traded off against each other in order to make a decision; of course, the number of measures being considered should be limited to the most important ones. As the saying goes, we manage what we know and we know what we measure. We need to measure the critical few rather than the trivial many.

Section 7-3 Reliability

A major contributor to a system's performance is the reliability of the system's components. In this section, reliability is discussed from the perspective of equipment. While it's easier to think of stoppages and breakdowns in terms of manufacturing machines, the same concepts hold for almost any system. For example, a simulation of a call center includes "breakdowns" when the person typing information from the caller makes a mistake and has to go back and correct it. In a similar way, a simulation of a bank contains stoppages when a teller makes a mistake and has to call a manager over to help correct it.

The problem with reliability is that, in general practice, it is a term that is often misunderstood or incorrectly used. It's often applied interchangeably to entire systems and to single machines. Also, data collection time to estimate reliability may be too short. For example, an engineer may go to an equipment vendor and watch a test of a machine for a short period of time and declare its reliability to be, say, 98%, a percentage that is not appropriate for the short data collection time.

Common terms for reliability include the following:

- Availability: Percent of the time that a machine or system operates as expected
- Reliability: The probability that equipment can perform continuously, without failure, for a specified interval of time when operating under stated conditions
- Mission length: The interval of time that the equipment or system is expected to perform; the mission length may be years for a space probe or may be the length of a production run
- MTBF: Mean time between failures, the average time between failure/stoppage occurrences
- MTTR: Mean time to repair, the average repair time for the failure/stoppage

Reliability at a given time, R(t), can be expressed as

$$R(t) = Pr\{T > t\} = \int_t^{\infty} f(x)\, dx$$

where f(x) is the failure probability distribution. If the failure distribution is assumed to be negative exponential, then the failure rate is assumed to be constant over time. The constant failure rate is based on the exponential distribution's "memoryless" property and does not reflect burn-in time (when the failure rate decreases) or wear-out time (when the failure rate increases).

In the exponential case,

$$R(t) = e^{-\lambda t}$$

where A is the constant failure rate.

Understanding the time involved is not as simple as it looks. The length of time indicated is usually not continuous clock time but is the time consistent with the activities associated with the failure process. This is called the failure clock. For example, a wrapping machine may jam while wrapping a product, so the time between failures is based on the time that the machine is actually wrapping. If the input flow is stopped for some reason, the time between failures should not accumulate. In a simulation, if the failure clock is not correctly maintained, a machine may show a simulated wrapping failure when the machine is actually sitting empty with no product in it; therefore, the failure clock in the wrapping example should only track the time the machine is in the wrapping state.

As shown in Figure 7.5, the downtime is the total time it takes to repair the failure and get the machine working again. Downtime includes the time for an operator to be aware of the failure, the time for the operator to complete a current task in order to become free to work on the failure repair, the time to travel to the machine, the time to actually repair the failure (including getting any tools and materials), and the time to start the machine back up.

Oftentimes when operating personnel mention repair time, they refer just to the time needed to actually fix a machine; however, others may consider repair time as all of the activities and wait times involved from the time a machine fails until it is restarted. Repair time can therefore refer to two very different sets of activities and resulting times. It would be better if the term downtime were used to refer to the total time a machine is down due

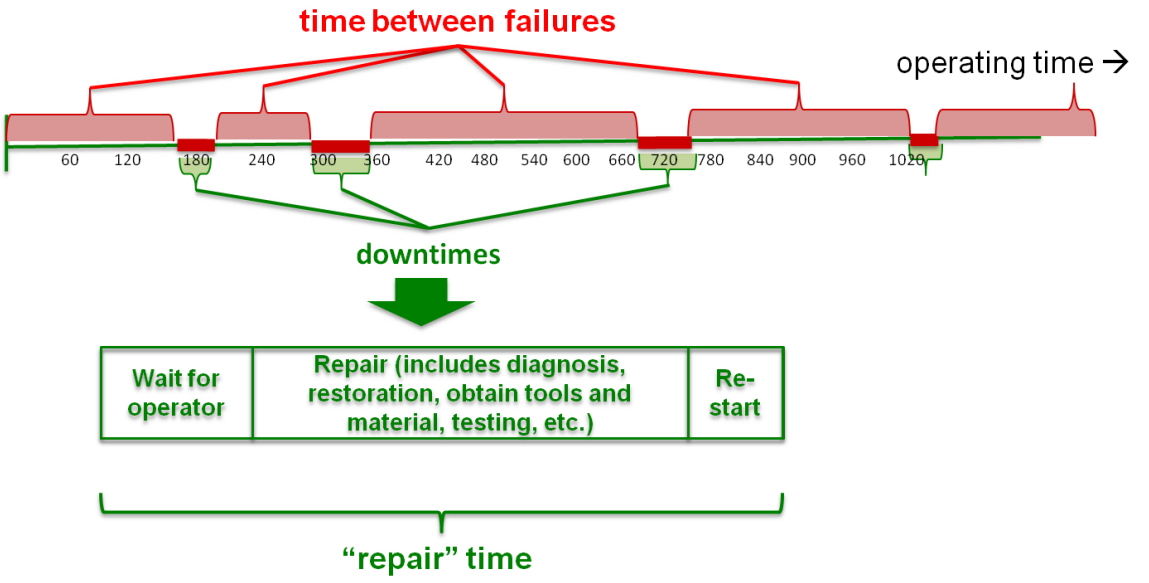


Figure 7.5: Elements of an equipment breakdown with repair.

to a failure event, and repair time was only the portion of that total that involves hands-on repair; however, typically in a simulation

- without operators (or other shared resources used for repair), the repair time is the total downtime, as defined above;
- if operator (or other shared resource) intervention and travel is simulated, then the repair time is only the time to actually perform the repair.

Section 7-4 Estimating MTBF and MTTR

The time between failures and time to repair are usually random times that can be characterized by a statistical distribution. The MTBF (mean time between failures) and MTTR (mean time to repair) are the mean values of those distributions. As with any statistical analysis, obtaining historical data over a longer period of time is always better; however, because of a lack of understanding about statistics in general, and reliability in particular, most organizations do not collect sufficient data.

A common metric for production lines is the total downtime during some period of time—a shift, a day, a run, etc. It's a metric that managers focus on and want to minimize; however, if only the total downtime is known, trying to focus on improving production is very difficult since the metric gives little detail for analysis. Plants will often keep records of the number of failures as well as the total downtime, and then use this data to calculate an average downtime. Unfortunately, these metrics still don't offer much more toward a good analysis.

Modern manufacturing systems have the ability to track downtime information for individual machines. While this feature is helpful, there can still be problems.

Providing only an average downtime along with the number of stops doesn't give any information about the underlying statistical distributions. Additionally, machines may stop for many reasons other than because of a failure. The machine may be blocked because of a bottleneck in a downstream machine or stopped for a reason related to production.

Exclusive downtime tracking software is available in many control systems. It not only tracks stoppages, but also indicates which machine is at fault as well as the nature of the fault. With such historical data, MTBF and MTTR can be calculated with some degree of certainty.

All is not lost if historical information is lacking or not available. For a new system with no history, a simulation can still be valuable for conducting a sensitivity analysis by experimenting with and analyzing various levels of MTBF and MTTR. Such analysis can lead to improved design decisions and provide reliability targets for new machines.

When simulating current operations with limited historical data, there are still some ways to estimate MTBF and MTTR. One way is to question operating personnel; however, simply asking for an operator's estimate of availability (or, in their opinion, reliability) may yield values that provide little useful information, such as stating the machine is available 95% of the time. It is better to ask specific questions, like the number of times a machine fails per hour (or any time period) and how long it takes to get it running again. It may then be necessary to ask additional questions to refine these estimates. Additional questions could help determine what activities the repair time values include, such as response time estimates, hands-on repair time, etc.

By knowing the number of downs per unit of time (in this case 1 hour) and the average time to repair, the MBTF and MTTR can be estimated since,

$$(\text{MTBF} + \text{MTTR}) \times N = \text{Total operating time } (T), \text{MBTF} = (T - N \times \text{MTTR}) / N,$$

where N is the number of stops in time T.

Such a simple estimate may have to be modified based on the specific piece of equipment and the failure clock time that should be used. Estimating the average downtime and repair time is only part of the work. The underlying statistical distributions also have to be estimated. The exponential and Weibull distributions are common starting points for downtime and repair time.

Section 7-5 Simulating machine failures

The level of detail used to simulate a system should be consistent with the reliability data available. If the available data is only for a system of machines as a whole, then the simulation could be simplified by using a single machine with downtime characteristics to represent the system. Trying to simulate each machine is difficult, although it is often done by assigning the system failure characteristics only to the first processor.

In some situations, a simulation may need specific information for individual pieces of equipment. For example, a case packer machine will have separate failure modes for different steps or processes, such as

- collecting materials to pack;
- a case building section;
- the filling process;
- a case closure section;
- an output transport section.

Additionally, machines may have different types of failure modes: short-term (jams), intermediate-term (in-feed or out-feed conveyor problems), and long-term (belt failure). These multiple failure modes are called competing failures. Each competing failure will have its own statistical distribution and may have its own failure clock.

Without exclusive downtime data, identifying and characterizing competing failures is nearly impossible. Figure 7.6 shows the independent histograms of two simulated competing failures—the first with an MTBF of 4 minutes and the second with an MTBF of 60 minutes.

The combined histogram of the times between failures shown in Figure 7.5 demonstrates the problem that arises when the data are combined with no indication of the individual components.

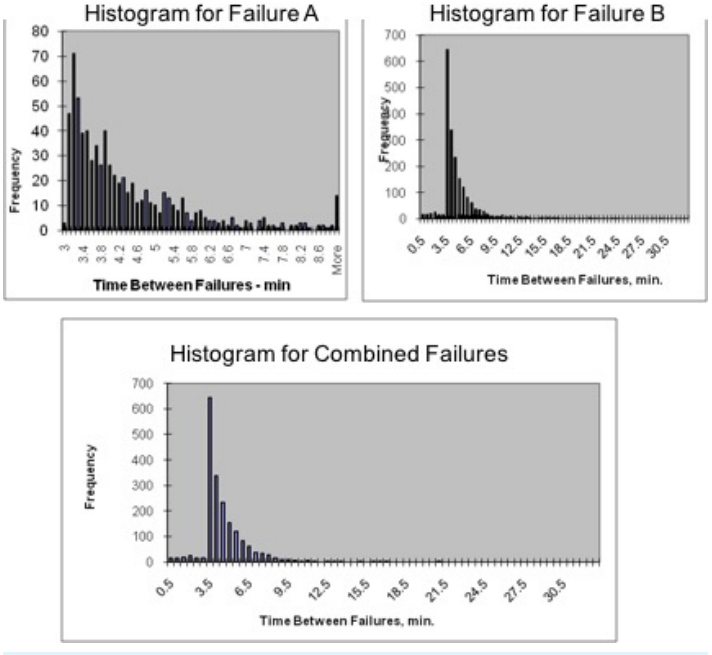


Figure 7.6: Competing downtimes.

Quite often all failures of a machine are grouped together and are then characterized by a single distribution and the resulting MTBF. Similarly, a single repair distribution is considered along with its MTTR. For machine design, however, the competing failures in different sections of the machine need to be considered separately since they both involve serial and parallel components.

Outside influences can also cause downtime; such downtimes may include operator break times, scheduled meeting times, and even preventive maintenance downtimes. Taking competing downtimes into account and correctly incorporating them into a simulation is important for conducting an accurate analysis. Many simulation programs work with only one failure clock, which is the same as the simulation clock. *FlexSim* allows for unlimited competing failure modes, each based on its own separate failure clock time; this is discussed in the next section.

Four types of information are needed to specify each type of downtime:

1. The object the specified downtime applies to (multiple objects can use the same downtime specification)
2. Distributions of time between failures and repair time
3. Actions that result when a downtime occurs (e.g., a call to an operator to make the repair)
4. The states that are used in the downtime or failure clock (for example, for a stoppage that only depends on operational or process time, the times the object is in the idle or wait for operator states are not counted).

Section 7-6 Setting MTBF and MTTR in a simulation

All simulation software applications provide some method for simulating random downtime events. The best way to understand how failures work in a simulation is to experiment with a small simulation model. The simulation, like the one shown in Figure 7.7, just includes a source, processor, and sink. To make checking the results easier, set the source interarrival time as well as the processor process time to a constant value of 1.

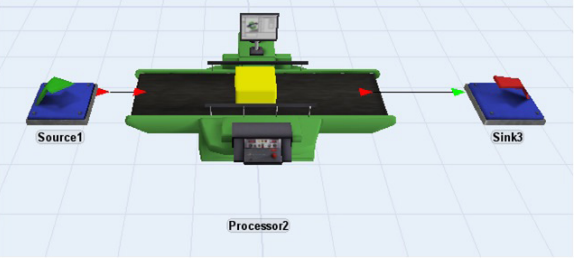


Figure 7.7: Simple model to test reliability functions.

In *FlexSim*, downtime information can be entered on the objects themselves by using the Breakdown tab on the object, as shown in Figure 7.8. Clicking the add button and then the name creates a new breakdown table

Each table created represents a particular type of downtime that can affect that object or many objects. A single object can have multiple entries representing various downtime events. The tables created are stored as global tables and can be accessed through the Tools menu as well.

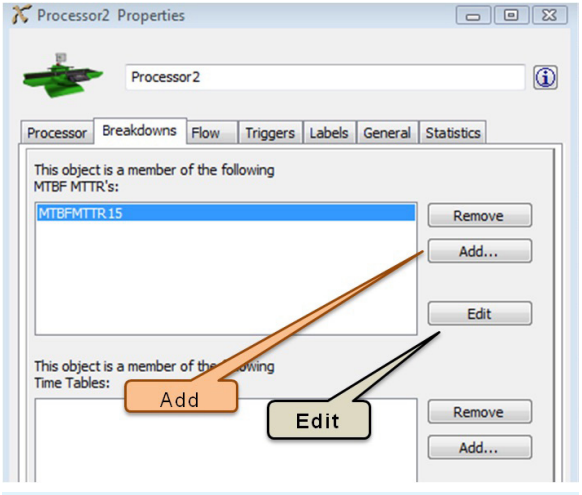


Figure 7.8: Generating a breakdown table.

The table parameters window has three tabbed sections that provide a very extensive and robust means for defining a particular reliability event such as what should take place when the machine fails or when it starts back up after repair. In the Members tab, objects that use this downtime table can be specified. They can all respond to the same event simultaneously or individually using the same base statistical distribution. The Functions tab, shown selected in the figure, is used to describe the statistical distributions associated with the breakdown. The Breakdowns tab is used

to indicate what object states should be considered for this particular event. Any state or combination of states can be specified. Details for filling out the reliability table are provided in the Appendix.

A quick experiment will demonstrate why assigning states is important for reliability simulations. In the simple model defined above with an MTBF/MTTR table assigned to the processor, disconnect the connection from the source to the processor. Click the Reset button and then the Run button. After a period of time using the default values for MTBF and MTTR, the processor will have a yellow box around it showing that it is in a breakdown state and being repaired. This is illustrated in Figure 7.9. Notice that the source isn't connected so there is no material going through the processor. However, it still shuts down periodically. This can be embarrassing when showing the simulation to a high-level manager or to others whom you are trying to convince that the simulation is accurate.

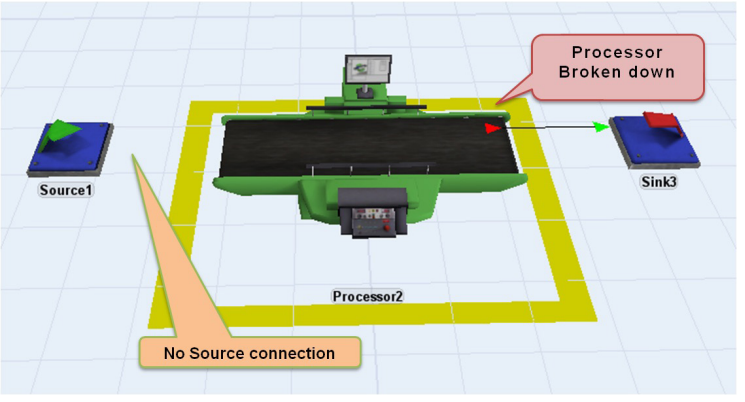


Figure 7.9: Processor in breakdown state.

This problem occurs because the failure clock, as discussed above, is the same as the simulation clock. In this case, the failure clock should only accumulate time toward failure when the processor is actually processing. To correct this problem, open the MTBF/MTTR table and click on the Breakdowns tab. As shown in Figure 7.10 check the box "Apply MTBF to a set of states" and then move the appropriate state(s) to the right column.

Repeat the experiment and notice that the processor no longer fails when it is empty. For an object such as a combiner, the failure time might be set up to apply to the collecting state as well as the processing state.

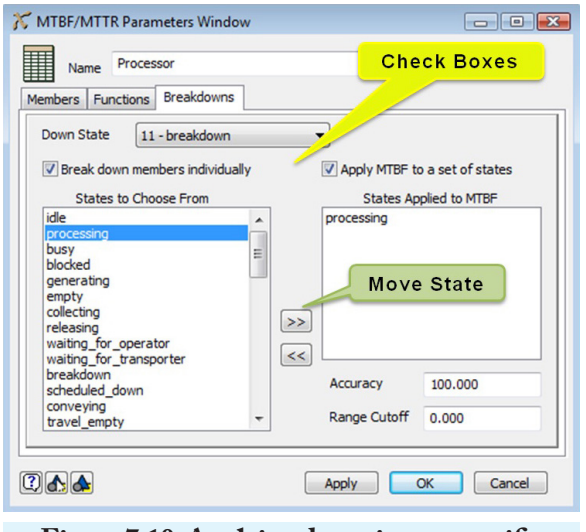


Figure 7.10: Applying downtime to specific states.

Try other experiments such as adding another processor in series or parallel. They can be given the same downtime characteristics (but not fail at the same time) by opening the MTBF/MTTR table, going to the Members tab, and bringing the new processor to the right-hand column.

Competing failures are added by creating a new MTBF/MTTR entry with the failure information and then choosing the same processor. A combiner may have one failure mode based on the collecting state and another based on the processing state.

Being able to accurately simulate reliability for so many cases means that data has to be collected, analyzed, and applied correctly.

Section 7-7 Using personnel for repairs

When dealing with downtimes, it is often necessary to simulate both the downtime itself and the response time of someone coming to perform the repair. To designate that another resource must be present to make the repair, choose the drop-down menu choice in the Down Function that will call an operator, as shown in Figure 7.11.

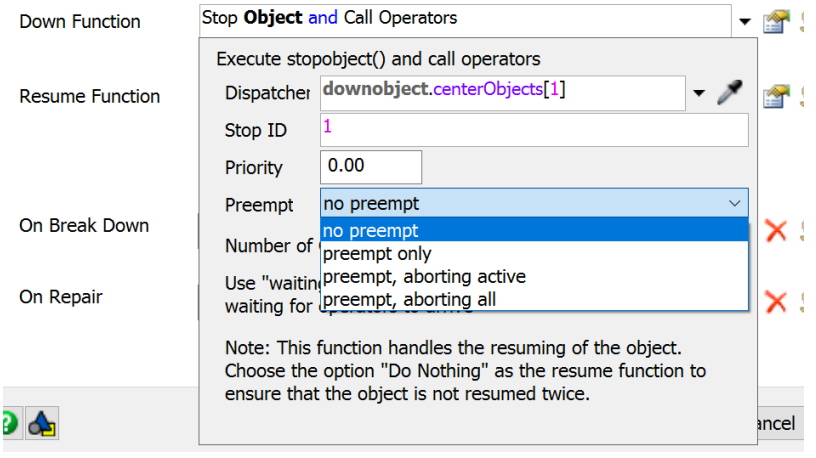


Figure 7.11: Calling an operator for a breakdown.

When it is time for an object in the list under the Members tab of an MTBF/MTTR table to break down, a call will be sent to the operator (or dispatcher) connected to the first center port of the down object. This is the default connection and can be changed if needed. If only one person can service the machine, that operator would be the one connected to the object with a center connection. Often the repair may be made by anyone in a pool of specialists. In the case that the repair can be made by any specialist, the down object can be connected to a dispatcher that has several people (operators) attached to it.

The Preempt Choice can influence what happens when an operator is called to perform a repair. By default, task executers (e.g., operators, cranes, elevators, etc.) always finish a task before starting another. If the operator being called for a repair should not stop its current task and go immediately to the repair task, the Preempt choice should be set to no-preempt. That means that only when the repair task is complete will the operator will be freed to go to another task. For example: if an operator is used for a process as well as a failure repair, there should be no preempt on the processor and preempt only in the MTBF table.

Choosing to preempt only means that the operator will break from the current job to go to another of higher priority. Preempt aborting means the operator will not return to the current task. If the preempt aborting all is chosen, the operator will not return to the current task or any other task previously preempted. The choices involving aborting are usually only used with advanced programming. Since this particular Down Function automatically restarts the down object and frees the repair operator, the Resume Function should be set to Do Nothing.

The Priority value is used to decide which task of many the operator should perform next—the higher the number, the higher the priority. A Stop ID can be specified in order

to trace the particular downtime event. If a "yes" is selected for "waiting for operators," the time it takes an operator to respond to a breakdown will be added to that state for the down object.

While this discussion of interrupting the work an operator is doing is focused on responding to a repair task, the concept applies to all task executers, including transports, cranes, elevators, etc. This functionality provides a very robust way of simulating nearly any work rule environment where priority-based decisions have to be made in allocating resources.

Section 7-8 Surge

Surges are areas where material is stored while awaiting processing. Common surges include a raw material holding area or a warehouse where finished products wait to be shipped. A general principal of lean manufacturing is to minimize the use of surges since there are no value-added functions being performed; however, there are places where surges are useful.

One method for improving throughput when machines incur failure modes is to use a surge in front of the processor. Although lean principles would argue against any such surge, a carefully designed surge can improve operations; however, there are some points to consider.

First, for any surge to be effective, the rate of the processor and all machines downstream of the processor (downstream resources, or DSR) must be greater than the rate upstream or those coming into the processor (upstream resources, USR). In other words,

Downstream rate (DSR) > Upstream rate (USR).

If this condition is not met, the surge would simply fill each time the processor stops and never empty out. Additionally, the surge needs a recovery time (RT), which is the time it takes to empty the surge while the processor is running and material is coming into it. Obviously, to be effective, the surge should empty before the processor fails again. If, over time, that doesn't happen, then the surge will continue to fill until it is full and then can no longer be used as a surge. That requirement can be represented by the following equations:

RT = MTTR / (DSR/USR – 1) and RT < MTBF

Try the following to experiment with surge in a simulation:

1. Build a simple model with a source, conveyor, surge, processor, and sink to experiment with surge size and the ratio of downstream and upstream rates.
2. Start with a source rate that is less than the processor rate.
3. Create an MTBF/MTTR table for the processor with processing as the failure clock.

The requirements for surge size are only approximate because of the statistical nature of the failure modes. Experiment with different values of surge size and ratios of source and processor rates.

Exercise 7-1 Keggler's Brew

Background

Keggler has found a niche market by selling their microbrew beers only in keg containers. Their main customer base consists of local taverns and restaurants. During an economic downturn they have found that demand for their product actually goes up. As director of marketing you have recognized that there may be a growing market for smaller sized kegs, so management made the decision to add two smaller sizes: A quarter keg and a fridge keg. Focus groups have been very enthusiastic about the new sizes.

The company has decided to update the current line so that it is capable of producing all three sizes of kegs. Their current line of standard kegs runs with an operating efficiency in the high 90s. Thinking there may be some inefficiencies associated with producing the smaller sizes, the business plan is based on an 85 percent operating efficiency.

Problem statement

Should the plant manager agree to the target numbers?

Operating data

As shown in the Figure 7.12 and the accompanying data tables, empty kegs are taken from storage and put on a conveyor by a placement machine. The kegs arrive in the filling room by conveyor and go through the filling machine. From the filling room the kegs move on a conveyor to the packaging area where they are placed on pallets.

Plastic sheeting is wrapped around the kegs and pallet to keep them together. After wrapping, the pallets are conveyed to a pickup queue (with a capacity of 2) where a fork truck gets the pallet and transfers it to cold storage.

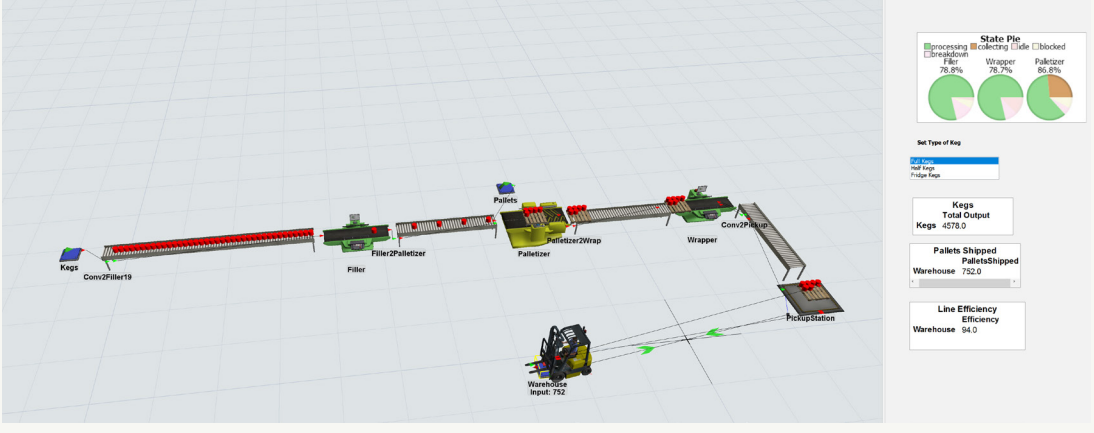


Figure 7.12: Keggler's Brew operations.

Product	Keg arrivals Kegs/min	Filler Rate Kegs/Min	Kegs/ Pallet	Pallet Cycle Time (sec)	Max Pallets Per 8hr Shift
Full Kegs	10	12	6	23	800
Quarter Kegs	16	18	8	28	960
Fridge Kegs	24	30	12	30	960

Table 7.1: Basic equipment data.

The wrapper takes 30 seconds regardless of the number of kegs on the pallet. The fork lift, which carries one pallet at a time, takes 15 seconds for a one-way trip between the pallet pickup queue and the warehouse.

Conveyor	Length – ft.	Capacity	Speed – ft./sec
Source to Filler	10	40	0.2
Filler to Palletizer	10	20	0.5
Palletizer to Wrapper	10	4	0.25
Wrapper to Pickup	5	5	0.5

Table 7.2: Conveyor sizes and speeds.

Reliability information, taken from historical downtime data of similar equipment, shows the following times and distributions.

Equipment	MTBF (sec)	Distribution	MTTR (sec)	Distribution
Filler	900	Exponential	90	Uniform: min 60; max 120
Palletizer	1000	Exponential	90	Uniform: min 60; max 120
Wrapper	1200	Exponential	120	Uniform: min 80; max 120

Table 7.3: Times and distributions.

Expected results

Run the simulation for a number of 8-hour shifts (28,800 seconds per shift), showing the number of pallets brought to the warehouse during that time; compare the results to the maximum and the expected total.

Modeling and analysis issues

- What time units should be used?
- How should the data be organized? What tables are needed?

- How can the keg size be set in just one place?
- What would be the advantages/disadvantages to the following ways to simulate the line operation:
    - A separate simulation for each product?
    - One simulation changing variables for each product run?
- For the machines with changing rates, where should the data be stored?
- How can you change the number of kegs on a pallet easily? If the standard on-entry trigger is used on the combiner, the pallet will have to know what size kegs are being used.
- How would you validate the model to determine if it is working correctly?
- What inter-arrival rate should be used on the keg source? The same question applies to the pallet source for the combiner.
- The keg size can influence the simulation since the number that can be set on a conveyor is specified. Can the conveyor virtual length help? If a virtual length is used, what should the speed be set at to insure the travel time is correct?
- What visualization improvements could be made?
- How can the fork truck travel time be set?
- Where is the bottleneck? Will the use of a queue be helpful and how can it be easily added to the simulation?

Exercise 7-2 Chairs for Tots

Background

Chairs for Tots builds children's chairs that are sold through major outlets and over the internet. The production facility is just keeping up with their order rate but is also trying to keep operating costs down. Work in process at the end of the week is continued the next week. Ideally, operators help clean the area at the end of the week; however, if there are orders still to be completed, the cleaning operation becomes difficult. The plant doesn't want to add overtime but management feels they have to do something to increase the weekly output and still have time to clean and service the equipment.

Problem statement

What will be the impact of higher order rates on the operations?

Operating data

The base order rate for chairs is approximately one every 25 minutes (distributed exponentially) but is expected to increase to one every 20 or even 15 minutes during a peak period. The plant starts operations at 8 a.m. Monday morning and runs 24 hours a day. It stops taking orders at noon on Friday so that the work in progress can be completed, the area cleaned, and the workers sent home at 4 p.m.

There are 4 major production steps:

1. For each order, a set of raw materials is conveyed to the cutting machine loading queue. An operator moves the wood from the queue to the cutting machine, which operates automatically.
2. Once cut, the cutting operator places the set of parts on a conveyor. The parts move to another queue where they wait to be assembled and painted. The assembly operator takes the parts and then assembles and paints each chair.
3. The painted chairs are conveyed to an oven to dry and set the glaze. The oven can hold up to five orders. Therefore, five orders are usually accumulated and placed in the oven at one time; however, any order shouldn't wait more than 150 minutes to be placed in the oven.
4. When dry, the orders are transported by conveyor to the packing area where they are inspected and prepared for shipment.

Operating times (in minutes; the varying times reflect the size distribution of the orders) can be found in Table 7.4.

Operation	Attribute – Minutes	Distribution
Order arrival rate	Mean = 30	Exponential
Cutting	Min 10; Max 20	Uniform
Assembly	Min 10; Max 20	Uniform
Heat Treatment	60	Constant
Inspection	Mean 20; Std 5	Normal

Table 7.4: Operating times.

Downtime occurs in a number of ways. The cutter and assembly operation both have frequent and infrequent downtimes. The frequent downtimes can be repaired by the operator. The operator will always stop the present job to repair the machine. The infrequent downtimes (changing a cutter blade and repairing the paint sprayer) have to be handled by the plant mechanic, who responds to breakdowns as they occur. The plant mechanic also has to fix problems in the other plant areas.

Operation	Time between failures (minutes)	Distribution	Time to repair (minutes)	Distribution	Repair Person
Cutter	Min. 20; Max. 80	Uniform	Min. 1; Max. 4	Uniform	Cutter Operator
Assembly	Min. 35; Max. 55	Uniform	Min. 1.2; Max. 3.0	Uniform	Assembly Operator
Oven	Mean 420; Std. Dev. 20	Normal	Mean 5; Std. Dev. 2	Normal	Mechanic
Cutter blade change	760	Constant	Min. 10; Max. 15	Uniform	Mechanic
Paint sprayer	Min. 400; Max. 560	Uniform	5	Constant	Mechanic
Other plant areas	Mean 100	Exponential	Min. 20; Std. Dev. 5	Normal	Mechanic

Table 7.5: Historical records for downtimes (minutes).

Material travel times

- Order entry to cutter: 3 minutes
- Cutter to assembly: 3 minutes
- Assembly to oven: 3 minutes
- Oven to packing: 5 minutes

Capacities

- Conveyor to cutter: 10
- Queue before cutter: 10

- Conveyor to assembly: 10
- Queue before assembly: 5
- Conveyor to oven: 10
- Conveyor to packing: 10

Expected results

- Run the simulation for 10 workweeks. Although in practice any incomplete orders would carry over to the next week, consider each week a separate run starting with a clean line.
- Report on production levels as well as how many orders are still in the system at the end of the week. The only operating personnel who need to be considered are the cutter operator, assembly operator, and plant mechanic.

Modeling and analysis issues

- What level of detail is needed? Should all chairs in an order and their parts be tracked?
- What should be the basic time unit?
- How can you start and stop the simulation?
- Although the combiner object can perform batching, is it a good choice here?
  - What other object can handle batching?
  - How can the max waiting time be handled?
- What is the best way to validate that the components are working correctly before adding the complexity of the downtime?
- How can the "rest of the plant" be simulated and given a downtime for the mechanic to fix?
  - Which downtimes will pre-empt other jobs?
  - How is the person to service the downtime selected?
- What performance measures are important?
- What can be done to improve operations so that all jobs are finished by 4 p.m. on Friday?
  - The union wants an additional mechanic added.
  - The cutter operator wants an automatic infeed from the cutter queue.
  - The lean team suggests changing the way orders are entered into the oven-even suggesting converting to a five-lane continuous oven.
- What are the issues with higher order rates and what can be done to improve operations?

Chapter 7 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important in the chapter.
2. Name two other forms of planned down time that can be simulated with a time table.
3. Describe three different performance measures, such as those listed in Section 7-2, including one that is not based on a rate measure, and give an example of each as applied to an operations system.
4. Discuss how does reliability can impact system performance.
5. Discuss the difference between "availability" and "reliability."
6. When including reliability in a simulation model, it is important to understand the definition of the word time in mean time between failures. Discuss why this is important in a simulation model.
7. Discuss the different times that can be referred to in mean time to repair.
8. Describe the different ways you can estimate MTBF and MTTR.
9. Explain what competing failures are and how they can be simulated.
10. Assume the reliability of a machine is given by

    $$R(t) = e^{-0.01t},\ where\ \lambda = 0.01$$

    that is, the failure rate is 1 per 100 hours.

    a. Describe in words the meaning of R(t).
    b. What is the probability that the machine will run 200 hours before failing?
    c. For this case does the answer change whether the previous failure occurred 10 hours ago or 300 hours ago?
11. Describe the two ways to enter reliability data in FlexSim. How would you set MTBF and MTTR for three machines that share the same distributions?
12. What does the appearance of a gold box around a piece of equipment signify?

13. Describe what happens when the Preempt box is checked for an operator who is being used to repair more than one piece of equipment.
14. Explain the basic concepts of using a surge to protect from down time.

Chapter 8 Modeling Randomness

Objectives

Since the inclusion of randomness is a major aspect of simulation modeling, describe how randomness is: (1) specified in simulations, i.e., how system knowledge and data are used to specify probability distributions and (2) realized in simulations, i.e., how simulations sample from probability distributions.

As should be apparent by this point in the textbook, probability distributions are used extensively in simulation models. They are used to represent the variability and randomness in customers' arrivals, the time to perform services, quality of operations performed, reliability of equipment, etc. In each of these cases, the system characteristic is considered a random variable and is defined by a probability distribution. As a simulation is performed, values must be repeatedly sampled from the distributions. This is one of the reasons simulations are performed by computer, since there are extensive computations involved in generating random samples. Of course, all simulation software, such as *FlexSim*, routinely perform the computations; however, it is useful to know how the samples occur within the software.

A probability distribution, which assigns probability to a random variable, is either continuous or discrete. A continuous distribution, referred to as a probability density function, and usually denoted as f(x), is used to represent a continuous random variable X, such as time and distance. A discrete probability distribution, referred to as a probability mass function, and usually denoted as p(x) is used to represent a finite or countable random variable X, such as the number of customers waiting for service or an indicator such as whether a server is in the busy or idle state.

A probability distribution is also considered either parametric or nonparametric. With a parametric distribution, a functional form is specified along with a set of the parameter(s). The function and parameter values determine a distribution's shape, degree of variability, skewness, etc. Common continuous parametric probability distributions are such theoretical distributions as the normal, exponential, Weibull, etc. Common discrete parametric proba-

bility distributions include such theoretical distributions as Bernoulli, binomial, Poisson, etc. These and other distributions are discussed extensively in basic statistics courses and thus are not described in detail here; for more information refer to an introductory textbook on engineering or business statistics. Nonparametric probability distributions typically do not use mathematical functions, but use sample data explicitly.

Probability distributions used in simulation models are specified either by: (1) analysis of sample data obtained from the system being modeled or (2) engineering judgment. In both cases, the best way to start is with a picture or plot of the density/mass function (histogram) to get a sense of the shape of the distribution, degree of variability, etc., to see if it is a feasible representation of the operation being modeled. Setting up a histogram of the data is described in Section 8-2.

This chapter begins with a brief discussion of the importance of probability distributions in simulation modeling and analysis. The chapter has two main parts that parallel the chapter's objectives—how to specify probability distributions in simulations and how simulations sample from the specified distributions. The first part discusses how continuous and discrete probability distributions are determined or selected for use in a simulation. This is covered in the first two sections. One section describes selecting distributions based on sample data obtained from the system being modeled; the other describes specifying the distribution without data. The second part of the chapter describes how random samples are obtained from probability distributions. This includes both sampling from continuous and discrete probability distributions and doing so randomly through the use of a random number generator. How the random samples drive a simulation model is discussed in the next chapter, both in terms of Monte Carlo simulation and discrete-event simulation.

Section 8-1 Importance of probability distributions in simulation

This section highlights the importance of probability distributions in the modeling and analysis of any system through simulation. Law [3] identifies two fundamental pitfalls in simulation input modeling: replacing a distribution by its mean and using the wrong distribution.Each time table can control many objects, and each object can be controlled by multiple time tables. The mode dropdown choices control how the data is viewed and repeated in the simulation. Note that filling in data under one mode selection does not automatically transfer to the other modes so only use one mode for each table. Complete information about time tables is included in the Help files.

Replacing a distribution with its mean removes all variability effects of that variable. Consider a simple single-server queueing system from Chapter 2 with exponentially distributed time between arrivals with a mean of one minute. If the service time is considered a constant 0.9 minutes, then the average wait time is 4.0 minutes; however, if the service time is exponentially distributed, with the same mean of 0.9 minutes, then the average customer wait time more than doubles to 8.1 minutes. In this case, just using the service process's mean time and not considering variability greatly underestimates a key performance measure, customer wait time. Such an underestimate may lead the decision maker to believe the system is performing adequately and thus not be amenable to making a change.

The second major pitfall identified by Law [3] is using the wrong distribution. The effect of incurring this pitfall is illustrated by a simple single-server queueing system example from Chapter 2. It considers the sensitivity of the effect of the assumed service time distribution on average customer wait time. Figure 8.1 provides excerpts from Figures 2.14 and 2.15 in Chapter 2. Both plots in Figure 8.1 are from a single server system with the time between customer arrivals being exponentially distributed with a mean of 1.0 minute.

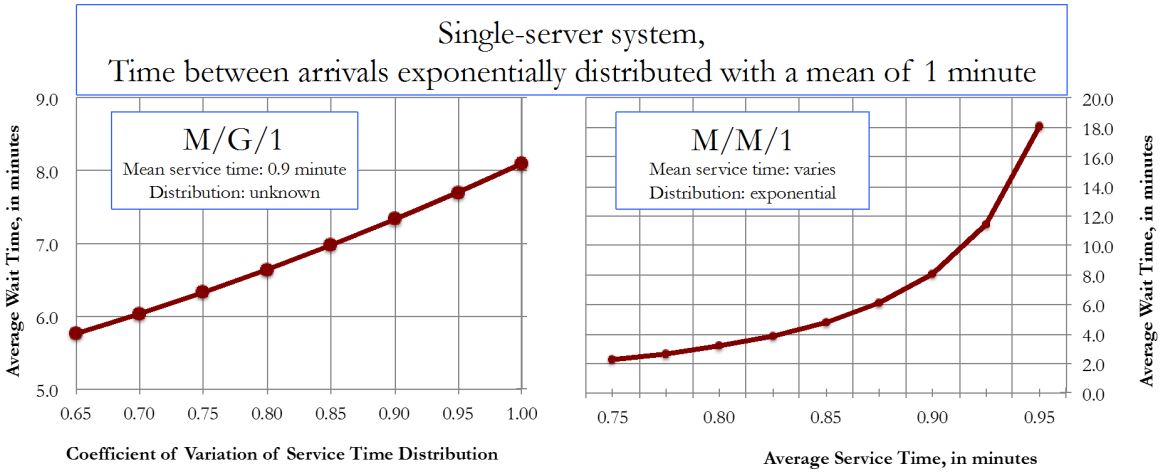


Figure 8.1: Effect of distribution parameter values on performance, M/G/1 system.

In the plot on the left-hand side of Figure 8.1, the mean service time is fixed at 0.9 minutes but its variability, as measured by the coefficient of variability (standard ÷ deviation mean), Cv, ranges from 0.65 to 1.0. Based on the plot, the average customer wait time varies from 6.0 minutes for Cv = 0.70 to 8.1 for Cv = 1.0.

In the plot on the right-hand side of Figure 8.1, the Cv is fixed at 1.0 (exponential distribution) and the mean service time ranges from 0.75 to 0.95. Based on the plot, the average customer wait time varies from about two minutes for a mean service time of 0.75 minutes to about 18 minutes for a mean service time of 0.95 minutes. Obviously there is a high degree of sensitivity to the mean of the exponential distribution—in this case, a 27% increase in the average service time leads to an 800% increase in average customer wait time!

The source of misspecifications of probability distributions oftentimes comes from making casual assumptions about a model's distributions. For example, many times modelers assume a random variable is exponentially distributed because it is often assumed in Queueing Theory. That assumption is used in Queueing Theory to make the modeling and computations simpler; using the most representative distribution in a simulation adds no complexity to the model (just another function name is used). The normal distribution is also assumed quite frequently, usually because most people are familiar with that distribution, and it is covered extensively in basic statistics courses. However, as Law [3] notes, the normal distribution oftentimes provides a poor fit to samples from systems being simulated. He goes on to offer that "it might be said with some truth that the greatest application of the normal distribution is writing statistics books."

The scarcity of attention to data analysis and trying to use the most representative probability distribution oftentimes are the result of modelers spending too much time on the model structure (incorporating more detailed logic and/or more refined graphics than are needed). Detailed logic cannot compensate for unrepresentative data and distributions. Of course the opposite case, using very representative probability distributions but insufficient logic, is a danger as well; however, this tends to not occur as often. In either case, model structure or model data, the adage "garbage in results in garbage out" applies.

One major factor, regardless of the method used to select a probability distribution for use in a simulation, is whether or not the distribution is bounded. If an unbounded distribution such as the normal (unbounded in both the positive and negative directions) or exponential (unbounded in the positive direction) is used, it must be done so with much care. Since simulation involves randomly sampling from probability distributions, it is possible, albeit unlikely, to sample a very extreme value that would be impossible in practice. For example, if process times are exponentially distributed with a mean of 10 minutes, then there is a 5% chance to sample a process time greater than 30 minutes and a 0.005% chance of sampling a process time greater than 100 minutes, or 10 times the mean. While it is not very likely for the latter to occur, if it does happen, it will likely greatly distort the system behavior and performance; therefore, it is recommended to use bounded distributions or at least check the sampled values and either resample or truncate them at some maximum practical value.

It is imperative that any simulation study provides the resources (time and expertise) to adequately analyze and specify the probability distributions that are used in a model. Misspecifying probability distributions will lead to erroneous estimated system performance that could lead to making a wrong decision based on the model.

Section 8-2 Selecting probability distributions based on sample data

When deciding what probability distribution to use to represent an aspect of a system, a common method is to collect sample data from the system and analyze it. The analysis may involve a simple visual assessment or a more rigorous statistical analysis of the data. There are two basic approaches to using sample data—parametric and nonparametric. A parametric approach uses the data to estimate the parameters of an assumed probability distribution function and then compare a variety of functions to determine which one fits the best. By fit, we mean the closeness the sample data resembles a probability function. A nonparametric approach uses the data explicitly and does suppose any functional form.

Using data to decide what distribution to use to represent a part of a system has many similarities to any type of data analysis. The larger the sample size the better the estimate; for selecting distributions, Law [2] recommends using at least 100 observations, while using 200 observations increases the user's ability to discriminate between two distributions. Law [2] also points out that the benefit of increasing the sample size from 200 to 300 is generally less than increasing the sample size from 100 to 200. While 100 to 200 observations is a good target, oftentimes in practice that much data are not available; therefore, you have to make the best of what you have. It is better to use and analyze what data you have than to skip the analysis because of the small sample size.

Another important preliminary assessment of the data involves reviewing it for outliers. Outliners are observations whose values are extreme or far from the values of the other observations. The extreme values need to be investigated to see if they are erroneous and thus need to be eliminated. Outliers can skew the results, affecting the choice of distribution family and parameter values such as the mean, standard deviation, etc.

All of the computer examples in this section use *ExpertFit*, input data analysis software developed by Dr. Averill Law. *ExpertFit* is a standalone product, but comes bundled with *FlexSim*. It is accessible through the Statistics option on the main *FlexSim* menu. Further information on the software, please consult the *Expert Fit* Version 8 User's Guide [2]. In addition to performing the appropriate statistical tests, the software and manual provide guidance on all aspects the distribution selection process. For additional information, see Chapter 7 in Law [1] and a Winter Simulation Conference tutorial Law [3].

Whether using a parametric or nonparametric approach, the best place to begin the analysis of the sample data is with a visual assessment. One of the best approaches is to construct a histogram. A histogram groups the data into "bins," or intervals of equal length, and displays the count or frequency of the number of observation in each bin. One downside of a histogram is that it does not convey all of the information in the sample; the individual data values are replaced with a frequency count of the number of observations within an interval.

Figure 8.2 provides a set of 50 observations that is a random sample of operation times and is used as an example to illustrate the methods for selecting a probability distribution to represent some aspect of a system in a simulation model. These observations are actually a random sample from a population that is normally distributed with a mean of 10 and a standard deviation of 3. The sample was generated by *MS Excel*. Of course, in practice the true population distribution is never known; and if we did know the true population distribution, we would just use it and not be concerned with sampling. As we will see in the analysis below, the sample does not appear to be from a normal distribution; therefore, it is important to note that with any statistical analysis based on sample data, the inference may or may not be correct. That is, there is always a risk associated with not examining the entire population; however, most of the time examining the entire population is impractical or impossible.

1.25	7.51	14.20	9.85	7.00	5.76	12.81	9.62	8.78	12.62
6.91	13.32	5.95	4.69	11.49	11.24	7.29	7.21	9.45	12.08
11.84	12.61	9.32	5.68	7.85	5.47	12.35	5.89	6.57	11.73
10.95	13.65	10.58	8.22	12.14	9.28	6.10	13.68	7.67	11.40
12.02	8.52	9.46	12.80	9.35	11.69	14.01	11.28	3.63	11.18

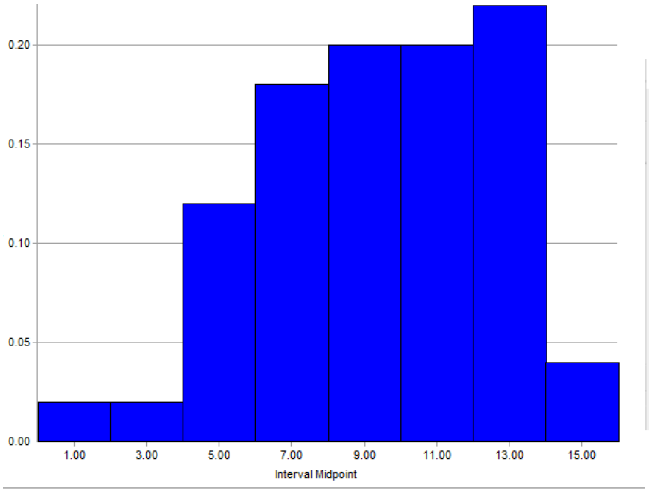
Figure 8.2: Sample of process times (n = 50) for distribution selection analysis.

The basic steps for constructing a histogram are:

1. Determine the number of bins or intervals, *k*. There is no best method, but a common and simple rule of thumb is to use the square root of the sample size,

n; i.e. $k = \sqrt{n}$. Since k needs to be an integer, round up. In the above example, $k = \sqrt{50} = 7.07 \rightarrow 8$.

2. Determine the interval width, w. Law[3] recommends selecting the smallest interval that gives a reasonably smooth histogram. One common approach is $w = \frac{max - min}{k}$, where max and min are the largest and smallest data values, respectively. For visualization, w is oftentimes set to a whole number, but given the range of the data, that is not always possible. In our example, $w = \frac{14.20 - 1.25}{8} = 1.62 \rightarrow 2$.
3. Determine the starting point of the histogram, s. Typically the starting point is the minimum data value or smaller and is a whole number, but doesn't need to be. In our example, the minimum value is 1.25 so we could use s=1 or even s=0. We choose s=0.
4. Set up the bins. Bin 1 ranges from s up to s + w (referred to as upper limit 1 or UL1). Bin 2 ranges from UL1 up to UL1+w, etc. In our example Bin 1 includes observations with values from 0 up to 2.00, Bin 2 observation values 2.00 up to 4.00, etc. The bins are mutually exclusive and collectively exhaustive; in other words, a data value will be in one and only one bin. Since the precision of the example data is to two decimal places, we could denote the bins as 0 – 1.99, 2.00 – 3.99, etc.
5. Place each data value into a bin. Reading across the first row of the data in Figure 8.2, the value 1.25 is in Bin 1, 7.51 is in Bin 4, 14.20 is in Bin 3, etc.
6. Count the items in each bin and construct a bar plot as in Figure 8.3. In this plot the bin midpoints are shown, as opposed to their endpoints. In this case, the 50 data values are replaced by eight bin frequencies. Figure 8.3 also includes a summary of the data in terms of basic descriptive statistics (e.g., mean, median, and variance).



Data-Summary Table

Data Characteristic	Value
Source file	<edited>
Observation type	Real valued
Number of observations	50
Minimum observation	1.25100
Maximum observation	14.19900
Mean	9.51914
Median	9.54000
Variance	9.15778
Coefficient of variation	0.31790
Skewness	-0.48778

Figure 8.3: Histogram and descriptive statistics for the sample data.

In the case of the example, it is difficult to discern the underlying distribution based on the plot, but at least it appears to be unimodal (one peak) and there are not any obvious outliers. Had we seen a value of 30, for example, we would want to investigate that data

point; it may be an error, or it might be that the system has occasional large values. Increasing the sample size, if possible, might help address this issue. Also, the closeness in values of the mean and median indicates symmetry, but the skewness measure indicates moderate asymmetry. Of course, as mentioned earlier, the data are just a sample from a typically unknown population; therefore, there is always risk in our inference.

Parametric approach

To specify a parametric or theoretical probability distribution for use in a simulation, one must identify the family, such as the normal or exponential, and then the parameters of the distribution (e.g., mean and standard deviation of the random variable for the normal distribution, mean of the random variable for the exponential distribution).

There are many ways to use the sample data to decide the parameter values for probability distribution functions. Only a few are presented here; for a fuller discussion, see Law [1].

Data analysis

1. Heuristic (judgment based on histogram)
2. Goodness of Fit test
3. Difference plot

All of these methods are basically quite similar: their objective is to compare various probability distributions to the data displayed in the histogram and to select the best one. The main differences are the degree of formality or rigor used and the amount of information that is used from the data to assess the fit. In the heuristic or judgment approach we mentally overlay various distributions on the histogram. For example, based on the histogram in Figure 8.3 we would not expect the uniform distribution, with all possible values of the random variable equally likely, to be a good choice; its density function would not match the histogram's pattern very well.

Figure 8.4 provides an overlay of a variety of theoretical probability distribution density functions. Most seem to fit reasonably well except the uniform distribution.

Goodness-of-fit testing from statistics provides a more rigorous assessment of the fit between the various distribu-

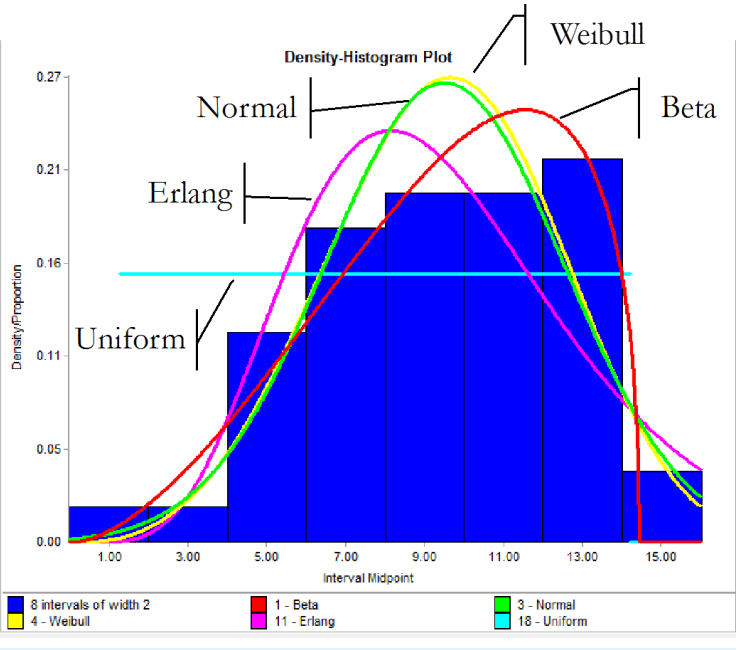


Figure 8.4 Visual comparison of sample data and theoretical parametric distributions

tions and the data. Goodness-of-fit testing is analogous to making a visual comparison on how well different theoretical distributions fit the sample data, but it is a more formal and rigorous approach since it is grounded in statistics. There are a number of statistical methods available for testing goodness of fit, such as the Chi-Squared test, Anderson-Darling test, and Kolmogorov-Smirnov test. The Chi-Squared test uses grouped data, like a histogram, whereas the other two use all of the observation. Details of these and other tests are found in basic engineering and business statistics books. The calculations involved in these tests are not difficult, but they are tedious. Therefore, such analyses are typically performed by software (e.g., *ExpertFit*).

Another approach for assessing the fit of a distribution to sample data is to plot the differences between the sample and hypothesized cumulative distributions over the range of the data. An example of such a plot, referred to as a Distribution-Function-Differences Plot, is shown in Figure 8.5. It is for the example sample dataset and the Beta and Normal theoretical distributions. If a distribution fits the data perfectly, which rarely occurs, then the plot would be horizontal line at 0. The greater the deviations from 0 in the plot, then the worse the distribution fits the data. For the sample data in our example, there seems to be some disagreement towards the upper end of the dataset. Also, the Beta generally fits better than the Normal; however, since the plots for both distributions are within the control limits and are mostly close to zero, either would provide a good representation of the data.

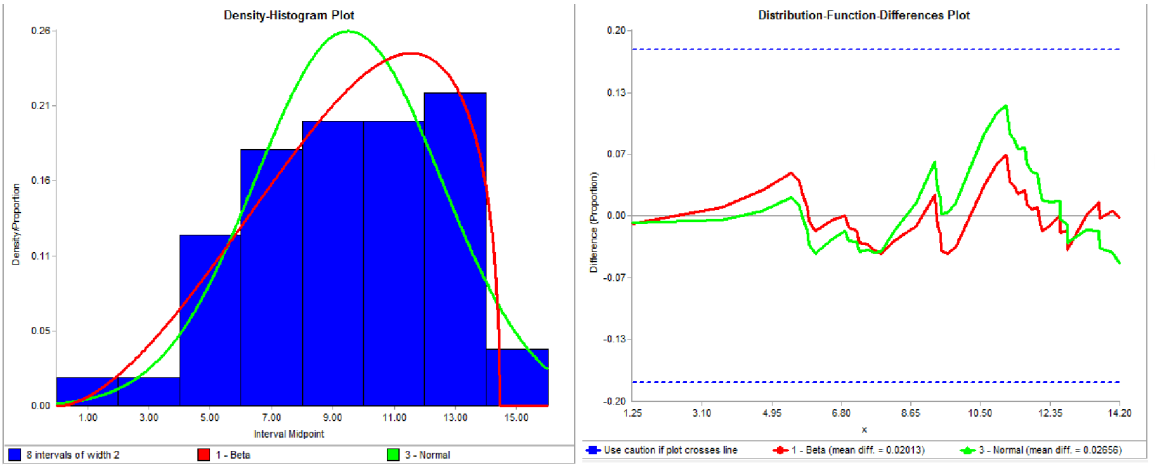


Figure 8.5 Comparison of sample data and parametric distributions using a Distribution-Function-Differences Plot.

A variety of statistical methods are used by the *ExpertFit* software to assess the fit of various distributions to sample data. The software (1) performs a variety of statistical tests on the sample data for each of many possible distributions, (2) combines the test results for each distribution, and (3) provides a rank and relative score for each distribution. Figure 8.6 shows *ExpertFit's* top 3 distributions for representing the sample data that was defined above. It provides the type of distribution and the estimated parameters for that distribution. *ExpertFit* also provides the syntax for using each distribution in *FlexSim*; Figure 8.7 provides that information for the sample data.

Relative Evaluation of Candidate Models

Model	Relative Score	Parameters	
1 - Beta	98.00	Lower endpoint	0.04303
		Upper endpoint	14.45215
		Shape #1	2.73506
		Shape #2	1.43259
2 - Johnson SB	98.00	Lower endpoint	0.11269
		Upper endpoint	14.93477
		Shape #1	-0.63629
		Shape #2	0.95206
3 - Normal	90.00	Location	9.51914
		Scale	3.02618

26 models are defined with scores between 0.00 and 98.00

Absolute Evaluation of Model 1 - Beta

Evaluation: Good
Suggestion: Additional evaluations using Comparisons Tab might be informative.
See Help for more information.

Additional Information about Model 1 - Beta

"Error" in the model mean
relative to the sample mean 0.01997 = 0.21%

Figure 8.6: Evaluation of possible theoretical parametric distributions to represent the sample data.

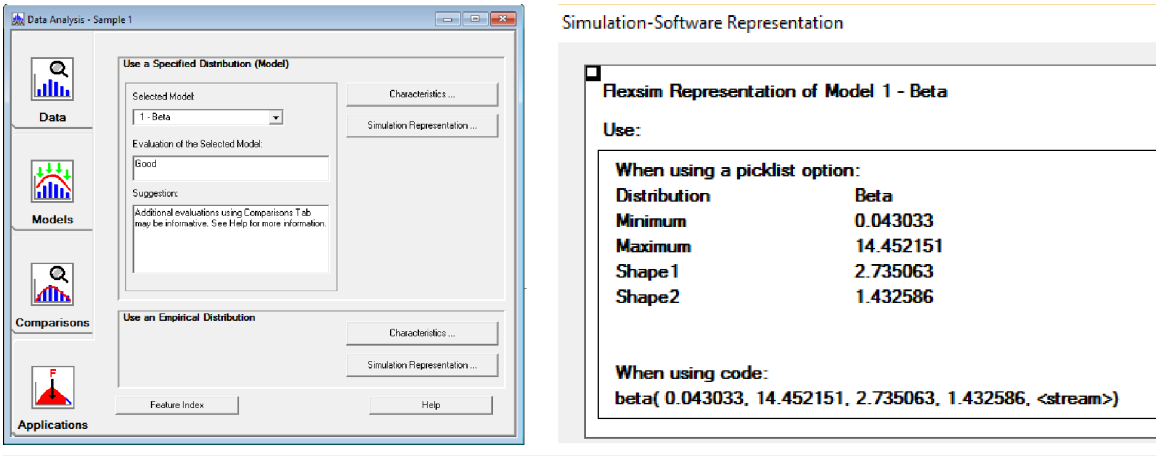


Figure 8.7 Representation of a selected parametric distribution (Beta) in *FlexSim*.

It is important to note that most parametric probability distributions are unimodal; that is, they have one peak. The analysis of data that appear to have multiple modes, such the histogram in the left-hand portion of Figure 8.8, will typically fail any parametric probability distributions test. In that case, an empirical distribution will typically be used.

If the data appears to be multimodal, however, then a deeper analysis of the data may present a better approach to represent the operation in a simulation model. A common characteristic of many operations is that the items they produce are of different types. If the process times are significantly different based on product type, then the data can be separated by type and analyzed separately. In the right-hand portion of Figure 8.8, the lighter colored bars are for a different type than the darker bars. The sample of 40 items in Figure 8.8 is actually a composite of two types of items: 25% of the items have process

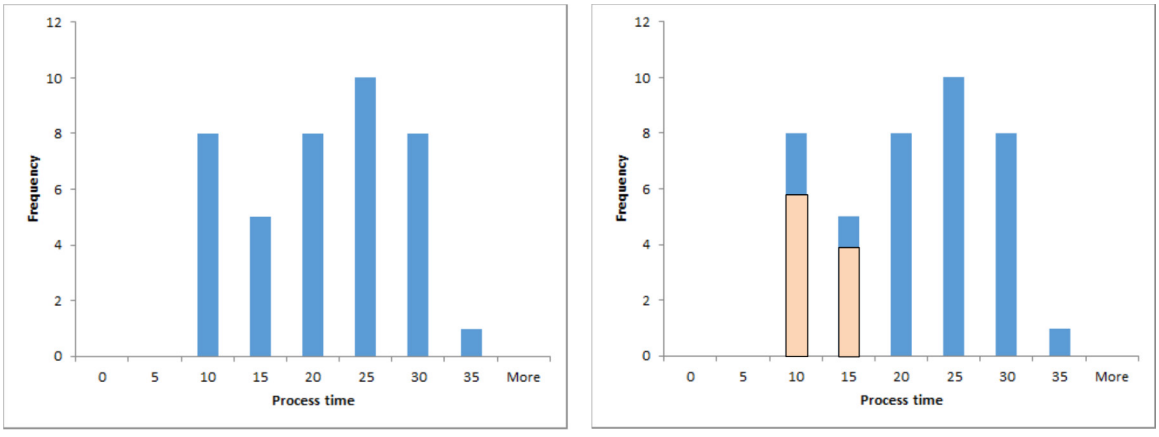


Figure 8.8: Bimodal process time distribution that is composed of two different types.

times that are normal(10,3), and 75% have process times that are normal(20,5). In this case, the two types of process times are from the same family but with different parameters; however, it is possible that the families might be different for the different product types.

While the above discussion and example have involved continuous probability distributions, the same analysis process is followed for discrete parametric distributions, such as the Poisson, Binomial, etc.

Distribution fitting using the parametric approach can also be used to estimate distributions of the output from a model, such as lead time, inventory level, service rate (probability customers wait more than a specified threshold), etc. For key performance measures, distributions are much more informative than point or interval estimates.

Nonparametric approach

If it is decided no parametric distribution fits the data adequately, or if the modeler prefers to sample from the collected system data and not from a theoretical distribution, then a nonparametric approach is used. This is often referred to as an empirical distribution, with the most common being a piecewise linear approximation of the cumulative probability distribution.

The cumulative distribution, or alternatively called the distribution function, is the area under the probability density function. It is also the probability that the random variable X has a value less than or equal to a specified value x, $P(X \leq x)$. The cumulative distribution values range between zero and one. Figure 8.9 illustrates the relationship between a standard normally-distributed probability density function f(x) and its cumulative or distribution function F(x).

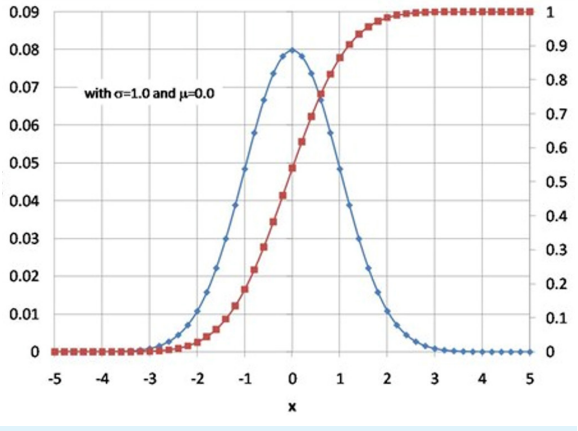


Figure 8.9: Standard normal distribution density and distribution functions

The density function is the bell-shaped curve and refers to the left-hand axis. The cumulative or distribution function is the S-shaped curve and refers to the right-hand axis. A standard normal distribution has a mean of zero and a standard deviation of one.

Figure 8.10 shows the empirical distribution for the example 50-observation sample data from Figure 8.2. The plot is constructed by sorting the data values in ascending order and assuming each is given an equal chance of occurring; in this case 1/50 = 0.02. The lowest value, 1.25, is plotted at 0.02, the next lowest value, 3.63, is plotted at 0.04, etc. Values between each data point are assumed to be equally likely. Empirical distributions for continuous variables are implemented in *FlexSim* via the **cempirical** command and an associated global table. This is illustrated in the next part of the example.

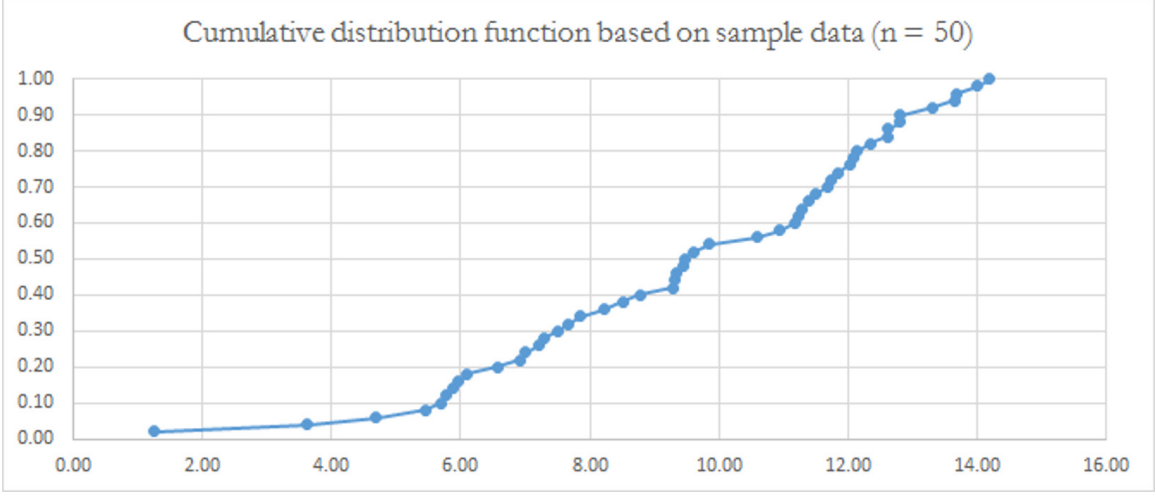


Figure 8.10: Sample data as an empirical distribution.

The histogram can also be used to represent a sample as an empirical distribution, instead of using all of the individual data values. Table 8.1 shows the intervals and frequencies used for the histogram of the sample data that was presented in Figure 8.3.

Interval		Frequency		
Lower limit	**Upper limit**	**Number**	**Percent**	**Cumul. percent**
0	2	1	2	2
2	4	1	2	4
4	6	6	12	16
6	8	9	18	34
8	10	10	20	54
10	12	10	20	74
12	14	11	22	96
14	16	2	4	100

Table 8.1: Histogram information for the sample data.

In *FlexSim*, the histogram information in Table 8.1 can be used as a continuous empirical distribution through the **cempirical()** function and the global table shown in Figure 8.11. By using **cempirical("MyEmpirical_1")** the function implements the distribution defined in the user-created global table named "MyEmpirical_1". Each row in the table corresponds to an interval in the histogram and

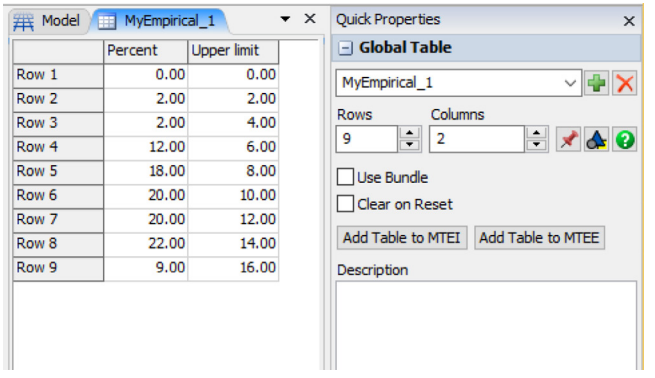


Figure 8.11 Global table used to implement a continuous empirical function based on a histogram.

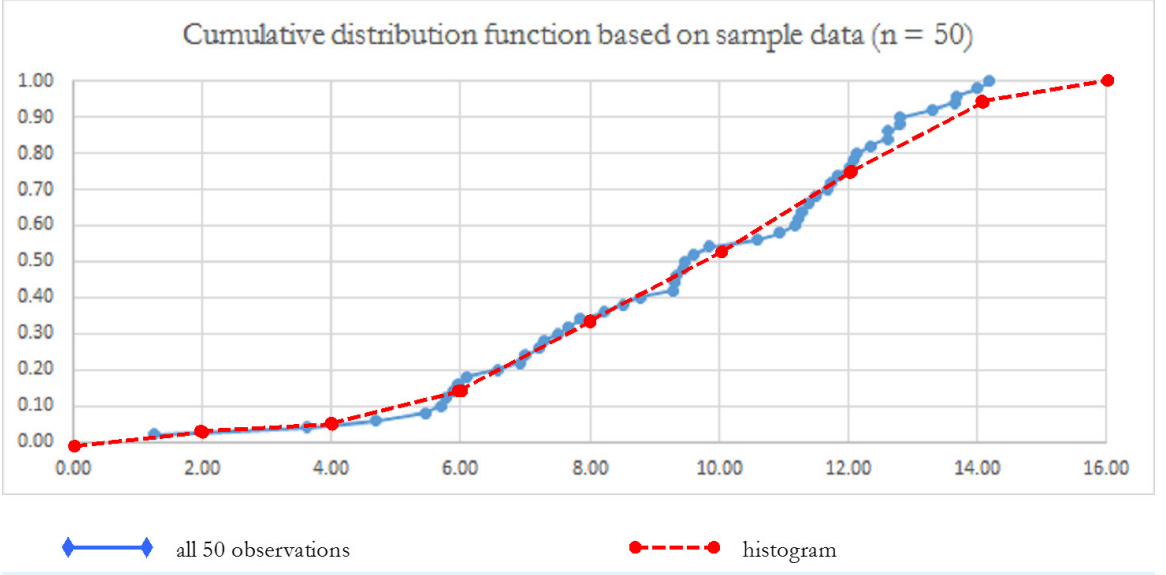


Figure 8.12: Sample data's empirical distributions based on (1) all observations and (2) histogram.

the interval's corresponding percentage frequency. The frequency is in Column 1, and the upper limit of the interval is in Column 2. Note that an extra row, with values 0 and 0 are added at the beginning of the table. The table implies 2% of the values in the distribution are from 0 up to 2, 2% from 2 up to 4, etc.

The two empirical distributions of the sample data are shown in Figure 8.12, one based on all observations, the other based on the histogram. As you can see, the distributions are quite similar.

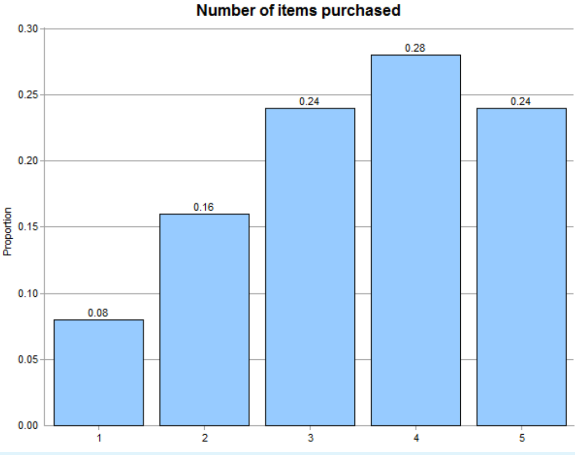


Figure 8.13: Sample discrete data – number of items purchased.

Discrete data can be analyzed in a manner similar to that used for the continuous case. Consider an example where the number of items purchased by customers in a retail store is a discrete random variable. A sample of 25 purchases is summarized in the histogram in Figure 8.13. In this case, 8% of the customers bought one item, 16% bought 2 items, etc. As shown in Figure 8.14, the sample data are analyzed in *ExpertFit*, and the best fit to the data is with the binomial probability distribution. The figure also compares, via a histogram, the sample data and the frequencies if the binomial distribution is assumed. The binomial function in *FlexSim* would be used to implement the recommended distribution in a model. Based on the parameters in Figure 8.14, the specific implementation would be **binomial(5, 0.688)**. This is an example of using the parametric approach with discrete data.

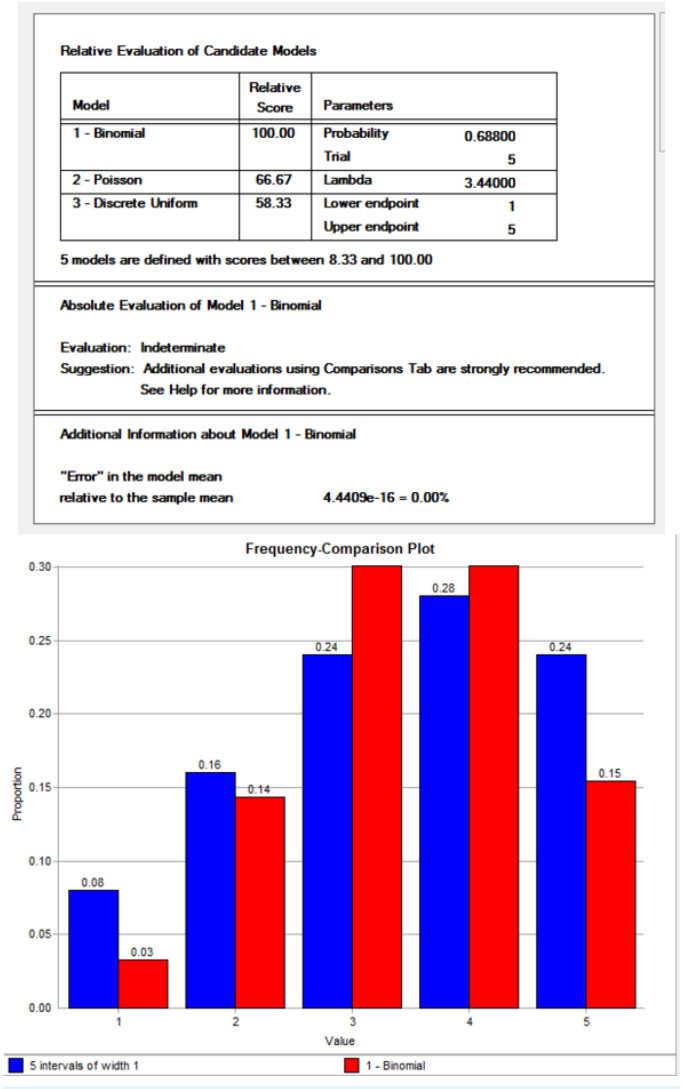


Figure 8.14: Results of *ExpertFit* analysis and comparison of sample data and the binomial distribution.

If the modeler prefers to use the sample data and not the binomial distribution, then the distribution would be specified using a nonparametric approach through a discrete empirical distribution. In *FlexSim* the probability distribution would be implemented through the **dempirical()** command and the mass function of the sample data would be stored in a global table, as shown

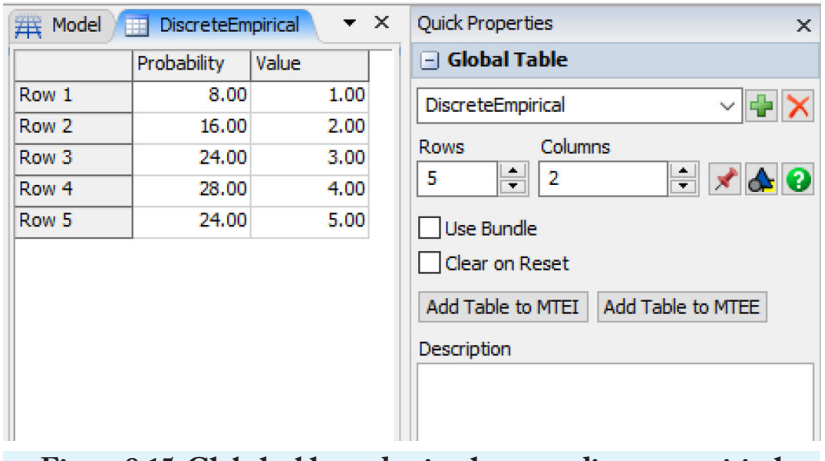


Figure 8.15: Global table used to implement a discrete empirical function based on a histogram.

in Figure 8.15. By using **dempirical("DiscreteEmpirical")** the function implements the distribution defined in the user-created global table named "DiscreteEmpirical". Each row in the table corresponds to a possible discrete outcome and its corresponding percentage frequency. The frequency is in Column 1 and outcome is in Column 2. The table implies 8% of the customers buy one item, 16% buy two, etc.

In general, it is better to use theoretical parametric distributions rather than empirical distributions because theoretical distributions are more general. For example, in simulations with empirical distributions no values will be sampled outside of the observed range. Since empirical distributions are based on sample data, they may not capture some of the more extreme values that might exist in the system. Also, theoretical distributions tend to "smooth" the data.

Section 8-3 Selecting probability distributions in the absence of sample data

There are situations when system data are not available; for example, the system may not yet exist. In those cases, engineering judgment based on knowledge of the proposed system or comparable systems may be used to subjectively specify a probability distribution.

Either a parametric or nonparametric approach can be applied. In the parametric case, a distribution would be specified and its parameters estimated based on judgment or comparable systems but in the absence of data from the system being modeled. In the nonparametric case, an empirical distribution could be formulated, again based on judgment or knowledge of comparable systems but in the absence of data from the system being modeled.

A common approach is to use a generally flexible distribution with easy-to-estimate parameters. One such distribution that is the triangular distribution. The density function is shown in Figure 8.16. The plot of the density function is generated by *ExpertFit's* Distribution Viewer, a handy utility to visualize a probability distribution based on specified parameter settings. The parameters needed to specify the triangular distribution are the minimum, maximum, and most likely values. These are usually easier to specify than the mean and standard deviation in the case of the normal distribution. Typically, the parameters can be set by discussing the operation being modeled with domain experts—those that perform or supervise the operation.

Depending on the values of the parameters, the triangular distribution can be skewed or symmetric. In the case of Figure 8.16 the minimum, maximum, and most likely values are 2, 9, and 4, respectively. If the distribution is used to represent processing times, then its use assumes that process times will not be less than 2 minutes or more than 9 minutes, with the mode or most likely value being 4 minutes. The mean or average

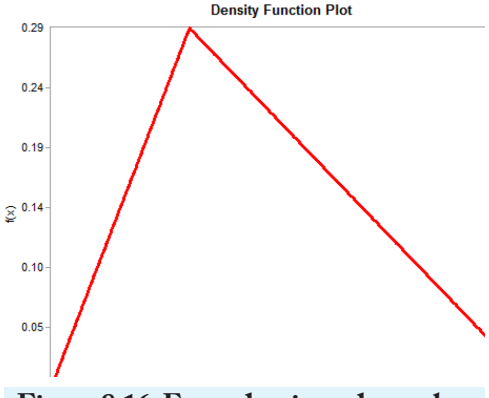


Figure 8.16: Example triangular probability distribution.

process time is 5 minutes; in general, the mean of a triangular distribution is the average of its three parameters.

Other distributions, such as the Weibull and lognormal, are used to model continuous distributions when data from the real system are not available. Since it is oftentimes difficult to specify the maximum value, those distributions allow the upper end of the distribution to specified by percentiles rather than using an absolute maximum. The three distributions —triangular, lognormal, and Weibull—are compared in Figure 8.17. All have the same minimum, and approximate mode and 90th percentile values of 2.0, 4.0, and 7.1, respectively. All have approximately similar means (5.0) and standard deviations (1.5). Of course, the triangular has an upper bound of 9.0. The lognormal and Weibull have infinite upper bounds; therefore, it is possible to obtain a large value in a random sample, although it is not very likely.

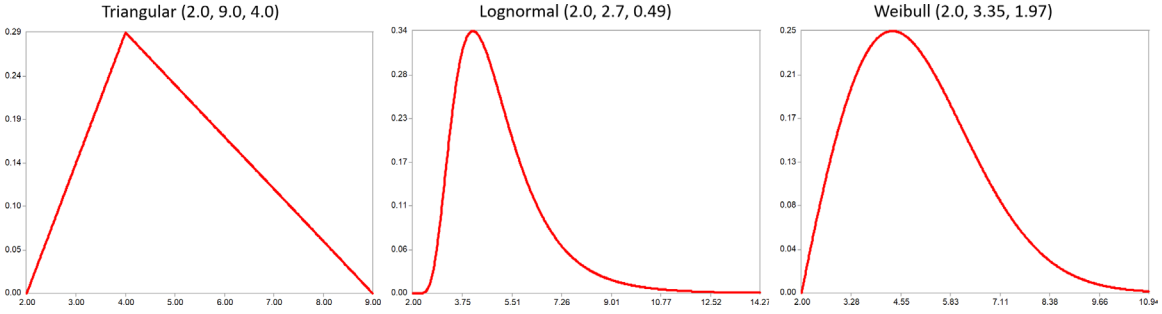


Figure 8.17: Comparison of similar distributions.

One downside to the lognormal and Weibull distributions is that their parameters (location, scale, and shape) are less intuitive to the layperson than those of the triangular (minimum, maximum, most likely).

Whenever data are not available to specify a probability distribution, it is advisable to perform sensitivity analyses on the uncertain distributions in order to identify which ones significantly impact the results from the simulation model. The distributions that have high impact need further scrutiny since they could affect the decisions that are being made based on the simulation model results. Not all inputs to a simulation model have the same impact on output; therefore, sensitivity analyses are used to identify the drivers (the parameters that are the most sensitive) and to direct further investigation into the value of the parameters for that variable. Time and resources are always scarce in a simulation project; therefore, efforts should be directed at performing better estimates only on those parts of the model that have the most impact on results.

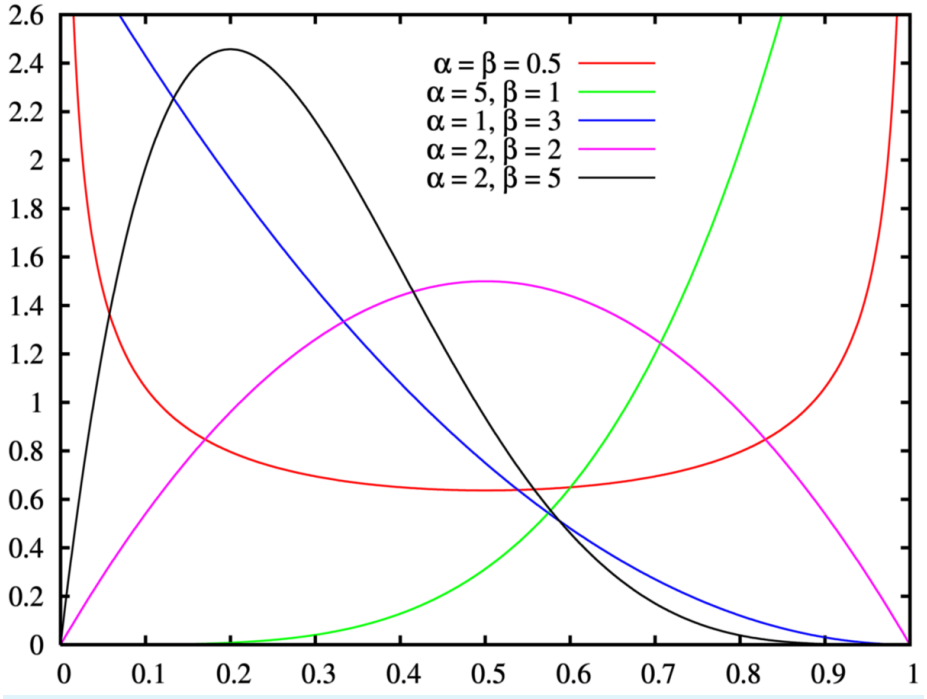


Figure 8.18: Various shapes of the Beta distribution (Source: [5]).

The Beta distribution is another theoretical probability distribution that is used to represent processes in the absence of data. As shown in Figure 8.18 it uses two shape parameters to generate a wide variety of shapes. The Beta distribution is bounded so sampling from it will not results in extreme values. Recall the Beta distribution was selected to represent the data from the example in Section 8-2.

Section 8-4 Sampling from Continuous Probability Distributions

As described throughout this book, probability distributions, both continuous and discrete, are used to model randomness in simulations. Every simulation involves obtaining random samples from one or more probability distributions. This section describes sampling from continuous distributions, such as the normal, uniform, exponential, Erlang, etc. These distributions are used to characterize continuous random variables such as interarrival times, service times, times between failures, etc. The next section describes sampling from discrete distributions. How to decide what distributions to use in a simulation model was discussed earlier in this chapter.

As defined earlier, continuous probability distributions are characterized by their density function, which describes the frequency of the values the random variable can take on, much as a histogram does for data analysis. An example density function, denoted as f(x), is shown in the left-hand portion of Figure 8.19. The example density function is for the normal distribution; the shape of the density is why it is sometimes referred to as the "bell-shaped curve." The mean of the distribution is denoted as μ. The normal distribution is symmetrical, so half of the possible values lie above μ and half are below. Also, most of the

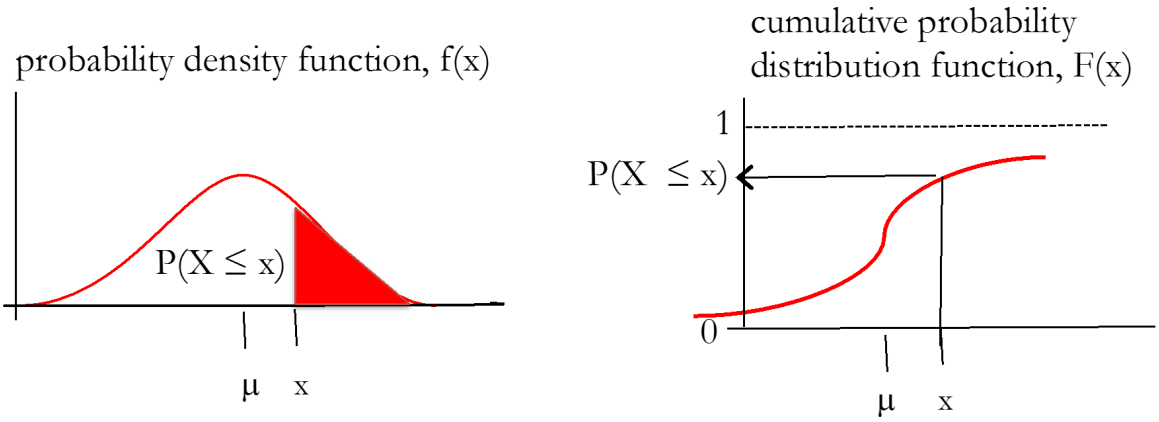


Figure 8.19: Probability distribution's density and distribution (cumulative) functions.

values will occur near the mean, hence the peak around μ; some, but few, values are expected at the upper and lower extremes.

The cumulative probability distribution, denoted as F(x) is the area under the probability density function. For continuous distributions, F(x) is as defined by the integral shown below and has the general form of the function plotted in the right-hand portion of Figure 8.19. F(x) is also referred to as the probability distribution function and is interpreted as P(X ≤ x), the probability that the random variable X is less than or equal to a specified value x. For any probability distribution, F(x) will always be between 0 and 1. For some functions, such as the normal distribution's density function, f(x) cannot be integrated; therefore, an approximation of F(x) is needed.

$$F(x) = \int_{l.l.}^{x} f(w)\, dw \qquad 0 \leq F(x) \leq 1$$

The most common, and probably the easiest and most intuitive, method for sampling from probability distributions in simulation is the Inverse Transformation Method. It uses the cumulative probability distribution, F(x), and exploits the fact that the range of F(x) is always between 0 and 1. The basic process is as follows.

1. Randomly choose a number between 0 and 1. The uniform distribution, denoted as uniform(0,1), is used to generate u_i where $0 \leq u_i < 1$.
2. Set $F(x_i) = u_i$
3. Solve for x_i. x_i becomes a random sample from f(x).

The approach is demonstrated using two common continuous probability distributions, the uniform and the exponential.

Density and distribution functions for the continuous uniform distribution are provided below and are shown in Figure 8.20. The figure also provides the formulas for calculating the mean and standard deviation of the distribution based on its parameters. The uniform distribution is used to represent situations in which all possible values over a specified range are equally likely. The values range between the specified lower and upper limits, α and β, respectively.

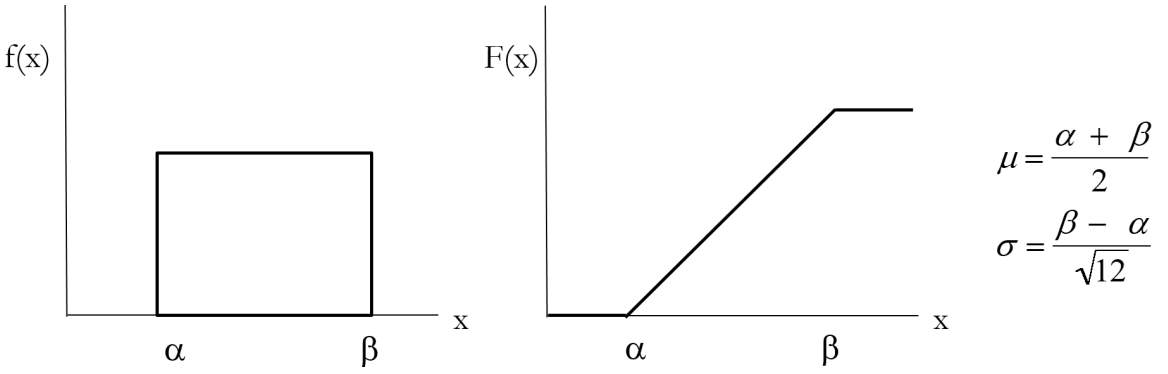


Figure 8.20: Uniform probability distribution's density and distribution functions.

The density and distribution (cumulative) functions are

$$f(x) = \frac{1}{\beta - \alpha}$$

$$F(x) = \int_{l.l.}^{x} f(w)\, dw = \int_{\alpha}^{x} \frac{1}{\beta-\alpha}\, dw = \frac{w}{\beta-\alpha}\Big|_{\alpha}^{x} = \frac{x-\alpha}{\beta-\alpha}$$

Since the density function can be integrated and is given by F(x), the inverse transformation method can be applied and the basic process defined above can be used.

1. Randomly choose a number between 0 and 1. The uniform distribution, denoted as **uniform(0,1)**, is used to generate u_i where $0 \le u_i < 1$.
2. Set $F(x_i) = u_i$,

$$F(x) \equiv u \;\Rightarrow\; \frac{x_i - \alpha}{\beta - \alpha} \equiv u_i$$

3. Solve for x_i. x_i becomes a random sample from a uniformly distributed random variable,

$$f(x) \sim uniform(\alpha, \beta)$$
$$x_i = \alpha + (\beta - \alpha)u_i$$

For example, assume travel times are uniformly distributed between 100 and 300 seconds; that is, α=100 and β=300. In order to get a random sample from the uniform(100, 300) distribution, a uniform(0,1) random number is generated (more on how that is done later), say u_1=0.67306, and applied to the equation above, as seen in the following:

$$x_1 = \alpha + (\beta - \alpha)\, u_1 = 100 + (300 - 100)\, 0.67306 = 234.6$$

As discussed in Chapter 2, the exponential probability distribution is widely used in Queueing Theory. It is also common in simulation. The exponential distribution's density and distribution functions are provided below and are shown in Figure 8.21. Based on its shape and negative exponent, it is actually the negative exponential distribution. The exponential distribution represents the situation where there are a high frequency of low values and a low frequency of high values. The only parameter required to specify the exponential is its mean (β); that is, the density implies that the random variable X is

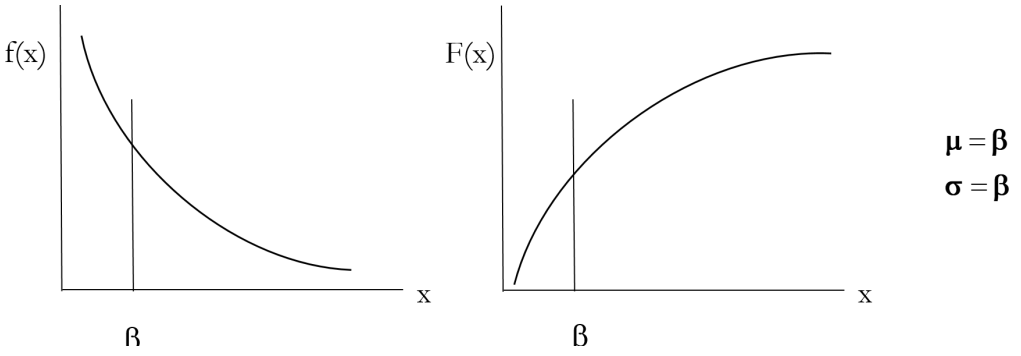


Figure 8.21: Exponential probability distribution's density and distribution functions.

exponentially distributed with mean (β), X~exp(β). It is a very highly variable distribution since its standard deviation is equal to its mean, β.

The density and distribution (cumulative) functions are

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

$$F(x) = \int_{l.l.}^{x} f(w)\, dw = \int_{0}^{x} \frac{1}{\beta} e^{\frac{-w}{\beta}}\, dw = -\frac{1}{\beta} e^{\frac{-w}{\beta}} \Big|_{0}^{x} = 1 - e^{\frac{-w}{\beta}}$$

Since the density function can be integrated and is given by F(x), the inverse transformation method can be applied and the basic process defined above can be used.

1. Randomly choose a number between 0 and 1. The uniform distribution, denoted as **uniform(0,1)**, is used to generate u_i where $0 \le u_i < 1$.
2. Set $F(x_i) = u_i$,

$$F(x) \equiv u \;\Rightarrow\; 1 - e^{\frac{-x_i}{\beta}} \equiv u_i$$

3. Solve for x_i. x_i becomes a random sample from an exponentially distributed random variable, f(x) ~ exponential(β).

$$f(x) \sim exponential(\beta)$$
$$x_i = -\beta ln\,(1 - u_i)$$

For example, assume travel times are exponentially distributed with a mean of 200 seconds; that is, β=200. In order to get a random sample from the **exponential(200)** distribution, a **uniform(0,1)** random number is generated (more on how that is done later), say u_1=0.67306, and applied to the equation above, as seen in the following:

$$x_1 = \beta ln\,(1 - u_1) = -200 ln\,(1 - 0.67306) = 223.6$$

The inverse transformation method can only be used if F(x) exists. For many distributions—such as the uniform and exponential illustrated above—the density function can be integrated to obtain the cumulative distribution. This is not the case for other distributions, such as the normal; therefore, other methods must be used to sample from these distributions. The inverse transformation method is considered a direct method; other direct

methods are the composition and convolution methods. The acceptance-rejection method is an example of an indirect method. For more information on these methods, see Law [1].

Table 8.2 provides a list of the typical methods used by simulation software to sample from some of the common probability distributions. Again, for more information, see Law [1].

Distribution	Method
Uniform	Inverse transformation
Exponential	Inverse transformation
Erlang	Inverse transformation & composition
Gamma	Multiple methods
Weibull	Inverse transformation
Normal	Multiple methods
Beta	Acceptance-Rejection
Lognormal	Multiple methods
Triangular	Inverse transformation

Table 8.2 Methods to sample from probability distributions

Section 8-5 Sampling from Discrete Probability Distributions

The concept for sampling from discrete distributions is the same as that for continuous distributions as described in the previous section. The process is illustrated through an example for an empirical distribution. Assume the probability of the number of items purchased by customers in a local store is defined as in Table 8.3. That is, 10% of the customers are expected to buy one item, 15% are expected to buy two items, 40% are expected to buy three items, etc.

x	p(x)
1	0.10
2	0.15
3	0.40
4	0.25
5	0.10

Table 8.3: Discrete probability distribution for the number of items purchased.

In order to sample from the distribution in Table 8.3, the cumulative probability distribution is used to set up ranges for the random numbers. This is analogous to the cumulative distribution function for continuous functions. The cumulative distribution sets the range of random numbers for each row. As shown in Table 8.4, if a **uniform(0,1)** random number

is between 0.000 and 0.099, then the simulation will have that customer purchase one item. Similarly, if the **uniform(0,1)** number is between 0.100 and 0.249, then the simulation will have the customer purchase two items. Using this approach, in the long run in the simulation, 10% of the customers will purchase one item, 15% will purchase two items (alternatively, 25% will purchase two or less items), etc.

x	p(x)	P(x)	RN
1	0.100	0.100	0.000 to 0.099
2	0.150	0.250	0.100 to 0.249
3	0.400	0.650	0.250 to 0.649
4	0.250	0.900	0.650 to 0.899
5	0.100	1.000	0.900 to 0.999

Table 8.4 Sampling from a discrete probability distribution

In a simulation, given a **uniform(0,1)** random number (as discussed in the next section), the simulation generates values for the discrete random variable via table lookup. For example, using the table above, if $u_1 = 0.67306$, then $x_1 = 4$. Similarly, if $u_2 = 0.25107$, then $x_2 = 3$ and if $u_3 = 0.86151$, then $x_3 = 4$.

Section 8-6 Generating Random Numbers

In the previous sections it was taken for granted that the simulation software could generate random numbers that are uniformly distributed between 0 and 1 (i.e., **uniform(0,1)**). These random numbers are used to sample from probability distributions or generate random variates. In the mathematical fields of probability and statistics, a **random variate** is a particular outcome of a random variable. This section describes how simulation software generates the random numbers.

One approach to generating random numbers is to include an extensive random number table in the simulation software. A small portion of a random number table is found in most introductory statistics books; however, accessing this table would be highly inefficient from a processing standpoint, given the large number of random numbers that are used in a simulation. The approach used in commercial simulation packages is one that generates numbers that are not actually random, although they appear to be random. These numbers are generated by a recursive relationship and are called **pseudorandom numbers**; however, for brevity, for the remainder of the book we simply refer to pseudorandom numbers as random numbers.

A key part of simulation software is a random number generator. Generators produce numbers that are uniformly distributed between 0 and 1 and are reproducible. Having reproducible random numbers is important for debugging and verification, and it enhances comparisons. Differences are due to changes in the model or system and not attributed to randomness. The generators are fast and require a small amount of storage, and they are portable; that is, they can be used with different compilers and on different computers.

Most simulation software uses either multiplicative or mixed linear congruential generators (LCGs). The ones that are used in commercial simulation packages have been extensively tested. It is important that these generators provide independent values and have long periods. A period is the length of a sequence of values before they repeat. Since most simulations require millions of samples from distributions and thus millions of random numbers, a generator with a long period is essential. As will be seen in the example below, the generators are quite simple; however, finding good parameters for the generator is very challenging. Much research and testing has been done to identify good parameter values.

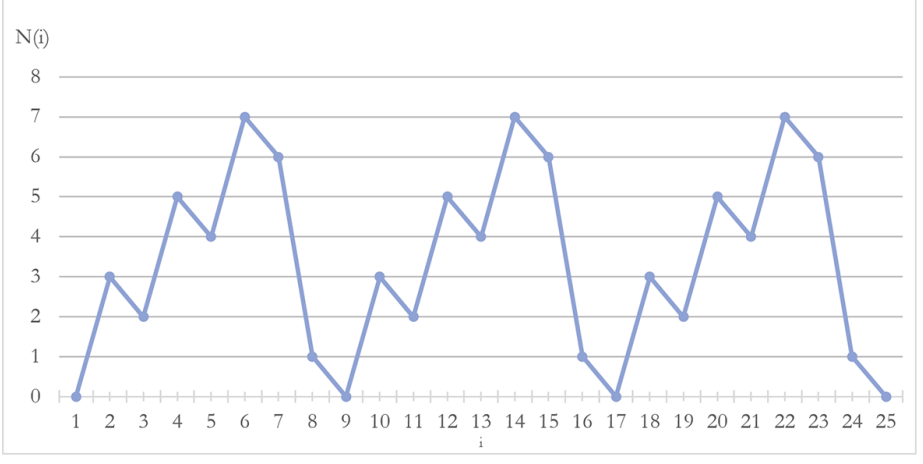
There are other types of generators besides LCDs (e.g., more general congruentials, composite, combined multiple recursive). For more information on these different types of generators and the types of statistical tests that are used to identify good generators, see Law [1].

The following mixed LCG is a recursive relationship that produces a sequence of integers:

$$N_i = (aN_{i-1} + c) \ mod \ (m)$$

where a is the multiplier, c the increment, m the modulus, and N_0 the seed. All are non-negative integers that, as discussed above, must be set carefully. The quality of the generator is very sensitive to the choice of its parameter values. Note, we use the notation N_i, N_i-1, and N_0 in this text; however, many references use the notation Z_i, Z_i-1, and Z_0. We change notation so as not to confuse this notation with that used for the standard normal distribution.

Since the generator produces integer values, noted as N_i, and the simulation needs uniformly distributed values between 0 and 1, noted as **uniform(0,1)**, u_i is obtained by dividing the random number by the modulus m:



i	N(i)	u(i)
0	1	
1	0	0.0000
2	3	0.3750
3	2	0.2500
4	5	0.6250
5	4	0.5000
6	7	0.8750
7	6	0.7500
8	1	0.1250
9	0	0.0000
10	3	0.3750
11	2	0.2500
12	5	0.6250
13	4	0.5000
14	7	0.8750
15	6	0.7500
16	1	0.1250
17	0	0.0000
18	3	0.3750
19	2	0.2500
20	5	0.6250
21	4	0.5000
22	7	0.8750
23	6	0.7500
24	1	0.1250

Figure 8.22: Random numbers produced by a simple generator.

$$u_i = \frac{N_i}{m}$$

In order to illustrate how the random number generation process works, consider the example where a=5, c=3, m=8, and the N_0=1; that is,

$$N_i = (5N_{i-1} + 3) \ mod \ (8)$$

While this is obviously a poor generator, it serves to illustrate the process. One problem with this generator is that its period will never be longer than *m*; therefore, in this case, the maximum period is just eight numbers before they repeat. With the seed N_0=1, the generator produces the following sequence of random numbers: N_1=0, N_2=3, N_3=2, etc. The resulting sequence of U(0,1) values is: 0.000, 0.375, 0.250, etc. The results are shown in Figure 8.22.

Notice that the generator produces all eight integers from 0 to 7 (m-1), the most integers possible with a modulo value of 8; therefore, the generator is considered to go full period, albeit a very small one. After generating those eight values, the pattern repeats. Again, the **uniform(0,1)** values are obtained by dividing the random number from the generator by *m*.

Also notice that if the seed value N_0 is 3, rather than 1, then the sequence is the same but it starts at a different value. That is, the sequence before repeating, N_1 to N_7, is (0, 3, 2, 5, 4, 7 6) for N_0 = 1; and, N_1 to N_7, is (2, 5, 4, 7 6, 1, 0) for N_0 = 3.

As will be discussed in the next chapter, for comparisons of alternative scenarios using a model, it is advised to use a different generator for each source of randomness in the model; however, separate generators are not needed if a single generator can produce a long stream of nonrepeating random numbers, and that stream can then be divided into smaller, yet still long, streams. This is what happens in most commercial simulation software, including *FlexSim*. The software's default generator is well tested and has a period of greater than two billion; that is, the generator produces two billion random numbers before they repeat. That single, long stream of random numbers is broken into smaller streams (e.g., 100 streams). That way up to 100 sources of randomness in a model can be assigned a different stream, each with a period of over 20 million. Each of the streams is accessed through a seed value. Instead of specifying the specific seed value, only a reference number is needed. That reference number is called a stream number and is typically between 0 and 99. All statistical distributions in *FlexSim*, and most other software, have a parameter in each statistical function for specifying the stream number. For example, normal(mean, standard deviation, stream number), triangular(min, max, most likely, stream number), etc.

In *FlexSim*, you are not limited to 100 random number streams; you can actually define as many as you need, though doing so is a bit of an advanced topic and will not be discussed here.

For simulations that require an extremely large number of random numbers, other generators are available. In addition to the default, *FlexSim* includes a generator that can have up to 10,000 steams, each with period ten quadrillion (2^{16}).

Section 8-7 Putting It All Together—Obtaining Random Samples from Probability Distributions

As indicated earlier and as should be apparent through the model building discussed so far in the text, probability distributions play a huge role in simulation modeling and analysis. The sections at the beginning of this chapter discussed ways to decide the appropriate probability distribution to use in a simulation model in order to best represent system behavior. The next section discussed how simulation software obtains samples from a specified distribution, in terms of generating random variates using random numbers. The following is a summary of that process.

1. Specify the distribution and its parameters that best represent the randomness in the system being modeled; that is, some element in the simulation model is a random variable X.
    a. Determine the distribution of X and its parameters; for example, X~normal(μ, σ) or X~exponential(β). This determination is made through a parametric approach using sample data from the system and with the help of software such as *ExpertFit*.
    b. The distribution may also be specified in the absence of sample data based on engineering judgment. Software, such as *ExpertFit*, can help with this as well.
2. Each time a random value is needed in the simulation, the following occurs:
    a. Randomly generate a **uniform(0,1)** number ui using a pseudo-random number generator. For example, use a linear congruential generator such as $N_i = (aN_{i-1} + c)(\text{mod } m)$ and convert the random integer to a **uniform(0,1)** random variable by $u_i = N_i/m$.
    b. Use u_i to sample from the specified probability distribution. For example, apply the inverse transformation method and use the cumulative distribution function, F(x); set $u_i = F(x_i)$ and solve for x; therefore, $x_i = F^{-1}(u_i)$. For an exponentially distributed random variable with a mean β, samples from the random variable X, where X~exponential(β), are obtained by executing $x_i = \beta\ln(1-u_i)$.

The complete set of statistical functions, or commands, that can be used in *FlexSim* can be found in the *FlexSim* User Manual, along with examples; however, Appendix A to this chapter summarizes some of the more commonly used distributions and their parameters.

References

1. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

2. Law, A. (2011) *Expert Fit Version 8 User's Guide*, Tuscon, AZ: Averill M. Law & Associates, Inc.

3. Law, A. (2013) "A Tutorial on How to Select Simulation Input Probability Distributions," *Proceedings of the 2013 Winter Simulation Conference*, R. Pasupathy, S. H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

4. http://nsspi.tamu.edu/nssep/courses/introduction-to-statistics/probability-distributions/probability-distributions

5. http://stats.stackexchange.com/

Chapter 8 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important in the chapter.
2. All of the distributions below have the same mean. Rank them in order of increasing variability.
   a. Exponential (100)
   b. Normal (100, 20)
   c. Triangular (80, 130, 90)
   d. Uniform (50, 150)
3. Identify an operations system and select two random variables that would be a part of a simulation model. Describe how you would decide what distribution to use to represent each of the random variables.
4. The time between failures on a machine is exponentially distributed with a mean of 1000 hours. Generate a random time to failure using a uniorm(0,1) value of 0.1209.
5. The time to repair a failed machine is uniformly distributed between 5 and 15 minutes. Generate a random repair time using a uniform(0,1) value of 0.7344.
6. The time to travel between two locations is normally distributed with a mean of 5 minutes and a standard deviation of 2 minutes. Generate a random travel time using a uniform(0,1) value of 0.9056.
7. Service times were found to have the following density. Generate a random service time using

$$f(x) = \frac{x}{12} \text{ for } 1 \leq x \leq 5$$
$$\text{and } u_1 = 0.7344$$

8. The distribution of the number of tests to be performed on a patient's blood sample is provided below. How many total tests are performed on the first four patients? Use the following random numbers: $u_1 = 0.2483$, $u_2 = 0.3637$, $u_3 = 0.0977$, $u_4 = 0.6424$

x	p(x)
1	0.10
2	0.35
3	0.35
4	0.20

9. The probability that an undergraduate student with a certain major will request assistance at a help desk is provided below. Using the following two random numbers, generate two types of student majors.

$$u_1 = 0.0977 \text{ and } u_2 = 0.6424$$

x	p(x)
Arts & Sciences	0.35
Business	0.25
Engineering	0.25
Other	0.15

10. For the LCG with parameters a=13, c=5, m=32, and N_0=4, generate two uniform(0,1) random numbers.

11. For the generator described in question 10, what is the maximum number of values that will be generated before the pattern repeats?

12. For an online bookseller, the time between customer orders is exponentially distributed with a mean of 15 minutes. Using the following generator, generate the time the next two orders arrive, assuming the last arrival occurred at time 100.

$$N_i = (27N_{i-1} + 4) \bmod (64) \text{ and } N_0 = 10$$

13. The time to assemble an item is uniformly distributed between 10 and 20 seconds. Generate the next two assembly times using the following generator.

$$N_i = (10N_{i-1} + 2) \bmod (32) \text{ and } N_0 = 16$$

Appendix 8A – Statistical Distributions in *FlexSim*

A complete set of statistical commands with examples can be found in the *FlexSim* User Manual. This appendix summarizes a few of the more common distributions by defining their parameters and providing an example application.

Note: The abbreviations *num* and *str* stand for number and string, which means that the corresponding parameter should be a numerical value or a text string. Text is always placed between double quotes, the symbols " and ". The last parameter *num stream* refers to the number of the stream for the random number generator. As briefly discussed above, and as will be discussed in more detail in the next chapter, the default value for the parameter stream number can be used for single scenario analysis; however, when using a model for comparisons, it is best to assign each source of randomness (interarrival time, service time, quality indicator, etc.) a unique stream number.

Discrete Probability Distributions

bernoulli(num percent-prob, num succeed-value, num fail-value, num stream)

A discrete random variable takes on the value succeed-value with probability percent-prob and fail-value with probability (100 - percent-prob).

Example: bernoulli(90, 0, 1, 2) Based on an inspection process, this generates either a 0 or 1 representing a good or bad part, respectively. There is a 90% chance the tested product is good (returns a value of 0); similarly, there is a 10% chance the part is bad (returns a value of 1). It uses random number stream 2.

To specify more than two possible values, such as good, bad, and rework, see the command dempirical.

binomial(num trials n, num percent-prob, num stream)

A discrete random variable of the count of the number of successes in n trials where each trial has a success rate percent-prob.

Example: binomial (20, 16.67, 2) This simulates 20 trials with a die where we count the number of times a 1 occurs (probability of a 1 on each trial is one-sixth or 16.67%) using random number stream 2.

duniform(num min, num max, num stream)

Generates equally likely discrete values between min and max.

Example: duniform(1 , 6, 3) simulates a die in that every value from 1 to 6 has 16.66 percent probability of returning 1, 2, …, 6, using random number stream 3.

See also the continuous distribution command uniform.

dempirical(str tablename, num stream)

This is a discrete empirical distribution function. It randomly returns a discrete value that is defined in Column 2 of a global table. It is expected that Column 1 of the table contains the percent occurrences of the values and Column 2 contains the values. The percents in Column 1 must sum to 100.

Example: int outport = dempirical("SendToTable",4); This sets the local variable outport to a random value based on the distribution specified in the global table named SendToTable using random number stream 4.

Continuous Probability Distributions

cempirical(str tablename, num stream)

This is a bounded continuous empirical distribution function. It randomly returns a continuous value that is based on the data in Column 2 of a global table. It is expected that Column 1 of the table contains the percent occurrences of the values and Column 2 contains the values. The percents in Column 1 must sum to 100. The values in Column 2 must be in ascending order because this function estimates values between the specified values continuously.

Example: double ptime = empirical("ProcessTimes", 1); This sets the local variable ptime to a random value based on the distribution specified in the global table named Process-Times using random number stream 1.

exponential(num location, num scale, num stream)

The location parameter, or lower bound of possible outcome values, is typically 0. The scale represents the parameter of the average service or inter-arrival time. Since the exponential is unbounded in the positive direction, it is possible to sample a very large value.

Example: exponential(0, 120, 2) This simulates a process with a mean time between arrivals of 120 (seconds), where the outcome will be in the range of 0 to infinity, using random number stream 2.

erlang(num location, num scale, num shape, num stream)

When the location parameter, or lower bound of possible outcome values, is 0 then the common notation is m-Erlang(β) or shape-Erlang(scale). The erlang distribution is the sum of m identically distributed exponential distributions with mean β. Thus, the exponential (location, scale) = erlang(location, scale, 1). Since the exponential is unbounded in the positive direction, it is possible to sample a very large value.

The mean is location + scale × shape, mode = location × scale(shape-1), and variance = shape × scale^2.

Example: erlang(0, 20, 4, 3) This simulates an assembly process composed of four steps, each taking 20 seconds, using random number stream 3.

normal(num mean, num stddev, num stream)(see Figure A.8.1)

This samples from the unbounded continuous distribution where the first parameter is the mean of the distribution and the second is the standard deviation.

Example: normal(100, 10, 4) simulates a process with a mean of 100 and a standard deviation of 10 using random number stream 4. Many of the results will be in the neighborhood of 100; however, since it is unbounded, extreme values in either direction are possible.

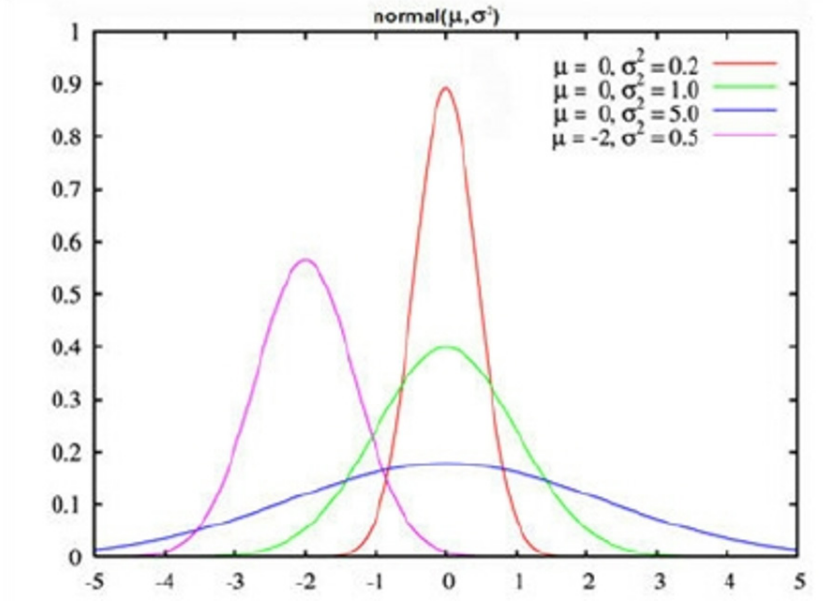


Figure A.8.1: Example normal probability density functions.

triangular(num min, num max, num mode, num stream)

This command returns a value between num min and num max; num mode is the value for which the probability density function (pdf) is maximized. Warning: the mode is different from mean, but in the case of the triangular, mean=(min+mode+max)/3. So, if you know the mean you can calculate the mode.

Example: triangular(3.5, 10, 5, 5) Time to complete a task is triangularly distributed with the shortest process time being 3.5 (min.), the longest process time being 10 (min.), and the most likely process time being 5 (min.). Random number stream 5 is used to sample the values.

uniform(num min, num max, num stream)

This generates a bounded continuous value that is equally likely between a specified lower and upper bound of the interval given by the parameters min and max, respectively.

Example: uniform(1, 7, 6) A travel time is uniformly distributed between 1 and 7 (min.). Random number stream 6 is used.

See also the command duniform for cases in which the sampled values need to be discrete.

beta(num min, num max, num shape1, num shape2, num stream)

This continuous distribution is bounded between min and max, but it can take on a variety of shapes based on the two non-negative shape parameters.

The mean is mean = min + (max – min) shape1/(shape1+shape2).

Note:

- beta (min, max, 1, 1) = uniform(min, max)
- beta(min, max, 1, 2) = triangular(min, max, min), a left triangle
- beta(min, max, 2, 1) = triangular(min, max, mode), a right triangle

A variety of beta distributions are shown in Figure A.8.2.

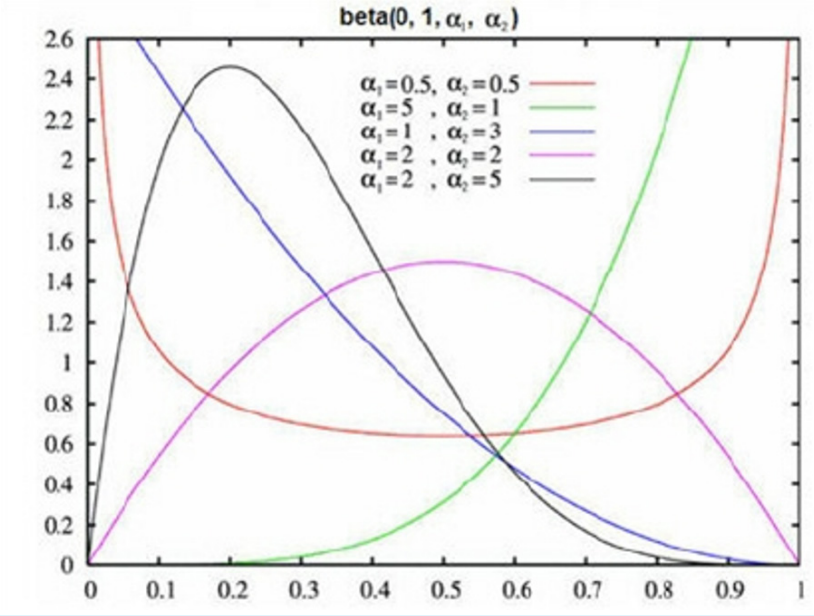


Figure A.8.2: Examples of beta probability density functions.

gamma(num location, num scale, num shape, num stream)

This continuous distribution can take on a variety of shapes, as shown in Figure A.8.3, based on its parameters. The gamma (location, scale, 1) = exponential(location, shape) and the gamma (location, β, m) = erlang(location, β, m).

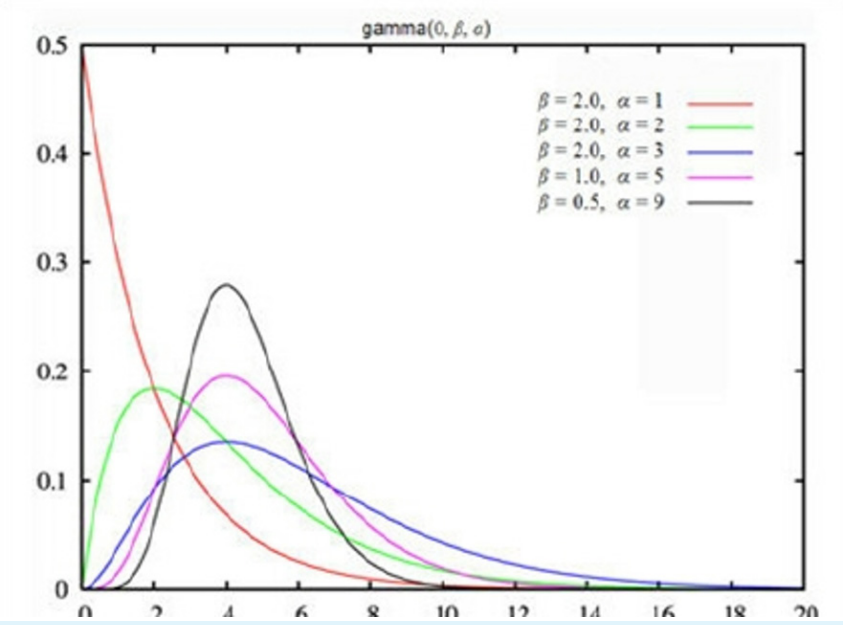


Figure A.8.3: Examples of gamma probability density functions.

weibull(num location, num scale, num shape, num stream)

This non-negative continuous distribution can take on a variety of shapes, as shown in Figure A.8.4, based on its parameters.

The weibull(location, scale, 1) = exponential(location, shape).

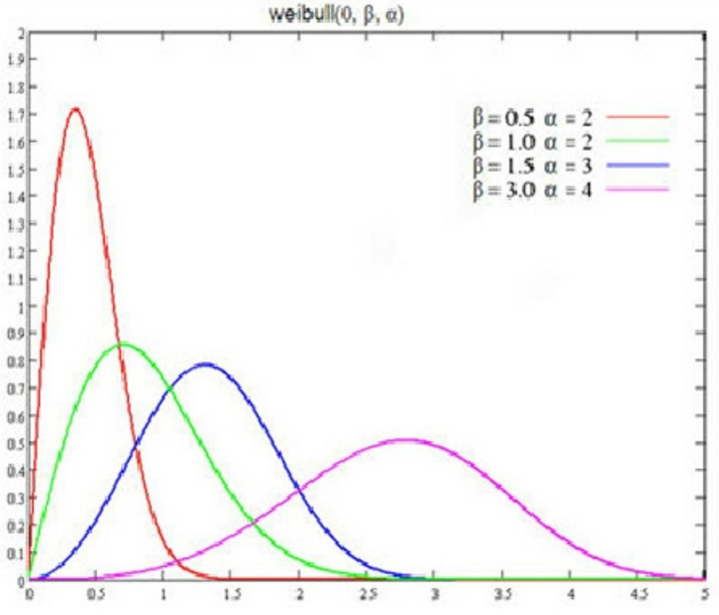


Figure A.8.4: Examples of Weibull probability density functions.

Chapter 9

Simulation Mechanics – How Monte Carlo and Discrete-Event Simulations Work

Objectives

To understand the underlying mechanics of the computations involved in simulation; to provide a basic description of how two types of simulation work: Monte Carlo and discrete-event.

As described in Chapter 1, systems, especially operations systems, can be simulated in four main ways depending on the objective of the analysis and the behavior of the system. This is illustrated in Figure 9.1, repeated from Chapter 1. Most of this book is dedicated to discrete-event simulation since it is the most common technique used to represent operations systems; however, Monte Carlo simulation is probably the most widely applied type of simulation in general and is the simplest to understand and use.

As shown in this chapter, the mechanics of the two simulation approaches are very different; however, they have a few things in common. Both, like all types are problem-solving tools, are used to (1) evaluate alternatives to enhance decision making, (2) understand interactions and variability in systems and their effect on performance, and (3) identify potential problems. In both Monte Carlo and discrete-event simulations, probability distributions are heavily used and are a key driver of the simulations. Both simulations use repeated sampling from probability distributions, a topic discussed in detail in the previous chapter, Chapter 8.

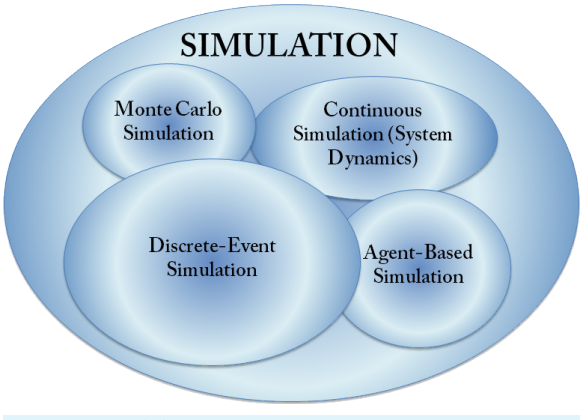


Figure 9.1: Types of systems simulations used to analyze operations systems

This chapter describes the basic mechanics of these two types of simulation and how random sampling plays a major role in the simulations. Section 9-1 describes the way Monte Carlo simula-

tions are conducted. Section 9-2 describes how discrete-event simulation works. In both cases, we use simple models to illustrate the process.

Section 9-1 Monte Carlo simulation

Monte Carlo simulation is basically the use of random sampling to obtain numerical results; in operations systems those results are typically a system characteristic. The random samples from any type of distribution are combined in some manner (e.g., added, multiplied, or in some way mathematically transformed) in order to obtain an estimate. For example, revenue from the sale of an item is the product of the item's selling price and the quantity sold. If both selling price and the quantity sold are random variables, then revenue is the product of two random variables, and thus is also a random variable. In this case, one sample from each distribution and their product is one simulated instance or outcome. The sampling and multiplication process is repeated many times. The resulting outcomes form a distribution of revenue, which may be summarized in terms of its mean and other measures.

Random sampling from probability distributions was discussed in Chapter 8. Recall, probability distributions can be parametric (e.g., normal, exponential, Weibull) or non-parametric (e.g., empirical) and can be specified based on sample data or in the absence of such data.

An effective way to learn how Monte Carlo simulation works is through a simple example. Consider the simple assembly problem illustrated in Figure 9.2. The assembly is of a shaft, labeled S, into a block with a hole labeled H. The turning process that is used to make the shaft has tolerance limits of ±1% of the desired diameter (1.49 in.). The drilling process used to make the hole in the block has tolerance limits of ±0.5% of the desired diameter (1.50 in.). Obviously, in order to assemble the two items, the shaft diameter must be less than the hole diameter; however, in this application this difference cannot exceed 0.025 in. An assembly is considered defective if a selected shaft does not properly fit within the hole of a selected block. The objective is to estimate the percentage of assemblies that would be defective in this process. If that value is too high, then changes must be considered and analyzed. For example, the tolerances may be revised; if so, then the percentage defective

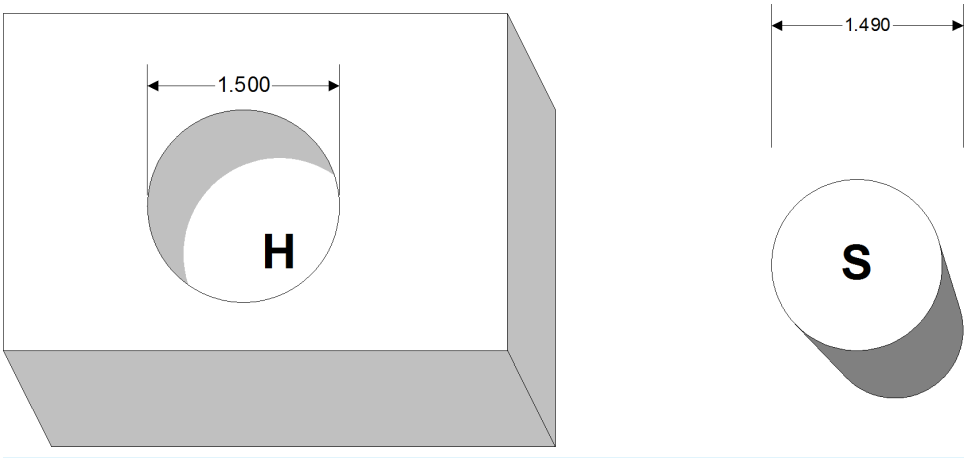


Figure 9.2: Simple assembly operation example.

calculation would need to be repeated for it and each suggested improvement and evaluated to see if the change is justified.

To begin the analysis process, some assumptions need to be made. Assume that

- diameters are normally distributed,
- samples are independent,
- tolerance limits are 6σ, which means 99.87% of the diameters are within the limits,
- and each assembly is only attempted once; that is, one shaft and one block are selected for assembly and both are rejected if the assembly does meet the "okay" criteria (shaft fits into hole, but the gap distance between the shaft and block is less than 0.020).

First, the specified tolerance limits for the shaft and the hole must be transformed to the parameters of the normal distribution (i.e., mean and standard deviation). The assumption that the tolerance limits (±1% of the mean 1.49) are 6σ implies the difference between the upper and lower limits represents 6σ. For the shaft, the upper limit UL = 1.49 + 1.49*0.01 = 1.5049 and the lower limit LL = 1.49 - 1.49*0.01 = 1.4751; their difference is 1.5049 – 1.4751 = 0.0298. If this difference is 6σ, then σ = 0.0049667 (round to 0.005). Similarly, for the hole, using the tolerance limits (±0.5% of the mean 1.50) are 6σ, then in σ = 0.0025. The shaft ~ normal (1.49, 0.005) and the hole ~ normal (1.50, 0.0025).

Recall from Chapter 8 that sampling from distributions involves the cumulative probability distribution function, F(x), which is the area under the probability density function f(x). F(x) is also the probability that the random variable X is less than or equal to a specified value of x, P(X ≤ x). Figure 9.3 provides an example of a continuous probability density function f(x) and its corresponding cumulative function F(x). If f(x) is a normally distributed random variable, then its cumulative function F(x) cannot be obtained by integration; it must be approximated. We cannot use the Inverse Transformation method discussed in Chapter 8; however, also as discussed in Chapter 8, there are other methods for sampling from such distributions.

Initially, this example problem is solved manually, and thus a standard normal table can be used for sampling. Since the normal density function cannot be integrated to obtain its distribution function, the table provides an approximation of F(x) for the normal distri-

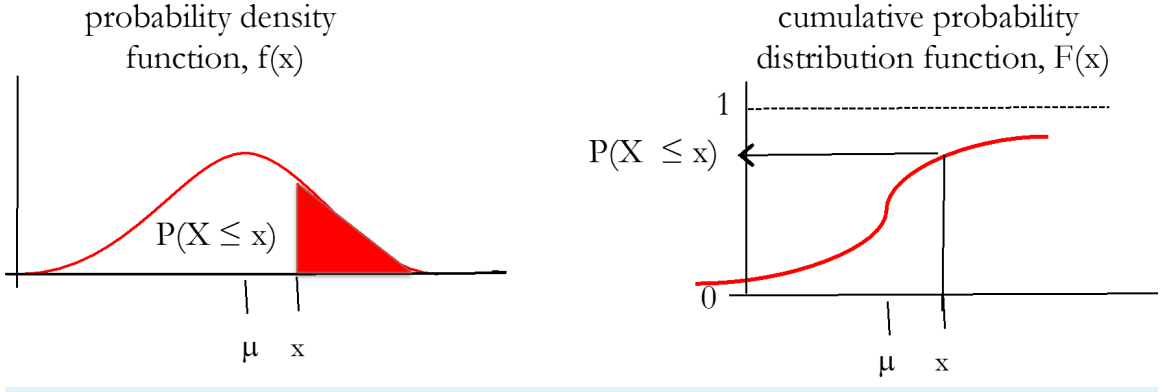


Figure 9.3: Example probability density function f(x) and cumulative distribution function F(x).

bution. A version of the standard normal table is provided in Appendix 9A, and a copy is provided below in Figure 9.4. The use of the standard normal table is covered in all basic statistics books and is not repeated here.

Probability Content from -oo to Z

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

Figure 9.4: Standard normal distribution table [1].

The density function in a standard normal table has a mean $\mu = 0$ and a standard deviation $\sigma = 1$; thus, it is standardized. The formula below is used to transform a standardized value from a value obtained from the table, denoted as z, to any normal distribution with mean μ and standard deviation σ. For example, the 90th percentile, or the value of x where $P(X \leq x) = 0.90$, is 1.28 standard deviations above the mean ($z = 1.28$ from the table). Using the hole size from the example, $X \sim$ normal(1.50, 0.0025), and the formula below, the 90th percentile value $x = 1.5032$ ($x = 1.50 + 1.28*0.0025 = 1.5032$). That is, 90% of the holes are expected to have a diameter of 1.5032 or less. The z value of 1.28 is obtained from the table by finding the cumulative probability 0.90 in the body of the table and then reading in the corresponding margin the values 1.2 and 0.08. Note that 0.900 is not found exactly in the

table, so 0.8997 is used and deemed a close-enough approximation. The z value for 0.900 could be found exactly by interpolating between the z value for 0.8997 and the z value for 0.9015.

$$x_i = \mu + z_i\sigma \text{ where } z_i \text{ is the standard normal, i.e. } z_i \sim \text{normal}(0,1)$$

Random sampling from a normal distribution, using the standard normal table, follows the same process as the example above, except the cumulative values (0.900 in the example) are generated randomly and not explicitly specified. Since the values of a cumulative distribution range between 0 and 1, a random number generator, as discussed in Chapter 8, can be used to generate uniformly distributed values between 0 and 1, uniform(0,1). For example, consider the simple random number generator from Chapter 8, $N_i = (5N_{i-1} + 3) \text{ (mod8)}$ and $N_0 = 3$, then $u_1 = 0.250$ and $u_2 = 0.625$, the corresponding z values from the table are $z_1 = -0.675$ and $z_2 = 0.32$. For the hole diameter in the example, normal(1.50, 0.0025), the corresponding sample values are $x_1 = 1.4983$ and $x_2 = 1.5008$.

These concepts are now applied to the example problem with the objective to estimate the percentage defective; i.e., the percentage of samples that result in an incompatible shaft and hole combination. The concepts are applied in two ways—manually and via *MSExcel*.

For the manual example, only three trials or outcomes will be generated, just enough to illustrate the mechanics of the process. Consider the following randomly generated uniform(0,1) values. The first two will be used in the first trial (one for the hole diameter and one for shaft diameter), the next two values will be used the same way but for the second trial, and the last three used similarly for the final trial.

0.389, 0.692, 0.896, 0.799, 0.077, 0.984

The small Monte Carlo simulation is carried out in Figure 9.5. The hole and shaft diameters are based on the random numbers provided above. The random numbers are used in the standard normal table to obtain random z values; these values are then used to determine the sample diameters. For each trial, the difference in the two diameters and a set of criteria together determine the outcome. The criteria that is applied to the difference are (1) (DiaH – DiaS) > 0.0 AND (2) 0 .0 < (DiaH – DiaS) < 0.020. That is, the hole must be larger than the shaft but the difference must be less than 0.020. In the first trial, the

	Hole		Shaft		Gap
Mean =	1.5	Mean =	1.49		V
Toler, % Dia	0.500%	Toler, % Dia	1.000%		0.025
Std Dev =	0.0025	Std Dev =	0.004967		

RandNo	Z	DiaH	RandNo	Z	DiaS	Diff	OK?
0.389	-0.28	1.4993	0.692	0.50	1.4925	0.0068	TRUE
0.896	1.26	1.5031	0.799	0.84	1.4942	0.0089	TRUE
0.077	-1.43	1.4964	0.984	2.14	1.5007	-0.0043	FALSE

Figure 9.5: Small Monte Carlo simulation for assembly example.

difference is positive and less than 0.020, so it is considered okay. The same is true with the second trial. The third trial, however, is considered to be a defective assembly since the shaft is larger than the hole. In this very small example, the estimated defective is 33%, since one out of three trial resulted in a defective assembly. This is obviously much too small of a sample. Obtaining larger samples via the computer is discussed next.

In practice, Monte Carlo simulation is carried out using a computer, oftentimes in a spreadsheet. Figure 9.6 shows a model of the assembly process represented in a spreadsheet that is divided into three main sections. The top section of the spreadsheet is used to input key parameters; this facilitates experimenting with different options. The second and main section is the actual model that contains the logic for each case (each trial fit of a shaft and hole). The third section provides output and summary measures.

The model uses several built-in *MSExcel* functions, as illustrated in Figure 9.6. All of the uniform(0,1) random numbers use the RAND() function. Since the RAND() function generates random numbers based on the computer's clock, they are not reproducible. Also, the generated values are updated each time *MSExcel* performs a calculation; therefore, when using RAND() it is preferred to invoke the manual calculations option, rather than the default automatic calculation mode. A better approach for generating random numbers is through the Data

	Mean dia	Tol limits, +/- % dia		Max gap	N(Mn)	N(StDev)
Shaft, S	1.4900	1.0000%		V	1.49000	0.00497
Hole, H	1.5000	0.5000%		0.025	1.50000	0.00250

Case	RAND()	Shaft(N)	RAND()	Hole(N)	Gap(N)	OK?
1	0.6401	1.49178	0.77180	1.50186	0.01008	1
2	0.6233	1.49156	0.51074	1.50007	0.00851	1
3	0.3051	1.48747	0.76325	1.50179	0.01432	1
4	0.3498	1.48808	0.90531	1.50328	0.01520	1
5	0.4602	1.48950	0.43172	1.49957	0.01007	1
6	0.2096	1.48599	0.68195	1.50118	0.01519	1
7	0.5984	1.49134	0.26030	1.49839	0.00716	1
8	0.5918	1.49		1.50145	0.01029	1
9	0.5045	1.49006	0.23153	1.49817	0.00811	1
10	0.8659	1.49550	0.11941	1.49706	0.00156	1
11	0.638					
12	0.8406	1.49495	0.64386	1.50092	0.00597	1
13	0.1312	1.48443	0.89720	1.50316	0.01873	1
14	0.4435	1.48929	0.53468	1.50022	0.01092	1
15	0.3623	1.48825	0.30583	1.49873	0.01048	1
16	0.4282	1.48910	0.26423	1.49842	0.00932	1
17	0.7577	1.49347	0.56379	1.50040	0.00693	1
18	0.9984	1.50469	0.09947	1.49679	-0.00790	0
19	0.0936	1.48345	0.39190	1.49931	0.01586	1
20	0.3408	1.48796	0.81133	1.50221	0.01425	1
Cases 1-20 n=20	0.50917	1.49048	0.53127	1.50022	0.00974 0.00577	0.95000
Cases 1-5 n=5	0.47570	1.48968	0.67656	1.50131	0.01163 0.00294	1.00000
Cases 16-20 n=5	0.52373	1.49173	0.42614	1.49943	0.00769 0.00943	0.80000

=NORMINV(B7,H3,I3)

=IF(AND(IF(H7>0,TRUE(),FALSE()), IF(H7<G4,TRUE(),FALSE())), 1,

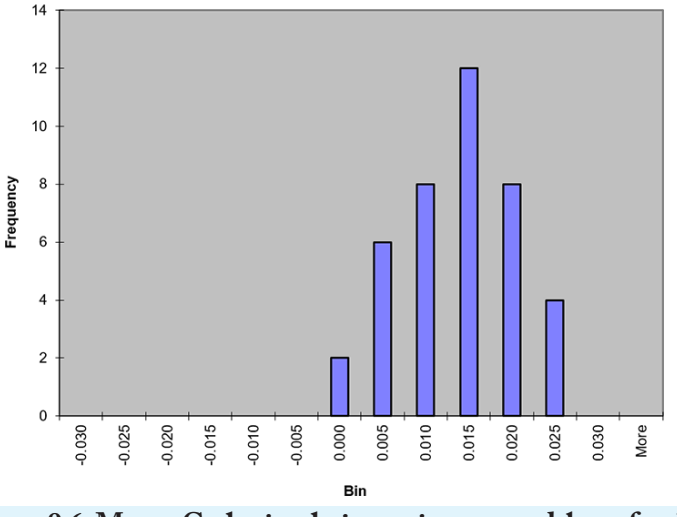


Figure 9.6: Monte Carlo simulation, using a spreadsheet, for the assembly example.

Analysis Pack Add-In and using the "Random Number Generation" picklist item. The main benefit is that it uses pseudorandom numbers, as discussed in Chapter 8, that are reproducible, and stream numbers can be specified. Since the generators are tied to stream numbers, the default automatic calculation mode can remain in effect.

To illustrate that RAND() generates uniform(0,1) random numbers, Kyd [2] generated 10,000 values from RAND() and created a histogram, as shown in the left-hand part of Figure 9.7. The right-hand portion of the figure shows the results of pressing the recalculate button – 10,000 additional random values between zero and one. Both examples show a fairly uniform distribution, as is expected. Of course, much more rigorous tests must be performed, especially on pseudorandom number generators that are discussed in Chapter 8, to be confident they perform as planned; for more information, see Law [4].

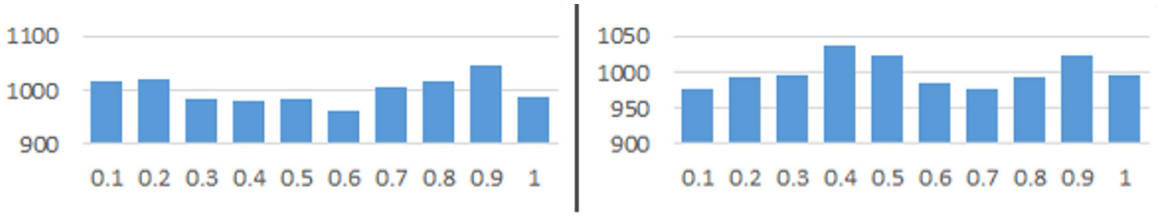


Figure 9.7: Random number generation from *MSExcel's* RAND() function.

The NORMINV(), or NORM.INV(), function is also used. This function returns a sample value from a normal distribution with the specified mean and standard deviation based on a randomly generated cumulative probability. Similar inverse functions for other probability distributions are available in *MSExcel*.

The model shown in Figure 9.6 is used for illustration purposes, and thus the number of trials or cases is very small, only 20. In practice the number of trials would be much larger. Since the main logic for each trial/case is contained in each row in the main section of the model, it is easy to extend the model to simulate many cases by just copying and pasting the rows.

Despite the small sample size, the example Monte Carlo simulation in Figure 9.6 can be used to illustrate the output from the simulation. The results of the simulation are shown in the first set of summary statistics in the bottom portion of the spreadsheet. They provide the average of the column values. The first four values are used to validate the model; all of the means are close to their expected values. For example, the average of 20 RAND() values is 0.53432 and is expected to be 0.5, the average of the sampled shaft values is 1.49049 and is expected to be 1.49, etc. The main measure is the last column, an indicator variable signifying whether the sampled shaft and hole are compatible; a 1 indicating yes, 0 no. The average of this column (multiplied by 100) is the estimate of the percentage of good assemblies, 80% in this case.

The histogram in the right-hand portion of Figure 9.6 gives a sense of the distribution of the gaps, or differences between the shaft and hole diameters. The bars on either end of the histogram represent the failed assembles—two where the shaft is larger than the hole and two where the gap is too large.

The last two rows in the output portion of the spreadsheet in Figure 9.6 illustrate the effects of sample size and sample error. Both have sample sizes of only five, but the first sample is from the first five trials in the main sample and the second is from the last five trials in the main sample. There is less confidence in the results from either of these samples, compared to the main sample of 20, because of the smaller sample size. Also, the estimate from the first sample indicates a good assembly rate of 60% and the second indicates 80%. This is due to sampling error (i.e., values vary from sample to sample). Of course, if we had two samples of five, it would be best to combine them into one sample of ten trials.

This section used the widely available *MSExcel* modeling tool to perform Monte Carlo simulations. Monte Carlo simulation can also be performed within general-purpose statistical software, such as Minitab [6], or through special-purpose software that focuses on Monte Carlo simulation. The special purpose software, such as *@RISK* [11] from Palisades Software, *Crystal Ball* [10] from Oracle, and *Risk Solver* [13] from Frontline Solvers, are add-ins to *MSExcel*. The add-ins provide additional capabilities and facilitate modeling and analysis using Monte Carlo simulation. A comparison of these software products is available in [7]. An example of a standalone Monte Carlo simulation software is *Goldsim* [9] from Goldsim Technology Group.

Case Study — Military Operation

The following case study is used at the Naval Postgraduate School to teach risk assessment and the flaw of averages. It is described by Savage [5], and a model of the situation, as a *MSExcel* file, is available at [12].

Monte Carlo simulation is used to analyze the failed U.S. military operation in April 1980 to rescue 52 American hostages that had been held by Iranian revolutionaries since November 1979. The mission, ordered by President Jimmy Carter, failed due to an accident at a desert refueling site.

To be a success (i.e., lift all the U.S. forces and return with all the hostages), it was determined that a total of six helicopters were required. A post-incident review indicated this decision was based on averages; it did not apply some simple risk assessment tools that would have them an assessment of possible overall mission failure due to helicopter reliability. The thought was that if the reliability of each helicopter is assessed to be 75%, then eight should be enough (0.75*8 = 6); however, based on Monte Carlo simulation, with eight helicopters there is only a 68% of success! The probability distribution for the number of surviving helicopters out of eight is shown in Figure 9.8.

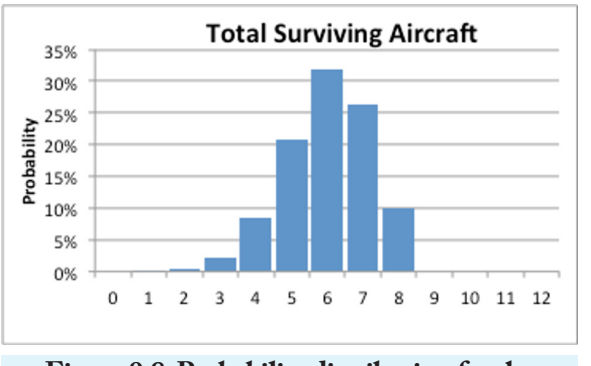


Figure 9.8: Probability distribution for the number of surviving aircraft, starting with 8, based on Monte Carlo simulation.

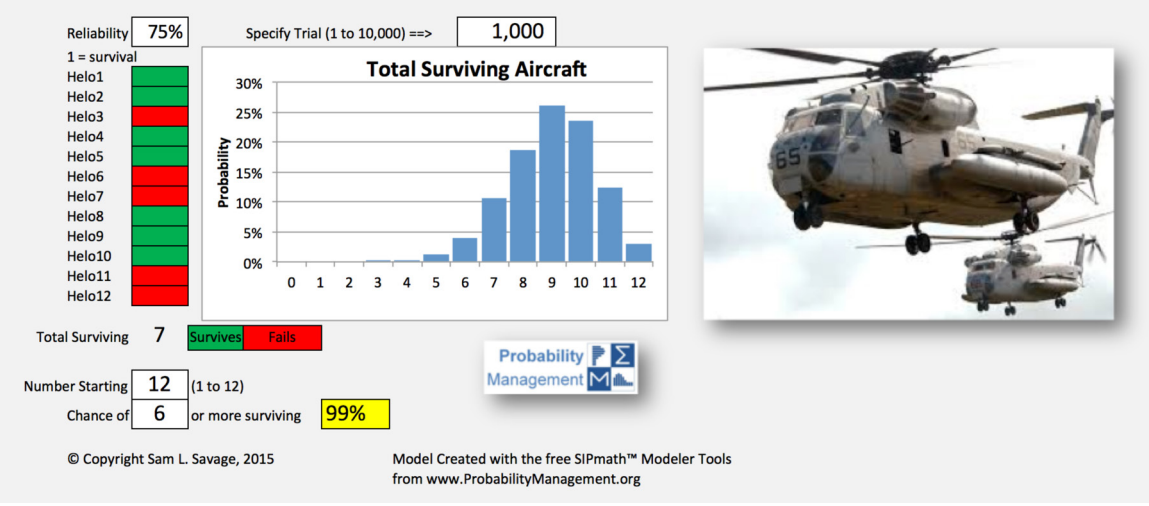


Figure 9.9: Probability distribution for the number of surviving aircraft, starting with 12, based on Monte Carlo simulation, *SIPmath* [12].

Figure 9.9. shows a simulation of 10,000 possible missions using 12 helicopters. In that case the estimated chance of success is 99%. The histogram shows while using 12 helicopters there is still a small chance of a mission failure (i.e., five surviving, four surviving, etc.). The left-hand panel in Figure 9.9 shows the outcome of one of the trials, the 1000th trial in the simulation, where seven of the 12 helicopters survived. This simulation, that can be run in *MSExcel*, is available for download at [12].

This is a great example of how simulation can enhance decision making. The impact of the decision to use eight helicopters is obviously enormous in terms of lives lost and American prestige. In fact, at an August 2015 press conference former President Jimmy Carter indicated that he wished he had sent one more helicopter and thus would have won re-election [12].

Section 9-2 Discrete-event simulation

While Monte Carlo simulation models are quite simple to develop, execute, and analyze, discrete-event simulations are much more complex because of the nature of the systems that they represent. As discussed in Chapter 4, all operations systems are complex, due to their interactions, variabilities, and dynamics.

Figure 9.10 presents the basic components found in all professional discrete-event simulation software. The second part of Chapter 8 discussed the first two components on the left-hand side of the figure—random number generation and generating random variates from continuous and discrete probability distributions—as well as how these two components work together to produce random samples from probability distributions. This section describes the last two components on the left-hand side of Figure 9.10: how the random samples drive events and the simulation clock.

Recall from Chapter 8 how simulation software obtains random samples from the distributions specified in a model. That process is summarized here:

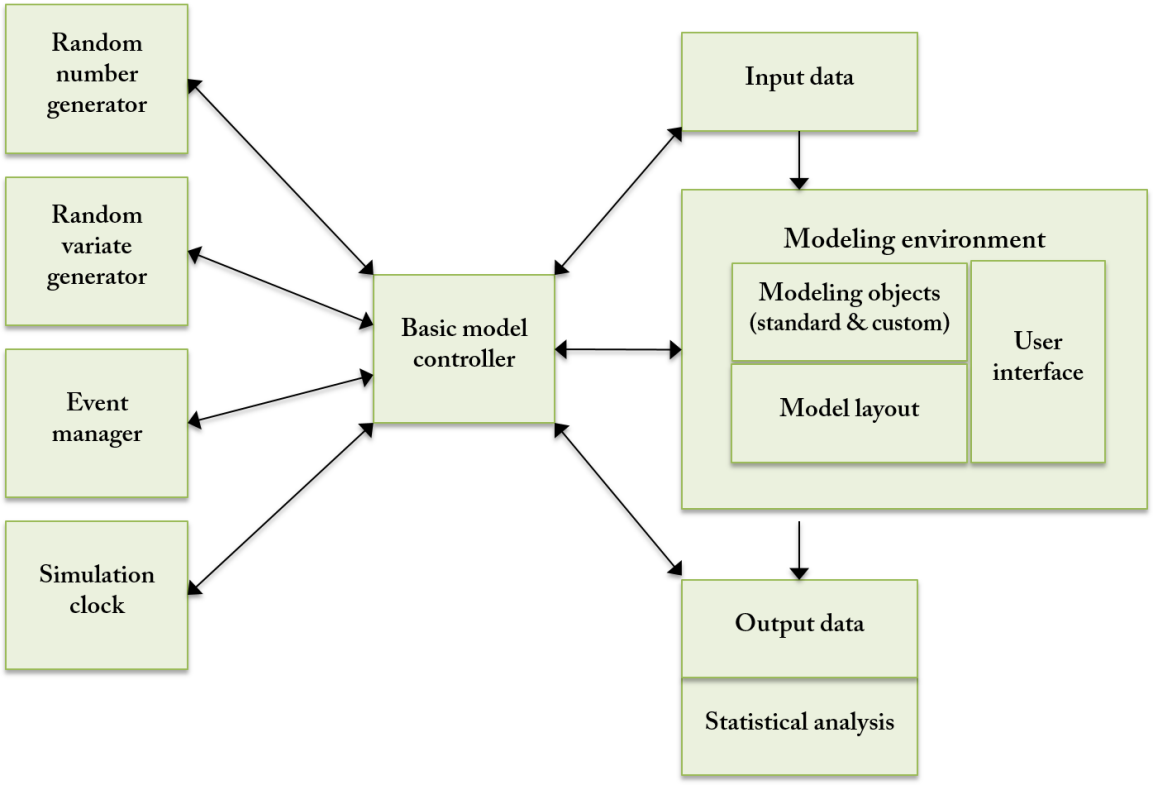


Figure 9.10: Key components in discrete-event simulation software.

1. Specify the probability distribution and its parameters that best represent an aspect of randomness in the system being modeled. The distribution is either specified through data analysis (e.g., with the help of software such as *ExpertFit*) or through engineering judgment in the absence of data.
2. Each time a random value is needed in the simulation, randomly generate a uniform(0,1) number using a pseudo-random number generator, and use the value to sample from the specified probability distribution using the inverse transformation method or some other approach.

The random samples from probability distributions create the behavior of the simulation. For example, the samples create event times that change states in the system and thus drive the dynamics of the simulation. This is illustrated in Figure 9.11 (repeated from Chapter 2) through a simple, single-server queueing system. In this case, the state of interest is the number of customers currently in the queue. It is assumed that simulation starts when the queue is empty but the server is busy; customers are processed in a first-in-first-out (FIFO) manner.

In Figure 9.11, a series of events (customer arrivals and departures) occur over time and change the state of the system. An arrival event, denoted as an upward-pointing triangle near the timeline, increases the number in queue, and a departure event, denoted as a downward-pointing triangle near the timeline, decreases the number in the queue. These events correspond to randomly-generated inter-arrival times (time between arrivals) and service times. The state of the system does not change between events. A third event

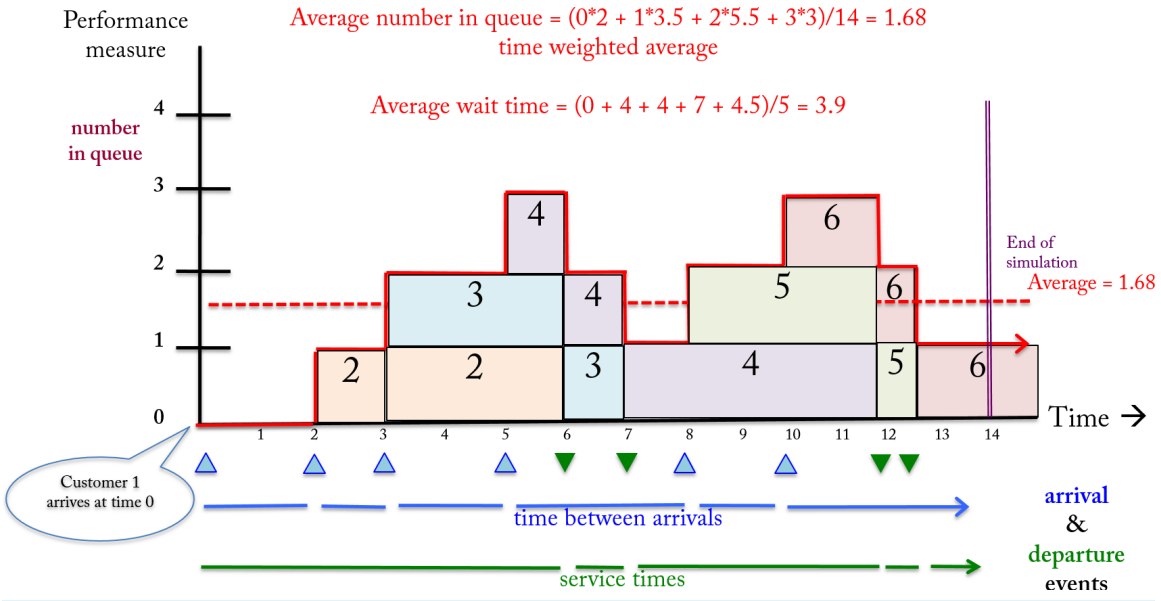


Figure 9.11: Example discrete-event queueing system.

type, shown in Figure 9.11, is the end of simulation event. In the figure the simulation is scheduled to run 14 minutes. In practice, this simulation time would be much too short for analysis; however, it is sufficient in this example to illustrate the event-processing concept.

States of the system are often summarized and used as performance measures. In many cases the state values are averaged (the mean) over the duration of the simulation using one of two common types of means: time-weighted and observational. For example, the time-weighted approach is used to calculate the mean number in queue; its calculation is shown at the top of Figure 9.11. An example of an observational average is the mean wait time; its calculation is also shown at the top of Figure 9.11.

For the example in Figure 9.11, the mean number in the queue is 1.68 customers over the 14 minutes of the simulation. This is the time-weighted average of how long the system was in each of the possible states; that is, there were 2 minutes when there were 0 in queue (state = 0), 3.5 minutes when there was 1 in the queue (state = 1), etc.

There is another approach for calculating a time-weighted average that provides the same answer but is much more efficient in terms of computing. It only requires maintaining a running total (weighted sum) and not the entire state history. The general time-weighted average calculation equations are provided below. This approach was discussed in Section 2.8 in terms of number in queue, but in general, X_j is the value of the state variable (e.g., the number in queue, when event j occurs) and t_j is the time event j occurs. The numerator is a weighted sum (i.e., for each interval between state changes, the state value that occurred over the interval is multiplied by the length of the time interval). This weighted sum can be kept as a running total ($TotX_{j-1}$) so that whenever an event j occurs, the average of the state variable through event j is the running total, $TotX_{j-1}$, plus the product of the value of the state variable since the last event and the weight, the time since the last event (TSLE). With this approach the current average is available throughout the simulation run and its final value is determined when the end-of-simulation event occurs.

$$\bar{X}_{tw} = \frac{\sum_{j=1}^{J} X_{j-1}[t_j - t_{j-1}]}{t_J} = \frac{\left(\sum_{j=1}^{J-2} X_{j-1}[t_j - t_{j-1}] + X_{j-1}[t_j - t_{j-1}]\right)}{t_J}$$

$$= \frac{(TotX_{j-1} + X_{j-1}[t_j - t_{j-1}])}{t_J} = \frac{(TotX_{j-1} + X_{j-1} * TSLE)}{t_J}$$

For an observational average, each observation value is summed and the total is divided by the number of observations. This is the most common method for calculating a mean and is described in basic statistics books. Its basic equation is provided below. For example, from the graph in Figure 9.11, the mean wait time is the sum of the five wait times for customers that have entered service (customer 1 waited 0 minutes, customer 2 waited 4 minutes, etc.) divided the total number entering service, five.

$$\bar{X}_{obs} = \frac{\sum_{i=1}^{n} X_i}{n} = \frac{\left(\sum_{i=1}^{n-1} X_i + X_n\right)}{n} = \frac{(TotX_{n-1} + X_n)}{n}$$

Similar to the time-weighted value, there is a more computationally effective method for calculating the mean of observational data. It also does not require that all data values are maintained, just a running total. The running total is the sum of all of the prior observations, TotXn-1 in the formula above; therefore, whenever an observation occurs, such as a customer entering service and ending its wait time, the current observation is added to the running total and their sum is divided by the total number of observations. As with the time-weighted approach, the current average is available throughout the simulation run and its final value is that when the end-of-simulation event occurs.

The logic that is used to produce the graph in Figure 9.11 is shown in Figure 9.12 This logic only pertains to a simple, single-server FIFO system, but is sufficient to describe the underlying mechanics of discrete-event simulation. Also, it only considers the collection of two statistical measures of system performance—average time and average number in queue.

As its name indicates, discrete-event simulation is event driven; that is, the simulation runs or is executed over time by processing events in the order in which they occur. Events cause states in the system to change. For example, an arrival might increase the number in queue or make a server busy; conversely, a departure event might decrease the number in queue or make a server idle. The flowchart in Figure 9.12 considers only two types of events—arrivals and departures—other than the end-of-simulation event; this was the case in Figure 9.11. In this example, an arrival is both an arrival to the system and an arrival to a server queue. Similarly, a departure event is both a departure from the single server and from the system. Another type of event, not considered in this example, might be a resource becoming available or unavailable, due to a failure or repair, or due to starting or ending a lunch break.

As shown in Figure 9.12, a set of actions create each event and the time in which the event is scheduled to occur in the future. The scheduled time of the event is placed in

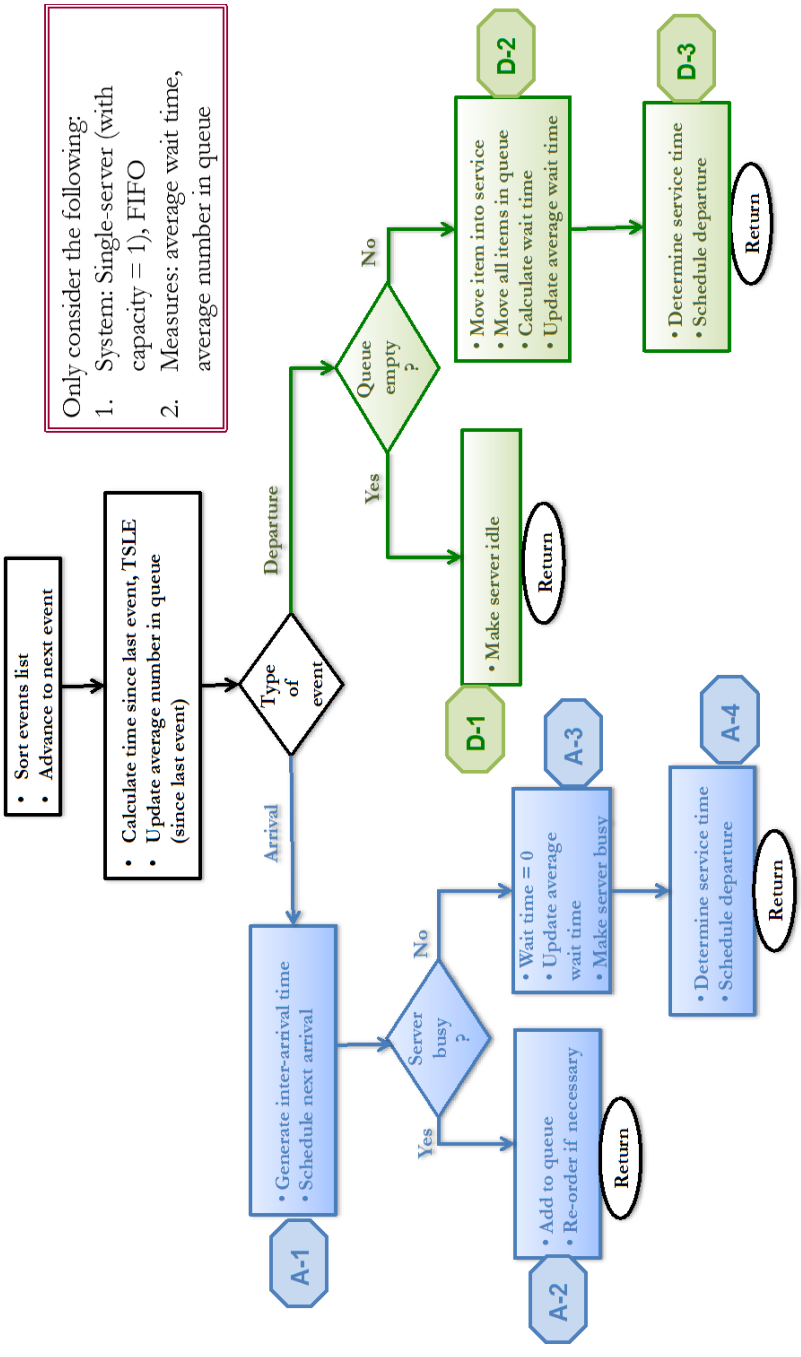


Figure 9.12: Simplified discrete-event simulation logic.

a list that is sorted in ascending order each time an event is to be processed. The event at the top or front of the list is selected for processing (i.e., the one to occur next). Once an event is selected for processing it is removed from the events list. Thus, the list only contains future events. When an event is selected, the simulation clock jumps to the time of the event. Since discrete-event simulation is event driven, no state variables change between events. Hence, the simulation clock moves ahead in time based on event times and not by a predefined "tick" or time unit.

As shown near the top of Figure 9.12, when an event of any type occurs, a system variable, TSLE (time since last event) is calculated; it is the size of the time jump between events. As shown in the time-weighted average equation earlier, TSLE is the time weight for the number in queue. Thus when an event occurs, the number in queue since the last event occurred is multiplied by TSLE and added to the running total. The average number in queue is this running total divided by the current simulation time.

If the selected event is an arrival, as shown in Figure 9.12, then the first action is to create and schedule the next arrival event; this is denoted as action A-1. This action involves

adding the value obtained from a random sample from the inter-arrival time distribution to the current clock time and placing that time in the simulation's events list. The remaining actions for the arrival event, A-2 or A-3 and A-4, depend on whether or not the server is busy. If the server is busy, action A-2 is performed—the arrival is just added to the queue, and the queue is reordered if necessary (not necessary in FIFO). If the server is idle, two actions are performed as the arriving customer enters service. The first, A-3, performs three tasks: (1) sets the wait time of the arriving customer to 0, (2) updates the average wait time (adds 0.0 to the running total wait time and adds one to the number of customers entering service), and (3) changes the state of the server from idle to busy. The second, action, A-4, performs two tasks: (1) determines the service time by obtaining a random sample from the service time distribution and (2) schedules the end-of-service or departure event by adding the service time to the current clock time and placing it in the events list.

Similarly, if the current event is a departure (end of service), then the actions depend on whether or not the queue is empty. If the queue is empty, then action D-1 is performed: the server is made idle. If there are customers in the queue, then two actions, D-2 and D-3, are performed. For D-2, four tasks occur: (1) the first customer is moved into service, (2) all remaining customers are moved up one position in the queue, (3) the wait time for the customer beginning service is calculated (current time – time customer arrived), and (4) the average wait time statistic is updated by adding the current wait time to the running total wait time and adding one to the number of customers entering service. For D-3, two tasks are performed: (1) determine the service time by obtaining a random sample from the service time distribution and (2) scheduling the end-of-service or departure event by adding the service time to the current clock time and placing it in the events list.

After the processing of each event is completed, the events list is sorted in ascending order in terms of time, and the next event in the list is processed. This procedure is illustrated through the following manual simulation example of a simple single-server model of Joe's Copy Shop.

The above actions are repeated each time an event occurs. Once all of the actions for an event have been processed, the events manager jumps forward in time to the next event on the list and processes its actions. This process continues until either there are no more events or the end-of-simulation event occurs.

Case Study: Joe's Copy Shop

Joe's Copy Shop can be represented as a simple queueing system. A schematic diagram of the system is shown in Figure 9.13. Joe enters each customer's copy order into a computer and the order is sent directly to the only copier in the shop; this is where our system begins, when orders enter the copier. The copies for each order are generated by the copier and then automatically sent to a holding area. Our system ends when the copy process is completed. Note that there is a buffer or queue in the system, but it is internal to the copier. An order is just information, but it is transformed by the copier to a set of papers. This model would be extremely simple to build in *FlexSim*, requiring just four serially-connected objects (source, queue, processor, sink).

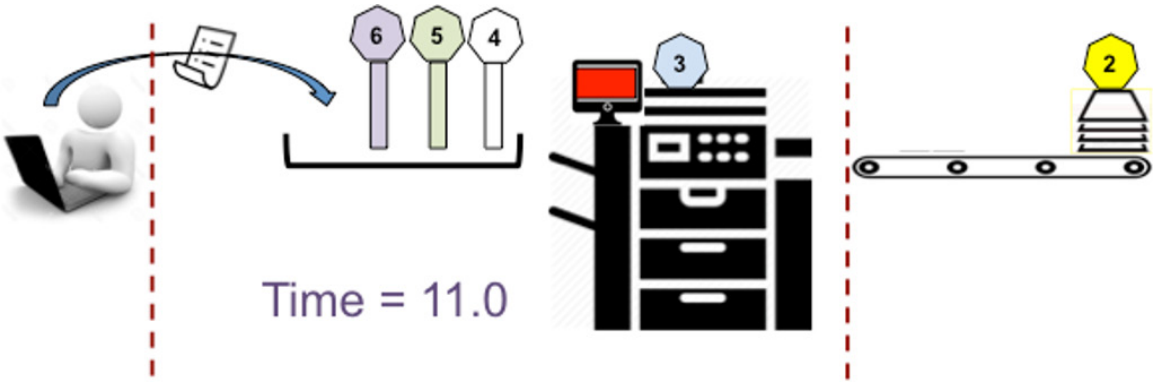


Figure 9.13: Schematic representation of Joe's Copy Shop.

The time between arrivals (inter-arrival times, IAT) of copy orders are obtained as needed in the simulation from an empirical distribution and a random number generator (as discussed in Chapter 8); the copy or process times for each order are obtained in the same manner. There is only one type of order, and they are processed FIFO.

Obviously this example involves many simplifying assumptions; there normally would be different types of orders (e.g., commercial versus personal, with commercial orders typically taking longer). Also, orders may have different priorities (e.g., rush standard, and economy). Another simplifying assumption is that the resources, Joe and the copier, are always available; Joe never takes a break, and the copier never needs to be resupplied or breaks down. *FlexSim* and other commercial simulation packages would have no problem modeling these system characteristics, but we simplify the system to demonstrate manually how discrete-event simulations work.

Figure 9.13 is a snapshot of the manual simulation that is discussed below—the status of the system at time 11.0, where order three is being processed and orders four, five, and six have been received and are awaiting access to the copier. In terms of states of the system, the number of orders in queue is three, total number of orders completed is two, and the server status is busy.

The complete details for the manual simulation of Joe's Copy Shop that demonstrates the mechanics of discrete-event simulation is provided in tabular form in Table 9.1. It is based on the simple queueing system described above and applies the logic defined in Figure 9.12. The manual simulation table is composed of two main parts: input (the first nine columns) and output (the next six columns). The last column is a textual description of the actions performed at each event action in column five. Two output measures are considered: (1) average wait time, calculated as an observational mean, and (2) average number in queue, calculated as a time-weighted mean.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List		Wait Time	Avg. Wait Time	# in Queue	TSLE	Avg. Num. in Queue	Event Action(s)
1	0.0		End of simulation		---		14.0	{14.0}							Schedule end of simulation
2	0.0		Start arrivals				0.0	{14.0, 0.0}							Schedule arrival of Customer 1
3	0.0	0.0	Arrival of Customer 1 [0.0]	A-1, A-3, A-4	2.0 (A-1)	6.0 (A-4)	2.0 (A-1) 6.0 (A-3)	{14.0, 2.0} (A-1), {14.0, 2.0, 6.0} (A-4)		0.0	(0.0 + 0.0)/1 = 0.0/1 = 0.0"	0	0.0	0.0	Update avg. Number in queue, Generate IAT, Schedule arrival of Customer 2, Wait time = 0, Make server busy, Determine service time, Schedule departure of Customer 1
4	2.0	2.0	Arrival of Customer 2 [2.0]	A-1, A-2	1.0 (A-1)		3.0 (A-1)	{14.0, 3.0, 6.0} (A-1)				1	2.0	0*2.0/2.0 = 0.0	Update avg. Number in queue, Generate IAT, Schedule arrival of Customer 3, Add Customer 2 to queue & reorder
5	3.0	1.0	Arrival of Customer 3 [3.0]	A-1, A-2	2.0 (A-1)		5.0 (A-1)	{14.0, 6.0, 5.0} (A-1)				2	1.0	(0.0 + 1*1.0) = 1.0 1.0/3.0 = 0.33	Update avg. Number in queue, Generate IAT, Schedule arrival of Customer 4, Add Customer 3 to queue & reorder
6	5.0	2.0	Arrival of Customer 4 [5..0]	A-1, A-2	3.0 (A-1)		8.0 (A-1)	{14.0, 6.0, 8.0} (A-1)				3	2.0	(1.0 +2*2.0) = 5.0 5.0/5.0 = 1.0	Update avg. Number in queue, Generate IAT, Schedule arrival of Customer 5, Add Customer 4 to queue & reorder
7	6.0	1.0	Departure of Customer 1 [0.0]	D-2, D-3		1.0 (D-3)	7.0 (D-3)	{14.0, 8.0, 7.0} (D-3)		6.0 - 2.0 = 4.0	(0.0 + 4.0)/2 = 4.0/2 = 2.0	2	1.0	(5.0 + 3*1.0) = 8.0 8.0/6.0 = 1.33	Update avg. Number in queue, End service on Customer 1, Calculate TIS, Move Customer 2 into service & others up. Calculate wait time. Determine service time. Schedule departure of Customer 2
8	7.0	1.0	Departure of Customer 2 [2.0]	D-2, D-3		5.0 (D-3)	12.0 (D-3)	{14.0, 8.0, 12.0} (D-3)		7.0 - 3.0 = 4.0	(4.0 + 4.0)/3 = 8.0/3 = 2.67	1	1.0	(8.0 + 2*1.0) = 10.0 10.0/7.0 = 1.43	Update avg. Number in queue, End service on Customer 2, Calculate TIS, Move Customer 3 into service & others up. Calculate wait time. Determine service time. Schedule departure of Customer 3
9	8.0	1.0	Arrival of Customer 5 [8.0]	A-1, A-2	2.0 (A-1)		10.0 (A-1)	{14.0, 12.0, 10.0} (A-1)				2	1.0	(10.0 + 1*1.0) = 11.0 11.0/8.0 = 1.38	Update avg. Number in queue, Generate IAT, schedule Arrival of Customer 6, Add Customer 5 to queue & reorder
10	10.0	2.0	Arrival of Customer 6 [10.0]	A-1, A-2	6.0 (A-1)		16.0 (A-1)	{14.0, 12.0, 16.0} (A-1)				3	2.0	(11.0 + 2*2.0) = 15.0 15.0/10.0 = 1.50	Generate IAT, schedule Arrival of Customer 7, Add Customer 6 to queue & reorder
11	12.0	2.0	Departure of Customer 3 [3.0]	D-2, D-3		0.5 (D-3)	12.5 (D-3)	{14.0, 16.0, 12.5} (D-3)		12.0 - 5.0 = 7.0	(8.0 + 7.0)/4 = 15.0/4 = 3.75	2	2.0	(15.0 + 3*2.0) = 21.0 21.0/12.0 = 1.75	Update avg. Number in queue, End service on Customer 3, Calculate TIS, Move Customer 4 into service & others up, Calculate wait time, Determine service time, Schedule departure of Customer 4
12	12.5	0.5	Departure of Customer 4 [5.0]	D-2, D-3		3.5 (D-3)	16.0 (D-3)	{14.0, 16.0, 16.0} (D-3)		12.5 - 8.0 = 4.5	(15.0 + 4.5) / 5 = 19.5/5 = 3.90	1	0.5	(21.0 + 2*0.5) = 22.0 22.0/12.5 = 1.76	Update avg. Number in queue, End service on Customer 4, Calculate TIS, Move Customer 5 into service & others up, Calculate wait time, Determine service time, Schedule
13	14.0	1.5	End simulation					{16.0, 16.0}					1.5	(22.0 + 1*1.5) = 23.5 23.5/14.0 = 1.68	

Table 9.1: Details of 14-minute manual simulation of Joe's Copy Shop.

The first three events in the simulation, as shown in Table 9.2, all occur at time 0. The first event created and placed in the events list is the end-of-simulation event, which in this case is scheduled to occur at time 14.0 since we are running a very short manual simulation. The second event starts the arrival process, and schedules the first arrival by placing it in the events list, which in this case occurs at time 0.0. Both of these events only occur at the beginning of the simulation. The events list is sorted, and the next event to process is the arrival at time 0.0. Once this event begins processing, it is removed from the events list.

Event	Sim Time	TSLE	Event Type	Event Actions	IntArr Time	Service Time	Event Time	Unsorted Events List
1	0.0		End of simulation		---		14.0	{14.0}
2	0.0		Start arrivals				0.0	{14.0, 0.0}

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List	Wait Time	Avg. Wait Time	# in Queue	TSLE	Avg. # in Queue
3	0.0	0.0	Arrival of Customer 1 [0.0]	A-1 A-3 A-4	2.0	 6.0	2.0 6.0	{14.0, 2.0} {14.0, 2.0, 6.0}	0.0	(0.0 + 0.0)/1 = 0.0/1 = 0.0	0	0.0	0.0

Table 9.2: Events 1, 2, and 3 of the manual simulation of Joe's Copy Shop.

Event 3 is processed according to the logic in Figure 9.12 The first operation is to calculate the time since the last event, TSLE. In this case, its value is 0.0. The average number in queue since the last event is also 0.0.

Event 3 is the arrival of the first customer to the system at time 0.0. The first arrival does not have to occur at time 0.0; it could be a specified time, or it could be randomly generated. Since it is an arrival, the left branch in Figure 9.12 is taken. The first action for every arrival is to schedule the next arrival; this is denoted as action A-1. The time until the next arrival is 2.0 minutes; it is obtained by using a random number generator and an empirical probability distribution. The event will occur 2.0 minutes from the current time (0.0) and is placed on the events list to occur at time 2.0. Since the arrival event occurs when the server is idle, and according to the logic in Figure 9.12, actions 3 (A-3) and 4 (A-4) are performed.

Action A-3 performs three tasks:

1. calculate the wait time of a customer entering service, which in this case is 0.0 since the customer just arrived;
2. update the average wait time, which in this case is 0.0;
3. change the server state from idle to busy, thus prohibiting any other customers from entering service (server capacity is one in this case).

Action A-4 performs two tasks:

1. determine the service time for the first customer by using a random number generator and an empirical probability distribution. The time is determined to be 6.0 minutes.
2. schedule the end-of-service or departure event, which will occur 6.0 minutes from the current time or at time 6.0. This event is added to the events list.

The events list now contains three future events. The events are sorted in ascending order {2.0, 6.0, 14.0} and the nearest event is processed next. Since the next event occurs at time 2.0, the system clock jumps from 0.0 to 2.0. No states in the system change between time 0.0 and 2.0 since states are only changed by arrival and departure events.

As shown in Table 9.1, the next three events happen to be, by coincidence, all arrival events. That is, three customers happen to arrive while the first order is being processed. The details for these events are shown in Table 9.3. They are again processed according to the logic in Figure 9.12.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List	Wait Time	Avg. Wait	# in Queue	TSLE	Avg. # in Queue
4	2.0	2.0	Arrival of Customer 2 [2.0]	A-1 A-2	1.0		3.0	{14.0, 3.0, 6.0}			1	2.0	0*2.0/2.0 = 0.0
5	3.0	1.0	Arrival of Customer 3 [3.0]	A-1 A-2	2.0		5.0	{14.0, 6.0, 5.0}			2	1.0	(0.0 + 1*1.0) = 1.0, 1.0/3.0 = 0.33
6	5.0	2.0	Arrival of Customer 4 [5.0]	A-1 A-2	3.0		8.0	{14.0, 6.0, 8.0}			3	2.0	(1.0 + 2*2.0) = 5.0, 5.0/5.0 = 1.0

Table 9.3: Events 1, 2, and 3 of the manual simulation of Joe's Copy Shop.

The first operation for event 4 is to calculate the time since the last event, TSLE. In this case, its value is 2.0. The average number in queue since the last event remains 0.0, or (0.0 + 0*2.0)/2.0) where the running total is 0.0, the number in queue during the two minutes since the last event is 0, and the duration of the simulation is 2.0.

Event 4 is the arrival of the second customer to the system at time 2.0. Since it is an arrival, as in the previous event, the left branch at the Type decision in Figure 9.12 is taken. The first action for every arrival is to schedule the next arrival; this is denoted as A-1. The time until the next arrival is 1.0 minutes; it is obtained by using a random number generator and an empirical probability distribution. The event will occur 1.0 minutes from the current time (2.0) and is placed on the events list to occur at time 3.0. Since the arrival event occurs when the server is busy, the "No" branch at the server status decision is taken, and according to the logic in Figure 9.12, only action 2 (A-2) is performed. Action 2 only adds the customer to the queue. Reordering the queue is not necessary since the FIFO queue discipline is used.

The events list now contains three future events. The events are sorted in ascending order {3.0, 6.0, 14.0} and the nearest event in time is processed next. Since the next event occurs

at time 3.0, the system clock jumps from 2.0 to 3.0. Again, no states in the system change between time 2.0 and 3.0.

The logic is exactly the same for events 5 and 6; however, note for event 5 the average number in queue becomes non-zero. The time since the last event, TSLE, is 1.0 and the number in queue since the last event is 1. Thus, total weighted sum is 1.0 (0.0 + 1*1.0) and the duration of the simulation is 3.0; therefore, the average is 0.33 customers in the queue. Sometimes students want to round this value to a whole number because the number in queue must be an integer; however, it should remains a decimal value since the average does not need to be integer valued and rarely is.

Event 7 is the next event. It occurs at time 7.0 and is the first departure from the system, the departure of customer 1. Event 8 is also a departure and the logic for the two events is the same, as shown in Table 9.4. At event 7 the average number in queue over the 6.0 minutes of simulated time is updated to 1.33 customers.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List	Wait Time	Avg. Wait	# in Queue	TSLE	Avg. # in Queue
7	6.0	1.0	Departure of Customer 1 [0.0]	D-2 D-3		1.0	7.0	{14.0, 3.0, 6.0}	6.0 - 2.0 = 4.0	(0.0 + 4.0)/2 = 4.0/2 = 2.0	2	1.0	(5.0 + 3*1.0) = 8.0, 8.0/6.0 = 1.33
8	7.0	1.0	Departure of Customer 2 [2.0]	D-2 D-3		5.0	12.0	{14.0, 8.0, 12.0}	7.0 - 3.0 = 4.0	(4.0 + 4.0)/3 = 8.0/3 = 2.67	1	1.0	(8.0 + 2*1.0) = 10.010.0/7.0 = 1.43

Table 9.4: Events 7 and 8 (departures) of the manual simulation of Joe's Copy Shop.

The logic that is applied for this event is shown in the right-hand portion of Figure 9.12. Since the queue is not empty at event 7, actions D-2 and D-3 are performed.

Action D-2 performs four tasks:

1. move customer 2 from the queue to service since the FIFO queue discipline is assumed;
2. move all customers forward one position in the queue since the FIFO queue discipline is assumed;
3. calculate the wait time of the customer entering service; it is the difference between the current time and the arrival time of the customer (6.0 – 2.0), which is 4.0 minutes for customer 2;
4. update the average wait time by adding the wait time calculated above to the running total and dividing by the total number of customers that have entered service, which in this case is (0.0 + 4.0)/2 = 2.0 minutes.

Action D-3 performs two tasks:

1. determine the service time for the second customer by using a random number generator and an empirical probability distribution. The time is determined to be 1.0 minutes;

2. schedule the end-of-service or departure event, which will occur 1.0 minutes from the current time or at time 7.0. This event is added to the events list.

The events list now contains three future events. The events are sorted in ascending order {7.0, 8.0, 14.0} and the nearest event is processed next. Since the next event occurs at time 7.0, the system clock jumps from 6.0 to 7.0 with no system states changing in this time interval. The next event is event 8, the departure of the second customer. The logic is exactly the same as the previous event.

As shown in Table 9.5, the next two events are of the same type: arrival events. In both cases the average number in queue is updated and actions A-1 and A-2 are performed—a new arrival is scheduled based on a random sample from the empirical inter-arrival time distribution and the customer is added at the end of the queue.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List	Wait Time	Avg. Wait	# in Queue	TSLE	Avg. # in Queue
9	8.0	1.0	Arrival of Customer 5 [8.0]	A-1 A-2	2.0		10.0	{14.0, 12.0, 10.0}			2	1.0	(10.0 + 1*1.0) = 11.011.0/8.0 = 1.38
10	10.0	2.0	Arrival of Customer 6 [10.0]	A-1 A-2	6.0		16.0	{14.0, 12.0, 16.0}			3	2.0	(11.0 + 2*2.0) = 15.015.0/10.0 = 1.50

Table 9.5: Events 9 and 10 (arrivals) of the manual simulation of Joe's Copy Shop.

As shown in Table 9.6, the next two events, 11 and 12, happen to be of the same type; they are departure events. In both cases the average number in queue is updated and actions D-2 and D-3 are performed—the first item in queue is moved into service, all remaining customers in queue are moved forward, the wait time for the customer entering service is calculated, the average wait time statistic is updated, the customer's service time is determined (random sample from probability distribution), and a departure event is scheduled based on the service time.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List	Wait Time	Avg. Wait	# in Queue	TSLE	Avg. # in Queue
11	12.0	2.0	Departure of Customer 3 [3.0]	D-2 D-3		0.5	12.5	{14.0, 16.0, 12.5}	12.0 - 5.0 = 7.0	(8.0 + 7.0)/4 = 15.0/4 = 3.75	2	2.0	(15.0 + 3*2.0) = 21.021.0/12.0 = 1.75
12	12.5	0.5	Departure of Customer 4 [5.0]	D-2 D-3		3.5	16	{14.0, 16.0, 16.0}	12.5 - 8.0 = 4.5	(15.0 + 4.5)/5 = 19.5/5 = 3.90	1	0.5	(21.0 + 2*0.5) = 22.022.0/12.5 = 1.76

Table 9.6: Events 11 and 12 (departures) of the manual simulation of Joe's Copy Shop.

As shown in Table 9.6, the events list for event 12 shows the next event to be processed occurs at time 14.0. That event is the end-of-simulation event. Also, note that if the end-of-simulation event were not to occur, two events would need to be processed at the same time, at time 16.0. The tie between the events must be broken since the simulation events manager can only process one event at a time. Typically, departure events are processed

before arrival events if they occur at the same time. By doing so the departure actions clear the way for the arrival so that it can enter service and not just join the queue. Similarly, if there is an item in the queue when an arrival occurs, the departure event allows it to enter service. Since in computer simulations, clock times have a high degree of precision, the chance that two random events occur at the same time is very, very small; however, the software must consider this possibility.

As shown in Table 9.7, the end of simulation event (1) updates the time-weighted statistics (in this example only the average number in queue is considered) and (2) stops processing events. In this example, the end-of-simulation event occurs 1.5 minutes after customer four departed or customer five entered service.

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List		Wait Time	Avg. Wait	# in Queue	TSLE	Avg. # in Queue
13	14.0	1.5	End simulation					{16.0, 16.0}					1.5	(22.0 + 1*1.5) = 23.523.5/14.0 = 1.68

Table 9.7: Event 13 (end of simulation) of the manual simulation of Joe's Copy Shop.

After the end-of-simulation event, the model is typically reset; all customers are cleared from the model, all statistics are reset, and the simulation clock is set to 0.0.

Notice that all actions defined in Figure 9.12 occurred in this short simulation except event action type D-1. Of course, if the simulation were carried on a bit further, the event type would eventually occur. As shown in the flowchart, Action D-1 is invoked when a departure occurs (service completion), and there are no customers in the queue. In that case, the only task is to change the state of the server from busy to idle and update server utilization statistics.

Also, notice that in the graphical approach in Figure 9.11 and the tabular approach in Table 9.1 (based on the flowchart in Figure 9.12), the statistics for the average wait time and the average number in queue are the same, 3.9 minutes and 1.68 customers, respectively. This is not a coincidence; both approaches are expected to produce the same results. One is simply easier to visualize while the other is easier for a computer to process.

Figure 9.14 provides a plot over time of the two averages considered in the simulation. In both cases the averages are increasing, indicating that the system is not stable. This is expected at the beginning of a simulation, especially when the system is empty and idle (empty of customers and the resources are idle). This illustrates one important decision that must be made when running the model for analysis, how long to run the simulation. This will be discussed in the next chapter. As indicated earlier, we know this simulation run time is much too short, but its purpose is to demonstrate the mechanics and not for analysis.

This very simple example is intended to only illustrate the basic concepts of how discrete-event simulators work. Simulation software is developed to model and analyze much more complex systems and their modeling views and implementations may differ. One approach is not necessarily better than another, but is just a matter of interpretation. As a result, it is very important to understand how individual software implement various concepts.

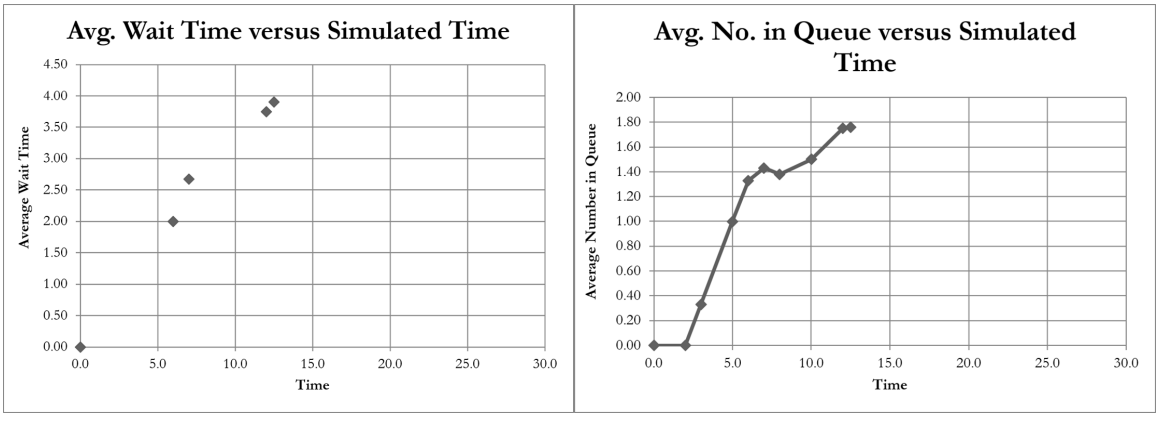


Figure 9.14: Change in means during the simulation of Joe's Copy Shop.

Building simple study or test models using deterministic times is a good way to understand, and validate, a modeling construct. For example, in the Joe's Copy Shop example, some simulators include the wait time for customer 6 in their statistics. As seen in Table 9.1 and Figure 9.11 this customer has been waiting 4 minutes when the simulation ends. The question becomes, should customers who have not completed waiting be included in the average (should 4 minutes be added to the total wait time and the total number of customers be six)? Most would say no, because this would bias the average lower since the sixth customer's wait will be longer than six minutes. How much longer, we don't know because the simulation is ended at this point. Of course, this becomes less of an issue when the simulation time is very long relative to the service and wait times.

Figure 9.15 shows the Joe's Copy Shop model implemented in *FlexSim* and its state at the end of the 14-minute simulation. As indicated earlier, this is a very simple model for *FlexSim* and other commercial simulators. The queue object is highlighted so some of

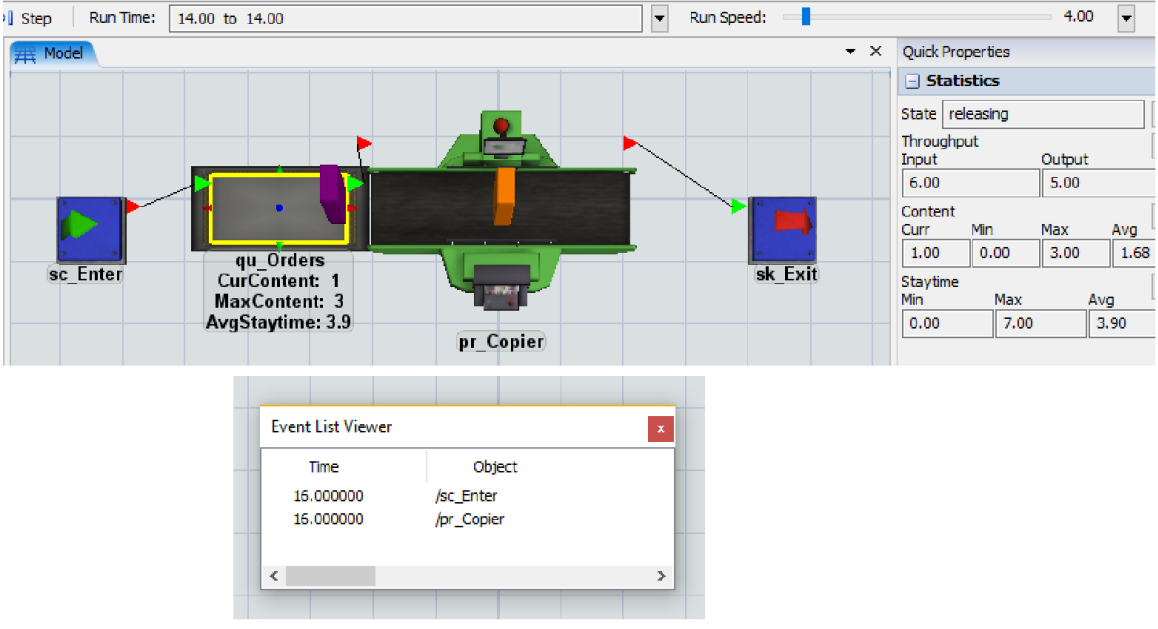


Figure 9.15. Simple *FlexSim* model of Joe's Copy Shop.

its characteristics are shown on the modeling surface and others are shown in the Quick Properties window. The Average Staytime in the orders queue (average customer wait time) is 3.90 minutes and the Average Contents of the orders queue (average number in queue) is 1.68. Both of these agree with the manual simulation in Table 9.1 and Figure 9.11.

Figure 9.15 also shows the current events list at time 14.0 in *FlexSim* for the Joe's Copy Shop example. As discussed above, there are two events on the list. The first is for an arrival at the source at time 16.0. The second is for the end of service or departure from the server at the processor object at time 16.0. Neither of these events will be processed since the end time of the simulation is set to 14.0. The events list is accessed in *FlexSim* via the Debug > Event List menu. A full history of events during the simulation is available via the Debug > Event Log menu; however, this list gets very long, very quickly because there are many more events than the arrival and departure events used in this example (e.g., OnEntry, OnExit, SendToPort triggers). The event log is usually only enabled for a short time interval, typically for model debugging.

In summary, the sequence of events that is processed by a simulator is determined by the values sampled from probability distributions and the logic built into the model. Thus, both the probability distributions and logic drive the output results from a simulation model. For example, the results would change if different distributions were used or if the distributions used different parameter values. Similarly, the results would change if the queue discipline used priorities or some other means besides FIFO (the order of arrivals). It is imperative to validate every model to ensure it is using the right logic and right data. The example also illustrates the way in which time is handled in a discrete-event simulator: during the simulation, clock time does not progress in ticks (preset steps) like a regular clock, but it jumps to the time that the next event in its events list is scheduled to occur.

References

1. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/NormTble.htm

2. Kyd, C. "Introduction to Probabilistic Simulations in Excel," http://www.exceluser.com/reports/introduction-to-probabilistic-simulations-in-excel.htm

3. Kyd, C. "How to Create Monte Carlo Models and Forecasts Using Excel data Tables," http://www.exceluser.com/reports/monte-carlo-models-forecasts-with-data-tables.htm

4. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

5. Savage, S. "Embracing the Uncertainty of the Iran Nuclear Deal," September 19, 2015, http://probabilitymanagement.org/library/Embracing%20Uncertainty%20in%20Iran%20Deal%209-19.pdf

6. Sheehy, P. and Martz, E. "Doing Monte Carlo Simulation in Minitab Statistical Software," https://www.minitab.com/en-us/Published-Articles/Doing-Monte-Carlo-Simulation-in-Minitab-Statistical-Software/

7. Sugiyama, S. "Monte Carlo Simulation/Risk Analysis on a Spreadsheet: Review of Three Software Packages," *Foresight*, Issue 9, Spring 2008.

8. www.edition.cnn.com/2015/08/20/politics/jimmy-carter-iran-hostages/

9. www.goldsim.com

10. www.oracle.com/us/products/applications/crystalball/overview/index.html

11. www.palisade.com/risk/

12. www.probabilitymanagement.org/index.html

13. www.solver.com/Monte-Carlo

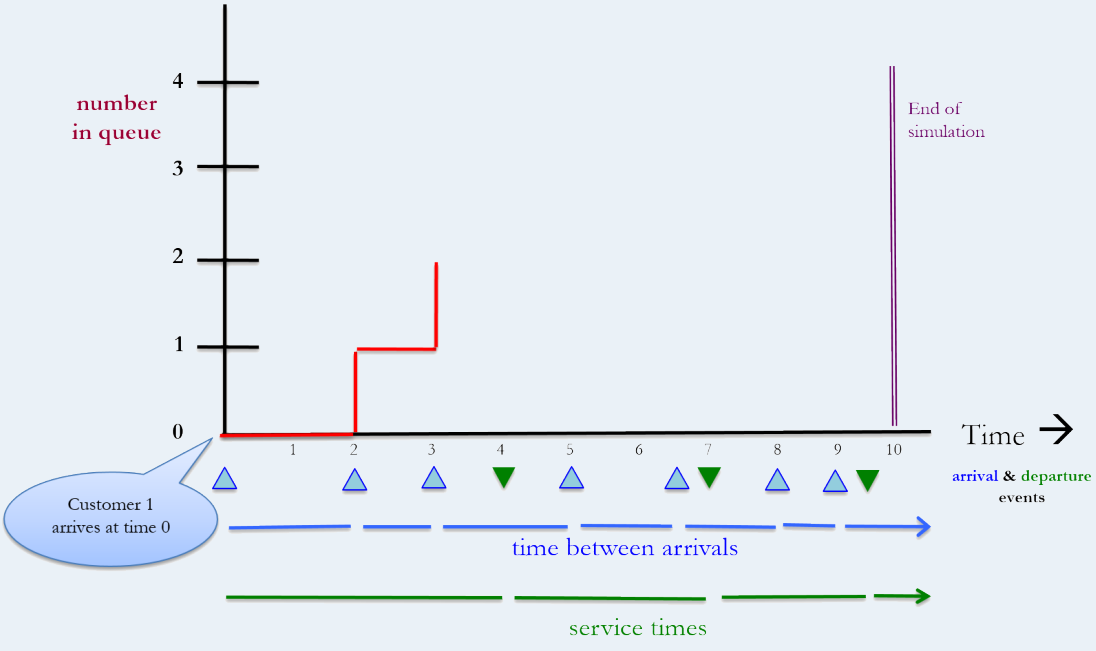
Chapter 9 Review Questions

1. A local entrepreneur is using Monte Carlo simulation to analyze profits for a possible venture. As part of that analysis, weekly revenue, in USD, is given as: NS*SP/EXR, where NS is the number of units sold, SP is the selling price in AMD, and EXR is the exchange rate from AMD to USD. Assume NS~normal(1000, 300), SP = 5000, and EXR~normal(400,25). Using the standard normal table in the appendix, generate
   a. one weekly revenue using the uniform(0,1) values 0.5691 and 0.8862.
   b. a second weekly revenue using the uniform(0,1) values 0.4410 and 0.6664.
   c. a third weekly revenue using the uniform(0,1) values 0.9903 and 0.1234.
   d. based on the very limited sample size, estimate the mean weekly revenue.
2. The weight of boxes to be processed depends on their size and the density (pounds per cubic foot, lb/cf) of the material in each box. Thirty percent of the boxes contain light material averaging 0.5 lb/cf; the remainder of the boxes contain heavier material averaging 5 lb/cf. All boxes are 1-foot tall but vary in length and width. Both length and width are normally distributed with a mean of 1 foot and a standard deviation of 0.5 feet. Any box weighing more than 5 pounds requires special handling.
   a. Based on a small sample of 5 boxes, estimate manually the percentage of boxes that require special handling. Use the following uniform(0,1) numbers as needed, by row from left to right.

0.185	0.289	0.028	0.006	0.349
0.899	0.482	0.905	0.653	0.854
0.873	0.548	0.921	0.015	0.773
0.081	0.571	0.110	0.097	0.196
0.469	0.692	0.073	0.124	0.643

   b. Based on a sample of 100 boxes, estimate using *MSExcel* the percentage of boxes that require special handling.
3. The first action in discrete-event simulation logic when an arrival occurs is to _____.
4. Briefly describe how time advances in a discrete-event simulator.

5. How are the events in an events list in a discrete-event simulator ordered?
6. Answer the following questions based on the figure below:
   a. Complete the plot of the state of the system, number in queue, through time 10, the end of the simulation.
   b. Calculate the average number in queue over the duration of the simulation.
   c. Calculate the average wait time over the duration of the simulation.
   d. At the end of the simulation, what is the
      i. status of the server?
      ii. number of customers in the queue?

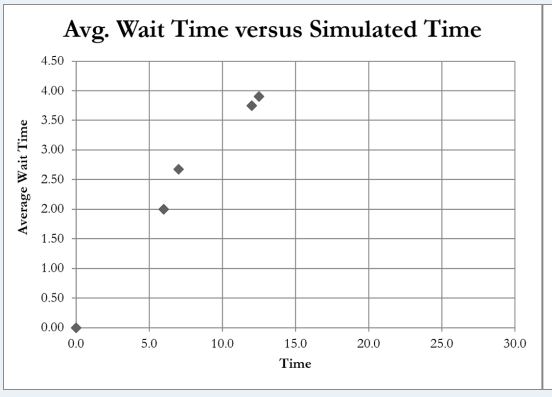


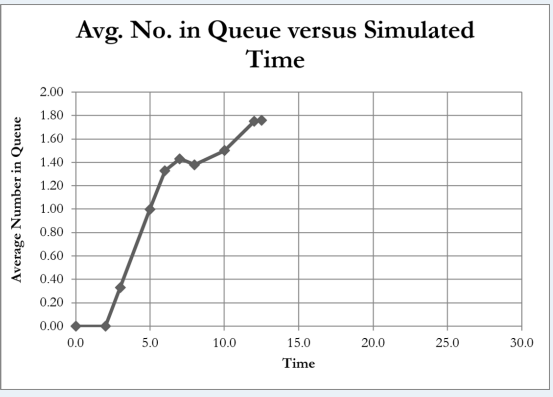
7. Using the following table, continue the manual simulation begun in the chapter until time 30.0. Use the following randomly generated inter-arrival and service times as needed; use them in the order in which they are listed.

Inter-arrival times:	4.0	4.0	1.5	5.0	
Service times:	3.0	2.0	1.5	3.0	4.0

Event	Sim Time	TSLE	Event Type	Event Action	IntArr Time	Service Time	Event Time	Unsorted Events List		Wait Time	Avg. Wait Time	# in Queue	TSLE	Avg. Num. in Queue
12	12.5	0.5	Departure of Customer 4 [5.0]	D-2 D-3		3.5	16	{16.0, 16.0}		12.5 - 8.0 = 4.5	(15.0 + 4.5)/5 = 19.5/5 = 3.90	1	0.5	(21.0 + 2*0.5) = 22.0 22.0/12.5 = 1.76
13	16.0	3.5	Departure of Customer 5 [8.0]											
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														

8. Complete the following average wait time and average number in queue plots based on the simulated results in the table above. What do you conclude about the run length of a simulation?





Appendix 9A – Standard normal distribution table

The following table is in the public domain and is produced by an APL program written by William Knight; accessed at: http://www.math.unb.ca/~knight/utility/NormTble.htm

Probability Content from -oo to Z

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

Chapter 10 Fundamentals of Output Analysis

Objective

To enable users of simulation to correctly analyze output and interpret results that are obtained from simulation models, especially considering the significant variation that is inherent in those models.

A simulation model is used to understand the behavior of a system in order to support decision making and solve problems. Simulation models are especially used to understand behavior when changes to the system are considered. In that sense, a model is like a laboratory, in that it is used for experimentation. Experiments are conducted by changing inputs to a model, oftentimes referred to as decision variables, and observing and analyzing the effect of the input settings on system behavior. The effects are characterized by performance measures and are the results, or output, of a simulation model.

Analyzing output from simulation models must be done with care because of the complexity of the models and the systems they represent. This complexity is due to the interactions among system/model elements, inherent variabilities and uncertainties, and dynamics. In this book, the analysis of simulation model output is discussed in two basic parts: (1) fundamentals of output analysis and the analysis of a single scenario and (2) analysis of multiple scenarios. A scenario is a set of specific value settings for the input variables that are being considered; the value settings result in output from the simulation model that represents measures of system performance.

The first part of simulation output analysis is discussed in this chapter, and the second part is discussed in the next chapter. This chapter begins with a brief discussion of how output analysis fits into the overall simulation modeling and analysis process. Subsequently, this chapter

- describes the analysis of output from simulation models in the context of basic statistical inference;

- discusses key tactical experimental design decisions that must be made to effectively use simulation models for analysis;
- introduces FlexSim's Experimenter, a very powerful tool for analyzing model output and conducting experiments with simulation models.

While analysis begins once a simulation model has been developed, validated, and verified, planning the analysis must start at the beginning of the simulation process. As defined and discussed in Chapter 16, simulation modeling and analysis (SMA) is a process. A portion of that process is shown in Figure 10.1. The SMA process starts with a clear set of objectives for why the simulation project is being conducted. From these objectives, the questions that need to be answered via simulation should become evident. Typically the objectives identify the system performance measures that will be used as a basis for decisions and choosing among alternatives. These measures are the output of the simulation.

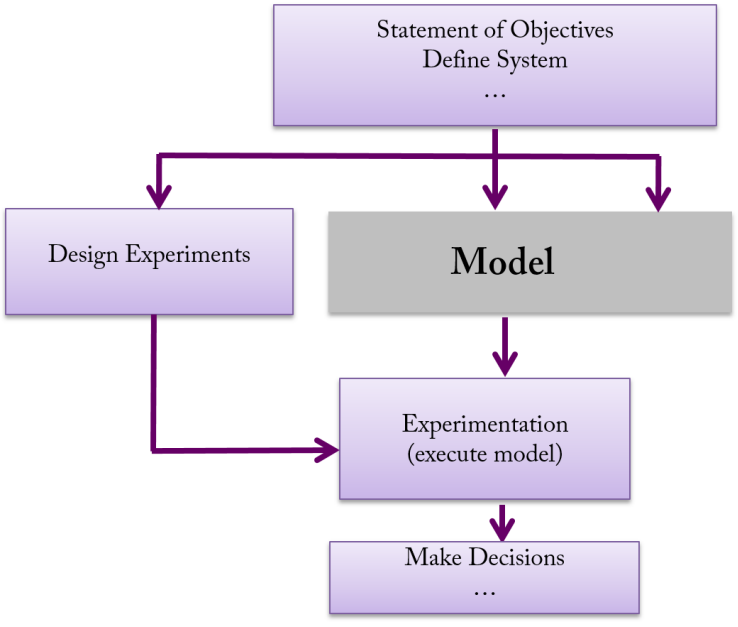


Figure 10.1: A portion of the Simulation Modeling and Analysis process.

Also as shown in Figure 10.1, the design of the experiments begins in parallel with the development of the model(s). The design process includes both the definition of (1) key performance measures, output of the simulation, and (2) the variables that will be changed across the various alternatives under consideration, the input to the simulation.

Section 10-1 Statistical inference from simulation models

As has been discussed throughout this book, many system characteristics (process times, reliability of equipment, arrival pattern of customers, etc.) are represented as probability distributions in simulation models. As such, simulations are driven by the use of random samples from probability distributions, along with logic and physical characteristics, to represent a system's operation and behavior; Since the systems are stochastic or probabilistic, so are models of those systems. As shown in Figure 10.2, stochastic inputs lead to stochastic outputs; that is, since input to simulation models are random variables, even if there is only one, then the output—response variables or performance measures—of the system are also random variables. Output from simulation models therefore must be analyzed using basic statistics; in other words, inference must be made about the true characteristic of a system based on a sample of information from that system. Random sampling from probability

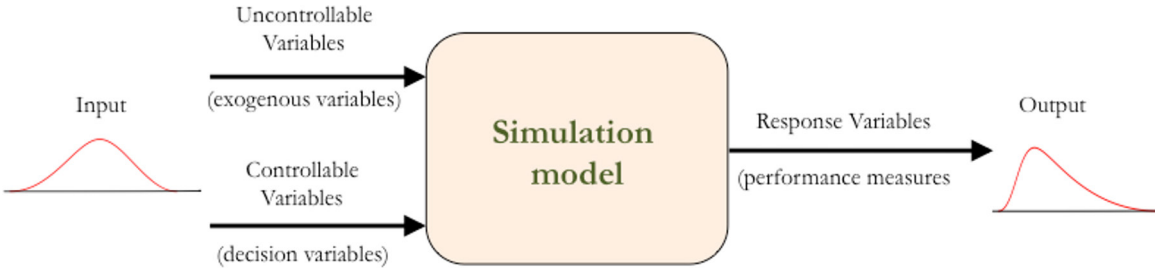


Figure 10.2: Stochastic input lead to stochastic output.

distributions was discussed in Chapter 8. Recall, probability distributions can be parametric (e.g., normal, exponential, Weibull) or non-parametric (e.g., empirical) and can be specified based on sample data or in the absence of such data.

Figure 10.2 also shows that there are two basic types of input to simulation models—uncontrollable variables and controllable variables. Controllable variables, also called decision variables, are those that are under immediate control of the decision maker. These are the variables that are subject to change in a simulation in order to compare alternatives. Some common examples of decision variables include the number of servers or operators to use, the queue discipline to follow (e.g., FIFO, priority), and the location of the equipment and other resources. Controllable variables are typically not random variables. On the other hand, uncontrollable variables are oftentimes random variables and are not under the immediate control of the decision maker (e.g., process times, reliability, and inter-arrival times). Of course, the distinction between controllable and uncontrollable depends on context. For example, if three alternative means for performing a process are being considered, then the means would be controllable. The means are not random variables, but the times resulting from their distribution are.

When making a decision, the decision maker would like to do so with perfect information. They would like to know the true characteristics of a system; or, in statistical terms, they would like to know the population values of these characteristics. In order to do so, all values of the characteristic need to be known, which is impossible for most systems, so the population characteristic, or parameter, needs to be estimated. This estimate is oftentimes based on statistics obtained from a sample. In general, the population is all members of a specified group, and a sample is a subset of the members.

The best way to understand the fundamental output analysis concepts is via example. We start with a simple example from Chapter 2. Consider a single-server queueing system with inter-arrival times that are exponentially distributed with a mean of 1.0 minutes and service times that are exponentially distributed with a mean of 0.9 minutes; hence, the average server utilization is 90% (0.9/1.0). The system also processes customers on a FIFO basis, there is infinite queue capacity, and the server is always available (no downtime). This is a basic M/M/1 queueing system. If the system is such that it meets the assumptions of the M/M/1 Queueing Theory model, then simulation is not needed—the performance measures are obtainable from the equations provided in Chapter 2. We use simulation here, however, to demonstrate the importance of the experimentation parameters and to compare the experimental results to known population values (i.e., those obtained from the M/M/1 formulas).

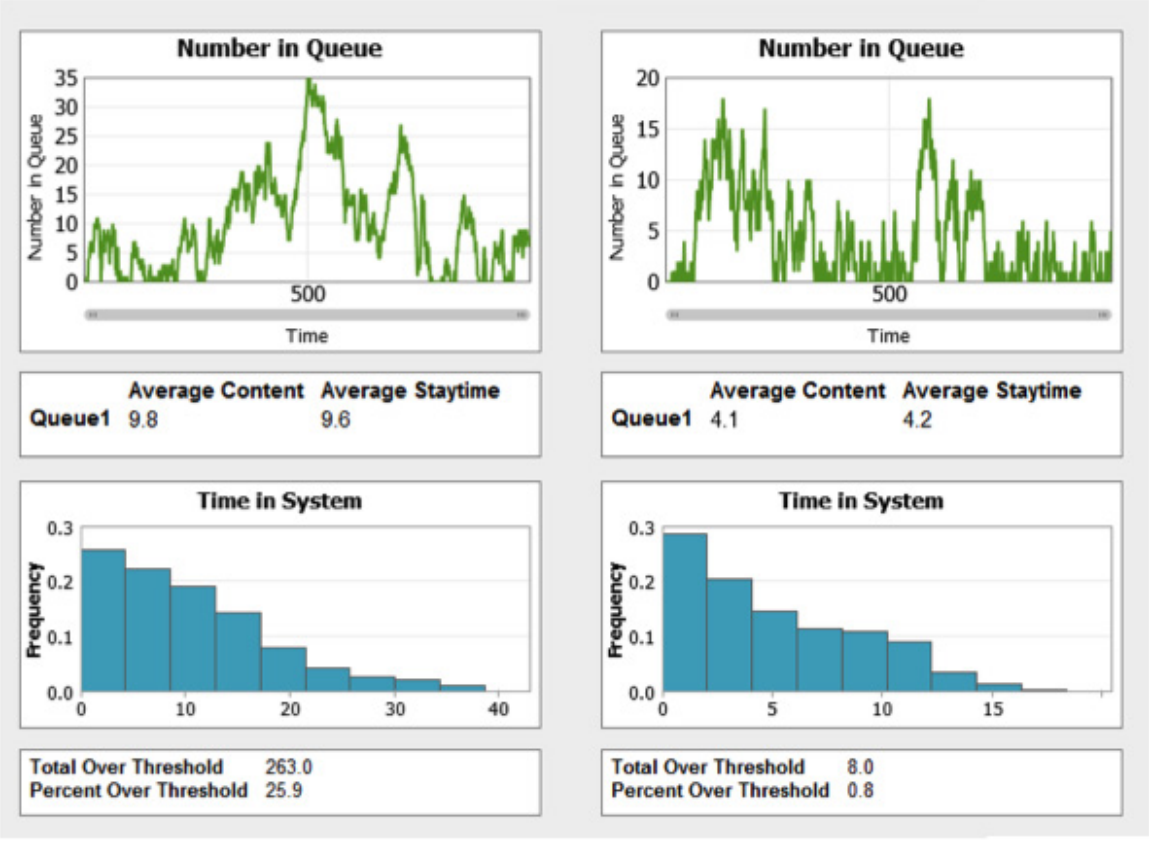


Figure 10.3: Two possible dynamics for M/M/1 system, approximately 1,000 customers.

Figure 10.3 shows the results of two independent runs of a simulation model of the M/M/1 system defined above. Each run is for 1,000 simulated minutes (i.e., since the inter-arrival time is 1.0 minutes, approximately 1,000 customers arrive to the system over the course of the simulation). Obviously, very different results are obtained from the two runs. This difference indicates that there is considerable variability in the system. But this is to be expected in a stochastic system: observations over two periods will likely provide different outcomes. A thousand minutes is just over 16 hours, so the model may represent one day of a two-shift operation. Certainly, in most systems, what happens on one day will be quite different than what happens on another day.

The graphs at the top of Figure 10.3 show the time series of the number of customers in the queue during the 1,000-minute simulation. They reflect quite different dynamics. The dynamics vary because of the interplay of randomly occurring arrivals with random service times for each arrival. The bottom graphs in Figure 10.3 show the distribution of the time customers spent in the system. Again, very different results are reflected. As shown by the statistics at the bottom of the figure, for the distribution on the left, 25.9% of the customers were in the system for at least 15 minutes (threshold). This compares to less than 1% for the distribution on the right. (Recall the average service time is only 0.9 minutes; therefore, there is a very high ratio of wait time to service time.) The differences between the two runs of the model are summarized by the descriptive statistics in the middle of the figure. The

simulation represented on the left has a considerably higher average number of customers in the queue and average time in system.

The two simulations represented by the information in Figure 10.3 are from the same system. Using the results from either one to make a decision would be very, very risky. Doing so would be making a decision or characterizing a system's performance on one observation! The mean that is calculated in each simulation is the mean over the simulation run (1,000-minute duration). For number in queue, a time-weighted average would be used, and for time in system, an observational average would be used; these types of means are discussed in Chapter 9. Either mean from Figure 10.3 by itself would not be a good estimate of the true system mean, or in statistics terms, the population mean; however, combining several of the means would provide a better estimate. One approach is to average the means. Using just two values is too few, but determining how many to use is discussed later in the chapter.

Each run of a simulation model is called a replication. In the example, Figure 10.3 presents the results of two replications; therefore, the estimated system performance should be based on multiple replications. This is analogous to the term *sample size* in statistics. Replications are discussed in more detail later in the chapter.

Figure 10.4 illustrates the basic concepts of estimating population parameter values based on sampling. This is a primary topic in all basic statistics courses and thus is only summarized here. First, a population of interest is identified, and then a population parameter, or characteristic of the population, is specified. In statistics the population parameter is denoted as μ. (This should not be confused with the average service rate from Chapter 2.)

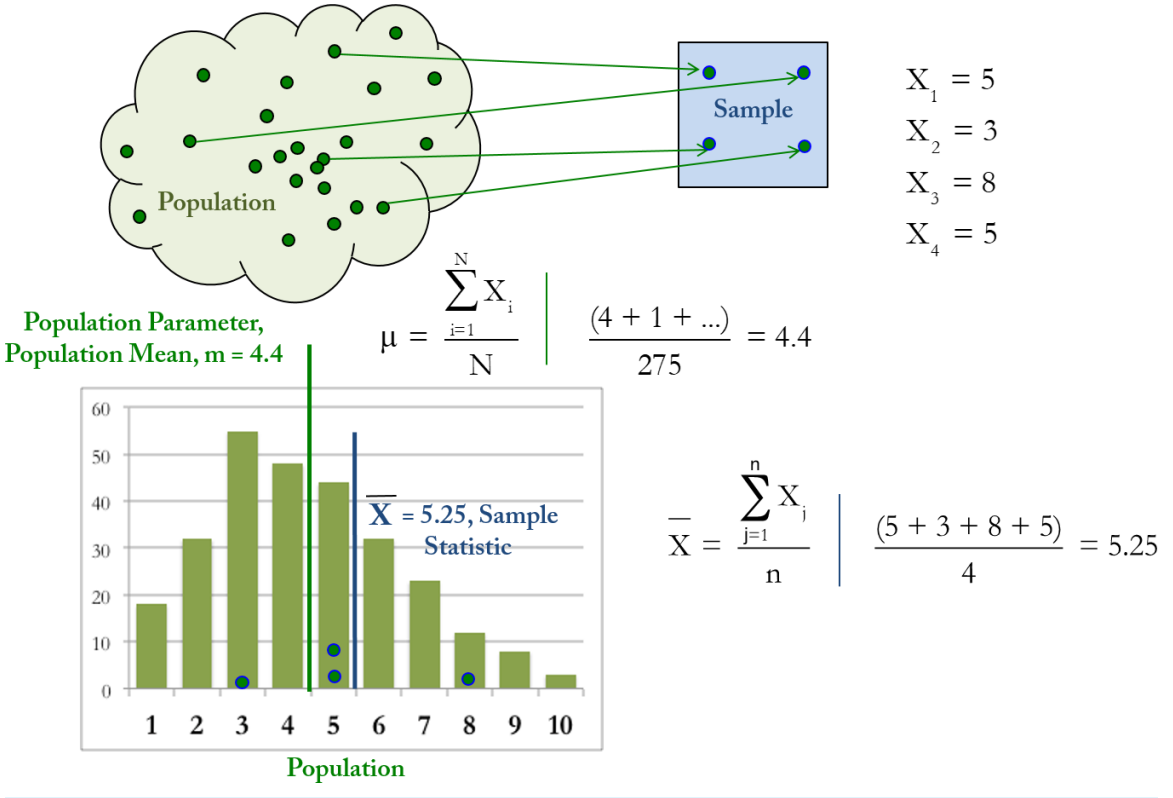


Figure 10.4: General sample-based process for estimating population parameters.

In the example in Figure 10.4, the population parameter of interest is the average weight of an item in a shipment of 275 items. Each item weighs 1, 2, 3, …, or 10 kilograms. Since the population is finite (275 items), the population parameter can be known with certainty by weighing each item and then averaging the weights; however, in this case that is not possible, both because the weighing process is too time consuming and the handling of the items destroys them. You might suggest weighing the entire shipment and dividing the weight by 275. That would be okay if the distribution of weights were uniform (i.e., equally likely between 1 and 10 kilograms); however, assuming the weights to be uniformly distributed would not be a good assumption in this case based on the histogram in Figure 10.4.

In Figure 10.4 the average weight of all of the items in the shipment is estimated by sampling four items, weighing them, and then averaging their weight values. The sample mean is the sample statistic $\bar{X}$ and its value in the example is 5.25 kilograms. The estimated weight of the population, all items in the shipment, is therefore 5.25. Note that true mean weight of an item in the shipment mean, μ, is 4.4 kilograms, based on the population distribution shown in Figure 10.4. Since the sample size n is much smaller than the population size N, it is highly unlikely that $\bar{X} = \mu$.

Figure 10.5 illustrates the basic concepts of population parameter estimation based on simulation. The process is very similar to that of basic sampling, but there are a few key differences. The first step is the same—to identify a population of interest and to specify a population parameter, or characteristic of the population, μ. In this example, a decision to choose among several process improvement options is to be based, at least partially, on

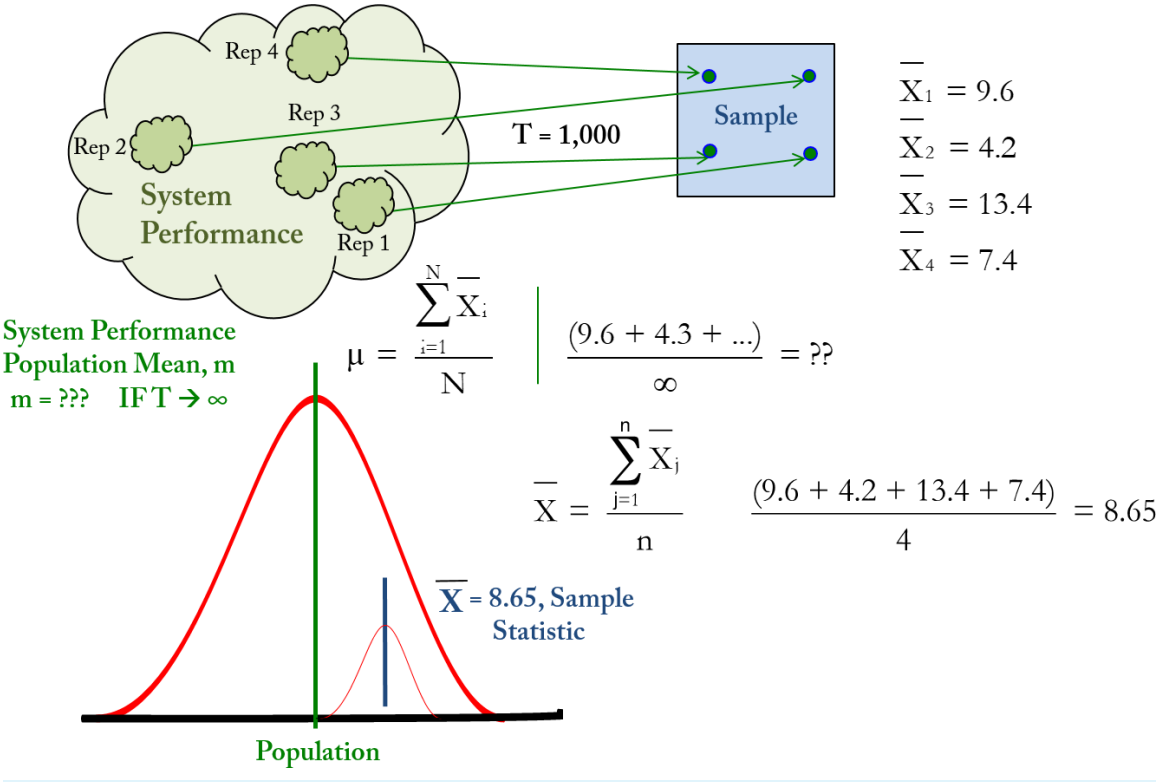


Figure 10.5: Simulation-based process for estimating population parameters.

the mean time a customer spends in the system. One difference is that in simulation we estimate characteristics of systems, such as wait times or lead times of customer, maximum inventory, business of servers, throughput of the system, etc., by sampling from a model, not the real system. Another difference, reflected in Figures 10.4 and 10.5, is the type of random variable being considered. In Figure 10.4 the weight of an item is considered to be discrete or integer valued. Weight is usually a continuous random variable, but in this case weight only takes on the values 1, 2, …, 10. For the example in Figure 10.5, time in the system is a continuous random variable and thus can have an infinite number of possible values.

Based on the process in Figure 10.5 one might estimate average time in the system by running a simulation and averaging the individual customers' times in the system that are observed in the single run of the simulation. This is the wrong approach for two key reasons: first, in the example in Figure 10.4 the weight of each item in the shipment is reasonably assumed to be independent of any of the other items in the shipment. The times in the system in Figure 10.5 are not independent but likely highly correlated. The correlation is due to the nature of the system, especially if the main component of the time in the system is wait time. If a customer in a queueing system has a large wait time, then customers before and after will also likely have large wait times. The second reason the average time in system from a single run should not be used as the estimate of the population value is illustrated in Figure 10.3, where two simulation runs exhibit very different behavior and result in very different measures of system performance. In Figure 10.3 an M/M/1 system was run twice, each for 1,000 minutes (approximately 1,000 customer arrivals to the system), and resulted in average times in system of 9.6 and 4.2 minutes.

Population parameters are estimated in simulation based on multiple runs (i.e. multiple replications) of a model. This is illustrated in Figure 10.5. and shown below, where X_{ij} is the jth observed value (e.g., time in system for the jth customer) in the ith replication. The mean of a performance measure in each replication is the average of all of the observed values in each replication (e.g., the time in the system for all customers processed in a replication). Thus, the output of each replication is an average (e.g., average time in system), and it becomes one sample value.

$$\begin{array}{llllll} X_{11}, & X_{12}, & X_{13}, & \cdots & X_{1m_1} & \rightarrow \bar{X}_1 \\ X_{21}, & X_{22}, & X_{23}, & \cdots & X_{2m_2} & \rightarrow \bar{X}_2 \\ \cdots & & & & & \\ X_{n1}, & X_{n2}, & X_{n3}, & \cdots & X_{nm_n} & \rightarrow \bar{X}_n \end{array}$$

In Figure 10.5 a simulation model, of an M/M/1 queueing system in this case, is replicated four times to obtain four independent average time-in-system values. The first two values are from Figure 10.3. The estimated average time a customer spends in the system is 8.65 minutes. This is based on the system running 1,000 minutes or receiving approximately 1,000 customers (based on the mean inter-arrival time of 1.0 minutes).

Note that while the individual values that comprise the mean for each replication are typically correlated, the mean values are not; they are independent. This is because in

discrete-event simulation software, each source of randomness uses independent random number values to do the sampling. This is important since most statistical procedures assume independent sample observations.

In simulation analyses we have no idea the value of the true average, and again, if we did, we would not need to simulate; however, in the example we can compare the estimated average time in the system obtained by simulation, 8.65 minutes, to the long run average value obtained by using the equation for W from Queueing Theory and provided in Chapter 2; that value is 9.0 minutes. These values are quite close, but only by coincidence because (1) the sample size of four is too small to provide much confidence in our estimate, (2) the simulation is run for 1,000 minutes (approximately 1,000 customers) but Queueing Theory assumes a very long operational time, and (3) Queueing Theory assumes the system is operating under steady-state conditions. Based on the time series in Figure 10.3, the simulated systems do not appear to be in steady state. Steady-state conditions are discussed in more detail later in the chapter.

A natural follow-on question is "is the estimate of system performance obtained from the simulation model good enough?" Since the population value is not known, the question cannot be directly answered; however, a statement can be made about the precision of the estimate. The mean is considered to be a point estimate. To address precision, an interval estimate, referred to as a confidence interval estimate, needs to be determined from the sample data. With an interval estimate, statements such as "we are 90% confident the true mean is between two calculated limits" can be made.

Figure 10.6 illustrates the notion of point and interval estimates using the simple M/M/1 defined earlier. In this case (1) the average time in the system is the population parameter of interest, (2) ten replications (n=10) were made of the model, and (3) each replication was run for 1,000 minutes, resulting in approximately 1,000 customers arriving to the system in each replication. The individual replication values are shown in the table on the left-side of the figure and in the box-and-whiskers plot. The average time in system values range

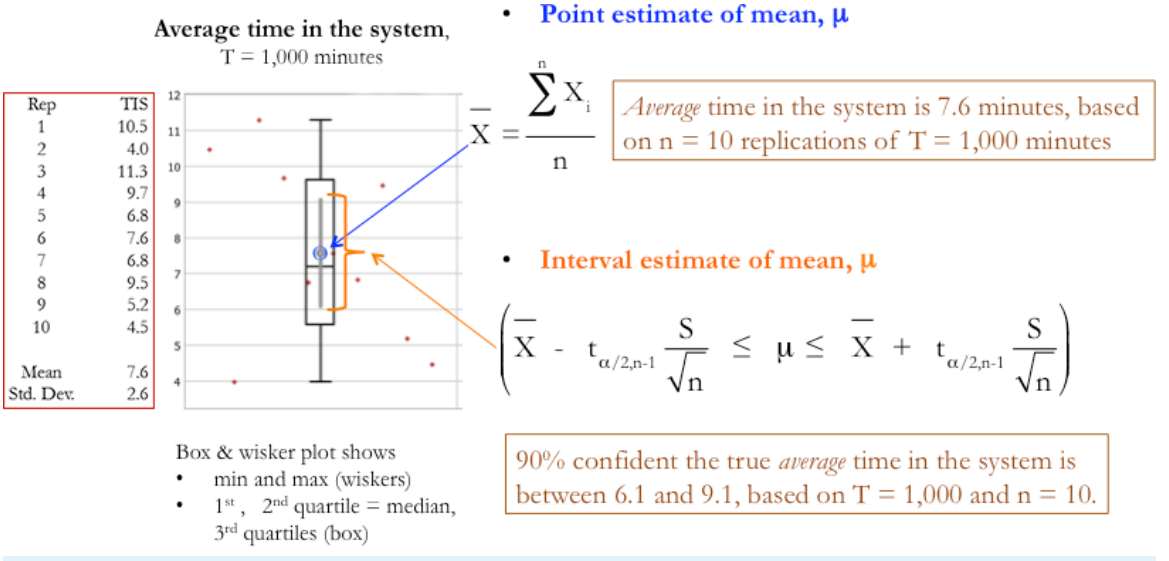


Figure 10.6: Point estimate versus interval estimate.

between 4.0 and 11.3 minutes. The estimated population parameter, average time in system, is 7.6 minutes and is the average of the replication values. (Recall the average service time is 0.9 minutes for this system where the server utilization is 90%.)

The interval estimate, or confidence interval, provides information on the precision of the estimate. In the example in Figure 10.6, there is 90% confidence that the true mean is between 6.1 and 9.1 minutes. The width of the interval (value of the upper limit minus the value of the lower limit) measures the precision of the estimate and indicates there is risk in our estimate due to sampling error. There is always uncertainty in estimates since the true population value is unknown unless the entire population is considered, (i.e., n = N).

The width of the confidence interval, the measure of precision, is affected by

1. how confident we want to be. Specifying more confidence results in a wider interval. The confidence is set by the α value for the t-distribution. If we want to be 90% confident, α = 0.10., if 95% confident, α = 0.05, etc.
2. the variability in the observed values. This is measured by the standard deviation of the replication values and is denoted as S. As the variability in the sample increases, S increases, as does the width of the interval, but precision decreases. Recall from basic statistics that the sample standard deviation is

$$S = \sqrt{\frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2}{n-1}}$$

3. the number of replications or sample size, n. The larger the sample size, the narrower the interval.

Careful statistical analysis is paramount when using output from a discrete-event simulation model. In order to draw conclusions and make valid inferences, simulation models must be replicated; that is, the models must be run multiple times. The information gathered from the multiple runs is combined to provide a basis for inference. Confidence intervals are used to estimate the means of output or response variables. Methodologies for making inferences from simulation models are discussed later in the chapter (and in the next chapter), following a discussion of some important tactical experimentation variables that must be set in order for the model to generate the data needed for analysis.

Section 10-2 Key tactical experimental design decisions

As previously discussed, models are built for analysis and experimentation and the experiments needs to be designed and considered early in a simulation project. The following are the key activities for preparing a simulation model to be effectively used for analysis:

1. Identify and define the experimentation variables:
   a. Performance measures are used in the decision making and problem solving processes to assess system performance. These are also commonly referred to as outputs of the simulation and are also called response variables, dependent variables, or statistics.

   b. Decision variables are the controllable inputs to a simulation model that are manipulated to represent different system characteristics. They are also referred to as independent variables.

2. Establish the type of system (i.e., terminating or non-terminating);
3. Determine the number of replications;
4. Define the starting or initial conditions;
5. Define the run length (how long the simulation is run for each replication).

Each of these activities is described in the following subsections.

Experimentation variables

The output or performance measures must be identified and defined early in a simulation project so that it can be ensured that the model is developed so that it provides the needed output. All simulation software packages contain many and varied types of standard measures of system performance, such as throughput, average waiting times for key resources, etc. If only standard measures are needed to solve a problem or support a decision, then little effort is needed during model development to access the measures.

However, many times a performance measure that is needed to decide the best option for improvement is not a standard output of the software and thus needs to be included through coding. For example, in some systems conformance to schedule or sequence is an important measure of system performance. Items can get out of sequence in a production process due to priorities, quality issues, etc.; therefore, a simulation model of the system needs to track how well an initial sequence of items is maintained as the items progress through the system. This is not a standard measure in simulation software so it needs to be clearly defined and incorporated into the model. In this case, part of the model development activities needs to include time for developing the means for capturing and summarizing the sequence measure(s) in the simulation model.

The model must also be developed in a way that facilitates the changing of key inputs, the decision variables. Input may be through tables or graphical user interfaces that are a part of a simulation model or may be obtained from external sources such as spreadsheet and database files. In any case, time must be allocated during model development to create the needed tables, interfaces, or links to input data.

Type of system

Early in a simulation project, it must be determined whether the system being considered is a terminating or non-terminating system. While this decision generally does not affect the modeling of the system, it does affect how experimentation and analyses will be conducted. This section only defines the terms and describes the operation of the differing systems.

How the system type affects the design of the simulation experiments and analyses are defined in subsequent sections.

Terminating systems are those that have well-defined starting and ending points. These points define a period of interest during which system performance is measured. Typically, these points correspond to specific times. For example, a clinic is interested in patient wait time during its eight hours of operation, or a restaurant is interested in the time customers wait for a table during a 2.5-hour lunch rush period. Terminating systems may also be defined in terms of a state of a system, in addition to time. For example, a simulation of the operation of a clinic may stop when the system is empty of customers after a specified closing time; that is, a clinic may accept customers for eight hours, but it continues to provide services until all customers have been processed. In this case, the closing time of the clinic is a random variable. In fact, simulation may be used to help define the time to stop accepting customers so that there is a high chance that employees can leave the clinic by a certain time (e.g., 5 p.m). If a simulation is terminated at the closing time, then customers may remain in the system with service incomplete.

Figure 10.7 shows the number of customers in the system during the simulation of the simple M/M/1 system that has been used in this chapter (exponentially distributed inter-arrival times with mean 1.0 minutes and exponentially distributed service times with mean 0.9 minutes); however, in this case, arrivals stop after 480 minutes (8 hours), but the simulation does not end until all customers in the system have exited. In the first case, all customers exited just eight minutes after the arrivals stopped. In the second case, the simulation ended after 510 minutes; it took nearly an additional 30 minutes to process the customers that were in the system when the arrivals stopped.

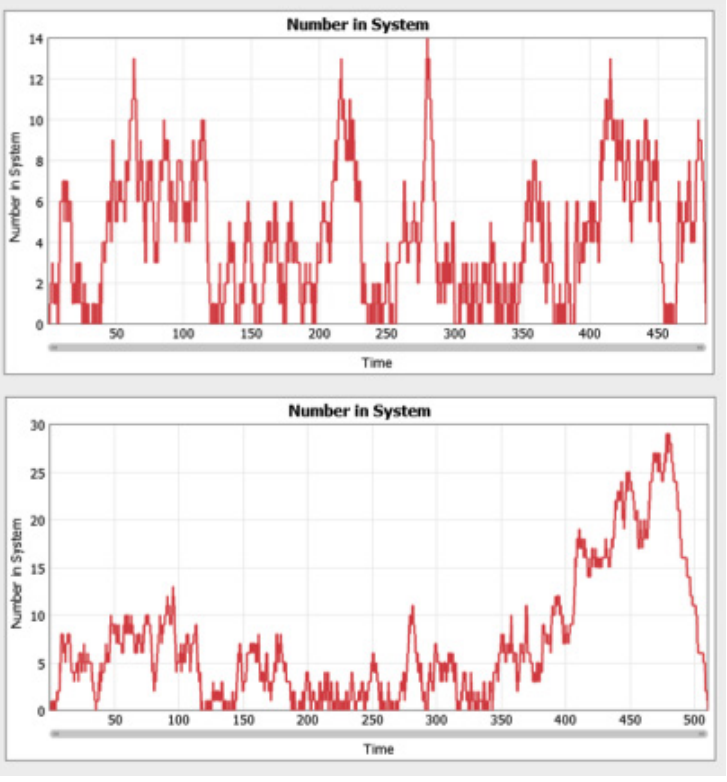


Figure 10.7: Example terminating system; system closes at 480 but system empty at 488 and 510.

Typically, a terminating system does not reach steady-state conditions during its specified operational period. Steady state means that the distribution of the performance measures settles down and approaches a constant. It does not mean that individual values, such as wait time, converge to a constant. For example, in steady state, an average may only fluctuate

slightly over time even though the values that make up the average continue to vary. In the clinic example, many performance measures will not reach steady state because the clinic is not open long enough for conditions to settle down. It would be inappropriate to use steady-state measures, such as those from Queuing Theory, or running a simulation model for a very long period, to assess system performance and compare alternatives.

Figure 10.8 shows the average time customers wait for service during the simulation for the second case in Figure 10.7 (arrivals stop after 480 minutes, but the system takes an additional 30 minutes to finish processing all customers). The average wait time clearly does not reach steady state; there are considerable fluctuations in the average, and it is rapidly increasing at the end of the simulation. Since the example is of a simple M/M/1 system, the long-run, steady-state average wait time is 8.1 minutes based on the Queueing Theory equation for Wq in Chapter 2. Obviously, based on this single replication, the system does not reach steady state in 480 minutes (or during approximately 480 arrivals).

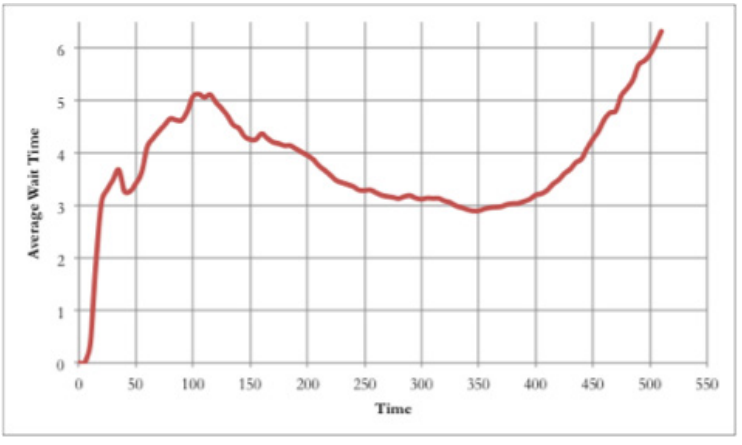


Figure 10.8: Average wait time over the length of the simulation.

Non-terminating systems are those that typically run for a very long period of time because, as the name implies, there is no known ending or terminating point. In non-terminating systems the focus is typically on long-run or steady-state performance. As shown in Figure 10.9 non-terminating systems go through a transient period and then possibly reach steady state. Again, steady state does not mean that individual values converge to a

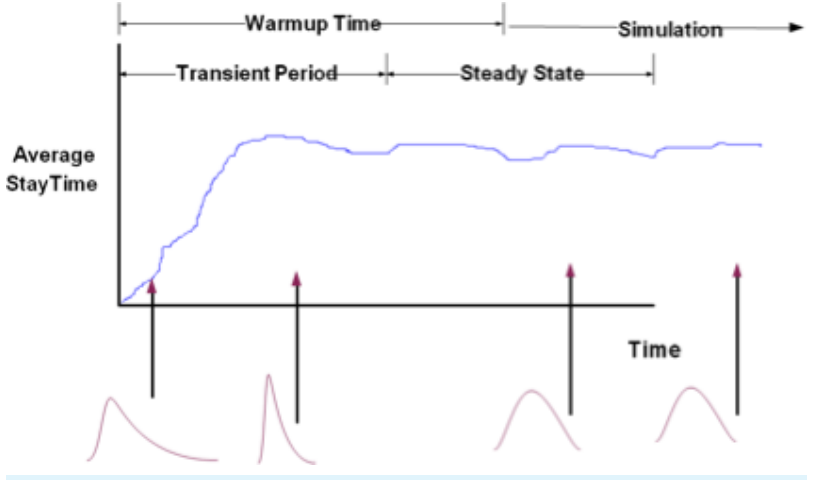


Figure 10.9: Simulation of non-terminating system and estimating steady-state measures.

constant, just that the underlying distribution settles down and approaches a constant. This is illustrated at the bottom of Figure 10.9, where the distribution of the state variable is quite skewed early in the simulation but becomes symmetrical as the simulation progresses through the transient period into steady state. The warm-up time shown at the top of Figure 10.9 is explained in the section below that discusses the run length of a simulation replication.

Number of replications

As discussed earlier, in order to avoid making inferences on a single, possibly extreme value, a model should be replicated a number of times and the values of the performance measures from each replication averaged. The general intent is to conduct experiments with the simulation model and estimate the population performance measures of interest, such as the mean.

Since the value of the population parameter will never be known with certainty, the more runs that are used to estimate the mean the better; that is, the larger the sample size, the better we feel about the estimate (i.e., the estimate is more precise). This was discussed in Section 10-1; however, there is a price to obtaining an estimate that is based on a large number of samples or replications. A complex model may take a long time to run, and there may not be time to run a model many, many times, especially if there are a considerable number of options or scenarios to consider.

The number of replications can be obtained from the following confidence interval formula that was presented in Figure 10.6:

$$\left(\bar{X} - t_{\alpha/2,n-1}\frac{S}{\sqrt{n}} \;\leq\; \mu \;\leq\; \bar{X} + t_{\alpha/2,n-1}\frac{S}{\sqrt{n}}\right)$$

In general, the confidence interval limits are obtained by adding and subtracting a calculated half-width value, *hw*, to/from the sample mean.

$$P\left(\bar{x} - hw \;\leq\; \mu \;\leq\; \bar{x} + hw\right) = 1 - \alpha$$

Thus, for estimating the population mean, the half width *hw* is:

$$hw = \frac{\left(t_{n-1,\alpha/2}\right)s}{\sqrt{n}}$$

A table of t-values is provided in Appendix 10A [2] for various values of n and α.

If the half width becomes the desired precision, *e*, then the above equation can be solved in terms of *n*. Since *n* is involved in two terms in the equation, the t-value from the student-t distribution is approximated by the standard normal distribution value z. A standard normal table is provided in Appendix 10B [3]. Thus, the number of replications, *n*, becomes

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

This is the same as the sample size formula discussed in basic statistics courses.

In order to determine the appropriate sample size, a pilot run of the simulation model is needed to obtain an estimate of S, the standard deviation.

The process used to determine the number of replications is illustrated by using the example from Figure 10.6. In that case, the model was replicated 10 times (i.e., the pilot run, and the estimated standard deviation of the time in the system is 2.6 minutes). If we want to be 90% confident of the estimate, then α = 0.10 (two tails). The standard normal value, z = 1.645, is obtained from the table in Appendix 10A. Also, assume we want to be within ±1 minutes of the true mean time in the system (i.e., e = 1).

$$n = \left(\frac{(1.645)\, 2.6}{1} \right)^2 = 18.29 \rightarrow 20$$

We can see that at least 19 replications are needed. To be conservative, the number of replications should always be rounded up, in this case to 20, but even 25 replications could be used. The larger value is justified because (1) the estimate of the standard deviation is based on the pilot run, which is a small sample and (2) the t-distribution is approximated by the normal distribution, and the t-values are always larger than corresponding z-values.

Notice that if we wanted to be 90% confident that our estimate is within 0.5 minutes of the true mean, then we would need at least 74 replications. Generally, if *e* is halved, then *n* is quadrupled; similarly, *n* can be halved by doubling *e*.

Sometimes it is more convenient to express the *e* in terms of a relative measure, rather than in absolute terms. For example, instead wanting to be within ±1 minutes of the true mean (absolute error), it may be easier to say within 5% of the mean (relative error). Since the population mean is unknown, the sample mean can be used; therefore, if the relative error is *re*, then *e* is:

$$e = \left(\frac{re}{(1 - re)} \right) \bar{X}$$

For the example above, where the sample mean is 7.6, if we want to be within 5% of the mean, then *e* = 0.4. The number of replications in terms of relative error is

$$n = \left(\frac{z_{1-\alpha/2} \ s}{\left(\frac{re}{(1-re)} \right) \bar{X}} \right)^2$$

When multiple performance measures are being considered, n should be determined for each measure and then the largest value should be used for experimentation.

In summary,

- A simulation model should replicated n times, where (1) n is based on a pilot run to estimate the standard deviation, (2) the desired precision is specified, and (3) the desired level of confidence is specified.
- Based on the n replications, an interval estimate of the population mean is determined so that we are 100*(1-α) percent confident that the true mean is between the calculated limits. If the width of the confidence interval is too wide, then the number of replications is typically increased.

Starting conditions

By default, most simulations start "empty and idle"—empty because there are no customers in the system and idle because the resources are available but not yet working; however, how a simulation starts can impact performance measures, especially in short simulation runs, which are more common in terminating systems. For example, in a clinic, there may be patients waiting for the clinic to open, or in a restaurant at lunch there may be some customers in the system from the pre-lunch period. If there are customers in the system at the start of the simulation, either awaiting entry or already in service, the model needs to generate the typical number of items and their state at time 0. The number in the system at the start of the simulation could be a random variable.

Figure 10.10 illustrates the effect of starting conditions using the simple M/M/1 system example (mean inter-arrival and service times of 1.0 and 0.9, respectively). Each of the two simulations are run for 100 minutes (approximately 100 arrivals). Also, both simulations

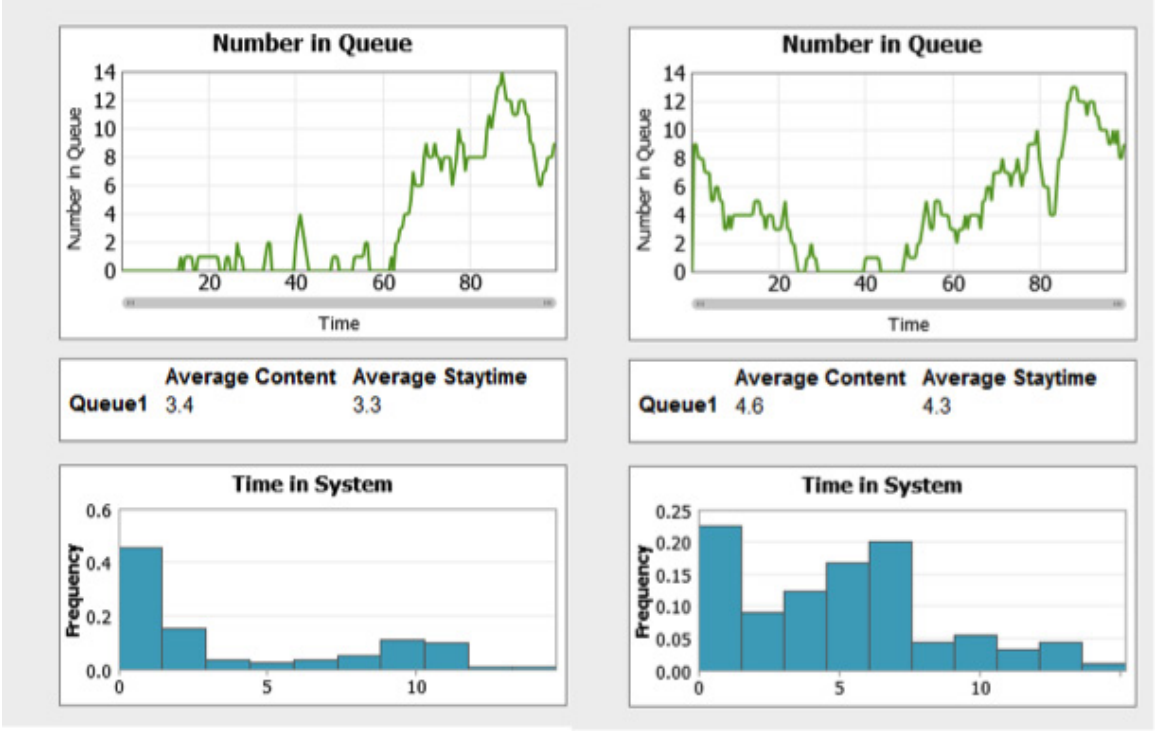


Figure 10.10: Two starting conditions, 0 and 10, of M/M/1 system with run length = 100.

use the same inter-arrival times and same service time; the only difference is that either 0 or 10 customers arrive at time 0. The case on the left started empty and idle; the case on the right started with 10 customers entering at time 0. Obviously, the different initial conditions result in quite different behavior, as illustrated by the different dynamics in the time series plots. Also, there is about a 25% difference in the mean number in queue and mean wait time. Similarly, the distributions of the time in the system, shown by the two histograms in Figure 10.10, are quite different. Of course, this comparison is based on a single replication, but in general, initial conditions can affect the observed system performance.

Starting conditions have less impact as the length of the simulation run increases; however, starting conditions should always at least be considered when simulating terminating systems. For non-terminating systems, the length of the simulation is usually quite long, especially when trying to estimate steady-state performance. While the potential start-up bias resulting from not properly initializing the system becomes less of a factor, it can impact how long the system takes to reach steady state. Thus, starting conditions could affect run length, as discussed in the next section.

Length of a simulation run

The length of a simulation, or its run time, depends on the type of system being modeled—terminating or non-terminating.

For terminating systems, determining the run length is usually straightforward: the simulation runs until the termination event occurs, which may be a specified time, such as the closing of a clinic or the end of a restaurant's rush period, or may be at a specified event, such as when the system becomes empty after some operational time. Also, in most simulation software, when there are no events in the events list to process, the simulation stops running (e.g., no items remain to be processed).

Determining the length of time to run a simulation model for non-terminating systems is nowhere near as straightforward as it is for terminating systems. It is problematic because there is no single agreed-upon approach for successfully determining the run length of a non-terminating system, and many of the techniques require sophisticated analysis techniques. A later version of this text may explore some of those approaches; however, for now, we suggest using a series of plots, one for each performance measure of interest, similar to Figures 10.8. and 10.9. We also suggest the plots include at least a few replications. For more information on output analysis of non-terminating systems see, Currie and Cheng [1], Law [4], or Robinson [5].

Replications are still important even for models that are run for a very long time. This is illustrated by the two replications in Figure 10.11. The two independent replications of 100,000 minutes (approximately 100,000 arrivals) to a M/M/1 system provide different estimates of system performance. One results in an average number in queue of 8.7, the other 7.8. Similarly, the estimated average waiting times are 8.7 and 7.8 minutes. The steady-state average queue length and wait time are both 8.1, using the M/M/1 equations from Chapter 2.

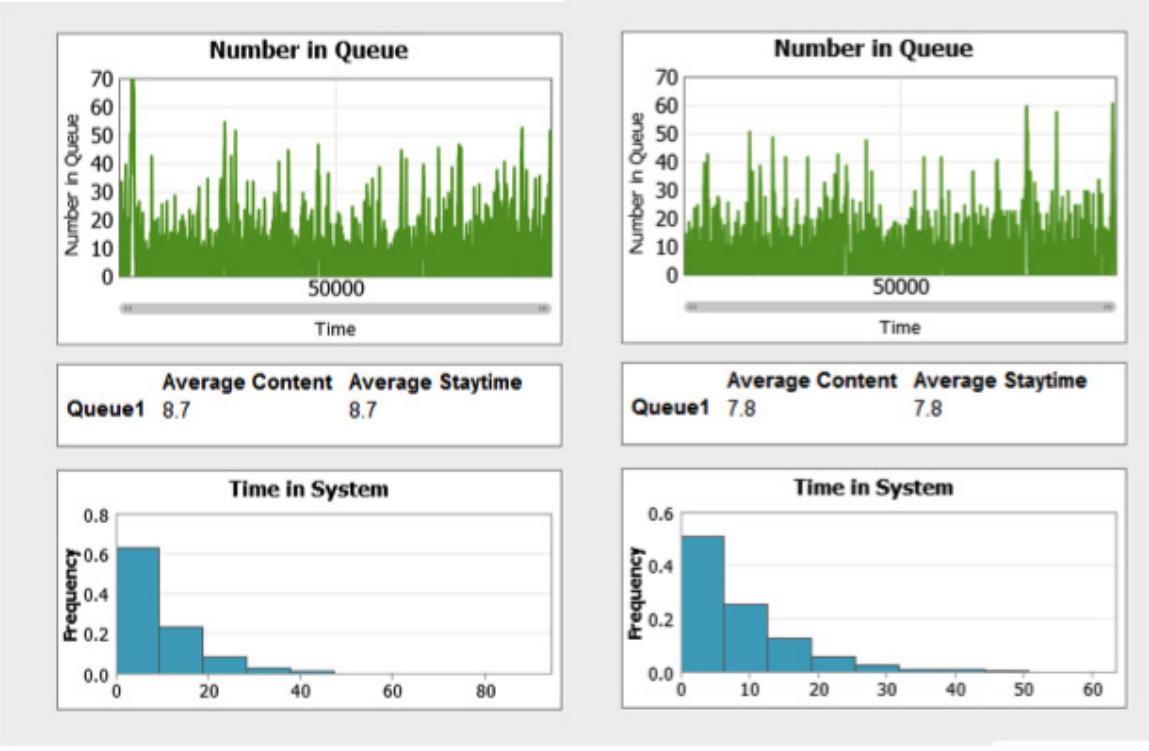


Figure 10.11: Two replications of M/M/1 system with run length = 100,000.

In order to get a sense of when a measure is in steady state, a time series plot of the measure is useful. For example, Figure 10.12 shows the plot of the average time in the system for four replications of a simple M/M/1 system. Each replication is 50,000 minutes (approximately 50,000 arrivals) and begins empty and idle. Even though the vertical axis is not scaled consistently, it should be clear that there is still considerable variability in the mean over time due to the system dynamics. The plots can be used to define the appropriate length of the simulation to estimate steady-state performance by identifying where the measure tends to level off. The vertical dashed lines correspond to 20,000, 30,000, and 40,000 minutes (approximately equal to the number of arrivals). For each replication at these levels the graphs include an error bar that corresponds to ±1 minute. This tends to standardize the plots in terms of the vertical scale by considering the relative position of the bars in each plot. While it is a matter of judgment, it appears that the simulation model should be run at least 20,000 minutes (20,000 arrivals) to provide a good estimate of steady-state performance. Beyond 20,000, the error bars tend to be horizontal or flat.

Throughout the discussion of output analysis, the length of the simulation run has oftentimes been expressed both in time units (minutes) and in terms of the number of arrivals. Of course, time units are the most typical characterization of run length, but to estimate steady-state conditions, the number of cycles, in this case the number of arrivals, is more meaningful because steady-state conditions typically depend on the number operations performed (i.e., number of cycles).

The horizontal dashed line in each time series plot in Figure 10.12 corresponds to the actual steady-state mean time in the system, 9.0 minutes, based on the M/M/1 Queueing

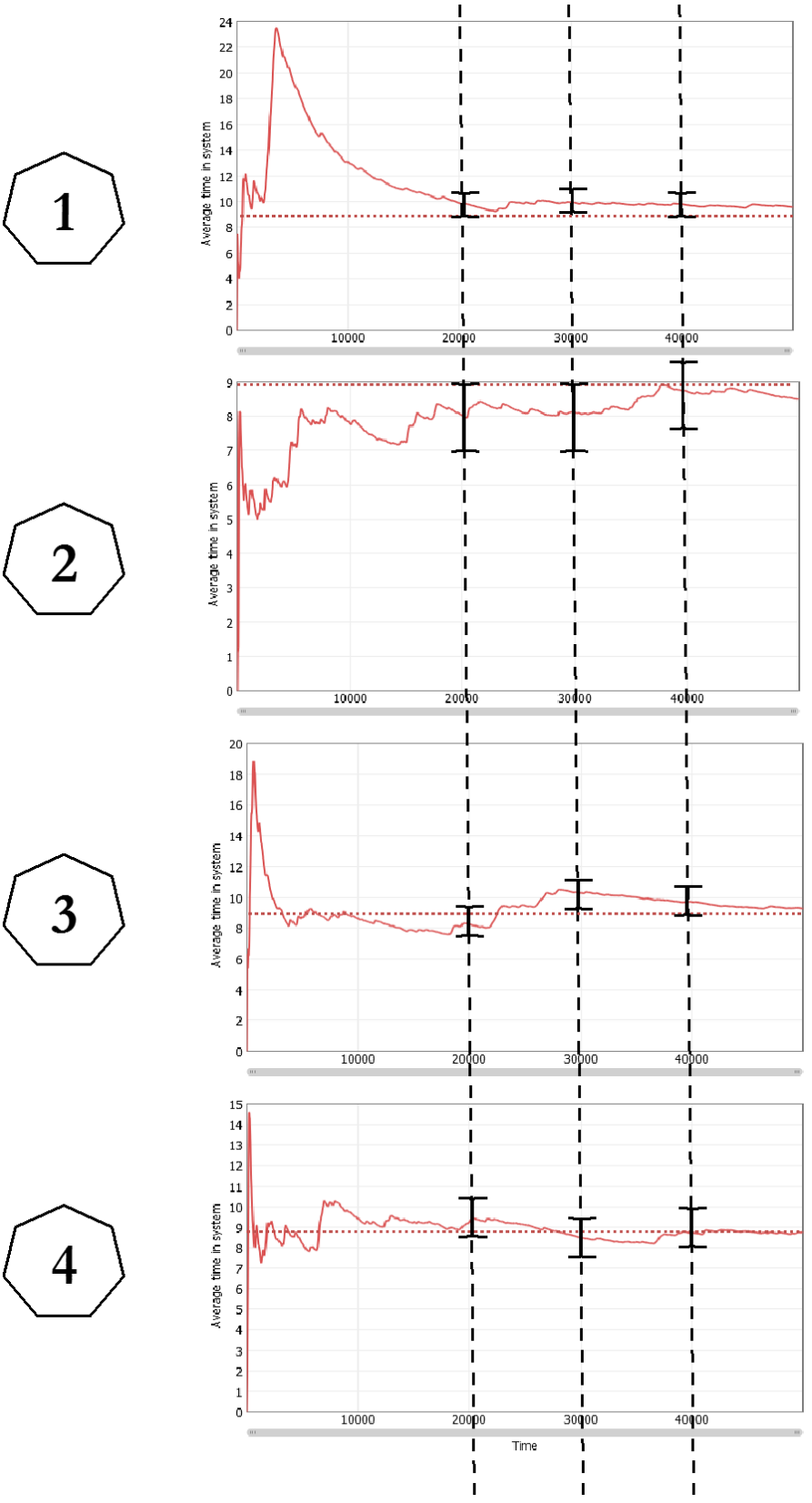


Figure 10.12: Four replications of average time in an M/M/1 system, run length = 50,000.

Theory equation for *W* in Chapter 2. Of course, the true mean is rarely known in a simulation project.

As was shown in Figure 10.9, in order to reach steady state conditions, a system must pass through a transient period. The transient period is oftentimes biased by the initial or starting conditions of the model. One approach for dealing with the transient period is to delete some of the initial observations and only use the later ones to calculate the performance measure. The early observations are typically not representative of steady state since they are affected by the starting conditions for the simulation (e.g., empty and idle). The period where the observations are deleted is called the warm-up period.

When a simulation reaches the warm-up time, all statistical counters are reset and the reported output statistics are based on the time period beginning with the warm-up time and ending with the completion of the simulation run. Note that the conditions in the model are not reset, just the statistical counters. Most simulation software packages include the ability to specify the warm-up period as part of the model run settings or as part of its experimentation module. When the warm-up period is specified, the software automatically handles resetting the statistical counters and outputting measures that are based only on the simulation time after the warm-up period. While the warm-up calculations are done for all of the standard statistics that are provided as part of the software, they are not performed for any custom performance measures that are developed by the modeler in code or script; the modeler needs to include the appropriate logic for revising their performance measures to reflect the inclusion of a warm-up period.

An obvious challenge for using a warm-up period is deciding how many observations to delete (how long to make the warm-up period) and then how long to simulate after the warm-up period. As with determining the length of a simulation run in a terminating system, there are no easy answers. In fact, there is no one agreed upon methodology for selecting the warm-up period and the subsequent data collection period. In addition, most of the proposed methodologies are quite complex. Law [4] provides an excellent discussion of the issue and the varied approaches that are proposed in the literature.

We suggest plotting a few replications of alternative warm-up periods and data collection periods and then assessing their effectiveness. This is illustrated in Figure 10.13 and 10.14 for the simple M/M/1 system that has been used as an example in this chapter. Both figures show a time series plot of the number in queue and a plot of the average time in the system. Figure 10.13 shows two replications with a total simulation run length of 20,000 minutes (arrivals) and a warm-up period of 5,000 minutes; that is, the steady-state values are based on 15,000 minutes of operation of the system between times 5,000 and 20,000. Figure 10.14 shows the same two replications, but with a total simulation run length of 10,000 minutes (arrivals) and a warm-up period of 1,000 minutes; in other words, steady-state measures are based on 9,000 minutes of operation between 1,000 and 10,000 minutes. Note that the average time in system plots y-axes are not scaled the same, but this does not distract from the visualization of the dynamics and the effect of varying the warm-up conditions.

One issue that often arises in setting up a simulation is the relationship between simulated time and real time. If the operation being simulated runs continuously or 24/7

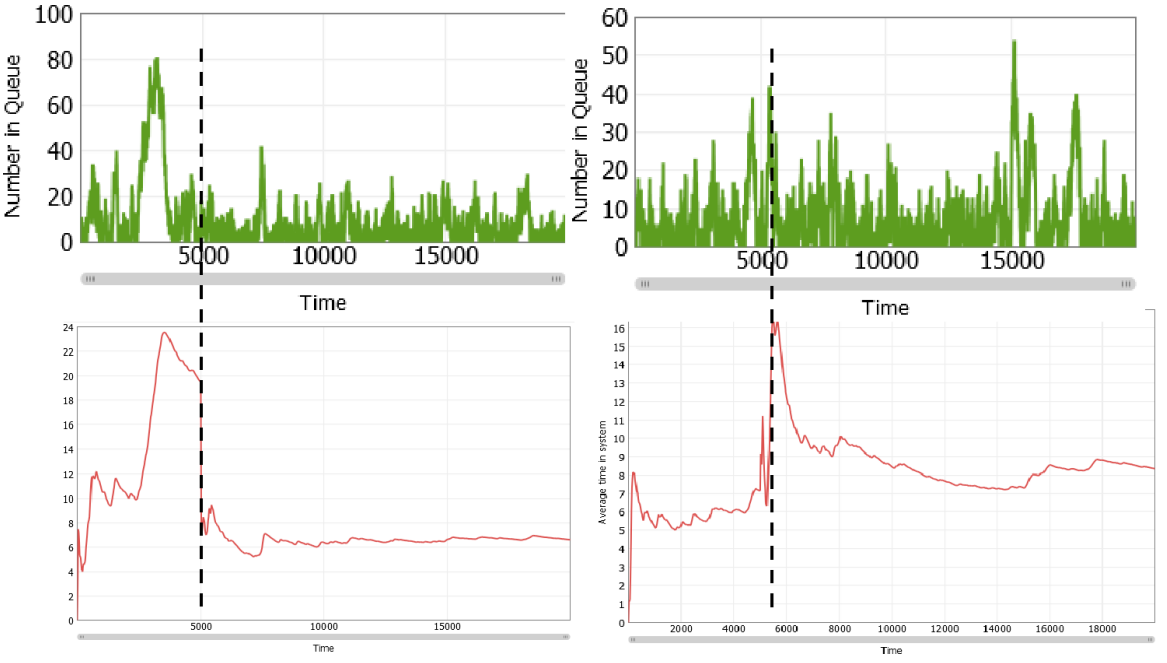


Figure 10.13: Two replications of an M/M/1 system, run length = 20,000, warm-up = 5,000.

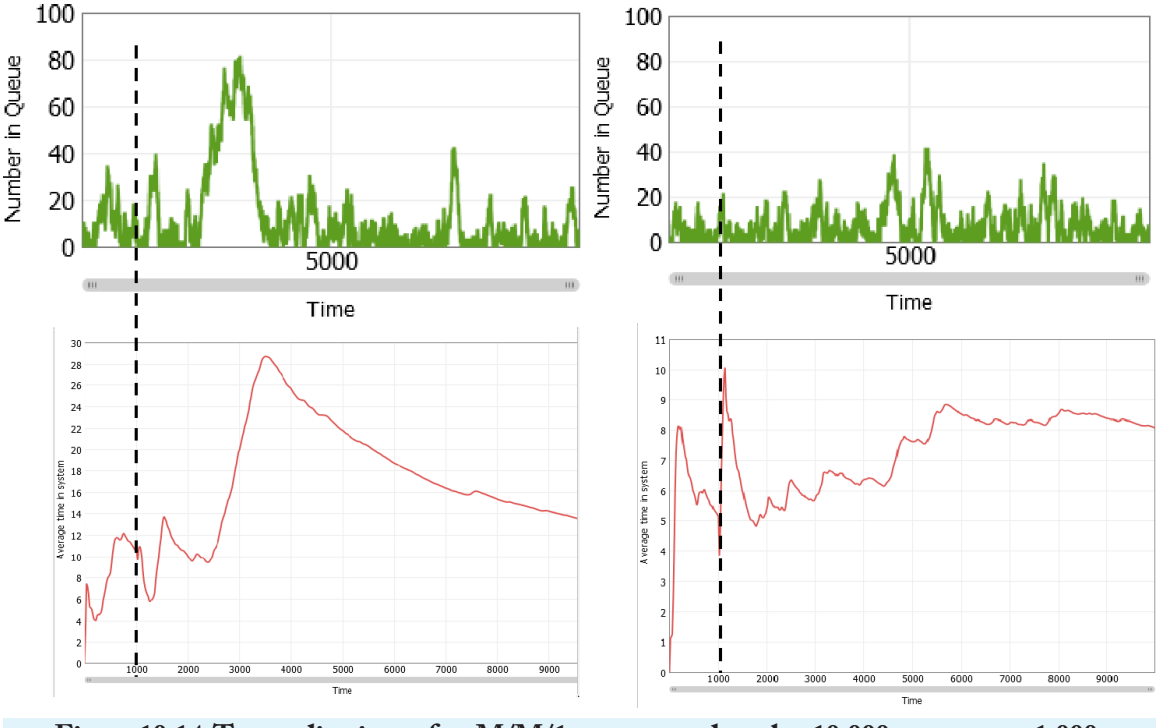


Figure 10.14: Two replications of an M/M/1 system, run length = 10,000, warm-up = 1,000.

(24 hours a day, 7 days a week), then there is a direct correspondence between system time and simulated time, as shown in the timeline (labeled System A) at the top of Figure 10.15. System A might represent a continuous manufacturing operation where workers come and go during the day but operations continue non-stop. The start of one 24-hour period picks up where the end of the last period stopped.

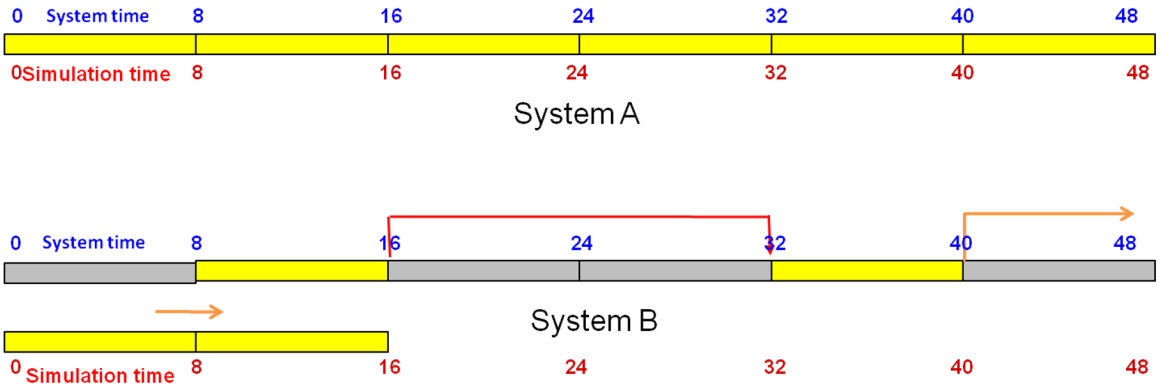


Figure 10.15: A state variable in a non-terminating system.

System B in Figure 10.15 might represent a small woodworking shop which operates 8 hours a day. At the end of the 8-hour shift work stops but, picks up again the next day where it left off. Since there is typically no need to simulate the time the shop is closed, the simulation may run the work time continuously and skip the closed time. That is, 8-hour periods are simulated back to back. In this case eight hours of simulation time actually represent 24 hours of system time. In order to represent a week of operation, only 40 hours of simulated time is needed and not 168 hours of clock time.

Section 10-3 Creating experiments using *FlexSim's* Experimenter

The basic structure for conducting experiments is shown in Figure 10.16. Experiments are composed of scenarios, and scenarios are composed of replications. A replication is a single run of a simulation model that generates a set of performance measures under a single set of conditions for a specified duration. A set of conditions defines a scenario; that is, each scenario is based on a set of variables for which values can be changed from scenario to scenario. These variables are typically the controllable or decision variables. Each scenario is replicated multiple times. An experiment is the set of all scenarios under consideration. An experiment allows in one execution the consideration of multiple scenarios and multiple replications of each scenario. This chapter considers only single-scenario experiments; experimentation using multiple scenarios is discussed in the next chapter.

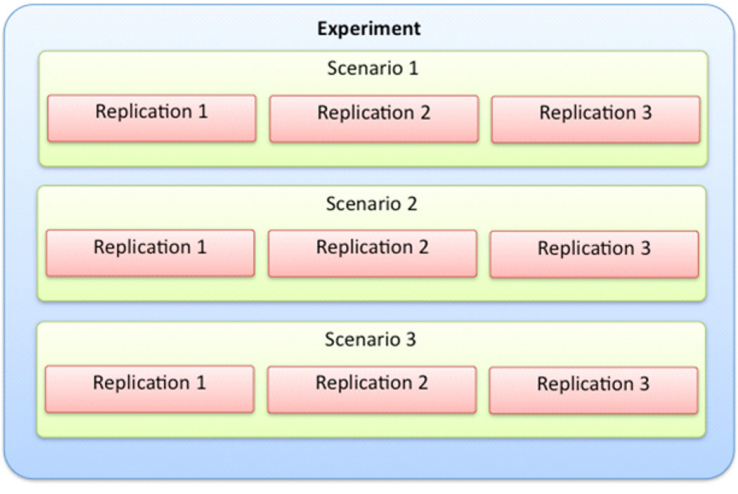


Figure 10.16: Replications, scenarios, and experiments.

All of the example models used in this book have been developed using *FlexSim*. Similarly, all of the analyses are conducted using *FlexSim's* Experimenter, unless otherwise noted. The Experimenter is accessed through

the Statistics drop down menu on the main toolbar and the primary interface, the Experiment Run tab, is shown in Figure 10.17. The Experimenter is configured by stepping through the tabs on the interface:

- Scenarios: Alternative model configurations or conditions that are being considered. This will be discussed in the next chapter since only a single scenario is considered here.
- Performance measures: Key variables that are used to decide which scenario is preferred
- Experiment Run: Run time, number of replications, and length of the warm-up period, if any
- Optimizer Design, Optimizer Run, and Optimizer Results: All pertain to optimization, or searching for the best solution; this is discussed in the next chapter.
- Advanced: Custom logic for additional control of scenarios and replications

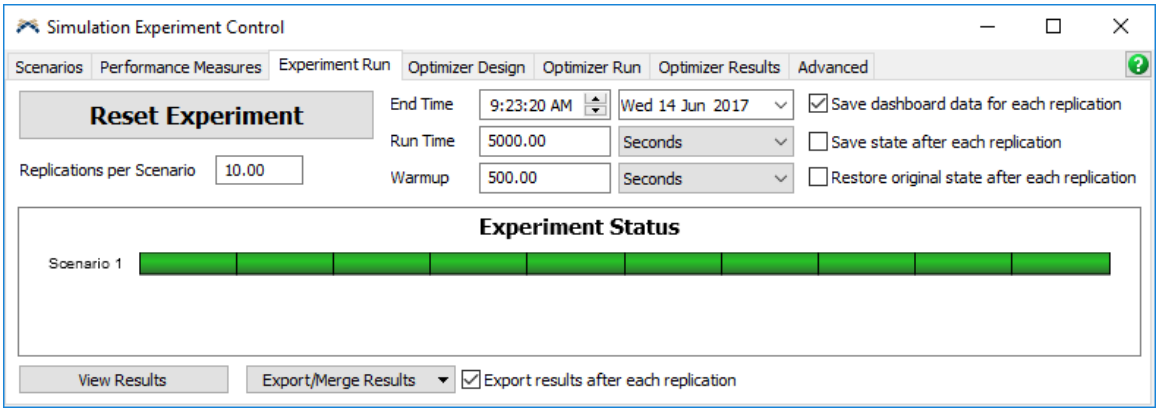


Figure 10.17: Main interface for controlling *FlexSim's* Experimenter.

The Performance Measures tab is used to define the output that is of interest in the simulation, such as average number in queue, average time in the system, or server utilization. Performance measures are primarily defined either from objects, the dashboard, or via custom code. Object-based measures include standard statistics, state percentages, and label values from individual or groups of objects. Measures may also be values from a global table or anything defined on a dashboard. Complex measures, or those derived from special-purpose algorithms, can be specified via custom computer code. These topics are described in Chapters 13 and 14.

The Experiment Run tab, as shown in Figure 10.17 is used to enter the number of replications to perform, the time length of each replication, and the length of the warm-up period, if used. In this case, ten replications are performed, each running 5,000 minutes, along with a 500-minute warm-up period; in other words, the values of the performance measures are based on 4,500 minutes with statistics being counted after the first 500 minutes.

Once all parameters have been defined, the Run Experiment button is pushed and the Experimenter tracks the values of the specified performance measures for each replication within each scenario. The Experiment Status section of the interface displays the experiment

in a grid format with scenarios as rows and replications of columns. As each part of the experiments is completed, each unit in the grid changes color from red to green. *FlexSim's* Experimenter is very powerful in that it takes advantage of the number of computer cores available and thus can perform multiple replications at the same time.

When all simulation runs that are defined in the Experimenter are complete, a variety of outputs are available for analysis by pushing the View Results button at the bottom of the interface. For example, in Figure 10.18 a replications plot is shown for the single scenario. The plot shows the performance measure values for average waiting time in the single queue for each of the ten replications. This plot is similar in format to the one shown in Figure 10.6. In addition to the plot of replication values, the box-and-whiskers plot shows the minimum and maximum values (at the ends of the whiskers), the 25th and 75th percentiles (the ends of the box), the 50th percentile or median, the mean, and the 90% confidence interval. Other information available on each performance measure include a frequency histogram, correlation plot, data summary (descriptive statistics and confidence interval), and the actual data values.

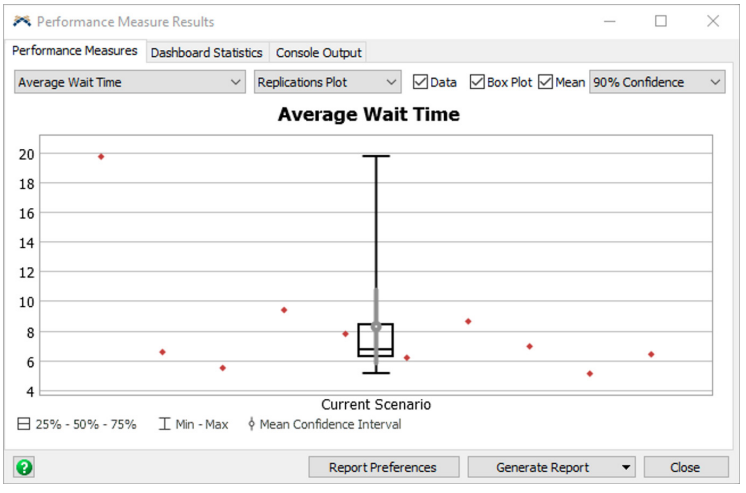


Figure 10.18: Example output from FlexSim's Experimenter (replication plot, single scenario)

All of the various results can be exported in HTML format to generate a custom report. Additional details for setting up the Experimenter and the types of graphs and results that are available, please refer to *FlexSim's* User Manual.

References

1. Currie, C. and Cheng, R. (2013) "A Practical Introduction to Analysis of Simulation Output Data," *Proceedings of the 2013 Winter Simulation Conference*, R. Pasupathy, S. H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

2. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/t-table.htm

3. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/NormTble.htm

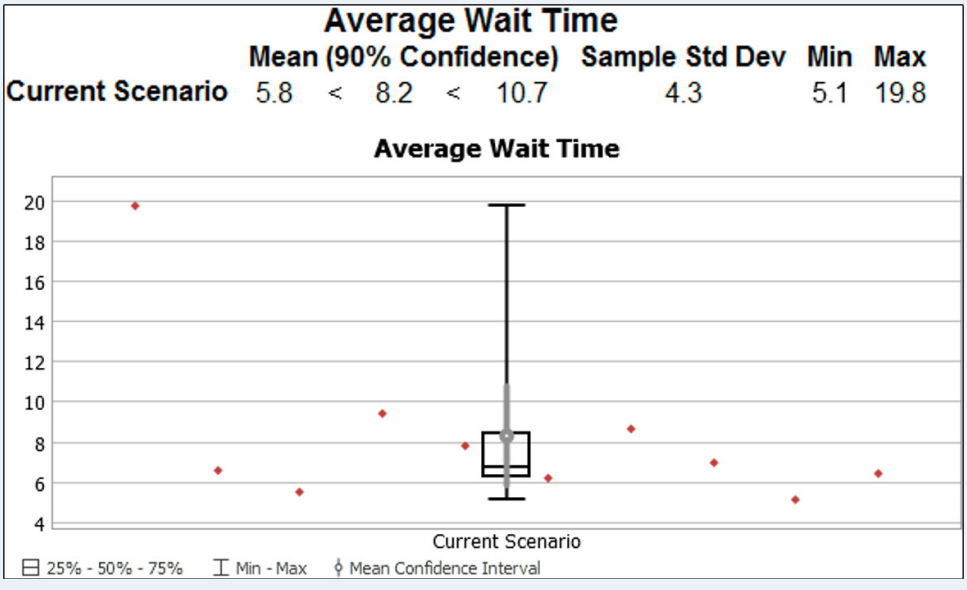
4. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

5. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, New York: John Wiley & Sons.

Chapter 10 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important—in the chapter.
2. A single run of a simulation model is called a(n) ---.
3. Briefly explain why simulations should be run more than one time; that is, conclusions should be based on more than one replication.
4. Discuss the difference between replications and scenarios.
5. Discuss the differences between terminating and non-terminating systems.
6. Consider each of the following system performance measures and indicate whether the system should be considered terminating or non-terminating.
   a. customer wait time at a fast-food restaurant between 11 a.m. and 1 p.m.
   b. inventory levels in a manufacturing cell that produces a commodity product
   c. throughput per week of a high-speed bottling line
   d. throughput per week in a job shop that produces custom products
   e. the time to produce a specified quantity of product
   f. patient wait time in a small, single doctor clinic
   g. the number of passengers waiting to clear security in an airport
   h. time between flight arrival and when baggage reaches its carousel
   i. the utilization of equipment used in aluminum production

7. Based on the information in the following figure, how many replications of the model should be performed in order to be 90% confident that your estimate is within 1.0 minutes of the true average wait time?



8. The data below are average patient waiting times, in minutes. They were obtained from 15 replications of a simulation model of a hospital imaging department.
   a. Provide a 95% confidence interval estimate of the true patient wait time.
   b. Determine the number of replications that would be needed in order to be 95% confident you are within 5 minutes of the true mean patient wait time.

Replication	Base
1	50.8
2	63.8
3	51.2
4	98.2
5	123.5
6	63.8
7	47.8
8	33.5
9	43.0
10	112.5
11	59.5
12	52.7
13	109.6
14	95.2
15	21.8
Mean	**68.5**
Std. Dev.	**31.3**

9. Jill owns a local copy shop and always finishes all work that came in during the day before she leaves. A simulation model of her shop is replicated 20 times with the following results:
   a. Provide a 95% confidence interval estimate of the mean time it will take to complete a day's work after closing.
   b. Determine the number of replications that would be needed in order to be 95% confident that you are within ten percent of the true mean time Jill spends in the shop after closing in order to finish the day's work.

	Base Case	
Rep	Close Time	Minutes after close
1	562.11	82.1
2	565.83	85.8
3	480	0.0
4	480	0.0
5	512.8	32.8
6	524.71	44.7
7	485.37	5.4
8	535.36	55.4
9	483.22	3.2
10	527.29	47.3
11	480	0.0
12	480	0.0
13	541.81	61.8
14	579.75	99.8
15	523.88	43.9
16	498.58	18.6
17	480	0.0
18	575.43	95.4
19	514.93	34.9
20	549.49	69.5
Mean	**519.0**	**39.0**
Std. Dev.	**35.0**	**35.0**

Appendix 10A – t-distribution table

The following table is in the public domain and isproduced by an APL program written by William Knight; accessed at: http://www.math.unb.ca/~knight/utility/t-table.htm

PERCENTAGE POINTS OF THE T DISTRIBUTION

Tail Probabilities									
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
D	1	3.078	6.314	12.71	31.82	63.66	318.3	637	1
E	2	1.886	2.920	4.303	6.965	9.925	22.330	31.6	2
G	3	1.638	2.353	3.182	4.541	5.841	10.210	12.92	3
R	4	1.533	2.132	2.776	3.747	4.604	7.173	8.610	4
E	5	1.476	2.015	2.571	3.365	4.032	5.893	6.869	5
E	6	1.440	1.943	2.447	3.143	3.707	5.208	5.959	6
S	7	1.415	1.895	2.365	2.998	3.499	4.785	5.408	7
	8	1.397	1.860	2.306	2.896	3.355	4.501	5.041	8
O	9	1.383	1.833	2.262	2.821	3.250	4.297	4.781	9
F	10	1.372	1.812	2.228	2.764	3.169	4.144	4.587	10
	11	1.363	1.796	2.201	2.718	3.106	4.025	4.437	11
F	12	1.356	1.782	2.179	2.681	3.055	3.930	4.318	12
R	13	1.350	1.771	2.160	2.650	3.012	3.852	4.221	13
E	14	1.345	1.761	2.145	2.624	2.977	3.787	4.140	14
E	15	1.341	1.753	2.131	2.602	2.947	3.733	4.073	15
D	16	1.337	1.746	2.120	2.583	2.921	3.686	4.015	16
O	17	1.333	1.740	2.110	2.567	2.898	3.646	3.965	17
M	18	1.330	1.734	2.101	2.552	2.878	3.610	3.922	18
	19	1.328	1.729	2.093	2.539	2.861	3.579	3.883	19
	20	1.325	1.725	2.086	2.528	2.845	3.552	3.850	20
	21	1.323	1.721	2.080	2.518	2.831	3.527	3.819	21
	22	1.321	1.717	2.074	2.508	2.819	3.505	3.792	22
	23	1.319	1.714	2.069	2.500	2.807	3.485	3.768	23
	24	1.318	1.711	2.064	2.492	2.797	3.467	3.745	24
	25	1.316	1.708	2.060	2.485	2.787	3.450	3.725	25
	26	1.315	1.706	2.056	2.479	2.779	3.435	3.707	26
	27	1.314	1.703	2.052	2.473	2.771	3.421	3.690	27
	28	1.313	1.701	2.048	2.467	2.763	3.408	3.674	28
	29	1.311	1.699	2.045	2.462	2.756	3.396	3.659	29
	30	1.310	1.697	2.042	2.457	2.750	3.385	3.646	30
	32	1.309	1.694	2.037	2.449	2.738	3.365	3.622	32
	34	1.307	1.691	2.032	2.441	2.728	3.348	3.601	34
	36	1.306	1.688	2.028	2.434	2.719	3.333	3.582	36
	38	1.304	1.686	2.024	2.429	2.712	3.319	3.566	38
	40	1.303	1.684	2.021	2.423	2.704	3.307	3.551	40
	42	1.302	1.682	2.018	2.418	2.698	3.296	3.538	42
	44	1.301	1.680	2.015	2.414	2.692	3.286	3.526	44
	46	1.300	1.679	2.013	2.410	2.687	3.277	3.515	46
	48	1.299	1.677	2.011	2.407	2.682	3.269	3.505	48
	50	1.299	1.676	2.009	2.403	2.678	3.261	3.496	50
	55	1.297	1.673	2.004	2.396	2.668	3.245	3.476	55
	60	1.296	1.671	2.000	2.390	2.660	3.232	3.460	60
	65	1.295	1.669	1.997	2.385	2.654	3.220	3.447	65
	70	1.294	1.667	1.994	2.381	2.648	3.211	3.435	70
	80	1.292	1.664	1.990	2.374	2.639	3.195	3.416	80
	100	1.290	1.660	1.984	2.364	2.626	3.174	3.390	100
	150	1.287	1.655	1.976	2.351	2.609	3.145	3.357	150
	200	1.286	1.653	1.972	2.345	2.601	3.131	3.340	200
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Tail Probabilities									

Appendix 10B – Standard normal distribution table

The following table is in the public domain and isproduced by an APL program written by William Knight; accessed at: http://www.math.unb.ca/~knight/utility/NormTble.htm

Probability Content from -oo to Z

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

Chapter 11 Multi-scenario Experimentation and Optimization

Objective

To enable users to design and correctly analyze experiments conducted with simulation models that involve multiple scenarios and multiple comparisons. This includes an introduction to finding the "best" alternative via optimization.

This chapter uses simulation models that were discussed in chapter 10 and expands their discussion to include multiple scenarios, including an introduction to optimization. Chapter 10 focuses on making inferences about system performance measures based on a single scenario; it introduced experimentation with simulation models in terms of its relationships to statistical inference, defined the key tactical experimental design decisions, and introduced *FlexSim's* Experimenter. Reading Chapter 10 is therefore a pre-requisite to this chapter.

All experimentation with simulation models begins with a set of activities that define key experimentation parameters. From Chapter 10, these activities include

1. identifying and defining the decision variables and the performance measures
2. establishing the type of system (terminating or non-terminating)
3. determining the number of replications to run per scenario
4. determining the conditions at the start of the simulation
5. determining how long to run each replication.

This chapter, using the foundational material from Chapter 10, discusses analyzing and comparing results from multiple scenarios of a simulation model, including

- comparing output obtained from two simulated alternatives

- using a variance reduction technique called Common Random Numbers (CRN)
- comparing output obtained from more than two alternatives
- finding the best solution (i.e., optimization)

These concepts and methodologies are explained through a comprehensive example that involves the design of a manufacturing cell. The system being considered is much more complex than the simple M/M/1 system used in Chapter 10. Section 1 defines the system and includes a preliminary analysis of the system.

Section 11-1 Definition of case study and preliminary analysis

A case study is used to explain and illustrate multi-scenario analyses. This section defines a production system that is being designed and the simulation model of that system. It also illustrates preliminary analyses that are performed prior to multi-scenario experimentation.

Definition of the production system

Good Vibrations is designing a work cell to automatically inspect its products. The cell conducts two types of sequential tests on automated inspection stations; the cell also includes buffer space prior to each inspection. Due to space limitations, a total of ten items can be stored in the buffers at one time. For example, if the buffer prior to the first machine has a capacity of two items, then the buffer prior to the second machine must have a capacity of eight items. If the buffer to the first machine is full, products are routed to a conveyor that is 38 meters (m.) long and travels at a speed of 10 m. per minute (min.). After traversing the conveyor, products wait at the end of the conveyor to enter the first buffer.

The cell inspects four equally likely (i.e., uniformly distributed) products, called A, B, C, and D; the inspection times for each product at each of the two machines are a constant 2, 4, 6, and 8 minutes, respectively.

Products are loaded onto each machine from the machine's buffer by a shared transporter. The transporter selects the product to be produced from the buffer and loads it onto the machine. The load time at the buffer and the unload time at the machine are each 15 seconds. The transporter travels at a speed of 20 m./min.

If a product that is to be inspected at a machine is different from the product previously inspected on that machine, then a changeover, or setup, is required. The setup occurs automatically and takes 30 seconds when it is required.

The products arrive to the cell from other areas within the facility. The average arrival rate of products to the cell is 7.5 products per hour. This corresponds to an average time between arrivals of 8 minutes (60 min./hr. ÷ 7.5 products/hr.). The distribution of the time between arrivals is unknown; it is assumed to be exponential. As noted above, there is a 25% chance an arriving product is of type one, 25% of type two, etc.

The inspection machines are not always available since they incur two types of downtime. The first downtime is the result of failures or breakdowns. The mean time between failures is two hours of processing time; the times are assumed to be exponentially distributed. The times to repair a breakdown are assumed to be uniformly distributed between 6 and 14 minutes. No other resources are needed for repair—the machine can repair itself. The second type of downtime involves periodic calibration checks by the machines. The time between checks is every 4 hours of clock time; the time to conduct a check is uniformly distributed between 3 and 7 minutes. No other resources are needed for checking; the machine checks itself.

Definition of the simulation model

A simulation model of the system described above is developed in *FlexSim*. A few views of the model of the system are provided in Figure 11.1. View A provides an overview of the work cell; view B shows the first inspection machine in the breakdown state; and view C provides a close-up of the transporter, the first inspection machine, and its buffer. Each view is briefly described below. The effective 3D graphics in the model makes it very easy to visualize the system and its operation, thereby greatly reducing the need for much textual description.

View A in Figure 11.1 provides an overview of the work cell: the two inspection machines, the two product buffer areas prior to each machine, the shared transporter, and the overflow conveyor. The 38-meter overflow conveyor is shown on the left side of the model view.

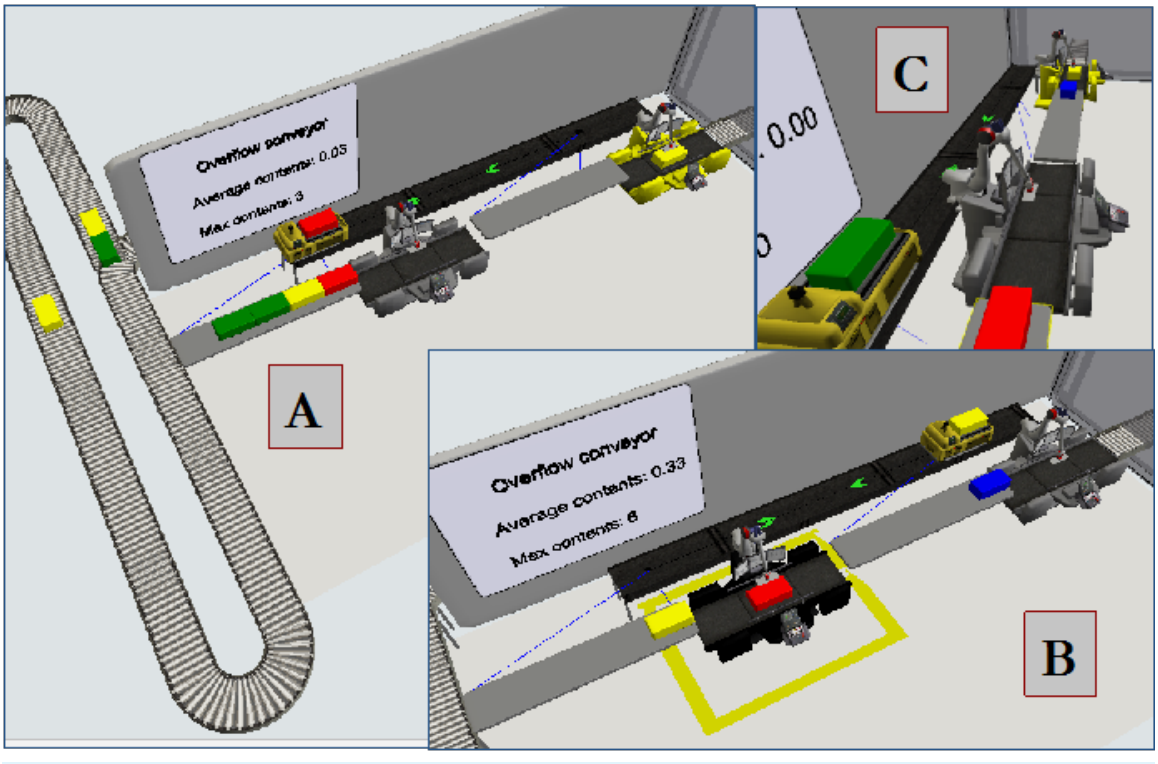


Figure 11.1: Model views of Good Vibrations' new production cell.

At the time the screenshot was made of the model in view A, the buffer prior to the first machine is limited to four items; therefore, it is currently full. As a result, two items are awaiting access to the buffer and a third one has entered the system and is traversing the overflow conveyor on its way to join the other two products awaiting access to the cell.

When the buffer sizes are changed in the model, the work cell is not rearranged to reflect the revised buffer size. For example, if the first buffer is made larger, the first machine is not actually moved closer to the second machine, as would be the case in the real system. All that is changed in the model is the capacity of the buffer. This is a simplification in the model that is assumed not to significantly affect system performance. While it is certainly possible to change the location of the machines in the model when buffer sizes change, it is a modeling decision not to do so. The modeler believes this detail does not justify the added complexity.

The four products (A, B, C, and D) are colored red, green, blue, and yellow, respectively. The use of color is a very effective way to help validate and verify a model. Through color, it is easy to observe the flow of items through a model and the states that objects are in. For example, in this model, while the machines are inspecting a product, they change to the color of the product type that they are inspecting; the second machine is inspecting product D (yellow) in model view A in Figure 11.1.

Color is also used to reflect when an inspection machine is idle or in a down state. If a machine is the breakdown state, then it is colored black, as shown for the first machine in model view B in Figure 11.1 (the first machine has a yellow box around it indicating a downtime). Once repaired and awaiting a product to inspect, the machine is colored purple. If a machine is the calibration checking state, then it is colored aqua. Once the check is complete and the machine is awaiting a product to inspect, it is colored brown. If a machine is just idle, then it is colored gray. In the cases of model views A and C, the first machine is idle; it is awaiting the transporter to deliver a product to inspect.

Model view C in Figure 11.1 shows the shared transporter moving a green product (type B) from the buffer to the first inspection machine.

An example dashboard of output from the work cell model is shown in Figure 11.2. In this case, as can be seen in the "Content vs Time" plot, the buffer capacity prior to the first inspection machine is eight and the capacity of the buffer prior to the second machine is two.

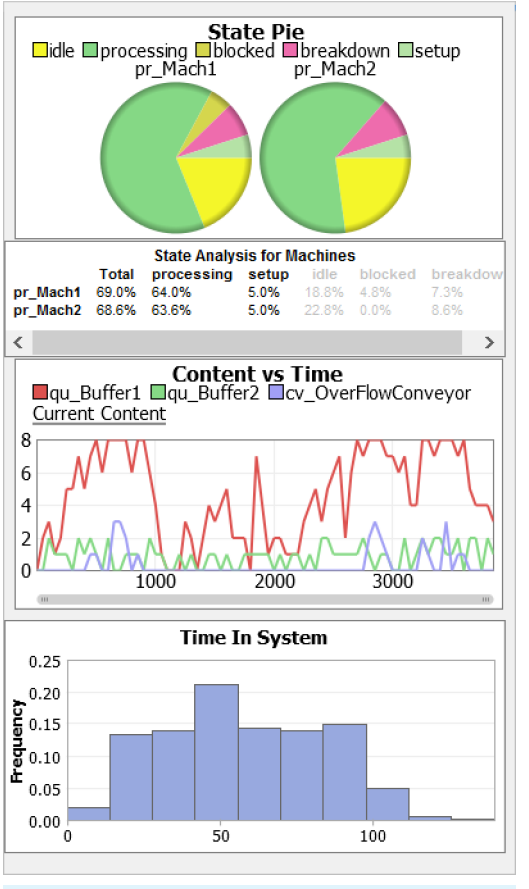


Figure 11.2: Dashboard of performance measures from the Good Vibrations model.

The two pie charts indicate the states that the inspection machines have been in up to the current time. For example, both machines have been inspecting about 65% of the time; they have been idle, down, blocked, or being setup during the remaining time.

Definition of the key experimentation design parameters

The objective of the simulation is to determine the size of the buffer at each machine prior to product inspection so that the average time that products spend in the system (the cell) is minimized.

Just as in the single-scenario case, described in Chapter 10, the experimentation process begins with a set of activities to define key experimentation parameters. These include

1. Experimentation variables
    a. Performance measure(s): average time in the system (the time a product spends between arriving to the system and either joining the first machine's buffer or traveling on the overflow conveyor and awaiting access to the first machine's buffer until the product finishes inspection at the second machine).
    b. Decision variables
        i. Primary: size of the buffers prior to the first and second machines, denoted as B1 and B2, respectively.
        ii. Secondary: queue discipline; the order in which products are selected for inspection from the buffer by the transporter; either FIFO (order in which the products arrive) or SPT (products with shortest expected inspection time first; i.e., red first if available, then green if available, etc.)
        iii. Secondary: speed of the transporter, denoted as, S1 (20 m./min.) and S2 (40 m./min.), respectively.
2. Type of system: terminating; significant changes in the products and/or the product mix is expected every two weeks; also every two weeks, considerable preventative maintenance is performed on the cell during the off hours. All products are out of the system prior to the beginning of maintenance.
3. Number of replications: to be determined below.
4. Starting/initial conditions: empty and idle; the cell will have been cleared out for preventative maintenance.
5. Run length: 80 hours (4800 minutes). All operations at Good Vibrations are on a single shift that operates eight hours per day, five days per week. Since the cell is restarted every two weeks (10 days), a single replication is 80 hours. Work remaining at the end of one shift is continued at the beginning of the next shift.

Pilot run

A pilot run is needed in order to determine the number of replications required for each scenario. *FlexSim* 's Experimenter is used to run the model for ten 80-hour replications with the buffers at the first and second machines set to 2 and 8, respectively. The other two

possible decision variables are set at their base level: FIFO for the queue discipline and 20 m./min. for the transporter speed.

The results of the pilot run are shown in Figure 11.3 in terms of Experimenter setup, Replications Plot, and data Summary. This is the same type of output that was discussed in Chapter 10 since the pilot run is a single scenario. The point estimate for the time in the system is 69.6 minutes, and the interval estimate indicates 90% confidence that the true average time in the system is between 39.4 and 99.9 minutes.

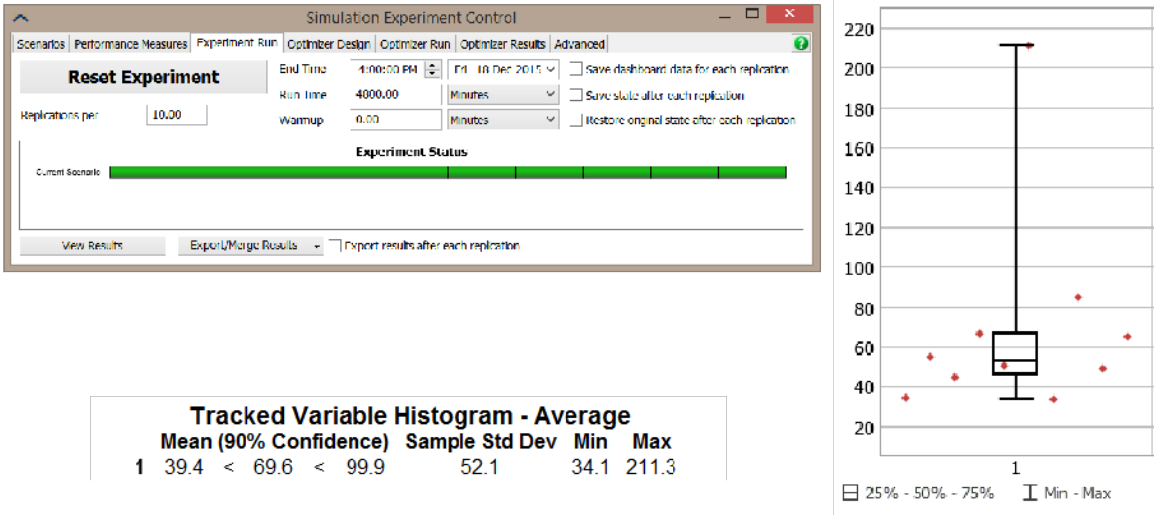


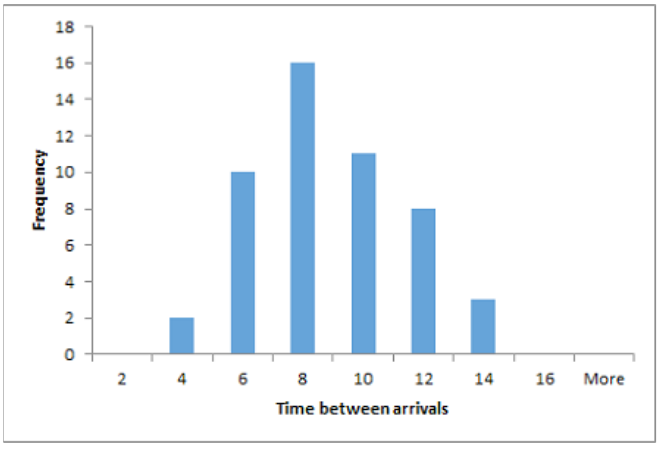
Figure 11.3: Results of pilot run.

Management's response to these initial results is that the estimate seems too high and it is not precise enough. Based on the pilot results and using the sample size formula from Chapter 10 at least 418 replications would be required in order to be 95% confident the estimate is within 5 minutes of the true mean. Because of this first step, the modeling team decided to check the model and try to find ways to provide a better estimate.

Response to pilot run - model changes

In response to management's feedback, the modeling team revisited all of their assumptions. One of the assumptions that lacked confidence was the distribution of the times between arrivals. In order to check the assumption, the team obtained a small random sample of inter-arrival times. A summary of that sample is provided in the histogram in Figure 11.4. The distribution does not appear to be exponential. The table on the right side of the histogram compares the frequency of the sample and what the sample frequency would have been if it were from an exponential distribution. A goodness-of-fit test from statistics confirms what is apparent from the comparison—it is unlikely that the sample came from a population that is exponentially distributed.

The modeling team used *ExpertFit*, as discussed in Chapter 8, to analyze how to use the sample data to model the randomness in the inter-arrival times. As a result, they decided



Time between arrivals	*Frequency*	*Expected frequency if exponenial*
2	0	11.1
4	2	8.6
6	10	6.7
8	16	5.2
10	11	4.1
12	8	3.2
14	3	2.5
16	0	1.9
More	0	6.8

Figure 11.4: Results from a sample of inter-arrival times (n=50).

to model the inter-arrival times as a continuous triangular distribution with the parameters minimum = 3 minutes, maximum = 15 minutes, and most likely = 8 minutes. Note that the mean of this distribution is 8 minutes, the same as that initially used for the exponential distribution.

The difference between the two distributions is that their shapes are quite different, and the triangular distribution is bounded and the exponential is not. Based on the shape of the exponential distribution, the most likely value is 0.0; there is also a high probability of very small times between arrivals. Many short inter-arrival times together can overwhelm a system. This could be the reason that the sixth replication has a considerably higher mean time in the system than the others, as seem in Figure 11.3. In this case, the average time in the system for the sixth replication is about 210 minutes, compared to between 30 and 80 minutes for the other nine replications. The unboundedness in the positive direction for the exponential distribution can generate some very long times between arrivals, which can skew the results in the other direction, causing a very low average time in the system. That is why it was generally suggested in Chapter 8 to use bounded distributions.

In addition to the change in the inter-arrival time distribution, the modelers incorporated Common Random Numbers (CRN), a variance reduction technique that can increase precision, into the model. Variance reduction is important because it enables a higher precision estimate of the performance measures for the same number of replications, or fewer replications for the same level of precision. Since CRNs are more important for comparisons, they are discussed in a later section of this chapter.

Pilot run #2

A second pilot run is made with the revised model (new inter-arrival time distribution and the use of CRN). As before, ten replications, each of 4800 minutes, are made using *FlexSim's* Experimenter. This time the point estimate for the time in the system is 29.6 minutes (compared to 69.6 minutes in the first pilot run). Similarly, the interval estimate indicates 90% confidence that the true average time in the system is between 26.4 and 33.2 minutes (compared to between 39.4 and 99.9 minutes). Based on the second pilot run

results, if the number of replications is at least 33, using the equation for n from Chapter 10, then there is 95% confidence that the estimate is within 2 minutes of the true mean. Note, this results in fewer replications and more precision!

To be a bit conservative, 35 replications are used in the subsequent analyses; however, it would not hurt to use 40 or 50 replications since we do not have an estimate for the effects that changes to the decision variables—different buffer allocations, queue discipline, and transport speeds—will have on precision.

Section 11-2 Comparison of two alternatives

This section describes how to statistically test to see if there is significant difference in performance between two alternatives. It also discusses the difference between statistical significant differences and practical significant differences.

Comparisons are typically used to improve the design and operation of a system, as is being done in the case study. While the case study defined in Section 11-1 involves many more alternatives to compare than just two, we start with a comparison of two in this section and then extend the number of alternatives in later sections.

The comparison is between the two ways to select items to inspect:

1. the order in which they arrive (FIFO)
2. choosing the product with the smallest inspection time first (SPT).

The transporter can accommodate either option since it can pick a product from anywhere in the buffer and transport it to the inspection machine.

Statistical comparison between two alternatives

Before comparing output from two simulation runs, it is helpful to relate common simulation terminology to basic Design of Experiments (DoE) terminology. This section provides only a brief introduction to DoE and simulation; for more information, see (for example), Winter Simulation Conference tutorials by Barton [4] and Sanchez [13].

In simulation experiments one or more input variables, usually the decision variables, are varied, to reveal the effect of the changes on the performance of the system. The performance measures are the output of the simulation. In DoE terms, the input variables are referred to as factors, and the performance measures, or output variables, are referred to as response variables. The value of a factor is typically called a level. For example, in a simple queuing system the experiment might be designed to evaluate the effect of the number of servers on customer wait time. In this case there is just one factor or input variable—the number of servers—at, say, three levels (4, 5, and 6 servers). There is also one output or response variable: average customer wait time.

In general, to formally compare two alternatives, start by performing a statistical test of the alternatives' sample means. The following hypotheses are tested: the null hypothesis is

that the population means are equal (i.e., the two samples came from the same population). Another way to say this is that the hypothesized difference in population means is zero. As shown below, the test can be made for a hypothesized difference other than 0 (d_0).

$$H_0 : \mu_1 = \mu_2 \rightarrow \mu_1 - \mu_2 = 0\,(d_0)$$
$$H_1 : \mu_1 \neq \mu_2 \rightarrow \mu_1 - \mu_2 \neq 0\,(d_0)$$

If H_0 (the null hypothesis) is rejected, then it is concluded that there is a significant difference in the mean of the two populations that represent the two alternatives. Hypothesis testing is a common topic in all basic statistics courses and therefore is not discussed in detail in this book; it is simply applied to simulation analysis.

Table 11.1 summarizes the assumptions of three commonly used approaches for statistical comparison. Again, these should be familiar from basic statistics. The two-sample t-test is almost always covered in introductory statistics courses, whereas Welch's method is oftentimes covered in such courses; similarly, the paired-t test is often covered in statistics courses, although not always.

Characteristic	**Two-sample t**	**Welch's**	**Paired t**
Observations within population	Independently and normally distributed	Independently and normally distributed	Independently and normally distributed
Observations between populations	Independent	Independent	Do not need to be independent
Population variances	Equal	Do not need to be equal	Do not need to be equal
Number of observations in each population	Do not need to be equal	Do not need to be equal	Equal

Table 11.1 Comparison of common means for statistical comparison

As shown in the table, the paired-t test does not require the observations between populations to be independent. This is an important difference and a key reason why the paired-t test is quite often used in simulation modeling and analysis. As was mentioned earlier, and as will be discussed in more detail later in this chapter, the use of common random numbers (CRN) in a simulation model oftentimes greatly reduces the variation in comparisons. It does so by introducing correlation, or dependency, between the populations. Therefore, when using common random numbers, the two-sample t and Welch's approach are not appropriate since they assume independence between populations or alternatives.

One general drawback to the paired-t approach is that it requires the same sample size for all of the alternatives. When performing physical or human-based experiments, this is

often a key restriction; for example, some subjects in an experiment may withdraw or be tainted. However, in simulation modeling and analysis, equal sample sizes are typically not a problem—it is easy to make the number of replications the same for all alternatives.

In general, the paired-t test, which tests the mean of the differences, uses the following hypotheses:

$$H_0 : \mu_{1-2} = 0\,(d_0)$$
$$H_1 : \mu_{1-2} \neq 0\,(d_0)$$

Hypothesis test can be performed by using confidence intervals. If the following confidence interval includes the hypothesized value, 0 in this case, then the null hypothesis cannot be rejected, and it is inferred, based on the sample results, that there is no significant difference between the two alternatives:

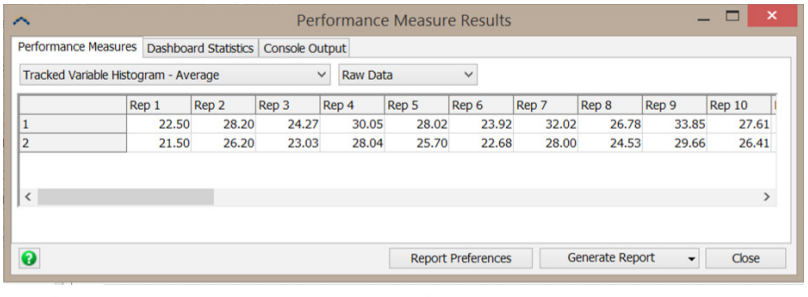
$$\bar{d}_{1-2} \pm t_{\frac{\alpha}{2},n-1}\frac{S_{1-2}}{\sqrt{n}}$$

For each of the *n* replications, the difference in the performance measure of interest is calculated. The mean and standard deviation of those *n* differences are $d_{1\text{-}2}$ and $S_{1\text{-}2}$, respectively.

The Good Vibration's work cell design case study is used to illustrate the application of the approach. The test shows whether there is any significant difference in system performance, as measured by the average time a product spends in the work cell, by using one of the two processing rules at the inspection station—FIFO or SPT.

Figure 11.5 shows the results from n=10 replications of each of the simulation models using FIFO and SPT queue disciplines. The example uses only ten replications for the sake of brevity; in general, at least 35 replications should be used, as determined in the previous section.

In the first replication, using the FIFO rule to process products at the inspection machines results in products spending one more minute in the system



Rep	FIFO	SPT	diff
1	22.5	21.5	1.0
2	28.2	26.2	2.0
3	24.3	23.0	1.2
4	30.1	28.0	2.0
5	28.0	25.7	2.3
6	23.9	22.7	1.2
7	32.0	28.0	4.0
8	26.8	24.5	2.3
9	33.9	29.7	4.2
10	27.6	26.4	1.2
Mean	27.7	25.6	2.1
Std Dev	3.6	2.6	1.1
t-value			2.262
LCIL			1.3
UCIL			3.0

$\bar{d}_{1\text{-}2}$

$S_{1\text{-}2}$

$\bar{d}_{1\text{-}2} \pm t_{\alpha/2,n\text{-}1}\frac{S_{1\text{-}2}}{\sqrt{n}}$

Figure 11.5: Comparison of FIFO versus SPT for inspection (n=10).

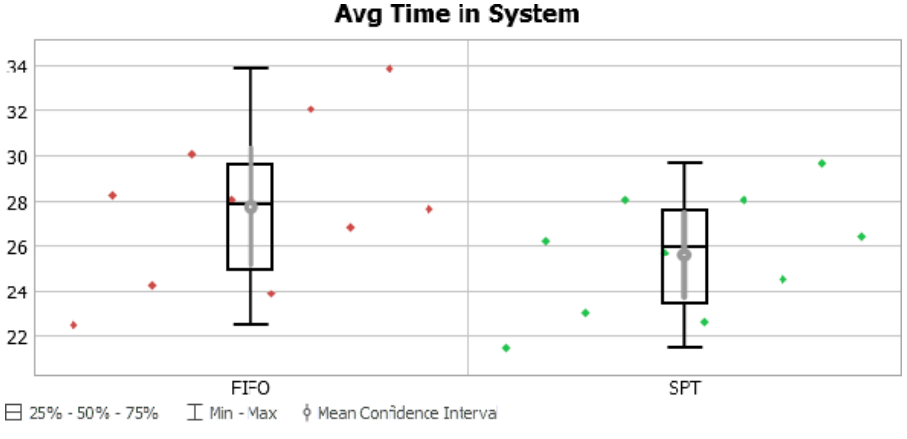
(22.5 - 21.5), on the average. Based on ten replications, the mean and standard deviation of the differences, $d_{1\text{-}2}$ and $S_{1\text{-}2}$, are 2.1 and 1.1, respectively. This means that based on the simulation, we expect products to spend 2.1 minutes more in the inspection system using FIFO than with using SPT. Also, we are 95% confident that the difference in time in the system is between 1.3 and 3.0 minutes. Since the confidence interval does not contain the hypothesized value 0.0, we conclude there is a significant difference in performance between using FIFO and SPT. Since FIFO results in longer times in the system, SPT is preferred.

Note that if independence is assumed, which would be incorrect since CRN is used in the model, a different conclusion would be reached. Assuming independence and unknown population variances, the two-sample t test is used, applying the following formula found in all basic statistics books:

$$\bar{Y}_1 - \bar{Y}_2 \pm t_{\frac{\alpha}{2},n*}\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}$$
$$n* = n - 1 \text{ if } n = n_1 = n_2$$

As a result, the confidence interval limits are 2.1±3.2, -1.1 and 5.3. Since the interval does not contain the hypothesized value of 0, the conclusion is that there is no statistically significant difference in the two processing rules. Also, notice that the interval estimate is less precise (i.e., the interval is wider than the one obtained using the paired-t test approach.

A similar conclusion would be reached if just the individual confidence intervals are compared. In Figure 11.6 the individual confidence intervals for FIFO and SPT overlap; therefore, we would conclude there is no significant statistical difference between the two processing rules. Of course, this conclusion might change if a larger sample size (more replications) is used.



Avg Time in System

	Mean (95% Confidence)				Sample Std Dev	Min	Max
FIFO	25.2	<	27.7	<	30.3	3.6	22.5	33.9
SPT	23.7	<	25.6	<	27.5	2.6	21.5	29.7

Figure 11.6: Comparison of FIFO versus SPT assuming independence.

Statistical versus practical differences between alternatives

As Banks, et al. [3] point out, there is an important distinction between statistically significant and practically significant results.

Results are statistically significant if the observed difference is likely due to a change in the system rather than occurring by chance as a result of random sampling. Statistical significance depends on the experiment—the amount of variability in the data, the sample size (number of replications), etc. In the example in Figure 11.6, the conclusion from the statistical analysis is that processing items according to the SPT rule is expected to result in shorter times in the system compared to using the FIFO processing rule.

Results are practically significant if the difference is large enough to effect the decision to choose an alternative; that is, would the observed difference in performance matter in the real system? Practical significance is independent of the simulation experiment and is specific to the problem and situation. In the example in 11.6, the decision as to whether 2.1 minutes, the expected difference in time in the system between the SPT and FIFO rules, is practically significant would depend on the situation (i.e., whether that difference would matter in the real system). Typically, the difference is used in an economic analysis that would explicitly address whether the expected difference in costs of an alternative is offset by the expected difference in benefits.

Section 11-3 Reducing variability in the samples using Common Random Numbers (CRN)

Before assessing the effect of using Common Random Numbers (CRN) on the experiments in the case study, consider a simple single-server system with random times between arrivals and service times and using a FIFO queue discipline. Items are inspected by the server and are sent either (1) out of the system, if okay, or (2) back to the queue to be reworked, if not okay. The system processes two types of parts, but it is assumed that the type of item does not affect process performance (same process times, same quality level). Partial results from four simulations of the system are provided in Figure 11.7.

Initially, only one type of item (Type = 1) is considered, even though the system actually processes two items. As stated above, it is assumed the item type does not affect performance; however, reworked items are designated Type = 10. Results for the first ten customers in the simulation are shown at the top of the first column in Figure 11.7. In this case, a single random number stream is used; that is, all sources of variability in the model (interarrival times, service times, and quality) use the same stream. Random number streams were discussed in Chapter 8 and are discussed further below.

In the second simulation of ten customers in the first column of Figure 11.7, an additional source of variability is added: the type of item being processed. Items are designated either type 3 or type 4 based on a sample from a probability distribution (in this case, uniform). Type 10 again identifies a reworked item. Note that the type is not used in any of the model's operating logic; the change is only made for aesthetics. (The change could also be used to gather statistics on the two types of items.) The model still uses a single random

Using a single RN stream

Cust Num	IAT	Type	Serv Time	Time in System	Arr Time	ReWork
1	12.3	1	12.16	12.16	12.3	0
2	8.3	1	13.91	17.77	20.6	0
3	11.01	1	1.48	8.24	31.61	0
4	1.08	1	12.78	19.94	32.7	0
5	0.05	1	3.98	23.87	32.75	0
6	3.53	1	0.76	21.09	36.28	0
7	5.39	1	9.98	25.69	41.67	0
8	0.54	1	3.27	28.41	42.21	0
9	14.27	10	10.88	30.02	56.48	1
10	39.64	1	14.43	14.43	96.12	0

Cust Num	IAT	Type	Serv Time	Time in System	Arr Time	ReWork
1	12.3	3	9.91	9.91	12.3	0
2	10.85	4	0.04	0.04	23.16	0
3	15.45	4	1.48	1.48	38.61	0
4	5.39	3	12.85	12.85	44	0
5	14.2	10	13.58	60.82	58.2	1
6	4.42	4	0.4	9.56	62.62	0
7	5.52	3	14.43	18.47	68.13	0
8	0.95	10	27.41	105.06	69.09	1
9	25.34	3	11.27	35.87	94.43	0
10	0.79	4	25.48	60.56	95.22	0

using Common Random Numbers

Cust Num	IAT	Type	Serv Time	Time in System	Arr Time	ReWork
1	12.3	10	7.78	12.78	12.3	1
2	8.3	1	3.38	7.86	20.6	0
3	10.85	1	9.55	9.55	31.46	0
4	13.51	1	17.72	17.72	44.97	0
5	11.01	1	4.8	11.52	55.98	0
6	6.3	1	0.18	5.4	62.28	0
7	15.45	1	0.81	0.81	77.73	0
8	1.08	10	11.71	105.2	78.82	1
9	0.05	1	18.73	30.39	78.87	0
10	3.53	1	32.9	59.76	82.4	0

Cust Num	IAT	Type	Serv Time	Time in System	Arr Time	ReWork
1	12.3	10	7.78	12.78	12.3	1
2	8.3	3	3.38	7.86	20.6	0
3	10.85	4	9.55	9.55	31.46	0
4	13.51	3	17.72	17.72	44.97	0
5	11.01	4	4.8	11.52	55.98	0
6	6.3	4	0.18	5.4	62.28	0
7	15.45	4	0.81	0.81	77.73	0
8	1.08	10	11.71	105.2	78.82	1
9	0.05	4	18.73	30.39	78.87	0
10	3.53	3	32.9	59.76	82.4	0

Figure 11.7: Comparison of using a single random number stream and Common Random Numbers.

number stream. Notice that when this random variable (that does not impact the model operation logic of performance) is added, all of the values change between the first and second simulations in column one of Figure 11.7. It should appear odd that just adding this feature completely changes the inter-arrival times, service times, and quality of the item.

Adding this aesthetic variable should not change the simulation and its results, but it does. This is because the model in both cases is sampling from the same stream of random numbers. The new variable (type of item) is sampled among the other variables. This changes which random value is assigned to which customer and to which variable. It also adds unnecessary variability into the model. However, this problem is easily remedied.

CRN is now applied to the simple model, and the results are provided in the second column in Figure 11.7. Notice that all values between the two simulations in column two are the same when the new random component (type of item) is added to the model. This is what we would expect!

To control this variability, all that needs to be done is to be sure each source of variability in a simulation model has its own random number stream. This is easy to do in *FlexSim*: just assign a different stream number to each probability distribution specified in a model. The stream number is specified as the last parameter in all statistical distributions in *FlexSim* (i.e., normal(Mean, StdDev, StreamNum), uniform(Min, Max, StreamNum, etc). For example, the *FlexSim* command normal(10, 2, 8) instructs the software to sample from a normal distribution with a mean of 10 and a standard deviation of 2 using random number stream 8. Since each stream is independent of the other streams, it does not matter which stream number value is assigned to a source of variability as long as the assignments are unique. The stream value can be any integer from 0 to 99. By default, *FlexSim* assigns the stream value in each distribution to be 0. As discussed in Chapter 8, by default *FlexSim* provides 100 steam numbers (0 to 99); however, more can be used if needed. To use more than 100 streams requires a small amount of coding; thus, it is considered an advanced topic for this book.

The approach described above is one way to implement the variance-reduction technique called Common Random Numbers (CRN). CRN and other techniques reduce the variability in comparisons among models. They reduce the number of replications required to attain a certain level of precision, or they result in a more precise estimate (narrower confidence interval) for the same number of replications. By using CRN, the values between replications are no longer independent, and thus the paired-t test becomes the appropriate parametric statistical test. There are other types of variance reduction techniques that can be used in simulation modeling and analysis; however, they are beyond the scope of this text. For more information, see, for example, Law [10].

CRN is also referred to as correlated as sampling because introducing correlation may reduce the overall variance. As shown below, in general the variance of a difference depends on the covariance (cov) or correlation (ρ) between the samples. If the populations or samples are independent, then the variance of the difference is the sum of the population or sample variances. (Independence means cov = 0 and $\rho = 0$.)

$$\sigma^2\left(\bar{Y}_1 - \bar{Y}_2\right) = \sigma_1^2 + \sigma_2^2 - 2\,\mathrm{cov}\left(\bar{Y}_1, \bar{Y}_2\right) = \sigma_1^2 + \sigma_2^2 - 2\rho_{12}\sigma_1\sigma_2$$

CRN usually results in a positive correlation so that the variance is reduced. This is based on the notion that both alternatives respond in the same manner to the variable, though this is not true in all cases. As Robinson [12] suggests, use CRN and perform the following test:

$$\sigma^2\left(\bar{Y}_1 - \bar{Y}_2\right) \overset{?}{<} \sigma_1^2 + \sigma_2^2$$

If the paired-t variance is less than the sum of the variances of each alternative, then use the paired-t results (confidence interval); if not, use the confidence interval based on the standard two-sample t test. In the example above, in Figures 10.6 and 10.7, the use of CRN is effective since

$$(1.1)^2 < (3.6)^2 + (2.6)^2 \rightarrow \sigma^2\left(\bar{Y}_1 - \bar{Y}_2\right) < \sigma_1^2 + \sigma_2^2$$

Therefore, use the paired-t confidence interval.

Having defined and described the use of CRN in simulation models, we now show the effect of their use on the Good Vibrations comparison that was made at the end of the last section. The comparison in Figure 11.6, between the use of FIFO and SPT queue disciplines, led to the conclusion that using SPT is expected to result in significantly lower times in the system (cell). In this case, there is 95% confidence that the difference is between 1.3 and 3.0 minutes. With all other parameters the same, Figure 11.8 shows the results if CRN had not be used in the model. The conclusion without CRN is that there is 95% confidence that the difference is between -3.6 and 9.6 minutes. This is a much wider interval (13.2 versus 2.7); that is, it is much less precise. It also results in a different conclusion: since the confidence interval contains the hypothesized value (0), the conclusion is that there is no significant difference in average time in the system between using SPT and FIFO. This conclusion would be counterintuitive because from production theory, for single-machine scheduling, using SPT minimizes average flow time.

Rep	FIFO	SPT	diff
1	26.45	29.21	-2.8
2	29.73	27.8	1.9
3	21.54	23.38	-1.8
4	25.42	27.92	-2.5
5	30.03	28.97	1.1
6	26.04	25.2	0.8
7	26.15	23.95	2.2
8	55.32	26.91	28.4
9	26.18	27.6	-1.4
10	27.28	23.5	3.8
Mean	29.4	26.4	3.0
Std Dev	9.4	2.2	9.2
t-value			2.262
LCIL			-3.6
UCIL			9.6

Figure 11.8: Comparison of FIFO versus SPT for inspection *without* Common Random Numbers.

A better way to synchronize the random numbers in this example would be to assign an item's process time(s) when the item is created (at the source), rather than at the processor

where the time is used. This would guarantee the process time is the same for each item regardless of the order in which the item is processed. This can easily be accomplished by assigning a label value that contains a randomly generated process time to each item at creation. The label value is then read at the processor and used as the process time.

Section 11-4 Comparison of multiple alternatives using Bonferroni's method

There are several approaches for comparing multiple (more than two) alternatives; however, we only present the most basic and common methods here. For a more detailed discussion of multiple comparison procedures see, for example, Law [10], Hsu [6], and many basic engineering statistics books. For an online source, see the *Engineering Statistics Handbook* created by the National Institute of Standards (NIST), an agency of the U.S. Department of Commerce [11].

In constructing confidence intervals and performing hypothesis tests, a level of significance α is set. From basic statistics, this is the probability of committing Type I error or incorrectly rejecting a true null hypothesis. In the case of a comparison, the null hypothesis is that there is no significant difference in the system performance between the compared alternatives.

The Bonferroni approach adjusts the level of significance to account for the multiple, and not single, comparisons. For example, if two 95% confidence intervals are constructed, then the overall confidence will be at least 90%; similarly, if ten 95% confidence intervals are constructed, then the overall confidence will be at least 50%. Thus, to have the overall confidence level to be 95% for ten comparisons, each comparison needs to be made at a 99.5% confidence level.

A downside to the Bonferroni approach is that the comparisons are less precise since α affects the width of the confidence interval. For example, for a 95% confidence interval, $z_{0.025} = 1.96$; whereas, for a 99.5% confidence interval, $z_{0.0025} = 2.807$. Thus, all other things being the same, the interval will be over 40% wider when adjusting α to account for multiple comparisons. More replications may be needed when the plan is to make multiple comparisons rather than estimating a single parameter or making a single comparison.

The following describes how to obtain the level of α for each individual comparison under two situations: (1) comparisons to a standard or base case and (2) all pairwise comparisons.

K alternatives can be compared in a pair-wise manner to a standard or base case. In this situation, there are *k-1* confidence intervals for *k-1* differences; for example, $\mu_1 - \mu_2$, $\mu_1 - \mu_3$, etc., where μ_1 is the designated standard. If the overall desired level of confidence is 1- α then each individual interval should be set so that its confidence level is

$$1 - \frac{\alpha}{(k-1)}$$

Therefore, each interval's level of significance should be

$$\frac{\alpha}{(k-1)}$$

For example, if the overall desired confidence is 0.95 (1 – 0.05), and if a base case is compared to each of the four alternatives (a total of five systems), then α for the four pairwise comparisons should be 0.05/4 ≈ 0.01.

K alternatives can also be compared in pair-wise manner with every other alternative. In this case, there are k(k-1)/2 confidence intervals. If the overall desired level of confidence is 1-α, then each individual interval should be set so that its confidence level is

$$1 - \frac{\alpha}{\frac{k(k-1)}{2}}$$

Therefore, each interval's level of significance should be

$$\frac{2\alpha}{k\,(k-1)}$$

For example, if the overall desired confidence is 0.95 (1 – 0.05) and if each of four alternatives is compared to another alternative, then there are six pairwise comparisons; therefore, α for each comparison should be 0.05/6 ≈ 0.01.

Applying multiple comparisons to the case study

In the cell design case study, the following two variables (factors) and their levels are considered:

- Buffer size: three levels (8/2, 5/5, 2/8); that is, for the first level, the buffer before the first machine is 2 and the buffer before the second machine is 8; for the second level, the buffer before the first machine is 5 and the buffer before the second machine is 5, etc.
- Queue discipline: two levels (FIFO and SPT)

These settings result in the six alternatives identified in the table on the left side of Figure 11.9. The resulting 15 pairwise comparisons of the six alternatives are represented by the highlighted cells on the right side of Figure 11.9. Recall that the comparison performed in Section 11-2 was between FIFO and SPT when the buffer sizes in front of each machine were both five. This corresponds to alternatives two and five in Figure 11.9; their comparison is noted by the * in the pairwise comparison table.

Obviously, the number of comparisons grows rapidly with the number of alternatives. If four options for buffer size are considered, instead of three, with the same two queue discipline options, then there are eight alternatives and 28 comparisons. Ten alternatives result in 45 comparisons, etc.

		Buffer capacities, 1/2		
		2/8	5/5	8/2
Queue discipline	FIFO	1	2	3
	SPT	4	5	6

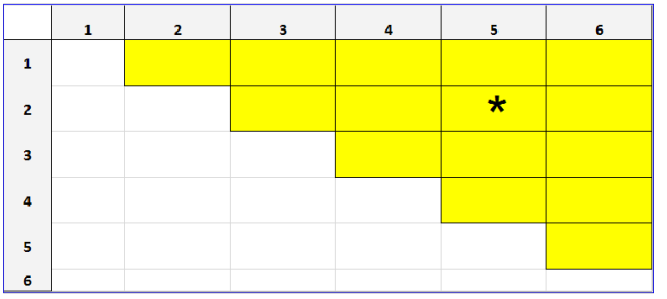


Figure 11.9: Alternatives and comparisons for the Good Vibrations example.

The simulated results for the six alternatives and 15 comparisons are shown in Figure 11.10. Each alternative is a scenario that is replicated 35 times, with each replication being 4800 minutes. The mean times in the system for each alternative are shown in the top part of the figure, and the pairwise comparisons of those alternatives are shown in the bottom part of the figure. For the pairwise comparisons, if the overall desired α is 0.05, then each comparison is made with $\alpha = 0.05/15 \approx 0.003$.

Means	2 / 8	5 / 5	8 / 2
FIFO	16.6	16.6	16.7
SPT	16.2	16.2	16.3

Cofidence Intervals	FIFO 2/8	FIFO 5/5	FIFO 8/2	SPT 2/8	SPT 5/5	SPT 8/2
FIFO 2/8		0.0 +/- 0.0	-0.1 +/- 0.1	0.4 +/- 0.1	0.4 +/- 0.1	0.3 +/- 0.1
FIFO 5/5			-0.1 +/- 0.1	0.3 +/- 0.1	0.4 +/- 0.1	0.3 +/- 0.1
FIFO 8/2				0.4 +/- 0.1	0.5 +/- 0.1	0.4 +/- 0.1
SPT 2/8					0.0 +/- 0.0	-0.1 +/- 0.1
SPT 5/5						-0.1 +/- 0.1
SPT 8/2						

Figure 11.10: Comparison of simulation results from the Good Vibrations example.

Note that there is not much difference in performance among all the alternatives. The best case appears to be the one using SPT with buffer sizes of five and five; however, there is not much difference between the SPT cases. This is confirmed with the pairwise comparisons—the shaded areas are those that have a statistically significant difference. All the SPT cases are significantly better than FIFO, but there is no difference between the FIFO cases themselves or the SPT cases. This confirms the findings in Figure 11.6 in Section 11-2 that SPT results in lower average time in the system.

However, from a practical perspective there really is not a significant difference in performance. This seems a puzzling, so the results are explored a bit deeper. The average time in

the system is not that high—about 16 minutes—especially when the expected process time is 10 minutes. (The inspection time at each machine is 2, 4, 6, or 8 minutes; thus the average is 5 minutes for each of two inspections in series.) It seems that the resources (inspection machines and transporter) are not heavily utilized. A closer look at the simulation verifies this—the inspection machines are busy about 60% of the time; set up time is about 5%, downtime about 7%; therefore, they are each idle about 28%. The transporter is idle nearly 75% of the time. The resources are therefore not much of a bottleneck so any improvement will not result in much performance improvement.

This analysis uncovered another issue: the engineering staff had decided to have the machines perform more inspections on the products, but did not inform the modelers. As a result of more inspections, the processing times will increase. The revised inspection times are 3, 4, 8, and 9 minutes for products A, B, C, and D, respectively. The simulation model is rerun with the new times and using the same experimental design as before, except each scenario is replicated 50 times to be more conservative since the calculation for n=35 is based on a single comparison. The results are shown in Figure 11.11.

Means	2 / 8	5 / 5	8 / 2
FIFO	107.7	116.6	170.3
SPT	98.8	93.2	131.5

Cofidence Intervals	FIFO 2/8	FIFO 5/5	FIFO 8/2	SPT 2/8	SPT 5/5	SPT 8/2
FIFO 2/8		-8.9 +/- 5.5	-62.6 +/- 14.1	8.9 +/- 6.7	14.6 +/- 5.3	-23.7 +/- 11.9
FIFO 5/5			-53.7 +/- 12.1	17.8 +/- 8.8	23.4 +/- 7.3	-14.9 +/- 9.9
FIFO 8/2				71.5 +/- 15.7	77.2 +/- 13.9	38.9 +/- 6.2
SPT 2/8					5.7 +/- 6.6	-32.6 +/- 13.5
SPT 5/5						-38.3 +/- 11.4
SPT 8/2						

Figure 11.11: Revised comparison of simulation results from the Good Vibrations example.

Based on the mean time in the system, (i.e. on the average, a product spends 93.2 minutes in the inspection area) the best option appears to be to use the SPT processing rule and equal buffer sizes in front of each inspection station. In terms of pairwise comparisons among the alternatives, all pairwise comparisons are significant except one (not highlighted). This implies there is not a statistically significant difference between using SPT with equal buffer sizes or with a buffer of two prior to machine one and eight prior to machine two. That is, based on the experiments it is not clear if the observed difference (98.8 versus 93.2 minutes) is due to system characteristics or sampling error. Additional replications could be made focusing on these two options in order to better understand the effect of these buffer sizes on system performance.

Section 11-5 Introduction to Optimization

Optimization basically means finding the best solution (i.e. the best values for the decision variables). What is best depends on the objective(s) and on the system's constraints or restrictions—no problem has unconstrained resources. The objective(s) and constraints are specified as mathematical relationships expressed in terms of the decision variables. The objective function is either maximized (e.g., throughput) or minimized (e.g., total cost). In general, any optimization problem is specified as

maximize/minimize Objective (mathematical relationship in terms of decision variables)

s.t. (subject to)

Constraints (mathematical relationship in terms of decision variables)

The objective is a mathematical function of one or more performance measures; thus, a problem may have multiple objectives. It determines whether one solution is better than another.

Constraints are mathematical expressions that must be enforced in the model (e.g., the number of operators (NumOps) must be between 1 and 5). A constraint is a mathematical function like an objective, but contains a comparison (e.g., NumOps ≥ 1, NumOps ≤ 5). Solutions are only considered where the evaluation of all constraint comparisons is true; any possible solution that violates a constraint is considered infeasible.

Optimization is a separate operations research technique from simulation; it involves a very different approach and thus a very different solution methodology. Simulation is used to describe how a system works and how it performs, whereas optimization is used to find the best, or better, solution based on a specified definition of a system and its operation. There are numerous optimization methodologies; many, such as linear programming, assume that the problem is deterministic (i.e., all of the values in the problem definition are known constants, not variables). Simulations contain one or many stochastic or random variables; therefore, the simulation-optimization problem is not deterministic. As a result, optimizing systems using simulation is especially difficult and challenging. Due to the stochastic nature of simulation, there is risk and uncertainty in optimization results, and thus, strictly speaking, optimal results are not guaranteed to be optimal. Also, simulation problems oftentimes involve many possible solution alternatives (i.e., have large solution spaces); we use the term optimal to mean the estimated best solution that was found during a search.

Simulation optimization generally involves an iterative approach as shown in Figure 11.12. This means that a simulation model generates responses (output) that are based on a set of input values. The optimizer typically uses heuristics to decide how to change the value of the input parameters to get an improvement in the value of the response variables. A heuristic is a "short-

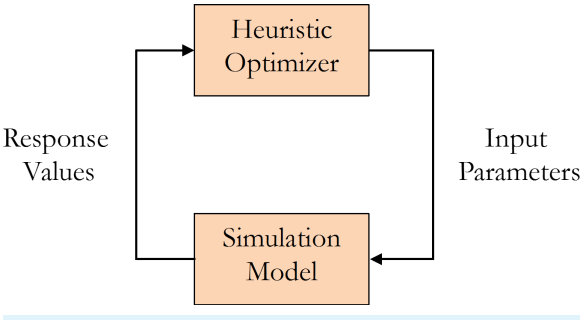


Figure 11.12: Input to the optimizer for the Good Vibrations example.

cut" method that, while typically fast, does not guarantee finding the best solution; however, good heuristics provide improved solutions that are hopefully close to the best solution.

There are many approaches to simulation optimization and many issues to address, but only the basics are presented in this book. For more information about simulation optimization approaches, see, among others: [1], [2], [5], [7], and [10]. As indicated by April, et al. [2], most commercial approaches use evolutionary approaches that search the solution space based on an evolving set of solutions. The set of solutions are used to create new trial solutions; thus, new solutions typically represent a combination of solutions that are in the current set. Common evolutionary approaches include scatter search and genetic algorithms. The main advantage of evolutionary approaches over sampling in the neighborhood of a single solution, according April, et al. [2], is that they can explore larger areas of the solution space with a smaller number of objective function evaluations (running a simulation model for multiple replications).

Simulation optimization in *FlexSim* using *OptQuest*

Simulation models developed in *FlexSim* can be optimized using the special-purpose optimization software *OptQuest*. *OptQuest* is a standalone commercial product produced by OptTek Solutions and is bundled with *FlexSim*. *OptQuest* is an industry-standard optimization engine that is seamlessly integrated with *FlexSim* through the Experimenter. Laguna [9] provides a description of *OptQuest's* approach to simulation optimization. The *OptQuest* interface in *FlexSim* is defined and illustrated in the examples in the subsequent subsections.

Variables for the optimization of a *FlexSim* simulation model are taken from Variables tab in the Experimenter. Variables are used to define the decision variables (those for which we desire to find the best values), and constraints (functions of the variables that place bounds on the possible solutions). For optimization, the variables need to be specified in terms of a variable type, such as *continuous* or *integer*. Continuous means the variable can take on any real value, whereas, integer is a whole number. Both types may be either positive or negative. There are special types of integers. *Discrete* variables are defined over a range of values, and there is a step, of size n, between values. *Binary* variables are limited to the values 0 or 1. *Design* variables are those that have a range, but a higher value does not mean more; they are typically used to denote options, such as product or machine type, policy or rule identifier, etc. *Permutation* variables are groups of number that have values between 1 and n, and each value appears once in the group. For example, the permutations of 1, 2, and 3 are: (1, 2, 3), (3, 2, 1), (2, 1, 3). An example of a permutation variable is the possible routings of a part between three work cells.

The objective function in *OptQuest* is formed from the Performance Measures defined in the Experimenter. Objectives may be single, weighted, or pattern (Pareto optimal). These types affect the way the objective function is used in the search for the optimal solution.

The optimization process involves running the simulation model for, in essence, one scenario—one set of values for the variables for the specified time and number of replications—and then adjusting the value of the variables based on the performance of the

objective function and running another scenario. The adjustment of the variables is based on a proprietary algorithm within *OptQuest*. The optimization engine uses the latest evolutionary algorithms to effectively search the design space.

The optimizer continues to search for an optimal solution until one of the following criteria is met: (1) all possible solutions have been considered, (2) the specified number of solutions have been considered, or (3) the specified time to search has elapsed (referred to as *wall time*). When the search ends, the optimizer reports the best solutions found. Unless all possible solutions can be evaluated, which rarely occurs, the best solutions are technically not optimal.

Optimization - Good Vibrations Example

The optimization of the Good Vibrations example that has been considered throughout this chapter has a very limited solution space, and thus all possible solutions can be carried out through the Experimenter that is, all possible scenarios could be simulated, and the one with the best performance would be considered the optimal solution. In this case, the total queue or buffer size is limited to 10 in the cell, and since buffer size is an integer value, there are only nine possible solutions, (B_1, B_2) = (1,9), (2,8), … (5,5), … (9,1). The example also considers two possible queue disciplines, FIFO and SPT; therefore, there are 18 possible solutions—each possible set of buffer sizes operating under each queue discipline. This is a two-factor experimental design with one factor at nine levels and the other at two levels; thus, in statistical terms, there are 18 treatment effects, or in simulation terms, 18 scenarios.

Despite the small solution space, we use this example to illustrate the application of the optimizer that is incorporated within *FlexSim*. There are three tabs in the Experimenter that pertain to the Optimizer—two for defining the problem and one for results. Figure 11.13 shows the two tabs for setting up the optimization run.

The interface at the top of Figure 11.13 provides the input for the optimizer design for the Good Vibrations example. The variables are those that are defined in the Scenarios tab. For the optimizer, the variable types are specified as Integer for the buffer sizes and Design for the queue discipline. The ranges of the variable values are set from 1 to 9 for the buffer sizes and 0 and 1 for the queue discipline. There is only a single constraint—the combined buffer size in front of both machines must be ten. The single objective is to minimize average time in the system.

The interface at the bottom of Figure 11.13 defines the optimization run. Each solution is replicated 50 times, with each replication being 4800 minutes with no warmup time. The specified maximum number of solutions to consider is 100 and the total maximum run time is 300 seconds (real time).

The results from the optimization run are shown in Figure 11.14. A total of 18 solutions were evaluated by the optimizer; it completely enumerated the solution space since these were all of the possible solutions. The plot shows the order in which the solutions were obtained. The best option is Solution 13. In this case the queue size in front of the first machine is three and in front of the second machine is seven and the SPT queue discipline

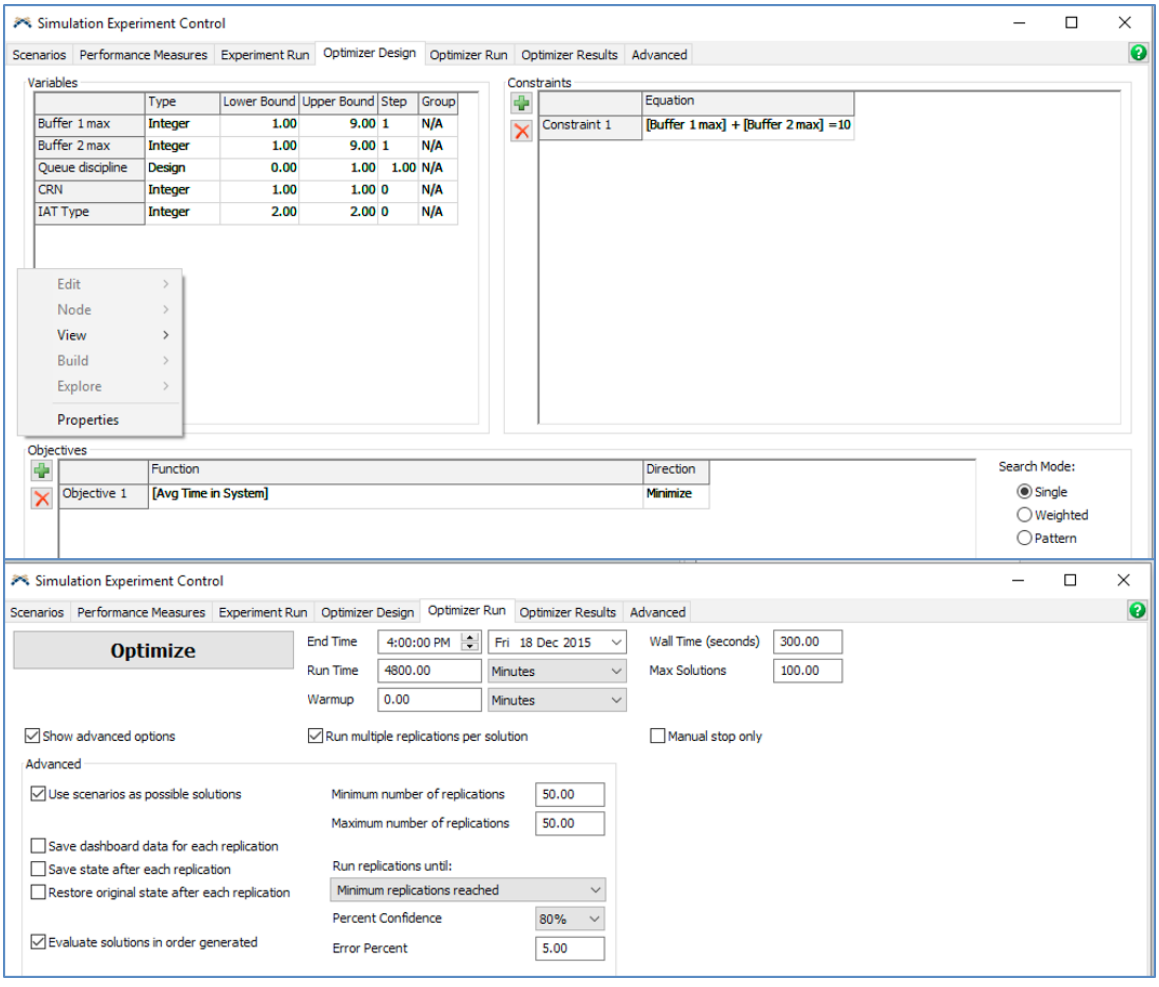


Figure 11.13: Input to the optimizer for the Good Vibrations example.

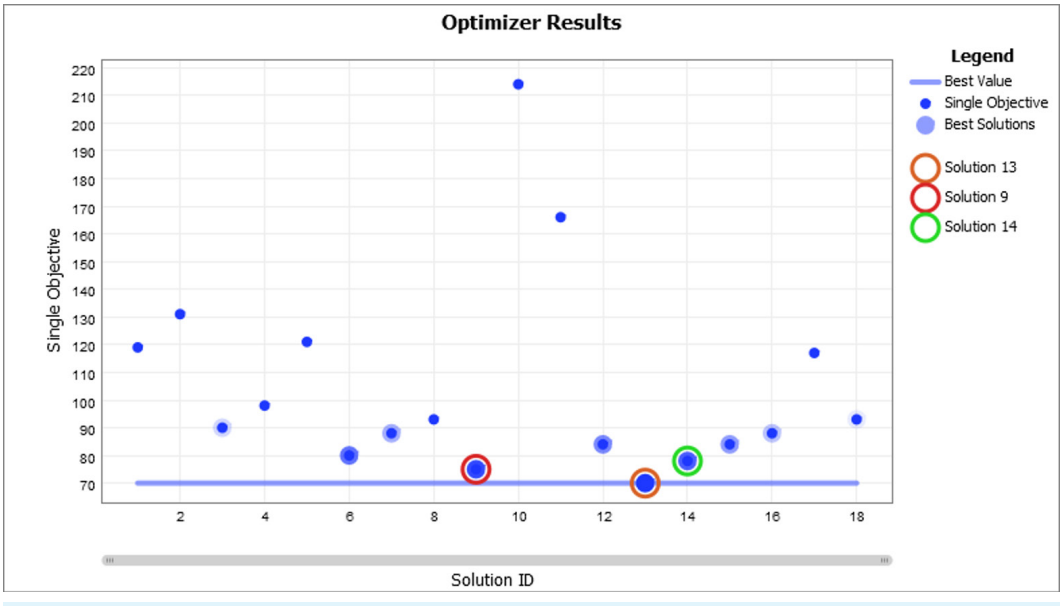


Figure 11.14: Output from the optimizer for the Good Vibrations example.

is used; this results in an average time in the system of 71 minutes. The third best solution is the same configuration but with a different queue discipline. In terms of performance, this solution is many times better than the worse case, Solution 10, with an average time in the system of approximately 215 minutes. In Solution 10 the queue size in front of the first machine is nine and in front of the second machine is one and the FIFO queue discipline is used. The next to worse case is the same configuration but with a different queue discipline.

The value of the variables for each solution is determined by highlighting the solutions in the Optimizer Results plot and pressing the Export Selected Scenarios button; this provides the solution values as additional columns in the Scenarios tab. In the case above Solutions 13, 9, and 14 are exported. Another option is Export as CSV, where information on all solutions is exported to a spreadsheet file.

Another view of the optimization results is shown in Figure 11.15 in terms of the single objective, average time in the system. This plot is generated from the exported spreadsheet file. The insert focuses on the best six solutions since the overall plot makes the differences seem negligible; however, the best solution is nearly 20% better than the sixth solution (71 minutes average time in the system versus 85 minutes).

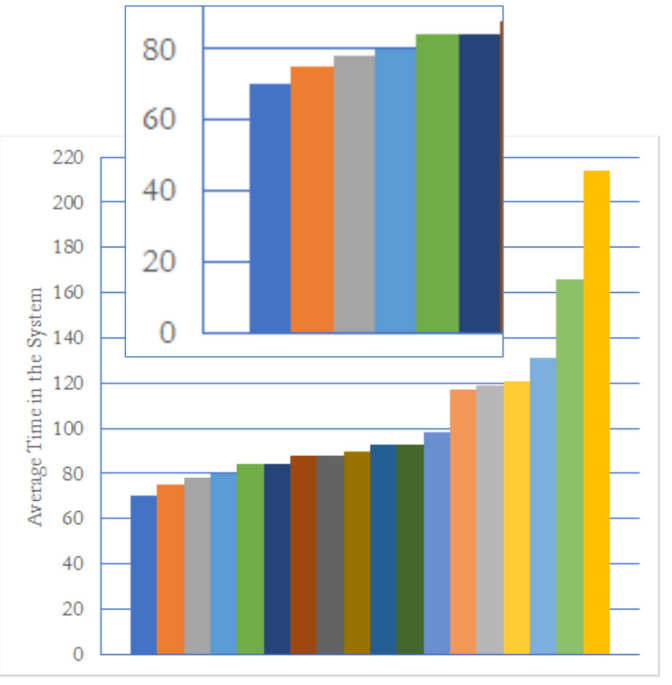


Figure 11.15: Output from the optimizer for the Good Vibrations example.

Of course, as with any analysis of simulation output, the results are random variables and subject to uncertainty. Thus, the optimization is not definitive, but is a guide for decision making. It appears from the simulation results that more buffer space should be allocated to the second machine, possibly because there is a large overflow buffer in front of the first machine's buffer. An item that spends time on the overflow buffer will have a longer time in system, but a full queue before the second machine will block the first machine and affect the time in the system for multiple items.

Optimization: Layout Example

The second optimization example is quite simple, but illustrates the use of the optimizer to evaluate alternative layouts.

The system is illustrated in Figure 11.16. Each grid unit represents one meter. Two machines that are two meters square are to be located within a 20-meter space. In this case,

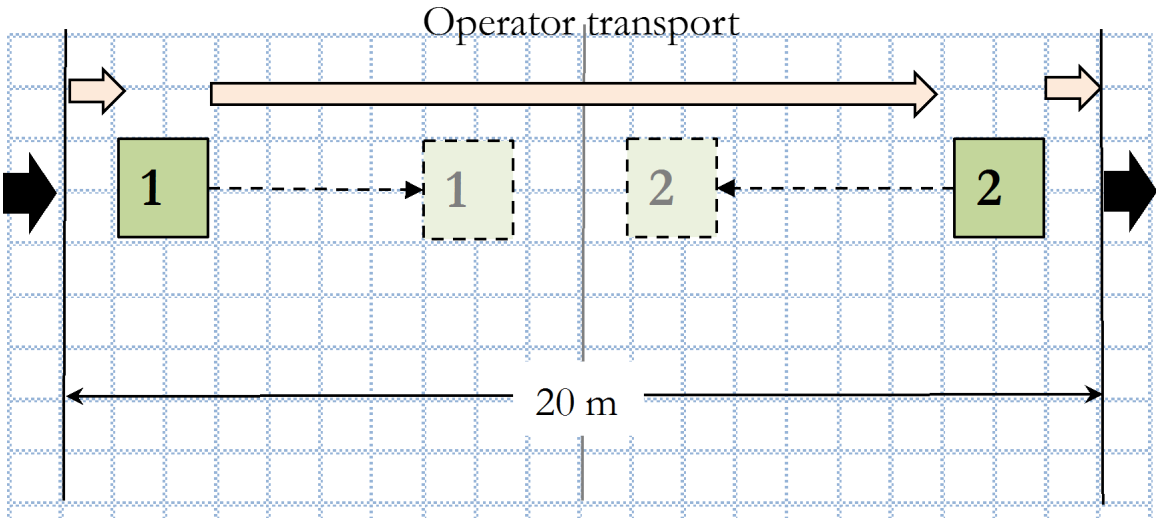


Figure 11.16: Definition of the layout example.

to make things simpler, only their horizontal location is considered. An operator moves material between the machines but is not needed for processing at either machine. The material handler carries items from an input source at the left boundary of the space to the first machine where it is processed; currently, that distance is one meter. Once processed, the operator moves the item to a second machine for processing; currently, that distance is 14 meters. Once processing is done at the second machine, the operator moves the item to an output conveyor located at the right boundary.

Each machine can be moved toward the other as illustrated by the dashed lines in Figure 11.16; that is, machine one can be moved six meters to the right and machine two can be moved six meters to the left. If both machines are moved their full extent, they will be two meters apart. The objective is to maximize throughput by adjusting the horizontal position of the two machines.

Processing at the first machine is normally distributed with a mean process time of 40 seconds and a standard deviation of 10 seconds. Process time at the second machine is normally distributed with a mean of 60 seconds and a standard deviation of 20 seconds. The operator travels at a speed of one meter per second. All resources are 100% available (i.e., no downtime).

The *FlexSim* model is shown in Figure 11.17. Assuming items are always available to be moved, then the source has a zero time between arrivals with no input queue. Even though the output is to a conveyor, a sink is used since the model scope is limited

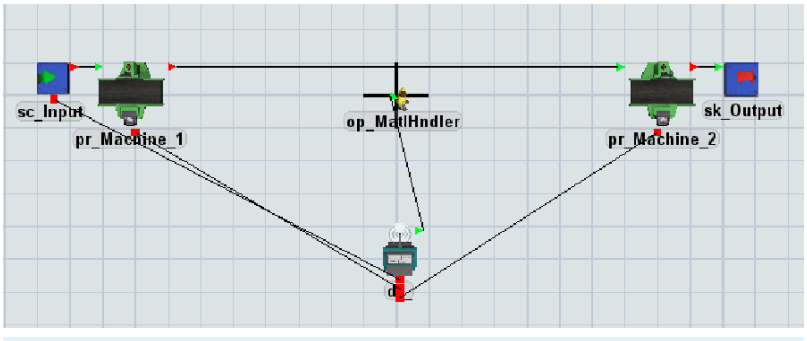


Figure 11.17: FlexSim model for the layout example.

to the machine area. Direct Spatials are used to set all object locations. For consistency between options, the operator returns to a home location after each replication. The On Reset trigger on the operator is used to set its location at reset to the origin, 0, 0, 0. Details for constructing a similar model can be found in *FlexSim's* User Manual (Experimenter/ Optimizer Example).

The Optimizer Design tab for the layout example is shown in Figure 11.18. The objective is to maximize throughput (input to the model's single sink. No constraints are needed since the problem is bounded through the specification of the variables—the range of variables for the location in the x direction for each machine. Each replication is run for 10,000 seconds. While this may sound like a lot, it isn't; the average total process time of an item, excluding the travel time by the operator is 100 seconds. Therefore, the performance estimated at each replication is only based on less than 100 items. At least 10 replications are considered for each scenario.

The results from 100 solution evaluations by the optimizer are shown in Figure 11.19. The top four solutions are all very close in value and correspond to where the machines would be placed as close together as possible. Note that the optimizer found that point on the fifth solution; of course, this is mainly coincidental—there is no way to know a priori how many iterations are needed to find the best, or near best, solution.

The optimal solution provides considerable improvement over other possible solutions, as illustrated in the graph in Figure 11.20. The plot is of average throughput through the simulation model for various locations of the machines. The left-most value is the best found solution; moving to the right are the average throughput for the other 99 solutions that were considered by the optimizer. Note that the throughput for the best solution is more than three times the worst case.

The best solution—locating the machines as close as possible—should be intuitive. Consider the movement of a hypothetical item, say item number 11, through the process. The operator moves item 11 from the source to Machine 1 and waits until it is processed. When item 11 is finished at Machine 1, item 10 is still processing at Machine 2. The operator waits until Machine 2 finishes, then walks empty to Machine 2 and moves item 10 to the sink. The operator then returns to Machine 1, again empty, to move item 11 to Machine 2. The operator then returns to the source to move item 12 to Machine 1. Since there are multiple trips between Machines 1 and 2, it makes sense to put them as close together as possible.

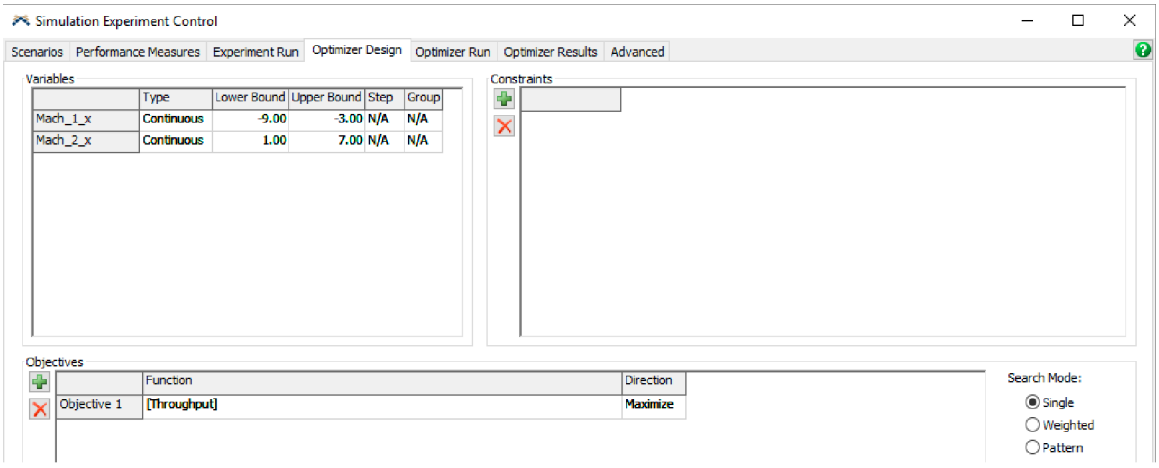


Figure 11.18: Optimizer setup for the layout example.

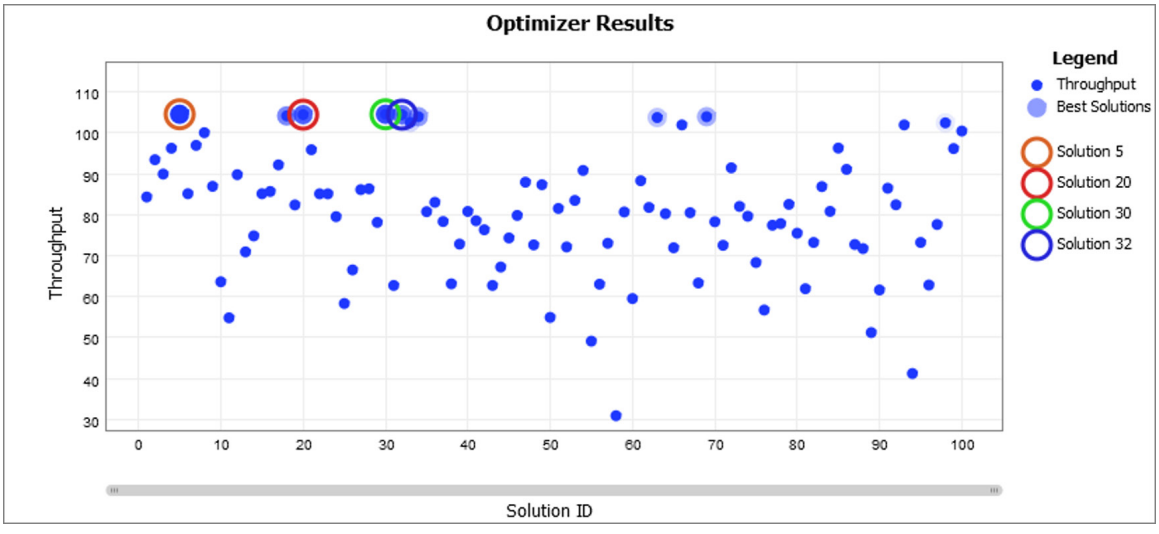


Figure 11.19: Optimizer output for the layout example.

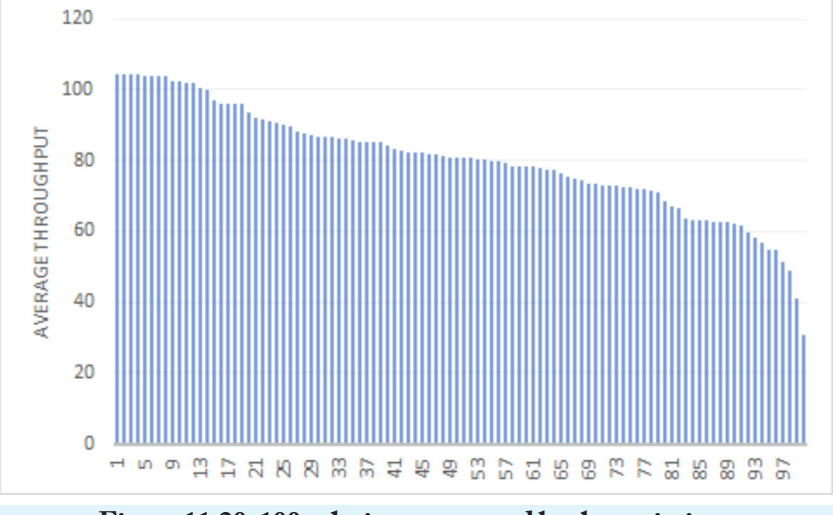


Figure 11.20: 100 solutions generated by the optimizer.

Optimization: Best process time example

The third optimization example is also quite simple, but illustrates a case that involves multiple objectives.

In this case a new process is being designed. The simulation model for the system is shown in Figure 11.18. There are alternative approaches being considered for the operation, which affect the average process time at the workstation. The dashboard in Figure 11.21 shows the performance of the system, based on one replication, when the mean process time is 9.7 seconds. Since the time between arrivals is exponentially distributed with a mean of 10 seconds, the server is heavily utilized, about 97%. However, the queue contents prior to the server becomes quite large; in the replication in Figure 11.21, the average content is 26.8 items and at one point the number of items in the queue reached a maximum of 91. While management likes the high utilization of the server, they do not like the high contents of the queue. They want to maximize utilization and minimize queue contents. Of course, these are conflicting objectives.

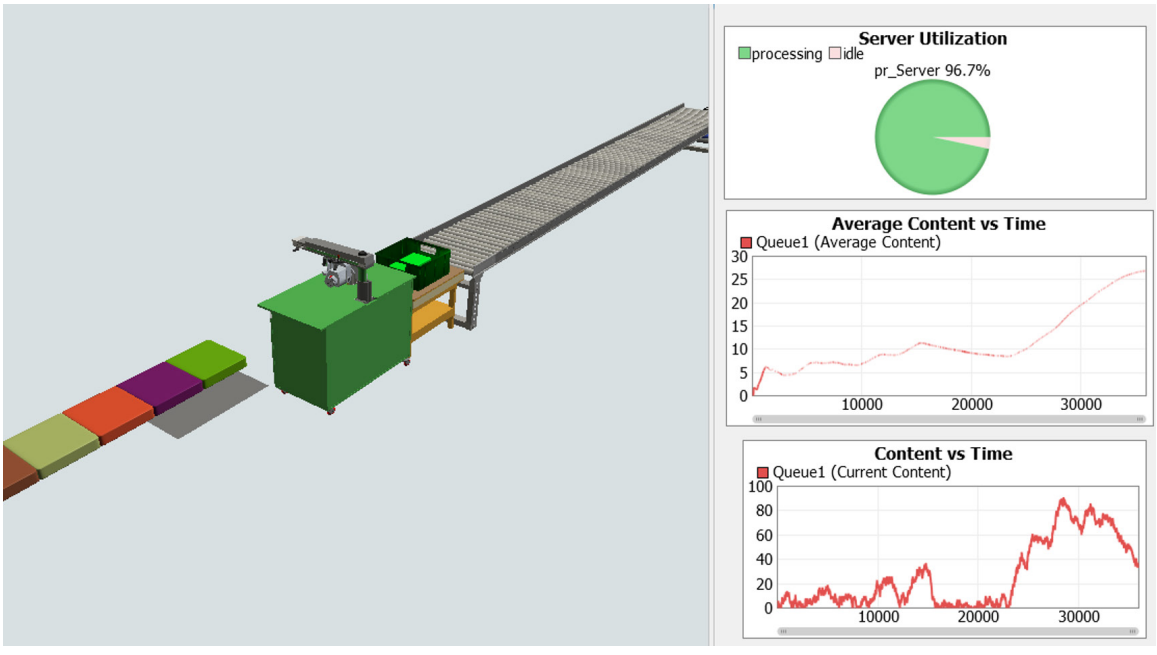


Figure 11.21: Simulation model for the process time optimization example.

The simulation model is used to consider the two objectives for various process times. The arrival process is fixed, but the service process can vary depending on the operational design. Due to the nature of the process, engineers indicate that the minimum process time is 3 seconds and the maximum time is 15 seconds. Based on basic queueing theory, the average arrival rate must be smaller than the average service rate; conversely, the average service time must be smaller than the average inter-arrival time. If this condition is not met, the system becomes overwhelmed and the queue grows infinitely large; therefore, the mean process time cannot be longer than 10 seconds.

Based on the above assessment of the service process, it is assumed that the service times are triangularly distributed. Recall, that the triangular distribution requires three parameters: minimum, maximum, and most likely. The mean is the sum of these parameters divided by 3. Therefore,

$$\mu = \frac{3 + 15 + \text{ML}}{3} \text{ and } \mu < 10$$

As a result, the ML value can vary between 3 and 12 seconds and the corresponding mean will vary between 7 and 12 seconds. This is illustrated in Figure 11.22.

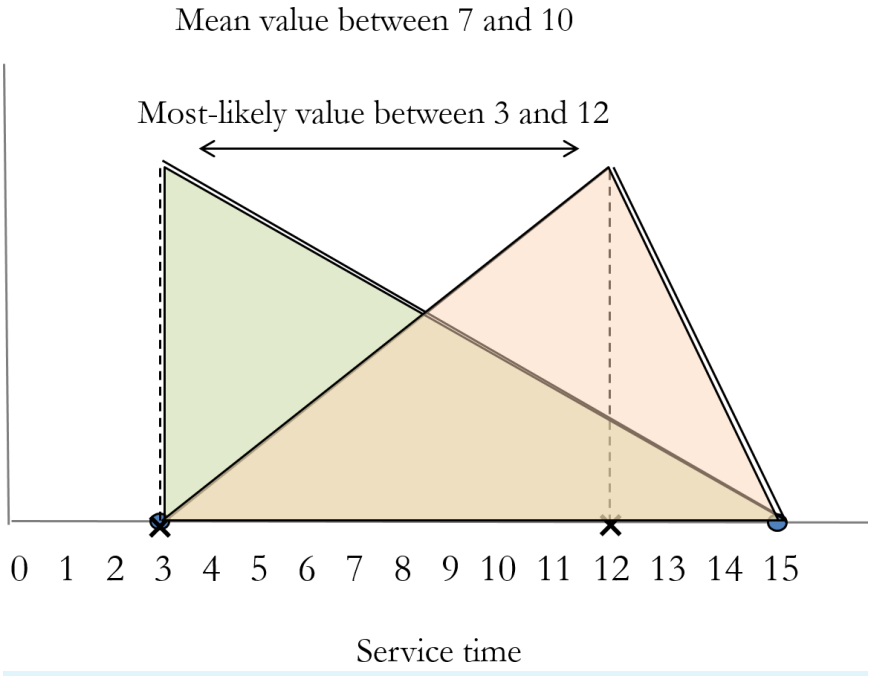


Figure 11.22: Distributions of process times.

The optimizer design is shown in Figure 11.23, with (1) a single decision variable, the most-likely process time that is constrained between 3 and 12 seconds, and (2) two objective functions, one to maximize server utilization and the other to minimize average contents in the queue.

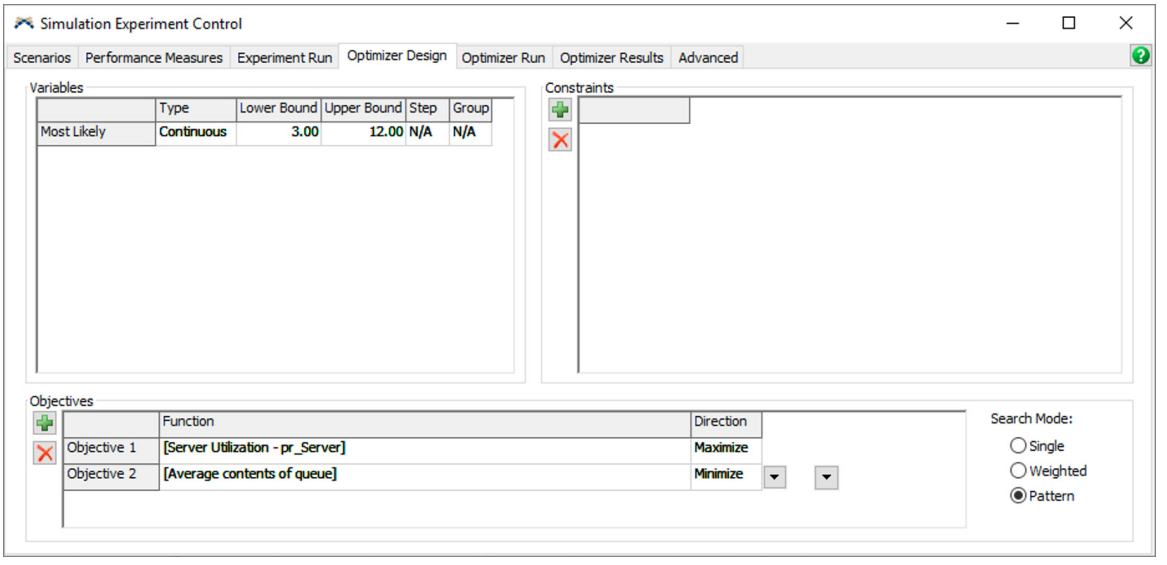


Figure 11.23: Optimizer design of process time example.

The output from the simulation optimization is shown in Figure 11.24. The optimization considered 100 possible solutions with each solution being replicated between 5 and 10 times. Each solution is replicated until an 80% confidence interval is satisfied with 5% error. It is interesting to note that there are multiple optimal solutions—in fact, the solutions

Figure 11.24: Output from the simulation optimization for the process-time example.

form a curve, similar to an efficient frontier. The curve provides the best operating condition based on a given server utilization and average queue contents. Any combination of server utilization and average queue contents that are above the curve, e.g. 95% and 3.0 items, is infeasible. Any combination of server utilization and average queue contents, that are below the curve, e.g. 70% and 10.0 items, can be improved.

As single optimal solution would result if a weight is provided for each objective, e.g. server utilization is twice as important as average contents. However, the results in Figure 11.24 are more useful—they allow the decision maker to select the optimal setting based on an explicit tradeoff between the two objectives. For example, three of the optimal solutions (39, 15, and 8) are selected for further analysis. To find the most-likely value that corresponds to each solution, use the Export Selected Solutions output option. The results are shown in the Scenarios tab at the bottom of Figure 11.24. The results are summarized in Table 11.1.

	Solution		
	8	**39**	**15**
Server utilization	81.3	90.0	95.0
Avg. queue contents	2.0	5.0	10.0
Most-likely process time	6.28	8.85	10.43
Average process time	8.1	9.0	9.5

Table 11.1: Sample results from the simulation optimization.

Based on the context, if management considered Solution 39 to be optimal (90% server utilization and average queue contents of 5.0), then the average process time should be 9.0 seconds.

Summary

This and the previous chapter provide guidance for performing statistical analyses on data obtained from discrete-event simulation models. It is critical that output from simulations is analyzed properly. In other words, no matter how good a model is or how well it represents the real system, improper analysis of the simulation output could, and likely will, lead to wrong recommendations and wrong decisions. As is pointed out in this and the previous chapter, much has been written in simulation literature about analyzing simulation output. It is not our intent to cover, or even survey, all of the methodologies and techniques that are available and their nuances, but to provide a basic and applied foundation for analyzing simulation output.

Any approach that is used to analyze simulation output is heavily dependent upon the type of system being considered—terminating versus non-terminating. In either case, the model must be replicated, or run more than once, since simulation models include random variables. Also, it is important to decide how long to run each replication and what the starting conditions are.

Of course, as with any experimental design, the decision variables and performance measures must be identified and clearly defined before using simulation. Since most analyses involve comparisons, the chapter provides basic methodologies for statistically comparing model output (performance measures) from multiple alternatives. In order to make comparisons more precise for a given number of replications, the chapter provides a simple approach for variance reduction (i.e., the use of common random numbers). The difference between statistically-significant results and practically-significant results is discussed.

The chapter concludes with an introduction to optimization—seeking the "best" alternative among many alternatives. The chapter discusses the use of *FlexSim's* Experimenter to facilitate analysis of simulation model output including the use of *OptQuest* for simulation optimization.

References

1. April, J., Glover, F., Kelly, J., & Laguna, M. "Practical Introduction to Simulation Optimization," *Proceedings of the 2003 Winter Simulation Conference*, S. Check, P. J. Sanchez, D. Ferrin, and D. J. Morrice, eds.

2. April, J., Better, M., Glover, F., Kelly, J., & Laguna, M. "Enhancing Business Process Management With Simulation Optimization," *BP Trends*, January 2005, pp. 1–11.

3. Banks, J., Carson, J, Nelson, B., & Nicol, D. (2010) *Discrete-Event System Simulation* 5th Ed. NJ: Upper Saddle River, Prentice Hall, Pearson Education, Inc.

4. Barton, R. "Designing Simulation Experiments," *Proceedings of the 2013 Winter Simulation Conference*, R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds., pp. 342-353.

5. Fu, M., Glover, F., & April, J. "Simulation Optimization: A Review, New Directions, and Application," *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

6. Hsu, J. (1996) *Multiple Comparisons: Theory and Methods*. New York: Chapman and Hall.

7. Jian, N., & Henderson, S. "An Introduction to Simulation Optimization," *Proceedings of the 2015 Winter Simulation Conference*, L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds., pp. 1780-1794.

8. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/NormTble.htm

9. Laguna, M. *OptQuest*: Optimization of Complex Systems. OptTek Systems, Inc. 9 May 2011.

10. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

11. NIST/SEMATECH *e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/handbook/, October 2013.

12. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, Chichester, England: John Wiley & Sons, Ltd.

13. Sanchez, S. "Work Smater, Not Harder: Guidelines for Designing Simulation Expreiments," *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds., pp. 69-82.

Chapter 11 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important—in the chapter.
2. Describe the activities that should be performed in order to define the key parameters in an experiment with a simulation model.
3. Describe how the interarrival time distribution was determined in the chapter's case study.
4. Based on the chapter's case study, describe the effect that the change in the definition of the system's arrival process has on estimated system performance.
5. Explain the difference between the results of an analysis being statistically significant and practically significant.
6. Describe the general effect of using Common Random Numbers in a simulation model.
7. Describe how to implement Common Random Numbers into a simulation model.
8. If a planned simulation experiment involves three factors, each at four levels, how many alternatives will be considered and how many pairwise comparisons would be made among all alternatives?
9. If six options are compared to a base case, what level of significance should be used in the analysis if the desired overall α = 0.10?
10. Assume each of the following variables is to be specified for an optimization analysis using *OptQuest*. Identify its variable type:
    a. batch size
    b. queue size
    c. process time
    d. time between preventative maintenance checks
    e. item processing rule based on the item's part family
    f. distance between two work cells
    g. sequence to process a set of five different products
11. A change is being proposed in the operation of the hospital's imaging department that was analyzed in Review Problem 8 in Chapter 10. The option was modeled and simulated for 15 replications. The mean patient wait time for the proposed option and the current system

(base case) are shown in the table below. Is there a statistically significant difference in mean patient wait time between the current system and the proposed improvement? Also, comment on the practical significance of the results.

Rep	Base	Option	Diff
1	50.8	93.3	-43.1
2	63.8	62.4	1.4
3	51.2	111.4	-60.2
4	98.2	51.5	46.6
5	123.5	29.3	94.2
6	63.8	37.6	26.2
7	47.8	60.7	-12.9
8	33.5	171.6	-88.1
9	43	85.5	-42.5
10	112.5	22.1	90.4
11	59.5	52	7.5
12	52.7	78.8	-25.1
13	109.6	61	48.6
14	95.2	35.6	59.7
15	21.8	60.5	-38.6
Mean	**68.5**	**64.3**	**4.2**
Std. Dev.	**31.3**	**29.2**	**55.4**

Appendix 11A – t-distribution table

The following table is in the public domain and isproduced by an APL program written by William Knight; accessed at: http://www.math.unb.ca/~knight/utility/t-table.htm

PERCENTAGE POINTS OF THE T DISTRIBUTION

Tail Probabilities									
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
D	1	3.078	6.314	12.71	31.82	63.66	318.3	637	1
E	2	1.886	2.920	4.303	6.965	9.925	22.330	31.6	2
G	3	1.638	2.353	3.182	4.541	5.841	10.210	12.92	3
R	4	1.533	2.132	2.776	3.747	4.604	7.173	8.610	4
E	5	1.476	2.015	2.571	3.365	4.032	5.893	6.869	5
E	6	1.440	1.943	2.447	3.143	3.707	5.208	5.959	6
S	7	1.415	1.895	2.365	2.998	3.499	4.785	5.408	7
	8	1.397	1.860	2.306	2.896	3.355	4.501	5.041	8
O	9	1.383	1.833	2.262	2.821	3.250	4.297	4.781	9
F	10	1.372	1.812	2.228	2.764	3.169	4.144	4.587	10
	11	1.363	1.796	2.201	2.718	3.106	4.025	4.437	11
F	12	1.356	1.782	2.179	2.681	3.055	3.930	4.318	12
R	13	1.350	1.771	2.160	2.650	3.012	3.852	4.221	13
E	14	1.345	1.761	2.145	2.624	2.977	3.787	4.140	14
E	15	1.341	1.753	2.131	2.602	2.947	3.733	4.073	15
D	16	1.337	1.746	2.120	2.583	2.921	3.686	4.015	16
O	17	1.333	1.740	2.110	2.567	2.898	3.646	3.965	17
M	18	1.330	1.734	2.101	2.552	2.878	3.610	3.922	18
	19	1.328	1.729	2.093	2.539	2.861	3.579	3.883	19
	20	1.325	1.725	2.086	2.528	2.845	3.552	3.850	20
	21	1.323	1.721	2.080	2.518	2.831	3.527	3.819	21
	22	1.321	1.717	2.074	2.508	2.819	3.505	3.792	22
	23	1.319	1.714	2.069	2.500	2.807	3.485	3.768	23
	24	1.318	1.711	2.064	2.492	2.797	3.467	3.745	24
	25	1.316	1.708	2.060	2.485	2.787	3.450	3.725	25
	26	1.315	1.706	2.056	2.479	2.779	3.435	3.707	26
	27	1.314	1.703	2.052	2.473	2.771	3.421	3.690	27
	28	1.313	1.701	2.048	2.467	2.763	3.408	3.674	28
	29	1.311	1.699	2.045	2.462	2.756	3.396	3.659	29
	30	1.310	1.697	2.042	2.457	2.750	3.385	3.646	30
	32	1.309	1.694	2.037	2.449	2.738	3.365	3.622	32
	34	1.307	1.691	2.032	2.441	2.728	3.348	3.601	34
	36	1.306	1.688	2.028	2.434	2.719	3.333	3.582	36
	38	1.304	1.686	2.024	2.429	2.712	3.319	3.566	38
	40	1.303	1.684	2.021	2.423	2.704	3.307	3.551	40
	42	1.302	1.682	2.018	2.418	2.698	3.296	3.538	42
	44	1.301	1.680	2.015	2.414	2.692	3.286	3.526	44
	46	1.300	1.679	2.013	2.410	2.687	3.277	3.515	46
	48	1.299	1.677	2.011	2.407	2.682	3.269	3.505	48
	50	1.299	1.676	2.009	2.403	2.678	3.261	3.496	50
	55	1.297	1.673	2.004	2.396	2.668	3.245	3.476	55
	60	1.296	1.671	2.000	2.390	2.660	3.232	3.460	60
	65	1.295	1.669	1.997	2.385	2.654	3.220	3.447	65
	70	1.294	1.667	1.994	2.381	2.648	3.211	3.435	70
	80	1.292	1.664	1.990	2.374	2.639	3.195	3.416	80
	100	1.290	1.660	1.984	2.364	2.626	3.174	3.390	100
	150	1.287	1.655	1.976	2.351	2.609	3.145	3.357	150
	200	1.286	1.653	1.972	2.345	2.601	3.131	3.340	200
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Tail Probabilities									

Appendix 11B – Standard normal distribution table

The following table is in the public domain and isproduced by an APL program written by William Knight; accessed at: http://www.math.unb.ca/~knight/utility/NormTble.htm

Probability Content from -oo to Z

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

Chapter 12 Logical Relationships Among Objects

While the Intermediate User can study a wide variety of dynamic systems using simulation, there will still be a need to simulate actions that cannot be described using the preset logic. An understanding of the simulation software is important to accurately create custom logic and use advanced features to analyze more complex situations. *FlexSim's* open architecture allows for this understanding without the need to be a software professional.

Objective

To define how simulation software is structured and describe how to modify the standard functionality of simulation objects to more accurately represent the logical actions being represented.

Most Advanced Users will either have a definite interest in simulating systems or will carry out simulations on a relatively frequent basis. An Advanced User normally will also be knowledgeable about the subject matter that makes up the simulation problem. While this section develops an extended knowledge of the simulation software, it does not completely describe all the advanced functionality available in the application.

The primary background required for the Advanced User is knowledge of the underlying software structure and the basic programming commands for building custom logic. The Advanced User also becomes more proficient in dealing with information that is exchanged between a simulation and external programs.

Additional skills include utilizing the charting and reporting functionality of the software, enhancing the visualization of the model, etc. This section provides an overview of the software architecture and its scripting commands. This book is not a software manual so for the most up to date information check the *FlexSim* User Manual, available through the Help menu.

Section 12-1 Hierarchical software architecture

The *FlexSim* software is unique in many ways from the other simulation applications discussed in Chapter 2. The most important is that it was designed with an object-oriented software structure. Object-oriented programming was developed to improve functionality and maintain software quality as computer hardware and software became increasingly complex. An object-oriented program is a collection of interacting objects. Each object can be viewed as an independent entity, which has a distinct role or responsibility, and which knows how to manipulate its own data structure.

This is different than the approach that had been dominant for many years that keeps data and functions separate and considers the application program as simply a long list of commands and tasks to be performed.

The use of object-oriented techniques in the development of *FlexSim* means that it is hierarchical and data oriented in nature and consequently can be visualized using a tree structure used for databases. It's called a tree even though the visualization is generally upside down compared to an actual tree, with the "root" at the top and "leaves" at the bottom. A structured representation of a simple simulation containing four elements is shown as a tree in Figure 12.1. A tree structure is conceptual and appears in several forms. Common examples of tree representations are the folder/file graphical interfaces found on computer systems (e.g. Windows Explorer). In *FlexSim*, the basic elements in the tree view are called nodes.

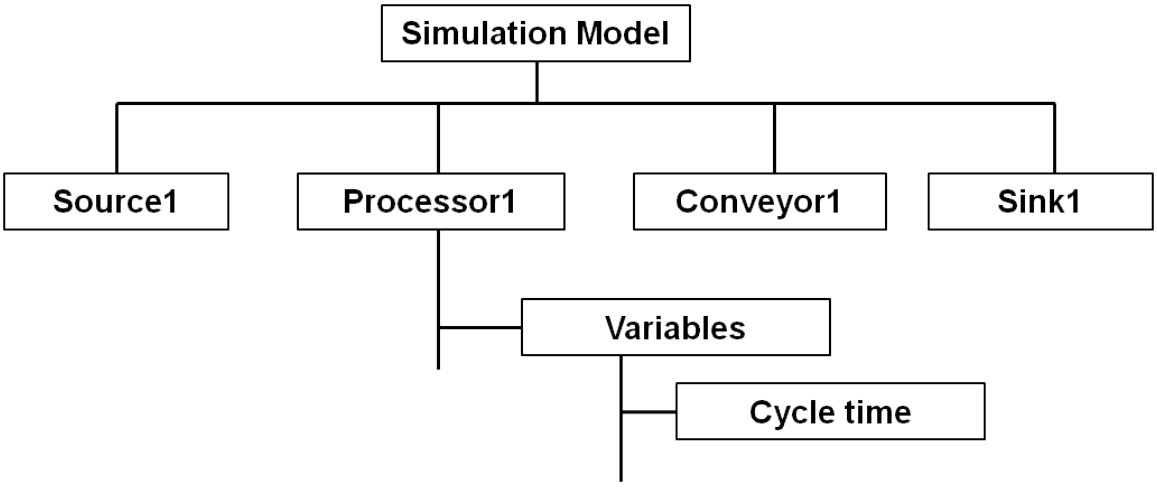


Figure 12.1: Basic tree structure.

The tree provides a very robust, visual, and efficient way to find where variables are stored or where particular sections of code reside. It is a modern alternative to the traditional way of programming applications. Older software applications were created by writing code as a long list of commands. Code that was reused in the application was established as a sub-routine that could be recalled for use; however, searching through all the lines of code and subroutines to find a reference to a variable, or knowing where certain commands are

located, was confusing and difficult. The tree, with its hierarchical, object-oriented structure changes all that.

The following list contains the primary attributes of the tree:

- Tree groupings
  - Main tree: Application structure and project-related objects and data
  - View tree: GUI related objects and picklists
  - Model tree: Model related objects and data, part of the main tree
- Nodes: the basic building blocks of the tree; they
  - hold all the model definition information "behind the scenes";
  - have a name;
  - can have a data item associated with them: a number, string, or object.

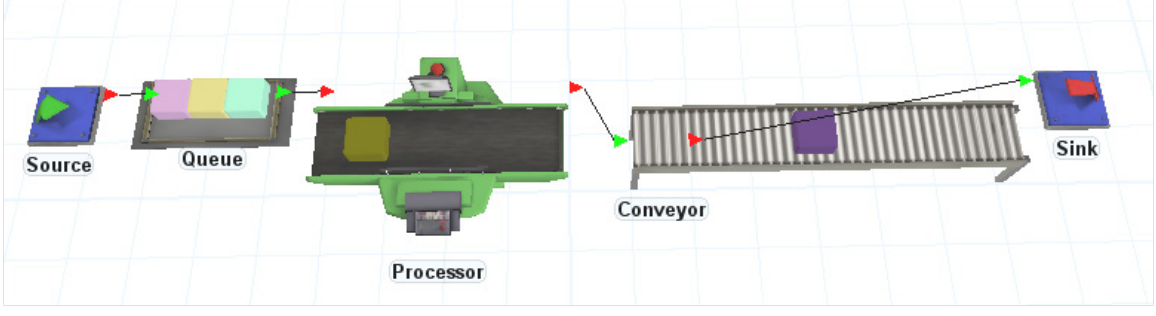


Figure 12.2: Simple model.

Consider a simple model made up of a source, queue, processor, conveyor, and sink, as shown in Figure 12.2. To see the tree view of this model in *FlexSim*, right click in a blank space on the surface and select Explore Tree from the popup menu. The tree structure appears with the model as the first reference. Quick Parameters for the tree appear on the right side of the screen. Figure 12.3 shows the tree view for the model in Figure 12.2.

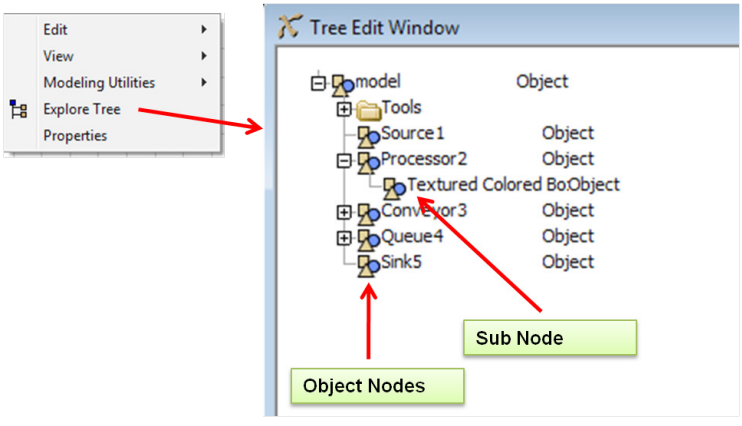


Figure 12.3: Tree Structure for simple model.

The nodes in the *FlexSim* tree hold all the information that a *FlexSim* model uses to run a simulation. The *FlexSim* tree stores all the parameters and properties for objects, graphical user interfaces, event lists, and other information. The nodes representing the objects in the model are called object nodes. Object nodes contain data nodes and functions that define how the object looks and behaves. Note that to the left of the object data node, there is a control that looks like a plus sign in a box. This control means that the node has subnodes, which can be viewed by clicking on the icon. The subnodes represent objects, such as flowitems, that are contained within the primary object node.

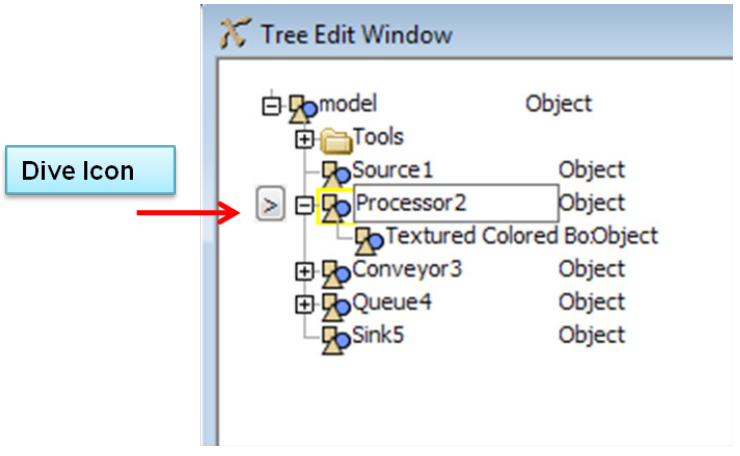


Figure 12.4: Using the dive icon.

The node names generally represent the type of data within them. For example, the class and superclass nodes contain information about the parent objects from which the selected object inherits its behavior. The visual and special nodes contain data about the appearance and location of the object. The labels node contains the data on the object's labels.

Many of the data nodes can be expanded into subnodes (▷) by clicking to the left of the node. As shown in Figure 12.4, clicking on the Processor2 node results in the dive icon appearing to the left of the object node icon. Pressing the dive icon expands the node to show the nodes containing the processor's data.

The variables node is especially important because it contains most of the operating variables and logic for the object. An expansion of the variables node is shown in Figure 12.5. Nodes like cycletime contain the code chosen for that item on the interface. If these nodes contain custom code that is to be executed, then the circle on the node changes to a square with an S in the middle, indicating that the code is Flexscript.

Values of nodes can be changed by using Flexscript commands and providing a path to the node from any location. The path can take the form of a typical file structure path or be defined automatically through simplified Flexscript commands.

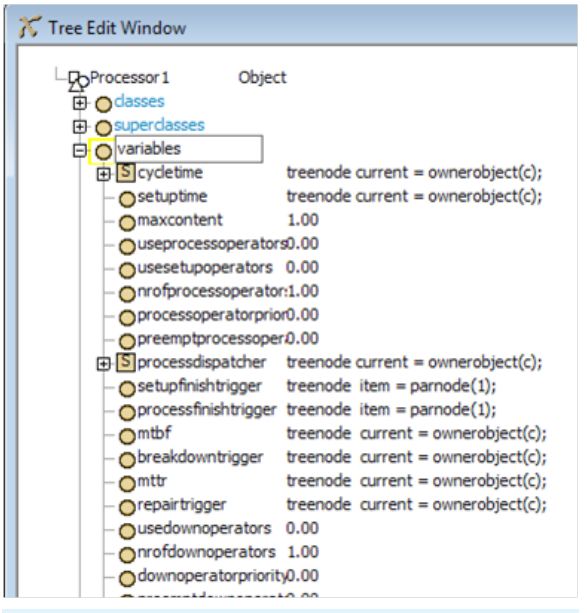


Figure 12.5: Expansion of the variable node.

A summary of tree of node symbols is provided in Figure 12.6; they indicate the functionality of the node.

With this important software structure, all nodes, variables, and other functions can be easily located and modified dynamically. Many of the nodes are directly linked to entries on the user interface of individual objects. That means that the value is dynamically synchronized in both locations.

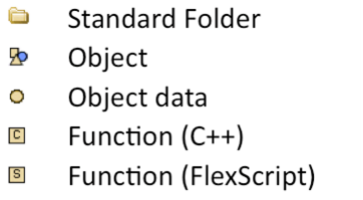


Figure 12.6 Tree node symbols.

Section 12-2 Understanding how objects work

The previous chapters of this book used various characteristics of objects without discussing how they worked in detail. This section examines what happens to an object when a flowitem enters it.

Figure 12.7 shows the order of events for a processor, but it is similar for other object types. The first thing to happen when an item enters an object is the execution of the OnEntry trigger. Its function is to provide a means of customizing what an object does when a flowitem enters.

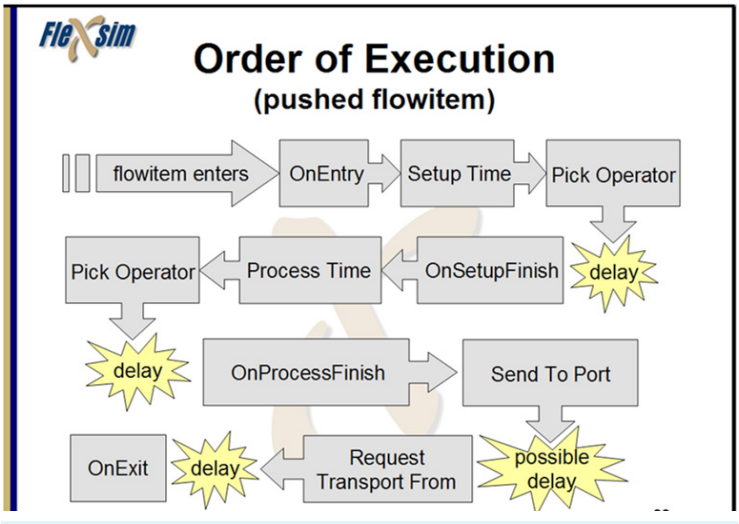


Figure 12.7: Processor order of execution.

For the processor's next step, a variable called Setup Time is evaluated. The setup time is a mathematical expression that returns to the object how long to delay the flowitem to represent a setup time. If a value of zero is returned, then the delay time is zero. This execution is different from a trigger in that it is always executed; triggers are only executed if there is logic associated with their designated event.

If an operator (task executer object) is required for use during a setup time, the Pick Operator field is evaluated. This field returns a reference (usually via a center port connection) to a task executer object. The next step is a delay that is equal to the value returned from the evaluation of the setup time trigger, plus any time required by the task executer to become available and to travel to the object.

Upon the completion of the delay, a trigger called OnSetupFinish fires to signal the end of setup time. This event, as with all such events, allows the modeler to add customized behavior.

The step marked Process Time behaves the same as setup time. The Process Time field is evaluated, and the value returned to the object represents the length of time needed to process the flowitem. Next an operator is picked, if applicable.

Next is another delay that corresponds to the delay given by process time plus any travel time for the selected task executer. The delay is followed by the firing of the OnProcessFinished trigger. This trigger signals the end of the process time and may be used to add logic that should occur before a determination is made as to where to send the item. Such logic may start a downstream machine to get it ready to accept the item.

Once the process time is complete, the SendtoPort field is evaluated, which returns a value to the object that signals what output port to send the flowitem through. The step marked in Figure 12.7 as "possible delay" is important because it will effectively stall execution of the object until any delay happening downstream of the object is over. The possible delay is caused by the situation where the downstream object may not be ready to receive the flowitem; the downstream object may be at full capacity, busy processing, etc. It is a possible delay (as opposed to a definite delay because it is purely a timing issue between the upstream and downstream objects. This delay happens in most models).

Request Transport From is the next field on the Flow tab to be evaluated. This field behaves like the Pick Operator fields for process and setup times. It returns a reference to a task executer that is assigned to come and transport the flowitem, followed by a delay. If a task executer is requested by this field, the delay corresponds to the time it would take the task executer to finish its current task (if applicable) and travel to the object making the request. This isn't listed as a possible delay, because the travel time to the object will always take some amount of time.

Lastly, the OnExit trigger fires just before the flowitem exits the object. When the flowitem causes this trigger to fire, the port and operator (if applicable) are already known and engaged in transporting the flowitem away. Neither port nor operator can be changed at this point, as the flowitem is already gone, as far as the object is concerned.

Because of this order of execution, the OnExit trigger of a Source is an ideal place to assign itemtypes to the model's flowitems. The OnCreation trigger is the other choice, but usually it is safer to assign the itemtypes during OnExit since it is the last possible chance to make a change to a flowitem before it is in a different object.

A common misconception is to confuse the functionality of the SendtoPort field with the OnExit trigger. These occur at different times and function in different ways. The SendtoPort field is a basic function of the object and always occurs as shown in the figure. The port number that it returns is set. The OnExit trigger occurs when the item leaves the object and, at that point, it cannot override the value returned by the SendtoPort function.

Section 12-3 Scripting in *FlexSim*

Flexscript Basics

The picklist options, introduced for the Intermediate User, provide a wide range of commands to control the simulation without writing any code; however, if a simulation is intended to examine truly unique strategies that provide a definitive advantage, then writing some level of unique, custom code is to be expected.

FlexSim uses a scripting language called Flexscript to create custom logic, which is modeled around the syntax of the popular programming languages C/C++ and follows most of the same conventions found in those languages. Anyone familiar with C/C++ can quickly begin writing custom modeling code. Code in *FlexSim* can be customized where the interface contains the "scroll" icon ( ).

Some basic rules and guidelines for using Flexscript include the following:

- *Language is case sensitive*. "`A`" does not equal "`a`"; `Processor1` is not the same as `processor1`.
- *No specific formatting is required*. Use of spaces, tabs and line returns is encouraged for more "readable" code.
- *Text strings are entered between quotes to be flagged as text*. For example, `"sometext"` would be treated as a string, `sometext` would not. It is important to note that the code error checker will NOT check the spelling or correctness of text fields. This can be confusing, however, since in some drop down menus text may be entered without quotation marks as they are added by the software association with the picklist item. When using picklist, follow the format in the pick list.
- *Parentheses follow a function call, and commas separate the arguments of the function*. For example, `moveobject(object1, object2);`
- *A completed statement (function or command) must always end with a semicolon (`;`).*
- *Parentheses can also be used freely to make associations in math and logic statements*. For example, `((x+3)*2)`. Parentheses must always be paired. If you put your cursor next to an opening parenthesis, *FlexSim* will highlight it and its closing partner. Putting your cursor next to a closing parenthesis will highlight it and its matching opening parenthesis.
- *Curly braces `{}` are used to define a block of statements*. Blocks of statements also define a scope and are used with if, for, and switch statements.

- *To comment a line use* **//** *followed by the comment*. Comments appear in green text and are ignored by *FlexSim*. The comment area ends with a line return.
- *Multi-line comments start with* **/*** *and end with* ***/**.
- *It is best not to use spaces or special characters in name definitions*. The underscore character (_) is the only "safe" special character. Think of it as a 27th letter of the alphabet.
- *Names may include numbers but may not begin with a number*. For example, **machine_9** is acceptable; 9machine is not.

A good way to help understand coding is to select a picklist option and study it's underlying code. For example, the picklist option in Figure 12.8 is used to set an item's color to red when it exits the source.

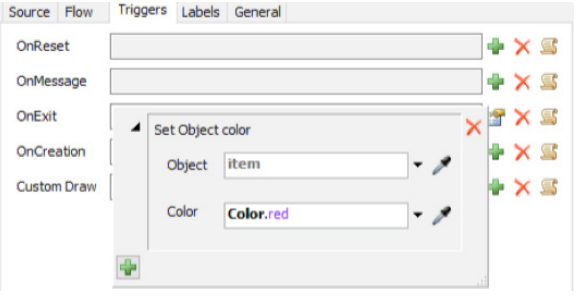


Figure 12.8: Selection to change color.

The code that implements this trigger (after clicking on the scroll icon) is shown in the figure 12.9a. Some of the code, shown in light gray, is used to create the picklist option in the user interface. It is not needed for direct coding and can be removed by pressing the clear-markup icon in the bottom left corner of the code window (the icon between the question mark and the check icons). The result is shown in figure 12.9b. The PickOption Start and PickOption End lines (Lines 6 and 11) can also be removed by manually deleting them.

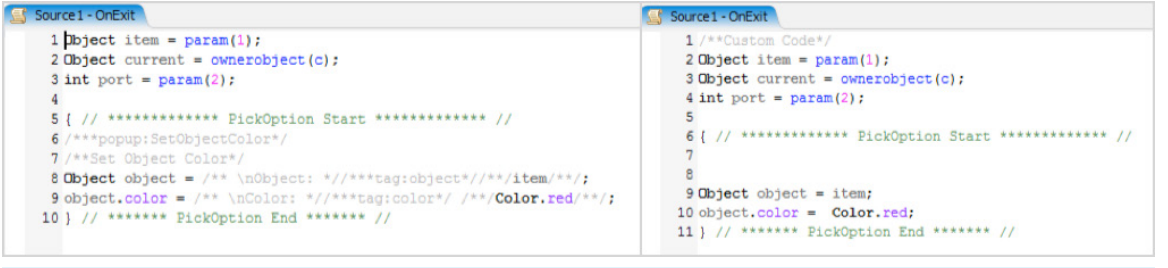


Figure 12.9: (a, left) Code with Mark-up commands; (b, right) mark-up code cleared.

This example also shows some important keywords when referencing within a single object:

- **current** references the present object—the one that owns that code
- **item** references the flowitem currently in the object that caused the custom behavior to execute
- **port** references the port associated with the trigger event

Flexscript Variables

Every variable in a model must have a data type declared before it is used in the coding statements. The declaration is only made once. In the example above, item and current and are both of data type Object (a class that represents objects in the model). The port number is of data type integer (int).

- *Integer*: declared **`int`**, can only hold integer values: the natural numbers, their negatives, and zero.
- *Double*: declared **`double`**, can hold the set of real numbers, such as 3.14159, -6.1, etc. The double type is considered a more precise variable type compared to the integer type: while a double type may hold an integer value, an integer type may not hold a real number.
- *String*: declared **`string`**, holds text which is specified within quotation marks; (e.g., **`"my Text"`**).
- *Object*: declared **`object`**, represents an object in the model.
- *Treenode*: declared **`treenode`**, is unique compared to the other three variable types and only exists within *FlexSim*. Treenodes hold references to nodes in the *FlexSim* tree.

Before a variable can be used it must be created or declared. A declared variable must have its type and name given in the declaration, but no value need be assigned in a declaration. For example, the following would correctly declare a double variable result that would hold the result of a sample from a uniform distribution:

```
double result;
result = uniform(0, 10);
OR double result = uniform(0, 10);
```

The single line of code is equivalent to the two lines above. Other examples are

```
int index = 1;
double weight = 175.8;
string category = "groceries";
treenode MyProcessor = node("/processor2", model());
```

Code that creates a change in an object's property or behavior is composed of statements that are executed sequentially unless the sequence is changed by a flow or control construct; these include:

- *if/else* – executes one set of statements if a specified test condition is true and another if the condition is false.
- *switch* – executes a set of statements depending on the value of the switch variable.

- *while* loop – executes a set of statements until a specified test condition is false.
- *for* loop – executes a set of statements for a specified number of times.

Flexscript Statements

Statements are used to obtain information about objects, manipulate information, and change information about objects. Statements contain variables that have their values set by operations or commands. As shown below, the commands take three general forms, with the first two using the new **dot notation** and the last being the global *FlexSim* command. The dot notation is commonly used in object-oriented systems.

1. **object.attribute** – obtains the value of an object's attribute. When used on the left side of =, the value of the attribute is set to whatever is on the right side of the =.
2. **object.method** – a method is applied to the specified object.
3. **name(param1, param2, …)** – function name and a set or arguments (param1, param2, etc.) within parentheses, separated by commas.

1. object.attribute commands

This command format starts with the object of interest (e.g., current or item) followed by a system characteristic of that object (e.g., its size, location, color) or a user-specified characteristic, i.e., a label. For example, to change the color of a flow item that is about to exit an object, e.g. Source2, the reference is made to the item object; similarly, to change the color of the Source2 object, the reference is to the current object. In the statement below, the flow item's color is changed to red as it leaves the source.

```
item.color = Color.red;
```

Both sides of the equal sign use the dot notation format **object.attribute**. In this example, the value of the current flow item's attribute color is set equal to the value of the Color object's attribute called red. This is all that is required to add this behavior to the source; it is not necessary to know the details of exactly how the computer makes this change and actually colors the object.

Of course, as indicated earlier in Figure 12.9, incorporating a color change in the trigger could have been implemented by merely selecting a dropdown-menu option. When using a dropdown option, it just adds the above statement to the trigger.

The object.attribute notation is used for a modeling object's or flow item's physical properties and for any label that has been defined for the object. The following properties are some that are available, where object typically refers to either item or current; however, it can reference any other object in the model. A full list is available in *FlexSim's* Help file.

```
object.name
object.color
object.location.x, object.location.y, object.location.z
object.size.x, object.size.y, object.size.z
object.rotation.x, object.rotation.y, object.rotation.z
object.labelname
object.subnodes.length (number currently in the object)
```

As shown below, an alternative means of setting an object's three dimensions in one command is to use the Vec3 data type. Similarly, any color can be defined through the three RGB (red, green, blue) parameters using the Color data type.

```
item.location = Vec3(0, 0, 0)   // object location is origin
Color myColor = Color(0, 1, 0)   // makes the object pure green
```

2. object.method commands

Another format for commands is one that uses *methods*, again using the dot notation. This command format starts with the object of interest (e.g., current or item) followed by a method name. For example, instead of setting the location of an object using three statements from above, one for each dimension, the change can be invoked in one statement using the method setLocation. If the method is invoked in the OnEntry trigger of a processor, the incoming flowitem would be located 2 grid units to the right, two grid units below, and one grid unit above the processor's reference point (e.g., upper left corner).

```
item.setLocation(2.0, -2.0, 1.0)
```

The following methods, in dot-notation format, are available for all modeling objects, where object refers to either the current object or a reference to another object in the model. The first four statements also pertain to flow items. A full list is available in *FlexSim's* Help file.

```
object.setLocation(x, y, z)
object.setSize(x, y, z)
object.setRotation(x, y, z)
object.labels.assert("labelName").value (for use when creating a new label and providing it a value)
object.centerObjects(port number) (points to the object connected through a center port; similarly, the following point to an object connected to an input port and output port, respectively)
object.inObjects(port number)
object.outObjects(port number)
```

```
node.find(path) e.g. model().find("Processor1")
```

A full list of object attributes, methods, and labels are available in the context-sensitive help as the coding statements are entered, as shown in the box to the right. In this case, the first two entries in the list are the item's labels; the remainder of the entries are attributes or methods.

The object.method syntax is also used when interacting with tables, where the object is the table name. As shown in the example below, the variable name is myTable, its data type is Table, and the reftable command points to the table that has been defined in the model and is named GlobalTable1.

```
Table myTable = reftable("GlobalTable1");
```

Tables have the following properties.

`myTable.numRows` (the current number of rows in the table)

`myTable.numCols` (the current number of columns in the table)

The following methods are used to act on a table; in this example, the variable that points to the table is named myTable.

`myTable[row][column]` (reference a specified cell in the table; if used on the right side of an assignment the cell value is provided to a variable; if used on the left side of an assignment the cell value is set by the operation on the right side of the assignment)

`myTable.clear()` (clear the contents of the table)

`myTable.setSize(rows, columns)` (define the table size)

3. Function-style commands

A third means to invoke custom actions in a model via code is with the format that was used for all commands in all versions of *FlexSim* prior to 2017. The format is similar to a function call and a set of arguments, **command(parameter1, parameter2, …)**, where the parameters are values the command needs in order to execute. For example, to find out how many flow items have entered the current object since the simulation began, the command is:

```
getinput(current);
```

The following is a list of commonly-used commands. A full list is available in *FlexSim's* Help file, organized either by command name or by category.

Modeling Objects

- Number of flow items: total number that have entered the object `getinput(object)`, and total number that have exited the object `getoutput(object)`.
- Ports: `closeinput(object)`, `openinput(object)`, `closeoutput(object)`, `openoutput(object)`. Affects all port of

the specified type on an object; individual ports can only be controlled in fluid objects.

- States: `getstatenum(object)`, `getstatestr(object)`
- Variables: `getvarnum(object, "variablename")`, `setvarnum(object, "variablename", value)`
- Messaging: `senddelayedmessage(toobject, delaytime, fromobject, param1, param2, param3)`

Flow items

- Time entered object: `getentrytime(item)`
- Time created: `getcreationtime(item)`

Modeling Objects & Flowitems

- Shape: `getobjectshapeindex(object)`, `setobjectshapeindex(object, indexnum)`
- Texture: `getobjecttextureindex(object)`, `setobjecttextureindex(object,indexnum)`

In addition, the FlexScript commands include:

- Math functions, such as trig functions (`cos()`, `acos()`, etc.), `log()`, `exp()`, `log10()`, modulo `fmod()`, `maxof()`, `minof()`, `pi()`, `pow()`, `round()`, `abs()`, `factorial()`, `sqrt()`, etc.
- Statistical distributions, such as `bernoulli(…)`, `poisson(…)`, `exponential(…)`, `normal(…)`, `uniform(…)`, `duniform(…)`, `weibull(…)`, `gamma(…)`, `beta(…)`, `empirical(…)`, `dempirical(…)`, etc.

Section 12-3 Scripting tips

Flexscript Basics

1. To see the code without the comment and template fields, and to prepare for adding or modifying the code, click the clear template view icon at the bottom of the window. The resulting window starts with a comment section containing the words Custom code. Replacing this string will change the text in the drop-down window that defines the logic. Comment sections that indicate the range of the pick option remain but can be deleted.
2. Check the syntax of the code by clicking on the checkmark icon at the bottom of the screen. Any errors will pop up in a compiler console

window at the bottom of the *FlexSim* window. The console indicates the type of error and the line number. In this case, a semicolon has been omitted.

There are also visual cues to help keep the code correct. It is easy, therefore, to spot a potential error even before running the code checker because it will not appear in the appropriate color. (Default colors may be changed by selecting File/Global Preferences.) For example, misspelling the keyword "current" would result in:

`curent` rather than `current`

or

`Current` rather than `current`

It is important to realize that string data contained in quotes is not checked for accuracy; therefore, any misspelling will result in the function not operating, but no error message will be given. For example, trying to read the maximum content variable of a queue coded as `getvarnum(current,"maxcotent")` (notice the missing 'n' in "maxcontent") would return a value of zero, but would not give an error message.

Use the // marks to create comments which can help in remembering what the logic is supposed to do. Comments begin at the // marks and extend to the end of the line. Use /* to start and */ to end when commenting out a series of lines.

Some of the most common logic expressions involve either "if" statements or "for" loops, so it's critical to have the syntax correct. In each case, both parenthesis ( ), and brackets { }, are used to identify the scope of the script. Putting script on separate lines, noting comments, and using indents makes for easier reading and debugging. For example, consider the following if statement, for loop, and switch statement (note the use of the semicolons (;) used on all statements except the if, else, for, switch, and case):

If Statement

```
/* If the current content of the current object is >5,
then do statements 1 & 2. Else do 3 & 4. */
if(content(current) > 5)
    {
    Statement1;
    Statement2;
    }
else
    {
    Statement3;
    Statement4;
    }
```

If the braces { } are omitted after an 'if' statement, only the first line will be executed.

For Loop

```
//Repeat statements 1 & 2 five times, increment x by 1
for(x=1; x<=5; x++)
    {
    Statement1;
    Statement2;
    }
```

Switch Statement

```
//Take action based on the value of variable
switch(variable)
{
//in the case that variable==1, execute statements 1 &
2
case 1:
    {
    Statement1;
    Statement2;
//the break statement takes control out of the switch
statement
    break;
    }
//in the case that variable==2, execute statements 3 &
4
case 2:
    {
    Statement3;
    Statement4;
    break;
    }
//default provides a place to go if variable is not a
1 or a 2
default:
    {
    break;
    }
```

Checking for errors

When finished writing the custom code, it is important to click the check mark icon at the bottom of the edit window. If there are errors, they will be noted along with the reason

and the line number of the code where the error occurs. The most common errors include missing a parenthesis or bracket, missing a semicolon at the end of the line, misspelling a command, or misplacing a comma.

As previously stated, some errors are not found by the syntax checker but will yield an incorrect result if not corrected. These include misspelling a text string enclosed in quotes, misplacing a comma (in a command that doesn't give a syntax error but changes the command meaning), or providing an incorrect table reference.

When finished checking the code, click on the OK button at the bottom of the window to save any changes. Clicking on Cancel or the X in the top right corner of the window will not save any changes.

Dividing by zero can cause unpredictable results, so to protect from dividing by a variable that may be zero because of an error or because of another reason (such as a count of items at the beginning of a simulation), test the value of the variable in the denominator before using that variable. Using the command `maxof(0.0001,variable)` in the denominator is a good way to accomplish this.

Description details and examples can be found in the Commands selection in the Help menu. Also in the Help menu is the *FlexSim* User Manual which contains a section on writing logic in Flexscript.

Global variables

A variable that is used many times in a simulation may be defined once as a global variable and then referenced whenever it is needed. Global variables are created by clicking on Tools and going to the tools menu on the left side of the screen.

As shown in Figure 12.10, there are four steps involved in setting up a global variable:

1. Click on Tools to bring up the toolbox list
2. Select Modeling Logic and then Global Variable
3. Set the name and type of variable
4. Set an initial value

If the variable is a treenode type, use the browse button to select the node, and the path will be automatically assigned to the variable.

The initial value of the variable, as seen in the Global Variables menu, does not change dynamically. The actual value will remain at its last assigned value; therefore,

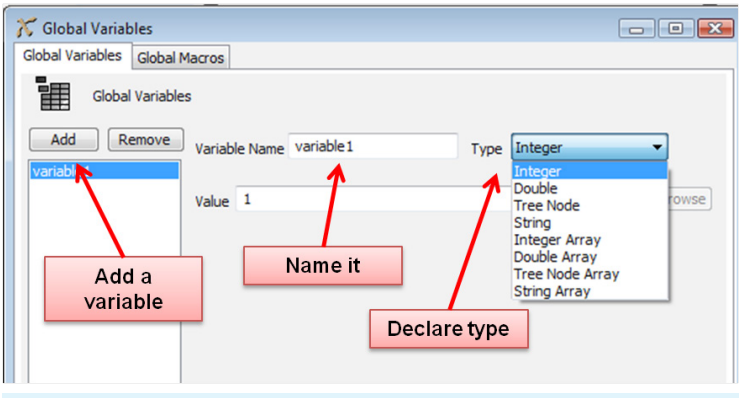


Figure 12.10: Creating a global variable.

it may be necessary to initialize the value with an OnReset trigger. You can use the Pin icon to place the global variable on a dashboard to keep track of it dynamically during the simulation.

Once a global variable has been established it can be used in logic statements anywhere in the simulation simply by referencing it since it has already been created. For example, an integer variable declared as a global variable and named project can be used as

```
project = 7;

x = project + 2;
```

A global variable defined as a treenode that points to Processor1 can be referenced

```
getoutput(Processor1);    // use the variable
```

Troubleshooting hints

There will be times when the simulation does not seem to work correctly even though the syntax checker does not indicate any errors. If the logic checker indicates that the behavior is unexpected, it may not indicate a real error in syntax. There is also the possibility that there may not be a problem, because, while the behavior is unexpected, it may be correct based on how the logic is implemented. In either case, it is important to find the logic that's causing the problem. There are many techniques and tools available to help troubleshoot.

One approach is to print out information at various points in the logic either to a file or to the Output Console, which is opened by selecting it from the Debug menu at the top of the screen. The following commands print to the console:

- `pt("text string")` prints a text message.
- `pf(number)` prints a number as a floating point value.
- `pd(number)` prints a number as an integer value.
- `pr()` with no parameters prints the next statement on a new line.

It is a very common and useful practice to use several **`pt()`** and **`pf()`** calls together to track the progress of a value during model execution in order to make sure things are happening as expected.

In addition to print statements, the **`stop()`** command can be inserted into the logic to stop the simulation when it reaches a particular point in the logic. The simulation can be continued by clicking on the Step icon or the Run icon. The command **`time()`** will print the current simulation time.

The simulation can also be stopped by setting a stop time using the stop time field option located on the simulation control panel next to the simulation time indicator. With the stop time set, the simulation can be run at higher speeds until it reaches the point of interest.

Example

The following example illustrates several ways of how coding can be used to customize model behavior. The coding examples are used in several different types of triggers. While the logic might be applied using the dropdown menu, using modified FlexScript is a good way of understanding how custom logic can be used.

System Description

The system inspects two types of products, A and B. The times between arrivals are exponentially distributed with a mean of 10 minutes, but no inter-arrival time is more than 25 minutes. While the product mix is a bit uncertain, initially 75% are assumed to be type A. The buffer before inspection is limited to five products. If the buffer is full upon arrival, the product is diverted to another area for inspection, which is considerably more expensive. After inspection, good product As are sent to Department 1 and good product Bs are sent to Department 2. Products that do not pass inspection, which is about 10%, are sent to a rework area. Inspection times are normally distributed. The mean inspection times are 7 and 15 minutes, respectively for As and Bs. The standard deviation is one third of the mean value.

The objective of the model is to test the sensitivity of the system to changes in product mix, capacity of the buffer located prior to inspection, and mean inspection times. The primary performance measure is the percentage of products that must be diverted to a higher cost inspection area due to the limitation on the buffer size.

A screenshot of the *FlexSim* model of the system is shown in Figure 12.11. The circled numbers indicate where code snippets are used to implement the custom logic.

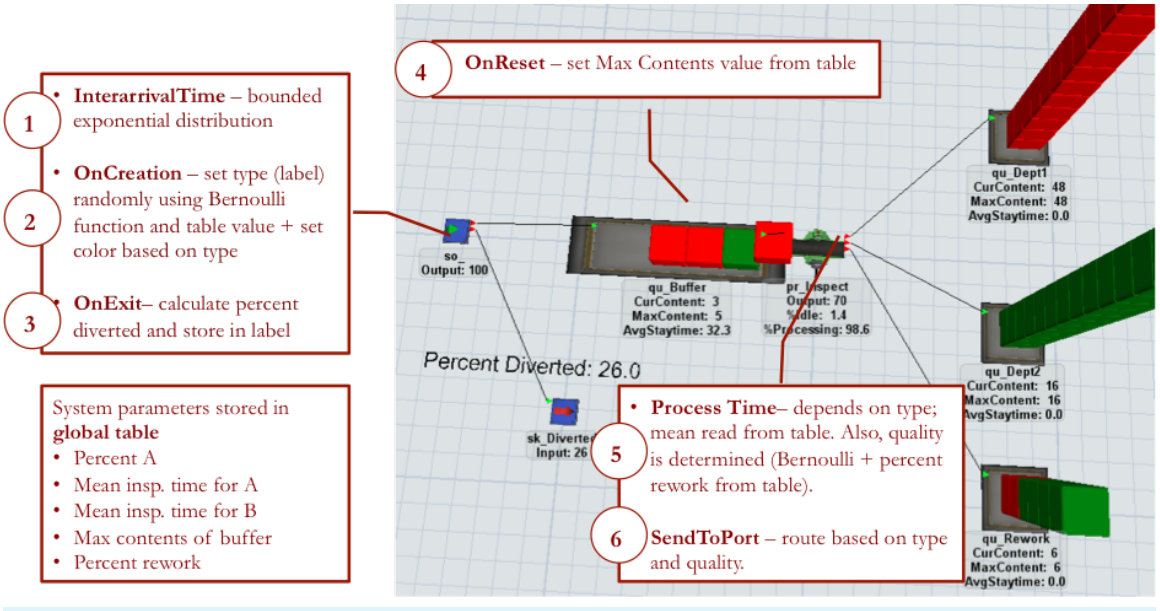


Figure 12.11: Processing system model.

Note that in the screenshot of the simulation, that 100 products have entered the system at this time and 26 have been diverted due to no space in the buffer, thus the performance measure is 26%. Also, a total of 70 products have made it through the inspection process – 48 red to Department 1, 16 green to Department 2, and 6 mixed color to Rework. The 6 rework is 8.6% of the total inspected; this is close to the expected 10% rework.

Also note, that the downstream departments are represented as queues. This is only to verify that products are routed properly. Once verified, the queues would be replaced by sinks.

In order to facilitate changing basic system parameters and enabling those variables to be varied in the Experimenter, store the values in a global table. For this example, the table, shown in Figure 12.12, is named Parameters and has five rows and a column; the row headers have been annotated for readability.

Model | Parameters

	Col 1
Percent A	75.00
Mean inspection time - A	7.00
Mean inspection time - B	15.00
Max Content of Buffer	5.00
Percent Rework	10.00

Figure 12.12: Global table of variables.

The following are explanations of how the custom coding is implemented in the triggers in the model.

1. **The time between arrivals is exponentially distributed with a mean of 10 minutes; however, the time between arrivals is never greater than 25 minutes. Print arrival times to the Output Console to verify.**

This change is implemented using the **Inter-Arrival Time** trigger in the source named **so_** and is shown in Figure 12.13. The custom code begins on Line 4; the prior statements are in the header and are not changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

so_ - Inter-Arrival Time*

```
1  /**Custom Code - Bounded distribution*/
2  Object current = ownerobject(c);
3
4  double IAT = exponential(0, 10, 11);
5
6  while (IAT > 25)
7  {
8      //pt("*** "); pf(IAT); pr();
9      IAT = exponential(0, 10, 11);
10 }
11
12 //pf(IAT); pr();
13
14 return IAT;
```

Header

Output Console

```
19.381664
6.526828
3.458095
2.512386
*** 50.424196
*** 47.088496
4.758149
3.764974
1.019899
7.723402
3.432714
15.312314
```

Figure 12.13: Source Inter-arrival time trigger.

Line 4 samples from the exponential distribution, using a statistical command, and stores the value in a local variable called IAT; its data type is double. The central part of the logic is the *while* control statement. As long as the sampled value IAT is above the threshold 25, it is resampled using the exponential distribution command. Also, the value is printed to the Output Console using several print commands on Line 8. In this case, as shown in the snippet from the Output Console, two values in a row were above 25; the third value was 4.758149 and thus is used as the inter-arrival time. Values that meet the threshold are printed without the *** notation as shown in Line 12.

Finally, the inter-arrival time, IAT, is returned to *FlexSim*. The return statement in Line 14 is needed since *FlexSim* is expecting a value from the code. It must have an inter-arrival time, a process time, port number, etc. However, in some code snippets a return is not needed, e.g. all of those in the Trigger tab. Actions in those triggers are optional so *FlexSim* is not expecting a returned value. Note that in general any code after a return statement will not be processed since processing stops once a return statement is executed.

The coding could be improved by storing two additional variables – mean inter-arrival-time and the threshold value – in a table, rather than within the code. This practice facilitates making changes to system parameters and including the variables as part of scenarios in the Experimenter.

An alternative approach to using the while construct would be to just set the high sampled value to the threshold value, in this case 25. To implement this approach, an if statement is used to test the value of the sample; and, if the sampled value is larger than 25, the value is set to 25. The result of using this approach is that it creates "bunching" at the tail of the distribution; i.e., the frequency would decline exponentially except at the threshold value where there would be a high frequency of inter-arrival times with a value of 25. In this example, that does not reflect the real system's behavior. One example where setting the high sampled value to the threshold would be appropriate is modeling a process, e.g. assembly, where an item is processed up to a point and then discarded. In this example, an operator may try to assemble some parts, but once the operator spends 25 minutes on the operation, the assembly attempt is stopped, the items involved are discarded, and the operator moves to the next item.

2. The random arrivals are composed of two types of products, A and B. 75% are As.

This change is implemented using the **OnCreation** trigger in the source named so_. The custom code, shown in Figure 12.14, begins on Line 7; the prior statements are in the header and are not

so_ - OnCreation

```
1 /**Custom Code - set type and color*/
2 Object current = ownerobject(c);
3 Object item = param(1);
4 int rownumber = param(2);  //row number of the schedule/seq
5
6 Table Parameters = reftable("Parameters");
7 double PercentA = Parameters[1][1];
8
9 item.Type = bernoulli(PercentA,1,2,10);
10 item.color = Color.byNumber(item.Type);
```

Header

Figure 12.14: OnCreation trigger code.

changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

The expected percentage of As is stored in row 1 and column 1 of the global table named Parameters. Line 6 sets up a reference to that table and Line 7 places the table value into a local variable named PercentA. (The data type of the local variable is double.)

Since there are only two types of products, the Bernoulli distribution is used to randomly generate two types of products. As shown in Line 7, the percentage of type As is stored in the local variable PercentA (obtained from a global table). This value is used in the Bernoulli statistical command in Line 9. Thus 75% of the flowtitems will be of type 1; the others will be of type 2. Also as shown in Line 9, the type is stored in the flowitem's's label named Type. The assignment is made using the dot notation, item.Type. The label was defined on the flowitem in the FlowitemBin.

Line 10 sets the flowitem's color (attribute item.color) based on its type (valued stored in the flowitem's label named Type). The value is obtained using the dot notation method Color.byNumber(item.type). The byNumber command sets the color to red if type is 1, green if type is 2, blue if type is 3, yellow if type is 4, etc.

3. Calculate the percentage of items diverted to another area because the buffer is full.

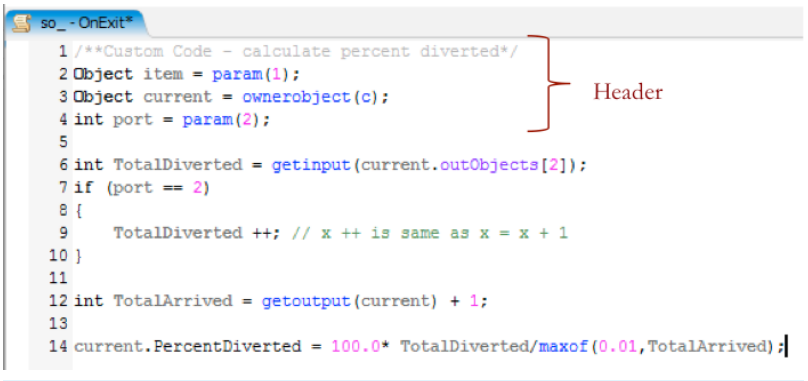


Figure 12.15: Items diverted calculation.

This change is shown in Figure 12.15 using the **OnExit** trigger in the source named **so_**. The custom code begins on Line 6; the prior statements are in the header and are not changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

If an item cannot join the inspection buffer because the buffer is full, it will be diverted out of the source through port 2 and sent to the sink connected to the port. Therefore, the total number diverted up until now (local variable named TotalDiverted) is the number of items entering the object connected to port 2, as shown in Line 6. This is accomplished through the use of the `getinput(...)` command and the method `current.outObjects[...]`.

If the item triggering this code is scheduled to go out port 2 then TotalDiverted is incremented by 1, as shown in Line 9.

The total number of products arriving (local variable named TotalArrived) is the number of items leaving the source, the current object, plus 1 for the current flowitem triggering the

code, as shown in Line 12. This is accomplished through the use of the **getoutput(...)** command.

In Line 14, the percent diverted is calculated as the total diverted divided by the total arrived, multiplied by 100. This value is stored in the source's label named PercentDiverted via the dot notation current.PercentDiverted. Note the use of the maxof command – this is used to prevent a division-by-zero error. It is good practice to use the maxof function in any denominator where it may be possible for the denominator to be zero.

4. **Set the buffer capacity based on the value in the Parameters table.**

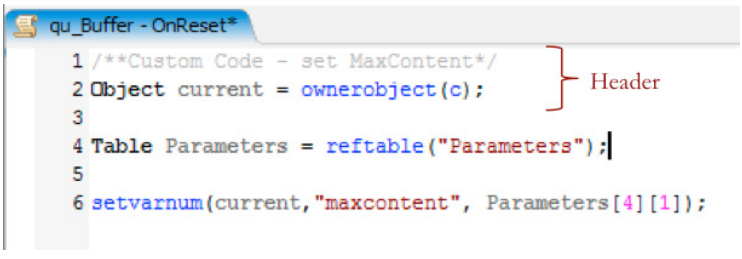


Figure 12.16: Setting buffer capacity.

This change is implemented using the **OnReset** trigger in the queue named **qu_Buffer**. This code is not triggered by a flowitem, but by pressing the Reset button. Alternatively, the code is fired each time the Experimenter starts a replication.

The custom code, shown in Figure 12.16, begins on Line 4; the prior statements are in the header and are not changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

The table Parameters is defined in Line 4 and used in Line 6. The object variable maxcontent for the queue object qu_Buffer is set to the value stored in row 4, column 1 of the global table Parameters using the **setvarnum(...)** command.

5. **Inspection times are normally distributed with a mean of 7 minutes for As and 15 minutes for Bs. The standard deviation is one-third of the mean time. The means are stored in the Parameters table.**

This change is shown in Figure 12.17 using the **Process Time** trigger in the processor named **pr_Inspect**. The custom code begins on Line 5; the prior statements are in the header and are not changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

The table Parameters is defined in Line 5 and used in Line 9 to set the mean of the process time.

A local integer variable Type is defined in Line 7. It is the product type of the flowitem that triggers the code. The type value is a label attribute of the flowitem, i.e., item.Type.

The variable Type is used to access the correct row in the Parameters table in order to obtain the correct mean processing time. Since the mean process times are stored in rows 2 and 3 of the table and the product types are either 1 or 2, the row value is specified as Type + 1. A better implementation might be to have a separate table for product data only. Note that if the table structure is changed, e.g. a row is inserted, then all of code snippets that use

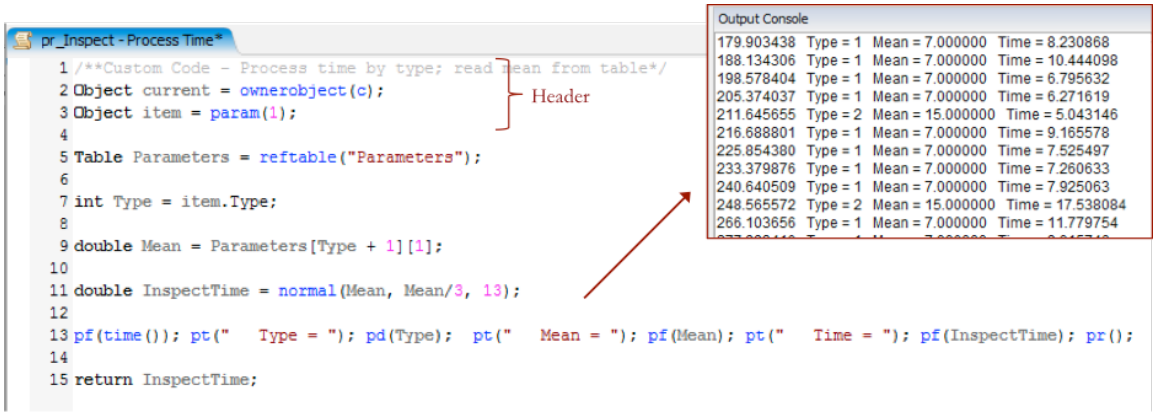


Figure 12.17: Setting inspection times.

the rows below the insertion row would have to be updated. Therefore, it is important to consider how tables are structured and to be used.

In Line 11 the inspection time is sampled from a normal distribution using a statistical command and stored in a local variable. The normal distribution uses the local variable Mean to specify both the mean and the standard deviation (Mean/3).

In order to verify that the code is working as expected, information on each process time is printed to the Output Console in Line 13. In this case, the current simulation time, product type, mean value, and the randomly-sampled process time are output. An excerpt of the output is show to the right in the figure above.

Finally, the process time is returned to *FlexSim* in Line 15, thus ending the code.

6. **10% of the items need to be reworked. Send the good As to Dept. 1, good Bs to Dept. 2, and all reworked items to Dept. 3. The percent rework value is stored in the Parameters table.**

This change involves determining the quality of the item and routing it to the correct port. The code could be implemented all in the **SendToPort** trigger in the processor named **pr_Inspect** or in both the **Process Time** and **SendToPort** triggers. The latter approach is used here since it may be more intuitive that the quality is determined as part of the process time. The routing decision must be made in the **SendToPort** trigger.

The code in the **Process Time** trigger in **pr_Inspect** is modified as shown below in lines 15 through 22. The comment in Line 1 is also changed to reflect the added action that is being performed in the Custom Code.

The table Parameters that was previously defined in Line 7 is used in Line 15 to set the percent rework. This is done through a local variable named PercentRework; it is assigned the value stored in row 5 and column 1 of the Parameters table.

In Line 17, the value of the local integer variable Quality is a random sample from a Bernoulli distribution with a PercentRework percent chance of having a value of 0 (bad) and a (100 – PercentRework) chance of having a value of 1 (good).

Figure 12.18: Setting rework.

Lines 20 through 22 are executed if the value of the local variable Quality is 0 (bad); i.e., the if-condition is true. The value of the label Quality does not need to be changed if the item is determined to be good since its default value is 1 (set when the item is created).

The routing of an item to either Department 1 or 2 if good and Rework if bad is implemented using the **SendToPort** trigger again in the **pr_Inspect** object. Both the flowitem's type and quality attributes determine the port from which the flowitem exits the processor. If either one of these variables, and not both, determine the port decision, then the flow logic could be specified via an option in the dropdown menu for the SendToPort logic. While multiple criteria can be used in dropdown menu options, coding is used here. The dropdown menu approach is provided at the end of this section.

The custom code begins on Line 5 of Figure 12.19; the prior statements are in the header and are not changed, except Line 1 is a user-entered comment that describes the function of the special code in the trigger.

Figure 12.19: Send to port logic.

A local integer variable ToPort is defined in Line 5. Its value is the product type of the flowitem that triggers the code. The value is obtained from a label attribute of the flowitem, i.e., item.Type. This will be the exit port if the product is good – product A to port 1 and product B to port 2.

Lines 8 through 10 are executed if the value of the item's label Quality is 0 (bad); i.e., the if-condition is true. These line change the value of the variable ToPort to 3.

Finally, the port number is returned to *FlexSim* in Line 12, thus ending the code.

The above coding in the SendToPort trigger for routing an item based two label values, Type and Quality, is not needed if implemented through the dropdown menu as shown in Figure 12.20.

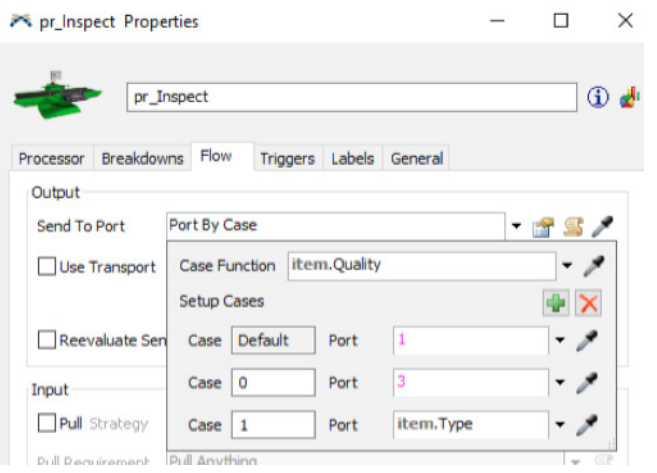


Figure 12.20: Using sendToPort dropdown choice.

In this case, the routing port number is determined in two steps. First, the value of the item's label Quality determines whether Case option 0 or 1 is executed. If Case 0 (Quality label = 0) is executed, then the flowitem will exit the processor from port 3. On the other hand, if Case 1 (Quality label = 1) is executed, then the flowitem will exit the processor from either port 1 or 2 depending on the value of its Type (value is either 1 or 2).

In general, it is best to use the dropdown menu options – there is no need to add the complexity of custom coding if it is not required.

References

1. Currie, C. and Cheng, R. (2013) "A Practical Introduction to Analysis of Simulation Output Data," *Proceedings of the 2013 Winter Simulation Conference*, R. Pasupathy, S. H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

2. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/t-table.htm

3. Knight, W., accessed at: http://www.math.unb.ca/ knight/utility/NormTble.htm

4. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

5. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, New York: John Wiley & Sons.

Exercise 12-1 Hilltop Steel Works

Background

Hilltop recently opened a new stainless steel rolling plant in Southern Alabama. The new facility includes a milling line that produces large specialty bolts for the offshore and ship-building industries in the area. Demand has exceeded expectations. The line has been operating at a target speed of 600 bolts/hour, but recent orders are calling for rates of 650 to 700/hour. As production has ramped up, the milling machines have become the bottleneck. The bottleneck backs up the forming equipment, which creates serious problems. Two options to help the situation are proposed:

1. Increase the size of the buffer area from the forming area to eliminate starting and stopping the forming line.
2. Use controls to change the machine speeds dynamically.

It was suggested that a simple simulation would help analyze the problem.

Problem statement

How can the plant respond to market demands?

Operating data

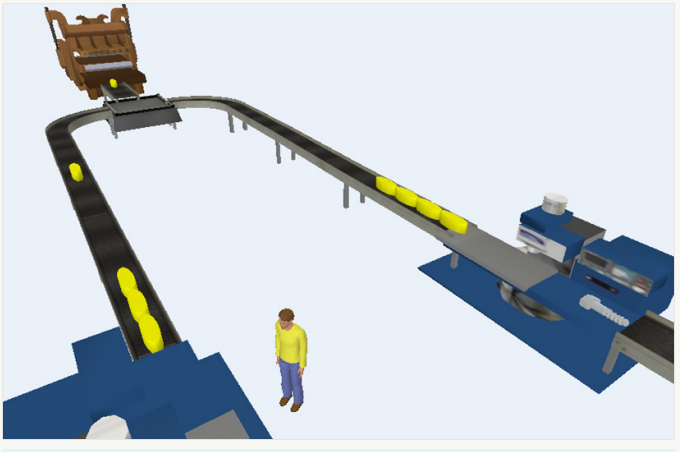


Figure 12.21: Hilltop milling macines.

As shown in Figure 12.21, the facility has two milling machines to fabricate the bolts. The rough formed bolts (blanks) come from the forming area to a buffer which sends them on to the milling machine, with the shortest queue having priority. The system was designed for a normal bolt production rate of 600 bolts/hour with the potential to increase that rate. Each machine is designed to run at 360 bolts/hour (10 sec. cycle time). Data has shown that the milling machines have a MTBF of 900 seconds, `exponential(0,900,1)`, and a MTTR of 200 seconds, `weibull(200,1,2,0)`.

The conveyors between the blank buffer and the milling machines have a capacity of 20 and a speed of 1ft/sec. Travel time is 30 seconds. The buffer itself has a capacity of 10. Additional conveyors carry the finished bolts away from the machines.

The machines are new and are driven by servo motors with adjustable speeds. Although designed for a cycle time of 9.5 seconds, the machines may be pushed faster to a cycle time of 7 seconds for short periods of time.

Expected results

Determine the impact of both proposals and recommend a course of action. Check results for the normal production rate and higher values.

Modeling and analysis issues

- What time unit would be appropriate? Be sure all time-related variables are constants.
- How can the speeds be changed easily? Can the values be changed on a dashboard?
- Before adding breakdowns, how can the simulation be validated to ensure that the correct number of parts are being produced?
- The cycle time of the machine will be dynamic; the processing time trigger is logic based, so where can the appropriate cycle time come from? Can a global variable be used as a flag to turn the dynamic speed change control on or off?
- Where can logic be placed to determine if a speed change is necessary? Consider what variable values are needed and what the conditions are to change speeds, both slow to fast and fast to slow.
- Use the experimenter to run replications of 40 hours.

Exercise 12-2 The Crafty Framer

Background

Your cousin has a great idea to sell customized picture frames and wants to get set up in time for the Christmas holiday season. The store is in a small location where customers can choose picture frames in all sizes and styles; however, the most profit will be made if customers choose to have the frame customized with specific decorations.

The customization process is straightforward and doesn't take much time, and the higher price can mean a greater profit margin. Since you mentioned having some knowledge of simulation, your cousin has asked you to simulate the shop operation and help decide how to best utilize the workers in the store. He currently has one checkout person and one custom framer but will hire another worker for the holiday rush.

Problem statement

Simulate the frame shop during a ten-hour period in order to help decide how to best utilize the three workers in the store.

Operating data

The frame shop operates from 9 a.m. to 7 p.m. At 7 p.m. the front door to the store locks. Customers in the store are serviced and the store is cleaned until 9 p.m. The procedure is repeated every day.

As shown in Figure 12.22, the store has two service counters—one used for customizing frames and one used as a check out area. The three workers can handle either the check out or the customizing station. During the holiday shopping period, customers are assumed to arrive at the shop at an average rate of 56 per hour; assume the times between arrivals are exponentially distributed.

Figure 12.22: Counters in the Crafty Framer.

There are three types of customers who enter at different rates:

1. Lookers, who just come inside without making a purchase, arrive at an average rate of 22/hr.
2. Base shoppers, who buy a frame without customization, arrive at an average rate of 20/hr.
3. Custom shoppers, who have their frames customized, arrive at an average rate of 14/hr.

Because of the size of the store and the local fire code, no more than 20 shoppers can be shopping at the same time; this does not count those in the checkout or custom lines.

Normally, customers spend about 15 minutes (exponentially distributed) looking around at the merchandise before they either decide to leave, purchase a plain frame, or have one customized. The customization takes about 5 minutes while the checkout can be accomplished in about 2 minutes. Assume all times are exponentially distributed.

Retail marketing data show that customers who see more than 4 people in the customization line will just purchase the plain frames instead. If more than 10 people are in the checkout line, the non-custom frame customers are likely to put their purchases down and walk out of the store.

Expected results

- Simulate 5 days of operation.
- Where should additional workers be used—cashier or custom desk?
- How many people were turned away because the store was full?
- How many settled for a standard frame because the custom line was too long?

Modeling and analysis issues

- How can the simulation be accomplished with lists instead of connections? Is there an advantage?
- How can you use the experimenter, choosing variables related to members of a team or members of a group, to change the number of people working or the number of checkouts or custom stations?
- How can you increase the number of people on the checkout or custom counter without having to add additional objects?
- What activities take place in the store?
- How can you stop the shoppers coming in when the store closes but keep the simulation running until everyone leaves? Consider what to use as a starting time in the simulation.
- How long should you run the simulation to represent 5 operating days?
- If you use pull logic from lists when should that logic be re-evaluated?

Chapter 12 Review Questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important-in the chapter.
2. Describe some of the important skills of an Advanced User.
3. Describe a hierarchical software architecture and discuss why is it important.
4. Describe the various types of nodes found in the FlexSim tree structure.
5. Describe the actions that take place between the time a flowitem enters a processor and when the process time is finished.
6. Give examples of the three types of Flexscript commands.
7. What symbol is needed at the end of each statement in Flexscript?
8. Explain why the variable “Carspeed” is not the same as “carspeed”.
9. Write the Boolean logic statement that tests to see if A is equal to B.
10. Write the command for referencing the object connected to the first center port of the current object using the dot syntax.
11. Write the command for obtaining the current content of the object connected to the second input port of the current object using the dot syntax.
12. Write the command for obtaining the total number of flowitems that have entered the object connected to the third output port of the current object using the dot syntax.
13. Write the commands to print the value of an integer and the current time to the output console.
14. What four steps are needed to create a global variable?

Chapter 13 Customizing Models

Most simulation languages have the capability to send information from one object to another in order to facilitate building complex logic. In *FlexSim* this communication is accomplished through messages that can be dynamically sent from any one point in the model to another. As simulations become more complex, additional ways of storing and dynamically displaying information become important. Additionally, enhancing the 3D representation of objects to better represent the actual process increases the acceptance and understanding of the simulation.

Objectives

To describe why communication between simulation elements is important; to use messaging between objects to carry out custom logic; to define additional ways of storing dynamic information while a simulation is running; to describe methods for enhancing the visual elements of a simulation.

Section 13-1 Communicating between objects

Communicating between objects with messages can be a convenient and efficient way of sending information to coordinate and control the simulation. A message can be sent from any trigger on any object in the model to any other object in the model at any time during the simulation. The OnMessage trigger (located in the Triggers tab) on each object executes when it receives a message.

There are two basic commands for sending messages from anywhere in the simulation:

```
sendmessage(to_object, from_object, param1, param2,
param3);
```

```
senddelayedmessage(to_object, delay_time, from_object,
param1, param2, param3)
```

Messages are either sent immediately or after a delay. The first command listed above sends a message when the object is executed in the simulation. The delay message command sends the message after a specified delay time.

The `sendmessage()` command sends the message immediately. This means that during the same calculation cycle, the message will be sent and the received message will be executed before the statement following the command is executed. At times such immediate response is required for synchronizing events that are very closely coupled.

There may be times, however, when the received message should be executed only after all other calculations are completed for the time cycle. In such cases, the correct command to use is `senddelayedmessage()` with a delay time of 0. A delay time of 0 will send the message at the end of the current calculation cycle of all objects. It is a good choice when an immediate message is desired but should wait until the states of all objects are updated.

Each message can contain up to three numeric parameters.

On the receiving side, the actions to be taken can be described by the choices in the drop down menu or through the script window; however, the script in the OnMessage trigger contains only a single line of code that defines the treenode variable current. To access the message parameters that may have come with the message, the command `msgparam(n)`, where n is the parameter number 1,2, or 3, must be added to the logic. This command can be used in an assignment statement, such as `double mp1 = msgparam(1);` or directly in the logic, such as `switch(msgparam(3))`.

Messaging examples

Messages expand the scope of logic and allow the simulation of events that would otherwise be difficult. They are often used in simulating events that have to be synchronized. Consider a just-in-time assembly area where products are assembled to order. As a part is being produced to meet a particular order in one manufacturing cell, a subassembly that will be added later has to be started in another cell, as shown in Figure 13.1.

In this case the first assembly unit sends a message at some point in the production cycle to the second unit building the subassembly. A variable containing the order number of the item being built is passed as a message parameter. Using the OnMessage trigger, the second unit looks up what is required and sets up its own parameters to create the required subassembly.

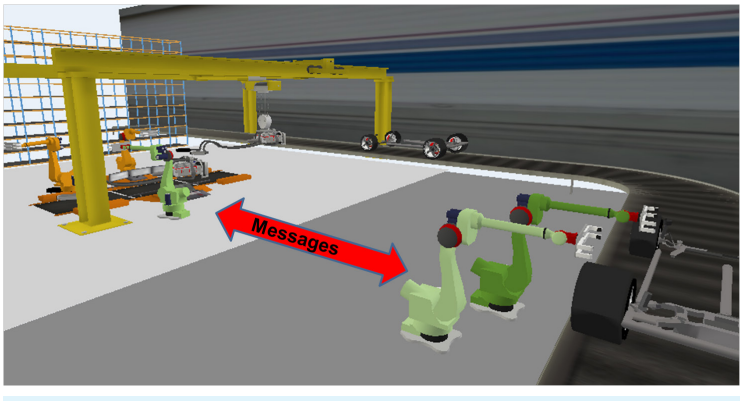


Figure 13.1: Manufacturing Cell.

Another example of the use of messages involves a machine that, when started from a stopped condition, goes through a speed ramp-up

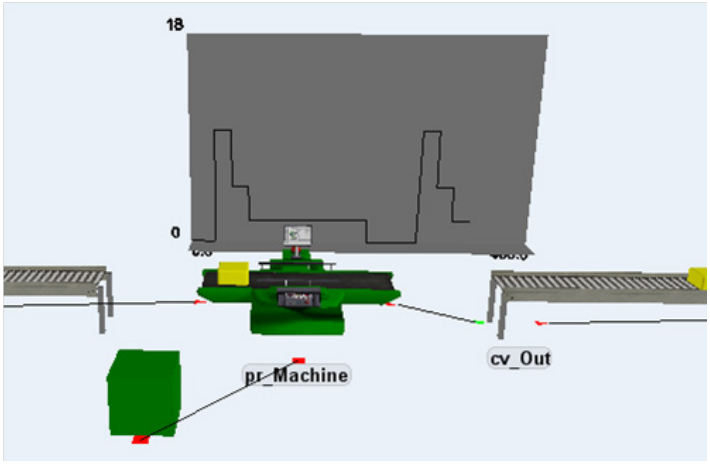


Figure 13.2: Machine Startup Control.

phase. In this case, messages are sent within the same object (the machine) and between objects (a controller and the machine).

The controller, represented in Figure 13.2 as a cube (created using a visual tool), uses its triggers to send start and stop messages to the machine. Whenever the machine receives a start-up message, it begins ramping up its speed as seen in the line graph in the background of the figure which shows the machine's speed over time.

In this example, as shown in the timeline in Figure 13.3, the controller is responsible for turning the machine on and off and, after the stop time, starts the machine using a startup ladder that increases speed from 10 seconds per cycle to 2 seconds per cycle. The machine is scheduled to start 30 minutes after the simulation starts. In order to accomplish this, the controller sends a delayed message to itself through its OnReset trigger. After 30 minutes of simulation, the controller sends a message to the machine to start the startup ramp and a delayed message to itself to stop the machine after 180 minutes.

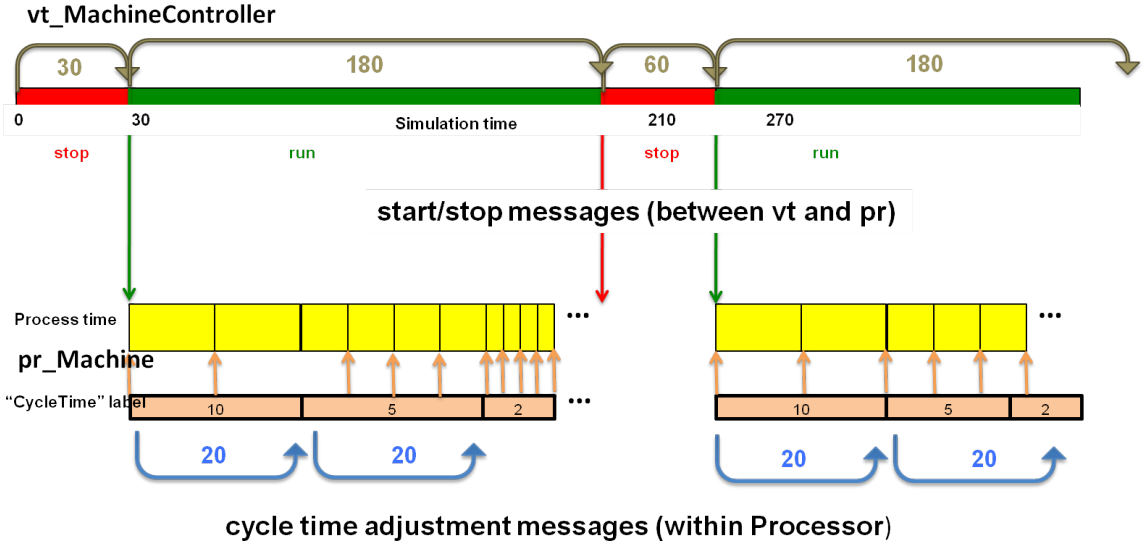


Figure 13.3: Control Timeline.

After three hours of simulated time elapses, the controller receives its message and sends a message to the machine to stop; the controller also sends a message to itself to restart the machine in one hour.

As shown in the code segment in Figure 13.4, this process continues throughout the simulation.

Note that the machine (processor object) also uses messages—in this case to represent the ramping up of the machine's speed. The processor sends a series of messages to itself to change the value of its CycleTime label as the ramp-up time progresses. This is accomplished via the same case construct in the code segment shown in Figure 13.5.

In this example, the cycle time is 10 minutes for the first 20 minutes after startup, then drops to 5 minutes during the next 20 minutes after startup, and then reaches the normal cycle time of 2 minutes. The machine uses a cycle time of 2 minutes until the next stoppage and restart. Note that on the processor's general tab the process time is set through a label called CycleTime.

The third message parameter is used to track the startup phase. Note that the starting and stopping of the machine is controlled by opening and closing, respectively, the processor's input ports. The **stopobject()** and **resumeobject()** commands could not be

```
/**Control Line*/
Object current = ownerobject(c);
{ // ************* PickOption Start ************* //

int step = msgparam(3);


switch (step)
{
    case 1:   //start the machine
    {
      colorgreen(current);
      senddelayedmessage(model().find("Processor1"),0,current,0,0,1);
      senddelayedmessage(current,180,current,0,0,2);
      break;
    }

    case 2:  // stop the machine
    {
       colorred(current);
       senddelayedmessage(model().find("Processor1"), 0, current, 0,0,4);
       senddelayedmessage(current,60,current, 0,0,3);
       break;
    }

    case 3:  // restart the machine and keep cycle going
    {
       colorgreen(current);
       senddelayedmessage(model().find("Processor1"),0,current,0,0,1);
       senddelayedmessage(current, 180, current, 0,0,2);
       break;
    }

    default: stop();

}
```

Figure 13.4: Controller Logic.

```
/**MachineStartUpControls*/
treenode current = ownerobject(c);


int step = msgparam(3);

switch(step)
{
    case 1:  // Start machine up for first sstartup cycle
    openinput(current);
    colorgreen(current);
    current.ProcessTime = 10;
    senddelayedmessage(current,20,current,0,0,2);

    break;

    case 2:    // second startup cycle
    current.ProcessTime = 5;
    senddelayedmessage(current,20,current,0,0,3);
    break;

    case 3: // at final speep
    current.ProcessTime = 2;

    break;

    case 4:    // stop machine
    closeinput(current);
    colorred(current);
    break;

    default:stop();break; // stop because something is wrong in logic
}

```

Figure 13.5: Processor Logic.

used, because if an object is stopped, it cannot, among other things, send and receive messages.

The effect of this ramp-up process on flow items serviced by the processor is illustrated in Figure 13.6 using a Gantt. For each flowitem, the leftmost or blue horizontal bar is the time the item spends on the input conveyor; the center or green bar is the processing time of the machine; and the rightmost or red bar is the time the item spends on the output conveyor.

Notice the intended decrease in processing time over time due to ramp-up; also, the time on the input conveyor decreases due to the machine becoming available and the processing time speeding up.

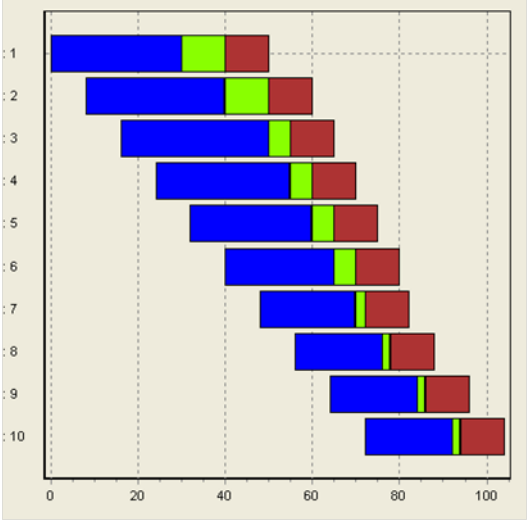


Figure 13.6: Flow item processing time.

Section 13-2 Label Tables

The Intermediate User utilized labels to hold individual pieces of information; however, labels can also be used to hold arrays of data. Such a label is called a label-table and is useful in the more advanced methods of managing where data can be stored in the simulation.

A label-table is created using the Label tab on the object and selecting to (1) add either a Number Label or a Text Label; (2) when the label is selected, click on the selection "Label Table" in the lower right corner of the tab as shown in Figure 13.7; (3) enter the row and columns and select "SetSize."

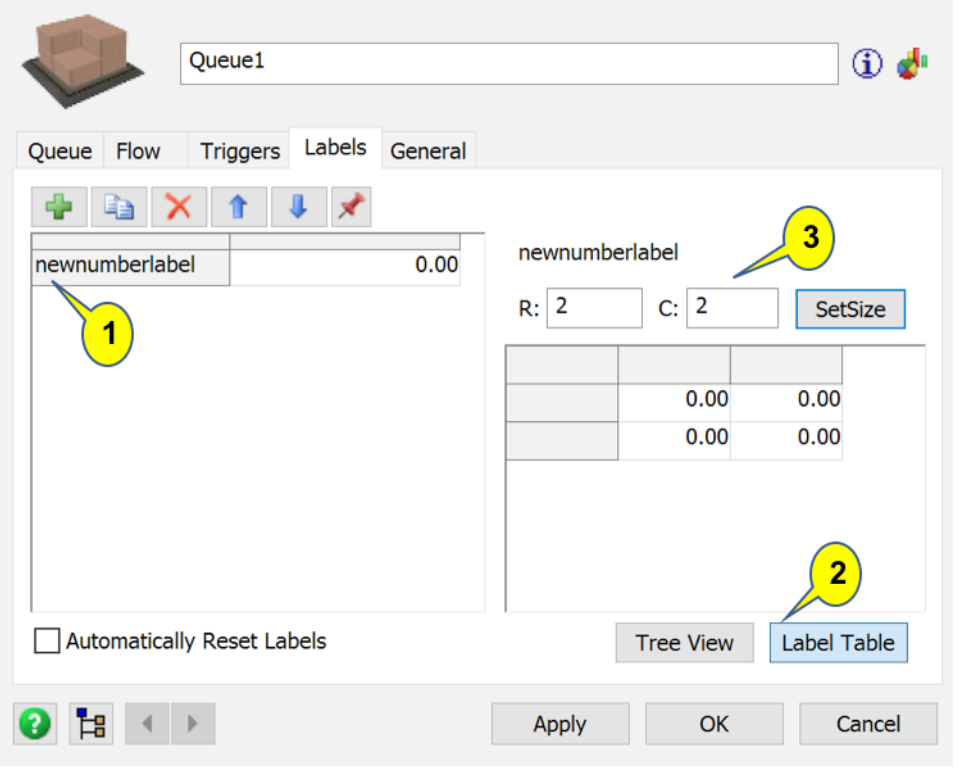


Figure 13.7: LabelTable Creation.

Addressing the label-table is similar to addressing a global table using commands such as

```
gettablenum() AND settablenum();
```

To obtain a value from a label- table that exists on the currently selected object from one of its own triggers, use the command

```
gettablenum(label(current,"labelname"),row,column);
```

If the label-table is on a separate item, then item would be used in the place of **`current`**.

To get the value of a label-table that is on an item that is the first or last item in a queue or a combiner, use the command

```
gettablenum(label(first(current),"labelname"),row,
column);
```

The label-table is useful when related data are required dynamically. An example would be a distributor that makes up boxes based on a real time order entry system. The number of items in a box can vary for each order. In the simulation shown in Figure 13.8, bottles from four entry lines are combined to meet custom orders.

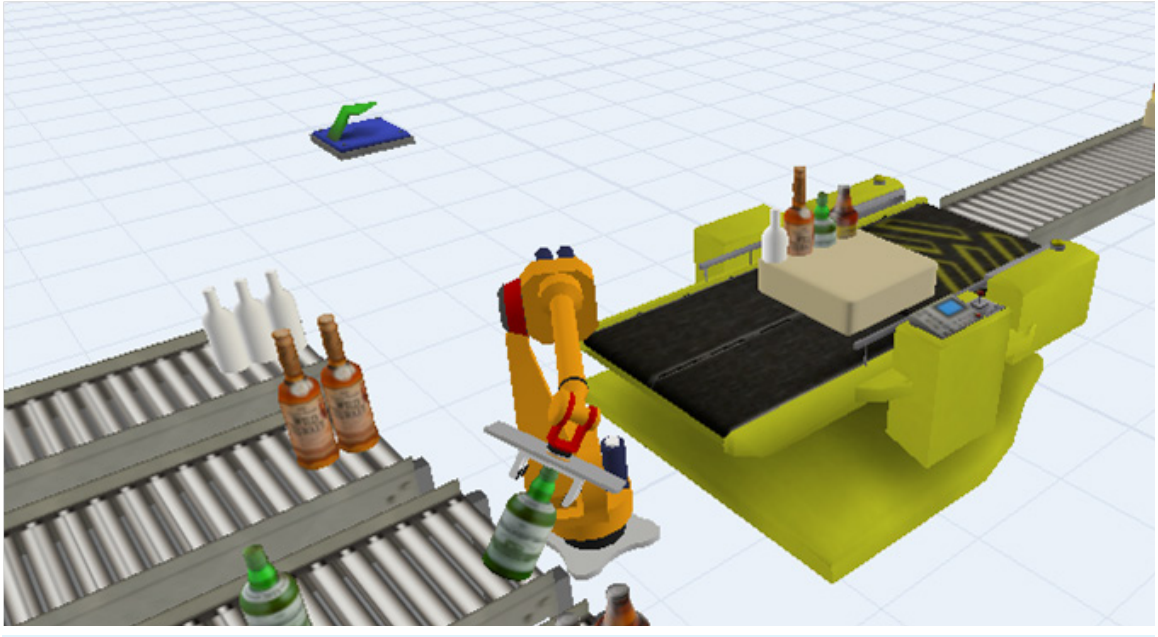


Figure 13.8: Custom Order Simulation.

A numeric label-table called "Wines" is created on the flowitem that represents the box being filled. The table has 4 rows and 1 column. The box source, connecting to the combiner, fills out the label-table on the flow item with the number of bottles needed from each entry line to represent the current order.

In the OnEntry trigger of the combiner, the standard code for updating the combiner's component list is modified to use the label-table instead of a global table as was done in previous chapter exercises. The resulting code is shown in Figure 13.9.

```
treenode thelist = getvarnode(current, "componenlist");
treenode thesum = getvarnode(current, "targetcomponentsum")'
setnodenum(thesum, 0);

for(int index = 1; index <= nrows(thelist); index++)
{
  setnodenum(cellrowcolumn(thelist, index,1);
  gettablenum(label(item, "Wines"), "index, 1));
  inc( thesum, gettablenum(label(item, "Wines"), index, 1)):
}
```

Figure 13.9: Combiner logic.

Section 13-3 Customizing object placements

The default graphics for objects in *FlexSim* are normally sufficient for visualization; however, there are times when changing the placement of flowitems and the visual representation of the object itself would benefit the simulation. For example, the processor by default moves flowitems across the top of the 3D object; however, if the processor represents a bank teller behind a counter and the flowitem is a person being served, then the default visualization would not be representative of the system and may cause confusion for a layperson.

Changing the visualization may involve both the object and the flowitem. The following example converts a processor's graphic representation from a machine to a service desk.

As shown in Figure 13.10, change the graphic by using the directions as follows:

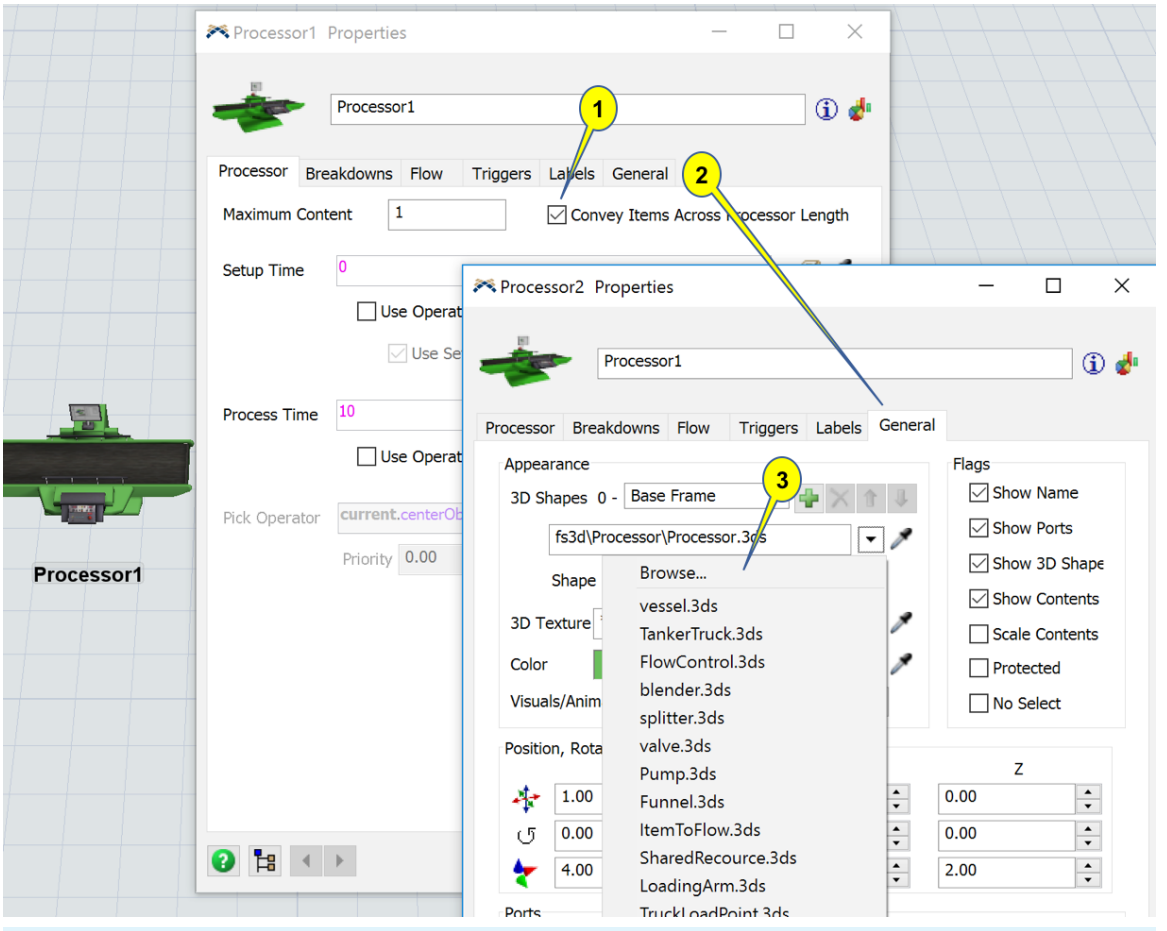


Figure 13.10: Changing an object's visual.

1. Open the processor and uncheck "Convey Items Across Processor Length."
2. Select the General tab and click on the dropdown menu.
3. Use the Browse selection to change the processor to some other shape —for example, a box. A box is in the *FlexSim*/fs3d/general folder. You

may have to navigate to the folder or select some other 3D shape from an external source such as Google 3d Warehouse (.skp).

Modify the size on the General tab as desired.

If a person is used as a flow item comes to the service desk simulated by the processor described above, that person may appear in the middle of the desk as shown in Figure 13.11. The following steps will set the person (flowitem) to the desired location as illustrated in Figure 13.11:

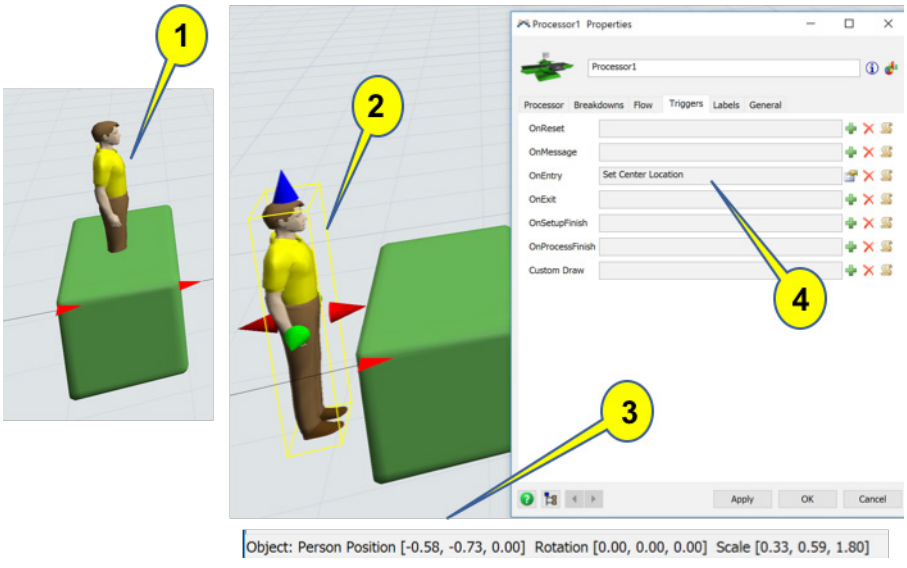


Figure 13.11: Changing the position of a flowitem.

1. Use a source to supply the processor and select a person as the flowitem. Start the simulation and stop when the person is in the processor shape.
2. Move the person with the mouse, or use the General tab on the flowitem.
3. When the position is correct, note the location at the bottom of the screen when the flowitem is highlighted. The numbers represent the positon relative to the processor.
4. Use those position values in a visual/setlocation command in the OnEntry trigger of the processor.

In a similar way, a task executer, such as an operator, can be shown to push a flowitem rather than carry it. Fox example, an operator can be shown to push a wheelbarrow rather than carrying it. For this example, a flowitem in the flowitem bin was created using a file from the 3D wharehouse.

The steps are outlined below and illustrated in Figure 13.12:

1. Run the simulation until the operator is carrying the flowitem.

2. Move the flowitem with the mouse, or use the General tab on the flowitem. When the position is correct, note the location at the bottom of the screen when the flowitem is highlighted. Use these position values in the set location command in the load trigger of the operator. You might want to also change the rotation of the object: in this case, the X and Z axis were rotation by 12 and 90 degrees to head the wheelbarrow in the right direction and tip the handles up.

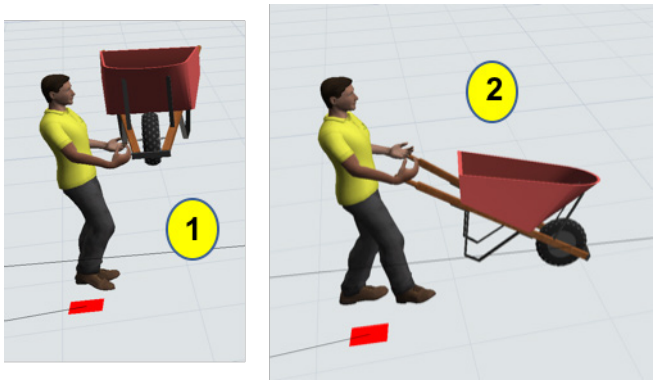


Figure 13.12: Changing the item's position.

Exercise 13-1 Fister's Express

Background

Fister's Frozen Foods, introduced in Chapter 4, wants to offer quick-delivery frozen food combinations that will provide a week's worth of ready-to-prepare meals. Customers will be able to choose a combination of meats, fish, poultry, vegetables, and deserts. Fister's guarantees that the order will be filled and shipped the same day.

To provide the most efficient service, the orders will be assembled automatically and moved to the existing shipping area. Control of the automated system is being left to an engineering contractor.

Operating data

Marketing expects orders initially to arrive at an average rate of 5 orders per hour (exponentially distributed) but that the order rate could increase to one order every 9 minutes, on the average. Customers can pick a combination that contains choices from five categories. For each of the seven days, a customer must pick a meat, chicken, or fish for the main ingredient each day. They will next pick if they want vegetables and a dessert with their meal. Consider the choices to be uniformly distributed.

Each category is stored in a separate aisle in the frozen food warehouse. Each aisle has its own robotic picker. The pickers require a base travel time to reach the first item and then an additional amount of time to collect the other items in the order.

Total travel times are:

Category	Base Time	Additional Mean	Additional Std. Dev.
Meat	0.5	0.6	0.3
Chicken	0.7	0.4	0.2
Fish	0.85	0.8	0.4
Veggies	0.6	0.5	0.3
Desserts	0.5	0.3	0.15

Note: All times are in minutes. The base time includes travel to and from the area of the first item. The additional time is normally distributed.

As orders arrive, they are printed on a card that is attached to a separate pallet. The pallets are maintained in a queue as they are received. When an order is ready to be processed, a pallet is placed at a holding station. When the pallet enters the holding station, the order amounts are transmitted to the picking robots for each food category.

Each picking stations places the food into a box and sends it to the assembly station where the order is placed on the pallet. The assembly task requires 15 seconds

(exponentially distributed). The full pallet then travels to the shipping room and the next pallet is released from the holding station.

Conveyor travel times are:

Conveyor	Travel time (minutes)
Meat pick station to assembly	1
Chicken pick station to assembly	0.8
Fish pick station to assembly	0.8
Veggies pick station to assembly	1
Dessert pick station to assembly	0.8
Pallet holding to assembly	2
Assembly to shipping	2

Expected results

Build the model, and run the simulation for 7 days (24 hour operation) and comment on such performance metrics as:

- How long does it take to complete an order?
- How many orders are completed?
- How many orders are backlogged?
- What order rate can be reasonably achieved?

Modeling and analysis issues

- What level of detail is needed? What time unit is used?
- Where will the order details be placed as they come in?
- How will the conveyor travel times be implemented?
- How should the data be managed? What tables are needed? Could a label-table be used?
- What information should be sent to each picking robot? Where should it be sent from?
- Do you need to model each picked food item or just the box they are put in?
- What are the initial conditions for each piece of equipment?
- How will the order time in the system be calculated?
- Where can the logic be set to let the assembly station know which food categories to load?
- What visualizations can be used?

When dealing with multiple lines of logic it may be useful to print out the logic to review and troubleshoot.

Exercise 13-2 Peoples Surgery Center

Background

Given the changes in medical care as directed by the government, Carson Memorial Hospital has decided to open a surgery center for common out-patient surgical procedures. It is felt that such a center could be operated more efficiently and cost effectively than what is currently offered. The hospital was originally designed for major procedures and longer-term care.

Before setting up the new facility, the hospital administration wants to have a simulation developed to check on the staffing levels that are needed and the amount of time patients would spend in the facility–especially in the admission and lab areas.

Problem statement

Provide a simulation to help establish operating conditions for the admissions and lab areas of the surgery center.

Operating data

Three types of patients are anticipated to use the new facility:

1. Those who arrive for the surgical appointment with all the required laboratory work completed.
2. Those who have appointments but need laboratory work before beginning the surgery.
3. Those who are using the lab facility for routine tests or pre-screening tests for a later surgery.

Patients who arrive at the facility first check-in with an attendant and then enter a waiting room and then wait until called to continue.

After admissions, patients who are called for their surgery continue to a small waiting area to wait to be taken to the surgical rooms. From there they must be accompanied by an orderly to the surgery area.

Patients who need lab work can walk to the lab area when called from the main waiting roon. Once at the lab, they register at the lab admitting area (one lab agent), and then wait in the lab waiting room until they are called by a lab technician. Patients only there for lab work can then go home when lab procedures are complete. Patients continuing for surgery (Type 2) return to the main waiting area where they are called for surgery admittance.

Admissions operating hours:

- Opens at 7 a.m. for patients with surgery appointments (Type 1 and 2).
- Patients for lab work arrive between 10 a.m. and 3 p.m.
- No new patients are allowed to register after 3 p.m.

- The admissions/lab areas are available until all patients who entered from 7 a.m. to 3 p.m. are served. It is planned that these areas will be clear of patients by 7 p.m.

On normal days, the mean time between arrivals for check-in is 10 minutes, distributed exponentially. However, the time between arrivals could decrease to 8 or even 5 minutes. Admitting rates, the percent of each patient type, and process times are provided In the table. A second table provides travel and lab times. All times are in minutes.

Patient Type	Percent before 10	Percent after 10	Admission mean time	Distribution
1	85	40	40	Normal, S.D. = 3.5
2 before lab	15	10	10	Normal, S.D. = 4
2 after lab work			40	Normal, S.D. = 3
3	0	50	15	Normal, S.D. = 2

Travel Times

Routes	**Distribution**
Admissions to Surgery	Uniform (min = 3, max = 8)
Between Admissions and Lab	Uniform (min = 2 , max = 5)
Surgery to Admissions (orderly)	Constant = 3

Lab Times

Procedure	**Distribution**
Lab Admissions	Triangular: min = 2, max = 5, mode = 3.3
Lab sample	Triangular: min = 6, max = 12, mode = 5.1

Current staffing levels are:

- 1 Admissions clerk
- 2 Orderlies
- 1 Lab clerk
- 1 Lab technician

Expected results

- Run the simulation for 20 days and comment on such performance measures as
  - The average wait time for each waiting area.

  - How many patients per day can the facility accommodate? Will patients still be in the area after 7 p.m.?
  - How long are patients in the simulation?
  - What is the impact of different staff levels or other changes?

Modeling and analysis issues

- How can an entire system be modeled in sections to test it before putting it all together?
- Entering the simulation
  - What are different ways to bring the patients into the system? How can you regulate the times the center allows patients to enter?
  - How can you identify the patient type and flag the type 2 patients who have come back from their lab work?
  - What logic is needed to send the patients to the right place?
  - How can you keep track of each type of patient and the total number of patients?
  - How will you be able to calculate how long each patient is in the simulation?
- Moving patients to surgery
- How can the orderlies be shared?
- How can the travel times be set up to be different in each direction?
- When an orderly takes a patient to the surgery area, consider how to make sure the orderly returns to the surgery waiting area.
- How will patients move to the lab at the appropriate speed?
- Patients coming into the lab waiting room are either waiting for a receptionist or a lab technician. How do you identify them and make sure they go to the right place?
- How will only the patients going for a lab sample have a technician take them?
- What variable on a flowitem can be used to track a patient's time in the system?
- In terms of visualization, how can the orderly or technician walk side by side with the patient?
- What are the startup and finish times for the simulation runs?
- How can you tell if everyone is finished at the end of the day?
- How would you add staffing?

Chapter 13 Review Questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the importance of communication between objects.
3. Describe two uses for communicating among objects in a simulation.
4. Describe the difference between sending a message immediately and sending it after a delay. Provide an example where each might be used.
5. Describe two examples where you might send a message from an object to itself.
6. Describe the importance of label-tables. Provide an example where one might be used.
7. If you change the graphical image of a processor and want to place the entering flowitem in the proper location, write the command and indicate where it should be placed.

Chapter 14 Advanced Logic Functionality and Representation

Objective

Develop simulation logic that is described, created, and visualized through a flow of logical statements that carry out logical behavior in a simulation.

Complexity in simulations usually involves additional levels of logic to accurately describe a physical environment. The previous two chapters described how an advanced user might go about building a more complex simulation using custom scripting and other techniques. The stated goal of many software applications is to allow all users to add required logic with standard choices or to add logic without having to write any custom logical descriptions or what is called "code." Both goals are simplistic.

First, simulations created to aid in understanding new concepts, or to achieve a design that is unique, are not likely to be accomplished using some standard set of elements. To make advancements, some degree of customization is usually needed.

Second, logic in inherently mathematic in nature and always must be specifically stated. It doesn't matter if the logic is called scripting, coding, of choosing—logic must be created to fit the situation. The only issue is how easy it is to make the logical statements in a simulation.

Often the logic needed in a simulation involves decision making that has no physical counterpart to display on the screen. It may also involve an activity that impacts the simulation but takes place away from the physical boundaries of the simulation.

FlexSim already has an extensive variety of standard, logical functions that can be selected using drop down menus. While the combination of standard functions and scripting can handle almost any situation, the use of scripting can be difficult and time consuming. To address this, *FlexSim* has introduced a new programming paradigm, called Process Flow, that makes it easier and more efficient to include advanced logic. Process Flow is a higher-level programming method utilizing visual logical objects along with additional functionality.

This chapter introduces the basics of the Process Flow environment and examples of how it can be effectively used. More detail is available in the *FlexSim* User Manual. The tutorials in the manual go into a lot more detail and indicate that entire simulations could be built with the Process Flow constructs alone. However, from a practical perspective, the 3D visualization of physical elements remains the best way for obtaining confidence and understanding of simulation results in discussions with management and/or non-technical individuals.

Section 14-1 Basic concepts of Process Flow elements

Process Flow logic is developed on a tabbed Process Flow Pane that is opened from the main tool bar by clicking the Process Flow button. A dropdown menu offers a choice of four options. The General choice will be used in this book. When the pane appears, and is activated by clicking in it, a library of Process Flow objects appears on the left side of the screen as shown in Figure 14.1. Objects are moved to the Process Flow pane in the same way as 3D objects. When an object is selected in the Process Flow pane, settings appear in the Quick Properties pane on the right side of the screen.

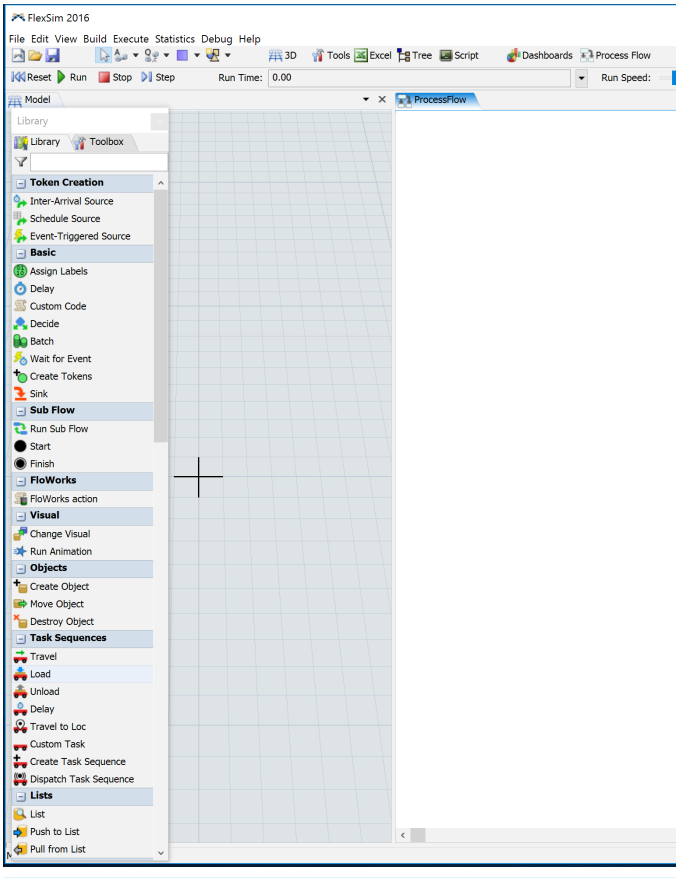


Figure 14.1: Process Flow Pane and Library.

The most basic object in Process Flow is the Token (). It operates in a similar manner as a flow item in the 3D window. The movement of the token visibly traces the logic being executed. When created, the token is given a unique ID number that can be used as a reference point. More than one token may be in a process flow so they can be named if desired to better identify their role. Most importantly, they may have labels assigned that carry data through the process flow. The data may represent variables that are used throughout the logic.

Navigating in the process flow pane is like navigating inside the model view pane. Elements can be moved around and the view can be zoomed. The entire pane can be moved around the simulation space by clicking on the tab and dragging it.

A process flow starts whenever a token enters a logical element and ends when the token reaches a Sink object. The same commands of Reset and Stop apply to the process flow as well as the model. The token is visible as it goes through the logic elements. Using the step command can be used to visualize logic that occurs during one clock tick. A simulation may contain more than one process flow logic path and each path may contain sub flows like subroutines in a coding environment.

Section 14-2 Defining characteristics of Process Flow elements

An **Activity** is a logical operation or step in a Process Flow. The first set of objects in the library pane create tokens. The first two, inter-arrival and schedule, operate in the same manner as their source counterpart in the 3D model. The third is unique, however as it will create a token when an event occurs in the 3D model. The event is selected by clicking the eye dropper icon and moving it over an object in the 3D model. Clicking on the object shows a list of events that can be selected.

The Basic activities are the ones mainly used. They are used to assign labels to tokens or any object or to delay a token for a time period. The custom code object allows FlexScipt code to be added for calculations or decisions. A decide activity is used to route tokens to the next activity. Rather than be delayed by a time period, a token can be delayed until an event occurs in the 3D model with the Wait for Event activity. This activity is very flexible as it can also be used to modify data on 3D objects. Additional activities can create tokens or destroy them. Changes are made to activities through the properties pane on the right side that is activated when the activity is active or by double clicking on the activity itself.

The visual, objects, and task sequences groups of activities directly impact a 3D model dynamically. The shared resource and lists activities are primarily used to manage items in the simulation. The list activity was discussed in Chapter 6.

Section 14-3 Building a Process Flow

The best way to understand when, and how, a Process Flow can be used is through an example. Figure 14.2 shows a simple model containing a source, queue, processor, conveyor, and sink. The source produces a flowitem every 3 minutes. The processor normally runs at

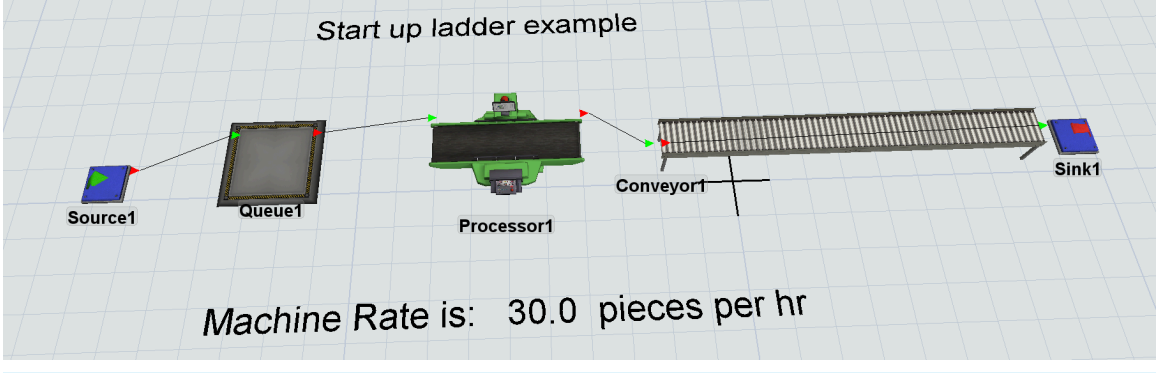


Figure 14.2: Machine startup simulation.

2 minutes per cycle (30 objects per hour). The process has a failure every 200 minutes with an associated down time of 20 minutes. The first failure occurs at 50 minutes.

When the processor starts up after a failure, it ramps up its speed in four increments of 5 minutes each. The percent of full speed at the start of each increment is:

Increment	Percent of full speed
1	25
2	50
3	75
4	100

This table is saved as a Global Table for ease of changing and later use. The variables in this example are all constants for simplicity.

Simulating a startup ramp was first discussed in Chapter 13. Using the standard *FlexSim* logic there are two places where logic must be placed – the processor and its MTBF table. Two labels are placed on the processor: one (ProcessTime) holds the current cycle time and the other (OldSpeed) is used to remember the standard cycle time when the machine is running at full speed. On the parameters screen of the processor, the Process Time is specified by writing the command to get the value of the label holding the current cycle time that was created:

```
Current.ProcessTime;
```

The logic for the MBTF table is shown in Figure 14.3.

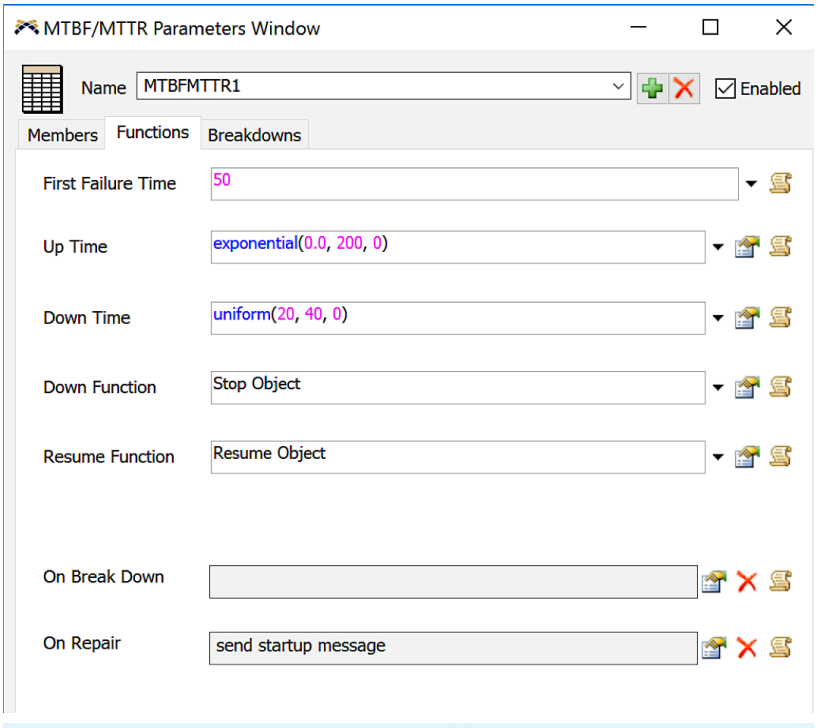


Figure 14.3: MBTF logic.

In the OnRepair trigger, which fires when the down time is finished, a message is sent to the processor with the command:

```
sendmessage(involved,0,current,1,0,0);
```

The value of 1 as the first message parameter indicates the first step in the ladder.

On the Triggers tab for the processor, logic is added to the OnMessage trigger to carry out the startup logic. The logic gets the step number from the message parameter. The new process time is then written to the label ProcessTime which will be used by the processor.

Finally, if all four steps have not been completed, the step number is increased by one and a delayed message is sent to the processor with the new step number as the first parameter of the message.

The code could look like:

```
int step = msgparam(1);

double newcycletime = current.OldSpeed / 100;
gettablenum("StartUpLadder",step,1);

current.ProcessTime = newcycletime;

if(step < 4)

    {
    step = step + 1);
    senddelayedmessage(current,5,current,step,0,0);
    int step = token.Step;
    double multiplier = 100 / gettablenum(
          "StartUpLadder",step,1);
    double newcycle = token.OldSpeed * multiplier;
    return newcycle;
    }
```

Notice that the code is written in more than one place and requires several lines. A good time to consider process flow logic elements.

For the Process Flow elements, the logic will follow a similar pattern to the previous startup ladder logic. A label on the process called ProcessTime will hold the current value of the processor cycle. Also, a global table called StartUpLadder will be used to hold the percent increase for each cycle. Next, create a space for the logic by opening a new general Process Flow window. Clicking in that window brings up the library on the left side of the screen.

Since the ladder should start when the processor resumes after a down time, begin by moving an Event Triggered Source activity to the process flow area that will create a token when the event occurs. Next use an Assign Labels activity to set labels called "OldSpeed" and "Step" on the token. The next assign labels block will be used to calculate the new cycle time and set it on the processor.

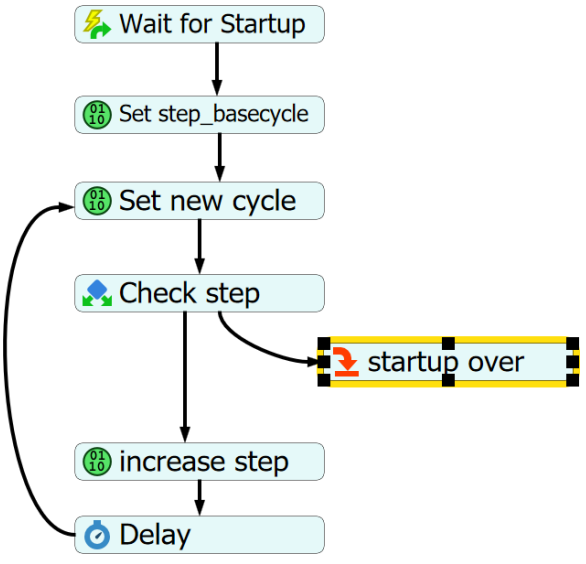


Figure 14.4: Process Flow Blocks.

direction the logical calculations will take.

Setting activity parameters starts with the Wait for Startup block. As shown in Figure 14.5. By using the eyedropper ion next to the Object field (1), then going to the 3D model screen and clicking on the process and selecting the Resume state (2), the triggering event will be set into the activity block.

The next assign labels activity is used to set label values on the token that can be used during the remainder of the startup ladder. A label is created called "Step" which is given a value of 1 since this will be the first activity after the token is created. The second label is called "OldSpeed" which gets its value from the value of the process label and still contains the value just before the process went down.

To set the OldSpeed label, as shown in Figure 14.6, the eyedropper next to the value section for the label is used to choose the processor. The pop up menu shows the label on the processor called ProcessTime. This will the current cycle time when the process goes down.

Once the new cycle is set, a check will be made using a decision block to see if the ladder is finished. If more steps are required, the next assign labels block will update the step label on the token and, after a delay, the token will be sent back to calculate another cycle value.

Clicking on a block or its icon on the left side of the block brings up the parameters for the activity. Names should be given to each block to make their purpose clear. The blocks are connected by clicking on the edge of a block until a chain appears and then dragging to the next block as shown in Figure 14.4. The arrows show the

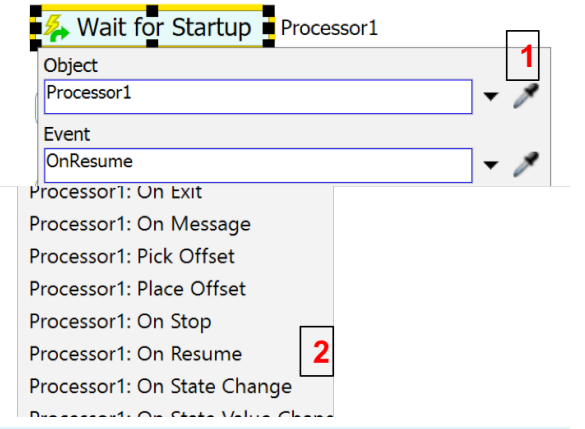


Figure 14.5: Assigning the Event.

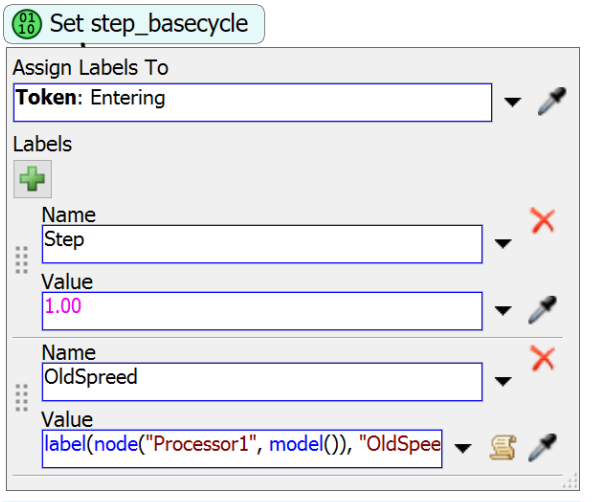


Figure 14.6: Assign labels to token.

The cycle times for the startup ladder can now be calculated. The logical statements are like the ones used previously. Again, an assign label activity is used. However, the label to be changed is not associated with the process flow but appears on the processor in the 3D model.

The assignment activity is shown in Figure 14.7. In this case, the eye dropper is used to select the processor on the 3D model and the label to be changed is listed the same as the label on the processor. Using the Code icon next to the value, four lines of logic are used to calculate the cycle time for the current step:

```
int step = getlabel(token, "Step");
double multiplier = 100/gettablenum("StartUpLadder", step);
double newcycle = getlabel(token, "OldSpeed") * multiplier;
return newclycle;
```

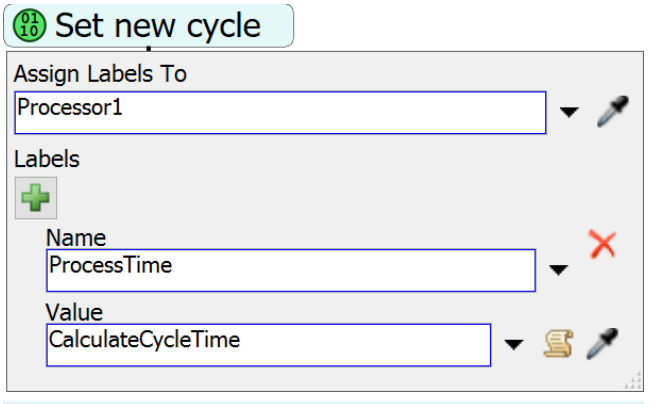


Figure 14. 7: Assign new cycle time.

With the new cycle time set, the next step is to see if the startup ladder is complete. This is accomplished using a decision activity as shown in Figure 14.8. The drop down choice of a Conditional Decide is selected. The condition test is whether the step number is less than 4. If it is, the token will go to the next activity, if the step number is 4 or greater, it will go to the sink. At this point the cycle time for the processor is the same as it was before the stoppage.

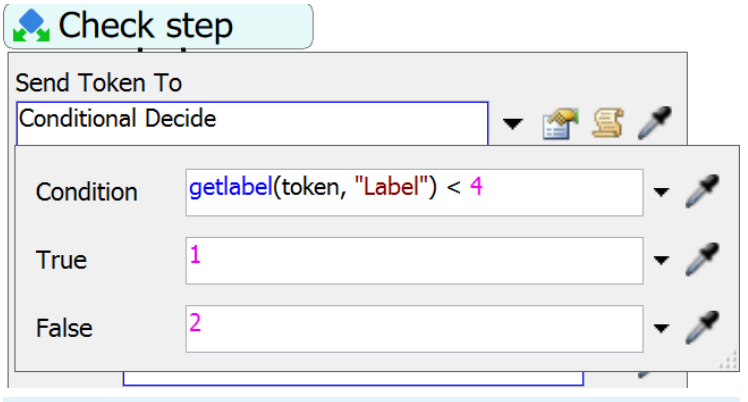


Figure 14.8: Checking the cycles.

Finally, a delay activity block is used to send the token back to the calculation block for the next cycle if needed. Blocks that directly

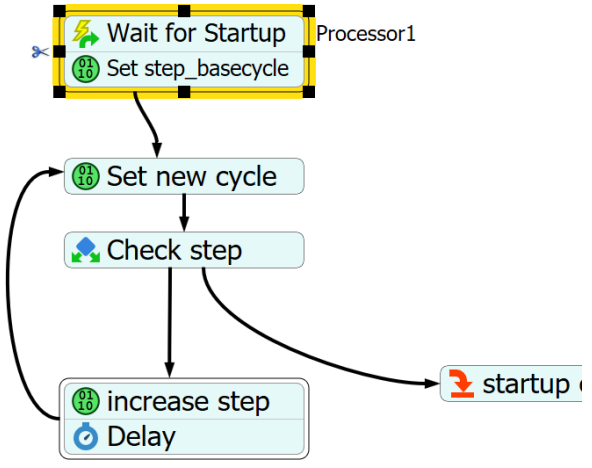


Figure 14.9: Stacked blocks.

go from one to the other may be combined into a single group by simply moving them together as shown in Figure 14.9. When the small arrowhead on the left of the block is clicked, the blocks separate. When the blocks are separated, a connector is no longer present and must be reconnected.

Except for the lines of logic to calculate the new cycle value, the logic using process flow elements is stored in one location and the logical flow is clearly visible. With a few changes, this set of blocks can be used as a general routine for starting up many processors. While these process flow blocks greatly enhance defining logical statements, some knowledge of how logic is used in a simulation is still required.

Section 14-4 Creating zones with Process Flow elements

There are many other areas of advanced functionality using Process Flow Elements for specific uses that can be explored in the *FlexSim* User Manual. The Zone functionality, however, is one that has general application. There are many standard ways to monitor items as they move through an entire simulation but Zones allow specific areas of the simulation to be identified and the items within them easily monitored.

Zones only refer to tokens in the Process Flow window. However, a token can represent a flowitem and while it is in the area defined by the zone, statistics are generated for it. Consider a manufacturing line where cases enter a circular conveyor system. They leave the conveyor if there is enough room in a staging area. From the staging area, the contents are checked before being taken to the warehouse. The amount of time required the check the case contents depends on the product type. The manufacturing floor shown in Figure 14.10.

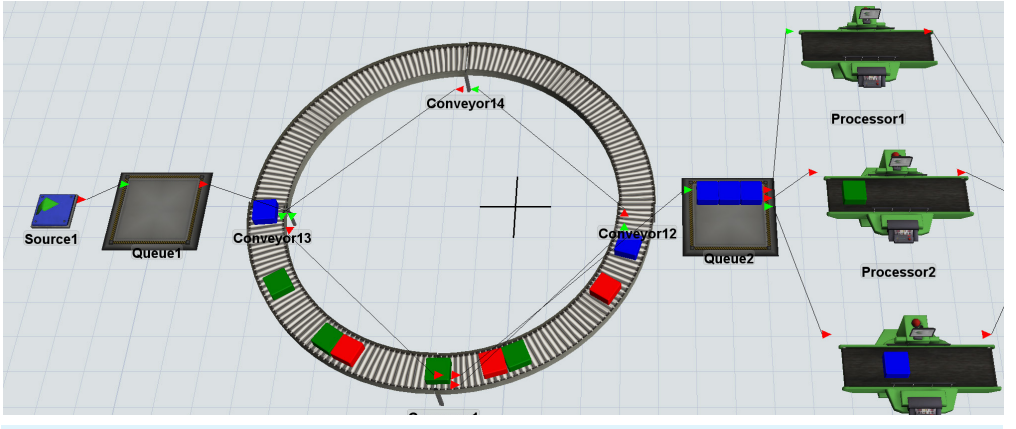


Figure 14.10: Zone Analysis Example.

The problem here is that the company wants to minimize the time the cases wait before the inspection. Presently each inspection station handles the same product type. This helps in the training process for the operators. They are now considering cross training operators so that the cases could go to the first available station.

A Zone function provides an easy way to monitor the cases within the waiting area by product type. Start by opening a general Process Flow window. Drag an event triggered source, assign labels, enter zone, wait for event, exit zone, zone, and sink logical blocks onto the surface and connect them as shown in Figure 14.11. The blocks should be given labels that indicate the activity that is happening within them.

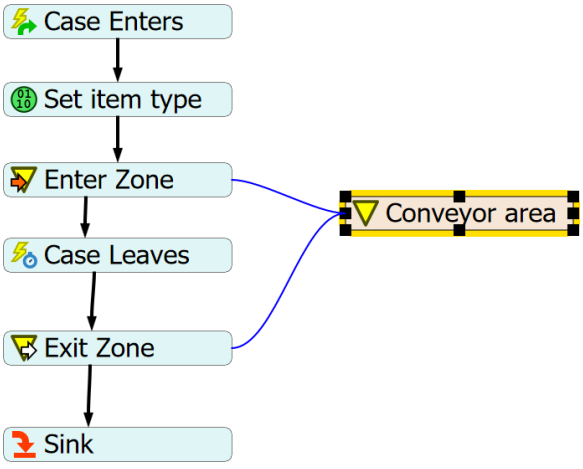


Figure 14.11: Logic blocks for a zone.

The source block is set up the same way using the eye dropper icon and should trigger a token when the On Exit is fired on the buffer just before the conveyors. The OnExit event is used since space on the conveyors is not expected to be a problem. On the Source, there is a label assignment that will create a label called "item" that will contain a reference to the flowitem in the model that just entered the conveyors as shown in Figure 14.12.

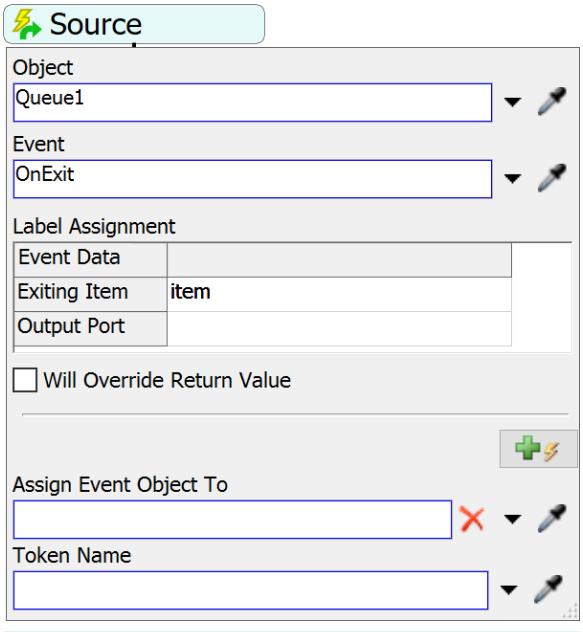


Figure 14.12: Source block.

The Assign label block is used to create labels on the token that contain variables that will later be used to determine the statistics collected while the token is in the zone. In this case, the item type is important so that the assign block creates a label that will carry the item type as shown in Figure 14.13. The command for the value indicates that the item type of the case that just enters is found on the reference item that was placed on the token when it was created in the first block.

Figure 14.13: Get item type.

Once the token is generated and the data assigned to labels on the token first zone border is added, the token will stay within the zone until the wait for event block hears that the case has been released to the processors. This is accomplished by using the eye dropper icon next the wait for event block, clicking on the queue before the processors, and selecting the OnExit event.

When this event happens, the block will match up the token leaving with the token that was created in the source block. It does this by matching the label called "item" on the token, as shown in Figure 14.14. Once matched, any statistics requested are calculated.

Having left the zone, the token goes on to the next block which is a sink and completes the journey through the logic.

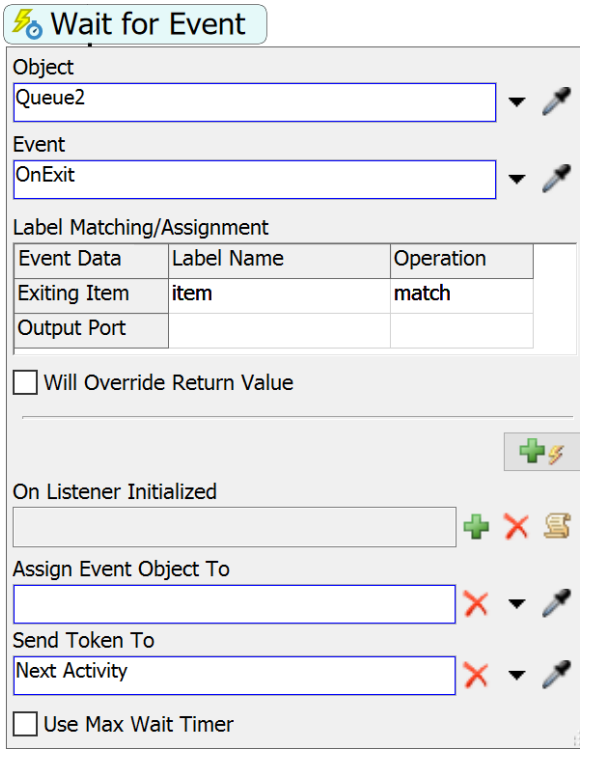


Figure 14.14: Wait for event block.

Next to the enter and exit zone blocks is an eye dropper icon which is used to connect each of them to the zone block on the Process Flow window. Immediately within the zone block is an opportunity to limit the number of tokens in the zone. If the tokens cannot enter the zone they stay in the enter zone block like flowitems in a queue.

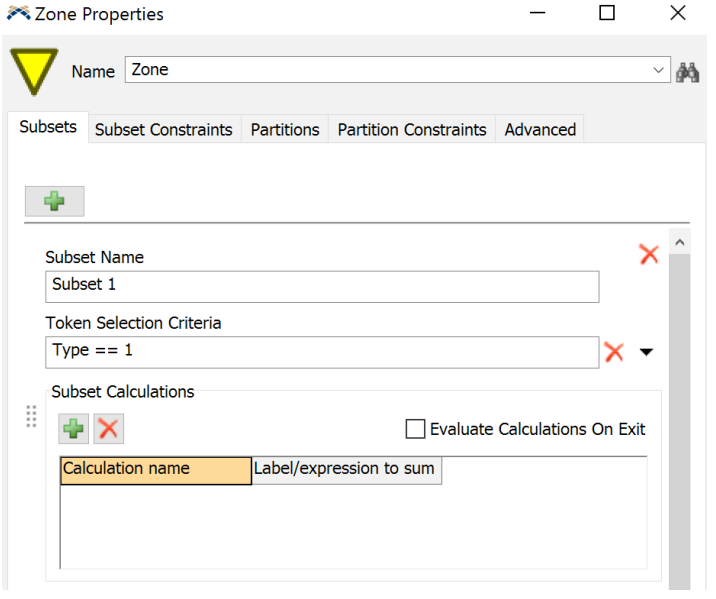


Figure 14.15: Setting up subsets.

While this functionality can be important with certain logic, the actions of tokens in the Process Flow tab must be synchronized with the 3D model through additional logic blocks. In this case, no action is occurring to limit cases flowing in the 3D model so the box to limit tokens in the zone is not checked.

To keep track of tokens in the zone based on the value of one of their labels, subsets are established by clicking the subset tab on the zone block as shown in Figure 14.15.

After the model is running, clicking on the zone element and choosing the statistics icon in the upper right corner of the quick parameters opens a window with dynamic details about the zone. These can be placed on a Dashboard using the pin icon.

Section 14-5 Synchronizing Process Flow elements with the 3D layer

Simulating operations that include many parts and tracking of inventories can require a lot of logic and data. Using the Process Flow elements can help to centralize the logic but care should be taken to correctly link the Process Flow to the 3D representation. The simple example shown in Figure 14.16 represents a work cell where two parts are placed onto a pallet. Parts move from the work cell queue to a loading queue (capacity of one) that transfers a flowitem to the pallet when needed. When the queues in the work cell reach a low level, more parts are moved to them from a main warehouse storage unit. When levels are low in the warehouse, additional parts are ordered.

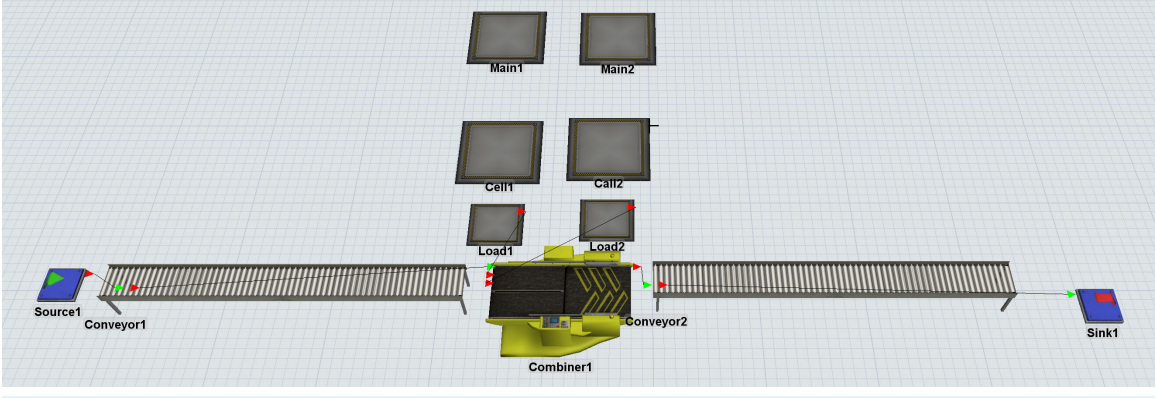


Figure 14.16: Simple work cell simulation.

Organizing the Data

The first step is to consider a logical plan for approaching the simulation and organizing the data. Starting by laying out the objects is always a good way to start such a simulation. Since the inventory flow part of the logic will be handled by Process Flow elements and will utilize the Lists functionality, no connections are needed from the main warehouse queues to the cell queues or from the cell queues to the loading queues.

Consistency and symmetry are important factors to consider when efficiently writing software. Since the main and cell queues will have similar logic start by creating organizing then as a group. Use the FlexSim group function by choosing view/show groups. The group box usually appears in the lower right corner of the screen. Create a new group by choosing the two main and then the two cell queues.

Referencing the queues in logic can be helped by creating a global variable (Toolbox/modeling logic/global variable). Select an array as the type and choose the main and cell queues using the eye dropper icon. A list of references to the individual objects is automati-

cally created. Keep the order of the queues consistent with the group created above. Figure 14.17 shows these groupings.

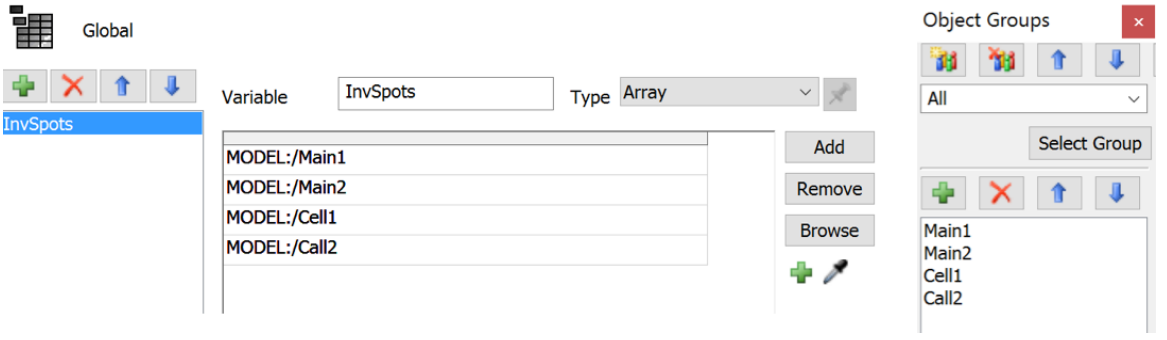


Figure 14.17: Global variable and groups.

An inventory global table can be created with a column for each queue. The rows indicate: the part number, starting inventory for the queue, the minimum value for the queue, and the number of parts to be transferred when the minimum value is reached.

When different combinations of parts are possible, a combiner is a good choice to use in the simulation. The number of parts can be changed in the OnEntry trigger based on the item type of the part coming in the first port (i.e. the pallet). To use that feature, a global table is needed to specify the part combinations. In that table, the rows represent the parts and the columns represent the combinations. In this instance, the number of columns would be three. The two tables are shown in Figure 14.18.

Model | Inventory | Products

	combo1	combo2	combo3
part1	1.00	0.00	1.00
part2	0.00	1.00	1.00

Model | Products | InventoryData

	Main1	Main2	Cell1	Cell2
Partid	1.00	2.00	1.00	2.00
Start	25.00	30.00	12.00	10.00
Minimum	10.00	15.00	4.00	3.00
Refreshamt	25.00	30.00	8.00	7.00
Orderflag	0.00	0.00	0.00	0.00

Figure 14.18: Global tables.

Each of the four inventory queues will have labels assigned to them providing the following information: the ID number of the queue (consistent with the order in the group and array created above), the part number, the starting amount of inventory, the minimum inventory, the refill amount, and an order flag that will be set so that additional orders will not be placed when one is already being filled.

An OnReset trigger can be used to populate the queue labels with the inventory data from the global table. Using the queue ID number to select the right column, the data is transferred using a gettablenum command. The order flag label is also set to 0 for each queue. This is a good method if, during runs of the simulation, changes will be made to the inventory data table. Otherwise, the data can be entered directly onto each queue and the

check box at the bottom of the label window checked to automatically reset the labels. In this case, the inventory data table is not needed. Note that setting up one queue and making copies can help save time.

Finally, a global item list (Tools menu) is needed to keep track of the entire inventory. The list function, as described in a previous chapter, can contain partitions. In this case, one will be for the main inventories and one for the cell inventories. On each of the Main queues use the Flow tab and select to push the items to the inventory list. Click the down arrows on the pop up window to be able to select partition 1 for their destination. Repeat the procedure for the two cell queues but select to send them to partition 2. Doing this will allow the system to automatically keep track of requests for parts and will backorder them if they are not needed.

The list automatically sets up a group of List Properties. The first one creates a property called PartType based on the item type of the flowitem moving in the list. This will be changed to identify the part with a label on the item instead. In the expression field choose "label" from the dropdown menu. Change the itemname to a name that will be set on each flowitem when they are created later. Remember both the property name and the flowitem label name as they will be needed later. This is a good time to make a list, outside of the simulation, of all the labels that are being used.

Creating the Process Flow logic

Open a new general Process Flow window. The logic that provides the initial loading of the inventory cells uses a Source schedule activity, a create object activity, and a sink. The source block schedules four tokens, one for each queue) to be created at time 0. A label is assigned to each of the four tokens that indicates the queue number (consistent with the created global array).

The Create Object activity sets the quantity, location and a label on the flowitem based on information contained on the labels of the queue where the items will be created. The queues are identified by the global array called InvSpots. The Process Flow blocks are shown in Figure 14.19.

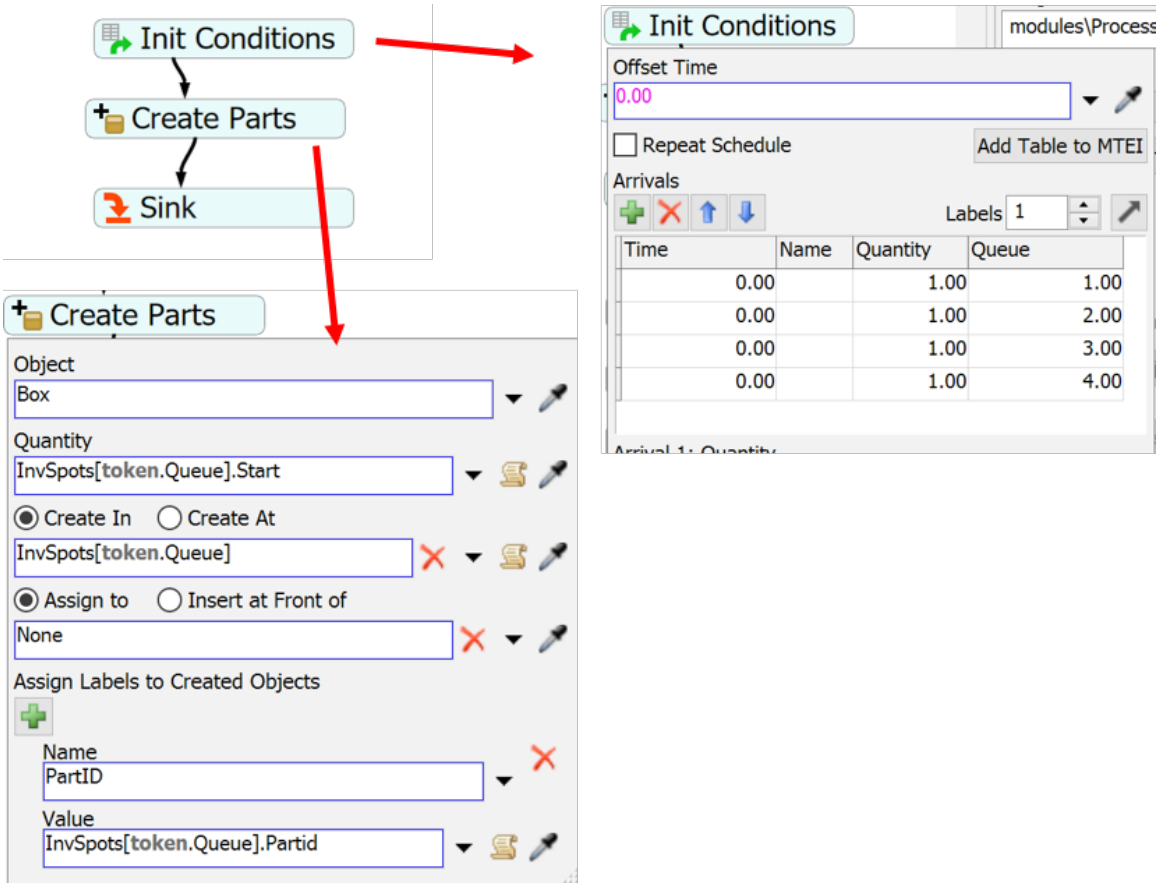


Figure 14.19: Initial inventory loading.

All the rest of the logic to move parts in the simulation is contained the another set of Process Flow elements. These elements listen for any part that leaves either the main inventory or a cell inventory. They then check to see if that queue has hit its low value and needs a refill order. If necessary, transfer logic is created to either move parts from the main inventory to the work cells or to order more parts for the main inventory. The logic elements for the Process Flow are shown in Figure 14.20.

The event listening element is set to listen for any exit trigger in the group of cells previously created. A label is added to the created token to reference the object where the exit occurred by providing a name in the area under "assign event object to:" using the assignment – *token.Spot*. Spot is the name used in this example.

The next activity is a delay that is important at this point. Without the delay, the token will travel directly through all the logic immediately - finishing before the exit trigger of the 3D object has finished its calculation cycle. Since this can possibly cause unintended actions, a very slight delay is added (0.5 time units) so all the 3D objects can complete their activities. The Process Flow activity then starts at the beginning on the next cycle.

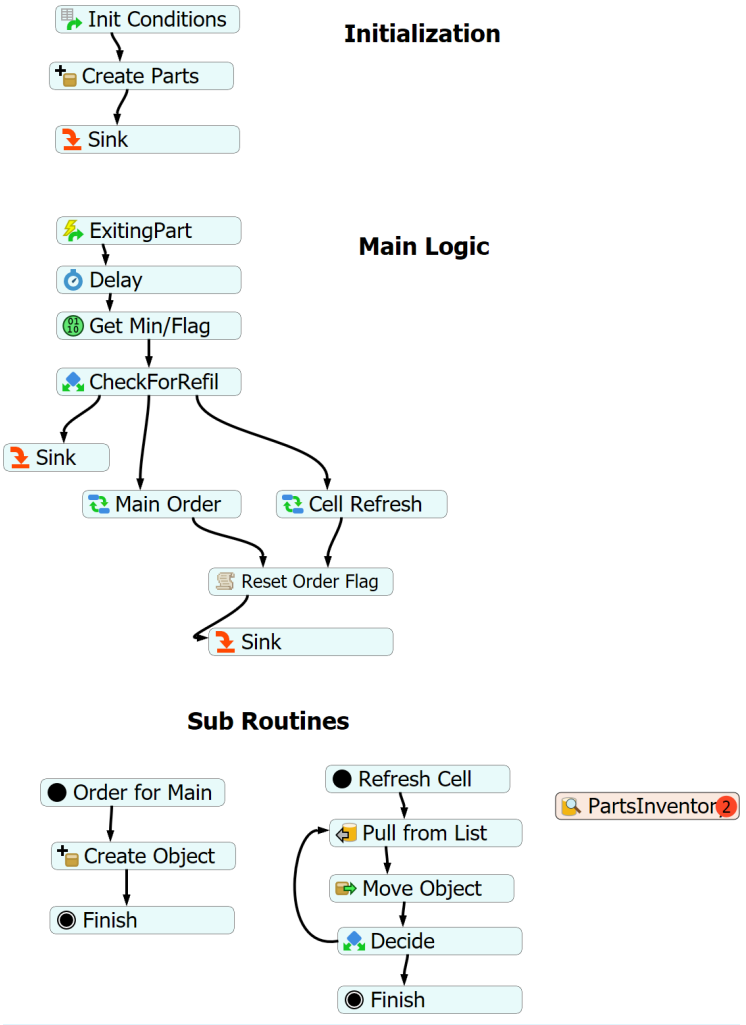


Figure 14.20: Process Flow elements.

By knowing the queue where the exit occurred, additional data can be transferred to the token for easy reference in the logic. For example, the minimum content for the queue is obtained by the command: *token.Spot.Minimum* – which references the appropriate label on the object identified by Spot. Similarly labels are made for the order flag value and the queue ID number. Additionally, a label is created that contains the number of items in the queue using the value: *content(token.Spot)*.

The decision activity provides the ability to direct the token based on a condition. In the "send token to" pick "Conditional Decide" from the dropdown menu. For this example, there are three things to be accomplished in this decision activity. First, check to see if the minimum level is reached and no other requests for material have been made. Second, if a refill is needed, set the order flag and get additional information from the queue creating the request, and third, determine if the queue is a cell or main inventory location.

To accomplish this, click the scroll icon to open the custom logic page and click the icon at the bottom left to clear the setup logic. Provide a name at the top of the code window. A possible set of instructions are shown in Figure 14.21.

CheckForRefil - Decision Point

```
1 /**CheckftheQueue*/
2 Object current = param(1);
3 treenode activity = param(2);
4 Token token = param(3);
5 treenode processFlow = ownerobject(activity);
6
7
8
9 if(token.Content <= token.Minimum  && token.Flag == 0)    // see if refill needed
10 {
11    token.Spot.OrderFlag = 1;               // set order flag on queue
12    token.Amount = token.Spot.Refreshamt;   // get amount to refill from queue
13    token.PartType = token.Spot.Partid;     // get part number from queue
14    if(token.queueid <= 2)    // this is a main queue
15      return 2;
16    else                      // this is cell queue
17      return 3;
18 }
19
20 else return 1;       // no action  send to sink
```

Figure 14.21: Decision activity code.

When using a decide activity it is important to check the logic to see that the result sends the token to the appropriate net activity. If no action is required, the token can go to a sink.

The cell and main inventory areas refill in different ways. An outside order enters the main area while the cell has parts transferred from the main area. Consequently, different logic is needed. The Sub Flow activity directs the token to a subroutine that carries out the logic and then returns the token to the main activity. The subroutine needs a start and a finish activity. When first putting a Sub Flow activity block in the page, a red circle appears. Moving over it, an eye dropper icon appears and, when clicked on a start activity, connects the Sub Flow activity to the proper subroutine.

The subroutine for the main queues is straight forward and utilizes Create Object activity based on the labels contained on the token: *token.Amount* and *token.Spot* (for the location). The activity also puts a label on the flow items being created that indicates their part number.

The subroutine for the cell queues moves a flow item (one at a time) from the associated main inventory queue to the cell. The Pull from List activity is set with a request and require number as 1 and assigned to "none". The Query is: *WHERE PartType = token.PartType* and the partition ID is 1 (the main queue partition).

The great strength to using the Pull for List activity is that, if the main queue were empty, a back order would be placed and filled when material arrived in the main queue. In process flow, the subroutine can handle multiple tokens so a request from another cell queue would go through without any problem. Creating this standard code in a custom way would be difficult.

The Move Object activity will transfer the first flow item from the main queue designated in the global variable array to the queue calling for the item. The expression to identify the place to take the item from is: *InvSpots[token.queueid – 2].first*.

The destination is: *token.Spot*.

Since the Move Object is only moving one item at a time a simple counter is needed to transfer the correct amount. A Decide Activity with the following custom code would work:

```
int amount = token.Amount -1;
if(amount <=0)
    return 2
else
    {
          token.amount = amount;
    return 1;
```

Once the subroutines have returned to the main logic flow, a Custom Code activity is used to set the reset flag on the proper queue with the single statement: *token.Spot.Orderflag = 0*.

The flow items are entered into the appropriate partition in the list by the flow tab on the queue which fires whenever an item enters.

Setting up the 3D logic for the simulation is straightforward. The source object sets the item type on each of the pallets created. The item type varies (duniform) from 1 to 3 which matches the three combinations of products. On the combiner's entry trigger, the choice to update the combiners component list is selected and the product global table inserted.

The two load queues each should have a maximum content of one and the pull selected on the flow tab. The option to pull from the inventory list should include the query: WHERE PartType = 1 or 2 depending on which load queue is being used. In each case, the Partition should be set to 2 – which is the cell queue inventory partition. When using the query on an object, the statement should be selected by using the dropdown choices.

The best way to test out the program using Process Flow is to set the speed slider to 0, then click reset and run. By clicking on the step button, the tokens can be seen going through the Process Flow. If the tokens go past some logic activities too quickly, a delay activity can be temporarily put into the logic flow.

It's obvious that the Process Flow Elements can provide a very rich, and efficient method for handling logic calculations. Because of its power, simply starting to create elements without first thinking through the entire simulation plan, can result in confusion and unexpected errors which may or may not be obvious. When used correctly, the combination of 3D visualization and complex logic will result in a superior simulation project.

Exercise 14-1 Greyson Operations

Background

As a professional photographer, Edward Greyson decided to start a mail order business targeting amateur as well as professional photographers. While starting mainly with photographic supplies, Greyson Operations began selling high quality cameras and lenses. They believe the demand will be great enough to increase their business and selections. Before going ahead with the expansion, they want to look at the impact on inventories, costs and the viability of their one packing line.

Problem statement

Provide a simulation to help determine how the increase in business will impact the packing line and its ability to process orders. Since the new products are more expensive, inventory carrying charges will be important as well as cash flow. A new engineer started with a model of the physical pieces as shown in Figure 14.22 but is having trouble organizing the data and logic. You are expected to help with simulation.

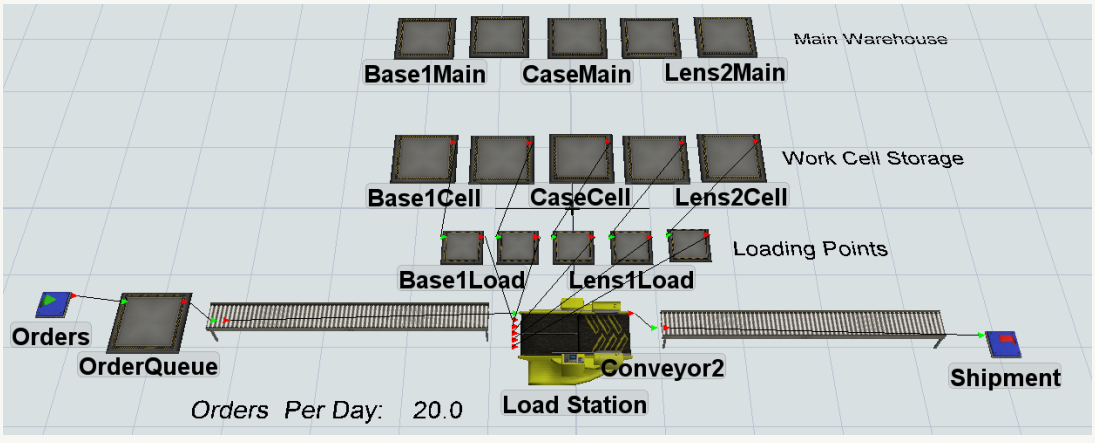


Figure 14.22: Physical layout for Greyson Loading.

Operating data

Customers can order either of two base cameras and optionally, include a case and one or two lenses for the camera.

Orders are expected to arrive at a rate of 20 per day (exponentially distributed) and may increase to at least 25 or 30 a day during peak seasons. An order tag, indicating the order configuration is placed on a carrier which travels down a conveyor to the loading station. At the loading station individual spots are maintained, each with a single camera body, case, or lens ready to be loaded into the shipping container. It takes 30 seconds to carefully load all required pieces into the container.

Each time a part is loaded it is replaced from a local inventory of parts. When the local inventories reach a minimal level a number, more parts are provided from the warehouse inventory area.

When the inventory levels in the warehouse reach a minimum, an order is placed with the suppliers. Shipping times from the suppliers vary. The time to receive material, the base cost of each part, and the profit from each part is shown below:

Item	Cost	Sale Price	Order Delay (days)
Camera Base 1	$150	$225	4
Camera Base 2	$250	$400	6
Case	$25	$45	2
Lens 1	$60	$90	3
Lens 2	$90	$135	4

Marketing has projected the following percentages for each of the 16 possible order combinations:

	Identifier	Percent	Product ID
Base 1 only	B1	15	1
Base 1 & Lens 1	B1L1	6	2
Base 1 & Lens 2	B1L2	5	3
Base 1 & both Lens	B1LL	3	4
Base 1 & Case	B1C	10	5
Base 1, Case, Lens 1	B1CL1	8	6
Base 1, Case, Lens 2	B1CL2	7	7
Base 1, Case, both Lens	B1CLL	6	8
Base 2 only	B2	6	9
Base 2 & Lens 1	B2L1	5	10
Base 2 & Lens 2	B2L2	4	11
Base 2 & both Lens	B2LL	3	12
Base 2 & Case	B2C	10	13
Base 2, Case, Lens 1	B2CL1	4	14
Base 2, Case, Lens 2	B2CL2	3	15
Base 2, Case, both Lens	B2CLL	3	16

Expected results

- Make estimates of starting inventory levels, minimum levels, and refill amounts.

- Provide guidance about the viability of the business and what could be done to improve it.

Modeling and analysis issues

- Build the simulation in steps saving them as separate names
- Think through the entire approach and basic logical steps
- What is the best time unit to use?
- What global tables, variables, and groups should be used?
- How can the orders be produced to reflect the marketing projections?
- Will setting different colors for each part help to visualize if the loading station is working correctly?
- Consider the various ways information can be reported from the Process Flow objects as well as the 3D objects using a dashboard or other methods.
- How can you tell if orders have been delayed at the loading station because of a lack of a part?
- Can zones be useful in the model? Can you easily create the zones using a few additional Process Flow elements?
- How long will it take for profits to cover the inventory investment and charges?

Chapter 14 Review Questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Describe three type of activities where Process Flow may be helpful in a simulation
3. Describe two examples of where zones can be used.
4. How does consolidating simulation logic improve understanding, debugging, and changing a model?
5. Why is it important to maintain a visual environment in a simulation? Compare the impact on people trying to understand and accept a simulation when they watch tokens moving in the Process Flow window with items moving in the 3D environment.
6. Explain why planning a simulation strategy is important when modeling a complex problem.
7. What are some of the challenges when trying to sync Process Flow logic and the 3D environment.

Chapter 15 Simulating Fluid Flow

The majority of simulation application packages on the market are based on discrete-event simulation; however, interest in simulation has grown to include operations involving the continuous flow of material such as liquids, powders, and high speed bottling. Simulating both a processing and packaging step in a production facility involves the transition between continuous and discrete events. A few simulation packages only simulate fluid flow, just as some only simulate discrete events. Others, like *FlexSim*, are capable of simulating both fluid and discrete systems in the same simulation model.

Objectives

Define the concept of continuous simulation; describe how discrete event simulation can be used to represent the characteristics of fluid flow, high-speed operations, and large-scale particulate product flow; describe how discrete event and continuous elements interact in a simulation.

Section 15-1 Basics of fluid-flow simulation

Fluid Concepts

A material flow is considered to be a fluid when its moving material cannot be identified by individual objects but appears as a continuous body. Fluid, however, doesn't necessarily mean liquids. Liquids, gases, and small particles such as rice, cereal, and sand are obvious examples of fluid flow. A less obvious example of a fluid is a high-speed bottling line (> 1600 bottles per minute); it can be thought of as either a fluid or a discrete material flow. A fluid (or continuous) flow is usually described in terms of units per time, such as gallons per minute.

Being able to simulate fluid flow is important to industries that have both continuous and discrete manufacturing processes. The manufacturing involves a process area where fluid-type materials are produced as well as a discrete area such as a packing line. Discrete

material flow has specific events associated with it, such as entry into a machine or exit from a queue. These events are placed into an events list for processing in time order by the discrete-event simulator. Fluid flow, however, moves continuously but also can have some events such as when a tank fills.

There are two basic approaches to simulate fluid flow in a discrete simulation software application. Each method involves creating discrete events. In the first method, which has been part of the standard *FlexSim* software, a clock (called a ticker in *FlexSim*) creates events at set intervals of time, as illustrated in Figure 15.1. When the first fluid object is brought onto the simulation surface, the ticker is automatically added. The ticker can be moved to any position on the screen.

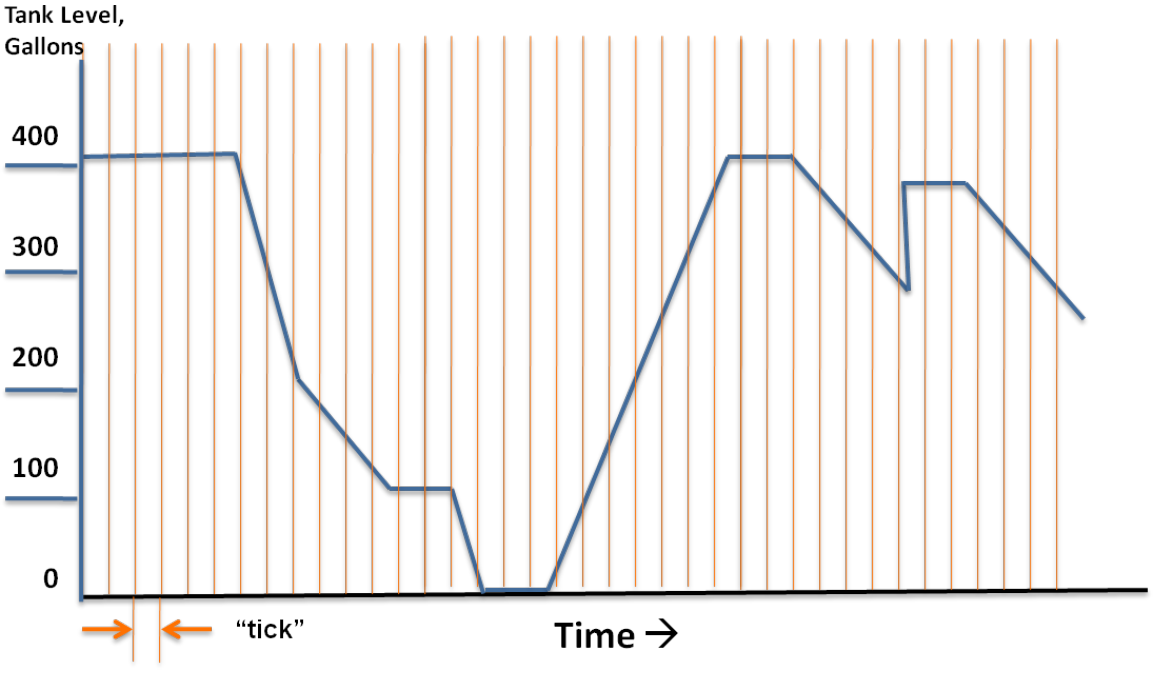


Figure 15.1: Fluid Intervals.

The intervals need to be small enough to capture all of the characteristics of the fluid flow. At each ticker event, the condition of each fluid object is updated by calculating the amount of fluid moved from one object to the next. The calculation has to take into account the material available, the space available, and the input and output rates. A recursive calculation is continued along the complete fluid network.

The second method, which is called *FloWorks* and is now available in *FlexSim*, was developed by Talumis B.V., an engineering firm and *FlexSim* distributor in the Netherlands. When a *FloWorks* object is added to the simulation a FlowController is placed on the main screen. The controller handles all the logic for simulating flow through the objects. Recursive calculations in this method start when the model is reset. Material only moves at the time an event occurs. At that time the next fluid event is determined by the flow controller. Events may be a specified level being met or a new flow starting as shown in Figure 15.2. Once the event occurs, new calculations are made to determine the time of the next

event – no further flow calculation are carried out until the time of the next event.

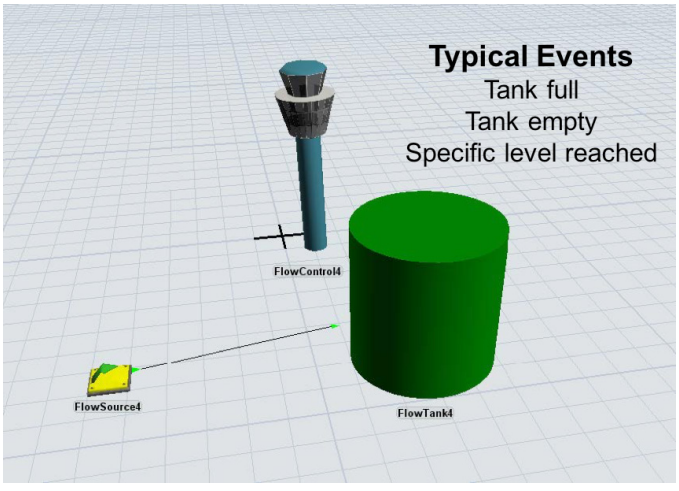


Figure 15.2: Fluid Events.

Both methods are capable of simulating fluid activities. The ticker method allows for complex logic to be taken at many points in the simulation since the ticker fires at a high rate. However, in cases where little logic is needed over long periods of time, this method can add many more calculation cycles than is needed. An example would be a large pipe network that fills a tank as very little happens until the tank level is satisfied. If larger tick times are used, the number of calculation cycles are reduced but the possibility exists for errors when a flow event, such as a level target, happens between ticks. While more accurate and quicker, the flow controller method makes changing any flow event targets dynamically difficult. New events, such a flow targets, need to be created to add logic between events.

Since the ticker-based flow objects are presently in *FlexSim* and are well described in the help files, the *FloWorks* objects will be discussed as the basis for the rest of this chapter. Much of the functionality is the same for both methods and even the user interfaces are similar.

Section 15-2 Fluid objects

In *FlexSim*, FlowWorks objects are found in library panel on the left. Fluid objects are similar in basic function to their discrete counterparts. The difference comes in setting their specific variables. The *FlexSim* User Manual in the help menu covers both sets of objects.

Fluid Generator

Discrete Analogy: Source

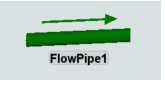
Function: Creates fluid material

Operation: Provides an infinite supply of fluid material. The tank will either continuously refill at a fixed rate or run empty and then refill after a specified delay.

Fluid Terminator

Discrete Analogy: Sink

Function: Removes material

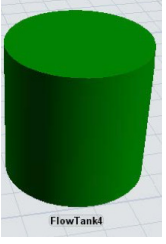


Fluid Pipe

Discrete Analogy: Conveyor

Function: Move material from one object to another

Operation: Provides a transfer object for fluid. Fluid flows out of the pipe as it enters. The pipe has a capacity and observes plug flow as a new product enters. The output rate will match the input rate. The output pipe has two options to direct flow: First Available sends the most fluid possible to the first available downstream object, and Flow Evenly tries to balance the flow to all downstream objects.



Fluid Tank

Discrete Analogy: Queue

Function: Stores fluid material

Operation: Acts as a surge for fluid material. Triggers are available when the tank level reaches specified levels. The input and output rates can be specified.

Fluid Blender

Discrete Analogy: Combiner

Function: Blends fluids into a single material based on a fixed percentage

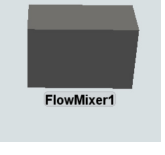
Operation: Used for in-line, continuous blending of fluid streams. The blender will automatically adjust input rates to maintain specified mix percentages.

Fluid Splitter

Discrete Analogy: Separator

Function: Splits flow based on fixed percentages

Operations: Used for in-line, continuous splitting of a fluid to different downstream objects. The splitter



Fluid Mixer

Discrete Analogy: Multiprocessor

Function: Combine materials in a batching operation

Operation: Maintains a processing Step Table that includes actions taken at each step, any delay times, and a Recipe Table that includes the source, amount, and addition step for each material used. Material leaves the mixer only when the batch is complete.

Item to Fluid

Function: Converts a flowitem to a fluid

Operation: Simulates emptying individual containers of fluid materials by converting the material content of each flowitem into a fluid stream.

Fluid to Item

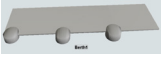
Function: Converts an amount of fluid to a flowitem

Operation: Simulates packing a specified amount of fluid material into a container.

Fluid Conveyor

Function: Conveys material in an open trough

Operation: Simulates the flow of fluid-like material (e.g., fine sand), in an open conveyor-like object. The direction of material flow can be changed as well as physical attributes of the object itself.



Fluid Berth

Function: The Berth object simulates a location at which a loading or unloading operation of a vessel takes place, such as a ship jetty.

Operation: The Berth may be used by itself to simulate a timed loading or unloading operation, or in conjunction with a Loading Arm in which the (un)loading time is determined by the flow to or from a flow tank on the loaded item. The Berth also supports pre- and post-loading events. The Berth is a vessel dedicated derived object from the Loading Point using the fluid Flowitem Bin objects.

Additional details for these objects and other *FloWorks* objects are provided in the User Manual.

Section 15-3 Specifying Fluid Operations

Fluid objects have more robust choices than their analogous discrete objects. The fluid tank is typical of the many flow choices available. Both the maximum and initial content can be specified as well as attributes for the flow both into and out of the object.

Flow attributes are specified on the flow tab of the object as shown in Figure 15.3. The input and output ports have a total maximum object flow rate that regulates the allowed flow in or out of the object Two flow options can be selected using the drop down choices.

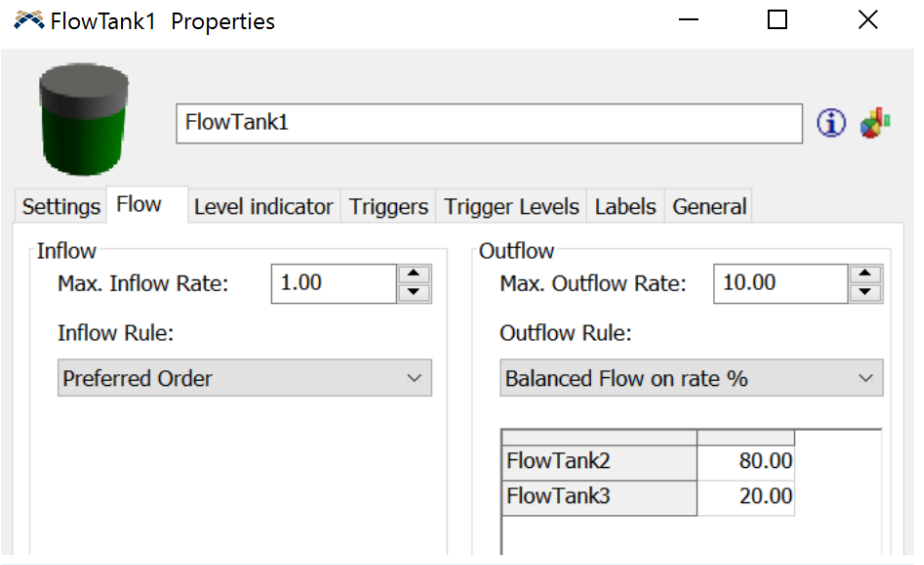


Figure 15.3: Flow tab.

- "Preferred order" will try to maximize the flow rate of the outgoing connections in the order of connection, possibly reducing or cutting off flow to lower ranked objects.
- The "Balanced flow rule" will distribute the available output in a fixed ratio to all connected objects. If one of the downstream objects is constraining, all objects connected to the source will receive less flow in order to maintain the specified ratio.

Fluid ports, unlike discrete object ports, can be managed independently. For example, individual fluid ports can be opened and closed. Consequently, additional commands are available which are listed in the User Manual.

All *FloWorks* objects have a product type defined. The initial product type is defined on the object by number and a color assigned. The product type is not automatically transferred from one object to the next, but only affects the product inside or flowing through the object. The product type can be used for modelling purposes and changes the color of the flow in the object. FlowWorks commands can be used to set the product type on any object.

Objects that can hold fluid display their current content as a percentage of the maximum content. This display is a colored bar called the level display or level indicator. For the Tank and the Mixer this bar is, by default, used to visualize the entire object. If a bar is fully colored, the object is full of the product assigned to it.

With all the variables associated with fluid objects, there are a few points to keep in mind:

- Pipes, like conveyors, are not required to move material from one object to another.
- When there are multiple fluid objects in series, be sure that the rate settings are correct in order to avoid creating an unintended restriction to the overall flow.

Mixer object

The fluid mixer is probably the most versatile fluid object. It represents one of the most important of all fluid operations—batch processing. The use of both a step and recipe table to describe a batch operation is typical of process industry practice for describing the mixing process.

While the Mixer tabs contain the normal fluid object variables, the Steps tab provides the batch description. The Steps tab, as shown in Figure 15.4, has three separate sections: Mixer Steps, Mixer Recipe, and mixer triggers. The Mixer Steps section allows you to add or subtract steps by using the Number of Steps field, and update changes by clicking the Update button. Changes will be reflected in the table below. The Description is a text field for descriptive use only. The Delay time, in simulation units, occurs either after any addition listed for that step or immediately if no addition is listed for the step.

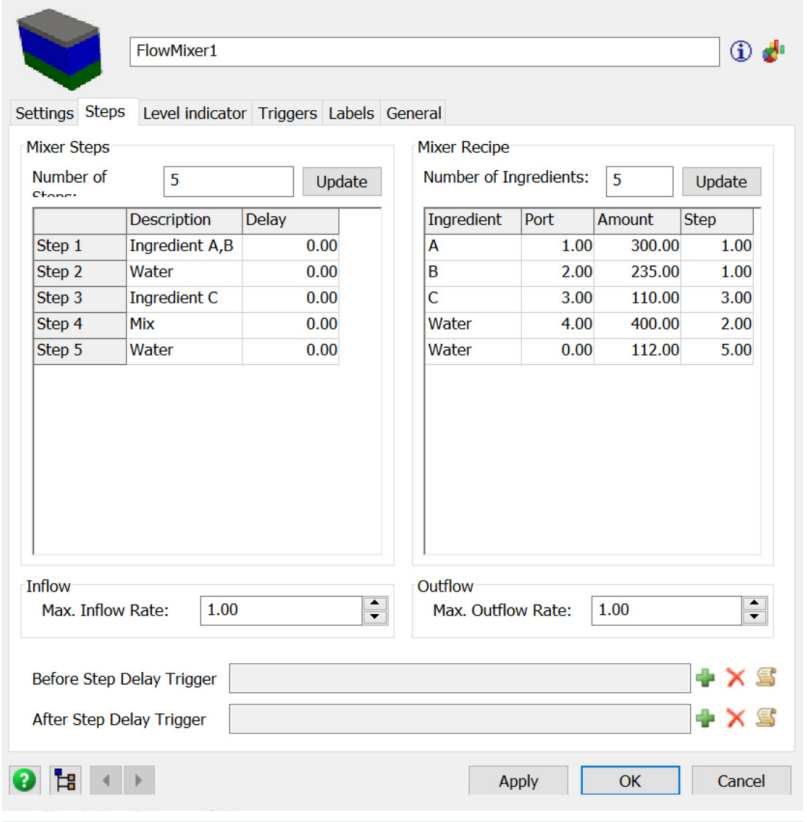


Figure 15.4: Mixer Steps tab.

The Recipe table contains the ingredients along with the port through which they enter the mixer. Each material entry has a step number associated with it to indicate when in the steps the material should be added. The ingredient field is a text field for descriptive use only. The number of recipe entries can be changed by adjusting the value in the field and clicking the Update button. An entry is made for each time a component is added to the mix. Note that in the

example, water is added more than once in the recipe and that the recipe list doesn't have to be organized by step number.

The recipe tab also contains triggers that execute before and after each step.

Berth Object

The Berth is another *FloWorks* object that has unique characteristics. It represents a station that can simulate the loading of multiple objects. The parameters page for the berth is shown in Figure 15.5. A discrete source can send standard flowitems to the berth. The number of items that the berth can handle is selected and the loading or unloading time specified. By itself, no material is transferred to the flowitem in the berth. In this way, the berth resembles a processor with a capacity of more than one item.

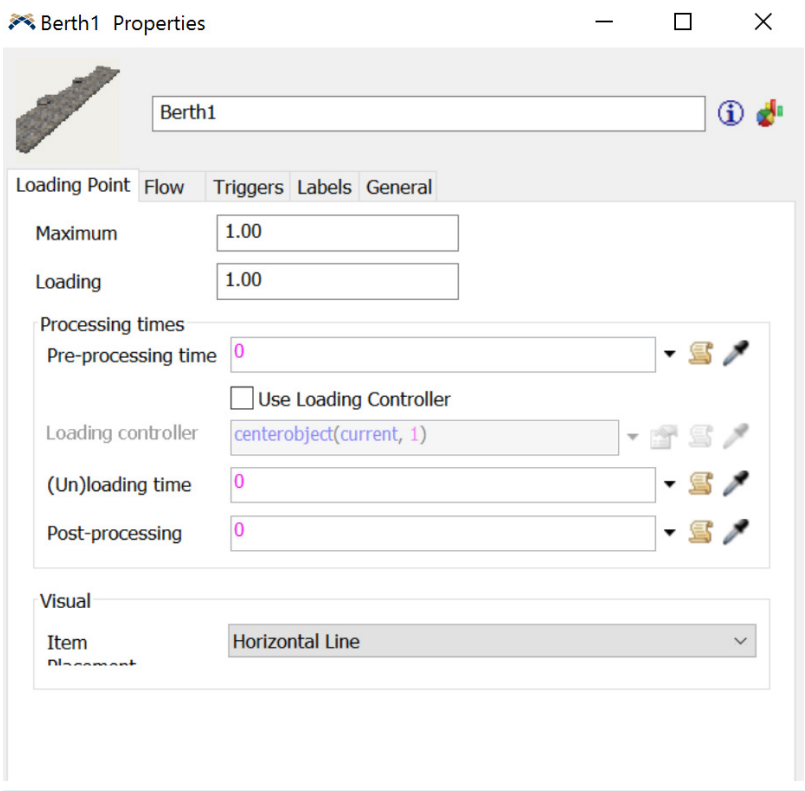


Figure 15.5: Berth parameters.

However, the berth can load multiple flow objects when used with the *FloWorks* loading arm and the special FlowVessel and FlowTruck items in the FlowItem Bin as shown in Figure 15.6. The loading arm is attached to the berth through a center connection while a fluid source is attached to the arm with an A connection. The Use Loading Controller box must be checked. The capacity of the vessel or truck must be specified in the FlowItem Bin or with a berth trigger on entry. When using the loading arm, the loading time menu on the berth is not used since the loading time will be dependent on the object capacity and the flow rate used.

Note that the vessel and truck in the FlowItem Bin are made up of two objects. A task executer with the shape of a truck or vessel and a tank shaped as a rectangular object. The two must be used together in *FloWorks*. To visibly use some other task executer to carry the imbedded tank, the vessel or truck 3D image can be hidden by selecting it and choosing to hide the image in the General tab. A different task executer can then be attached to the source and the Use Transport box checked on the flow tab. The on load trigger of the task executor may have to be used to correctly position the tank.

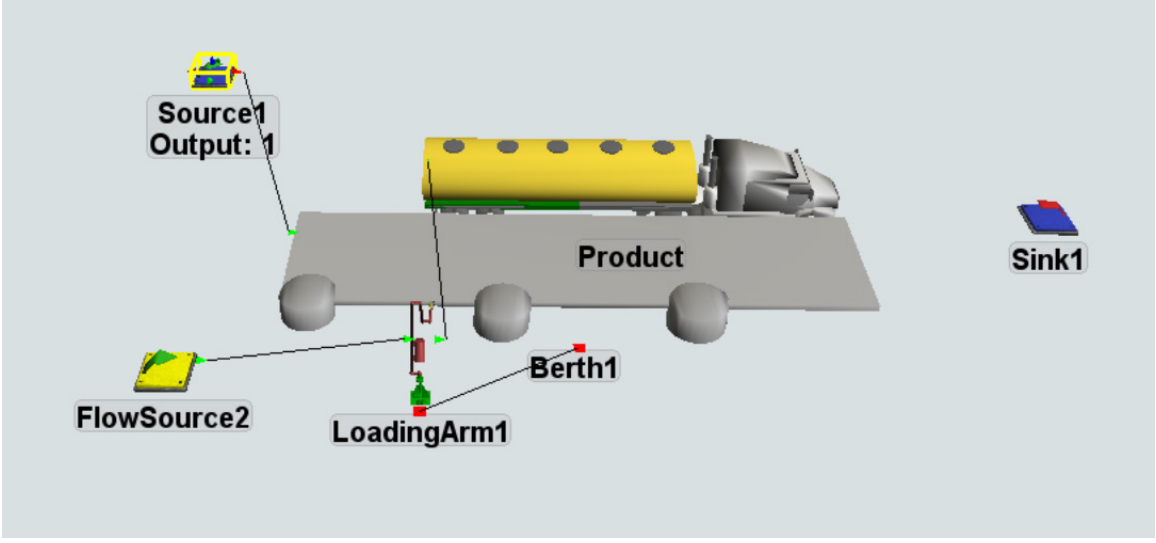


Figure 15.6: Flow Truck in Berth.

Exercise 15-1 James Peanuts

Background

James Peanuts, a family owned and run business, has seen significant growth over the last few years. After producing a homemade gourmet potato chip for many years, James finally gave in to his advisors, who recommended expanding their line with "flanker" products, namely flavored peanuts.

The small manufacturing group came up with a design for producing the new products that they felt was flexible and lean. It involved a batch makeup system of flavoring to add to the peanuts. Regular peanuts could be run on the line while the flavor system was cleaned for the next product. Everything went well until demand started increasing and the flavor system became a bottleneck. The group could not understand the problem since their calculations showed the batch make-up time was short enough to handle the higher rates. As a member of the manufacturing team, you decided to use simulation and take a different approach to the problem.

Problem statement

What will it take for the flavor system to run at higher production rates without becoming bottlenecked?

Operating data

Flavor additives are made up in a mixer that gets material from three liquid flavor tanks as shown in Figure 15.7. Additionally, a powder additive is dumped into a hopper by an operator and metered into the mixer. The mixer mixes 4,325 pound batches that are transferred to a smaller use tank that acts as a buffer to maintain a continuous flow into a machine that coats the flavor onto the peanuts.

The plant follows the general practice of using pounds as a common measurement, thus eliminating density conversion factors when dealing with liquids.

Also, mass flow and weight measurements are more common and accurate than volumetric meters.

Flavored peanuts are produced by feeding the peanuts, after they're roasted and taken from the shells, into a blender (capacity 20 pounds)

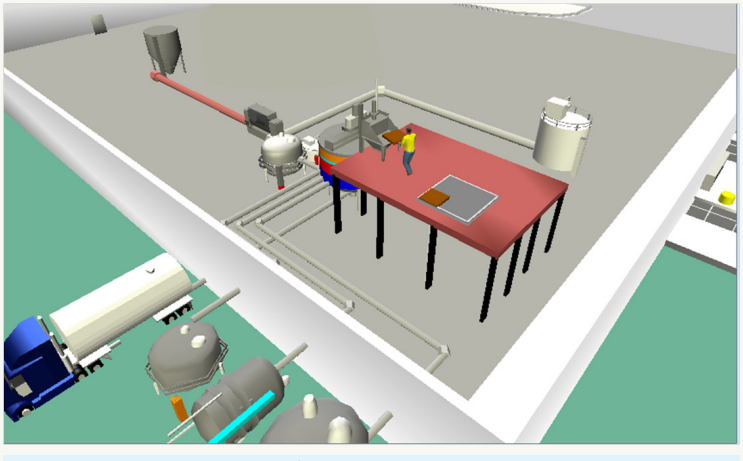


Figure 15.7: Peanut operations.

where the flavor coatings are sprayed onto them. The weight percentage of the flavored product is:

- 60% peanuts
- 40% flavor

Normal dry peanut production is 95 lbs/min (1.583 lbs/sec).

Flavoring additives 1, 2, and 3 are maintained in storage tanks that are automatically refilled. The powdered flavoring comes in 40 lb sacks that are dumped by an operator into a hopper next to the mixer. Details of the tanks are:

Unit	Capacity (lbs)	Initial Level	Discharge Rate (lbs/min)	Discharge Rate (lbs/sec)
Flavor 1	1500	1500	128	2.133
Flavor 2	2000	2000	130	2.167
Flavor 3	1000	1000	105	1.750
Dry mix hopper	400	0	143	2.383
Use tank	3300	3000	150 max	2.500

The recipe follows standard nomenclature and is divided into tables of steps and components; the tables are provided below. Ingredients are added individually. Once the mix cycle is complete, the material can transfer to the use tank. The transfer pump is either on or off. The mixer cycle is automatic and starts again as soon as a batch is transferred. The mixer transfer rate is 570 lbs/min (9.5 lbs/sec).

Ingredient	Recipe Table Amount (lbs)	Step
Flavor 1	1460	1
Flavor 2	1605	2
Flavor 3	932	3
DryMix	328	4

Step	Step Table Description	Delay (sec)
1	Add Flavor 1	0
2	Add Flavor 2	0
3	Add Flavor 3	0
4	Add DryMix	0
5	Mix	420

The plant's lean team calculated that they could make and transfer a batch approximately every 45 minutes, which could support a dry peanut rate of over 130 lbs/min. Since the normal rate was 95 lbs/min of dry peanuts, the lean team decided that they would reduce the buffer (use tank) size to 3300 lbs. It is important that the use tank not run dry so as to stop the peanut flow as it reduces production and may degrade the peanuts.

To save wear and tear on the mixer's transfer pump that would be caused by excessive cycling, the maintenance group set up level triggers on the use tank. When the level gets to 2000 lbs, the transfer pump turns on to fill the tank from the mixer. The pump is turned off when the level in the tank reaches 3200 lbs. The maintenance group wants at least 500 pounds between the high and low levels.

Expected results

Starting with the standard dry peanut rate, determine when the flavor additive system becomes bottlenecked.

Consider rates from the base of 95 to 120 lbs/min of dry peanuts. Run the simulation for 2 shifts (16 hours), the typical run cycle. Explain why the estimates of the lean team were incorrect.

Modeling and analysis issues

- What assumptions can be made about the operation?
- What time unit should be used and why?
- Should the peanut flow be fluid or discrete?
- Are pipes needed to connect equipment?
- How are bags of dry flavoring transferred to the mixer?
- What performance measures should be calculated?
- Are there improvements that can be made that do not require any physical changes to the equipment?
- What visualization elements can be used?
- How can the dry peanut rate and other variables be easily changed from one location?

Exercise 15-2 Western Grain

Background

The Western Grain Cooperative is planning a new grain loading facility to transport grain from their main silos to barges on the Mississippi River. Grain is loaded onto barges at the facility's four docks. Before they go ahead with the plans, the members of the board want to be assured that the concept will work and that grain will not be left in the main facility. They want to be sure that they can ship between 55,000 and 60,000 tons of grain per day.

Problem statement

Determine the feasibility of the loading operation including the number of tugs that are required.

Operating data

Empty barges are dropped off at a holding area where they are picked up by tugs and transported to the docks as shown in Figure 15.8. When the barge is full, it is picked up by a tug and moved to a holding area to await transportation down river.

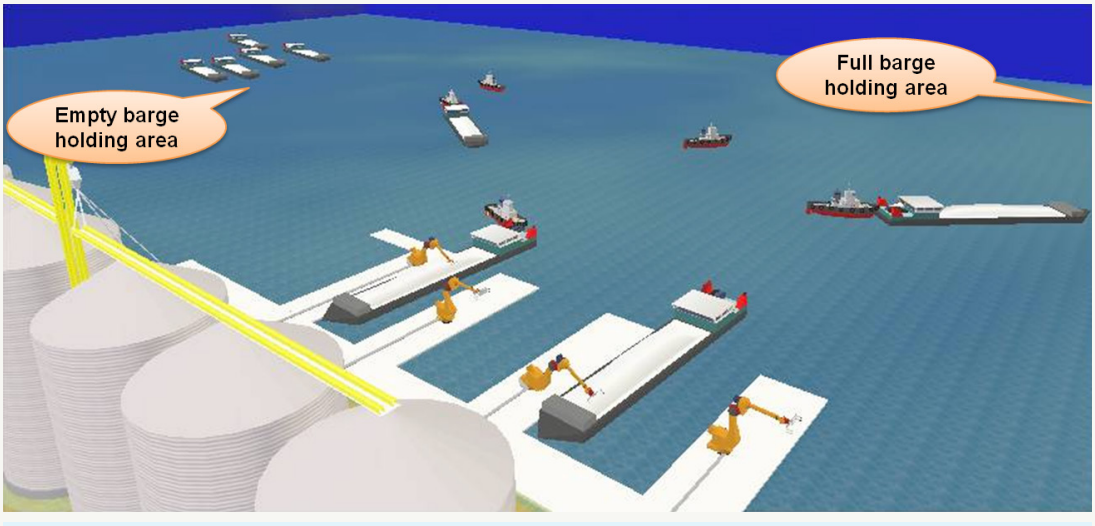


Figure 15.8: Grain loading.

Grain is discharged from the storage area at a fixed rate of 40 tons per hour and distributed to one barge at a time until the barge is full. Each dock can load at the same rate.

At the dock, time is required to connect the barge to the chute and then to disconnect it before it is free to be taken out. The times (in minutes) is

- Connection: Triangularly distributed with min: 6, max: 15, most likely: 9

• Disconnection: Triangularly distribute with min: 4, max: 8, most likely: 6

The dock operates 24 hours a day. The average number of barges arriving at the empty barge holding area are as follows. Assume Poisson-distributed arrival rates.

Time period	Avg. no of barges per hour
00:00 – 07:00	1.0
07:00 – 15:00	3.5
15:00 – 21:00	5.0
21:00 – 24:00	1.5

Capacity (tons) percentage:

Capacity (tons)	Percentage
400	15
600	20
800	15
1000	40
1200	10

The distances in the operations area are as follows:

From/To	Distance (miles)
Empty barge area to any dock	1.0
Any dock to Full barge area	1.3
Between docks	0.2
Full barge area to Empty barge area	1.5

Assume the tug boats operate at an average speed of 5 miles/hr.

Expected results

Develop a simulation model and analyze loading operations in terms of the number of barges awaiting loading, their wait, the number of tug boats used, and the tons of wheat shipped per day. Run the simulation for a 10-day period.

Modeling and analysis issues

- What time unit should be the basis of the simulation?
- What level of detail will be used? What assumptions are made?
- What variables should be used and where should data be kept?
- Should fluid be used to represent the grain?
- What can represent a barge? Does it actually have to be "loaded"?
- How will the barge arrival schedule be simulated?
- How will the barges get to and from the docks?
- How will the travel time be simulated?
- What are the performance metrics?

Chapter 15 Review Questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important in the chapter.
2. What is the basic unit of measure that is commonly used with materials that are considered “fluids”? Provide two examples of specific measures.
3. Name two materials that are not liquid that can be considered fluid for a simulation.
4. Describe three operating systems that might use fluid objects; describe why they would be used.
5. Describe the difference in how states change between a continuous (or fluid) flow simulation and a discrete-event simulation.
6. Describe how you might simulate a liquid flow using normal discrete simulation objects rather than fluid objects.
7. Define a “tick” time unit.
8. Describe the difference between a fluid blender object and a fluid mixer object.
9. Describe the difference in the operation of ports on a fluid object (e.g., a pipe) and a discrete object (e.g., a processor).
10. Describe operating systems where you might use a fluid-to-item object and an item-to-fluid object.
11. Select three fluid objects and identify their analogous discrete objects.

Chapter 16 Applied Simulation Environment

Objectives

To describe how successful simulation projects are actually carried out in practice, the overall process involved, the importance of balancing both hard and soft skills, and all of the roles/people who take part and contribute to the effort. Also, to briefly describe the simulation software marketplace and the relationship between simulation and other software tools.

It is of paramount importance to have the results and recommendations derived from a simulation model accepted and implemented. After all, the power of simulation and all of the effort expended to build models and conduct analyses is lost if it is not used. The most technically elegant model is of no value if it does not support decision making in a timely manner. This chapter addresses the non-technical aspects of simulation modeling and analysis, including the importance of user confidence, understanding that simulations have a life cycle and involve a definitive process, and the many roles that must be fulfilled in order for simulation to be applied successfully. This chapter also discusses the simulation software marketplace, model-based decision-support systems, and the relationships between simulation and other software tools. It concludes with a set of recommended success factors that are critical in simulation projects.

Section 16-1 Confidence

When a person has a serious medical condition and a new technology is available, confidence is a major factor in whether or not the person chooses to use the new approach—not only confidence in the technology itself, but also in the person using the technology and in the process by which the technology is applied. Of course, a person who is desperate for help may choose the new procedure even without total confidence. Decisions to use simulation can be very similar. Simulation is a tool for analyzing and solving problems associated with operations systems. As with the medical example, it is a technology that may be new and

unfamiliar to those seeking help in solving a problem. Although an intense desire to find an answer to a problem may outweigh other considerations, confidence is critical to the successful use of simulation and to the professional success of the person using it.

Most of this book deals with the technical issues associated with simulation modeling and analysis (SMA). Of course, the model and resulting analysis must be technically sound for a project to succeed; however, that alone is not sufficient to guarantee success. For example, an elegant model may address the wrong problem, or the key stakeholders may have little confidence in the model, the person doing the analysis, or the results it generates. Any of these issues could cause the simulation project to be deemed a failure. It often takes a considerable amount of time and effort to get an organization to accept the use of simulation, yet that acceptance can be lost very quickly with a single, poorly run simulation-based analysis. It is necessary, therefore, that significant attention be paid to the non-technical issues involved in a simulation study and the way confidence is achieved.

Confidence in the technology

The first several chapters of this book provided numerous examples of how simulation is used to successfully solve problems. For some, seeing examples from a single source may not be sufficient enough to be convinced of its effectiveness. They may require additional examples from published papers, conference proceedings, or roundtable discussions with others in their industry. Senior managers may suddenly be interested in simulation based on a presentation at an executive conference. Building confidence in the technology requires educating others through examples.

Confidence in the person

Since simulation is a tool for solving problems, the primary way confidence is built in a person who uses simulation is through their problem solving ability. When that ability is demonstrated, with or without the use of simulation, people will have confidence in that person. For a person new to an organization, being aggressive and eager to solve problems will raise management's confidence in you. Being a successful problem solver means focusing on listening to people, finding the underlying issues, and responding in a timely manner. Even with a powerful tool such as simulation, the emphasis must be on solving the problem and not on the technology.

Confidence in the process

While simulation may, at times, be used solely by the person solving the problem as an aid to their analysis, it is more commonly used by a team as an integral part of the problem solving process on a large issue. In these cases it is critical that everyone involved in the process clearly understands the methodology and takes an active role in the process. Only when all are actively involved will there be confidence in the results of the simulation analysis. With such confidence, it will be easier to gain the support of those outside the simulation project who may be required to accept the simulation results.

Building confidence often begins with small steps. A good way to start is by observation and analysis, which can lead to the identification and resolution of a problem. The entire confidence building process can be summarized in the following statement, first put forth in print by Sir Arthur Helps in 1868 [5]:

Nothing succeeds like success.

In this chapter, we will define an SMA life cycle and process, introduce the use of decision support systems, define the roles that must be filled in an SMA project, and offer some key non-technical practices that can improve the chances that a simulation project is a success and delivers the most value. This chapter provides the outline of the process while the next chapter will provide the details of actually carrying out the project.

Whether the problem analysis is simple enough for one person or large enough that a team is involved, the lifecycle and process remain the same.

Section 16-2 The Simulation Modeling and Analysis (SMA) Life Cycle

When simulation is used to address a problem, it is not an event or an activity; it is a project. As such, simulation studies, like all projects, have a life cycle, similar to the one shown in Figure 16.1. Key phases in the life cycle can be described as follows: determining the objective(s), developing the model(s), performing analyses, implementing a change, assessing performance, and ultimately realizing results.

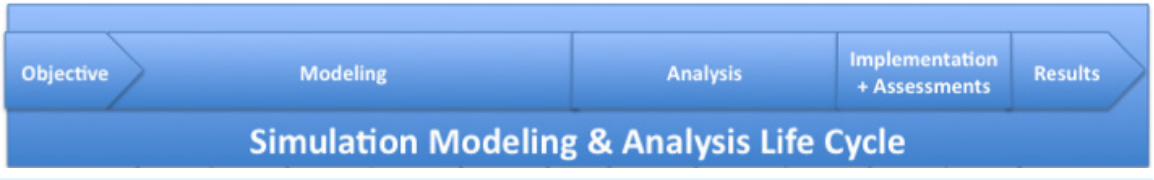


Figure 16.1: The simulation modeling and analysis (SMA) life cycle.

Normally, a project would conclude with implementing the suggested changes, assessing how well the project was carried out, and documenting the lessons learned. We include the results phase to indicate the outcome of a simulation may be relatively short-lived, as is the case when simulation is used to support an investment decision. Or, the results may continue for a long time with recurring benefits accruing from the use of a simulation model, as is the case when a simulation model is used routinely to enhance planning or operations. It is important to think about results right from the beginning, as Stephen Covey posits, "Begin with the end in mind." [4] Thinking about the outcome helps to set the objective, define the requirements, and assess the level of model detail that is needed to adequately address the problem.

Figure 16.2 identifies the roles of stakeholders that typically participate in a simulation project. While some of the roles may be carried out by the same person, it is important to understand all of the various roles in the project. For example, if simulation is used to make production planning decisions, the vice president of manufacturing may be the ultimate decision maker (on what to produce or where to produce it) as well as the sponsor (the one providing funds for the study and development); the production manager may be the

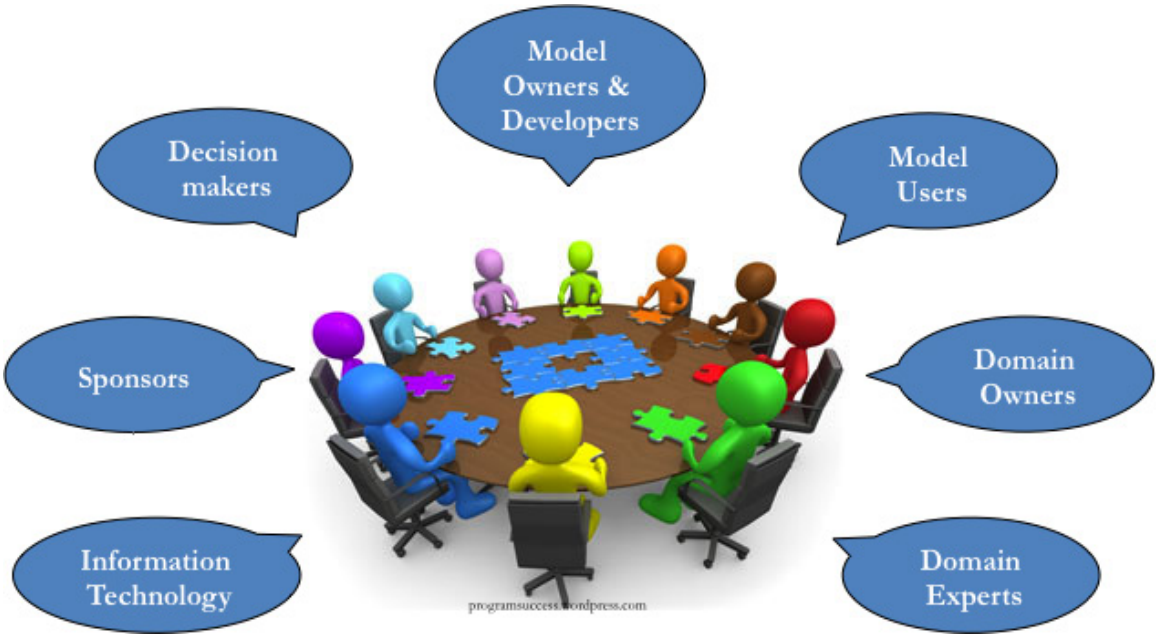


Figure 16.2: Stakeholders in a simulation project.

domain owner; the production workers and technical staff may be the domain experts; the model owners may be the industrial engineering department. The actual development may be shared between internal engineers and external consultants, and the model users may be production planners. Much of the data may need to be extracted from company databases, and thus the information technology must provide access and possible database support.

The example above illustrates that most simulation studies are multi-disciplinary, cross-functional projects and must be managed that way; focusing only on the modeling phase will likely lead to an unsuccessful project.

Section 16-3 The modeling and analysis process

Simulation modeling and analysis (SMA) is a process – a goal-directed set of related activities that contribute to results. Figure 16.3 defines the SMA process as a process-improvement process and represents it as a Deming or Shewhart Cycle, also known as the four-phased Plan-Do-Study-Act cycle. In simulation terms, the first phase is to define the system being considered for improvement and set the project's objectives. The initial phase is followed by building and validating a simulation model. In the third phase the model is used for experimentation and to evaluate alternatives. The cycle concludes with implementing the decided change and assessing project performance.

The simple process represented in Figure 16.3 can be expanded to include more detail on typical activities in a simulation project. These activities are shown in Figure 16.4. While each simulation study is a separate project, each involves a fairly standard set of interrelated activities. A critical first step is to clearly articulate the objectives of the simulation study, define the system being considered, and to develop a project plan. The project plan is basically a more detailed breakdown of the activities shown in Figure 16.4. The plan will

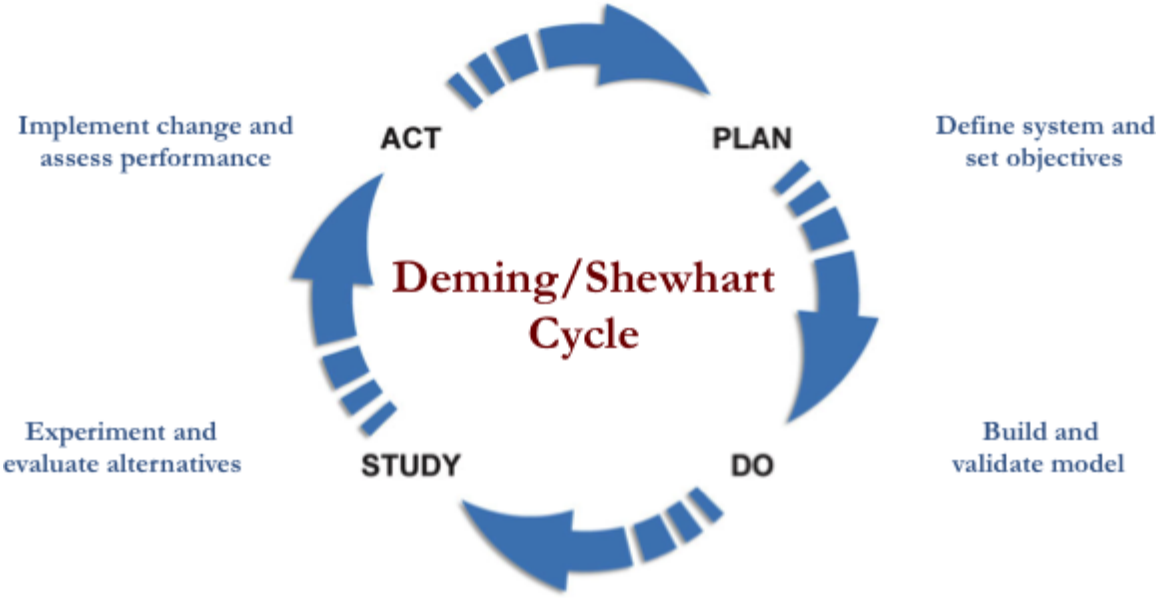


Figure 16.3: Simulation modeling and analysis as an improvement process.

typically indicate key milestones, provide estimates of the duration of each activity, and assign responsibility as to who leads and performs each activity. The plan itself is oftentimes summarized graphically in a Gantt chart. Figure 16.4 provides an operational view of simulation activities, similar to the way they would be represented in a project plan.

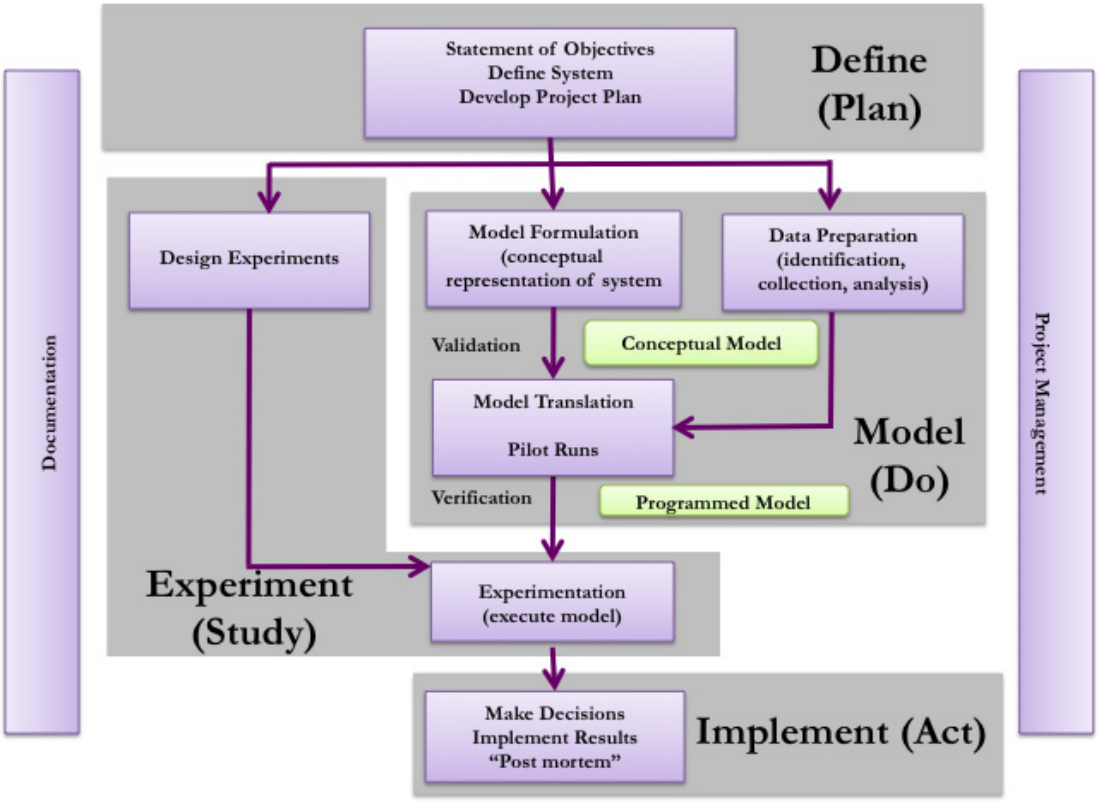


Figure 16.4: The simulation modeling and analysis (SMA) process.

The main portion of the SMA process can proceed along three parallel paths. The central path is model related and associated with model formulation and development. The two related paths are output or analysis related and input or data related respectively. Obviously as the project progresses, there needs to be good communication among the people performing the tasks along these paths—while the tasks may be conducted in parallel, they are not independent. Also, the tasks typically involve different stakeholders.

The modeling path begins with the formulation and development of a conceptual model, typically a high-level diagram that captures the key elements in the system and their relationships. Its purpose is to document how the system works, provide an understanding of how detailed the model needs to be to meet the analysis objectives, identify data needs, and uncover possible modeling challenges.

The conceptual model is a key means of validating the simulation model and addressing whether the right model is being built. Validation determines whether the model is a meaningful and accurate representation of the real system. The model's accuracy is closely linked to how the model will be used.

The programmed model (the one that is used for experimentation) is a translation of the conceptual model into computer-executable form, using such commercial simulation software as *FlexSim*. Prior to experimentation, the programmed model must be verified; that is, it must be determined that the model is working as intended. In this step, the focus is on building the model right.

The experimentation results are the primary input to decision making. Once an alternative is selected, it is implemented in the real system. As with good project management practice, the simulation project should be reviewed with all stakeholders via a "post mortem," to assess what aspects went well and what needs improvement. This is an often-overlooked step, but it is crucial to building confidence in simulation within the organization.

There are two activity boxes on either side of the main process flow in Figure 16.4. The one on the left—documentation—indicates that the model and project need to be documented throughout the process and not delayed until the end. Most people don't like developing documentation, but it is extremely important for future modifications to the model, as well as explaining rationale, identifying and justifying assumptions, providing details of model logic and analysis, and capturing lessons learned. The activity box on the right side of the main process flow—project management—emphasizes the need to use good project management practices throughout the study.

Figure 16.5 shows the main technical aspects of a simulation study. It also indicates that the use of software is critical in conducting a simulation project. The technical focus includes simulation model development, the details of which make up the majority of this book. Models are developed through commercial simulation software packages available in the marketplace. This book's focus is obviously on making the translation from conceptual models to programmed models through *FlexSim*. Ideally, the computer models are based on solid, software-neutral conceptual models that depict the key elements in a system and their relationships.

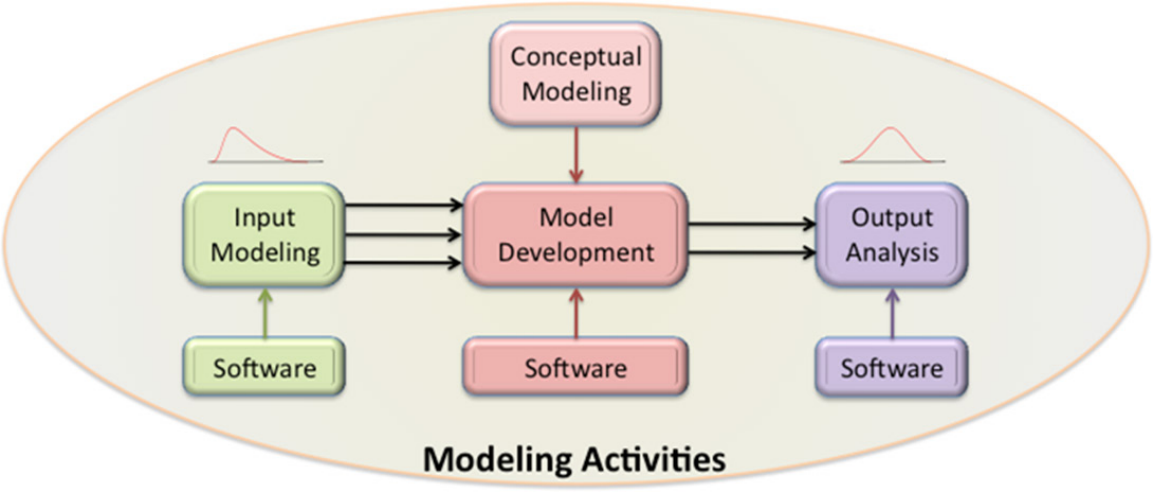


Figure 16.5: Primary technical aspects of simulation.

Since simulation models are data driven, considerable time is dedicated to input modeling, such as characterizing input data as probability distributions. As Figure 16.5 indicates, these activities are supported through software as well. The software can be special purpose, such as *ExpertFit*, or general-purpose statistical analysis packages like Minitab, SAS, or SPSS.

The main reasons to build models are to conduct analyses, increase understanding, and obtain results; therefore, considerable effort is associated with analyzing output obtained from simulation models. These activities are also supported by software. All of the major commercial simulation software packages contain means to capture and analyze the data output from simulation models, such as the Experimenter in *FlexSim*. Depending on the analysis needs, these built-in capabilities can be supplemented with special purpose software, such as those mentioned above, for input modeling. In addition, if the simulation model is used to drive an optimizer, then special purpose software, such as *OptQuest*, may be used.

Section 16-4 Roles in a SMA project

Too often the roles of individuals in an SMA project are considered solely within the nomenclature associated with the project activities described above. As a result, the only roles that are explicitly addressed are those that deal with model construction, and maybe analysis; however, there are many roles that can be valuable to a successful SMA project. Of course, some individuals may fulfill a variety of roles. Those roles are defined below.

Developer: Develops and constructs the simulation model and/or model support software, such as database connectivity, user interfaces, decision support systems, etc.

Designer: Establishes the model and/or decision support system architecture; this role is closely related to that of developer.

Analyst: Designs (e.g., defines what variables need to be changed in each scenario and their value, determines run length, warm-up period, number of replications, etc.), conducts, and summarizes experiments.

Researcher: Conducts research early in a project to see if similar issues have already been addressed elsewhere and what can be learned from previous work.

Investigator: Probes and "digs" to clearly define the processes and investigate all the available data in order to ensure the system is clearly understood and adequately represented.

Educator: Helps stakeholders involved in a simulation project understand enough about SMA to be an effective contributor; this is necessary because stakeholders have diverse backgrounds and oftentimes have limited expertise in simulation.

Implementer/user: Converts the results from the simulation into actions and changes; this includes planning, obtaining buy-in, and follow-up. If results from the SMA project are not implemented, then it has added little value to the organization and can in fact be considered a liability.

One way to clearly articulate who is responsible for performing what role(s) is to construct a responsibility matrix early in the project. As shown below, a table is constructed that identifies key project activities associated with each role and identifies the specific project member who is responsible. Oftentimes, more than one person is responsible for different activities and roles. When this is the case, one individual needs to be indicated as the "lead" so that he or she can coordinate and properly manage the work.

Role	Activities	Responsibility

As should be evident from this discussion, simulation projects involve people. In fact, SMA projects require a balance for success – a balance between technical skills (sometimes referred to as hard skills) and people-related skills (sometimes referred to as soft skills). Each is necessary, but each by itself is not sufficient, for success.

When dealing with people, it is important to remember that they have needs, concerns, suspicions, and biases. This is especially important in simulation studies since many stakeholders are not familiar with simulation. Their lack of knowledge is likely to result in: (1) serious doubts that the work they do could possibly be modeled, (2) skepticism about the model results, and (3) fear that the project will significantly affect their work level, performance, and even their job itself.

People also have expectations, which need to be managed. Care must be taken not to oversell what simulation can do. It must be clear to all stakeholders that simulation will

not solve their problems or make decisions for them – it is an input, albeit a valuable input, to the problem-solving and decision-making processes. It is important to stress that simulation involves models that are representations of systems and that no simulation can capture all aspects of a system. Therefore, it is important that assumptions are clearly stated.

People have expertise and thus provide most of the understanding and information that is the basis for any simulation model. For example, they provide, or at least strongly influence, the project definition, objectives, and constraints. As discussed further below, they are the key providers of domain knowledge that is required to build a simulation model. They are also the ones who validate a simulation model and the output from the model. Validation of the model is necessary for credibility and validation of the output is necessary for confidence.

It is important to value the information that stakeholders share with the modeler. Most systems are too complex for any one person to know everything about it. Therefore, information must be gathered from multiple sources and assimilated. Inconsistencies in information must be identified and reconciled. Knowledge and data are sources of power, so they may be difficult to obtain. Also, as mentioned above, people may feel threatened by a project and therefore may not be forthright with information. All these issues must be dealt with in a simulation study, thus it is important that the project team include a people with a variety of skills.

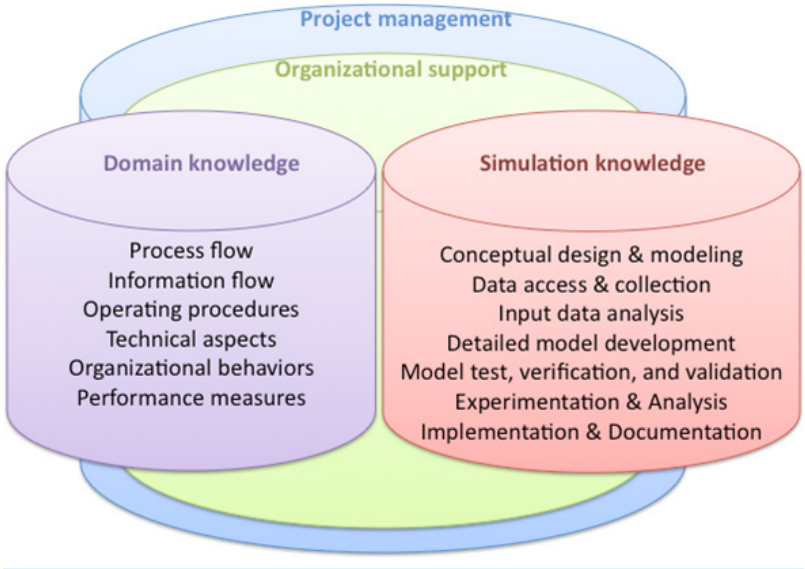


Figure 16.6: Knowledge needed in a simulation project.

As Figure 16.6 illustrates, SMA projects involve two types of knowledge: domain knowledge and simulation knowledge. Domain knowledge is specific to the system being modeled and is usually provided by the organization where the work is being done. A key part of domain knowledge is defining how a system works – its process and information flows, operating procedures, equipment layout, etc. It also includes data that will drive the simulation model, e.g. system parameters, schedules, and raw data that are summarized into distributions using analysis. Domain knowledge also includes the definition of performance measures – those variables that are used to compare alternatives and define what "better" means.

Simulation knowledge is acquired through books like this one, courses, and practice. It includes conceptual modeling, detailed model development, input data analysis, experimental design and output analyses, etc.

The figure also illustrates that SMA projects must have organizational support and must be managed as projects. Good project management includes defining clear objectives, developing a plan, effectively engaging stakeholders, monitoring performance, etc.

Section 16-5 Model-based decision-support systems

Most applications of SMA are to support decisions. These may be one-time decisions, such as selecting the best layout for a new facility, choosing the best policies or rules to use under varying conditions, exploring whether a proposed change is feasible, etc. On the other hand, the decisions may be ongoing, such as selecting where to produce certain products, deciding whether an order can meet the desired delivery time, identifying the best sequence to process a set of products, etc.

Oftentimes in one-time analyses, simulation practitioners or advanced users are involved and are the ones interacting with the simulation software. In this case, the user is knowledgeable about simulation and comfortable with the software. However, that is not usually the case when simulation is used on a regular basis to make decisions. In this case, it is often impractical to have the user be very knowledgeable about simulation; therefore, the user needs assistance in interacting with the simulation model.

This assistance involves providing, at a minimum, a friendly graphical user interface (GUI) that enables the user to input data easily (e.g., through a form or spreadsheet), and obtain information in a familiar format (e.g., graphs or spreadsheets). The assistance can be much more extensive, as illustrated in Figure 16.7. In this case, the decision support system (DSS) not only provides interfaces to the user, but to schedule, product, and process

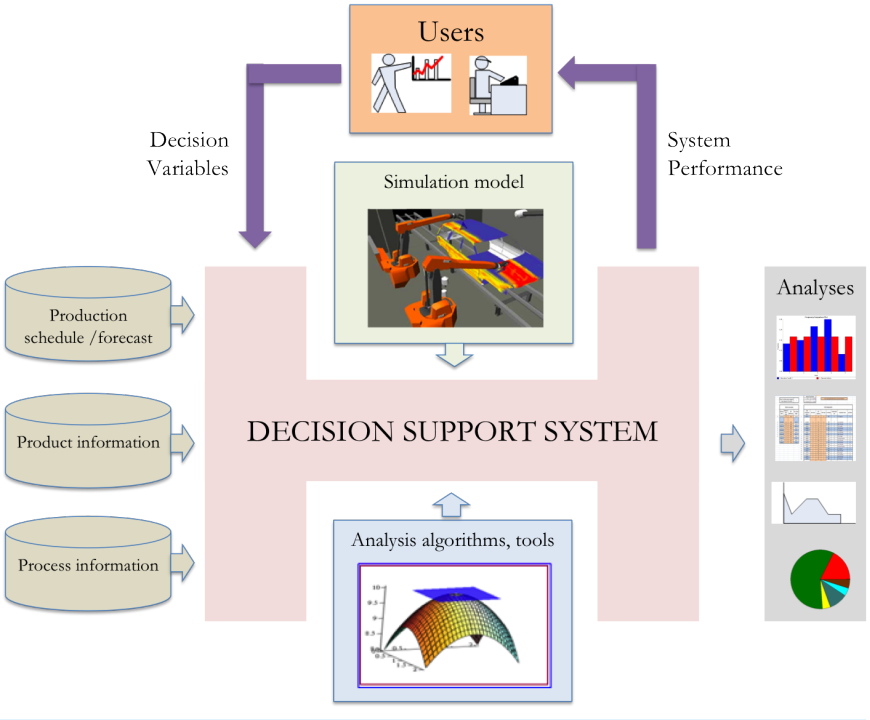


Figure 16.7: Components of a simulation-based decision-support system.

databases as well. It also interfaces with analysis routines and algorithms, such as optimization, in which the simulation model is used to provide feedback to an optimizer in order to help guide its search for better solutions.

An example of a decision support system is shown in Figure 16.8. This application was developed by Mississippi State University to help decision makers assess the impact of operating under either a batch or lean manufacturing approach.

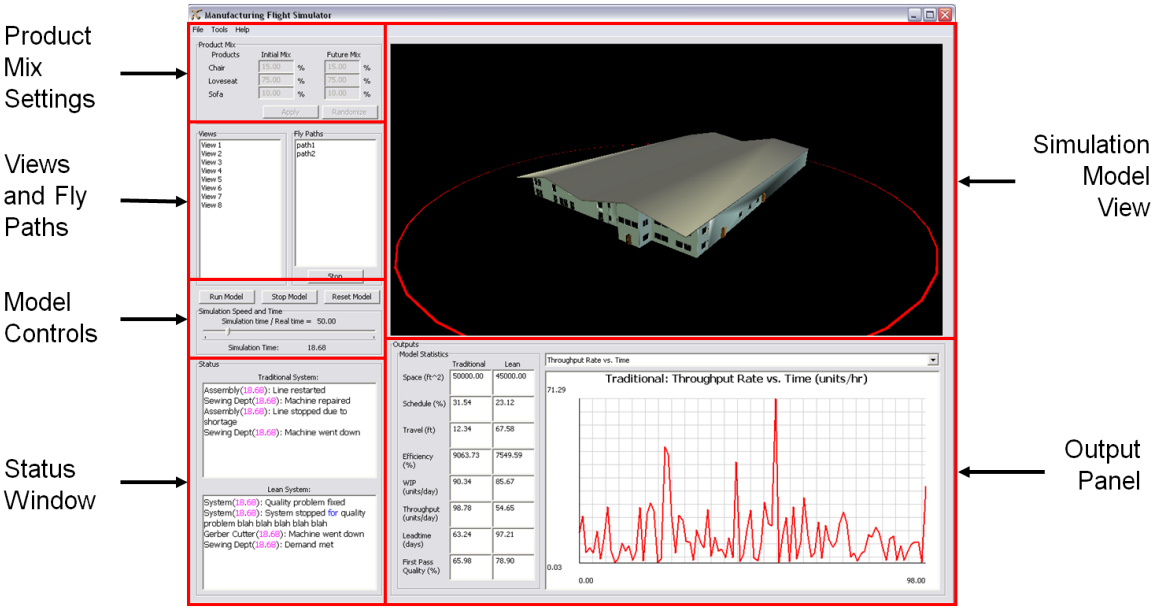


Figure 16.8: Example simulation-based decision support system [3].

In this single user interface DSS, the user, who needs minimal understanding of simulation,

- inputs data on the left side of the interface;
- observes output from the model via a graph and table at the bottom of the interface; and
- views the model execution in the center window.

This DSS was developed all within *FlexSim* using its GUI builder.

Section 16-6 Simulation and other tools

Simulation is only one of a number of tools available for effectively studying dynamic systems. Other types of tools used extend from directing day-to-day operations to planning changes to meet new business opportunities. It is difficult, if not impossible, to find one tool that easily satisfies the needs of all users across such a wide range of activities; however, all of these tools share in the objective to improve system performance as they also share in the data and metrics that drive operations. It is therefore critical that the tools are able to share

information across applications as well as organizational boundaries. Figure 16.9 shows the interactions between the various organizations and the tools that support the operations.

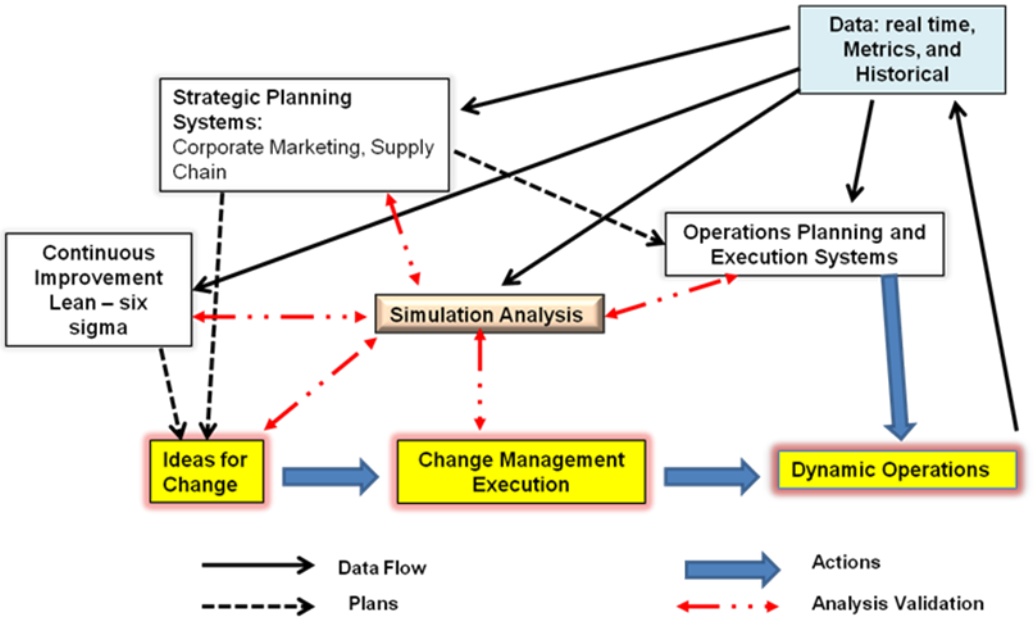


Figure 16.9: Interaction between simulation and other applications.

The Dynamic Operations block can be a production line, a plant, a service organization, or even a single operation. It is this operation that is the focus of the improvement efforts. Understanding its dynamic behavior and performance measures is the underlying goal. Two forces directly change the operating characteristics of the system. The first force is the day-to-day directives for operation.

These may come from a direct response to economic or market conditions or from events causing the system to react. The second force is the introduction of new products or services, as well as changes made in hopes of improvement.

There are many tools and methods available to direct the planning and execution of day-to-day operations. Major corporations use enterprise software applications such as those included in the *SAP™* family of enterprise resource planning software. More focused applications such as *Supply Chain Guru™* are also available. These applications monitor the current state of operations, inventories, and sales to determine what should be produced on a specific day at a particular location or manufacturing line.

Understanding the roles of various applications

In a small operation or service area, operational strategy can be as simple as having someone with experience in the operation using personal judgment to make decisions, or it can be as complex as requiring a software application to make decisions.

In either case, the decision method usually involves some form of simplified model of the operations. The model may be a mental model or a mathematical model and is used

to transform business or demand inputs into operating decisions. Because models at this level tend to be generic in nature and do not incorporate the dynamics and interactions that exist, operating decisions will likely not be optimal for the enterprise and may actually increase the cost of producing the goods or services.

Another basic force that impacts an operations system is the physical or procedural changes to the system that are caused by the introduction of new methods, equipment, projects, or goods/services. While some of these changes can be executed with little effort by people closest to the operation, others involve significant resources and time. There may also be multiple changes being considered, thus requiring priorities to be set. Each such major change would require a justification for the cost of capital and possible disruption of operations. Most organizations have some form of change management process to oversee these projects.

Project concepts come from a number of sources. The adoption of Six Sigma, Lean, and Continuous Improvement programs has resulted in people directly involved with operations across organizations recommending changes to improve performance. These changes are based on observations as well as analysis of real time and historical operating data. Tools developed for these programs are simple to use and do not involve operational dynamics or interactions. Change recommendations are typically local in nature. When local solutions are applied to systems that interact with others or have operational complexities, they may not be, and typically are not, optimal for the broader system, such as the enterprise.

Project concepts to modify operations that come from business or marketing groups tend to be expansive and costly in nature. These concepts are derived from model-based tools that reflect business, market, and logistic conditions. The driving force for justifications is heavily weighted toward business revenue. Operations may be modeled but oftentimes at a very basic level, usually only considering averages of rates and costs. Changes to operations are often major in scope and have to undergo various levels of justification. Their validation of operability (i.e., degree of confidence that the concept will actually work as planned) is usually minimal. These high-level concepts compete for resources proposed at lower levels in the organization that result from Continuous Improvement activities.

Simulation plays both a direct and supportive role in all decision-making activities that potentially impact the operation of a currently operating system. By using real time and historical operating data, a simulation analysis can generate ideas for change and improvement. Because the simulation characterizes the dynamic interactions within an operation, it is easier to visualize the impact of making changes. Simulations are most often created to seek answers to specific operating issues that seem to defy simple or conventional analysis techniques.

During the change management or project execution effort, simulation is used directly to analyze designs and estimate performance. A simulation can be used to determine the number of operating lines required or the optimal size of surge or Kanban areas. Simulations also check the control logic assumptions for dealing with various pieces of equipment, such as conveyors. The staffing levels needed for operations and equipment loading can be determined through the use of simulation. Performance of transfer systems for raw materials, warehouse layout design, batching control logic, and packing line analysis are

all examples of simulation's direct involvement in supporting project design and analysis activities.

Simulation can also play a much broader role, in that it can be used to validate opinions or ideas. Simulation is used by companies with active Lean programs to both validate and help prioritize competing ideas that result from process improvement teams. Simulation also indicates where local optimizations may be affected by other areas and consequently will not provide the desired result. Lean teams view simulation as a quick and easy way to test concepts.

When simulation is used at an early stage with business and marketing groups, it can eliminate major projects that may not meet expectations. On the other hand, major projects are also often justified through the use of simulation. Outputs from simulation provide a more accurate basis for return on investment calculations since it takes operations into account as well as interactions and variability. As a validation tool, simulation helps in the prioritization process where competing concepts and projects are involved.

Simulation is used to validate the results of execution and operations planning systems. Planning and scheduling tools that produce multiple week schedules do not necessarily take into account all of the operating factors. As a result, the schedule for an operation may meet some specified business goal but may increase the cost of operations. In most cases, business and operating organizations are independent and wish to maximize their profitability. By simulating the proposed schedule, operating costs can be more accurately estimated. By comparing schedule options with operating costs, a closer-to-optimal plan is possible since it addresses both business and operational requirements.

By interacting with all the various activities that impact operations, simulation plays the important role of a common communication and continuity tool. As Babin and Greenwood [1] point out, simulation builds collective understanding of the process being improved. From idea concept to daily operation, simulation provides a means for people across an organization to visualize, analyze, and discuss operations and change concepts from a common basis Discussions of priority or value can be compared on an equal basis. Rather than trying to discuss issues based on anecdotal experience, people can base their analysis on simulated results of the actual and proposed processes.

Section 16-7 The simulation software marketplace

Early commercial simulation programs focused on specific areas, such as discrete-event simulations or fluid/continuous simulations. Also, because of the computer and software technology available at the time, they were not user-friendly and required a long learning curve. Consequently, groups of simulation specialists were formed to develop simulation models and analyze the results. Many companies had large groups of simulation practitioners who focused on detailed simulations. When more individuals within the organization wanted to make use of simulation technology, they were disillusioned by the learning curve of the applications; the license costs; the time to complete a simulation; and the placement of a separate group between them, the process, and the problem they wanted to study.

Simulation languages

As early as the 1960s and 1970s, simulation languages and simulators were being developed. High-level languages such as *SIMULA*, *GPSS*, *GERT*, and *SIMSCRIPT* started to replace the lower-level language of *FORTRAN* as the language of choice for simulation. These simulation languages were designed to have constructs that made it easier to develop simulation models.

When developing simulation programs using these early simulation languages, the modeler had to use language statements, or macros, to specify the system. Data were passed to these constructs through a set of arguments, and the modeler had to understand what each parameter in the arguments list meant and keep track of the system state variables. The language constructs or macros were not very robust and oftentimes required writing supporting code in *FORTRAN*. Simulating systems that required anything more than the basic functions could take a long time and be very expensive to develop.

Early models that utilized simulation languages did not have animation. The first language to add animation was *SIMAN*. *SIMAN* had to use a separate second language called *CINEMA* to build animations. *SIMAN* and *CINEMA* were later combined and called *Arena*.

The first simulation software products for DOS-based PCs were primarily for commercial use and were not widely used or even available until the early 1990s. By today's standards, simulation products during these early days were hard to use, had very limited graphic capability, and pushed the limits of the computers. 3D animation was virtually unknown and required very expensive workstation computers that could cost up to $60,000. *AutoMod* and *QUEST* are examples of early simulation packages that could support a level of 3D animation. The cost of hardware and software to run the system could approach $100,000. Even then, the model ran slowly and required extensive programming.

In 1986, *WITNESS* released the first product with higher-level prebuilt constructs that also had a visual representation. These advanced constructs further reduced the need for direct programming in simulation languages. The new generation of simulators were easier to use, but still lacked flexibility. Modifications to the default constructs were either very difficult or impossible to accomplish because the computer code behind the construct was not available to the user. By the late 1980s, there were several PC-based simulators on the market, including *WITNESS*, *ProModel PC*, *SLAM*, and *SIMFACTORY*.

Simulators

In the 1990s several newer simulators began to appear in the market. In Europe there was *Simple++*, *Simul8*, and *Taylor II* (the first 3D simulator to run on a PC). Several of these simulator products are still being used today as a result of the availability of textbooks and student versions.

In the United States, where the majority of simulation software was used commercially, several small companies were introducing a variety of products. Some products were short

lived, and others are still being used today. The more notable ones are *Extend* and *Simcad*. In 1998 *Taylor ED* was released. It was the first object-oriented 3D simulation software that had virtual reality 3D. It also operated on the Microsoft Windows software platform.

FlexSim was released in January 2003, and proved to be substantially different from previous simulators. It was also a departure from the standard simulation language paradigm both in its usage as well as its underlying architecture.

While these simulation languages and simulators were primarily focused on discrete-event simulations, applications for the fluid/process industries, such as chemical plants and oil refineries, started to appear in the market.

For the fluid industries, the fundamental principles of dynamic material and energy balances and computational fluid dynamics were used to create time-variant simulations of very specific operations. These applications are used to simulate such conditions as temperature profiles in jet engines, moisture migration in dryers, and pressure profiles in extruders. *Aspen™* became the premier simulator for the chemical and refining industry. That application creates detailed simulations of transformational events associated with chemical processing.

Hybrid Simulation

Not all simulated systems fall exclusively into either the discrete event or fluid categories. For example, many fluid operations contain a batching or blending step where the products' components are added to and leave the equipment as fluids; however, they are then placed into discrete units such as bottles or boxes. Operations that combined both approaches are termed "hybrid" systems. An example hybrid system is a bottling line, as shown in Figure 16.10.

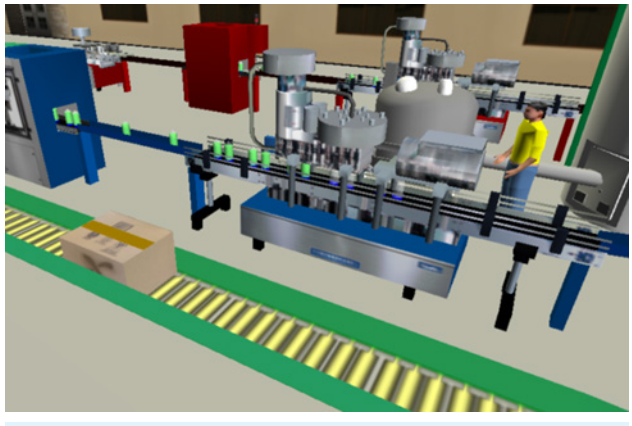


Figure 16.10: Bottling line is both fluid/continuous and discrete

Some simulators have the capability to handle both discrete and fluid events in the same application. *FlexSim*, *Extend*, *Arena*, and *WITNESS* are examples of such applications. They all use different techniques to simulate continuous, fluid operations in a primarily discrete-event simulation environment. The main characteristic of a fluid operation is the specification of material processing in terms of the flow of material per time unit (e.g., gallons/minute) rather than as a discrete cycle time.

Additionally, *fluid* might not mean a liquid. In the production of potato chips, the flow of chips is characterized by a flow in chips/minute or pounds/minute. In high-speed packaging operations, fillers are rated at bottles/minute. Trying to simulate each individual piece of

material in either of these situations would be very time consuming and typically would not add much value to the analysis. Modeling fluid, or continuous, systems are discussed in Chapter 15.

The Trend toward "Ease of Use"

Since the year 2000, the simulation marketplace has dramatically changed. Companies, seeking to reduce costs, eliminated their simulation groups with their high overhead costs. While still seeing simulation as a valuable technology, companies found it difficult to justify the overhead associated with maintaining a separate simulation organization. The combination of a new level of simulation software and more powerful, less expensive computers helped to create a change in the marketplace. The new key attribute that modern simulators strive to achieve is known as *ease of use*. This trend has helped simulation to become a more common tool that is no longer focused on simulation professionals in large corporate groups.

While *ease of use* is basically a marketing term, there are a few basic characteristics generally associated with the term; some of these are provided in Figure 16.11.

Simulators that are considered easy to use incorporate modeling procedures that are intuitive and maintain consistent terminology and interfaces. Following a Windows approach to file structure helps to accomplish this, as do user interfaces that can be customized to reflect the user's level of simulation experience.

Visualization is an important factor. While not always required, a good three-dimensional representation can aid understanding of the simulation by managers, operators, and others who are not involved in building the simulation. Visualization also makes it easier for the simulation builder to compare the simulation to actual operations.

Training is a major issue for software. Many users are familiar with basic computer applications and gaming systems. They expect to be able to use a program without reading the instructions. Simulation software certainly requires training, but it can be done at a variety of levels that are aligned with users' roles in the simulation project. Training should be short in order to get a large number of users started with a base, then that base should be easily built upon as users gain experience. For users who do not use simulation every day, refresher training is often needed. This level of training

- Intuitive operation
- Consistent use of terminology
- Short training and re-training periods
- Allows the focus to be the application/problem
- Robust
  - Multiple user capabilities
  - Fluid and discrete modeling
  - Basic and advanced programming
- Robust
  - Models
  - Individual constructs/ objects

Figure 16.11: Ease of use characteristics.

should also be minimal. Simulators that can be used with little training, free the user to focus on the system being simulated rather than on the mechanics of using the simulator.

Simulators in the current marketplace also try to be robust – meaning that while they are easy to use for the new user, they can also provide advanced features for the expert user. Robustness also means that the simulator can handle fluid, discrete, and hybrid simulations in a consistent manner.

Reusability is a key factor in keeping engineering costs low. Development work that is done for one application should not have to be duplicated on another. The ability to create and keep partial and full versions of simulation models is important. Of course, if the simulator is not easy to document, then past work may be lost because no one will be able to tell how it was built.

FlexSim was selected as the simulator for use in this book because of its robust architecture and ease of use. It was used by Dr. Malcolm Beaverstock, the first author, to establish simulation as a common tool for use within General Mills by individuals with various levels of simulation expertise. It has also been used by Dr. Allen Greenwood, the second author, and the simulation group at Mississippi State University to help industry address a wide range of problems through the use of simulation.

Section 16-8 Simulation modeling and analysis success factors

In summary, the following are a few key non-technical practices that can help simulation projects be a success:

- Manage the simulation modeling and analysis activities as a project.
- Define clear objectives early and get buy-in from stakeholders.
- Define the basis for comparing alternatives (key performance measures)
- Set realistic expectations and manage them.
- Identify early how the simulation will be used and by whom; understand the users' needs.
- Obtain management support.
- Involve stakeholders throughout the process; gain their acceptance.
- Document definitions and assumptions.
- Test, validate, and verify the model.
- Understand the data involved (its availability, format, reliability, quality, etc.).
- Start simple; add complexity as needed.

The References section contains some articles that discuss many of the success factors in more detail. They include Balci [2], Hugan [6], Jurishica [7], Law [8, 9], Sargent [10], Shannon [11], and Sturrock [12].

References

1. Babin, P, and Greenwood, A. "Discretely evaluating complex systems." *Industrial Engineer*, February 2011, vol. 43, no. 2, pp. 34-38.

2. Balci, Osman. "Guidelines for Successful Simulation Projects," *Proceedings of the 1990 Winter Simulation Conference*, Osman Balci, Randall P. Sandowski, and Richard Nance, eds. pp. 25-32.

3. Center for Advanced Vehicular Systems – Extension, "A Decision Support System for Rapid Comparison of Lean-to-Batch Systems," Mississippi State University, 2007.

4. Covey, Stephen R. *The 7 Habits of Highly Effective People*, New York: Free Press, 1989.

5. Helps, Albert Sir. *Realmah*. 1868.

6. Hugan, Joseph C. "A Practical Look at Simulation Project Management," *Proceedings of the 2014 Winter Simulation Conference*, A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds. pp. 98-102.

7. Jurishica, Carley and Zupick, Nancy. "Tutorial: Tools and Methologies for Executing Successful Simulation Consulting Projects," *Proceedings of the 2012 Winter Simulation Conference*, C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds. pp. 95-107.

8. Law, Averill M, and McComas, Michael G. "Secrets Successful Simulation Projects," *Proceedings of the 1991 Winter Simulation Conference*, Barry L. Nelson, David Kelton, and Gordon M. Clark, eds. pp. 21-27.

9. Law, Averill M. "How to Build Valid and Credible Simulation Models," *Proceedings of the 2009 Winter Simulation Conference*, M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and B. G. Ingalls, eds. pp. 24-33.

10. Sargent, Robert G. "An Introduction to Verification and Validation of Simulation Models," *Proceedings of the 2013 Winter Simulation Conference*, R. Pasupathy, S.H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds. pp. 321-327.

11. Shannon, Robert E. "Introduction to the Art and Science of Simulation," *Proceedings of the 1998 Winter Simulation Conference*, D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, eds. pp. 7-14.

12. Sturrock, David T. "Tutorial: Tips for Successful Practice of Simulation," *Proceedings of the 2014 Winter Simulation Conference*, A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds. pp. 90-97.

Chapter 16 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the three areas of confidence that are important for getting simulation to be used to support decision making in most organizations.
3. Discuss why it is a necessary, but not sufficient, condition for simulation project success to have a technically sound model and analysis.
4. Discuss the key phases of the SMA Life Cycle and describe the stake-holders involved in the life cycle.
5. Outline the key activities that need to be performed in a simulation project and discuss the relationships among the activities.
6. Define conceptual model and identify the benefits of constructing a conceptual model.
7. Discuss the differences between a conceptual and programmed model.
8. Discuss the differences between validation and verification.
9. Identify the main reasons for building models.
10. Discuss the various roles that are prevalent in a simulation project.
11. Define the two main classes of knowledge that are key in a simulation project and identify specific knowledge areas within each.
12. Discuss the types of assistance that a decision support system can provide to an occasional user.
13. Describe how to employ key non-technical practices in a simulation project to help ensure that it succeeds.
14. Describe some of the changes in simulation software and in the simulation marketplace that have occurred over the years.
15. Identify some of the characteristics of "ease of use" as it applies to simulation software.
16. Describe some of the ways simulation interacts with other applications.
17. Describe how simulation can be used in conjunction with lean and continuous improvement tools.
18. Describe three key success factors for a simulation project.

Chapter 17 Managing a Simulation Project

Objective

To describe a set of tools that support the modeling and analysis process, including a simulation project template, a diagram for conceptual model design, and a means to initially size resource requirements.

The previous chapter provided a general outline of how the Simulation Modeling and Analysis (SMA) process is used for analyzing and solving problems with simulation. It highlighted the roles/people and organizational issues involved with the successful use of simulation. This chapter focuses on the technical details of carrying out the SMA process and provides tools that can be used in the process. Those tools include the Object Flow Diagram for conceptual model design, initial resource sizing, and a structure for documenting and managing a simulation project.

Section 17-1 The starting point

An organization that needs to use simulation for analysis must follow a series of steps in order for the simulation to be successful. Building a simulation involves translating reality into a time-based model. As illustrated in Figure 17.1, when individuals look at the actual system, they develop their own conceptual model of how the system operates. Each person may see the system slightly differently or notice different characteristics—all of which may be important. Consequently, a methodology is needed to aide in communications and general understanding.

The Occasional User may not be directly involved with the actual building of the simulation but will be in a position to define what the simulation should accomplish and then possibly carry out simulation runs for analysis. Knowledge of all the steps involved with a simulation project is critical for assuring the project and results are valid.

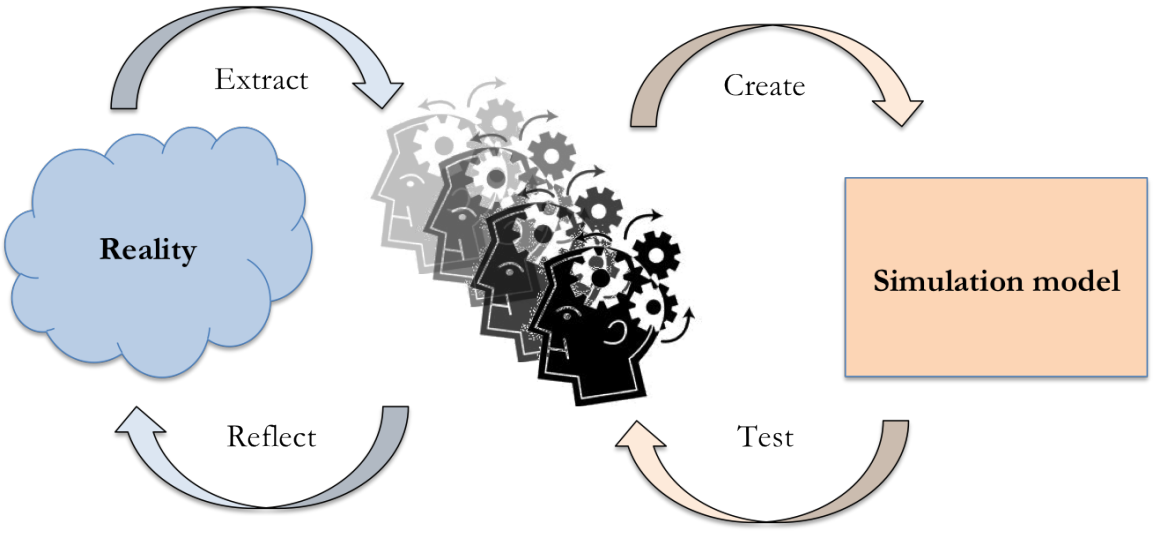


Figure 17.1: A simulation is a representation of stakeholders' reality (Adapted from [4], p. 18).

Simulation project steps

As introduced in Chapter 16, a simulation project is a joint effort between those wanting the simulation and those developing it. The result will only have real value to the people who are involved with the work.

There is a straightforward methodology for defining and carrying out a simulation; this methodology will assure that the resulting model will be of value. It requires working through the steps shown in Figure 17.2; the level of detail for each step will depend on the complexity of the system.

As was also discussed in Chapter 16, the general structure of this process follows Deming's Plan–Do–Study–Act cycle. The first phase is all about planning and understanding. The second phase involves abstracting the system's behavior into model form. The third phase concerns using the simulation model for analysis or study. The final phase is action oriented; that is, it involves gaining insight and information and deciding how to improve system performance based on various experiments conducted with the model.

Activities performed during these steps should be documented using the Simulation Project Template found at the end of this chapter.

Documentation is essential for helping all the individuals working on the simulation to have the same view and understanding of the project. The project template was specifically developed for defining and managing actual simulation projects. The first item in the project template is the project name. The simulation should be given a name that has meaning to the team members and others. The simulation models should contain the name followed by the current version number (e.g., BankSim1).

Since the simulation project is normally approved and carried out for a particular purpose, a summary of the results is usually required. This executive summary is normally limited

1. Define the system and the problem
   - Review existing and proposed facilities and processes
   - Establish goals and objectives
   - Define system and model assumptions
   - Collect data
   - Prepare a simple diagram of system elements

2. Build the simulation model
   - Establish basic parameters for the simulation (e.g., time units)
   - Define and develop user interfaces
   - Layout objects on the screen (simulation surface)
   - Define where and how data will be used
   - Define and implement logic

3. Use the simulation model for analysis
   - Validate and verify the simulation model.
   - Design and run experiments
   - Analyze simulation outputs

4. Use the results to develop an action plan
   - Draw conclusions, present results and recommendations
   - Gain consensus for action plan and implement it.

Figure 17.2: Simulation project steps.

to one page or one slide and focuses on the reason for the simulation and the result of the analysis. The three main parts of the summary are as follows:

- Background (reason for the simulation and objectives)
- Results
- Conclusions and recommendations

Section 17-2 Define the system and the reason to simulate

The first phase of a simulation project involves the critical task of actually defining the project. This definition and the associated material are often referred to as the functional

specifications. The functional specification can be the basis for requesting a quote from a third party to build the simulation.

This phase of the process involves listening to those who are responsible for the problem in question. It also means reviewing the facilities and the processes that will be represented by the proposed simulation model. This review can include a tour of an existing facility, and/or a review of design and operations documentation, especially for a proposed system if it doesn't already exist. In either case, everyone involved should have a complete and consistent understanding of the operational characteristics of the system. The level of detail at this point helps to focus the effort.

Several things need to be accomplished in this review:

- *Tour existing facility*: Even if simulation project team members work in this facility, a complete review is still helpful.
- *List system components*: During the tour, list all the system components including machines, storage locations for work in process, work tables, tool locations, or anything that is used to produce the product. Product components, parts, and assemblies should also be listed.
- *List system resources*: Also list all system resources, including conveyors, fork trucks, AGVs, maintenance personnel, machine operators, overhead cranes, robots, or any other resource that is used for production.
- *Make note of operational characteristics*: As each system element is reviewed, make notes of the operational characteristics. Is there any special processing logic? How is work selected? How do equipment and people interact with each other and other elements of the system?
- *Make a list of system terminology and acronyms*: Make a note of names and acronyms that are used to describe equipment or parts; using these names will make the model familiar to all who know the process.
- *Draw or obtain a drawing of the system layout*: Before the tour, obtain a drawing of the facility, or while on the tour draw the facility and make notes on the drawing for each element in the system. After the tour is complete, this drawing will serve as a "trip to the floor" whenever you need to review the facility. Photos are also helpful, if allowed. Also, drawings can be used as backdrops for the simulation; that is, a drawing can be imported into the simulation software and the model developed to scale on the drawing image.
- *Get to know the system in detail*: One of the best ways to get to know the system is to spend time in the operations area and discuss the process with the operators—those that do the work. Remember, the

more you know about the processes and capabilities of the system, the more capable you will be in modeling it correctly.

What is the objective?

Determining the objectives of a simulation project is a very important step since the simulation model will be designed around and focused on these objectives. It is important to realize that the objectives themselves may be the only understanding that management and others will have of the project. While the other steps will help manage the project, the objectives will manage the expectations.

The specific goals and objectives that are identified will be the criteria for success. The goals may be more general statements, but the objectives should include some specifics.

The following questions should be addressed and answered early in the simulation process as part of determining the project objectives:

- What is the primary reason for developing the simulation model?
- What is the value of performing the simulation analysis?
- What is the scope of the model?
- What questions will the simulation project answer; what problems will be solved?
- What issues need to be investigated?
- What are the appropriate performance measures (e.g., throughput, overall production time, resource utilization, system delays)?

What is the output?

The objectives indicate the performance metrics, but the format of the response should also be specified. This may include a mockup of how the final reports should look and what should be emphasized. The precise details will emerge during the project activity.

What is being simulated? (boundaries and assumptions)

A simulation model should contain only as much detail as is necessary to fulfill the objectives. Too much detail burdens the model building, slows the model run time, complicates the results and analysis, and inhibits effective reuse. More detail can always be added as it is needed. Other important parts of the functional specification are discussed below.

Simulation scope—specify the breadth and depth

The definition of the simulation scope is especially important as it defines the boundaries for the analysis. The boundaries should be kept as narrow as possible in order to allow the simulation to focus on a specific region. The following are example questions that can be posed in order to define a simulation's scope:

- Is only one section of a production line being simulated, or is the whole system, including raw material handling and warehouse, included?
- Is only the activity around an airport security station sufficient, or do aircraft arrival and departure details and gate information need to be included?
- Is the workload of only a bank teller important, or are the workloads of all bank departments to be considered?

Boundary Assumptions

Once the boundaries are defined, the assumptions of how the system being studied interacts with its external environment are defined. The following are some example questions that can be used to help define boundary assumptions:

- A production line being simulated may send finished material to a warehouse that is not being simulated. The entire warehouse doesn't have to be simulated, only its characteristics; for example, how often is the storage area full, or can it be assumed that it is never full?
- Will the needed raw materials for a production process always be available?

Operating Assumptions

Not every operation in the simulation scope may need to be simulated. There are some activities within the scope of the simulation that may be addressed in various ways. Consider the following questions to help you know what needs to be included in the simulation:

- Do all the steps need to be included or can some be combined? For example, the details of how an object is taken off a conveyor, manipulated to a particular position, and then put into a box may be combined into one operation. The simulation could take the object from the conveyor into a machine and then output the completed box.
- Are resources always available to do a task? What work breaks are to be included? Will supplies always be available? Will equipment ever break down or need to be resupplied or maintained?
- Will downtime be assumed from historical data or from a similar piece of equipment?
- What are the specific rules for transferring material from one location to another (will current practice or contemplated procedures be simulated)?
- What times are included with the processing time for a piece of material? For example, a fraction of the total setup time can be added to the overall operation time, thus eliminating the need for setup times to be defined for each part at each location. This proration of setup

time is often done in various planning activities; however, this practice is discouraged in building simulation models. In the real system, setup is a separate activity from processing; therefore, it should be modeled that way in a simulation. The intermittent effect of setups in the real system is masked when using prorated values.

Specialized Logic

Specialized rules—either currently in place or planned— should be described. Consider the following examples:

- Dynamically changing processing rates depending on operating conditions (e.g., speeding up or slowing down process times based on number of items in the system)
- Needing to call for a supervisor if particular events occur
- Unique rules governing the flow of materials from one source to another (e.g., tank levels, tank materials, length of time in a tank)
- Specialized cleaning and setup rules
- Constraints, if any, that are created as a result of the interaction with other system elements

Associated documents

The simulation will only be as good as the data that are used. Collecting that data can be very time consuming. It is important to develop and maintain a list of all the specific documents that are used in building the simulation model. Copies of the documents should be kept with the simulation project file or referenced for the simulation builders to use. Documents can include any of the following:

- CAD drawings, block diagrams, value stream maps, etc.
- Data tables or records containing the operation characteristics, such as operation times, move times, setup times, and downtimes, for each element in the system
- Lists of information and estimates developed for system elements when actual data is in short supply, or non-existent
- Spreadsheet files and any data that are imported to the model
- Detailed records obtained for each system element
- Operation times for each part and process
- How and when parts arrive at each location
- Production schedules

- Scheduled and unscheduled downtime, as well as information on time needed to make repairs and time between failures
- Changeover activity and setup time for different products or part types
- Material handling information such as speeds and capacity
- Any other information or data that is required for the system to operate correctly

Section 17-3 Prepare a conceptual model

This section introduces a diagramming methodology that is used to provide a conceptual representation of the system that is to be simulated. It establishes the foundation for the programmed model that is implemented in commercial simulation software such as *FlexSim*.

The best way to document the material obtained for the functional specification is with a flow diagram of the system. Such a diagram also becomes a useful communication tool when discussing the project.

Most operating systems in manufacturing, service, logistics, or other areas can often be described at their simplest level with a diagram showing each step as a block. There are many ways to represent processes diagrammatically (for example, detailed CAD (Computer Aided Design) diagrams and P&IDs (Process and Instrument Diagram)). The P&IDs provide very detailed descriptions (usually in 2 dimensions) of all equipment, piping, electrical, and other connections.

Another type of diagram, the value stream map (VSM), is often useful in modeling since it contains considerable process data that have already been investigated, discussed, and compiled. An example value stream map is provided in Figure 17.3. A VSM addresses three important aspects of operations, albeit at a high level: what, where, and when. This type of diagram, however, is not sufficient by itself for defining a simulation model since it typically lacks the detailed logic needed in most simulation models (e.g., rules for making material transfer decisions or detailed sequence of operations).

Other types of operations-focused diagrams include a bill of materials (BoM) that describes product structure, assembly, or operations diagrams for indicating what materials are needed at what operations and the sequence of operations; Gantt charts for considering time and precedence; facility layouts for defining resource sizes and locations, etc. All of these are common and important input for simulation modeling, but none of them integrates all, or even most, of the definition needed for simulation.

Object Flow Diagram and IDEF0

An Object Flow Diagram (OFD) [3] is a diagramming methodology that was designed specifically for the conceptual design of a discrete-event simulation model. As in product

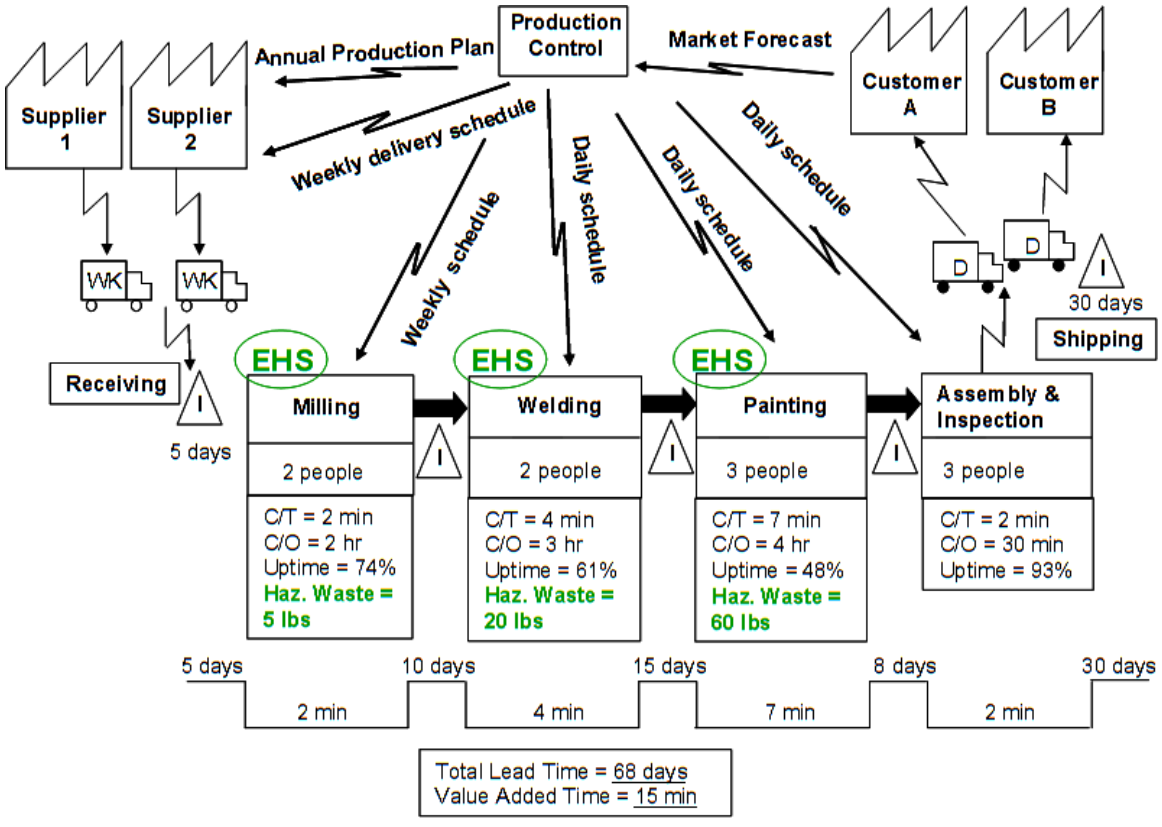


Figure 17.3: Value Stream Map as a starting point for simulations.

design, conceptual modeling is a necessary precursor to detailed design. In product and model design, it is dangerous and oftentimes wasteful to focus on the details without a clear understanding of the problem, system operation and behavior, and objectives of the project (i.e., needs of the decision maker). In terms of the name of the diagram, object refers to something in the real system that needs to be represented and flow refers to an interaction among objects.

As Robinson notes [5 and 6], while there is general agreement in the modeling community that a conceptual model needs to precede a programmed model (the detailed portion of model development), there is little agreement as to what that conceptual model should be. Greenwood, et al. [3] outline a variety of approaches that can be used to help formulate discrete-event simulation models; however, as the article indicates, most were developed for other purposes (process definition, enterprise operations, information systems design, etc.) and are not ideal for simulation modeling. A previous version of this book [1] introduced the use of OFDs as a conceptual modeling tool; Greenwood, et al. [3] extended the methodology.

The OFD is the simulation modelers' "wireframe" or "schematic sketch" since it abstracts and outlines the basic structure of a system in simulation terms. It is a high-level tool for thinking, communicating, and initiating model design. Preparing an OFD encourages modelers to think through the many aspects of a system before translating mental models to a computer model—a step that is unfortunately all too often ignored. The use of an OFD early in a simulation project helps a modeler to

1. recognize which objects and behaviors need to be considered;
2. define the salient relationships that exist among objects;
3. determine the level of detail needed to represent the behavior of the objects;
4. identify the types of data that may be needed;
5. communicate with non-modelers.

An OFD is not intended to represent detailed procedures, complex logic, and the like. It is typically prepared as a sketch, at least in the earliest phases of a project. It is not intended to be a rigid document; it is expected to change often and be updated as the system and project become better understood and better defined.

The OFD methodology is intended to be general so that it (1) can represent a wide variety of types of systems, (2) is simulation software neutral, and (3) defines and describes system boundaries that will be captured by a model.

In addition to its design attributes, an OFD is also a key communication tool between modelers and domain experts and other stakeholders. Since it contains and depicts a collective understanding of a system, it is an effective tool for model validation.

An OFD considers two broad classes of resources—fixed and shared (oftentimes mobile)—that are common in most simulations and that provide the basic flow structure of most models. In most simulation models, physical items, as well as information, flow between objects over time. These dynamic flows among objects result in system behavior and performance.

Many of the symbols used in an OFD are based on the IDEF (Integration DEFinition) methodology and especially IDEF0. See Giachetti [2] for a summary of the IDEF methodology. As shown in Figure 17.4, OFD symbols use IDEF0's ICOM notation (i.e., all four sides of the symbol have a specific meaning). The left side is for inputs to an object where an operation, activity, or process takes place. Items are transformed or consumed by the operation at the object (e.g., a part is machined, an item is purchased, a scanned image is produced). The right-side of the symbol is for outputs from the object (i.e., items that result from the transformation process or operation at the object). The bottom side of the object is for denoting "mechanisms," other items or resources that are used to support the operation, such as operators, empty containers, etc. The top side of the symbol is for "controls," those things that constrain operations or are conditions used in the operation, such as plans and schedules. OFDs, like IDEF0 diagrams, are hierarchical—details of an object can be expanded in a lower-level representation of the object.

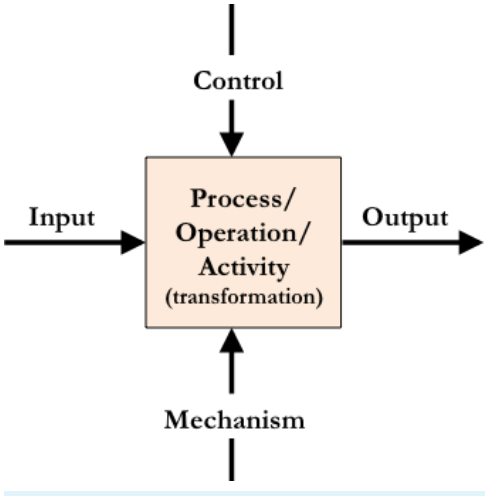


Figure 17.4: IDEF0's ICOM notation.

An example application of the ICOM notation is shown in Figure 17.5. In this case, the focus of the process is a metal forming process, which is shown in the center of Figure 17.5. This process transforms sheet metal (input) into a formed metal part (output). The input is an output of other processes, such as the purchasing process, delivery process, etc. The output is an input to other processes, such as the intra-plant logistics process.

The mechanisms that are needed to complete the transformation process in Figure 17.5 are a forming machine and an operator, both of which incur downtime. The breakdown of the machine is a link to another process—the repair process. The operator may be a shared resource (not dedicated to the one forming machine), and be a link to other operations.

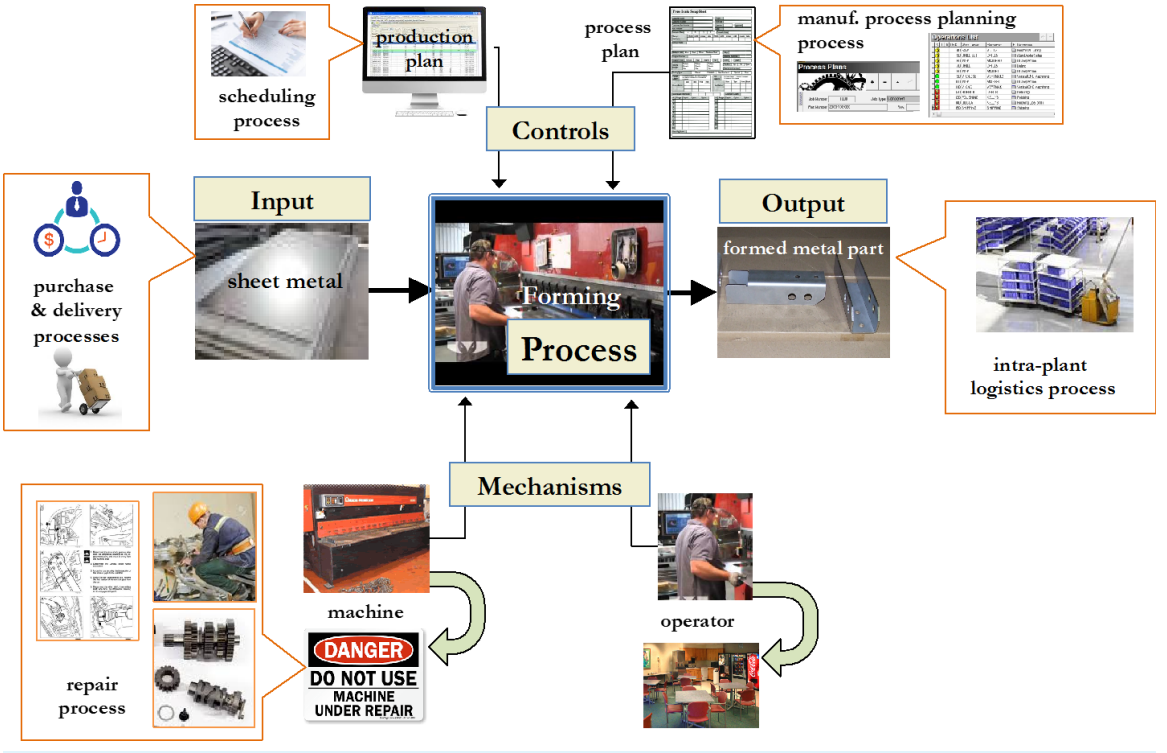


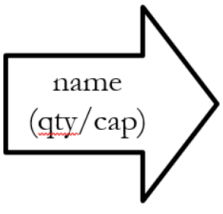
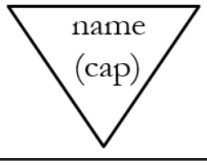
Figure 17.5: Example application of IDEF0's ICOM notation.

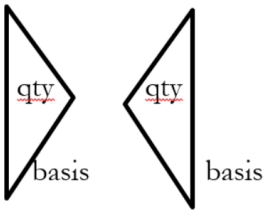
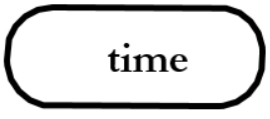
The controls that are needed to complete the transformation process in Figure 17.5 include a production plan and a process plan. The production plan is used to decide what to produce next. It is the output of a scheduling process that transforms customer needs into specific product/machine schedules. The process plan is used to decide how to produce an item. It is the output of a manufacturing process planning process that generates the specific sequence of production steps, machine settings, etc.

The other processes shown in Figure 17.5 show how each process in an operation is linked to other processes. Part of the initial phase of a simulation project is to decide the boundaries of the system being considered and how much of these linked processes to include in the simulation model. For example, are the input and output logistics considered (e.g., material handling to/from the system), or are items just expected to arrive at certain times and can be moved on to the next operation when completed?

OFD Notation

The symbols that are used in an OFD, along with a functional definition of each symbol, are presented in Tables 17.1 through 17.3. The symbols are not specific to particular software but are designed to represent functions commonly used by simulators. Each symbol may or may not represent a specific object in any one simulator. For example, the function of bringing things into a simulation (source) may be a separate object in one simulator or included as part of a processing object in another.

OFD Symbol	Functional Description
Process 	**Transformation of a flow item.** Typically involves a fixed resource and a known delay (thus oftentimes includes a duration symbol). *name* = identifier *cap* = capacity of the resource; maximum number of items that can be contained within the object. If capacity is not an issue, denote as (inf) for infinite.
Transport 	**Transportation or movement of a flow item.** Typically involves a fixed resource, e.g. a conveyor, and a known delay (thus oftentimes includes a duration symbol). The duration may be determined by the conveyance speed and distance traveled (distance transported may be annotated on the symbol). *name* = identifier *qty/cap* = the quantity of items moved at a time and/or the total capacity of the resource (maximum number of items that can be contained within the object). If capacity is not an issue, denote as (inf) for infinite.
Storage 	**Temporary holding area for flow items.** *name* = identifier *cap* = total capacity of the resource (maximum number of items that can be contained within the object). If capacity is not an issue, denote as (inf) for infinite.

OFD Symbol	Functional Description
Group/Ungroup 	**Combining items into a single unit, or conversely, splitting a unit into multiple items.** *qty* = number of items grouped or number of items ungrouped or split (copied) *basis* = criterion for grouping or ungrouping, e.g. combine based on product type. May involve a known delay before or after grouping; thus it may include a duration symbol. Oftentimes the symbols are associated with another operation and thus if the symbol on the left is placed on the left or input side of an object, it denotes a combine operation. Similarly, if the symbol on the right is placed on the right or output side of the object, it denotes a split or separation operation.
Duration 	**Activity or task performed by a basic operation object.** Oftentimes used in conjunction with another symbol. There may be multiple processes conducted sequentially, such as a setup and then a basic process activity. *time* = time to complete an activity

Notes (for all operations symbols):

1. They usually represent a fixed-location physical resource.
2. For all, except the Duration symbol, the sides are significant. They use IDEF0's ICOM (Input-Control-Output-Mechanism) notation.
   a. Left-side: inputs to the resource; the primary item(s) that are transformed or consumed by the object.
   b. Right-side: outputs from the resource; the primary item(s) that are produced by the object.
   c. Bottom-side: "mechanisms," other resources used to support operations of the object.
   d. Top-side: "controls," things that constrain operations or are conditions for functions of the object.

Table 17.1: OFD symbols denoting basic operations.

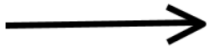
OFD Symbol	Functional Description
Flow/Route (item) (item/qty) 	**Link between operations; route that items use to flow through a model.** The parameters are optional; used if clarification is needed. For example, if a single type of item is flowing through the model, this can be stated as a note on the diagram and not on each link. *item* = name of the item(s) following the route *qty* = quantity of items that flow together Flows usually require time for items to move between objects; therefore, a duration symbol would be included. If the flow is made through a shared (mobile) resource, the duration may be determined by the speed of the resource and distance traveled. The distance transported may be annotated on the symbol.
Decision 	**Decision rule is typically used to either route an item out of an object or to "pull" an item into an operation.** *criterion* = rule used to make the selection Typically added to an operations symbol. If placed on the left or input side of the object, it denotes a pull operation; if placed on the right or output side of the object, it denotes a routing decision.
Source (freq,qty) 	**Creates items that flow in the model.** Defines the input system boundary. *freq* = time between arrivals *qty* = quantity of items that enter the system at each arrival event
Sink 	**Destroys items that flow through the model.** Defines the output system boundary.
Shared (mobile) resource 	**Resource that performs activities at multiple operations. Typically involves processing or moving items, repairing equipment.** *name* = identifier *qty* = number of resources needed to perform an operation
Travel (resource) 	**Path followed by shared (mobile) resource to perform activities at, or between, various objects.** • Planned segment, e.g. transport by shared resource • Cycle completion segment, e.g. return to a location
Communication 	**Link between objects involving information, messages, etc.** These links do not represent the flow of an item.

Table 17.2: OFD symbols related to flows between operations.

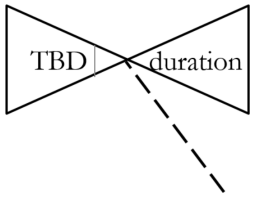
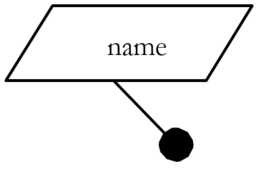
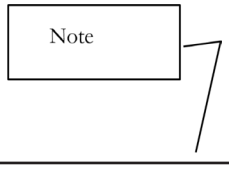
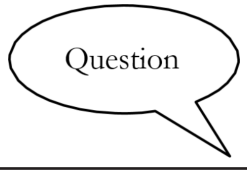
OFD Symbol	Functional Description
Downtime 	**Unavailability of both fixed and shared(mobile) resources.** Downtimes can occur for a variety of reasons, e.g. breakdowns, preventative maintenance, operator breaks, shift schedules. *TBD* = time between downtimes *duration* = how long the resource is down Objects may undergo several types of downtime (e.g., planned and unplanned breakdowns). Multiple downtimes can be represented with multiple symbols or with a single symbol and a note describing the downtimes.
Performance Measure 	**System variable that is of particular interest.** An output of a simulation model; sometimes referred to as a response variable or dependent variable. *name* = identifier Attached to the object of interest; e.g. "average content" might be associated with a Storage symbol or "utilization" might be associated with an operation or a shared (mobile) resource.
Note/Comment 	**Annotation that provides information about the system, an object, a flow, etc.**
Question 	**Annotation that raises one or more questions that need to be addressed.** Identifies the need for additional information, or denotes issues that need to be discussed and resolved.

Table 17.3: Supporting symbols for OFD diagrams.

Table 17.1 contains the symbols used to represent basic operations; these include Process, Transport, Store, and Group/Ungroup. Since many operations include known delays, the Duration symbol is included in this category. Thus, an operation symbol and Duration symbol are oftentimes used together. The operations symbols all use the ICOM (input, control, output, and mechanism) notation from IDEF0. That is, the sides of the symbols are significant as described above. Also, the Transport symbol is typically used for a fixed resource, such as a conveyor, that moves items in a model.

Table 17.2 contains symbols that are flow related. This includes the flow of items (often referred to as routing) and the flow of resources (mobile or shared). Since routing implies that alternative paths exist, decision points and criteria need to be specified. This is done through the Decision symbol. Items that flow need to be created and destroyed once they reach a system boundary, so there must be a Source to create the items and a Sink to

remove them from the model at the appropriate time. Sources indicate the manner in which flow items enter a model. Typically the arrival process is specified as a time between arrivals (oftentimes a random variable) and the quantity of items per arrival (may or many not be a random variable). Arrivals may also occur through a schedule. Mobile or shared resources travel between objects to perform services such as material handling (moving items) or performing maintenance (repairing a fixed resource that is in a down state). These mobile resources typically have paths, composed of segments, which need to be identified and specified. Two types of travel segments are considered. One denotes planned travel segments, such as moving an item between two objects; the other denotes unplanned segments, such as return to a "home" location after an activity is performed (e.g., transport). Finally, processing objects oftentimes need more than just physical items to complete an activity; they may need information, signals, and other forms of communication, from other objects.

Table 17.3 contains the remaining OFD symbols; these symbols support the operation- and flow-oriented symbols. Resources, both fixed and mobile' are not always available; therefore, there is a Downtime symbol that indicates the time between breakdowns and the duration of the breakdown. Performance measures are used to decide which of the alternatives being considered is better. Simulation can collect information on any measure of performance and most software automatically collect statistics on many common measures. It is important to identify early in the modeling process what measures are the most important. Those measures are noted on the OFD through a special symbol. Since the OFD is intended to be used more like a sketch than a final drawing, annotation is encouraged. A Note symbol is used for clarification; a Questions symbol is used to call out a point of uncertainty that needs further definition or specification.

Applying OFDs

Since an OFD is used to help modelers identify and understand the key components of a system and their interactions, and to convey that information to project stakeholders, there is considerable latitude in how it is applied. After all, it is a model itself, albeit a conceptual model, and just like the resulting simulation model, it should only be as complex as it needs to be to be in order to address the problem at hand. The OFD symbols should be used sparingly, and it should only be as detailed as needed to adequately represent a system so that it can be modeled appropriately.

While an OFD can be created in many ways, the following are the typical and suggested ordered steps for initially creating an OFD. Since the conceptual design process is iterative, the construction of an OFD is also quite iterative. It is modified and updated as more is understood about the system and as more detail is needed. Detail is oftentimes added hierarchically in order to enhance readability and meet the needs of various stakeholders.

1. Denote input and output system boundaries. This is based on where items enter and leave the system being considered and modeled. To do this, indicate the items' source(s) and sink(s). Typically, a source includes the specification of the frequency and quantity of arrivals.
2. Indicate all relevant physical fixed resources (e.g., machines, workstations, conveyors, storage areas) using the appropriate OFD symbol.
3. Connect the resources with directional arrows denoting the flow of items through the system using the Flow/Route (item) symbol.
4. Identify points in the flow where choices in the flow need to be made. These points are noted using the Decision symbol. The criterion for the decision is also specified.
5. Identify points in the flow where items are grouped or ungrouped (e.g., for transport or through assembly) and denote in the diagram using the OFD symbols. Specify the quantity of items that are combined.
6. Identify where known delays occur (e.g., for processing or transport) and denote in the diagram using the OFD Duration symbol. Specify the time duration (for processes, this is typically a probability distribution).
7. Associate shared resources (also referred to as mobile or dynamic resources) with processing and transport operations.
8. Identify any flows between objects other than item flows (e.g., communication and information flows).
9. Link shared resource associations to form travel paths. First, identify Planned Travel segments, then analyze the segments to see if any Cycle-Completion segments are needed to form resource paths. This later step oftentimes identifies missing segments, since they are not part of the planned operation, and completes a resources travel cycle or "loop."
10. Identify and note on the diagram any planned or unplanned resource downtimes that need to be addressed in the model.
11. Identify and note on the diagram the key performance measures that should be captured by the model. These are usually the factors that are considered when deciding among competing alternatives.
12. Annotate the diagram with comments, notes, and questions.

An illustrative example

A small manufacturing cell is used to illustrate the application of the OFD methodology. Of course, OFDs can and have been applied to a variety of operations systems, including logistics, healthcare, and other types of service operations. The cell in question is designed to meet production orders for a product called Widget. Orders that are received from customers are entered into the company's information system every hour. The number of orders received in each hour is uniformly distributed between 0 and 2. The number of items in each order is uniformly distributed between 1 and 3. The expected, or average, arrival rate of orders for the product is therefore 2 per hour.

The two key system performance indexes are 1) the amount of time orders wait to be fulfilled and 2) the utilization of the two cell operators.

The product produced by the cell, a Widget, is represented in Figure 17.6. A picture of this odd product, along with a Bill of Material (BoM), is provided on the left side of the figure. As the BoM indicates, a Widget is composed of four basic components—A, B, C, and D. Assembly of a Widget in the cell is performed using one of four reusable fixtures. Another product-related diagram, an operations diagram, is shown on the right side of Figure 17.6. It describes the order in which the assembly operations are performed and where the operations take place in the cell. The operations are denoted as SR-1 through SR-5. Note that the assembled product is packaged at station SR-5. While packing material is a key component of the finished product, it is assumed to always be available; therefore, it is not modeled in the simulation. Of course, the packing time is considered in the model. This is based on the notion that work in the cell is never delayed due to a lack of packing material.

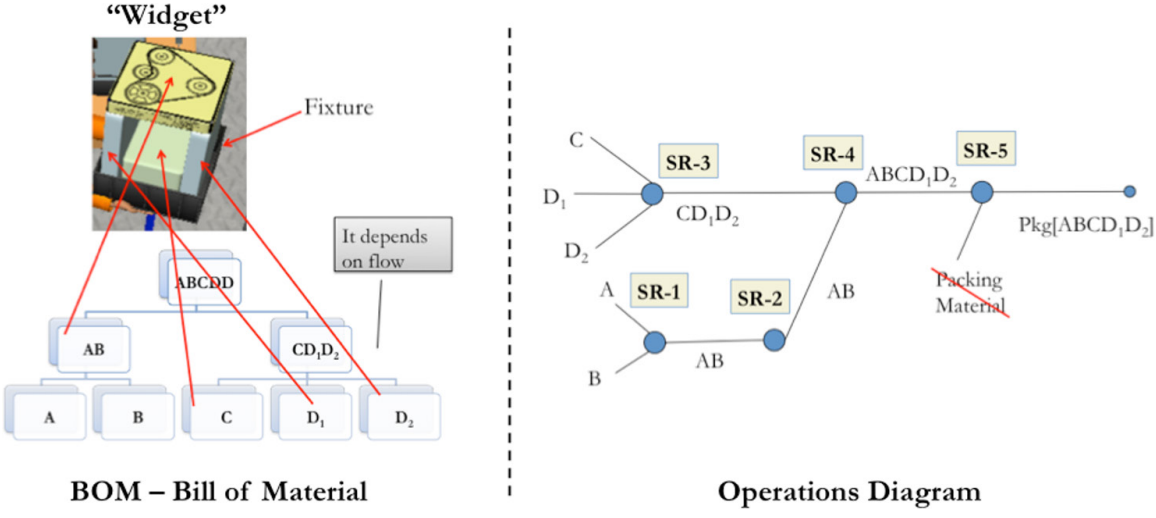


Figure 17.6: Representations of the product "Widget."

Another way to consider the operation of the cell is via a Gantt Chart, as shown in Figure 17.7. The chart shows the timing of the activities to produce a Widget and the coordination between the activities. The chart indicates that operation SR-1 can begin when there is one unit each of components A and B; similarly, operation SR-3 can begin when there is

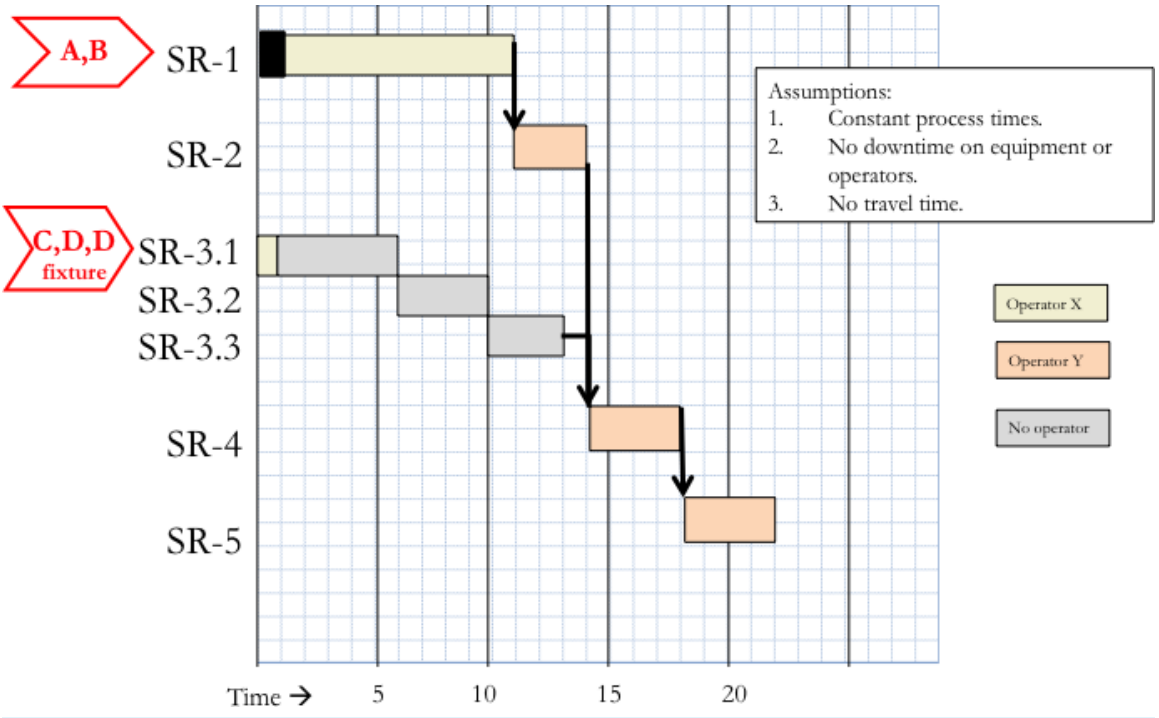


Figure 17.7: Gantt Chart representation of the operations of the Widget cell.

one unit of component C and two units of component D available. Both SR-1 and SR-3 are performed by one operator (named X). If all of the components are available for both operations to begin, then operator X should choose SR-3 since it has the shortest duration and its immediate subsequent activities do not require an operator. Based on the Gantt Chart, the estimated cycle time for the cell is 22 minutes, once all of the components are available; however, this is overly optimistic since the Gantt Chart assumes no variability in arrival times of the components, no variability in process times, all machines and operators are available 100% of the time, and no travel time between operations.

While the three types of diagrams (BoM, Fishbone, and Gantt), and the product picture are important inputs to the model-building process, they are not sufficient. As will be seen later in this example, an OFD more fully defines the system for the purpose of building a simulation model.

A *FlexSim* model of the cell is shown in Figure 17.8. The reason a model is shown, and not a picture of the actual system, is that the company does not want photographs of the facility released to the public; therefore, they cannot be published. The screenshot of the model shows the 1) overall layout of the cell (built to scale from an AutoCAD drawing), 2) position of the five operations areas (SR-1 through SR-5), 3) input conveyors for the four components (A through D), 4) finished product storage area, and 5) two operators that run the cell. One operator, X, has a yellow shirt; the other operator, Y, has an orange shirt. In Figure 17.8 the yellow operator (X) is transporting component A to operation SR-1, and the orange operator (Y) is transporting a packaged product to the finished product storage area.

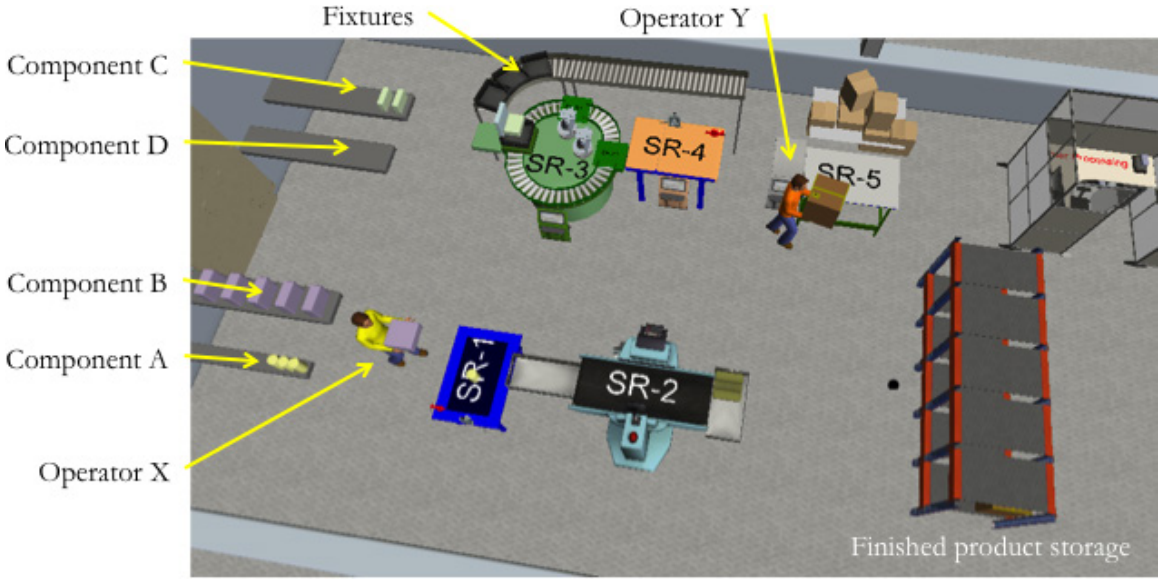


Figure 17.8: Overall view of the Widget cell.

The yellow operator (X) transports components A and B to SR-1 and performs the operation at SR-1. Similarly, the yellow operator (X) transports components C and D to SR-3 and sets up the operation at SR-3, as shown in the view on the left side of Figure 17.9. Operator X only loads the components into a fixture at SR-3. The setup takes one minute. Once a fixture is loaded with the three components, SR-3 automatically processes the components through three steps at positions SR3.1, SR3.2, and SR3.3; the time to perform the operations are a constant 5, 4, and 3 minutes, respectively. The fixture and components move between processing positions via a turntable. When subassembly CDD is completed at SR-3, it is automatically transferred to SR-4 with one buffer space between SR-3 and SR-4.

SR-3 is subject to breakdowns and is repaired by operator X. The time between failures is exponentially distributed, with means of 2 hours; the time to repair is uniformly distributed between 10 and 20 minutes. The view on the right side of Figure 17.9 Shows a breakdown

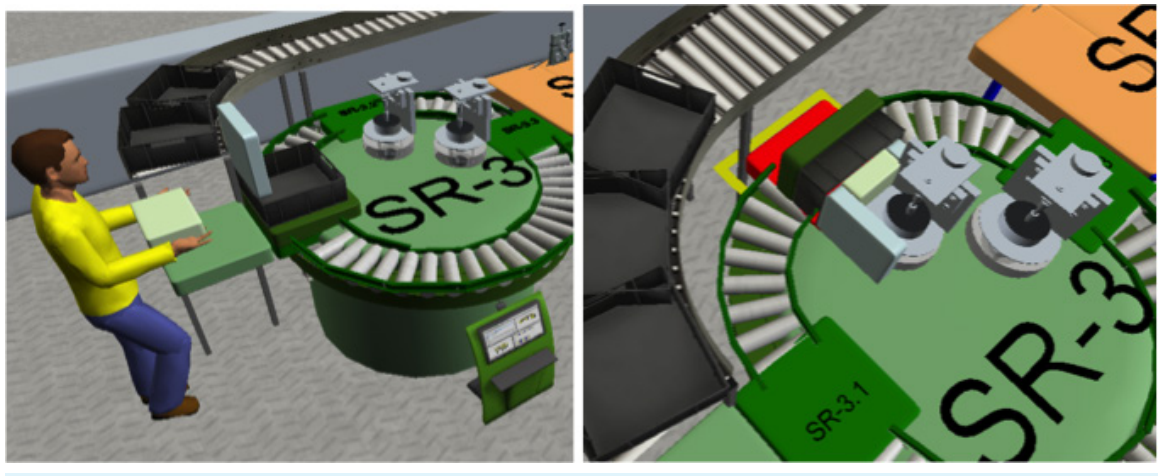


Figure 17.9: Details at operation SR-3.

at position SR-3.2. A failure on SR-3 preempts operator X so that the equipment is brought back online as soon as possible; that is, when a breakdown occurs, operator X stops its current task, goes to fix the problem on SR-3, then returns to the activity it was performing when the breakdown occurred.

The orange operator (Y) performs the operations at SR-2, SR-4, and SR-5 and transports the work in process between these operations. Work in process is automatically transported between SR-1 and SR-2 and between SR-3 and SR-4. In the view on the left side of Figure 17.10, operator Y is transporting subassembly AB from SR-2 to SR-4. On the right side of the figure, operator Y is completing the assembly of the Widget at operation SR-4. When the operation is complete at SR-4, operator Y separates the product from the fixture and transports the product to SR-5; the operator also places the fixture on a conveyor so that it is transported to SR-3 for reuse. Operator Y moves packaged Widgets from SR-5 to the final product storage area.

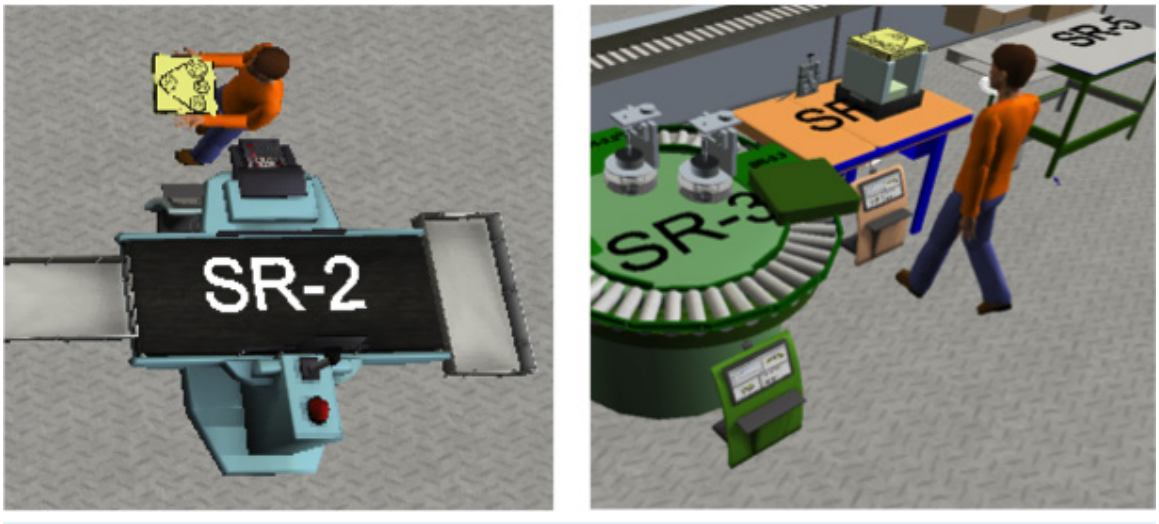


Figure 17.10: Example activities performed by one of the operators in the Widget cell.

Both operators take breaks, but not at the same time. Each one takes a 15-minute break about two hours into a shift and again about two hours prior to the end of a shift. Both take a 30-minute lunch break about midway through a shift.

For the purpose of the analysis, the system boundaries of the work cell are from arrival of components to the cell until a packaged product is placed in finished-product storage. That is, any processing of the components prior to their arrival to the cell and after finished product storage are not a concern of the cell and thus are not considered in the model.

Components arrive to the cell as follows. Four component A's are delivered every two hours. When the number of B's that are available drops to two, then four B's are reordered; it takes 15 minutes for the order to be fulfilled. Components C and D arrive individually and their mean time between arrivals is 30 and 15 minutes, respectively; the times are exponentially distributed.

Using the symbols defined in Tables 17.1, 17.2, and 17.3, an OFD for the Widget production cell is shown in Figure 17.11. Notes and special segments are shown in shaded

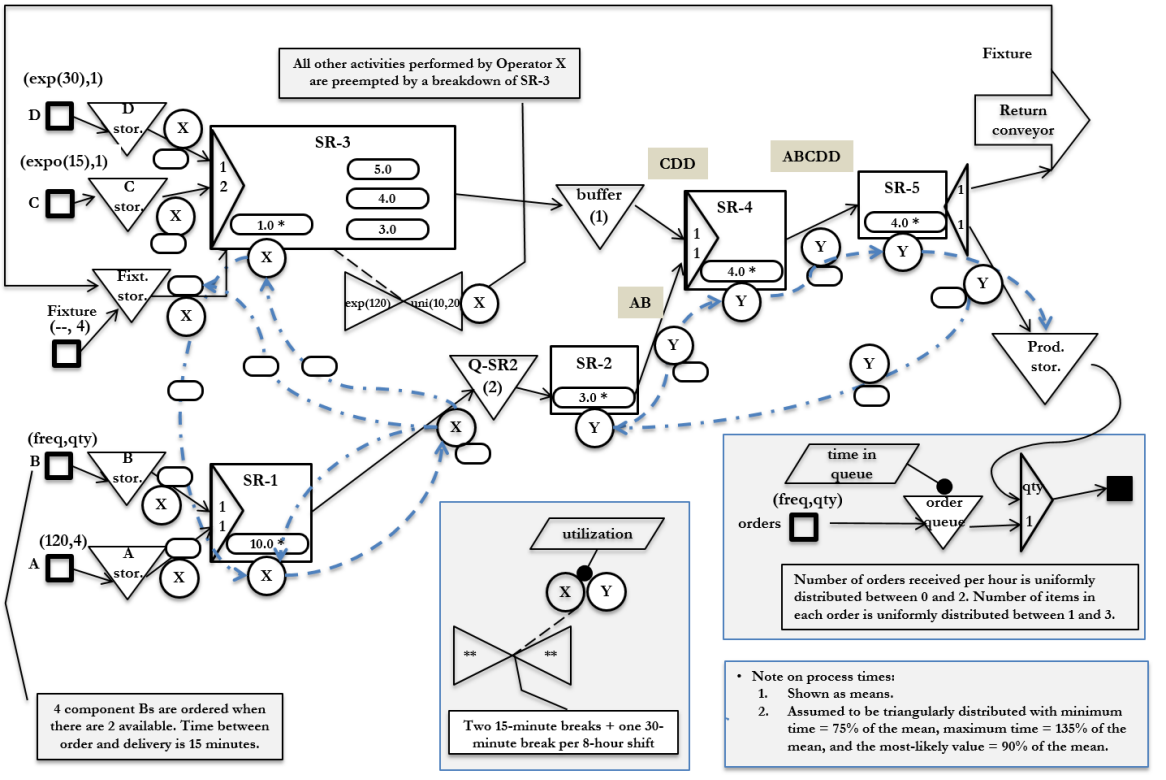


Figure 17.11: OFD for the operation of the Widget cell.

boxes. A simulation model could easily be built from this diagram regardless of the simulation software being used; that is, the OFD is software neutral.

As described earlier, the machine at operation SR-3 is comprised of three sub-processes that are connected via a turntable for transport between the sub-process steps. In Figure 17.11 the processing at SR-3 is abstracted to a higher level (i.e., it is represented by a single object). In other words, the turntables to transport the product between SR-3.1, SR-3.2, and SR-3.3 are not modeled explicitly, but as three sequential process steps after the manual setup operation performed by operator X. This approach simplifies the model considerably.

Note that the conveyor for the fixture return is represented in the OFD by two symbols; one represents the transport and one represents the storage.

Instead of showing the downtime schedule on each operator symbol in the diagram, the schedule is described in a shaded-box segment located in the bottom center portion of the diagram. The segment also indicates that operator utilization is an important performance measure.

Most of the logic in the OFD in Figure 17.11 is concerned with pushing components through the cell to finished-product storage. Products are pulled from storage based on orders. The order fulfillment process is represented in the shaded box in the lower-right portion of the figure. Orders for the product are generated hourly and await fulfillment in a queue. An order is fulfilled when there is sufficient quantity of the product available in the finished stores to meet the requested order quantity.

The shared or mobile resource paths for Operators X and Y are constructed from the locations of the calls for service from the fixed resources. It is important to note that if only the component and subassembly flows are considered, then some of the shared (mobile) resource travel segments would not be identified. It is only by connecting the activities for operators X and Y that resource paths or cycles are completed. Those "missing" flow segments are shown with dot-dashed flow lines, rather than the dashed flow lines that are created by the component and subassembly flows.

There are several Cycle-Completion segments for Operator X. For example, Operator X processes items at SR-1 and sets up SR-3 and thus must travel between the two work areas. Approaches that are only process driven will overlook situations where the resources have tasks to complete which are not directly flow related. An example for Operator Y is the return from finished product storage to SR-2.

All of the process times for manual operations (those indicated with an * in the OFD) are random variables; however, only the mean is shown within the symbols in the diagram. The distribution is specified in a note. In this case, all process times are assumed to be triangularly distributed with the minimum time being 75% of the mean, the maximum time being 135% of the mean, and the most likely value being 90% of the mean.

Figure 17.12 shows how to represent the detailed operation of SR-3 considering the turntable. At SR-3.1, the components are gathered by operator X and placed in a fixture. The operator also performs a one-minute setup operation at SR-3.1. After the setup, 1) SR-3.1 processes the subassembly for five minutes, 2) the turntable transports it to SR-3.2 where it is processed for four minutes, 3) the turntable transports it to SR-3.3 where it is processed for three minutes, and 4) the completed subassembly moves to a single buffer space prior to SR-4. The turntable is returned from position SR-3.3 to SR-3.1 to process the next subassembly; this operation is another example of a Cycle-Completion path segment.

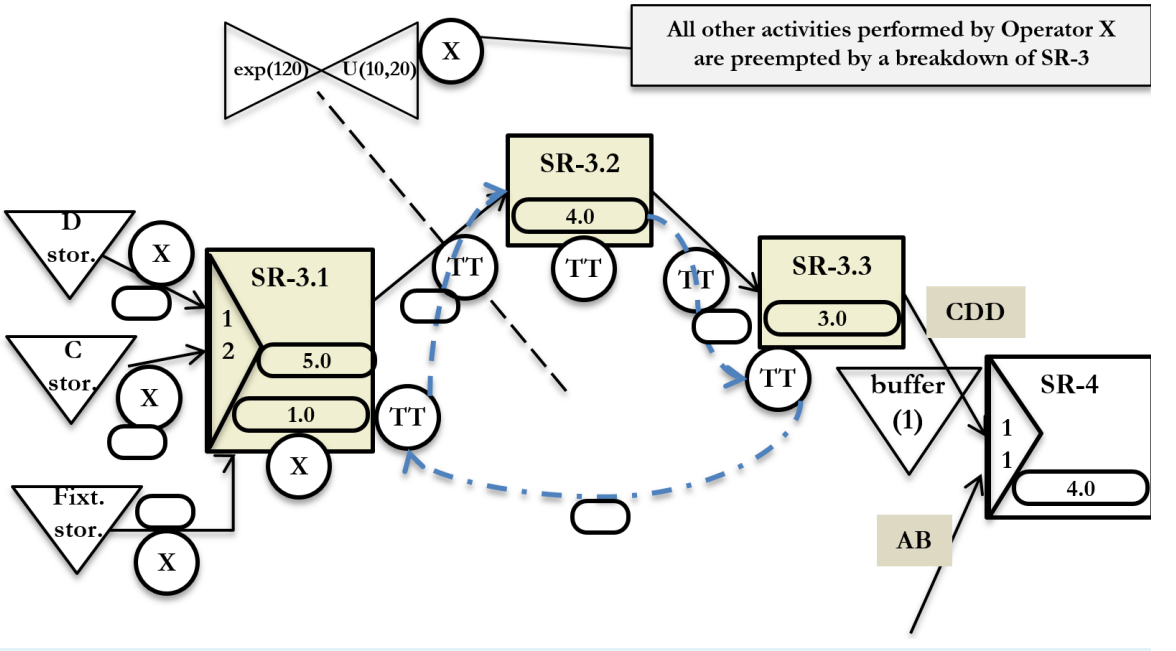


Figure 17.12: Hierarchical representation of SR-3 in the

Section 17-4 Initial sizing of resources

In the example above it was clear how many resources are in the system; however, this will not be the case for all systems. For example, the system may not yet exist, and the simulation model is being developed to analyze and assess the proposed system. Oftentimes, the model may be used to determine how many transporters are required (e.g., car carriers in an automotive assembly plant). It would be a waste of time to keep incrementing the number of carriers and running the model, guess-and-check fashion, to reach a feasible solution; therefore, this section provides a methodology to determine the minimum number of resources required.

The approach is based on the notion from queuing theory that the number of resources available must be able to keep up with demand; that is, the overall service rate, μ, must be greater than the arrival rate, λ:

$$\frac{\lambda}{\mu} < 1$$

Stated another way, and shown in the expression below, the ratio of the total amount of work that needs to be performed must be less than the capacity of the resources; therefore, the minimum number of units of the resource is the next highest integer of the ratio:

$$\frac{\bar{N}_T * \bar{W} + O_T}{n * R_T * \left(\frac{1-\bar{DT}}{R_T}\right)} < 1$$

$$\therefore n = \left\lceil \frac{\bar{N}_T * \bar{W} + O_T}{R_T * \left(\frac{1-\bar{DT}}{R_T}\right)} \right\rceil$$

where

n = min. number of units of resource;

$\overline{N}_T$ = mean number of items arriving at the resource in time period *T*;

$\overline{W}$ = mean amount of work to be performed per item;

O_T = amount of other, non-item work performed in time *T*;

R_T = amount of time resource is available in time *T*;

$\overline{DT}$ = mean downtime in R_T.

Consider the following example. Swell Computers is considering starting a new facility to produce its newest model. They expect orders to arrive at their website 24/7 at an average rate of 1.5 orders per hour. They estimate the assembly operation to take 35 minutes, on the average. The operators are required to clean up their workspace and compile their paperwork at the end of the shift—this typically takes 20 minutes. Assembly will only work one shift or 40 hours per week; operators take 2-15 minute breaks and a 30-minute lunch break per shift. Swell wants to estimate the number of assembly resources it will need.

The sizing expression is applied below and indicates at least 5 assembly resources are needed. Of course, a simulation model would consider the effect of variability in the order arrival times and assembly times, as well as interactions with other production processes, level of service, etc. Nonetheless, the simulation model would be built with 5 assemblers as a starting point, as shown below:

$$\left[\frac{(168\ hr/wk * 1.5\ orders/hr) * 35\ min./order + 20\ min./shift * 5\ shift/wk}{(40\ hr/wk * 60\ min./hr) * \left(1 - \frac{1}{8}\right)}\right]$$

$$= \left[\frac{8920\ min.\ of\ work\ /\ wk}{2100\ avail.\ min./\ assembler\ wk}\right] = [4.2] = 5\ assemblers$$

Section 17-5 Building the simulation

The second phase of the project concerns the physical work of the project team. This part of the documentation template details how the simulation model was built and how it operates. This information is key for present and future users who want to successfully make use of, and modify, the simulation model.

While the first section of the template should be stable (meaning, the objectives should remain consistent throughout the project) this section is a "living" document and will change and mature as the project proceeds. It should be updated during the project so that any new member who may join the team will have an initial reference point.

The section starts by documenting the simulation software used, as well as the software version number. Other items to be documented include details and definitions of the basic parameters used, such as the base unit of time and distance; abbreviations and acronyms; and any color coding standards. Additionally, any simplifications that are being made for the simulation should be listed; such simplification may include combining a machine's setup time with the processing time or combining all breakdowns into a single common event.

The documentation should include all parameters associated with each of the simulation objects as well as any specific logic used in their operation. There should be a description of what data is used and where it is stored (e.g., in tables or on individual objects). One of the most important parts of building a simulation model is to fully define and document the use of variables, attributes, and functions. If possible, this definition should be done within the simulation software. Much of a simulation model is self-documenting as a result of entering the data required to build the model; however, the more complex the logic within a model, the more important it is for information about variables and functions to be documented.

Using comment lines within the code segments to explain logic wherever possible makes it easier for others to understand. Logic that may seem obvious at the time it is written may not be so obvious to another person in the future.

Areas where specialized logic is developed should also be explained in Part II of the template documentation. That documentation should include why the custom programming is needed and a detailed description of how it is implemented.

Additionally, if any new or customized objects are developed for the simulation, they should be documented in the template. A description of the object, how it works, any limitations, and the possibility for reuse should be included.

Any use of subroutines or specialized program files should always be included in the model documentation. It is important to include everything that is used to make the model run.

User interface

Since the simulation may be run by individuals who were not involved with its development, a description is needed of how a user interacts with the simulation and where the results can be found. Screen shots of the simulation will help further define the model's operation.

For data input, the documentation should identify the key variables and describe the organization of the input data sheets or tables. It should describe how and where data may be changed.

All reports or other outputs of the simulation project should be documented with samples. Details of how the reports were calculated should also be noted. Instructions for exporting, including any templates that were established for the project, should be included.

Section 17-6 Results and analysis

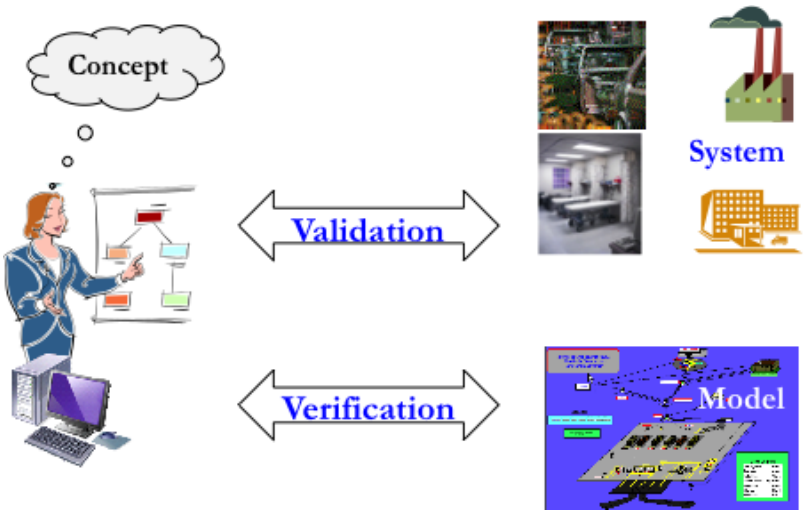
The third section of the project template documents the completion of the project. It has two major functions. First, it documents that the simulation is operating correctly. A validation plan describes how the simulation is checked. This often is done by checking output for known conditions or checking output when all randomness is removed. Second, it documents the procedure for defining how the simulation is used and how the results are analyzed.

Validation & Verification

A key part of any simulation project, large or small, is validation and verification (V&V) of the model with key stakeholders. The basic V&V concepts are illustrated in Figure 17.13. One way to characterize and differentiate validation and verification is to consider validation as "building the right model" and verification as "building the model right."

Validation is concerned with determining whether a model is a meaningful and accurate representation of a real system; in other words, it must be sufficiently accurate to meet the model's intended use. The focus is typically on the conceptual model, such as the OFD, and the real system; however, the programmed model, especially when using realistic graphics for visualization (as is the case with *FlexSim*), can be an effective means for validation. A simulation model that looks like the real system, oftentimes obtained through 3D graphic capabilities in the simulation software, can facilitate validation, enhance credibility, and

increase confidence in the simulation. Of course, the look of the model cannot supplant a correct representation of the system, appropriate modeling logic, correct input data, and valid statistical analysis of the model's output.



17.13: Validation and Verification of a simulation model.

Verification is concerned with determining whether a model works as intended While the focus of verification is typically on the programmed model and the conceptual model, the use of a model with realistic graphics can aid in verification as well.

A common means for validating and verifying a simulation model is through a structured walkthrough of the model operation and logic with domain experts and owners. This can be done both with a conceptual model, such as an OFD, and with a programmed model. In fact, structured walkthroughs are encouraged throughout a simulation project. This allows validation and verification to evolve in a gradual and controlled manner, builds confidence and trust, and leaves time to adjust to changes that need to be made. Again, a good graphical representation is a big help as well as using dynamic colors to easily distinguish key system states.

Another common means for V & V is to run actual data through the model and compare the results in terms of key output measures. For example, a production schedule could be input to a model and the simulation results compared with those obtained from the real system.

If the system is new, the simulation should be checked against the functional specification and be reviewed by developers and operations personnel.

The following are other areas that should be checked, if applicable to the project:

- Material balance (often checked by turning statistical distributions off)
- Downtime/reliability (note number of downs, total downtime, etc.)
- Production schedules (run variation of schedules to assure that changes are handled correctly)
- Specialized logic (test individual units and step through operations if necessary)

- Visual characteristics (Does it "look" correct? Would an operator say "that's the way it works"?)

A basic fact is that a simulation will never act exactly like the real life process; there are just too many variables. It should, however, capture the major operating characteristics of the system. A rule of thumb is that the model behaves as expected. Success can come when someone familiar with the operation can look at the model and results and say, "That's the way it works."

Just like any other aspect of a project, both validation and verification must be planned early in the simulation project. The project plan should define exactly how the model will be evaluated (what means, time period, data source(s), evaluation criteria, etc.), as well as define when the evaluations will occur and who will be the evaluators.

Operability

The following list contains criteria that must be met for project operability:

- The data is checked by personnel who understand the operations to ensure that the data still correctly represents the system.
- The reports are accepted as being correctly generated according to calculated values similar to the actual production reports. The data entry methods are understood by all involved in the model building.

Analysis

Once validated, the model can be used for extensive analysis. Simulation analysis is a process itself. (See Chapters 10 and 11 for a detailed discussion of how to correctly analyze output obtained from simulation models.) It requires a plan to ensure that the analysis will be consistent with the project's goals and objectives.

The focus of the output analysis should be on the performance measures defined in the functional specification. Simulation software can often be intimidating and require a user to "cut through the smoke" to find the important results. Correctly defined performance measures should help identify what's important and what's not. A major part of the analysis plan is a well thought out design of experiments. (Again, see Chapters 10 and 11 for a full discussion.)

The design of experiments defines how the simulation is to be run. It does so by specifying such things as what variables should be changed, how many runs should be conducted, and what results should be collected to provide the information needed to achieve the goals and objectives of the simulation project. Experiment design specifies different simulation scenarios, which in turn define how the system configuration will vary. Results from the experimental runs need to be statistically analyzed. The definitions at this point may include what specific statistics should be collected and the level of confidence desired.

- Define the base scenario: The base scenario is the basic system configuration; it is typically a representation of the current system.

This scenario is used to validate the model, and all other scenarios are compared to it.

- Define the changes that need to be investigated: The changes that need to be made will correspond directly to the changes defined in the goals and objectives. For example, machines may be added, operation times changed, or different methods of material handling used.
- Establish the number of scenarios for each change: There needs to be a limit on what is changed for each scenario. If too many things are changed, then it will be impossible to determine what change caused what effect.

The simulation output is the final step of the simulation project process. It needs to be stressed and understood that if any of the previous steps have been neglected or performed incorrectly, then the results may be invalid. Good output analysis involves correct application of statistical analysis.

The focus on the output analysis should be on the performance measures defined in the functional specification. Simulation software can generate a lot of output by default. The developers try to make as much information available to the user as possible. The volume can be intimidating and requires a user to be able to distinguish the results that are relevant and important to the project at hand. Correctly defined performance measures should help identify what's important and what's not.

With the important data identified, make charts and graphs that will visually convey this information to management. The way the data is presented is sometimes more important than the model itself.

The full, blank template follows at the end of this chapter.

Section 17-7 Example of a project template

This section provides an example of how an Occasional User might utilize the Simulation Project template to define a simulation project for an aluminum plate production line. The following includes background information on the problem. Part I of the template has been completed based on the background information.

Allied Aluminum Company-Dexter Plant
Polished Aluminum Plate Production Line

Background

Aluminum, along with other construction materials, has experienced a downturn in business. New uses for aluminum, in ship building and other areas, has caught the interest of Allied Aluminum senior management. The marketing organization has indicated that to be successful, the company will have to be able to react to orders of varying sizes of

aluminum plates. Allied Aluminum has two plants capable of producing the aluminum plates but neither plant is performing well.

Four years ago the Dexter plant put in a new high speed cutter line in an attempt to increase production and be more efficient; however, in comparison to other plants, the plant still lags in linear footage across its new cutter and in equipment utilization.

The Dexter team has come up with a number of possible areas where they can improve performance. They are focusing on adding a new conveyor that will eliminate delays caused by the manual transfer of material in the plant. The union has also agreed to discuss work rules and downtimes. To get the work done as quickly as possible, the team agreed to outsource the simulation development but wanted to be involved with the analysis. The development will be based on a functional specification using the simulation project template.

Plant layout

There are two production areas in the plant—a newer high speed line and an older, conventional cutting line. Both lines utilize the same wrapping and packaging system as shown in the figures.

Aluminum coils of varying lengths are brought to the high speed cutter (HSC) where they are loaded on to the cutter and cut to the correct width. Protective paper is matched with the aluminum at this time. This paper serves as a protection for the polished surface of the plates during the stacking, packaging, and shipping of the product. The cutter produces the correct width while the shear section cuts the length. There are two stacking sections at the end of the HSC. The stacking sections are used alternately when possible. Regular orders move on to be wrapped while bulk orders go directly to the warehouse.

Stacks currently are moved by fork lift to the warehouse or automated wrapping unit (AWU) input conveyor. Units are wrapped with foil. Wrapped stacks then either move on to the automated packing unit (APU) or go to a unit pack station by conveyor. The AWU gets packs to wrap from the HSC or from the older, conventional cutter line.

- Finished production is moved to warehouse areas by fork truck.
- Break times for operators and maintenance times on the machines are scheduled.
- The speed through the cutter system changes dynamically based on 1) the length into coil, 2) inspections, and 3) metal and paper coil changes.

The first section of the project template might look like this when filled out:

Simulation Project: HSC cutting line Version/Date: 7 October 2016

Company: Allied Aluminum Company—Dexter Plant

Start Date: 12 September 2016

End Date: 1 February 2017 (as of date: 7 October 2016)

Key words: Continuous Improvement, High Speed Cutting, Production Scheduling, Work Rules

Part I - Problem Definition

1. Background

While the new high speed cutting line has increased the plant's ability to produce plates to order, the plant still lags behind other facilities in overall throughput and efficiency. Unless changes are made, new production orders will be issued to other facilities resulting in loss of revenue and jobs to the Dexter site. Simulation will be used as the primary analysis tool to help the plant's continuous improvement teams validate and prioritize their recommendations.

2. Objectives (*reason for simulation; expected outcome*)

Create a simulation that can be used by the continuous improvement (CI) team members to determine the most cost-effective actions.

3. Key performance measures (*basis for choosing among alternatives; define "best" in terms of measures*)

- Linear footage across the high speed cutter
- Number of packs delivered to the warehouse per shift
- Machine efficiency for the cutter, shearer, wrapping, and packing units
- Percent orders delivered on schedule

4. Key decision variables (*what is changed among alternatives*)

- Use of conveyors for material handling
- Machine speeds
- Rules for allocating work to old and new cutters
- Work and break schedules

5. Simulation scope

c. Defined systems boundaries
   i. All operations within the new cutter line building
   ii. All operations requiring staff

d. Boundary assumptions and conditions
   i. Steel and paper rolls are always available and brought in to keep floor areas supplied.
   ii. Corrugated box material and foil wrapping is always available.
   iii. There are no bottlenecks moving material out of the area.

e. Operating assumptions

i. A product list of plates produced on the line is available.
ii. Specification for coils of materials is available from the purchasing department.
iii. Current production schedules are available from the planning department.
iv. Future schedules will be provided by corporate marketing.
v. All operating rules, machines rates, and control logic are included in the Operating Procedure book.
vi. Exact layout information and conveyor speeds are contained in plant CAD drawings.
vii. Historical machine downtimes are recorded in the control system and will be used as a starting point. Impact of improving maintenance, response, and repair times will be determined by the simulation.
viii. Current work rules governing break times and staffing levels will be used but may be modified in the simulation.

References

1. Beaverstock, M., A. Greenwood, E. Lavery, and W. Nordgren. 2012. *Applied Simulation Modeling and Analysis using FlexSim*. 3rd ed. Orem, Utah: *FlexSim* Software Products, Inc .

2. Giachetti, Ronald E., *Design of Enterprise Systems: Theory, Architecture, and Methods*, 2010. Boca Raton, FL: CRC Press.

3. Greenwood, A. G. Pawlewski, P., and Bocewicz, G., "A Conceptual Design Tool to Facilitate Simulation Model Development: Object Flow Diagram," *Winter Simulation Conference*, (R. Pasupathy, S.H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, Editors), December 2013, Washington, DC, pp. 1292-1303.

4. Pidd, M. *Tools For Thinking* 2nd Ed., Chichester, England: John Wiley & Sons, 2003.

5. Robinson, S. 2008. "Conceptual Modelling for Simulation Part I: Definition and Requirements." *Journal of the Operational Research Society* 59 (3): 278-290.

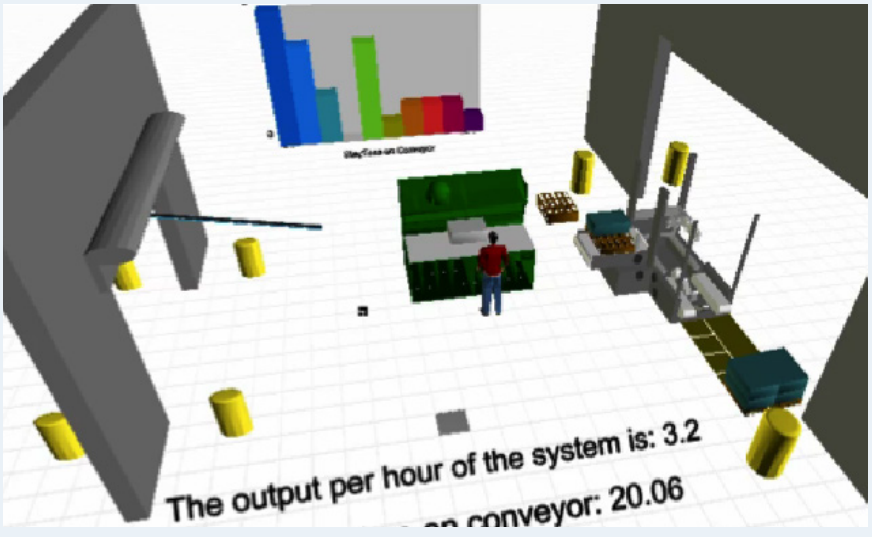
6. Robinson, S. 2012. "Tutorial: Choosing What to Model – Conceptual Modeling for Simulation." In *Proceedings of the 2012 Winter Simulation Conference*, Edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chapter 17 Review Questions

1. Identify three "nuggets"—the things you found to be the most interesting—or most important—in the chapter.
2. Describe the main activities that need to be performed in the four key steps in a simulation project.
3. Discuss why it is so important to have a clear objective for the simulation project.
4. Discuss the difference between boundary and operating assumptions. Provide examples of each.
5. Describe the significance of each side of the fixed resource symbol that is used in an OFD; in other words, what is inferred from what flows in and out of each side.
6. Using the OFD notation, represent a technician (a mobile resource) that takes a 15-minute break every 2 hours.
7. Using the OFD notation, represent a 100-foot conveyor that travels at 300 feet per minute and breaks down periodically. The time between breakdowns is exponentially distributed with a mean of 4 hours; the time to repair the conveyor is exponentially distributed with a mean of 120 seconds. Items traveling on the conveyor are 2-foot cubes.
8. Compu-Help is considering a new call-in help system. They are planning on using automated help desk devices that operate 24/7. They expect calls to arrive at an average rate of 15 per hour, Poisson distributed. Each device can handle one call at a time. The time spent with each caller is normally distributed with the mean time of 3 minutes and a standard deviation of 1 minute. The system is designed such that a maximum of 6 calls can be waiting at any time; i.e., if a call arrives and there are six waiting, then the incoming call is lost. The call system is expected to crash occasionally—the time between crashes of each device is exponentially distributed with a mean of 4 hours; the time to recover is uniformly distributed between 4 and 8 minutes. Construct the object flow diagram (OFD) for this system.
9. For the Compu-Help problem, calculate the minimum number devices needed to meet the anticipated demand.
10. For the Compu-Help problem, assume the time to handle each call is triangularly distributed with the minimum time being 1 minute, the maximum time 20 minutes, and the most likely time 9 minutes.

Also, assume each device must process a report on its activities every 4 hours; the time to process the report is uniformly distributed between 4 and 6 minutes. Calculate the minimum number devices needed to meet the anticipated demand.

11. The following figure shows a simple manufacturing operation, referred to as FinishCell. In this case, items arrive by an overhead conveyor, are placed onto a workstation by an operator, and are moved to a pallet once a finishing operation is performed. Notice the items are gray when they arrive on the overhead conveyor and are blue when placed on the pallet. When a pallet is full, it is picked up by a fork truck.



    a. From a modeling perspective, what do you see in the picture? What are the key elements and their relationships?
    b. Sketch out an OFD based on what you see in the picture; use the symbols in Table 17.1 through 17.3. There is no need to draw it on the computer, just sketch it on paper.
    c. Build and test the model in FlexSIm.

12. The time between arrivals of patients to the imaging area of a hospital is exponentially distributed with a mean of 30 minutes. The time to process a patient is exponentially distributed with a mean of 25 minutes.

    Currently, the only type of imaging available is MRI; however, the hospital is considering adding a CT scanner. Due to current operations problems in the imaging area and plans that are underway for an expansion, the hospital is undertaking a detailed analysis of the

proposed system. It plans to use simulation to support the analysis.

The imagining department expects a 50% increase in patients with about 40% requiring a CT scan and the remainder requiring an MRI. Arriving patients wait in a waiting room until a technician is ready for them. When ready, a technician escorts them to the appropriate equipment and gets everything ready for the procedure. When ready, an imaging specialist joins the technician to perform the imaging. When finished, the specialist analyzes the image and creates a report. The time to ready a patient for the procedure, regardless of whether it is a CT scan or MRI, is normally distributed with a mean of 10 minutes and a standard deviation of 3 minutes. The imaging time is normally distributed with a mean of 6 minutes for a CT scan and 30 minutes for an MRI. The standard deviation of the times is one-third of the mean. The time required for the specialist to analyze the image and prepare a report is triangularly distributed with the shortest time being 5 minutes, the longest time 18 minutes, and the most likely time 7 minutes. The imaging machines occasionally require maintenance; the time between maintenance needs for each machine is exponentially distributed with a mean of 4 hours; the time to perform the maintenance is exponentially distributed with a mean of 12 minutes. A technician performs the maintenance activity. Each technician takes a 20-minute break every two hours (i.e., a technician is available 100 minutes out of 120).

a. Name two stakeholders in the simulation project described above and one role they would serve.
b. Provide a key reason why basic queueing theory would not be applicable for analyzing this problem.
c. Sketch an object flow diagram of the proposed system.
d. What is the minimum number of technicians required?

13. Based on the situation described below for Fister's Foods
    a. complete Section I of the Simulation Project Template;
    b. sketch an object flow diagram;
    c. decide what scenarios should be included in the simulation analysis.

Background

Arnold Fister's family is in the business of quick-freezing gourmet food that is sold through the internet. Over the years their operation has grown from their garage to a new plant. Once frozen, it's important to get the food into shipping containers and into the mail as fast as possible. In the main section of the plant, which is kept at a low

temperature, the food orders are assembled and frozen. The frozen pieces are placed in boxes and sent to the shipping area.

The boxes are moved by a robot from the cold area of the plant to the shipping area, which is not refrigerated. In the shipping area, the boxes are taken from a conveyor to the packing machine where the proper amount of dry ice is added and the box hermetically sealed. Shipments are divided into two categories depending on the destination distance and region. Those travelling the greatest distance or going to southern locations require more dry ice in the boxes.

After being sealed, the boxes are placed on pallets by destination type. Once four boxes are placed on the pallet, the pallet is moved by overhead crane to a pickup location where a fork truck takes them to waiting delivery trucks.

A single operator has been able to manage the shipping area. The operator must pick up the box from the conveyor and bring it to the packing machine and set the machine for the proper amount of dry ice. The operator also takes the completed package from the machine to the appropriate pallet. Once a pallet is moved, the operator places a new pallet in place.

Problem

Business is picking up and there is a worry that the operator will no longer be able to keep up with the packaging process. It's critical that boxes not stay in the shipping area for long periods before being packed for shipment. Options include adding more staff, adding automated loading and unloading for the packing machine, or adding bar coding so that the packing machine does not have to be setup manually by the operator.

Additional Information

Boxes needing more dry ice are color-coded red while standard boxes are coded blue. Some important measures would include the percent of the time the operator is busy and the length of time a box stays in line before packing. For the project, consider what other information is needed and indicate that on the project template.

Appendix 17A – Simulation Project Template

Simulation Project: <name> Version/Date:

Company:
Start Date:
End Date: (as of date:)
Key words:

Part I - Problem Definition

1. Background (*brief system description, project/problem history*)
2. Objectives (*reason for simulation; expected outcome*)
3. Key performance measures (*basis for choosing among alternatives; define "best" in terms of measures*)
4. Key decision variables (*what is changed among alternatives*)
5. Simulation scope
    a. Defined systems boundaries
    b. Boundary assumptions and conditions
    c. Operating assumptions

Part II - Operational Description

1. System Description (*with supporting diagrams, photos, and historical data*)
2. Key stakeholders

Name	Organization	Position	Contact Information	Role

3. Conceptual model /Object Flow Diagram (OFD)
4. Special logic or other considerations to be included
5. References (*for supporting documents*)

Id. No.	Name	Location	Type	Date/Version	Comments

Part III – Simulation Model Development

1. Model file(s)
    a. Software and version

b. File name(s):

2. Simulation basic units of measure
   1 unit of simulation time =
   1 unit simulation distance =
3. Abbreviations and acronyms (*include color coding*)
4. Object naming convention
5. Modeling simplifications and assumptions
   a. Simplifications
      1. …
      2. …
   b. Assumptions
      1. …
      2. …
   c. Assumptions (to be relaxed)
      1. …
      2. …
6. Object definitions
   a. Flowitems
      1. Basic properties

Flowitem name	Type	Source	Graphic	Description

      2. Labels

Flowitem name	Label name	Label type	Where used / updated	Description / value

   b. Fixed resources (including Visual objects)
      1. Basic properties

Object name	Type	Capacity	Graphic	Downtime (type, frequency, duration)	Description

2. Labels

Object name	Label name	Label type	Where used / updated	Description / value

Planned delays

Object name	Setup time	Setup Condition	Process time	Comments

3. Custom logic

Object name	Trigger	Description

c. Task executers (mobile resources)

1. Basic properties

Object name	Description	Capacity	Graphic	Downtime (type, frequency, duration)

Object name	Capacity	Speed	Accelera-tion	Decelera-tion	Comments

2. Labels

Object name	Label name	Label type	Where used / updated	Description / value

3. Path network diagram and/or node and path definitions

Object name	Network node	Description

Path name	Network node 1	Network node 2	Distance	Rules

Custom logic

Object name	Trigger	Description

d. Fluid objects

1. Base units of flow (1 <unit> per <time>)
2. Tick value = 1 <time>
3. Operations

			Port Rates			
Object name	Description	Capacity	Obj. In	Obj. Out	Port In	Port Out

7. Tools

a. Global Tables (*include tables here or attach*)

Table name	Where used	Purpose	Row defin.	Col. defin.

b. Time Tables (*include tables here or attach*)

Table name	Where used	Purpose

c. Downtime Tables (*include tables here or attach*)

Table name	Where used	Purpose

d. Dashboards

Table name	Purpose

e. Process Flow logic

Table name	Purpose

f. Global Lists

Table name	Where used	Purpose

g. Global Variables / Macros

Name	Type	Where used	Purpose	Init. value

h. User Events

Name	Where used	Purpose

i. Custom objects (*those developed or brought in from other libraries*)

Name	Type	Where used	Purpose	Library name

j. Other (*specify*)

8. Random number stream assignments

Stream number	Where used	Use

9. Interfaces (*describe how users interact with the simulation model*)
   a. MSExcel import/export
   b. MSAccess import/export
   c. Graphical User Interface
   d. Output and report
10. Model views (*screen shots of model*)
11. Model Validation
   a. Validation plan

b. Validation results

12. Model Verification
    a. Verification plan
    b. Verification results
13. Input data analyses
14. Simulation model version history

Version	Date	Main changes

Part IV - Analysis and Results

1. Analysis questions
2. Definition of performance measures
3. Definition of decision variables
4. Description of analysis process/methodology
5. Tactical information
    a. Initial conditions and basis
    b. Run length and basis
    c. Number of replications and basis
    d. Length of warm-up period and basis
6. Experiments
    a. Description and rationale for experimental design
    b. Input/Output Summary

Scenario	Decision variables					Performance measures			
	Name 1	Name 2	Name 3	...		PM 1	PM 2	PM 3	...
<name>	Val_1								

7. Analysis of simulation experiments
8. Conclusions and recommendations

About the Authors

Malcolm Beaverstock, Ph.D. After a 40 year career applying advanced control and simulation to industrial problems, Malcolm Beaverstock retired to join FlexSim. During his 13 years with General Mills, he developed and led their simulation program which was involved with more than 300 projects and resulted in significant savings attributed to the use of simulation.

Following graduate studies, Malcolm joined UniRoyal Chemical as manager of their advanced control group where he installed direct digital control systems. During his 17 years at the Foxboro Company he managed various technology groups and established the systems research department that resulted in the early development of Foxboro's I/A control system. As Vice President of Automation Technology, Malcolm initiated work into model-based control applications.

Malcolm holds a Bachelor's degree in Chemical Engineering and Labor Relations from MIT and a Doctorate from Cornell University in Chemical Engineering and Computer Science. He's the author of more than 200 papers on the application and use of advanced technologies.

Allen G. Greenwood, Ph.D., P.E. Allen is Professor Emeritus of Industrial and Systems Engineering at Mississippi State University (MSU) and is currently Professor of Engineering Management at Poznan University of Technology (PUT) in Poland. Allen teaches systems simulation, logistics systems design, enterprise systems engineering, and project management. His research interests/expertise include the design and analysis of production and project systems; simulation modeling, analysis, and optimization; and the design and application of decision-support systems.

His professional experience spans a wide variety of domains -- engineering design and development (military aircraft and aerospace), manufacturing and production systems (military aircraft, shipbuilding, automotive, textile fibers, healthcare, and electrical systems), and project management. His work has been funded by such organizations as the US Air Force Research Laboratory, Office of Naval Research, Naval Sea Systems Command, NASA, Northrop Grumman Ship Systems, Nissan North America, General Electric Aviation, and the Center for Advanced Vehicular Systems (MSU). He has authored or co-authored over 150 creative works, including journal and conference papers, technical reports, software programs, etc.

Allen received his B.S.I.E, M.S.I.E, and Ph.D. (Management Science) degrees from North Carolina State University, University of Tennessee, and Virginia Tech, respectively. In addition to the faculty positions at MSU and PUT, he held positions at American Enka Company, General Dynamics Corporation, Virginia Tech, Northeastern University, the American University of Armenia, and Prince Sultan University.

William Nordgren, MS CIM. Bill is the President and CEO of FlexSim Software Products, Inc., and currently teaches the Manufacturing Simulation graduate course as an adjunct professor at Brigham Young University. In 1988 he founded ProModel Corporation and was Vice President until he left in 1992. In 1993 Bill founded F&H Simulations, Inc. (now FlexSim Software Products, Inc.) and introduced Taylor II, Taylor ED, and FlexSim into the market.

Bill has authored several papers dealing with simulation project management, queuing theory, and has taught hundreds of classes in the use of simulation software. He is listed in Marquis Who's Who in America for his accomplishments in the advancement of simulation technology. Bill received a Bachelor of Science in Manufacturing Engineering Technology, and a Master of Science in CIM (Computer Integrated Manufacturing) from Brigham Young University.

Index

A* algorithm 155, 157, 414
activity 17, 49, 52, 66-67, 373, 375, 377-379, 381, 385, 387-389, 413-416, 437-440, 442, 448, 453, 467
Advanced User 35-37, 325, 354, 373, 414
AGV 149, 414
Analyst 103, 414, 418
Arrival
process 45, 47, 60, 78, 80, 243, 314, 320, 448
sequence 18, 88, 93, 111
ASRS 152, 414
assumption 49-50, 147, 197, 229, 241, 262, 292, 414
attribute 97, 133, 153, 334, 345-346, 348, 414, 427
availability 31, 47-48, 50, 58, 81, 173, 178, 180, 192, 414, 425, 428
balking 46-47, 56, 58, 66, 70-73, 80, 414
batch 18, 27-29, 88, 97, 111-112, 140, 158, 320, 399, 401, 404-406, 414, 421
Beta distribution 210, 414
binomial distribution 207, 414
bottleneck 26-27, 95, 169, 180, 188, 305, 350, 404, 414
boundaries 43, 373, 414, 422, 437-438, 442-443, 449, 453, 463, 469
Boundary assumptions 414, 438, 463, 469
breakdowns 18, 60, 81, 86, 115, 177, 182-183, 190, 289, 351, 414, 448, 452, 457, 465
Breakdown tab 182, 414
by expression 124, 135, 138, 414
C++ 157, 331, 414
CAD 414, 439-440, 464
center port 184, 329, 335, 354, 414
changeover 48, 288, 414, 440
combiner 112, 139-141, 149-150, 159, 162, 183, 188, 191, 360-361, 384, 389, 398, 414
Common Random Numbers 288, 293, 295, 298, 300, 318, 320, 414
competing failure 36, 181, 414
complexity 18, 28, 30, 33, 35, 37, 39, 55, 105, 119, 191, 197, 257, 290, 349, 373, 414, 428, 434
conceptual model 414, 416, 431, 433, 440-441, 448, 458-459, 469
confidence interval 264-265, 269, 271, 279, 282, 296-297, 300-302, 316, 414
connections 115-119, 122, 137, 157-158, 353, 383, 400, 414, 440
container 22, 112, 140-141, 164, 390, 399, 414
content 104, 111, 123, 125, 162-163, 177, 290, 314, 338, 354, 387, 389, 399-400, 414
continuous simulation 14, 395, 414
controller 357, 396-397, 402, 414
Conveyor 95, 105, 111-112, 114, 118-124, 128-129, 131-132, 144, 161, 181, 185-186, 188-191, 288-291, 311, 327, 359, 367, 375, 380, 390, 398-399, 414, 438, 447, 453-454, 462, 464-466, 468
object 118, 121, 123
current object 136-137, 334-336, 338, 345, 354, 414
custom code 177, 278, 328, 331, 337, 339, 343-348, 375, 389, 414
custom logic 36, 137, 142, 278, 325, 331, 342, 355, 387, 414, 471-472
cycle time 58, 78, 80, 107, 350-351, 358, 376-379, 414, 426, 451
data node 328, 414
decision support 18-19, 39, 87-88, 413-414, 417, 420-421, 429, 431
decision variables 86-87, 89, 125, 257, 259, 266, 277, 287, 291-292, 294, 306-307, 318, 414, 463, 469, 474
degree of complexity 33, 414
dempirical 207-208, 222-223, 337, 414
Designer 104, 414, 417
design of experiments 36, 294, 414, 460
developer 133, 414, 417

Discrete
empirical 207, 223
event 395, 426
object 400, 410
dispatcher 154-155, 184, 415
domain knowledge 415, 419
dot notation 334-335, 345-346, 415
Down Function 184, 415
downtimes 31, 48, 74, 111-112, 173-174, 181, 184, 190-191, 415, 439, 449, 462, 464
drop-down menu 109-110, 113, 124, 136, 158, 184, 415
Editing objects 114, 415
Educator 415, 418
efficiency 16, 19, 29, 95, 176, 186, 415, 463
empirical distribution 203-208, 214, 223, 241, 415
errors 26, 337, 339-341, 389, 397, 415
experimental design 258, 265, 287, 305, 308, 318, 415, 419, 474
Experimenter 87, 258, 277-279, 287, 291-293, 307-308, 312, 318, 343-344, 346, 351, 353, 415, 417
ExpertFit 199, 202, 207-208, 218, 236, 292, 415, 417
exponential distribution 56-57, 59, 62, 83, 178, 197, 201, 212, 292-293, 344, 415
failure clock 178, 180-183, 185, 415
failure mode 183, 415
FIFO 52, 55, 58-59, 68, 82-83, 169, 236, 238, 240-241, 244-245, 249, 259, 291-292, 294, 296-298, 301, 303-304, 308, 310, 415
first available 47, 49, 118, 136-137, 158, 381, 398, 415
fixed resources 110-112, 117-119, 132, 139, 415, 449, 455, 470
flag 159, 351, 370, 384, 387, 389, 415
Flexscript 157, 328, 331, 333-334, 337, 340, 342, 354, 415
Flexscript commands 328, 337, 354, 415
Flow
Flowitem 110-112, 114, 118-119, 122-123, 134-137, 140-141, 149-150, 153, 159, 172, 329-330, 332, 335, 345-346, 348-349, 354, 359, 361-364, 370-371, 375, 380-381, 383, 385, 399, 402, 415
logic 124, 129, 137, 141, 348, 374-375, 377, 385, 393, 473
trigger 150, 161-162
Fluid 14, 28-29, 37, 110, 337, 395-402, 406, 409-410, 415-416, 424, 426-428, 472
blender 398, 410
flow 37, 395-396
Generator 397
mixer 398, 401, 410
objects 110, 337, 397, 399, 401, 410, 472
pipe 398
splitter 398
tank 398-399
terminator 397
to item 399
flypath 165-166, 415
fly-through 165-166, 415
functional specification 85, 415, 436-437, 440, 459-462
General tab 115-116, 119, 121, 162-163, 358, 362-364, 402, 415
global table 138-139, 150, 205-208, 223, 278, 343, 345-346, 360-361, 376-377, 384, 389, 415
global variable 340-341, 351, 354, 383, 388, 415
grid 108-109, 112-113, 153-155, 279, 310, 335, 415
GUI 89, 327, 415, 420-421
healthcare 13, 23, 32, 49, 415, 450
highlighting 138, 310, 415
ICOM notation 415, 442-443
Ideal rate 176, 415
IDEF0 415, 440, 442, 447
Implementer 415, 418
inference 53, 126, 199, 201, 257-258, 265, 287, 415
input port 117-118, 150, 335, 354, 415
Intermediate User 35-37, 325, 331, 360, 415

interval estimate 264-265, 271, 282, 292-293, 297, 416
Investigator 416, 418
Item to Fluid 399, 416
itemtype 134-137, 147, 160-161, 416
job shop 174, 280, 416
Jockeying 47, 416
join 31, 46, 95, 112, 122, 140, 159, 247, 290, 345, 416, 457
Kendall's notation 55, 416
label 134-135, 145, 159, 302, 334, 345-346, 348-349, 358, 360, 377-379, 381, 385, 387, 416
Label Table 161, 360, 416
lean 15-16, 18, 27, 31, 34, 43, 50, 185, 191, 404, 406, 416, 421, 423-424, 431
level indicator 400, 416
library 34, 109, 112-113, 119, 122, 125, 142, 145, 150, 152, 157, 162, 164, 249, 374-375, 377, 397, 416, 419
linear congruential generator 218, 416
list 115, 158-163, 172, 239-240, 243-244, 246, 249, 326, 336, 375, 385, 388-389, 436
logic 32, 35-37, 66, 69, 107, 120, 124, 129, 133-138, 154, 157, 159, 162, 198, 238, 243-245, 249, 325, 330-331, 338, 340-342, 348, 351, 353, 355-356, 367, 373-377, 380, 382-383, 385-389, 393, 397, 416, 423, 457, 459
main screen 108-109, 396, 416
Main toolbars 109, 416
manufacturing cell 280, 288, 356, 416, 450
material distribution 30, 416
Maximum rate 176, 416
members 182-184, 259, 353, 407, 416, 434, 436, 463
message 134-135, 138, 338, 341, 355-358, 371, 377, 416
message trigger 134, 416
methodology 104, 117, 275, 306, 412, 416, 433-434, 440-442, 450, 456, 474
metrics 30, 86, 89, 91, 131, 144, 147, 177, 179, 367, 409, 416, 421, 437
mission length 178, 416
Mixer Recipe 401, 416
Mixer Steps 401, 416
mobile resources 110, 151, 155, 157, 416, 448, 471
Model
  M/D/1 62-63, 416
  M/G/1 60, 62-65, 80, 416
  M/M/1 55-56, 58, 60-63, 65, 72, 81, 259-260, 263-264, 267-268, 271-273, 275, 288, 416
  M/M/2 56, 58, 62-63, 65, 259, 263, 272-273, 416
  M/M/s 55-56, 58, 60-63, 65, 72, 81, 259-260, 263-264, 267-268, 271-273, 275, 288, 416
modeling 38, 41, 76-77, 88, 103, 106-107, 151, 195-196, 218, 234, 249, 292, 414, 416-417, 419, 427-428, 433, 440-441, 464
Monte Carlo simulation 14, 196, 227-228, 231-235, 250-251, 416
MTBF 173, 178-185, 192, 350, 376, 416
MTTR 178-185, 192, 350, 416
multiple comparisons 287, 302-303, 318, 416
Multiprocessor 111, 398, 416
naming convention 114, 416, 470
network node 155, 416
node 155, 328-329, 333, 336, 340, 414, 416, 419, 471
non-terminating system 266, 272, 416
normal distribution 57, 197, 199, 201, 205, 208, 210-211, 216, 229-231, 233, 255, 269-270, 285, 300, 323, 347, 417
Object
  attributes 336
  flow diagram 35-36, 433, 440, 464-465, 467, 469
  nodes 328
  statistics 35, 125
  variables 401
objective 13, 66, 85, 126, 133, 151, 158, 173, 201, 227-228, 231, 257, 287, 291, 306-308, 310-312, 315, 317, 325, 342, 373, 413, 417, 421, 433, 437, 465

Occasional User 34-37, 85-86, 88, 102-103, 417, 431, 433, 461
OFD 35-36, 417, 440-442, 444, 448-451, 453-455, 458-459, 465-466, 469
operability 417, 423, 460
Operating assumptions 417, 438, 463, 465, 469
operations system 19, 27, 39, 102, 150, 172, 192, 220, 417, 423
optimization 278, 287-288, 306-308, 310, 314, 316, 318, 320, 417, 421
output port 117, 330, 335, 354, 417
pack 94, 112, 140, 181, 233, 417, 462
paired-t test 295-297, 300, 417
passing 154, 417
performance 18, 26-27, 30, 35, 41, 43, 45, 50-52, 54, 56, 58-59, 61, 63, 66-67, 69, 73, 76, 86, 90, 94-95, 125, 176-177, 192, 198, 257-258, 261, 263, 265-269, 272-273, 275, 278-279, 287, 294, 296, 298, 304-305, 307, 318, 417, 423, 437, 448, 460-461
performance measure 67, 69, 77, 196, 263, 266, 272, 275, 279, 291, 296, 342-343, 417, 454
picklist 115, 136, 140-141, 163, 233, 331-332, 417
picklist options 115, 140-141, 331, 417
pilot run 270-271, 291-293, 417
Planning rate 176, 417
point estimate 264, 292-293, 417
preempt 152, 184, 193, 417
priority 46, 52, 96-97, 146, 152-153, 168-169, 184, 259, 350, 417, 424
probabilistic 125, 249, 258, 417
probability distributions 45, 47-48, 77, 86, 195-198, 201, 203-204, 208, 210-211, 214-215, 218-219, 222-223, 227-228, 233, 235-236, 249, 258-259, 417
procedure 135, 142, 240, 352, 385, 411, 417, 458, 464, 467
Process Flow 373-375, 377, 379-383, 385-389, 392-393, 416-417, 473
processor 114, 118, 124, 135, 182-183, 185, 328-329, 331, 335, 341, 349, 358, 362-363, 375-379
product data 346, 417
production line 27, 31, 54, 417, 422, 438, 461
production schedule 18, 417, 459
programmed model 416-417, 431, 440-441, 458-459
project template 417, 433-434, 458, 461-462, 467-469
Properties window 114-115, 121, 134-135, 159, 249, 417
pull 15, 32, 49-51, 120, 136-137, 150, 158, 160-163, 353, 388-389, 417
push 50-51, 146, 160-162, 363, 385, 417
Queue 46-47, 50-52, 55, 58-59, 66-72, 76-78, 80, 83, 100-101, 111, 114, 118-119, 122-123, 131-132, 137, 150, 154, 159-162, 186-191, 236-241, 243-249, 252, 254, 259-261, 272, 275, 278-279, 291-292, 294, 296, 298, 301, 303, 308, 310-311, 314-315, 317, 320, 327, 338, 346, 350, 360, 366, 375, 382-389, 396, 398, 417, 454
discipline 51-52, 55, 58-59, 83, 244-245, 249, 259, 291-292, 294, 298, 303, 308, 310
object 118, 248, 346
queuing theory 268, 418, 456
random number generator 196, 215, 222, 231, 241, 243-245, 418
random sampling 227-228, 231, 258, 298, 418
Recorder 166, 418
reliability 28, 35-36, 48, 115, 173, 177-180, 182-184, 187, 192, 195, 234, 258-259, 418, 428, 459
reneging 47, 56, 58, 66, 70-73, 80, 418
repair time 178-180, 182, 220, 418
replication 75, 81, 261, 263-266, 268-269, 272-273, 277-280, 287, 291, 293, 296, 304, 308, 312, 314, 317, 346, 418
Researcher 418
reset 89, 109, 114, 120, 135, 183, 247, 275, 312, 346, 375, 385, 389, 396, 418
resources 31, 50, 110-112, 117, 132, 139,

151, 155, 185, 289, 305, 423, 436, 442, 448-449, 455-456
resource sizing 418, 433
round robin 137, 150, 418
run length 254, 266, 269, 272-273, 275, 291, 418, 474
run time 97, 109, 174, 247, 272, 278, 308, 418, 437
sample size 125, 198-199, 201, 233-234, 251, 261-262, 264-265, 269-270, 292, 295, 297-298, 418
sampling 58, 135, 196, 198-199, 210-211, 214, 227-229, 231, 234, 258, 261-265, 298, 300, 305, 307, 418
scenario 15, 73, 222, 257, 277-279, 287, 291-292, 304-305, 307-308, 312, 418, 460-461
screen elements 163, 418
selecting 116, 198, 420
senddelayedmessage 337, 355-356, 377, 418
sendmessage 355-356, 377, 418
Send to Port 136-137, 150, 160-161, 418
separator 112, 140-141, 149-150, 398, 418
service operations 13, 22, 106, 418, 450
service process 46-48, 53, 60, 62, 77-78, 196, 314-315, 418
Simulation
control panel 109, 341
knowledge 20, 33, 419
languages 355, 425-426
project 33, 35, 87, 209, 258, 265-266, 275, 389, 412-414, 416, 418, 427, 429, 431, 433-437, 439, 441, 443, 458-462, 465, 467, 469
simulators 21, 86, 247-248, 418, 425-428, 444
sink 111, 124, 145, 182, 185, 240, 311-312, 327, 345, 375, 379, 381-382, 385, 388, 397, 418, 447, 449
source 124, 138, 145, 160, 182-183, 185, 188, 300, 311-312, 334, 345, 363, 375, 381, 385, 402, 449
space 19-20, 25, 74, 88, 90, 92-93, 105, 107-108, 111, 119, 123, 148, 163, 178, 288, 307-308, 310-311, 327, 343, 374, 377, 381, 396, 418, 452, 455
SQL 158, 160-161, 418
staffing 15, 21, 26, 29, 88, 90-91, 368-370, 418, 423, 464
stakeholders 25, 43, 85, 412-413, 416, 418-420, 428, 431, 442, 448, 458, 467, 469
starting conditions 271-272, 275, 317, 418
state 52, 55, 58, 66-67, 69, 72, 106, 116-117, 125, 134-135, 138, 151, 158, 175, 178, 183, 185, 195, 236-237, 239-240, 243, 247-248, 252, 264, 267-269, 271-273, 275, 278, 289-290, 378, 418-419, 421-422, 425, 428-429, 448
statistics tab 89, 164, 419
Staytime 125, 177, 249, 419
steady state 264, 267-269, 272-273, 275, 419
stochastic 41, 86, 102, 125, 258, 260, 306, 419
stop time 91, 109, 341, 357, 419
subnodes 328, 335, 419
success factors 411, 419, 428-429, 431
surge 27-28, 31, 97, 185, 193, 398, 419, 423
switch 158-159, 162, 331, 333, 338-339, 356, 419
system dynamics 14, 27, 67, 77-78, 93, 273, 419
Systems modeling and analysis
life cycle 413, 431
process 258, 414, 416, 433
project 413, 417-418
target 140, 176-177, 186, 198, 350, 397, 419
Target rate 176, 419
task executers 110, 112, 132, 151-155, 157, 176, 184-185, 419, 471
terminating system 267, 275, 419
throughput 26-27, 58, 78, 80, 88, 97, 116, 125, 131, 148, 170, 177, 185, 263, 266, 280, 306, 311-312, 419, 437, 463
ticker 396-397, 419
Time controls 109, 419
token 374-375, 377-382, 387-389, 419
Tools tab 174, 419
transient 27, 268-269, 275, 419

tree 326-329, 333, 354, 419
triangular distribution 169, 208-209, 293, 315, 419
Trigger 123, 134-135, 137-138, 145, 147, 150, 161-163, 188, 312, 329-330, 332, 334-335, 341, 343-349, 351, 355-357, 361, 363-364, 377, 381, 384, 387, 389, 402, 415-416, 419
OnCreation 330, 344
OnEntry 137, 329, 335, 361, 363, 384
OnExit 137, 330, 345
OnReset 341, 346, 357, 384
tab 134, 344
Troubleshooting 341, 419
uniform distribution 201, 211-213, 233, 333, 419
Unpack 112, 141, 150, 419
User 20, 34-37, 85-89, 103, 105, 107, 138, 142, 218, 325, 397, 417, 420-421, 427-428, 458, 461, 473
interface 88-89, 105, 107, 329, 332, 420-421, 458, 473
library 142, 150, 164
Use Transport 136-137, 152, 162, 402, 419
utilization 25, 50, 53, 56, 60, 62-65, 67, 69, 73, 76-78, 95, 106, 131, 144, 147, 247, 259, 265, 278, 280, 314-315, 317, 419, 437, 450, 454, 462
validation 33, 416, 419, 423-424, 429, 431, 442, 458-460, 473-474
value stream map 35, 419, 440
variables node 328, 419
verification 215, 419, 429, 431, 458-460, 474
virtual distance 123, 155, 419
Visual Display 163, 419
visualization 119, 151, 153, 163, 188, 200, 275, 325-326, 362, 370, 374, 389, 406, 419, 427, 458
visual tool 163-164, 172, 357, 419
wizards 103, 419
zone 380-383, 419