School of Engineering and Design

MSc Programme

Course notes

INTRODUCTION TO SYSTEMS MODELLING & SIMULATION

Edition 3: October 2011

Alireza Mousavi

With Special Thanks to Alexander Komashie, Ali Moen-Taghavi and Vahid
Pezeshki

Electronics and Computer Engineering / Advanced Manufacturing and Enterprise
Engineering

School of Engineering and Design, Brunel University

Important Note to the Students

This course book is written in two parts, Part A and Part B. Part A mainly covers the <u>theoretical</u> part of the module. Part B covers the <u>practical</u> part of the module. So we may not exactly follow the chapters and the subjects in the numerical sequence of 1, 2, 3… as they appear in this book.

To maximise your learning experience and ensure that all the materials discussed in the module are covered, in addition to this book, I recommend the following further readings:

On Theory:

1. R. G. Askin and C. R. Standridge (1993); Modelling and Analysis of Manufacturing Systems; John Wiley & Sons, Inc.

2. M. P. Groover (2001); Automation, Production Systems, and Computer Integrated Manufacturing; Second Edition; International Edition; Prentice Hall International, Inc.

3. G. L. Curry and R. M. Feldman (2011); Manufacturing Systems Modeling and Analysis; Second Edition, Springer.

On Practice:

4. D. Kelton, R. Sadowski and N. B. Swets (2010), Simulation with Arena 5th Int. Edition, McGraw-Hill.

Table of Content

PART A

CHAPTER 1

INTRODUCTION TO SYSTEMS AND SYSTEMS ENGINEERING

This Chapter Covers:

1. Definition of Systems

2. Systems Schools of Thought

3. Challenges that Systems Managers Face

1.1 System and System Engineering

A system is a set of interacting elements that seek a common goal. It can represent a transformation process in which it converts a set of inputs into a set of outputs. The inputs and outputs of a system are the main interfaces between the system and the outside world. The process within a system encompasses the totality of constituent elements including objects and their relationships. (Figure 1.1)
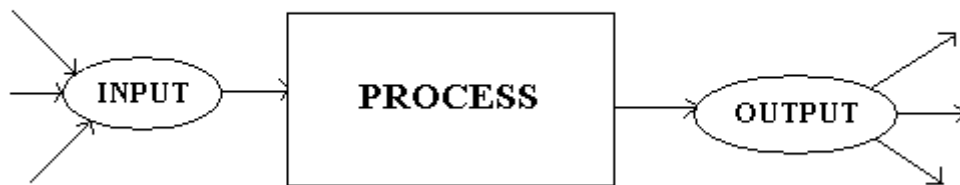
Figure 1.1: Schematic representation of a system

A system may represent ongoing processes, and at any time the state of one or more objects within the system may change (state change). A system consists of a number of different elements that normally follow a specific logic and discipline[1]. The property and behaviour of these elements contribute to the property and behaviour of the system as a whole and in an organised manner. So a system can be defined as a collection of components which are interrelated in an organised way and work together - e.g. people and/or machines- towards the accomplishment of certain logical and purposeful end. This definition implies that a system must have the following features:

1. An assembly of components: These components are the structural, operation, and flow parts of the system which can be individually identified. System components can also be identified as input, process, output, feedback control and constraints. The

---

[1] Although in modern systems and mathematics "Chaos" has become a fascinating subject.

input and output in a system may sometimes also be referred to as the cause and effect, respectively.

2. Components connected in an organised manner: This indicates that the relationship between system components is important. Each component is related directly or indirectly to every other component in the system and is affected by them. Without relationships there will be no system.

3. A logical objective or purpose: For every output or effect, there exists a definite set of inputs of causes that influence and produce the expected output.

4. Components which work together towards the common objective: It is the totality of the components which together with their attributes and relationships, constitute a particular system and provide the output for each given set of inputs.

In order to design or study the state of a system, a systems engineer should be able to:

- Identify the components of the system that they are designing and/or studying;

- Understand the role and the relationship between the components of the system;

- Recognise and capture the logical relationship between the components and the sets of inputs and outputs of the system; and

- Infer from the sets of inputs, outputs and the interrelationship between the system components, the state of the system (if it is to be designed) or the objectives of the system (if it already exits).

Systems Engineering therefore, not only requires theoretical knowledge but also the ability to visualise things in their totality. So you could consider it to be a form of *Art*!

Using the same analogy one could consider having the capability to design, maintain and interpret the state of something using scientific means makes one a Systems Engineer. A Mechanical Systems Engineer is an engineer that studies the

relationship between mechanical entities and designs mechanical systems. An electrical or electronics engineer has a strong appreciation for electrical or electronic components and their interrelationship, so they can build complex control systems that manage and predict the inputs and outputs of an electronic system. This analogy can be extended to other systems e.g. manufacturing, financial, transport, aerospace … In short all of us are in one way or another *Systems Engineers!*

There are three schools of thought in approaching systems. Some scholars approached systems as a set of predefined interlinked components *(Mechanical Systems). The Organists* challenged the static world view of Mechanists by introducing human factors (*Adaptive Systems).* And finally, the modern *Sustainable Systems Theorists* that try to explain the state of systems by understanding the more complex interrelationships between the building blocks of the system and the systems effect on its environment (*Viable Systems).*

1.1.1 The Mechanical System

From a Mechanists point of view, a system is the aggregation of interrelated parts where the whole is equal to sum of parts. The constituents of a mechanical system are standard parts with defined relationship between each component (e.g. bicycle, car, computer etc). The emphasis here is on the performance of each part, in which they follow *pre-determined* and *repetitive* set of rules and functions to fulfil specified objective(s).

Other features of Mechanical systems are that they have minimal adaptability to the changes that happen in their environments. They are designed as closed feedback loops and any sudden changes to the environment may significantly impact their performance and survival.

For example, a bicycle is a mechanical system that consists of a number of components that are joined together for a specific purpose. A missing component of a bicycle renders it as an incomplete bike or obsolete apparatus.

Mechanical System
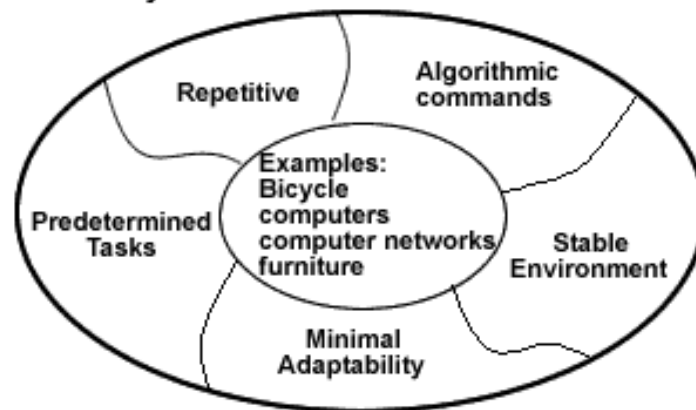
Figure 1.2: A schematic representation of a Mechanical System

Some analogy and food for thought…

- *Imagine how a computer would be if it does not get regular software or hardware updates…*

- *Imagine if you leave a bike in the outdoors for a long time…*

For a long time[2], the *Mechanists* enjoyed and propagated their view of system very successfully. But with better understanding of nature and theories of adaptation, fuelled with advances in technology the first challenges to *Machinists* came from biologists and later human relations theorists. The understanding of the principles of *natural selection* and *evolution of biological systems[3]* will make a very useful read to appreciate the foundations of adaptive systems. But in order to keep this chapter short and straight to the point, we will only concentrate on industrial adaptive systems in which human relationships and smart computing generates *synergetic* properties.

---

[2] Probably from the dawn of civilisation, and later development of science and technology.

[3] Charles Darwin.

1.1.2 The Adaptive System

The Organists challenged the Machinists, by arguing that respect for people social and psychological needs will improve the effectiveness and efficiency of operations.

The *organists* describe a system as a set of interlinked elements with synergetic properties. The whole can be greater than simply the sum of the constituents of the system. The components of the system constantly or at defined intervals interact with their environment (open architecture). They sense their surroundings and try to correct their internal relationships to respond to the changes in their environments. Also they are capable of adapting to the changes of their input signal and adapt their processes to produce the expected output. Thus, adaptation and survival in complex situations are the objectives that adaptive systems pursue.
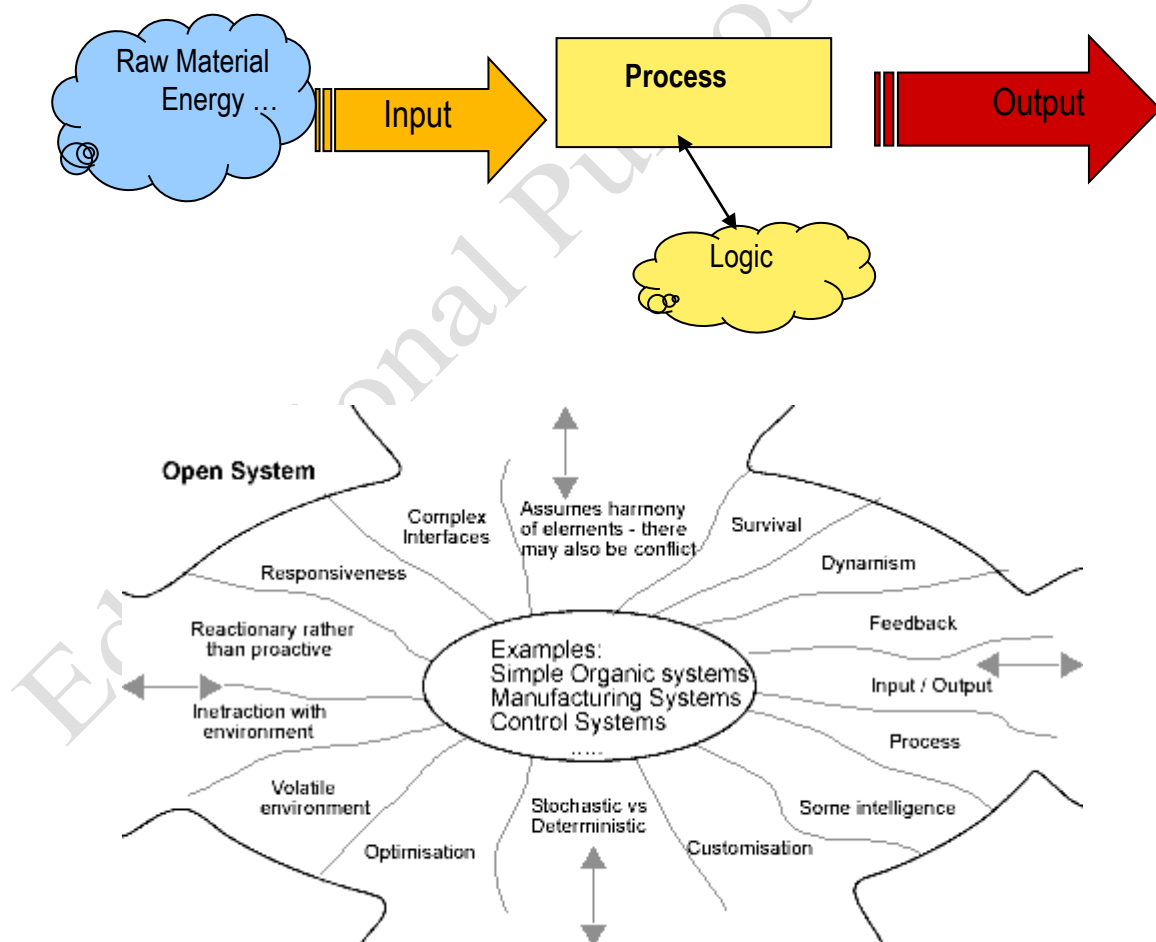
Figure 1.3: An Adaptive System (Open Architecture)

In the last 30 to 40 years successful companies have aspired and demonstrated that they are highly dynamic and adaptive organisations. They have successfully steered their ship in the stormy waters of economical, industrial, social and technological changes. They most importantly have adapted to the ever changing consumer (customer) needs and tastes.

Some analogy and food for thought…

- *Imagine a plant when sudden climate change occurs…*

- *Imagine animals and what happens if their surrounding is not capable of supplying them with sufficient food and shelter…*

The adaptive systems theorists managed to explain and design industrial systems that were capable of capturing and interacting with the dynamism of their environment. But their models are now falling short of interpreting and capturing the more complex systems that have emerged in the interrelated world of 21st century.

1.1.3 The Viable System

You may consider the Viable Systems theorists as *Holists.* Holists describe systems as interacting networks that in addition to their constituent elements <u>govern</u> the complex interactions between functional, socio-economical, cultural and political elements. These systems not only adapt to their environments but have the cerebral capability to influence and change their environment to their advantage. Rather than follow the trend, they have the ability to accurately predict the future and lead the changes.

Viable systems emphasis is on: (a) aggressive prediction, (b) active learning and (c) persistent monitoring and control of the environment. Viable systems aggressive evolution and success is heavily dependent on enhancing their capacity and ability to obtain data (Data acquisition) to utilise the information of the past, combine it with the present data to understand the current state. Moreover, use the present and past information to accurately predict the future. These system have substantial resources to process information whilst actively monitoring their environment in real-time. They are capable of not only adapting to changes but also *influence and change* the

environment to their advantage. The so-called sustainability cycle, that requires Creativity, Innovation and Reinvention at given times in their life cycles.

The highly creative and innovative nature of viable systems allows them to expand and contract at the right times during the global socio-economical fluctuations.
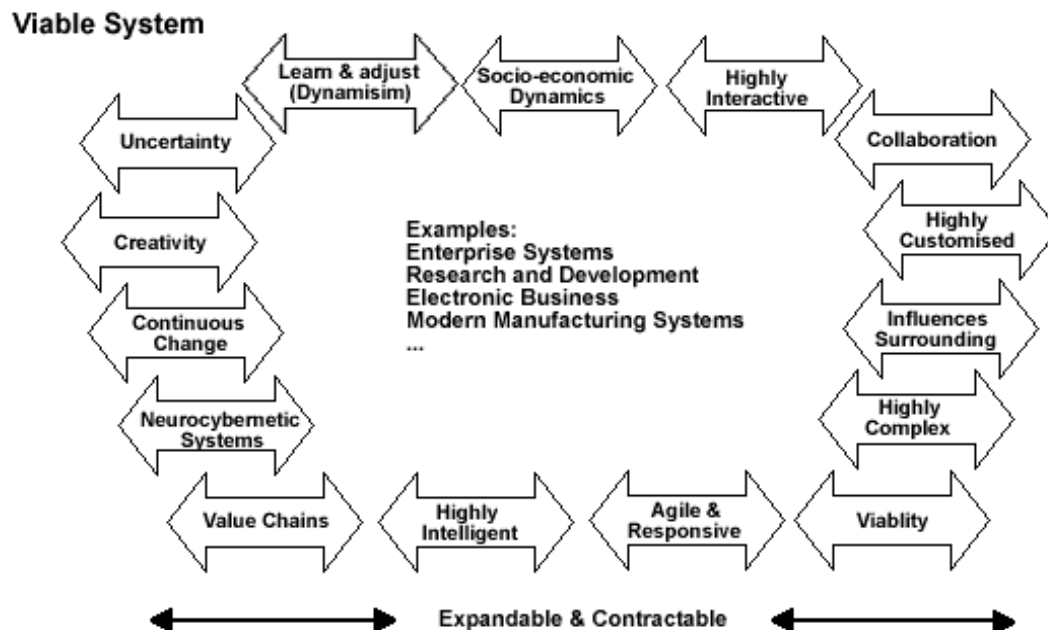
Figure 1.4: A schematic overview of a Viable System

Some analogy and food for thought…

- *Humans can be considered one of the most successful viable systems…*

- *Electronic and Telecommunication consumer product developers …*

- *Fashion industry…*

My aim here is to make you start thinking about all industrial systems around you. Make a distinction between them and find ways to categorise them into the type of system they are. Ask the question whether these systems are suitable candidates for evolving into viable systems. Suggest the necessary technologies and capabilities that those particular systems need to acquire for it to evolve into a viable system. You as a

systems engineer if given the opportunity design a viable system that can sustain itself for its given life time.

Human body is a perfect example of a viable system, think about this …
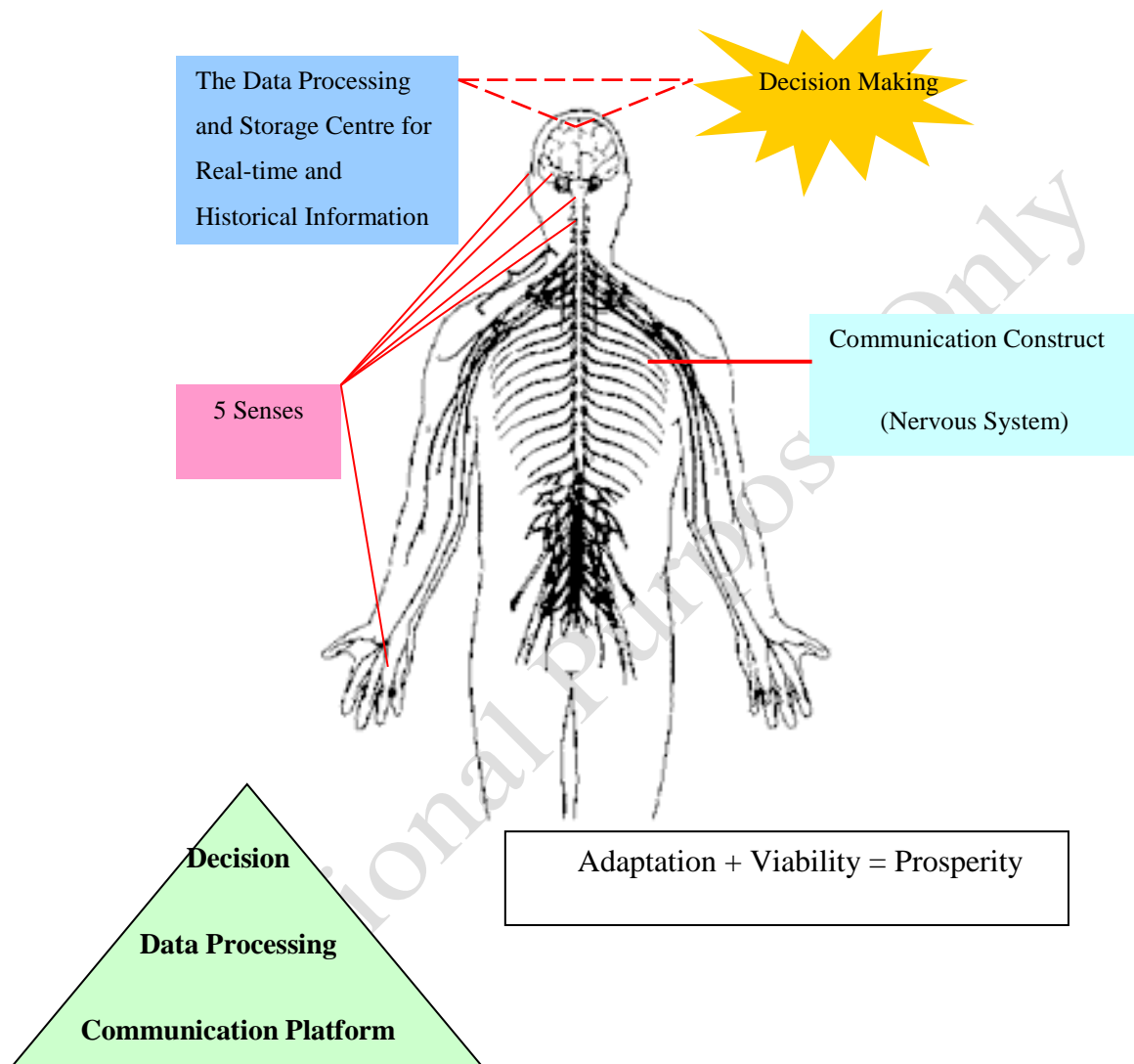
Figure 1.5: Human Body as a Viable System

One could use this analogy to describe a possible architecture of a viable industrial system.
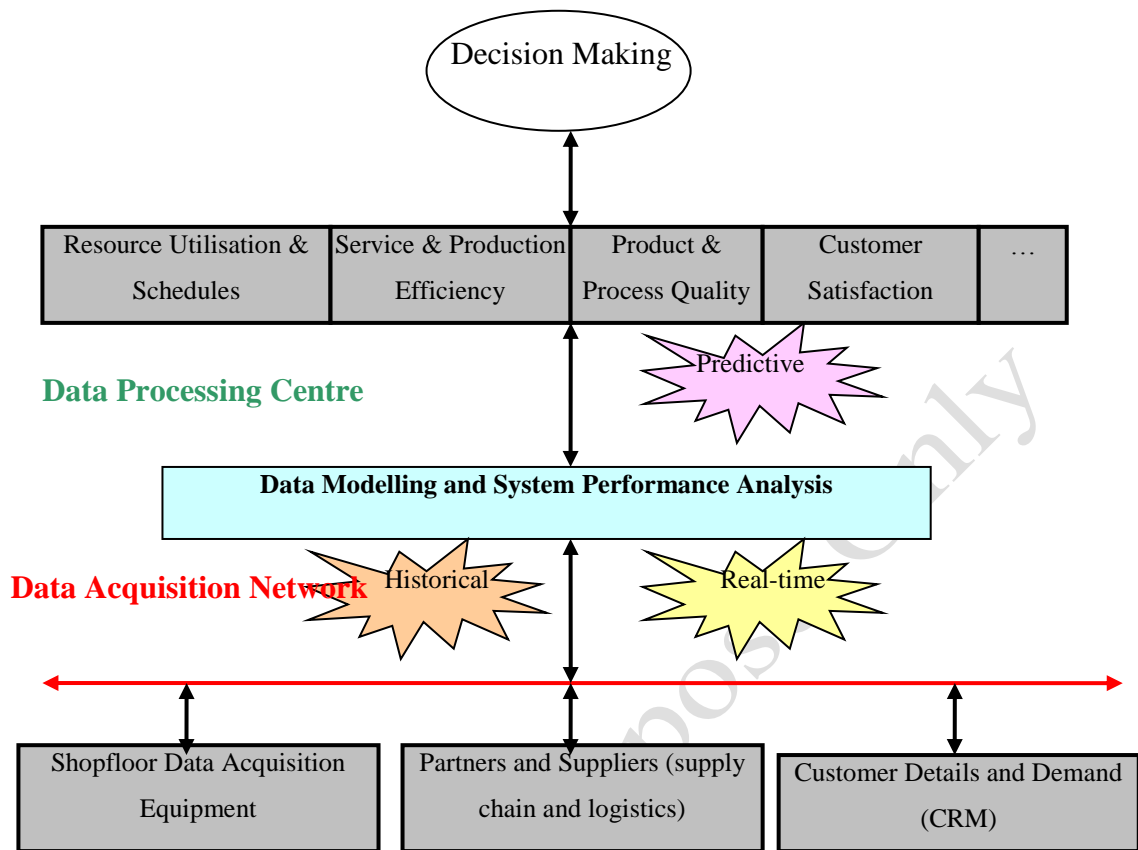
Figure 1.6: The Information Architecture of a Viable Industrial System – SinglX by A. Mousavi et al.

1.2 Data Modelling and Systems Performance Analysis

In this section we briefly discuss the importance of data collection and the science of translating input data into meaningful information.

The process of preparing and translating input data into meaningful information for systems performance analysis is called data modelling. There are various techniques that can be used for this purpose. These techniques can be as simple as logical *And, OR* and *IF* statements for binary system to complex data mining techniques such as: Statistical Process Analysis, Genetic Programming, Fuzzy Inference Analysis, Bayesian Belief Networks, etc. These analytical and physical models allow system analysts to interpret a series of input data into system state. Normally the input data are captured in a given time span.

In the following chapters, we will describe Systems Modelling and Discrete Event Simulation techniques as one of the most powerful mechanisms that is used to translate collected data during a time span into performance analysis tool. Here you just need to note that there is a mechanism and technique in acquiring information in which it is then used for modelling purposes.

We have two types of information *Historical* and *Real-time*. The historical data is collected over a period of time, validated and verified through statistical means and presented for modelling purposes. For example, average time that an operator processes a work that is assigned to her/him, or the average time it takes the computer processor to implement an algorithm. This data is normally collected at different times and for a period of time. By validating and verifying input data modellers can utilise the information to produce *Predictive* data that is derived from historical data. For example average number in a queue or waiting time can be estimated using the information about average meantime between arrival of work at a work station and the average processing time for that work station. *Do not fret!* This module is all about this, or mainly about this!

With the advent of modern real-time data acquisition technologies and their ubiquity, systems analysts are exploring the vast opportunities that access to real-time data provides. We are now utilising real-time data inputs into quick response decision management systems and also using Real-Time data to improve the quality of previously gathered historical data. At this stage it may suffice to intrigue you and conclude this chapter by Figure 1.7. This figure illustrates the relationship between Data Acquisition systems, real-time data modellers and Discrete Event Simulation packages. Combined together, the technologies produce one of the most sophisticated systems performance analysis capabilities available to us.
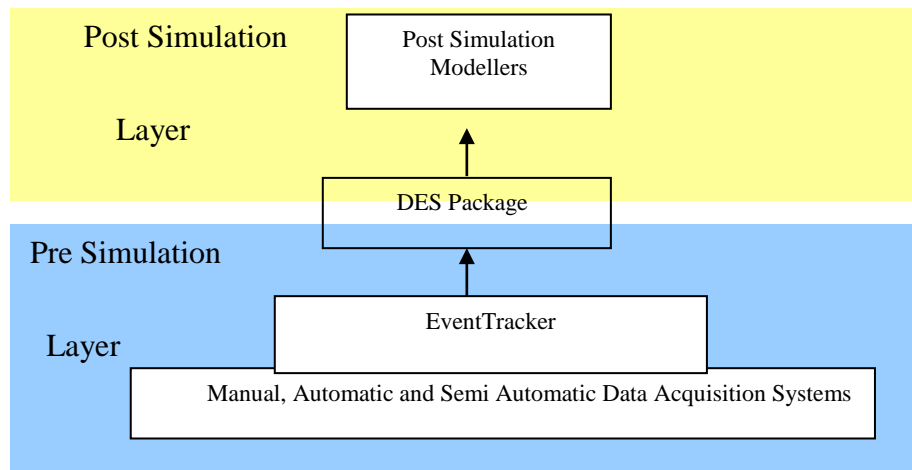
Figure 1.7: A Schematic overview of Integration of Data Acquisition Systems with Real-time Data Modellers, Simulation Packages and Post Simulation Modellers

An Introduction to Simulation Modelling

(Discrete Event Simulation)

This Chapter Covers:

1. Definition of simulation modelling

2. The potential advantages of simulation modelling

3. The types of simulation

4. Principles of simulation

5. Successful simulation project

6. Application of simulation in industry

2.1   What is simulation?

Simulation is the imitation of the operation of a real-world process or system over time. In other words, Simulation is the process of designing a model of a real system and conducting experiments with the model for the purpose of understanding the behaviour of the system and evaluating various strategies for the operation of systems. According to Schriber (1987) simulation involves the modelling of a process or system in such a way that the model mimics the response of an actual system to events that take place over time.

Simulation involves the generation of an artificial history of the system based on historical observations and translating that artificial history to draw inferences concerning the operating characteristics of the real system that it represents.

Simulation is used to describe and analyse the behaviour of a system, ask *what-if* questions about the real system, and aid in the design or improvement of real systems. Both existing and conceptual systems can be modelled using simulation.

In short, simulation reflects the behaviour of the real world in a small and simple way.

2.2   Importance and popularity of simulation

The number of businesses using simulation is increasing rapidly. More managers are realising the benefits of utilising simulation for more than just the one-time remodelling of a facility. Rather, due to advances in software, managers are incorporating simulation in their daily operations on an increasingly regular basis.

For most companies the benefits of using simulation go beyond simply providing a look into the future. These benefits are mentioned by many authors (Banks et al., 1996; Law and Kelton, 1991; Pegden et al., 1995; Schriber, 1991) and are included in the following:

1. Choose correctly: Simulation allows you to test every aspect of a proposed change or addition without committing resources.

2. Compress and expand time: By compressing or expanding time, simulation allows you to speed up or slow down phenomena so that you can investigate them thoroughly. For example you can examine an entire shift in a matter of minutes if you desire, or you can spend 2 hours examining all the events that occurred during 1 minute of simulated activity.

3. Understand why: Managers often want to know why certain phenomena occur in a real system. With simulation you determine the answer to the "why" questions by reconstruction of the scene and examining the system to determine why that phenomenon occurs.

4. Explore possibilities: One of the greatest advantages of using simulation software is that once you have developed a valid simulation model you can explore new policies, operation procedures, or methods without the expense and disruption of experimenting on the real system.

5. Diagnose problems: The modern factory or service organisation is very complex and it is impossible to consider all the interactions taking place in a given moment. Simulation allows for better understanding of the interactions among the variables that make up such complex system and subsequently increases ones understanding of their important effects on the performance of the overall system.

6. Identify constrain: Production bottlenecks give manufactures headaches. It is easy to forget the bottlenecks are effect rather than a cause. However by using simulation to perform bottleneck analysis, you can discover the cause of the delays in work in process, information materials, or other processes.

7. Develop Understanding: Simulation studies aid in providing understanding about how a system really operates rather than indicating someone's predictions about how a system may operate.

8. Visualise the plan: Depending on the software used, you may be able to view your operations from various angles and levels of magnification and even in three dimensions.

9. Build consensus: Using simulation to present design changes creates an objectives opinion. You avoid having inferences made when you approve or disapprove of designs, because you simply select the designs and

modifications that provide the most desirable results; whether it increases production or reduces waiting times for a service.

10. <u>Prepare for change:</u> We all know that the future will bring change. Answering all of the *what-if* questions is useful for both designing new systems and redesigning existing systems.

11. <u>Prudent investment:</u> Since the cost of a change or modification to a system after installation is so great, simulation is a wise investment. The typical cost of a simulation study is substantially less than 1% of the total amount being expended for implementation of a design or redesign.

12. <u>Train the team:</u> Simulation models can provide excellent training when designed for that purpose. It can provide the team and individual members with decision inputs to the simulation models as it progresses.

13. <u>Specify requirements:</u> Simulation can be used to specify requirements for a system design. For example, the specifications for a particular type of machine in a complex system to achieve a desired goal may be unknown. By simulating different capabilities for the machine, the requirements can be established.

14. <u>Capture complexity:</u> By providing a platform for abstraction complex relation between various elements of the system can be modelled and system performance indicators measured based on valid assumption.

2.3   Types of Simulation

Simulations can be classified as iconic and symbolic. Flight or driving simulators are examples of iconic simulation. Iconic simulation is not our concern in this book.

The Symbolic simulation models are those which the properties and characteristics of the real-system are captured in mathematical and/or symbolic form.

The Symbolic simulations include:

- Detailed information about system components
- Closely conform to the unique aspects of the industrial system
- Evaluate time-variant behaviour

- Provide system specific quantities to measure performance

Here are the types of symbolic simulations:

- Static vs. Dynamic
- Continuous vs. Discrete
- Deterministic vs. Stochastic

Simulations can take many forms from spreadsheets to three dimensional representations and projection of things moving in space. A simulation can be stochastic or deterministic - it is important that the developer understands the difference between these two types of simulation. Stochastic models consist of some probabilistic element (uncertainty) in a process. Typical outputs are boundary conditions, upper and lower limits and degree of certainty. If a model is stochastic it needs to output confidence limits, so that the end user understands that the process under scrutiny has elements of randomness and is an estimation. A determinist model can also have uncertain outcomes. Be cautious of simulation outputs that do not state their assumptions.

2.4   Modelling and types of models

A model is a representation of an actual system. Immediately, there is a concern about the limit or boundaries of the model that supposedly represent the system. The model should be complex enough to answer the questions raised, but not too complex.

There are different types of models: prescriptive models (e.g. Operational Research), descriptive models (Simulation), and statistical models.

As part of descriptive models, discrete-event model, attempts to represent the components of a system and their interactions to such an extent that objectives of the study are met. Most mathematical, statistical and input-output models represent a system's inputs and outputs explicitly represent the internals of the model with mathematical and statistical relationship. Discrete-event simulation models include a detailed representation of the actual internals.

Discrete-event models are dynamic; that is, the passage of time plays a crucial role. Most mathematical and statistical models are static, in that they represent a system at a fixed point in time. Consider the annual budget of a firm. The budget resides in a spreadsheet. Changes can be made in the budget and the spreadsheet can be recalculated, but the passage of time is not a critical issue.

The components that flow in a discrete system, such as people, equipment, orders and raw materials, are called entities. There are many types of entities and each has a set of characteristics or attributes. In simulation modelling, groupings of entities are called files, sets, lists or chains. The goal of a discrete simulation model is to portray the activities in which the entities engage and thereby learn something about the system's dynamic behaviour. The purpose of this book is for us to discuss this form of descriptive simulation i.e. the Discrete Event Simulation.

2.5   Fundamental principles of simulation

Simulation Modelling is considered as a creative activity and may conform to the following principles:

Principle 1: Conceptualisation: a model requires system knowledge, engineering judgement and model-building tools. A modeller must understand the structure and operating rules of a system and be able to extract the behaviour of the system without including the unnecessary details. The crucial questions in model building is to focus on what simplifying assumptions are reasonable to make, what components should be included in the model and what interactions occur among the components.

Principle 2: The secret to being a good modeller is the ability to remodel. Model building should be interactive and graphical because a model is not only defined and developed but is continually refined, updated, modified and extended. An up-to-date model provides the basis for future models.

Principle 3: The modelling process is evolutionary because the act of modelling reveals important information. Information obtained during the modelling process supports actions that make the model and its output measures more relevant and accurate. The modelling process continues until additional detail or information is no

longer necessary for problem resolution or a deadline is encountered. During this evolutionary process, relationships between the system under study and the model are continually defined and redefined. The resulting correspondence between the model and the system not only establishes the model as a tool for problem solving but provides system familiarity for the modellers and a training vehicle for future users.

Principle 4: The problem or problem statement is the primary controlling element in model-based problem solving. A problem or objective(s) drives the development of the model. Problem statements are defined from system needs and requirements. Data from the system provide the input to the model. The availability and form of the data help to specify the model boundaries and details.

The first step in model-based problem solving is to formulate the problem by understanding its context, identifying project goals, specifying system performance measures, setting specific modelling objectives and in general defining the system to be modelled.

Principle 5: In modelling combined systems, the continuous aspects of the problem should be considered first. The discrete aspects of the model – including events, networks, algorithms, control procedures and advance logical capabilities – should then be developed. The interfaces between discrete and continuous variable should then be approached.

Combined discrete-event and continuous modelling constitutes a significant advance in the field of simulation. There are distinct groups within the simulation field for discrete-event simulation and continuous simulation. The disciplines associated with discrete-event simulation are industrial engineering, computer science, management science, operation research and business administration. People who use continuous simulation are more typically electrical engineers, mechanical engineers, chemical engineers, agricultural engineers and physicists. A large number of problems are in reality a combination of discrete and continues phenomena and should be modelled using a combined discrete-event/continuous approach. However due to the type of problem, either a continuous or a discrete modelling approach is normally employed.

Principle 6: A model should be evaluated according to its usefulness. What inferences can be made from it? And how it will address the dilemmas of modern systems management and decision making? Simulation modelling is performed to induce change. To achieve change, the results of the modelling and simulation effort need to be put to use.

2.6   Steps to be taken for successful simulation project

The twelve steps crucial for successful design, implementation and completion of a discrete event simulation project are:

1. Problem definition: clearly defining the goals of the study. (Why are we studying this problem? and what questions do we hope to answer?).

2. Project planning: being sure that we have the sufficient resources to do the job.

3. System definition: determining the boundaries and restrictions to be used in defining the system (or process) and investigating how the system works.

4. Conceptual model formulation: developing a preliminary model either graphically (e.g. block diagram) or descriptively to define the components, descriptive variables, and interactions (logic) that constitutes the system.

5. Preliminary experimental design: what data need to be gathered from the model, in what form, and to what extent.

6. Input data preparation: identifying and collecting the data required by the model.

7. Model translation: formatting the model in an appropriate simulation language.

8. Verification and validation: confirming that the model operates the way the analyst intended (debugging) and that the output of the model is believable and represents the output of the real system.

9. Final experiment design: designing an experiment that will yield the desired information and determining how each of the test runs.

10. Experimentation: executing the simulation to generate the desired data and perform a sensitivity analysis.

11. <u>Analysis and interpretation</u>: drawing inferences from the data generated by the simulation.

12. <u>Implementation and documentation</u>: putting the results to use, recording the findings, and documenting the model and its use.

I suggest you carefully observe these steps in all your present and future simulation projects. This also includes your assignments in this module.

2.7   Simulation modelling applications

In this section we discuss the application of simulation in a wide variety of industrial sectors ranging from manufacturing, public and service industries. For example, simulation projects are regularly used for analysis of:

- manufacturing processes and material handling applications,

- public sector e.g. health care, defence performance; or explaining a natural phenomena,

- service industry e.g. transportation, logistics, computer systems performance, communication systems, retail and supply chain management.

In following sections the focus will be on the application of simulation in manufacturing, logistics and transport systems.

2.7.1 Manufacturing Application

Manufacturing and material handling systems provide a wealth of applications for simulation. Simulation has been used to solve manufacturing problems for many years. There are several reasons for this:

- Motivation for manufacturers to stay competitive,

- A high level of automation is applied to manufacturing,

- Initiatives that can be tested with minimal disruption to daily activities,

- Manufacturing systems can be well defined for modelling purposes,

- Manufacturing and material handling systems are usually too complex for other analytic techniques.

In a global economy, successful manufacturers are constantly changing the way they do business in order to stay competitive. The questions that companies deal with are:

1. When should the next piece of equipment be purchased?
2. How many people will be needed next month to meet the orders?
3. Can new orders be accepted without delaying other work?
4. How will the new plant operate five years from now?
5. How can work-in-process inventory and cycle time be reduced while increasing throughput?

Savolainen et al. (1995) indicate that simulation models are really formal descriptions of real systems that are built for two main reasons. Firstly, to understand conditions as they exist in the system; and secondly, to achieve a better system design through performing 'what-if' analysis.

Law and Kelton (1991) and Banks et al. (1997) give many benefits for simulation. Perhaps the most important benefit is that it is the most cost effective way to explore new initiatives and changes.

a. Manufacturing systems

Manufacturing is the process of making a finished product from raw material using industrial machines. Examples include automobiles, air-planes, ships, home appliances, computers, and furniture.

Firstly there are several issues that need to be addressed in managing a manufacturing process. One major issue for manufacturing is competition. Competitive pressures force manufactures to look for different ways of doing business so they can continue to produce at a reasonable cost. Manufacturing and industrial engineers are tasked with finding ways to improve operations through analysis. Secondly, to manage change and to stay competitive, manufactures are changing their operations constantly. The companies that manage change most effectively come out on top.

b. Guidelines for levels of detail in simulation modelling

Every model is an approximation of the real world. It is a given that a modeller will leave out some details when building a model of the actual system. In the simulation community, this concept is referred to as the level of abstraction. The model will be an abstraction or approximation of the actual system. The important point to note is that some details will be omitted from a model, and choosing the right details to omit determines whether a simulation will be successful or not.

Simulation modellers often discuss the accuracy of their models in terms of a percentage. The percentage is usually how close the models get to the results of the actual system. To get from 95% accurate model to a 98% accurate model may take more effort than it takes to build the original model. A good rule is that it is easier to add detail later than it is to recoup time lost by adding unnecessary detail. Figure 2.1 below shows how details are added as the model approaches an acceptable level of accuracy.
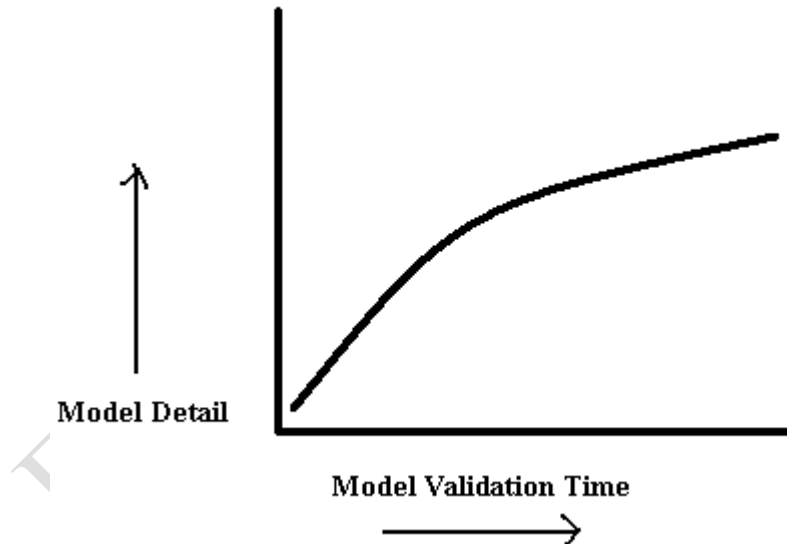
Figure 2.1: Model detail during validation [1]

The process of validation is an iterative one. The modeller adds new details to the model, runs the model, and presents the results to the project team. If the results are not sufficiently accurate, the project team identifies other details that should be

included. The modeller adds these details, and the cycle starts anew. At some point, the project team must agree that the model is "close enough" to provide useful information, and the validation process leads to experimentation.

c. Components of manufacturing system

Even though there are many types of manufacturing systems that produce a wide variety of products available today, but there are common elements that describe most manufacturing operations. These common elements should be the basis for input data used by a simulation model. Table 2.1 shows these common elements in manufacturing systems.

Table 5.1: Manufacturing components [1]

Product	Resource	Demand	Control
Parts/pieces	Equipment layout	Customer orders	Inventory control
Routings	Number of machines	Start date	Shop floor control
Process times	Downtime	Due date	Station rules
Setup times	Storage areas		
	Tools/fixtures		

To build an accurate simulation model, the data in this table should be validated and verified.

Product: Part, lots or products are entities being manufactured. Products may move in manufacturing groups called lots that are made up of a number of pieces.

Resources: Resources are used to manufacture products. Resources include machines and human beings as well as tools, fixtures, material handling systems, storage areas and so on.

Demand: The demand on a manufacturing system is defined by customer orders. Customers usually order specified quantities of products and want them delivered on a particular date.

Control: Computer-based control systems make decision about how product should be routed, collect information about current status of product or maintain proper inventory levels. These control systems interface with simulation in two ways. First they can provide input data to be used in the simulation. Second these systems often make operational decisions that should be represented in a simulation.

d. Downtime

Downtime is an aspect of manufacturing that is sometimes overlooked when building a simulation. Downtime and failure can, however, have a significant effect on the performance of manufacturing systems. Banks et al. (1996) state that there are four options for handling downtime:

1. Ignore it
2. Do not model it explicitly but adjust processing time appropriately
3. use constant values for time-to-failure and time-to-repair
4. Use statistical distributions for time-to-failure and time-to-repair

Of the four options, using statistical distribution for time-to-failure and time-to-repair is preferred. What this means to the manufacturer is that a sufficient number of downtime data has to be collected to fit a statistical distributions with desirable accuracy.

Events such as acts of nature, labour strikes and power failures can literally shut down a manufacturing operation. Because they are not part of normal operation and are very difficult to predict, catastrophic events can be ignored for most simulation activities.

e. Rework and Re-entrancy

Re-entrant process flow occurs when a particular station or work cell must be visited more than one time by the same part. Rework occurs when a part must be run through a work cell because the prior processing step was not completed successfully. Figure 2.2 shows the difference between rework and re-entrancy.
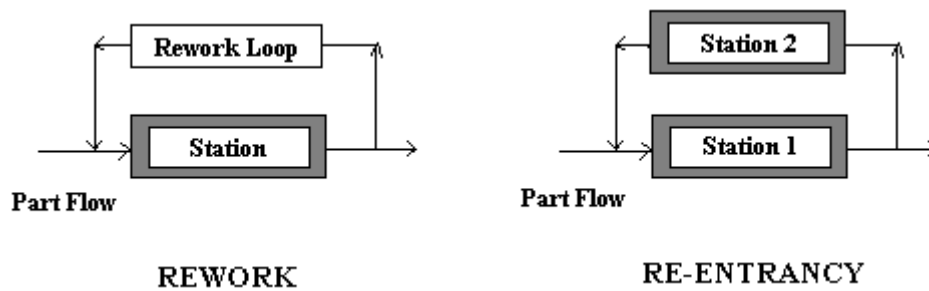
Figure 2.2: Rework and re-entrancy [5]

By using simulation it is possible to determine the effects of rework and re-entrancy on a system. Rework is typically given as a percent of the parts processed, while re-entrancy is provided in the part routing as explicit steps where the same machines must be used. In either case, the true effects on queuing and congestion can be determined using simulation.

f. Handling Stochastic (Random) Events

One of the challenges for modelling most manufacturing systems is the presence of random events. Random events in manufacturing systems can be associated with variances in:

- Processing time
- Setup time
- Downtime time to fail and time to repair)
- Yield percentage
- Transportation time
- Shipment

For all random events it is important to represent the distribution of randomness accurately in the simulation model. Choosing the right distribution is a very important part of the simulation process. When a known distribution cannot be found for a set of data, an empirical distribution can be used.

g. *Measures of Performance*

The methods used to measure model performance should be the same as those used in real system. Otherwise, it may be difficult to validate the model. With any of the performance measure, it is important to collect the average as well as the variability. Variability is usually indicated by the standard deviation, but maximum and minimum are also helpful in measuring performance. The following statistics are typically collected from manufacturing systems and should thus be provided by models of such systems:

- Production Throughput
- Production Cycle time
- Queuing behind work stations
- Transportation of material on the shopfloor
- Work in process
- Utilisation of resources (Equipment and labour)
- System specific performance measure (scrap rate, waiting time at a process)

It is important to note that optimising on one measure of performance can adversely affect another measure of performance. For example, if WIP (work in process) is reduced, equipment utilisation usually goes down. Understanding the relationships between measures of performance can help in the experimentation phase of a model.

h. *Analysis*

Using the performance measures described in the last section, model users (analysts and engineers) experiment with a model to understand the behaviour of the system under changing conditions. The issues often encountered in system analysis include:

- Determining the bottleneck
- Determining required staffing levels
- Evaluating the scheduling of tasks
- Evaluating the control system

- Recovery strategies for random events and surges

i. Business process simulations

Identifying the right area to change and improve is paramount to the overall success of an organization. The dangers of implementing business process improvement changes without a clear understanding of how the changes will impact the entire process can be substantial. Therefore tools are needed to help managers truly understand their business processes and appreciate the impact of modifications to those processes on the overall performance of the company.

The Business Modelling method is a technique to model business processes. Business models provide ways for expressing business processes or strategies in terms of business activities and collaborative behavior so we can better understand the business process and the participants in the process. Models are helpful for documenting, for comprehending complexity and for communicating complexity. By documenting business processes from various perspectives, business models can help managers to understand their environment. This allows managers to clearly see where a problems may lie, and give indications of how to improve them. Once the problem areas are identified, the software can be used to change any parameter the user wishes. Run the simulation once again and immediately see the impact of the change. In this way, companies can change their business processes in a computer environment, without risking costly setbacks of real world trial and errors.

Another factor that has contributed to the increasing usage of the business modelling method is the increasing pace of change in business. There is not enough time to try out new products in reality, and correcting mistakes, once they have occurred, is often extremely costly.

Typical uses of business modelling and simulation can be in the following areas:

- Financial Planning, quantifying the impact of business decisions on balance sheet and P&L.
- Risk Management, determining, measuring and managing the balance between profitability and certain types of risks.

- Forecasting, analysing historical data and using that to predict future scenarios and trends.
- Business Process Modeling, mapping processes, tasks and process steps in a visual representation to the resources required

Logistics, transportation, and distribution applications

In highly demanding modern economical systems there is a need for a sophisticated and widespread provision of passenger and freight movements. Due to the unprecedented need for global mobility, there is a requirement not only for various modes of transport but also increasingly sophisticated interfaces between customers, suppliers and manufacturing and service industries (Wright and Ashford, 1989).

For the past several decades, the design, analysis and control of transport systems were carried out mostly by field engineers (civil, structural and traffic engineers) and operations research (OR) scientists (Ashford and Covault, 1978; Hamzawi, 1986; Ashford 1987). A large number of Logistics and Transportation (L&T) systems have evolved over time and become fairly huge and complex. The primary goals of an L&T business enterprise are to store, distribute and/or transport freight of varying size, form and shape from its origin to its destination at the lowest cost in order to deliver the right quantities at the right time to its customers who are geographically dispersed; however he underlying logistics and transport systems that become extremely complex and often require expensive administrative, information and decision support systems (Ashford and Clark, 1975).

Major challenges face the analysts in applying simulation technologies to the L&T domain. These can be broadly listed as follows:

- L&T networks are quite complex and involve a very large number of entities and resources
- Existing simulation software do not support all the modelling/analysis features required.
- There is unfamiliarity of simulation technology in logistics and transport industry.

2.7.2 Logistics and transport problems for simulation modelling and analysis

In general, L&T problems appropriate for simulation studies are divided into three major categories:

1. New design
2. Evaluation of alternative designs
3. Refinement and redesign of existing operations

Accordingly, simulation models in L&T domains are built for the following purposes:

- Models for strategic planning
- Models for tactical planning
- Models for network/traffic control

  - Off-line control
  - Real-time satellite/telecommunication control

- Models for scheduling and dispatching

  - Off-line scheduling
  - Exception handling
  - Real-time monitoring

2.7.3 Simulation of warehouse and distribution systems

A growing number of logistics firms utilise discrete-event simulation concepts to model the various issues of large-scale logistics networks. In one extreme, a logistics simulation model may be developed to investigate and improve the operations of a warehouse; on the other extreme, it may involve modelling and analysis of the operations of an entire supply chain. In most cases there is a common goal for developing the simulation model which is to evaluate the performance of individual value-adding resources, facilities and operations as well as the flow of goods between the plants, warehouses and customers.

The simulation models are developed to perform a variety of 'what-if' scenarios to accomplish the objectives of a logistics network management or its customer. These include:

1. To evaluate strategic decisions

   - Warehouse location and allocation
   - Warehouse/distribution centre design
   - Transportation mode analysis

2. To test tactical solutions

   - Inventory management policies
   - Pull ordering between customers and plants
   - Push ordering between warehouses
   - Service levels

3. To identify operation problems on an ongoing basis

   - Change in transportation modes
   - Changes in warehouse operation parameters
   - Change in parts and finished products
   - Customer demand fluctuation

A simulation model for a logistic network is usually developed to investigate the impact of the variations associated with the production schedules, customer demand and transportation delays. The simulation model must combine the behaviour of a physical logistic network with the activities and operations of the various logistics entities within the problem domain. In general the simulation model may emphasise the internal logistics and operations of warehouse, or the pickup and delivery of freight within a city or a zone, or the movement of physical goods across an entire country or continent.

Often, logistics simulation models incorporate a geographic map showing the physical relationships among plants, terminals/hubs, warehouses and customer

locations are separately modelled at appropriate level of detail. These individual models are then integrated with the underlying logistics network superimposed on a geographic map. Often, a hierarchical modelling approach is preferred to represent the logistics network as well as the operations at the individual nodes (a node may refer to a plant, a customer or a warehouse). In this way the logistics user/designer can visualise the movement of goods at the map level as well as the operation at the plant or warehouse level.

Depending on the level of detail specified to generate the desired results, the simulation/analyst may decide to represent some or all of the entities, resources and activities in a logistics system.

In majority of cases, simulation models are developed to find the best locations for warehouses, analyse transportation modes between plants, and flow of material and customers. The input data required for these models include the following:

- Number of plants
- Number and location of warehouses
- Number of customers
- Customer demand to warehouse
- Part numbers produced at different plants
- Bill of materials
- Transportation times
- Between plants and warehouses
- Between warehouses and customers

It should be mentioned that customer demand, transportation times and so on, are stochastic in nature and vary over time. Accordingly, these data elements correspond to probability distribution generated using the information collected over long periods of time.

References

[1]   Handbook of Simulation, Edited by Jerry Banks. (ISBN 0-471-13403-1), 1998, John Wiley & Sons, Inc.

[2]   Manufacturing Systems Design and Analysis. By B. Wu (ISBN 0-412-40840-6), 1992, Chapman & Hall

[3]   Handbook of Simulation, Edited by Jerry Banks. (ISBN 0-471-13403-1) , 1998, John Wiley & Sons, Inc.

[4]   Manufacturing Systems Design and Analysis. By B. Wu (ISBN 0-412-40840-6), 1992, Chapman & Hall

[5]   Positioning of modelling approaches, methods and tools. By Savolainen T., D. Beeckmann, P. Groumpos and H. Jagdev (Computers in industry, Vol. 25, pp. 255-262 , 1995)

[6]   Set management: minimising synchronisation delays of prefabricated parts before assembly. Mittler, M., M. Prum, and O. Gihr (Association for computing machinery, pp. 829-836 , 1995)

[7]   Transportation Engineering: Planning and design. Wright P. H. , Ashford N. J. Wiley, (2002)

[8]   Level of service design concept for airport passengers,. Ashford N.J. (Transportation planning and technology, Vol. 12 , 1987)

[9]   The mathematical form of travel time factors. Ashford N. J. , Covault D. O. (Highway Research Record 283, Washington D.C. , 1978)

[10] Management and planning of airport gate capacity. Hamzawi S. G. (Transportation planning and technology, vol. 11. No. 3., 1986)

[11] Business Process Modeling, Simulation, and Design. By Manuel Laguna, Johan Marklund, Laguna. (ISBN 0-131-09979-5), 2004 , Prentice Hall.

Chapter 3

A Brief Introduction to Probability and Statistical Inference

This Chapter Covers:

1. Basic probability and statistical inference
2. Random variables and Distribution Functions
3. Some of the most relevant distribution functions
4. Introduction to Markov Chains
5. Basic Queuing Model

Note: In this course I am expecting that you have some basic background in statistics and probability. If you do not, for the purpose of maximising your learning experience I suggest you study a book or two on these two subjects.

To maximise your learning experience for this chapter, I recommend the following readings:

1. G. L. Curry and R. M. Feldman (2011); Manufacturing Systems Modeling and Analysis; Second Edition, Springer

2. R. G. Askin and C. R. Standridge (1993); Modelling and Analysis of Manufacturing Systems; John Wiley & Sons, Inc.

3. D. A. Santos (2011); Probability – An Introduction; Jones and Bartlett Publishers, *Sudbury, Massachusetts.* [The book starts from origins probability].

4. R. V. Hogg and E. A. Tanis (2010), Probability and Statistical Inference, Eighth Edition; Pearson.

3.1　Our World of Deterministic and Probabilistic Events

Before we start discussing probability and its application in systems analysis, we need to differentiate between *Deterministic* and *Probabilistic* events.

As the word deterministic implies, events that are determined are the type of events that will occur with 100% probability! Oops – I thought we were not talking about probability yet! May be I better rephrase the title of this section or may be the title of this chapter should be: *"Our world of Probabilistic events"*.

In the real world what ever occurs as an **event** is a collection of **outcomes** of things that happen. In other words a set of outcomes can cause an event (go back to Chapter 1 where we talk about input data). The type of events that happen with certainty are called deterministic events. For example after night comes day and after day comes night. However, in our real world of complex socio-economical development, things happen or may not happen with a degree of probability. People, machines and systems behaviour in majority of cases is **random**. For example, machines may suddenly breakdown during a working shift; or the number of customers arriving at a fast food restaurant at lunch time is different every day of the week. A professional football player may score or miss a penalty….

As systems analysts, we need to be able to understand and capture this random behaviour. By capturing randomness in nature and socio-economical events, I mean interpret them into mathematical constructs so that we can build models. Use those models to describe the behaviour and also to predict the behaviour of the systems.

3.2　Probability and Statistical inference

In this section we will briefly touch on the basics and the surface of probability theory and you are advised to enrich your knowledge by further reading the references I have mentioned earlier. I hope this short introduction would encourage you to read more about probability theory.

3.2.1 Probability

a. Basic Concept of Probability

Probability is a real-valued set function denoted by *P(E),* that assigns a probability value between 0 and 1 for event *E* in the sample space *S*. *P(E)* is therefore called the probability of event *E*. Provided that the following properties is satisfied:

$P(S) = 1$

$P(E) \geq 0$

$if\ E_1, E_2, E_3, ...E_n$ are events where $i \neq j$, and $E_i \cap E_j = \phi$ then

$P(E_1 \cup E_2 \cup ...E_n) = P(E_1) + P(E_2) + ...P(E_n)$

Here we will point to one of the fundamental theorems (theorem 1 and 2) of probability and leave the other 4 for you to research. A good starting point would be the book by (R. V. Hogg and E. A. Tanis (2010).

**Theorem 1:** for a given event *E*, $P(A) = 1 - P(A')$.

This theorem implies that the probability of an event occurring in a sample space is equal to 1 minus the probability of that event not occurring. Tossing a fair coin where the sample space is S = {H, T}, *P(S) = 1* so *P(H) = 1-P(T) = 1/2*.

From this theorem we can also deduce that $P(\phi) = 0$, because $P(\phi) = 1 - P(S)$.

Theorem 2: if $E \subset F$, then $P(E) \leq P(F)$.

b. The Conditional Probability

As the name implies conditional probability is about the probability of event *E* occurring provided that event *F* has occurred. It can therefore be expressed as:
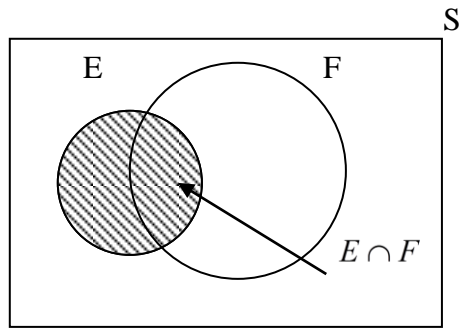
Figure 3.1: Venn diagram showing conditional probability

$$P(E|F) = \frac{P(E \cap F)}{P(F)} \text{ or } P(E \cap F) = P(E|F).P(F) \text{ or } P(E \cap F) = P(F|E).P(E)$$

$$P(F) \neq 0$$

Example: A manufacturer in China produces two brands of Volleyballs (indoor (*i*) and beach (*b*)) and sells each type in packs of 6. A random quality control exercise requires an operator to open a pack and test the balls for any defect. The operator will then report the number of defects and the type of the ball.

The sample space (*type, no. of defects)* in this example will be:

$$S = \{(i,0),(i,1),(i,2),(i,3),(i,4),(i,5),(i,6),(b,0),(b,1),(b,2),(b,3),(b,4),(b,5),(b,6)\}$$

Each incident (detected Volleyball type and number of defects) could be associated with a probability of occurrence.

$P(i,0) = 0.38$	$P(b,0) = 0.35$
$P(i,1) = 0.1$	$P(b,1) = 0.06$
$P(i,2) = 0.05$	$P(b,2) = 0.02$
$P(i,3) = 0.01$	$P(b,3) = 0.01$
$P(i,4) = P(i,5) = P(i,6) = 0.005$	$P(b,4) = P(b,5) = P(b,6) = 0.005$

The probability that beach volleyball pack is selected and at most 2 of the balls to be defective is:

$$P\{(b,0),(b,1),(b,2)\} = 0.43$$

Imagine a pack has been selected and we find out that it contains indoor volleyballs. We have not yet checked of there are any defects in this pack. In order to find out whether at most 1 ball is defective, we need to describe two events. Event *E* shows the incidences of at most 1 defective ball i.e. *{(i,0),(i,1),(b,0),(b,1)}* and event *F* shows selection of an indoor pack i.e. *{(i,0),(i,1),...(i,6)}*. Therefore the conditional probability for finding at most 1 defective volleyball provided the inspected pack is an indoor volleyball pack can be expressed as:

$$P(E|F) = \frac{P(E \cap F)}{P(F)} = \frac{P\{(i,0),(i,1)\}}{P\{(i,0),(i,1),(i,2),(i,3),(i,4),(i,5),(i,6)\}} = \frac{0.48}{0.555} = 0.86$$

3.2.2 Random Events and Statistical inference

A statistician's job is to collect data and analyse that data for the benefit of understanding a trend and predicting behaviour. In the real world even though the conditions for measurements may remain the same, but results could vary. The science of statistics is to determine the pattern despite variability. Statistics is all about recognising the pattern of a random behaviour, minimise the errors in their interpretation of the data with respect to the noise (anomalies in the data). The variability in data is the recipe for uncertainty.

The best thing to do now is to introduce a few concepts that you need to get familiar with and know to be able to become a discrete event simulation *expert*.

Random Experiments: conducting a number of experiments in a specified length of time where the outcome is not certain. Recording the time between arrivals of people at an airline check-in counter between 6:00 to 21:00 is a good example.

Sample Space: is a collection of outcomes of random events that took place in a specified time. For example, the recorded passenger processing times in minutes during the working hours at the same check-in counter (e.g. 6:03, 6:08, 6:15, 6:34… 20:39, 20:52).

Random variables: values of outcomes observed during an experiment denoted by X (e.g. $X_1 = 6:03$, $X_2 = 6:08$, …$X_n = 20:52$).

Random Variable Distribution: could be described as a series of random values that would repeat itself in time.

Have you ever wondered why goal keepers dive in the wrong direction when trying to save penalties[4]? Who says there is no Maths in sports! The reason is very simple; the professional goal keeper conducts a *statistical inference*. Before the match, a good/professional goal keeper would watch 100 penalties that the opposition star has taken. He or she realises that the opposition penalty taker has directed 80 (repeated occurrence) of the 100 penalties to the goal keepers' right hand side and only 20 (repeated occurrence) to the left. If you were the goal keeper how would you dive (left or right) if this penalty taker steps behind the ball? I bet you are wondering which players would be better penalty takers – the ones who make this prediction difficult! But How?
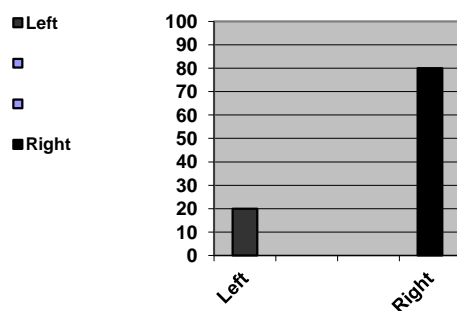
Figure 3.2 a: Random variable distribution for the penalty shooter
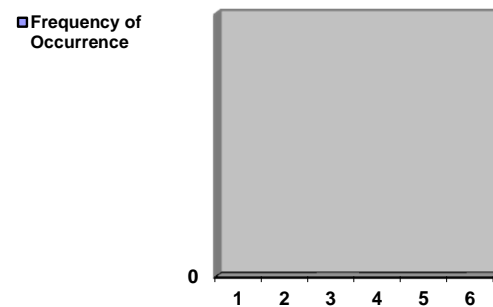
Figure 3.3 b: Draw the random variable distribution expected from a fair Die

Probability Mass Function: discrete random variables can assume positive countable (integer) random values. For example the number of people *X* that call a call centre between 9:00-10:00. A set of probabilities that is associated with a random

---

[4] I should apologise if you are not interested in football and its rules. But I feel compelled here to explain the penalty rule: A penalty is taken on referee's instruction (by blowing into his whistle). The goalkeeper can only dive when the player touches the ball with his foot; therefore, he/she has little time guess the direction of the penalty kick. A keeper therefore, normally makes an instant decision on the direction of his/her dive. So it is normally random and based on the keeper's experience and preferences. *Imagine if I had to explain the off-side rule here!*

variable *X* can create its probability mass function. Thus if the possible values of a random variable *X* is given by the non-negative integers, then the probability mass function for every *k* in the range of *X* is given by the probabilities of:

$$f_k = P(X = k) \text{ for } k = 0,1,2,...$$
$$\text{such that } P(X = k) \geq 0 \text{ and } \sum_k P(X = k) = 1$$

Continuous random values can assume any real value, so instead of being countable they can be in a continuous range, for example the length of time *T* that an operator answers a telephone call in a call centre.

The *cumulative distribution function* gives the accumulated probability up to and including to the point that it has been calculated. It can be expressed as:

$$F(a) = P(X \leq a) = \sum_{k \leq a} f(k) \text{ for all real number } a$$

The *probability density function* is thus the derivative of the cumulative distribution function and is used to express the probability of an interval (values between two numbers) occurring. An example of interval is the time between calls that occur 9:00-10:00 in a call centre.

$$f_X(a) = \frac{dF_X(a)}{da}$$

In order to better explain the randomness of random variable and the behaviour of a distribution function we use measures such as *mean, variance* and *standard deviation.*

Mean: is the arithmetic average of a large number of random observations.

$$\mu = E(X) = \sum_{x \in S} xf(x) = u_1 f(u_i) + u_2 f(u_2) + ... + u_k f(u_k)$$

where $u_i f(u_i)$ is the product of distant (u_i) to its weight (moment).

**Variance:** represents the variability of random observation.

$$\sigma^2 = E(X^2) - \mu^2$$

Standard Variation: is the square root of the variance $\sigma = \sqrt{\sigma^2}$.

3.3   Some of the Important Distribution Functions

In discrete event simulation and modelling we use distribution functions to match the input with the known functions. Using *goodness-of-fit* techniques we will then try the find the most fitting function that best matches the collected data. By conducting statistical test (e.g. Kolgomorov-Smironov or Chi square tests) we find the most appropriate distribution. We will then use that distribution to generate random numbers/variables for prediction purposes. In this section we will briefly discuss some of the most common probability density functions that you will be encountering and using for your discrete event simulation projects. Note that the reason for mentioning these commonly used distributions here is for their application in standard discrete event simulation software packages (for example Arena[TM]). If you would like to research further in their mathematical construct and origins, I advise you to study the reference books mentioned at the beginning of the chapter.

Uniform-Discrete: Imagine throwing a fair die several times (Figure 3.2b) and counting the number of times each number comes. In long term you will observe that the number of times each side shows as about 1/6 of the total throws. How would a lottery numbers uniform discrete event probability mass function would look like?
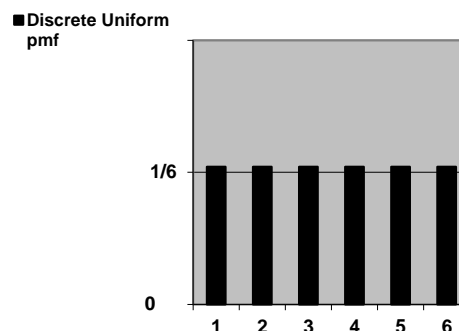
Figure 3.4: Uniform Discrete Event probability mass function (pmf)

The probability mass function (pmf) of random variable N given two integers a and b can be expressed as:

$$P(X = k) = f(k) = \frac{1}{b - a + 1} \text{ for } k = a, a+1, \ldots, b$$

$$E(N) = \frac{a + b}{2}$$

$$V(N) = \frac{(b - a + 1)^2 - 1}{12}$$

In simulation software packages you can also have non-uniform discrete event probability function and you can use it for example in defining the percentage of different job types enter a system, batch sizes, disassembly of an artefact and other things. By the way how would a non-uniform discrete event distribution probability mass function or cumulative distribution function would look like?

Bernouli: concerns experiments with two possible outcomes, for example tossing a coin ($k=0$ for Tail and $k=1$ for Head). The random variable N would have a Bernouli distribution provided:

$$P(X = k) = f(k) = \begin{cases} p & \text{for } x = 1 \\ 1 - p & \text{for } x = 0 \end{cases}$$

$$E(N) = p$$
$$V(N) = p(1 - p)$$
$$P(X = 1) = p$$
$$P(X = 0) = 1 - p$$
$$0 < p < 1$$

Binomial: is when you carry out an experiment that has two possible outcomes for n number of times.

$$P(X = k) = f(k) = \frac{n!}{k!(n - k)!} p^k (1 - p)^{n-k} \text{ for } k = 0, 1, \ldots, n$$

$$E(N) = np$$
$$V(N) = np(1 - p)$$

X is the number of times that outcome k has occurred.

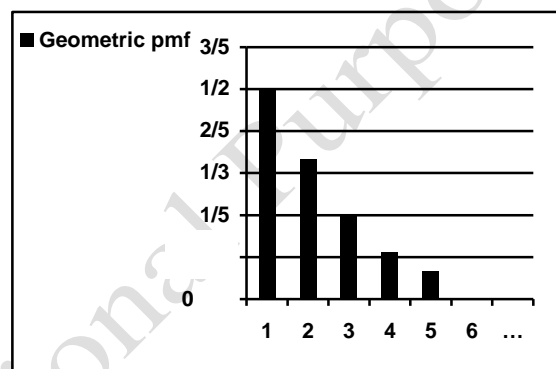For example, what is the likelihood of having 6 heads when tossing a fair coin 10 times?

$$P(6) = \frac{10!}{6!(4!)}(0.5)^6(0.5)^4 = 0.20$$

Geometric: is when you wish to calculate the probability mass function of number of trials X that needs to take place for an outcome to occur (success). This is again for events with two possible outcomes (occurring or not occurring).

$$P(X = k) = f(k) = p(1-p)^{k-1} \quad \text{for } k = 1,2,...$$
$$E(N) = 1/p$$
$$V(N) = \frac{1-p}{p^2}$$

Figure 3.5: Geometric probability mass function

Poisson: deals with the random number events that occur in a given time. For example; the average number of people that may call a call centre is $\lambda = 9$ people. The random variable N therefore follows a Poisson distribution if there is a $\lambda > 0$ so that the probability mass function can be expressed as:

$$P(X = k) = f(k) = \frac{\lambda^k e^{-\lambda}}{k!} \quad \text{for } k = 0,1,2...$$
$$E(N) = V(N) = \lambda$$

For example, if the number of calls to a call centre follows a Poisson distribution with mean value of $\lambda = 9$ per hour. The likelihood of 6 people calling between 12:00-13:00 would be 9.1%.
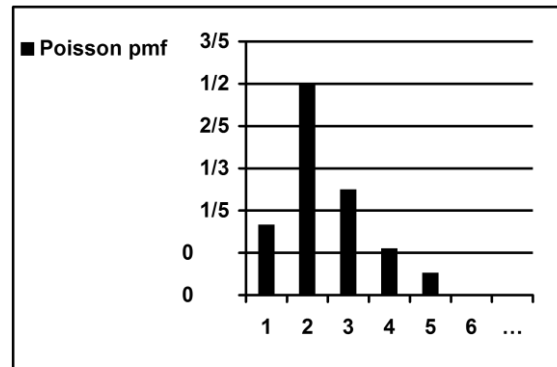
Figure 3.6: Poisson probability distribution function for a mean value of λ

Exponential: One of the most relevant models in continuous probability modelling, the Exponential distribution has no memory. That means probability of an incident occurring is independent from the previous incident. The random variable *X* follows an exponential distribution if its probability density function can de expressed as:

$$f(x) = \begin{cases} \dfrac{1}{\beta} e^{-x/\beta} & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

where the parameter of distribution $\beta = 1/\lambda$

$P(X > x) = e^{-x/\beta}$

$E(X) = \beta = 1/\lambda$

$V(X) = \beta^2 = 1/\lambda^2$

Random arrival times of customers and the breakdown of electronic equipment are some of the common application of this distribution and continuous random variable generations.

For example, in a busy airport, aircrafts arrive based on a Poisson process with mean rate of 10 per hour on a single runway. What is the probability of the runway waiting more than 8 minutes for the first aircraft to arrive?

$$\beta = 60 / 10 = 6$$

$$f(x) = 1/6 e^{-(x/6)}$$

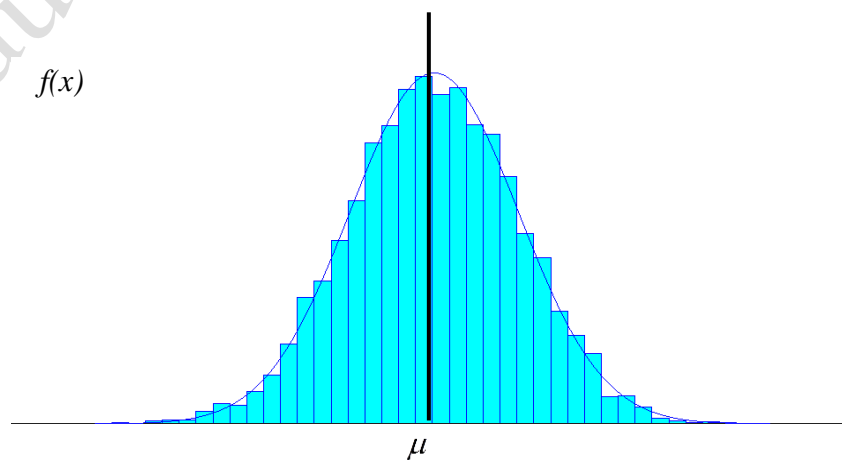$$P(X > 8) = e^{-(8/6)} = 0.26$$

f(x)

$1/\beta$
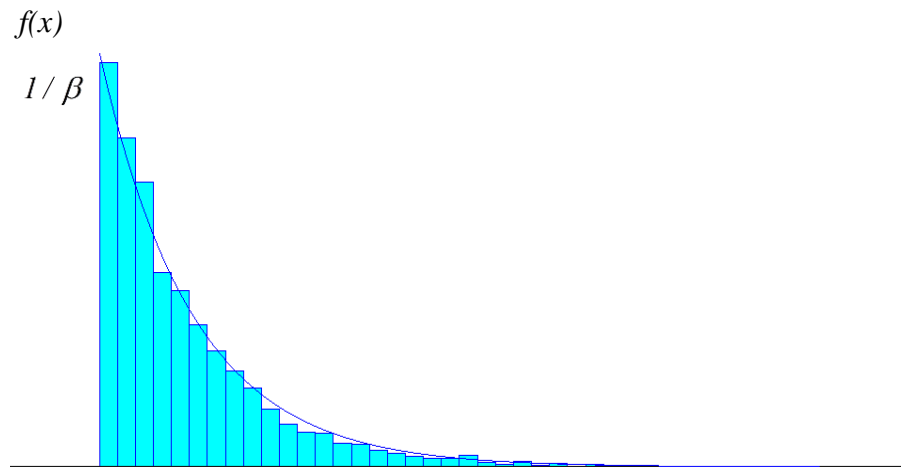
Figure 3.7: Probability density function for exponential distribution with mean β

Normal: The random variable *X* follows a Normal distribution with μ as its mean value and σ standard deviation when the probability density function can be expressed as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 /(2\sigma^2)}$$

$$E(X) = \mu$$

$$V(X) = \sigma$$

f(x)

μ

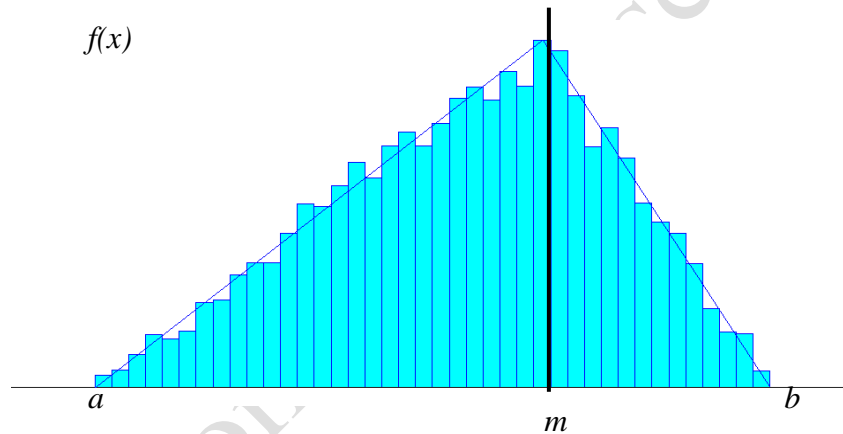**Figure 3.8:** Probability density function for Normal distribution with mean of μ

Triangular: A random variable follows a Triangular distribution if it has a minimum *a*, maximum *b* and most likely occurrence (mode) of *m*. The probability density function is then expressed as:

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(m-a)(b-a)} & \text{for } a \leq x \leq m \\[2mm] \dfrac{2(b-x)}{(b-m)(b-a)} & \text{for } m \leq x \leq b \\[2mm] 0 & \text{otherwise} \end{cases}$$

$$x \in [a,b]$$
$$E(X) = (a+m+b)/3$$
$$V(X) = (a^2 + m^2 + b^2 - ma - ab - mb)/18$$

Figure 3.9: Probability density function for Triangular distribution

3.4　Markov Process

The purpose of introducing Markov Chains at this stage is for you to appreciate one of the most important subjects in Discrete Event Simulation projects and model, *the Queuing principles.*

Markov processes are powerful tools for describing and analysing dynamic systems that are probability based. Markov processes constitute the fundamental theory underlying the concept of queuing systems. Each queuing system can be mapped onto an instance of a Markov process and then mathematically evaluated in terms of this process.

Markov processes are a special type of stochastic processes. Earlier we discussed what a stochastic process is (i.e. random behaviour within a state space, within a time spec). A stochastic processes $X_t : t \in T$ and $T \subseteq N_+ = [0, \infty]$ is therefore a Markov process if for all $t_0 = 0, t_0 < t_1 < t_2 < ... < t_n < t_{n+1}$ and all $s \in S$ the cumulative density function of X_{t_n+1} is dependent on the last value X_{t_n} and not on earlier values of $X_{t_0}, ... X_{t_{n-1}}$. A Markov process is therefore a conditional probability:

$$P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, ..., X_{t_0} = s_0) = P(X_{t_{n+1}}, s_{n+1} | X_{t_n}, s_n)$$

Thus a stochastic process with Markov property is a Markov Process. A good example of this is the difference between a Dice game in which the result of the next throw is absolutely independent from previous throw (stochastic process); whilst playing the second card in a card game like Trumps, is normally dependent on the previously played card.

3.4.1 Markov Chains

Markov processes can be homogenous (time independent) or non-homogenous (time dependent). Parameters of Markov processes can be discrete or continuous. Both Discrete-parameter (Set T to be discrete) or Continuous-parameter (Set T to be continuous) Markov processes may have discrete or continuous state spaces. Markov processes with discrete state spaces are usually called **Markov Chains**.

a. Discrete Time Markov Chains

A given stochastic process of $(X_0, X_1, X_2, ..., X_{n+1}, ...]$ at consecutive points of observation $0, 1, 2, ..., n+1$ constitute a Discrete Time Markov Chain provided that the conditional probability mass function is defined as:

$$P(X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0) = P(X_{n+1} = s_{n+1} | X_n = s_n)$$

Where $P(X_{n=1} = s_{n+1} | X_n = s_n)$ is the conditional probability mass function of transition from state s_n to s_{n+1} at time step $n+1$.

$$p_{ij}^{(1)}(n) = P(X_{n+1} = s_{n+1} = j | X_n = s_n = i)$$

When one continues the Markov Chain, its evolution from state s_0 to s_n is step by step and according to a transition probability. There are many applications for Markov chains such as genetic programming and many other dynamic and evolutionary processes in which the probability of state transition is known. The one-step transition probabilities are usually summarized in a non-negative, stochastic transition matrix P:

$$P = P^1 = [\ p_{ij}\ ] = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \cdots \\ p_{10} & p_{11} & \cdots & p_{1,n} \cdots \\ p_{20} & \cdots \cdots \\ \cdots \end{bmatrix}$$

The elements of each row sums up to 1.

A Discrete Time Markov Chain state transition can be expressed in the diagram below:
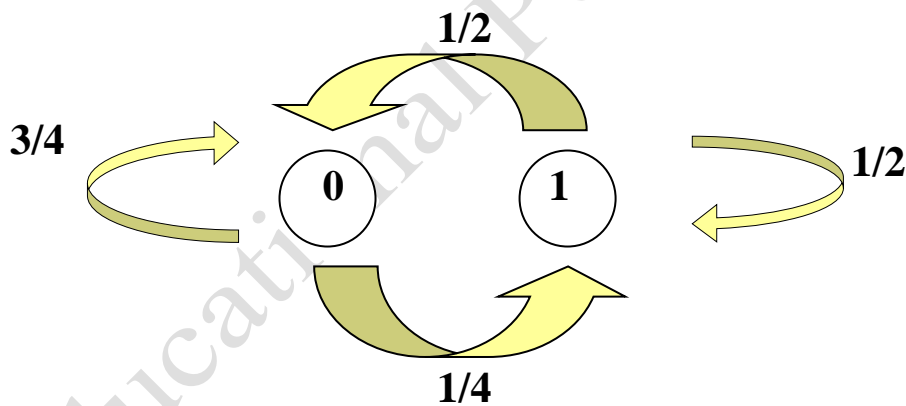
The one-step transition matrix can therefore be written as:

$$P^1 = \begin{bmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{bmatrix}$$

To conclude this section lets do a simple example.

The probabilities of weather conditions, given the weather on the preceding day, can be represented by the state transition matrix:

$$P^1 = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}$$

State 0 = sunny and State 1 = Rainy.

Reading the matrix, the probability of a day being sunny and the following to be sunny is 0.9. The probability of sunny to rainy will be the remaining 0.1. Can you decipher the second row?

If on day 0 the weather is sunny, then $X^0 = (1 \quad 0)$ meaning the day is sunny then 100% and rainy 0%.

To predict weather in day 1:

$$X^{(1)} = X^{(0)}.P \Rightarrow (1 \quad 0)\begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = (1 \times 0.9 + 0 \times 0.5 \quad 1 \times 0.1 + 0 \times 0.5) = (0.9 \quad 0.1)$$

Day 2

$$X^{(2)} = X^{(1)}P = X^{(0)}P^2 = (0.9 \quad 0.1)\begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = (.9 \times .9 + .1 \times .5 \quad .9 \times .1 + .1 \times .5) = (.86 \quad .14)$$

The general rule for *n* days will be as:

$$X^{(n)} = X^{(n+1)}P$$
$$X^{(n)} = X^{(0)}P^n$$

In long run this terrain when number of *n* goes to infinity the steady state will be $(p_0 \quad p_1) = (0.833 \quad 0.167)$ in other words if you want to bet on a day to be sunny in the future you better put your money on a sunny day!

b. *Continuous Markov Chains*

A given stochastic process $X_t : t \in T$ constitutes a Continuous Markov Chain for all arbitrary $t_i \in N^+$, $0 = t_0 < t_1 < t_2 < ... < t_n < t_{n+1}$ and taking its values from state space $\forall s_i \in S$ the conditional probability mass function will be:

$$P(\,X_{t_n+1} = s_{n+1} \,\big|\, X_{t_n} = s_n, X_{t_n-1} = s_{n-1},..., X_{t_0} = s_0\,) = P(\,X_{t_n+1} = s_{n+1} \,\big|\, X_{t_n} = s_n\,)$$

Have another look at the Discrete Markov Chain conditional probability mass function and compare it to the Continuous one. The difference is the time factor.

3.4.2 Markovian Queues

All of us have experienced queues, especially the ones who are not privileged or considered as an immediate priority by the service provider. Queues represent waiting to be served either by a person or a machine. Queues form because there is difference between arrival rates and processing time. If the world around us was a deterministic one, then service providers would have been able to design their production or service processes in a way that no queues will form and no waiting time incurred. For example if the arrival time between two jobs 2 minutes and the processing time 1.5 minutes there would be no queues. But if the time between arrivals was random between 1 minute to 5 minutes and the processing time fixed, at times you would observe queues forming. We will try (or have tried this already in the lab) using the simulation software having a single resource (machine or a person), one queue and a random inter-arrival.

Therefore to measure the numbers of jobs in a queue or the waiting times, there are three key data that need to be known denoted by (*A/B/m*). The *A* indicates the distribution of inter-arrivals (e.g. Poisson for number or Exponential for time between arrivals). The *B* indicates in distribution of the processing time and the *m* is the number of servers (in Arena software package defined as resources). The Markovian queues are then described as *M/M/1* for random arrival rates, random processing times with a single server queues. *M/M/c* denotes random arrival rates, random processing time with *c* servers.

If λ is the average arrival rate and μ is the average processing time for *c* server then the utilisation factor can be estimated as: $\rho = \dfrac{\lambda}{\mu c}$.

Assuming that the probability of *n* parts being at the workstation at time *t* to be $p_t(n)$, for a steady state situation will then be $p_t(\,n\,) = p_{t+\delta t}(\,n\,)$. Thus, table 3.1 shows the queuing results for an *M/M/1* situation.

Table 3.1: *M/M/1* queuing results, for a more complete table on *M/M/c,* I suggest you see: R. G. Askin and C. R. Standridge (1993); Modelling and Analysis of Manufacturing Systems, Wiley & Sons.

	Notation	*M/M/1*
Probability of 0 jobs at the workstation	$p(0)$	$1 - p$
Expected no. of Jobs waiting in Queue	L_q	$\dfrac{\rho^2}{1-\rho}$
Expected no. of jobs at workstation	L	$\dfrac{\rho}{1-\rho}$
Expected Queuing Time	W_q	$\dfrac{\rho}{\mu(1-\rho)}$
Expected Throughput time	W	$\dfrac{1}{\mu(1-\rho)}$

Note that queues also have rules, First-Come-First-Serve (FCFS) or priority rules such as Last Come First Serve (LCFS), Lower Value First (LVF), Higher Value First (HVF) and so on and so forth. We will examine these rules when using a simulation software package and assigning priorities to entities that are served by the servers.

To end this chapter we bring a simple example of an *M/M/1* system.

Example: A security and metal detection machine at an airport has a service rate that follows an Exponential distribution with $\mu = 10$ passengers per minute. Passengers arrive at the machine with an Exponential rate of $\lambda = 8$ per minute. The queuing rule is FCFS. Find the expected machine utilisation, passenger throughput time and average waiting time.

Solution: We treat this as a single station process therefore:

$$\text{Machine Utilisation}: \quad \rho = \frac{\lambda}{c\mu} = \frac{8}{10}$$

$$\text{Probability that the machine would be idle}: \quad p(0) = 1 - 0.8 = 0.2$$

$$\text{Throughput time}: \quad W = \frac{1}{\mu(1-\rho)} = \frac{1}{10(1-0.8)} = 0.5 \, \text{min}$$

$$\text{Waiting time}: \quad W_q = \frac{0.8}{2} = 0.4 \, \text{min}$$

CHAPTER 4

The Simulation Modelling Environment

This Chapter Covers:

1. The steps to a successful simulation project

2. Simulation input and output data analysis

3. The main techniques of experimental design

4. The approaches to model verification, validation and testing

4.1 Steps for simulation study

Simulation, particularly discrete event simulation (DES) is used as a problem solving technique. Discrete event simulation literature has been substantially developed since inception of digital computers. In the last fifty years of its history, modelling has been developed from theoretical process into decision support tool. There are two reasons for the success of discrete event simulation in day to day problem solving. Firstly, advances in computing technology; and secondly, restricted budgets and the affordability of computer hardware and processor time. These factors make simulation projects very cost-effective to commission and complete [1].

As a result, simulation provides huge support for a wide variety of purposes including, training, interaction, visualisation, hardware testing, and decision support in real-time.

According to Shannon, digital computer simulation is the process of designing a model of a system and conducting experiments with this model on a digital computer for a specific purpose of experimentation. Digital computer simulation can be divided into three categories; (1) **Monte Carlo**, (2) **Continuous**, and (3) **Discrete event**. Monte Carlo simulation is a method by which an inherently non-probabilistic problem is solved by a stochastic process; the explicit representation of time is not required. In a continuous simulation, the variables within the simulation are continuous functions, e.g. a system of differential equations. If value changes to program variables occur at precise points in simulation time (i.e. the variables are "piecewise linear"), the simulation is discrete event. Also, Nance (1987) states that three related forms of simulation are commonly used in the literature. A **combined** simulation refers generally to a simulation that has both discrete event and continuous components. **Hybrid** simulation refers to the use of an analytical sub-model within a discrete event model. Finally, **gaming** can have discrete event, continuous, and/or Monte Carlo modelling components. The focus of this chapter is limited to discrete event simulation. [1]

Investigation in modelling methodology has persisted some 35 years, beginning with the General Simulation Program of Tocher in 1958 [2] and continuing in the

writings of Lackner [3], Kiviat [4], Nance [1] and Zeigler [5] to cite the most prominent.

A simulation involves modelling a system. A **system** is a part of the world which we choose to consider as containing a collection of components each characterised by a selected set of data items and patterns, and by actions which may involve itself a component and other components. The system may be real or imaginary and may receive input from, and/or produce output for, its environment. Simulation modelling is a fundamental discipline that affects wide variety of scientific fields from engineering to social sciences.

According to Nance (1987), a model is comprised of **objects** and the relationships among objects. An object is anything characterised by one or more **attributes** to which **values** are assigned. The values assigned to attributes may conform to an attribute typing similar to that of conventional high level programming languages.

Within a discrete event simulation, there are two concepts of *time* and *state* that are of paramount importance. Nance (1987) identifies the following primitives which permit precise delineation of the relationship between these fundamental concepts (see Figure 4.1):

- An **instant** is a value of system time at which the value of at least one attribute of an object can be altered.
- An **interval** is the duration between two successive instants.
- A **span** is the contiguous succession of one or more intervals.
- The **state of an object** is the enumeration of all attribute values of that object at a particular instant.
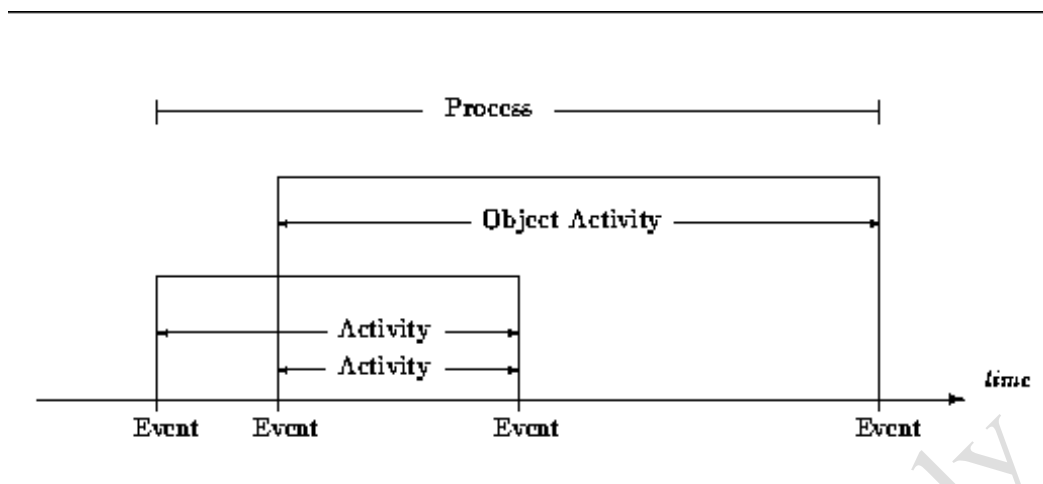
Figure 4.1: Event, Activity and Process

These definitions provide the basis for some widely used (and, historically, just as widely misused) simulation concepts:

- An **activity** is the state of an object over an interval.
- An **event** is a change in an object state, occurring at an instant, and initiates an activity precluded prior to that instant. An event is said to be **determined** if the only condition on event occurrence can be expressed strictly as a function of time. Otherwise, the event is **contingent.**
- An **object activity** is the state of an object between two events describing successive state changes for that object.
- A **process** is the succession of states of an object over a span (or the contiguous succession of one or more activities).

In other words, modelling is the process of describing a system (producing a model of that system) with the goal of experimenting with that model to gain some insight into the behaviour of the system. The model itself is a collection of interacting objects, these objects being described by attributes.

4.1.1 A model classification scheme

In modelling, understanding the concept of model is very important because of different forms that models may take. Therefore, describing the characteristic properties is difficult to formulate. Figure 4.2, shows a model classification that

provides four main dimensions. The classification scheme for models adopted from Balci (1987) [6].
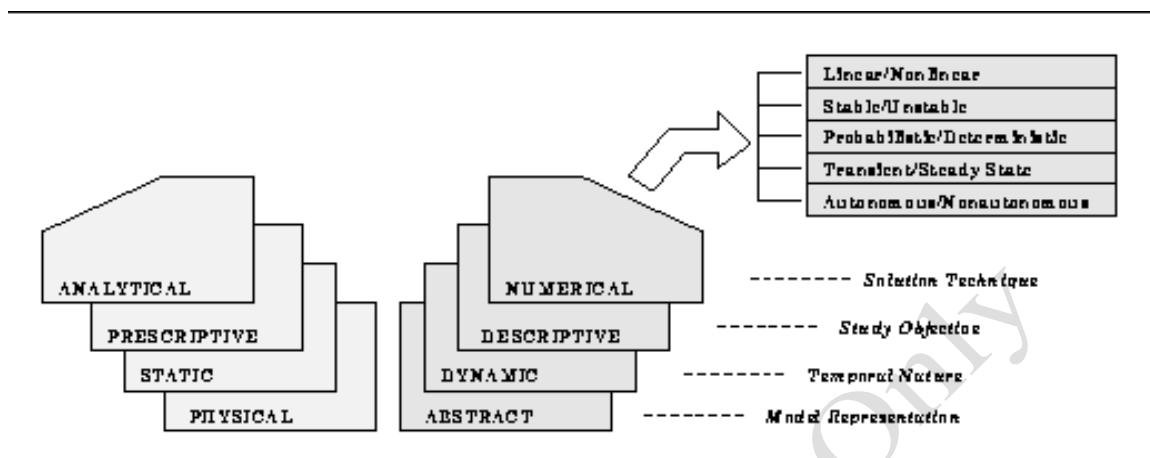
Figure 4.2: A Classification Scheme for Discrete Event Simulation (DES) Models [6]

The first dimension characterises the model representation. An abstract model is one in which symbols constitute the model. A verbal or written description in English is an abstract model. A mathematical model is described in the symbology of mathematics and is a form of abstract model. A simulation model is built in terms of logic and mathematical equations and is considered an abstract model. The second dimension characterises the study objective underlying the model. A descriptive model describes the behaviour of a system without any value judgement on the quality of such behaviour.

A third dimension relates to the presence of temporal properties in the model. A *static* model is one which describes relationships that do not change with respect to time. Static models may be abstract or physical. An architectural model of a house is a static physical model. An equation relating the area and volume of a polygon is a static mathematical model. A dynamic model is one which describes time-varying relationships. A wind tunnel which shows the aerodynamic characteristics of proposed aircraft design is a dynamic physical model. The equations of motion of the planets around the sun constitute a dynamic mathematical model.

The fourth dimension identifies a solution technique. An *analytical* model is one which provides closed-form solutions using formal reasoning techniques, e.g.

mathematical deduction. A numerical model is one which may be solved by applying computational procedures.

Discrete event simulation models are considered in the class of abstract, dynamic, descriptive, and numerical models.

Discrete event simulation models may be defined with various combinations of the following characteristics: (1) a linear model is one which describes relationships in linear from, and a nonlinear model describes nonlinear relationships; (2) a stable model is one which tends to return to its initial condition after being disturbed, while an unstable model is one which may not return to its initial condition after being disturbed; (3) a steady-state model is one whose behaviour in one time period is of the same nature as any other time period, while a transient model is one whose behaviour changes with respect to time; (4) a probabilistic (stochastic) model is a model in which at least one state change is a function of one or more random variables, otherwise, the model is deterministic, and (5) an autonomous model is one in which no input is required (or permitted) from the environment, other than at model initiation, while a model that permits input to be received from its environment at times other than model initiation is a non autonomous model.

4.2   Input Data Analysis

One of the primary reasons for using simulation is that the model of the real-world system is too complicated to study using the stochastic processes models. Examples of such random inputs include arrivals of orders to a job shop, times between arrivals to a service facility, times between machine breakdowns, and so on. The major sources of complexity are the interrelationship between different elements within the system and how they process the input to achieve the desired output, so-called the prevailing *logic*. Each simulation model input has a correspondent both in the real-world system and in the simulation program, as shown in figure 4.3 [7].

| Real world process or phenomenon | | Simulation Models *Random Input* | | Simulation Programme *Random Variate Generator* |

Simulation input modelling

Random Variate Programming
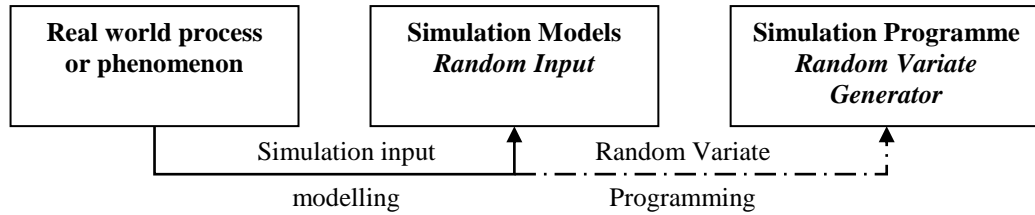
Figure 4.3: Role of input distributions [Adopted from J Banks].

A random input variable to a simulation model can be viewed as a stochastic process. A stochastic process is often defined as a collection of random variables $\{X(t), t$ in $T\}$, where T is called the index set of the process and t usually represents time. In the discrete-event simulation context the index set is typically taken to be the non-negative integrates, so the stochastic process itself is referred to as being discrete-time. For such a process we can use the simpler notation $\{X_k, k = 1, 2...\}$. Notice that subscript K dictates the order of the variables but not the specific time of occurrence (i.e. X_2 occurs some time after X_1 and some time before X_3, not necessarily at equal time intervals. Here each X_k is a distinct occurrence of the same general random phenomenon X with probability distribution function $F_k(x) = Pr\{X_k <= x\}$.

Perhaps the most fundamental assumption to be made about a process is the dimensionality of X as univariate versus multivariate. If each random variable X represents a single quantity such as the service time of a customer, the process is called univariate; X_k would be the service time of the *kth* customer to arrive at the system. If, instead, X represents a number of quantities such as the amounts for different items within a single order, the process is called multivariate; here $X_k = \{A_k, B_k, ...\}$ could present the amounts of items A, B, and so on, on the *kth* order submitted to an inventory system. In general, whenever a multivariate process is considered, the assumptions concerning the interrelationships of random variables become more complicated due to the increased dimensionality.

Discrete-event simulation models typically have stochastic components that mimic the probabilistic nature of the system under consideration. Successful input modelling requires a close match between the input model and the true underlying probabilistic mechanism associated with the system. The input data analysis is to model an element (e.g., arrival process, service times) in a discrete-event simulation

given a data set collected on the element of interest. This stage performs intensive error checking on the input data, including external, policy, random and deterministic variables. System simulation experiment is to learn about its behaviour. Careful planning, or designing, of simulation experiments is generally a great help to save time and effort by providing efficient ways to estimate the effects of changes in the model's inputs and on its outputs. Statistical experimental-design methods are mostly used in the context of simulation experiments. Simulation modelling and the consequent system analysis mainly discuss:

1. *Performance Evaluation and What-If Analysis:* The 'what-if' analysis is at the very heart of simulation models.

2. *Sensitivity Estimation:* Users must be provided with affordable techniques for sensitivity analysis if they are to understand which relationships or changes to parameters have the highest effect on the system.

3. *Optimisation:* Traditional optimisation techniques require gradient estimation. As with sensitivity analysis, the current approach for optimisation requires intensive simulation to construct an approximate surface response function.

4. *Gradient Estimation Applications:* There are a number of applications which measure sensitivity information, (i.e., the gradient, Hessian, etc.), Local information, Structural properties, Response surface generation, Goal-seeking problem, Optimisation, What-if Problem, and Meta-modelling.

5. *Report Generating:* Report generation is a critical link in the communication process between the model and the end user.

To perform statistical analysis of the simulation output we need to establish some conditions, e.g. output data must be a **covariance stationary process** (e.g. the data collected over *n* simulation runs).

a. Stationary Process (strictly stationary): A stationary process (or strictly stationary process) is a stochastic process whose probability distribution at a fixed time or position *{X (t), t ∈ T}* is the same for all times or positions. As a result, parameters such as the mean and variance also do not change over time or position. This can well be used in Economics and Process Analysis where a trend in a time series is observable. For example, an operator conducting a task in a period of time

with a specified variance and mean or mean time between arrivals at a check-in boot in an airport.

b. First Order Stationary: A stochastic process is a first order stationary if expected of *X (t)* remains the same for all t. For example in economics time series, a process is first order stationary when we remove any kinds of trend by some mechanisms such as differencing.

If we let x_{t1} represent a given value at time t_1, then we define a first-order stationary as one that satisfies the following equation:

$$f_x(x_{t1}) = f_x(x_{t1+\tau})$$

The physical significance of this equation is that our density function, $f_x(x_{t1})$, is completely independent of t_1 and thus any time shift, τ. [8]

The most important result of this statement, and the identifying characteristic of any first-order stationary process, is the fact that the mean is a constant, independent of any time shift. Below we show the result for a random process, X, that is a discrete-time signal, $x[n]$.

$$X = m_x[n]$$

$$= E[x[n]$$

$$= \text{constant (independent of } n)$$

c. Second Order Stationary: A random process is classified as second-order stationary if its second-order probability density function does not vary over any time shift applied to both values. In other words, for values x_{t1} and x_{t2} then we will have the following be equal for an arbitrary time shift τ.

$$f_x(x_{t1}, x_{t2}) = f_x(x_{t1+\tau}, x_{t2+\tau})$$

From this equation we see that the absolute time does not affect our functions, rather it only really depends on the time difference between the two variables. Looked at another way, this equation can be described as:

$$\Pr[X(t_1) \leq x_1, X(t_2) \leq x_2] = \Pr[X(t_1+\tau) \leq x_1, X(t_2+\tau) \leq x_2]$$

These random processes are often referred to as strict sense stationary (SSS) when *all* of the distribution functions of the process are unchanged regardless of the time shift applied to them.

For a second-order stationary process, we need to look at the autocorrelation function to see its most important property. Since we have already stated that a second-order stationary process depends only on the time difference, then all of these types of processes have the following property:

$$R_{xx}(t,t+\tau) = E[X(t+\tau)] = R_{xx}(\tau)$$

d. Covariance Stationary: A covariance stationary process is a stochastic process *{X (t), t $\in$ T}* having finite second moments, i.e. expected of *[X (t)]*^{2} be finite. Clearly, any stationary process with finite second moment is covariance stationary. A stationary process may have no finite moment whatsoever.

Consider the following two extreme stochastic processes:

- A sequence *Y_0, Y_1 ...,* of independent identically distributed, random-value sequence is a stationary process if its common distribution has a finite variance then the process is covariance stationary.

- Let *Z* be a single random variable with known distribution function, and set *Z_0 = Z_1 = ...Z.* Note that in a realisation of this process, the first element, *Z_0,* may be random but after that there is no randomness. The process *{Z_i, i = 0, 1, 2, ...}* is stationary if *Z* has a finite variance.

Output data in simulation fall between these two types of process. Simulation outputs are identical and mildly correlated (depends on e.g. in a queuing system how large is the traffic intensity ρ). An example could be the delay process of the customers in a queuing system.

4.2.1 Techniques for the Steady State Simulation

Unlike in queuing theory where steady state results for some models are easily obtainable, the steady state simulation is not an easy task. The opposite is true for obtaining results for the transient period (i.e., the warm-up period).

Gathering steady state simulation output requires statistical assurance that the simulation model reaches the steady state. The main difficulty is to obtain independent simulation runs with exclusion of the transient period. The two techniques commonly used for steady state simulation are the Method of Batch means, and the Independent Replication.

None of these two methods is superior to the other in all cases. Their performance depends on the magnitude of the traffic intensity. The other available technique is the Regenerative Method, which is mostly used for its theoretical nice properties; however it is rarely applied in actual simulation for obtaining the steady state output numerical results.

Regenerative Method
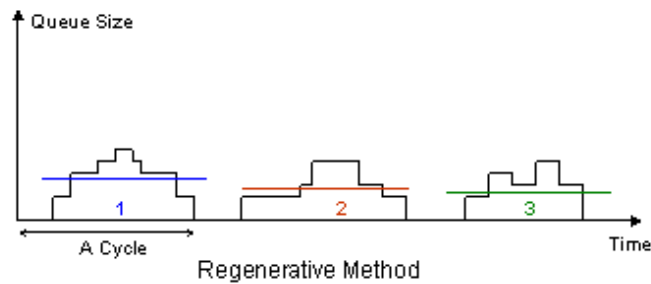
Suppose you have a regenerative simulation consisting of m cycles of size $n_1, n_2, ...n_m$, respectively. The cycle sums is:

$$y_i = \Sigma\, x_{ij}\,/\,n_i, \quad \text{the sum is over } j=1,\ 2,\ ..,\ n_i\ (\text{size of cycles})$$

The overall estimate is:

$$Estimate = \Sigma y_i\,/\,\Sigma\, n_i, \quad \text{the sums are over } i=1,\ 2,\ ..,\ m\ (\text{number of cycles})$$

The $100(1-\alpha/2)\%$ confidence interval using the Z-table (or T-table, for m less than, say 30), is:

$Estimate \pm Z. S/(n. m^{1/2})$ where, $n = \Sigma n_i /m$,   *the sum is over $i=1, 2, ..,m$* and the variance is: $S^2 = \Sigma (y_i - n_i . Estimate)^2/(m-1)$,   the sum is over $i=1, 2, ..,m$

The Batch Means method involves only one very long simulation run which is suitably subdivided into an initial transient period and n batches. Each of the batches is then treated as an independent run of the simulation experiment while no observation is made during the transient period which is treated as warm-up interval. Choosing a large batch interval size would effectively lead to independent batches and hence, independent runs of the simulation, however since number of batches are few one cannot invoke the central limit theorem to construct the needed confidence interval. On the other hand, choosing a small batch interval size would effectively lead to significant correlation between successive batches therefore cannot apply the results in constructing an accurate confidence interval.

Method of Batch Means
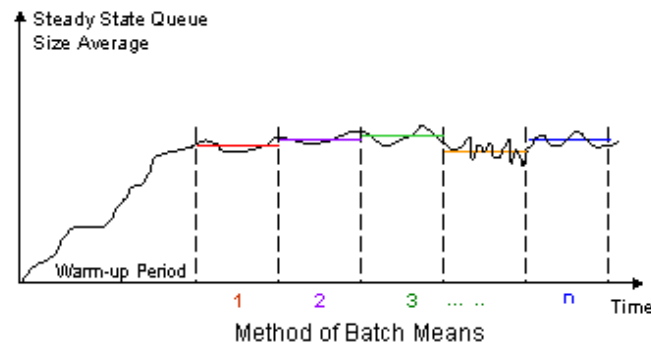
Suppose you have n equal batches of m observations. The means of each batch is:

$mean_i = \Sigma x_{ij} / m$,   the sum is over $j=1, 2, .., m$ (number of observations)

The overall estimate is:

$Estimate = \Sigma mean_i / n$,   the sum is over $i=1, 2, ..,n$ (number of batches)

The 100(1-α/2)% confidence interval using the Z-table (or T-table, for n less than, say 30), is:

$Estimate \pm Z. S$ where the variance is: $S^2 = \Sigma (mean_i - Estimate)^2/(n-1)$,   the sum is over $i=1, 2, ..,n$.

The Independent Replications method is the most popular for systems with short transient period. This method requires independent runs of the simulation experiment different initial random seeds for the simulators' random number generator. Due to extraordinary conditions of systems at start of a simulation run e.g. systems being empty and the short period of simulation a transient period is defined. For each independent replications of the simulation run the results of the transient period is removed from the analysis. For the observed intervals after the transient period data is collected and processed for the point estimates of the performance measure and for its subsequent confidence interval.

Method of Independent Replications
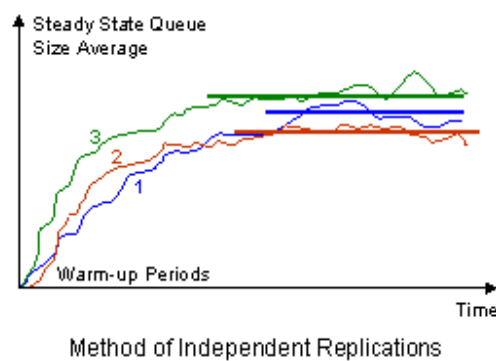
Suppose you have n replications with of m observations each. The means of each replication is:

$$mean_i = \Sigma x_{ij} / m, \quad \text{the sum is over } j=1, 2, .., m \text{ (number of observations)}$$

The overall estimate is:

$$Estimate = \Sigma mean_i / n, \quad \text{the sum is over } i =1, 2, .., n \text{ (number of replications)}$$

The 100(1-α/2)% confidence interval using the Z-table (or T-table, for n less than, say 30), is:

Estimate $\pm$ Z. S where the variance is: $S^2 = \Sigma (mean_i - Estimate)^2/(n-1)$, the sum is over $i =1, 2, .., n$.

The primary purpose of most simulation studies is the approximation of prescribed system parameters with the objective of identifying parameter values that

optimise some system performance measures. If some input processes driving a simulation are random, the output data are also random and runs of the simulation result in estimates of performance measures. A simulation run does not usually produce independent, identically distributed observations; therefore, "classical" statistical techniques are not directly applicable to the analysis of simulation output.

4.3   Simulation Experiment Design

In experimental-design terminology, the input parameters and structural assumptions composing a model are called factors, and the output performance measures are called responses. The decision as to which parameters and structural assumptions are considered fixed aspects of a model and which are experimental factors depends on the goals of the study rather than on the inherent from of the model. Factors can be either quantitative or qualitative. Quantitative factors naturally assume numerical values, while qualitative factors typically represent structural assumptions that are not naturally quantified.

We can also classify factors in simulation experiments as being controllable or uncontrollable, depending on whether they represent action options to managers of the corresponding real-world system. In a mathematical modelling activity such as simulation we do get to control everything, regardless of actual real-world controllability.

In simulation, experimental design provides a way of deciding before the runs are made which particular configurations to simulate so that the desired information can be obtained with the least amount of simulating.

There are some advantages through simulation experiments:

1. We have the opportunity to control factors such as customer arrival rates that are in reality uncontrollable. Thus, we can investigate many more kids of contingencies than we could in a physical experiment with the system.

2. Another aspect of enhanced control over simulation experiments stems from the deterministic nature of random-number generations. In simulation experiments, we can control the basic source of variability, unlike the

situation in physical experiments. Thus, we might be able to use variance-reduction techniques to sharpen our conclusions.

3. In most physical experiments it is prudent to randomise treatments (factor combinations) and run orders (the sequence in which the treatments are applied) to protect against systematic variation contributed by experimental conditions, such as steady rise in ambient laboratory temperature during a sequence of biological experiments that are not thermally isolated.

If a model has only one factor, the experimental design is conceptually simple: we just run simulation at various values of the factor, or levels, perhaps forming a confidence interval for the expected response at each of the factor levels. For quantitative factors, a graph of the response as a function of the factor level may be useful. In the case of terminating simulations, we would make some number n of independent replications at each factor level. At the minimum there would be two factor levels, thus needing $2n$ replications.

4.3.1 Response surfaces and metamodels

A simulation model can be thought of as a mechanism that runs input parameters into output performance measures. In a sense simulation is just a function, which may be vector-valued or stochastic. The explicit form of this function is also unknown, since we are going through the trouble of simulating instead of merely plugging numbers into some formula. There are some models that develop simple formulas that approximate this function. This approximate function could then be used as a proxy for the full-blown simulation itself in order to get at least a rough idea of what would happen for a large number of input-parameter combinations. This ability is especially helpful if the simulation is very large and costly, precluding exploration of all but a few input parameter combinations.

For example, the simulation of an inventory model; we take s as input the reorder-point parameter and the order size parameter d, and produced as output the average total cost per month, a random variable. We could thus in principal define the:

Average total cost per month = $R(s,d)$

For some function *R* that is stochastic, unknown, and probably pretty messy; indeed, it is the whole simulation program itself that evaluates *R* for numerical input values of *s* and *d*.

Gradient estimation: One of the goals of simulation is to find how changes in the input parameters affect the output performance measures. If the parameters vary continuously, we are essentially asking a question about the partial derivatives of the expected response function with respect to the input parameters. The vector of these partial derivatives is called the gradient of the expected response function, and is dimensionally equal to the number of input parameters considered. The gradient is interesting in its own right, since it gives the sensitivity of the simulation's expected response to small changes in the input parameters. It is also an important ingredient in many mathematical programming methods that we might try to use to find optimal values of the input parameters, since many such methods rely on the partial derivatives to determine a direction in which to research for the optimum.

As a simple example, consider an *M/M/1* queue operating in steady state, with arrival rate λ and service rate *w*; we assume that the traffic density to be $\rho = \lambda / w < 1$. The first "M" stands for the arrival process (interarrival times) are independent and identically distributed and is also exponential (Markovian). The second "M" stands for the service time distribution and the "1" stands for a single server single Queue situation. In this model, the steady state expected delay in queue of a customer is analytically known and is given by:

$$d\,(\lambda / w) = \lambda / w^2 - \lambda w \qquad [7]$$

4.4   Validation, Verification and Testing

A simulation study is normally conducted for problem solving, conducting "what if" scenarios and training. It consists of complex processes of field data collection, formulation, analysis, modelling, and experimentation. A typical simulation requires an overall knowledge in diverse disciplines such as operations research, computer science, statistics and manufacturing processes engineering. A successful simulation

study may be a credible solution that is accepted and used by senior management and key decision makers.

Model verification is substantiating that the model is transformed from one form into another, as intended, with sufficient accuracy. This requires an accurate modelling construct that handles the transition from one state into another.

Model validation is to demonstrate that the model behaves with satisfactory accuracy consistent with the study objectives. Model validation deals with building the appropriate model accounting for the nature of activities in the system, the relationship between various resources and the flow of material within the system. Model verification and validation revolves around assessment of the accuracy model compared with performance of the real system.

Model testing is ascertaining whether inaccuracies or errors exist in the model. In model testing, the model is subjected to test data or test cases to determine if it functions properly. Test failed implies the failure of the model, not the test. A test is devised and testing is conducted to perform either validation or verification or both. Some tests are devised to evaluate the behavioural accuracy (i.e., validity) of the model, and some tests are intended to judge the accuracy of model transformation from one state into another (verification).

In this chapter the steps for a successful simulation project were described. Also techniques for experimental design and simulation output analysis were discussed. The chapter was concluded by explaining the approaches to model verification, validation and testing.

References

[1] Nance, R.E. and Overstreet, C.M. (1987). "Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications", *Transactions of the Society for Computer Simulation,* Vol.4, No.1, pp. 33-57, January.

[2] Tocher, K.D. (1979). Keynote Address, In: *Proceedings of the 1979 Winter Simulation Conference,* pp. 640-654, San Diego, CA, December 3-5.

[3] Lackner, M.R. (1962), "Toward a General Simulation Capability", In: *Proceedings of the AFIPS Spring Joint Computer Conference,* pp. 1-14, San Francisco, CA, May 1-3.

[4] Kiviat, P.J. (1963). ``Introduction to Digital Simulation,'' Applied Research Laboratory 90.17-019(1), United States Steel Corporation, April 15 (stamped date).

[5] Zeigler, B.P. (1976). *Theory of Modelling and Simulation,* John Wiley and Sons, New York, NY.

[6] Balci, O. (1987). CS 4150: Modelling and Simulation Class Notes, Department of Computer Science, Virginia Tech, Blacksburg, VA, pp. 10-13, Spring.

[7] Banks, Jerry (1998): Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practices, John Wiley and Sons, New York, NY.

[8] www.cnx.org/content/m10684/latest

PART B

Chapter 5

Simulation Modelling with ARENA: An Introduction to Arena Software Package

This Chapter Covers:

1. The Arena simulation software environment

2. The modelling approaches used in Arena

3. The key concepts and terminologies in Arena

4. The building blocks in Arena

5. Building and running a simple simulation example.

Note: D. Kelton, R. Sadowski and N. B. Swets (2010), Simulation with Arena 5th Int. Edition, McGraw-Hill. And the Arena Simulation Software help files.

5.1   An introduction to the Arena simulation software

In this chapter you will be introduced to simulation software called Arena[TM]. In this and the next two chapters, you will learn how to use Arena for basic process modelling and we will follow a specific example to demonstrate the facilities and capabilities of Arena discrete event simulation tool.

There are several simulation modelling software packages that are available in the market today. Some of the other common ones besides Arena are, SIMUL8, WITNESS, AutoMOD, ED, ANYLOGIC and SIMUL. The minimum requirement for running the examples in this and the next two chapters of the book will be the academic version of the software tool Arena.

Arena provides an integrated environment for building simulation models for a wide variety of applications. It integrates all the functionalities required for a successful simulation including: Input and output data analysis, Model logic construction and Animation.

5.2   Arena's hierarchical structure

The flexibility provided by Arena is shown by its hierarchical structure shown in figure 5.1. Within a single graphical user interface, Arena provides a means of high level modelling using user-created templates whilst at the same time supporting the ultimate flexibility of user-written programmes in visual basic and C/C++. Between these two, the user has the flexibility of combining modules from different panels to obtain a unique and accurate solution to a given problem.

In the next section we will take a tour of Arena environment in order to become familiar with its menus and commands. Along with this chapter you are also advised to read chapter three of the reference text (Kelton et al, 2010).

Higher

Modelling Level

Lower

User-Created Templates
Commonly used constructs, Company-specific processes and Company-specific templates

Application Solution Template
Contact centres, Packaging lines, etc.

Basic Process Panel
Many common modelling constructs
Very accessible, easy to use
Reasonable flexibility

Advance Process, Advanced Transfer Panels
Access to more detailed modelling for greater flexibility

Blocks, Elements Panels
All the flexibility of the SIMAN simulation language

User-Written Visual Basic, C/C++ Code
The ultimate in flexibility VBA is built in C/C++ requires compiler

In a single graphical user interface consistent at any level of modelling

Figure 5.1: Arena's Hierarchical Structure (Adopted from Kelton et al, 2004, chapter 1, pp 13)

5.3   A quick tour of the Arena environment

A snap shot of the Arena simulation environment is shown in figure 5.2. As shown, the main part of the screen is the model window which is split into two views: the flowchart view and the spreadsheet view. By default, Arena displays both views with the spreadsheet view below the flowchart view. If you do not see the two views on your screen, click on View on the menu bar and select split screen. Repeat the above process to turn off split screen.

The flowchart view accommodates the model's graphics including the process flowchart, animations and other drawing elements. The spreadsheet view if active displays model data in any selected module. It provides an easy way to enter and edit model data and set relevant parameters. Most model data can be entered and edited through the flowchart view but the spreadsheet view gives access to many more data at the same time, arranged in a way convenient for editing.

To the left of the Arena window is the project bar, which hosts various panels containing the objects which are used as building blocks in Arena models. Figure 5.2 shows the basic process panel displayed. Above that are buttons for the advanced process and advanced transfer panels. Arena displays only one panel at a time. Clicking on the advanced process button will hide the basic process panel and display the modules in the advanced process panels.
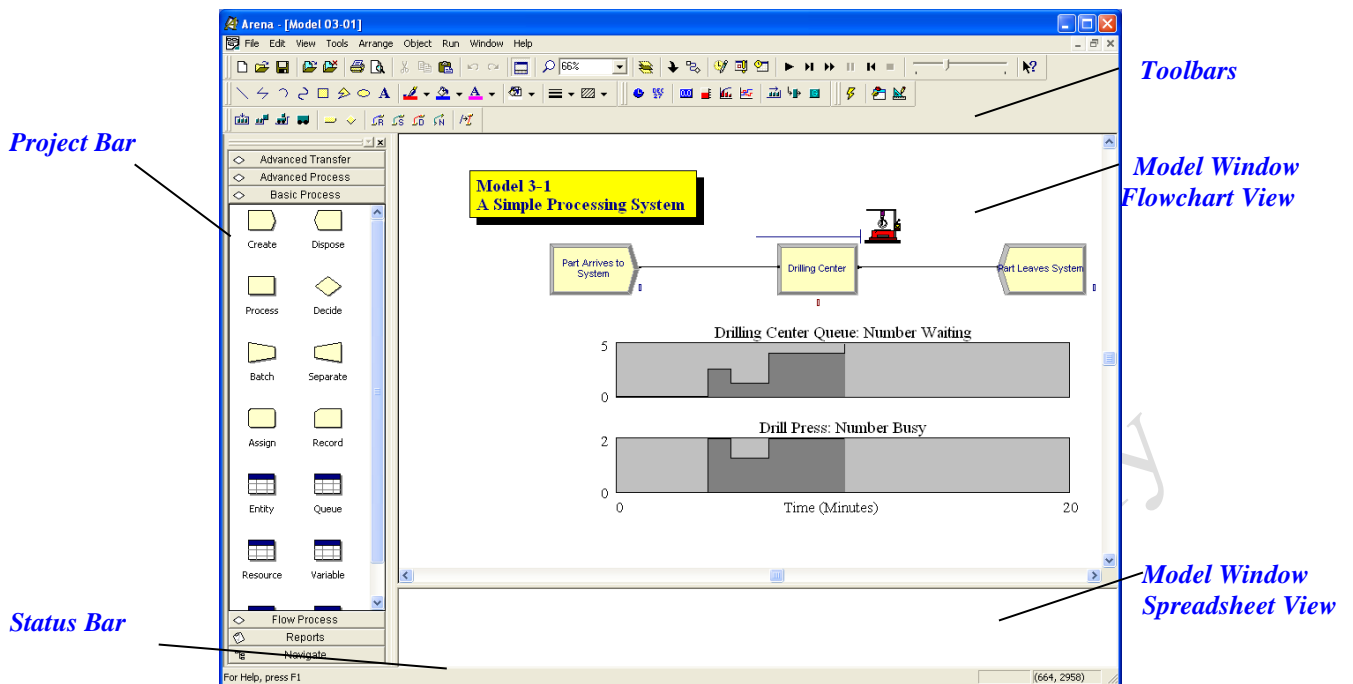
Figure 5.2: A snapshot of the Arena Simulation environment

In order to remove a panel, right click anywhere in the panel and select detach. Similarly, you can add a panel by right clicking in the project bar, selecting attach then the name of the panel from the displayed dialogue. You may want to attach and detach a few panels now.

Above the model window are the tools bars. These are mainly shortcuts to the menu items just as in most windows applications.

Once you become familiar with the Arena modelling environment, the next question is probably, how do we build a model in this environment? Well, before you learn how, there are a few things you need to look at such as some of the basic terminologies that are used in simulation in general and Arena in particular, what the building blocks are and some description of the most basic building blocks. We will start with a quick review of some of the basic concepts and terminologies in the next section.

5.4   Review of basic concepts

In chapter 3 we explained what simulation is and presented some major concepts such as systems, types of systems, models, types of models etc. In the next few subsections, we will briefly explain the key concepts or terminologies that you will be encountering as we go through the rest of this course. We will be looking at Entities, Attributes, Variables, Resources, Queues, Stations, Routes, Transporters, Conveyors, Statistical Accumulators, Time Persistent Statistics, Observed Statistics and Tally Statistics.

5.4.1 Entities

In every simulation model, *entities* are objects that undergo processes and move along the system. Kelton et al (2010) describe *entities* as the dynamic objects in the simulation. Thus they are usually created, move around for a while, and then are disposed of as they leave the system. They further noted that in as much as all *entities* have to be created, it is possible to have entities that are not disposed but keep circulating in the model or system.

For example *entities* in a manufacturing system may be raw materials or products. *Entities* in a bank system may be customers. In a hospital the *entities* may be patients and so on.

In any case however, *entities* represent the "real" things in a simulation. There can be in a typical system, especially if there are different types of parts that are processed in the modelled system.

5.4.2 Attributes

Attributes are common characteristics of entities but with specific values that can differ from one entity to another. For example, in a hospital system all patients may have an attribute called *Arrival Time* but the exact value of this arrival time attribute for each patient will depend on the time that patient arrived into the system.

The key thing to note about attributes is that, their values are tied to a specific entity or group of entities and would always remain the same until updated at some

point in the process. In a typical system, we can define as many attributes as we need for our entities. Arena however, has some default attributes such as Entity Type, Entity Picture, and Entity Sequence etc which are very helpful when building a model.

5.4.3 Variables

Variables are used to store information or values that describe or reflect some characteristics of your system, irrespective of the number, state or type of entities around. The information variables are available to all entities and not specific to any.

There are two types of *variables* in Arena: Built-in variables and User-defined *variables*. Some examples of Arena's built-in variables are Work-In-Process (WIP), current simulation time, current number in queue etc. User-defined *variables* depend on the system modeller and needs to be built into the model.

Entities can access and change the value of *variables* but they do not take up the values as they do with an attribute. Note however that the value of a variable may be assigned to an attribute at anytime. For example if you are interested in knowing the day of the week on which a product arrives into your system, you may have an attribute called *Arrival Day* and a variable called *DayOfWeek*. *DayOfWeek* will be incremented by 1 after every 24 hours of simulation time and would vary from 1 through 7 for each day of the week. Hence each time a product arrives you can assign its *Arrival Day* attribute with the following expression:

Arrival Day = DayOfWeek

In this way if the product arrived when the variable *DayOfWeek* is 2 then the product's (entity's) *Arrival Day* value will be 2.

5.4.4 Resources

Resources are facilities or persons in a system that provides services to the system entities. *Resources* usually have capacities and entities seize units of the *resource* when they are available and must be released when processing is over. It is possible for an entity to seize various units of different *resources* at the same time. An example of this is for an entity patient to require the *resources*: doctor, bed and a

nurse at the same time. *Resources* may be defined individually or as a set for modelling purposes.

It must also be noted that a resource can also serve one or more than one dynamic entity at the same time depending on its capacity. Entities will always wait in a queue when a required resource is not available.

Resources may be machines in a manufacturing simulation, cashiers in a banking simulation or Doctor in a hospital simulation. The term *"Seize"* is used to describe an entity taking up a *resource.* When the entity gives up the resource after processing is complete, it is said to have *"Released"* the resource.

5.4.5 Queues

Entities normally compete with each other for resources. When the resource required is not available, the entities need a place to wait until the required unit of the resource is available for them to seize. This waiting place is called a *Queue.*

In Arena, queues have names and can also have capacities to represent, for example, limited floor space for a buffer or storage. There are a number of rules that determine how a resource serves entities waiting in a *queue.* Arena by default applies the First-Come-First-Served (FCFS), or First-In-First-Out (FIFO) rule to all *queues.* Other *queuing rules* are: Last In-First-Out (LIFO), Lowest Attribute Value First (LVF), Highest Attribute Value First (HVF) or other criteria which might influence the way entities can be served in the *queue.*

5.4.6 Transporters

In Arena, *Transporters* may be referred to as moveable resources. These are used for moving entities from point to point in the system. This puts a constraint on the number of entities what can be transferred at one time. The number of entities transferred will depend on the number of *Transporters* available and their capacities. Kelton et al 2004 refer to entity transfer using *Transporters* as resource constrained transfer. Some examples of *Transporters* are AGVs, trucks, fork lifts, cranks, carts etc. (figure 6.3).

Figure 5.3: Picture of a Fork Lift, an example of a Transporter
(Source of photo: www.rollsscaffold.com)

5.4.7 Conveyors

Conveyors are similar to *Transporters* in that they are also used to transfer entities in the system. *Conveyors* however, are devices that move entities from one station to another in one direction only, such as escalators and horizontal (roller or belt) unidirectional conveyors (figure 5.4).

Figure 5.4: Picture of a Conveyor belt moving boxes
(Source of photo: www.fotosearch.com)

5.4.8 Statistical accumulators

The statistical accumulators are types of variables that "watch" (observe) what goes on during a simulation run. They are "passive" elements in the model, they do not participate but just observe. Most of them are built into Arena and are used automatically but they may also be user-defined for special cases. Some examples of statistical accumulators are:

- Number of parts produced so far
- Total of waiting time spent in queue so far
- No. of parts that have gone through the queue
- Maximum time in queue we've seen so far
- Total of times spent in system
- Maximum time in system we've seen so far
- Area so far under queue-length curve Q(t)

- Maximum of Q(t) so far
- Area so far under server-busy curve B(t)

All of the above need to be initialized to 0 at the start of the simulation. As the simulation progresses, Arena updates all of them and at the end of the simulation run, it uses them to calculate the output performance measures.

5.4.9 Time persistent statistics

Time Persistent Statistics are those that result from taking the (time) average, minimum and maximum of a plot of some attribute or variable during the simulation, where the x-axis is continuous time. Time persistent statistics are also known as *continuous-time* statistics.

A classic example of a *Time Persistent Statistics* is number in queue (queue length). A queue of infinite capacity may theoretically have any number of entities from one to infinity in queue. Finding the time average for this value takes into consideration the duration of time for which the queue was at a particular level and not just a simple average. Figure 5.5 shows a time persistent plot of the number of entities in a queue. Note that at time "A", the number in queue went down to 1 and remained at that level till time "B". Tally statistics on the other hand are not time dependent as discussed in the next subsection.
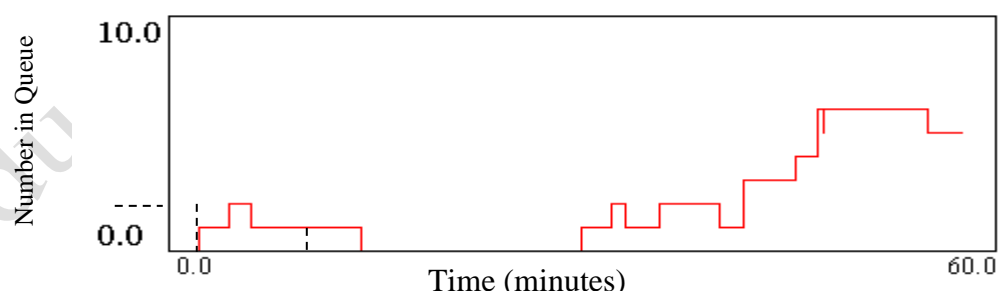
Figure 5.5: Time Persistent Plot for Number in a Queue

5.4.10 Observed (Tally) statistics

Tally statistics, sometimes called *discrete-time statistics* are those that result from taking the average, minimum, or maximum of a list of numbers. An example of this is the average and maximum total time in system. These statistics are observed at

discrete time intervals and are not continuous. Considering a queue at any process, whilst the number of entities in the queue is time persistent statistic, the time spent in queue is tally statistics. The difference is that, whilst the number in queue may remain at some value say 1 over a period of time, you can only tell how long an entity spent in a queue only after the entity has left the queue and occurs at a specific instant in the simulation period.

5.4.11 Counter statistics

Counter statistics are accumulated sums of a specified statistics. They are usually simple counts of how many times something happened during the simulation. An example of counter statistics is to count the number of entities that have entered a process. Counter statistics could also be accumulations of numbers that are not equal to 1, such as accumulating the wait time for each entity at a particular process to obtain the total waiting time at that process. This is a sum of all individual wait time and not an average.

5.5 The building blocks in Arena

Simulation models normally represent complex systems and are sometimes complex to build. It is therefore important to understand what pieces are put together and how, particularly in Arena. The main building blocks in Arena are the flowchart and data modules. We will look at these in detail before the first simulation model is built.

5.5.1 Flowchart modules

According to Kelton et al (2010), you can think of flowchart modules as being nodes or places through which entities flow, or where entities originate or leave the model. These modules mainly describe the dynamic process of the model. The modules are normally contained in their respective panels and displayed in the project bar as mentioned earlier. To place any flowchart module in the model window, click on the flowchart module once, hold down the mouse button and drag the module where you want it in the model window as shown in figure 6.6.

The panels typically contain a collection of flowchart modules suitable for some aspect of modelling. The *Basic Process Panel* contains the following flowchart modules; Create, Dispose, Process, Decide, Batch, Separate, Assign and Record. These are suitable for building basic high level models. A close look at each of the flowchart modules reveals that it has a distinctive shape which is suggestive of what it does. There are many other kinds of flowchart modules in all the panels differing in shape and colour but clearly labelled in words to suggest their functionalities. You may want to open the various panels and examine the various collections of flowchart modules they contain.

A flowchart module may be edited by double-clicking on it once it is placed in the flowchart view in the model window. This brings up a dialog box in which all data specific to the particular module could be entered. An alternative way for editing a flowchart module is to click to select it and Arena will always display a row of data in the spreadsheet view that is specific to the selected module. If there are more than one of the same kind of module in your model, Arena will display all of them as rows in the spreadsheet view.
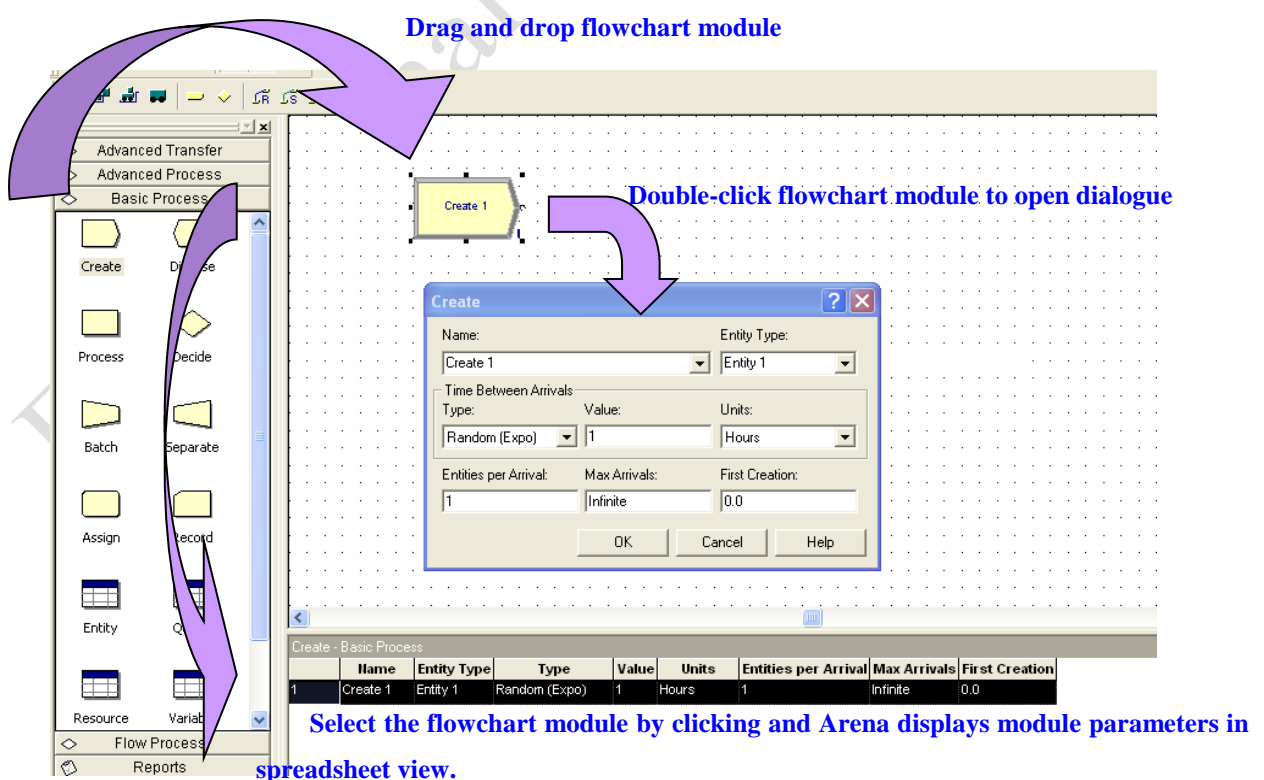
Figure 5.6: Placing a module in the model window and ways to edit data

5.5.2 Data modules

Data modules are primarily used to define the characteristics of various system elements such as queues, resources, variables and entities. They are also used to create variables and expressions. Some data modules in the basic process panel are Entity, Queue, Resource, Variable, Schedule and Set. Refer to the reference text for further discussion on data modules.

To define a data module, click once on the module's icon in the Project bar to activate its spreadsheet. Double-click in the designated space to add a new row. (Each row in the spreadsheet represents a separate module.) Then edit the data as you would in a standard worksheet.

Data and flowchart modules differ in several ways. First, data modules exist only in spreadsheet form, while flowchart modules exist both as an object in the model workspace and as a row in the spreadsheet. Second, data modules can be added or deleted via the spreadsheet, while flowchart modules can only be added or deleted by placing the object or removing the object from the model workspace.

5.6   Three (3) basic modules

With only three modules in Arena, you can build and run a very simple simulation model. These modules are the *Create, Process* and *Dispose* modules found in the *Basic Process Panel*. We want to introduce you to these basic modules before we start to do some basic modelling. We present a very detailed treatment of these modules and different ways in which they may be used in a simulation model. Similar treatment of all other modules in the *Basic Process Panel* and others in the *Advanced Process* and *Advanced Transfer Panels* are presented in chapter 7 where we introduce the module by module approach to learning Arena.

5.6.1 Create module

The main purpose of the *Create Module* is to provide a starting point for entities in a simulation model. In other words, this module is used to create entities into the simulation model. Entities can be created in four (4) major ways:

1. According to a random (Expo) distribution
2. According to a predefined schedule
3. According to a constant value (rate)
4. According to an expression

Figure 5.7 shows the module shape and its dialog. The *Name* field represents a unique identifier or name that should be given to the module. This name is displayed on the module shape. It is helpful to use names that are descriptive of the type of entities that the module creates for example, "create parts", create products", "create patients", and "create customers" etc.
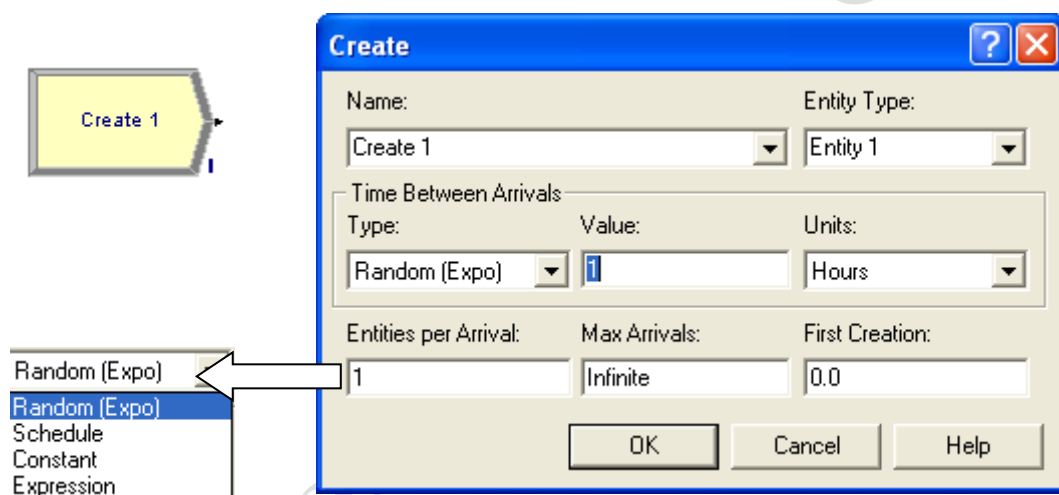
Figure 5.7: Create module and its dialog

The *Entity Type* field is the name that would be given to the entities that would be created from this particular instance of the module. This could be for example Parts, Customers, Patients, Part 1, Customer 1, Product 1 etc. Arena sets this value to Entity 1 by default.

The group of fields labelled *Time Between Arrivals* determine the way in which and the rate at which the entities are created. When *Type* is *Random (Expo)* then the *Value* field represents the mean value of the exponential distribution and *Units* represents the time units in Hours, Minutes, Seconds or Days. As can be seen from the insert in figure 5.7, Random (Expo) is just one method of creating entities.

When you select the option *Schedule* in the *Type* field the dialog changes to the view shown in figure 5.8. Arena now gives you the option to specify a schedule name. To use this type of entity creation, you should have already defined a schedule in your module (we will discuss the subject of schedule later). The number of entities therefore created, and the rate of arrival would depend on the details of your schedule.
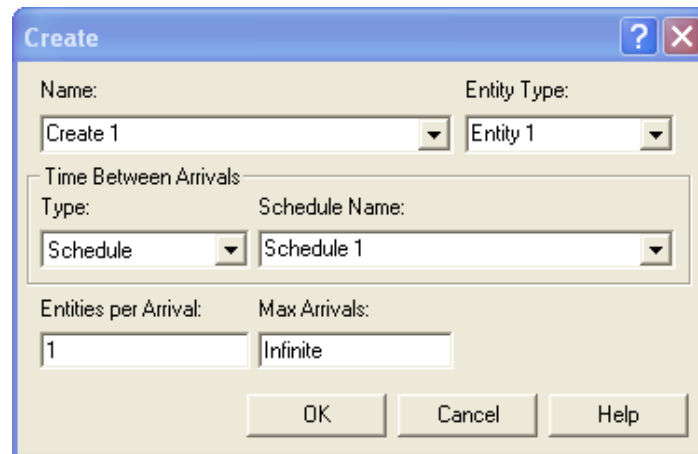
Figure 5.8: Create dialog with schedule option

When *Type* is *Constant,* the dialog view is the same as in figure 5.7. The *Value* field may be for example 30 and the *Units* minutes. This means that Arena should create 1 entity (i.e. if the *Entities per Arrival* field is 1) every 30 minutes starting from time 0.0 (i.e. if *First Create Time* is 0.0).

When *Type* is *Expression,* the dialog view remains as in figure 5.7 except that the *Value* field changes to *Expression* and Arena gives you a drop down list of standard expressions to choose from or to specify your own expression using the Arena's expression builder (figure 5.9). For example you could build the expression

DayOfWeek×5

Where DayOfWeek is a variable that varies from 1 through 7 depending of which day of the week it is. Thus on Sunday,

DayOfWeek = 1

Hence DayOfWeek×5 = 1×5 = 5

Therefore Arena will create entities every 5minutes assuming units is minutes.

Similarly, if DayOfWeek = 2 for Monday then

DayOfWeek$\times 5 = 2\times 5 = 10$

Therefore Arena will create entities every 10minutes and so on.

The *Entities per Arrival* field refers to number of entities that will enter the system at a given time with each arrival. This may also be a single value or specified as an expression.

The *Maximum Arrivals* field also refers to the maximum number of entities that this module will generate. When this value is reached, the creation of new entities by this module ceases. The value of this field may also be an expression as described above.

Finally we have the field *First Creation* which refers to the time for the first entity to arrive into the system. When *Type* is *Schedule* then this field does not apply because the start of creation will be determined by the schedule.
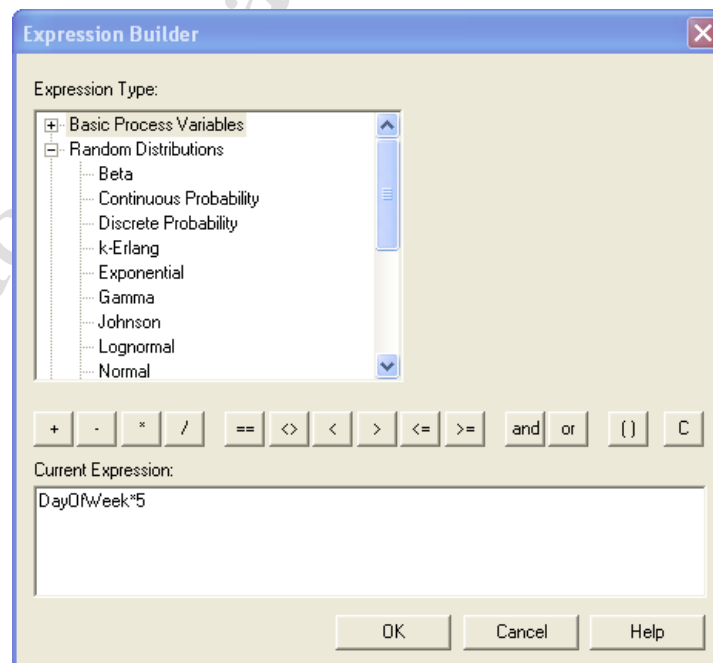
Figure 5.9: Arena's expression builder

5.6.2 Process module

The *Process Module* is the main processing method in the simulation model. With module shape and dialog as shown in figure 5.10, the *Process Module* can be used for both standard and "submodel" processes. When the process type is *Standard* as shown in figure 5.10, there are four possible actions that can be taken. The first option is a delay. When modelling a process that does not require the use of a resource, then this may be an appropriate option.

The next option of *Seize Delay* is used when the process is such that an entity has to seize one or more resources, delay them but will not release them until a later time in the simulation period. When this option is selected Arena displays a different dialog view as in figure 5.11 with an option to add resources. It can be seen from the *Resource* dialog in figure 5.11 that the *Type* field may be either a *Resource* or a *Set* of resources. When there is only one resource available for the process, then the type would be resource and the *Resource Name* field would be the name of the resource for example Machine, Doctor, Nurse, Cashier etc.
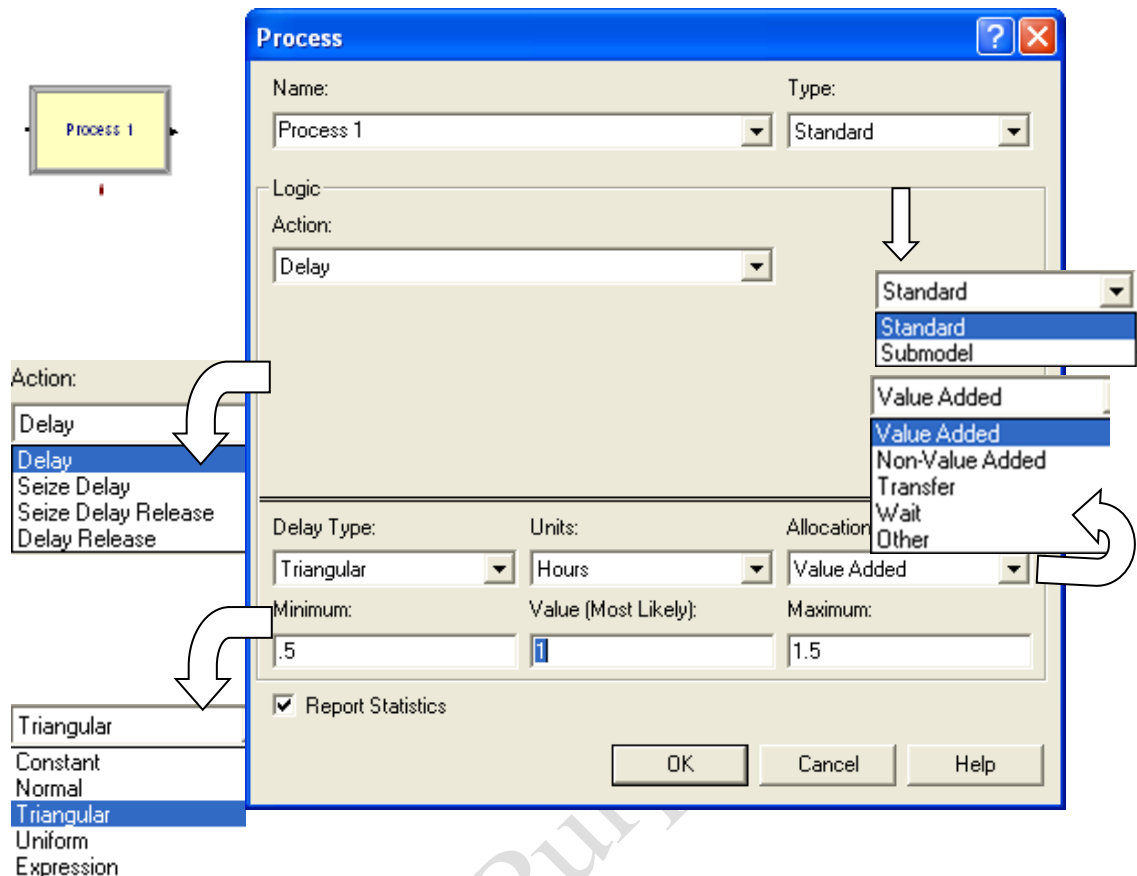
Figure 5.10: Process Module and its Dialog

On the other hand when there is a group (or a defined set) of resources available to the entity, then the type field should be *Set*. Selecting the type, *Set* changes the resource dialog view to figure 5.12. The *Set Name* field is requires a unique name or identifier since there may be more than just one resource set in a real model. The *Selection Rule* field contains options such as *Cyclical, Random, Preferred Order, Specific Member, Largest Remaining Capacity or Smallest Number Busy.* If you have for example four (4) machines in a work area that do the same thing, you may want to use them one after the other (cyclically) or just at random whenever a new entity arrives at the process. However, if you have a senior nurse amongst a group of nurses, who is the only one to decide on a patients condition, then when that patient (or entity) arrives he or she needs to first see that specific member of the group (or set of nurses). Therefore an appropriate selection rule will be the *Specific Member* option. There is in fact not a right or wrong selection here. It only depends upon what situation you are trying to model.
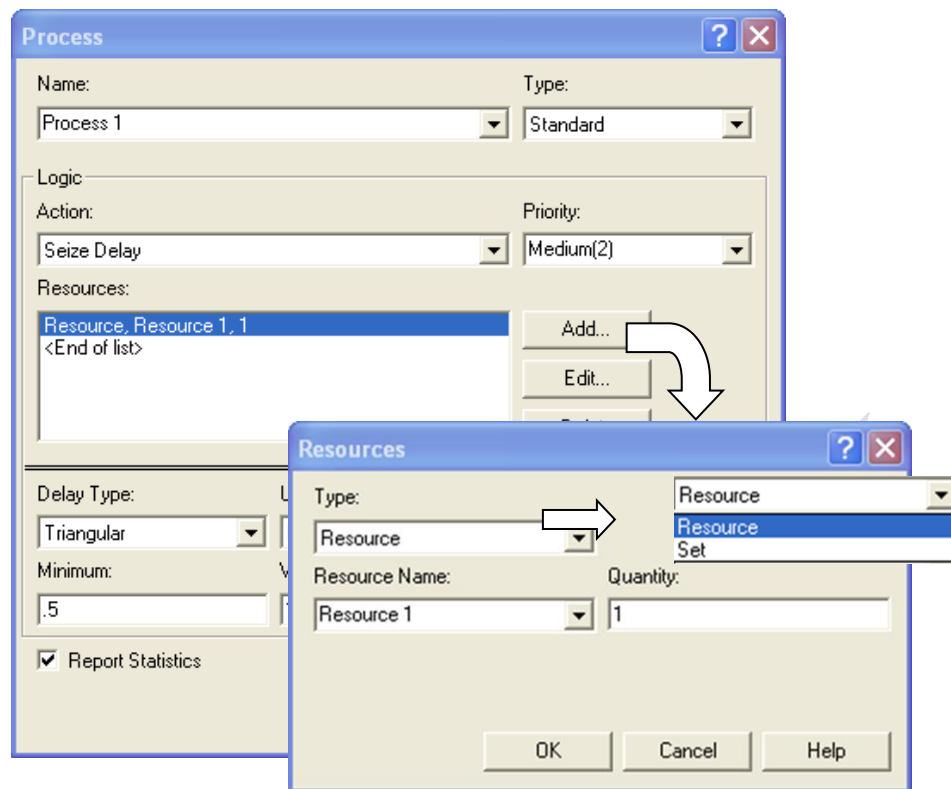
Figure 5.11: Adding a resource to the process module

The *Save Attribute* field is requires an attribute name that would be used to store the index number into the set of the member that is chosen. This attribute can later be referenced with the *Specific Member* selection rule. This applies only when Selection Rule is other than *Specific Member*. It does not apply when Selection Rule is *Specific Member*. If Action is specified as *Delay Release*, the value specified defines which member (the index number) of the set to be released. If no attribute is specified, the entity will release the member of the set that was last seized.

The *Quantity* field thus refers to the number of resources of a given name or from a given set that will be seized or released. For sets, this value specifies only the number of a selected resource that will be seized or released (based on the resource's capacity), not the number of members of a set to be seized or released.
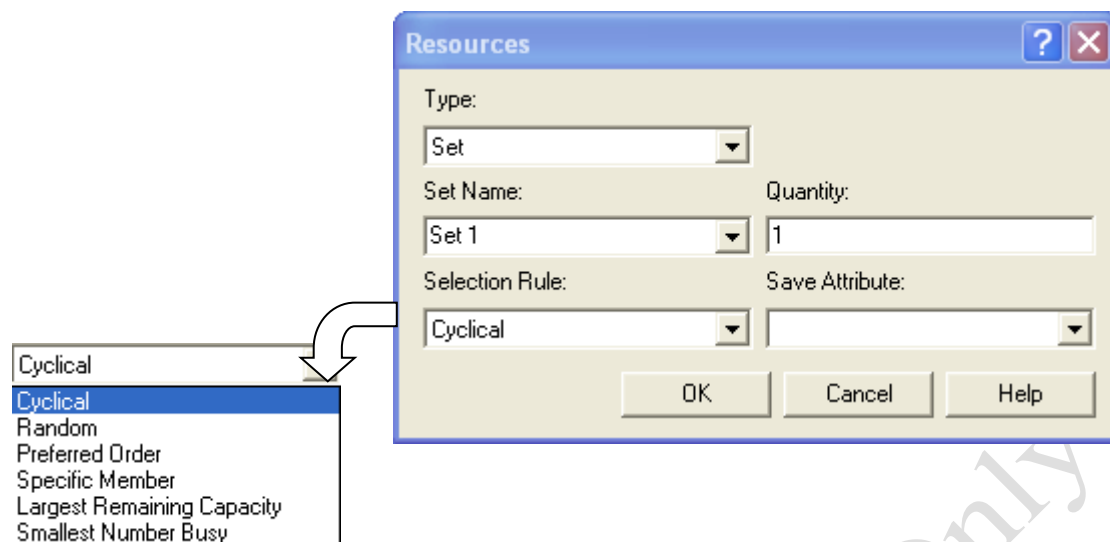
Figure 5.12: Resource dialog with type Set selected.

When the Action *Seize Delay* was selected as shown in figure 5.11, Arena added another field labelled *Priority.* This requires the priority value of the entity waiting at this module for the specified resource(s). It is used when one or more entities from other modules are waiting for the same resource(s). A classic example of using this option is when a Doctor sees both a minor category of patients and emergency patients. You may have one process module for the minor category patients' process and another module for the emergency patients and make sure they seize the same resource, the doctor. Now in order to let the emergency patients have the Doctor whenever they need him or her, you set the priority in the emergency patient process module to high (1) and that for the minor category patient process to medium (2). Note that this field does not apply when Action is *Delay* or *Delay Release*, or when the process Type is *Submodel*. We have so far been looking at the *Delay* and *Seize Delay* Actions. The next we want to consider is the *Seize Delay Release* Action.

The *Seize Delay Release* Action means that a resource(s) will be allocated (or seized) followed by a process delay and then the allocated resource(s) will be released. The fields required for this action are the same as having a Seize *Delay* action as in figure 5.11. This is the most common action in most discrete event systems for example machines processing parts, cashiers serving customers, doctor seeing a patient etc. Note that for a patient however the action on a bed resource would rather be a *Seize Delay* since he or she would release the bed resource only when about to leave the system after discharge thus later on in the process.

Finally, we look at the Action of *Delay Release*. This normally indicates that a resource(s) has previously been allocated and that the entity will simply delay and release the specified resource(s). Note that all the Actions described above apply only when Type is *Standard*.

Before we finish with the *Standard* Process Type, let's look at the set of fields to the bottom of the dialog box. As shown in figure 5.10, the "Delay Type" refers to a list of standard probability distributions that you can select from to describe the nature of your process delay in this module. There is also the option to build your own expression using the Arena Expression builder (see figure 5.9 and corresponding section). Any type of expression you select in the list, Arena will provide all the necessary fields to specify its parameters. For example selecting a triangular distribution in figure 5.10, Arena provides the fields for the minimum value, modal (most likely) value and the maximum value. For more on the statistical distributions used in Arena, refer to Kelton et al, 2010.

The other important field on this dialog is *Allocation*. This determines how the processing time and process costs will be allocated to the entity. The process may be considered to be *value added, non-value added, transfer, wait or other* and the associated cost will be added to the appropriate category for the entity and process. By definition, a *value added* process or time is that which transforms a product or service, causing it to be worth more, for example the process spraying a car in the manufacturing system. Thus if on the other hand the process or time spent does not add any value to the product then it is a *non-value added* process or time. The time spent in moving the product around the system is allocated as *transfer* and that during which the entity has to wait for another step of event to be allocated as *wait*. If the description of the time allocation does not fit any of the above then this may be assigned the allocation *other*.
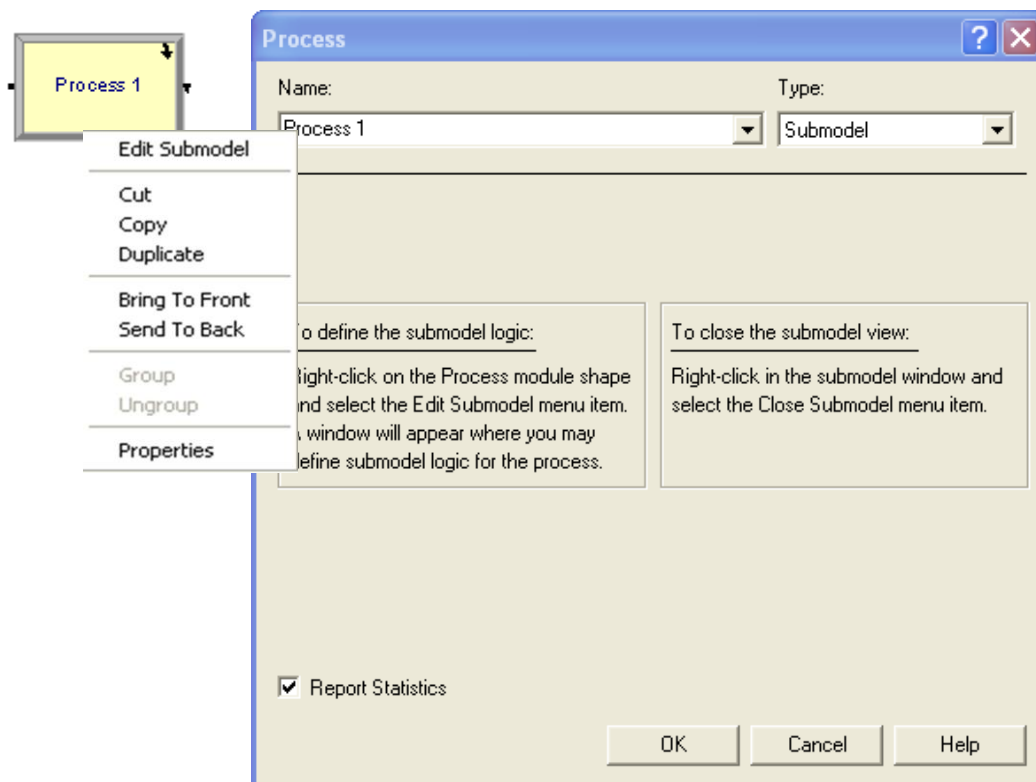
Figure 5.13: Process Module dialog with Type Submodel selected.

Now, going back to the Type field, you will realise that we have only been dealing with the standard process type till now. We will now look at the submodel Type. *Submodel* indicates that the logic will be hierarchically defined in a "submodel" that can include any number of logic modules. It is important to note that all the logic that would be defined in the submodel should be understood as taking place within the process that is represented by this particular instance of the process module.

When the Type *Submodel* is selected, the dialog view changes completely to what is shown in figure 5.13. Notice the change in the module shape (a small downward arrow at the top right corner of the module shape) to indicate that this is a submodel process.

Arena displays the new dialog view with two pieces of information, one on how to define the *submodel* logic and the other on how to close the *submodel*. To begin your *submodel*, you first have to click the OK button to accept the *submodel* Type selection and to close the dialog box. Now right click the module shape and select "Edit Submodel" from the menu list that pops up as in figure 5.13. This will open a

blank model window for the *submodel* with an Entry point and an Exit point as shown in figure 5.14. In this environment, should be able to hierarchically define a *submodel* that can include any number of logic modules.
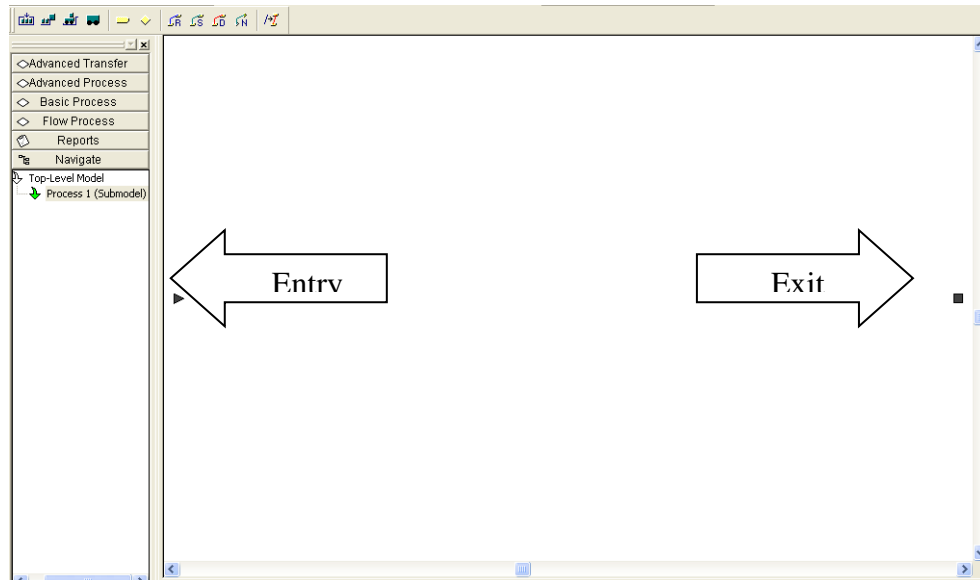
Figure 5.14: Process Module Submodel editing environment.
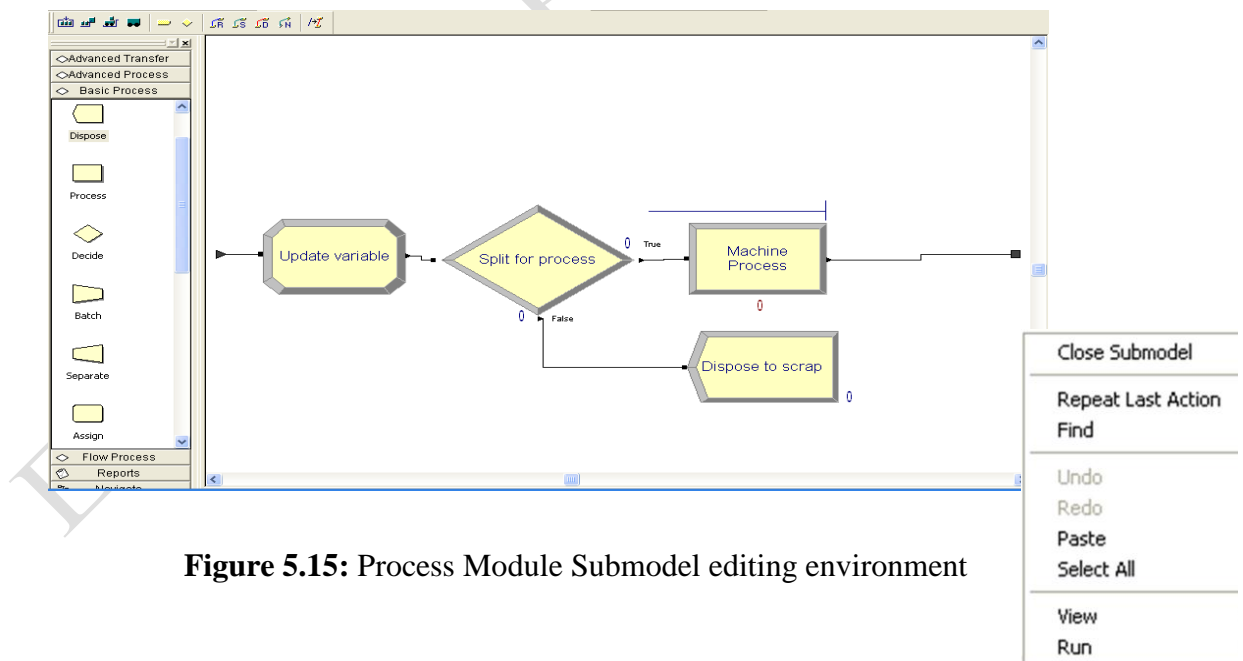
Figure 5.15: Process Module Submodel editing environment

As shown in figure 5.15, we have defined the logic between the entry and exit points of our submodel. In this case we decided to update a variable, split the

incoming parts and send a percentage to scrap and then allow the rest to through the machine process. Remember that all of these are going on within the same process module of Type Submodel. Now to leave this environment, right-click any where in the model view and select "Close Submodel" from the menu that pops up as shown in figure 5.15 above. This takes you back to the *Process 1* module shape as in figure 5.13 and you can then continue to build your main model in the usual way by adding the required modules. You can always go back to your *Process 1* submodel by following the same procedure as above at anytime.

You might have realised that one option that is common to both the *Standard* Type process dialog and *Submodel* Type process dialog as shown in figures 5.10 and 5.13 respectively is the *Report Statistics* check box. This option mainly specifies whether or not statistics will automatically be collected and stored in the report database for this process. Checking the box enables statistics collection and vice versa. Arena by default will check this option each time you add a new process module.

5.6.3 Dispose module

The *Dispose Module* (figure 5.16) is intended to be the exit point of the model where all entities leave the system. The *Name* field is the unique identifier for the module. Th*e Record Entity Statistics* check box determines whether or not the incoming entity's statistics will be recorded. Statistics include value added time, non-value added time, wait time, transfer time, other time, total time, value added cost, non-value added cost, wait cost, transfer cost, other cost, and total cost.
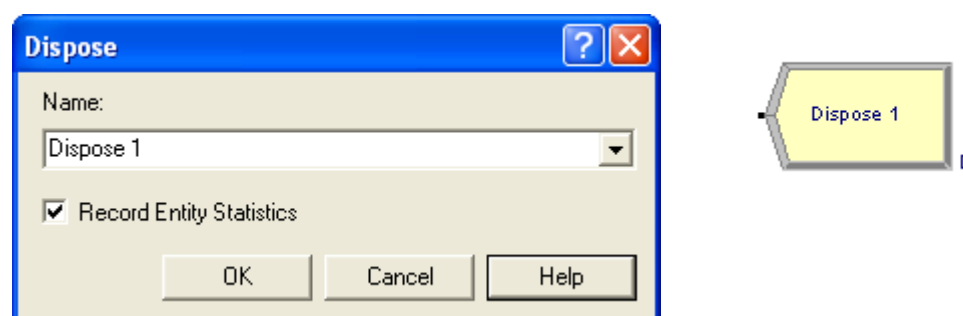
Figure 5.16: Dispose Module and its Dialog

Arena uses this module to calculate how many entities have left the system (Number out) and how many are currently in process (Work-In-Process, WIP). Entities that have been put into temporary batches must be split before being disposed else Arena will give an error when the entity is being disposed of. Similarly, all entities must release any previously seized resources before being disposed. The effect of unreleased resources is an accumulation of waiting entities at the process where that resource is needed.

5.7   Model 5-1: Basics of modelling in Arena

With a good understanding of flowchart and data modules as the building blocks in Arena, you are now ready to build your first simulation model.

If you think of simulation as a journey towards reality, you can start from anywhere so long as you are aiming to capture what happens in the real world. The closer you get to it the better. In chapter 3, we looked at the major concepts in simulation modelling including the concepts of systems. We realised that the key things in the definition of a system are its scope and level. These refer to the boundaries and levels of detail of the system, Stuart, 1998.

To start with, consider the simple single process system shown in figure 6.17. It starts with parts entering the hypothetical system, going through a process and then exiting the system. We need only three flowchart modules in Arena to model the logic of this system.

To do this we have to first create the arriving parts, send them off to the processing area where they will take some time as they are being worked upon. After the process, the parts are then sent out of the system through the exit point.

A. Mousavi – Brunel University, UK

103
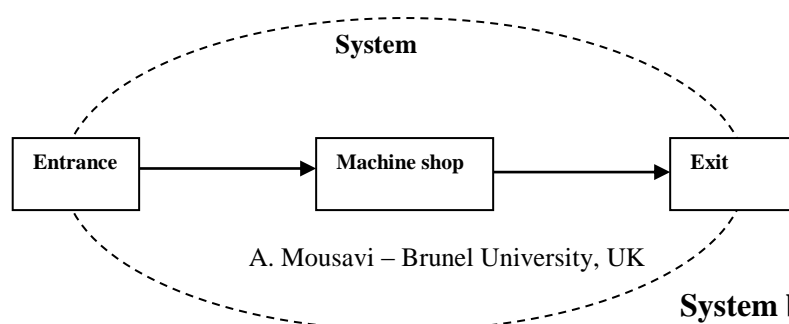
5.7.1 Building the model

To create entities or parts in Arena, we use the Create flowchart module. Drag and drop a Create flow chart module into your model window flowchart view and double click the module to display the property dialogue box. Fill in the required information as shown in the figure 5.18.
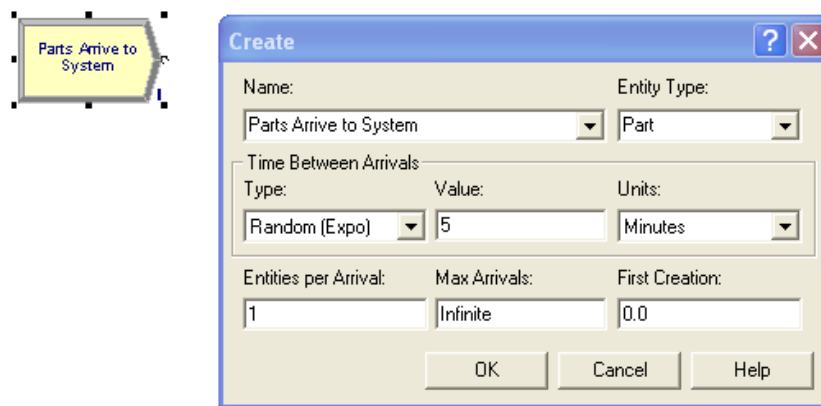
Figure 5.18: The Create Property Dialogue Box for Model 5.1

The module name, *Parts Arrive to System* is mainly for identification purposes. It will uniquely identify this instance of the Create module within the model. It is very helpful as with other modules to make the name descriptive of the process for which it represents for ease of identification and clarification. The Entity Type is specified as *Part* to show that what come into the system are parts. This could be *Patients* in a healthcare system or *Customers* in banking or other business systems. Note that once the *Create Module* is selected, Arena displays the alternative view in the spreadsheet view for the selected module as shown in figure 5.19.

Figure 5.19: The Create module spreadsheet view

In a similar way, add a process module to your create module. Arena should automatically connect these two modules for you if you have your auto-connect option on. Double-click the module and update its data as shown in figure 5.20. Refer to section 5.6.2 for a detailed treatment of the process module the different ways in which it may be used.

Finally, add a *Dispose Module* to your model and double-click on it to open its dialog box as shown in figure 5.21. Ensure that the *Record Entity Statistics* box is checked so that Arena collects statistics on the entities before they are disposed of.

The completed model should now look as shown in figure 5.22.
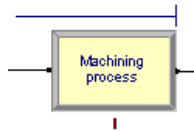
Figure 5.20: The Process module dialog box

Figure 5.21: Dispose module dialog box
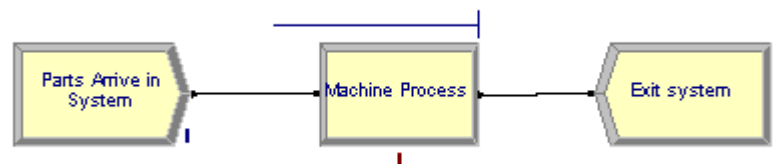
Figure 6.22: The completed model

5.7.2 Before running the model

Before running the model, we need to set the run conditions. That is to tell Arena how long to run for, what kinds of statistics to collect and what king of report to generate etc. This is done in the run setup dialog by selecting setup from the run menu.

There are five (5) tabs in this dialog thus, *Reports, Run Control, Run Speed, Project Parameters and Replication Parameters.* At this stage we will only briefly look at two of the tabs, *Replication Parameters and Project Parameters.*
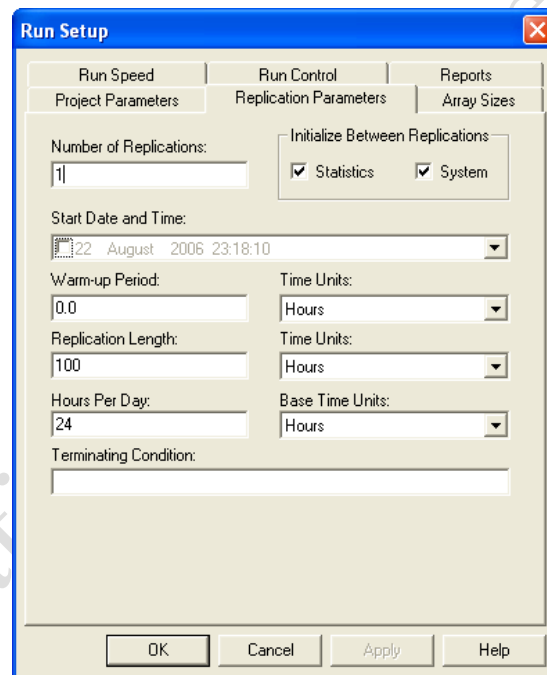
Figure 5.23: Run Setup dialog with Replication Parameters tab displayed

The dialog is shown in figure 5.23 with the *Replication Parameters* tab displayed. The first item to the top left of the display if "Number of Replications". This is the Number of simulation runs to execute. For example, if your model runs for 100 hours and the "Number of Replications" is set to 10, then Arena will execute your 100 hour run over and over again for 10 times. This helps generate sufficient data for statistically valid analysis. This value must always be an integer greater than or equal to 1.

"Initialize Between Replications" refers to whether or not Arena should empty system and statistics and start afresh after each replication. If the statistics option is checked, then Arena will empty all the statistical accumulators after each replication and start collecting fresh statistics. Similarly, if the system option is checked, Arena empties the system, getting rid of all entities and starting again each time. When modelling a banking system where each replication is for example 24 hours long, then it may be a good option to initialize the system since there will not be customers in the bank before the start of any working day.

The "Date and Time" field is basically for associating a specific calendar date and time to with the simulation start time of zero. If this field is not specified, Arena will start from midnight of the current date. For example, if the current date and time on the computer clock is "Feb 10, 2015 08:45:32", then the *Start Date and Time* will be automatically set to "Feb 10, 2015 00:00:00".

When the system being modelled is continuous, it would be useful to specify a "Warm-Up Period". This is the time period after the beginning of the run at which statistics are to be cleared. This value should be a real value greater than or equal to 0.0 time units. If the warm-up period is larger than the replication length, the warm-up time will be ignored and no statistics will be cleared.

The "Replication Length" is simply how long a simulation run should last and is the time used to evaluate the system. This value may be a real value greater or equal to 0.0. If no value is specified, the simulation model will run infinitely unless stopped by some other means. Other methods of stopping a simulation run are by specifying the maximum batches on a Create type module, specifying a terminating condition (as described below) or defining a limit on a counter, as specified in a Statistic module or Counters element.

The **"**Hours Per Day" field refers to the number of hours the model runs in each day. This value depends upon the number of hours the real system operates in a day. The default value for this field is 24 hours per day but can be any expression greater than 0. Note that the number of hours per day specified will affect the number of slots shown on the graphical schedule editor for any resource, arrival or other schedules. This field is useful to exclude a part of the day from statistics when your entire facility

shuts down for part of each day. For example, if your facility works only 2 shifts (16 hours), if you leave hours per day at its default of 24, all of the statistics will be based on 24 hours even though activity only occurs during 16 hours. Hence, the average utilization for a fully utilised resource is $16/24 = 67\%$. If you specify hours per day at 16 hours, that same statistic would report as $16/16 = 100\%$ utilised.

"Terminating Condition" defines a particular condition for stopping the simulation. This specification of an expression or condition is evaluated throughout the simulation run and brings the simulation process to a stop as soon as the condition is met. This is one method, besides specifying a replication length, for terminating the simulation.

Arena needs to use a uniform unit for all time values collected in the simulation. This is done with the setting of the "Base Time Units" field. This is the time units for reporting, status bar, simulation time (TNOW) and animated plots. All time delays, replication length, and warm-up period times will be converted to this base time unit.

"Time Units" defines the units of time used for the warm-up period and replication length. These are used to convert the warm-up period and simulation run length to the base time unit specified.

Now set your "Replication Length" field to 100 as shown in figure 5.24 and leave the rest at their default settings.
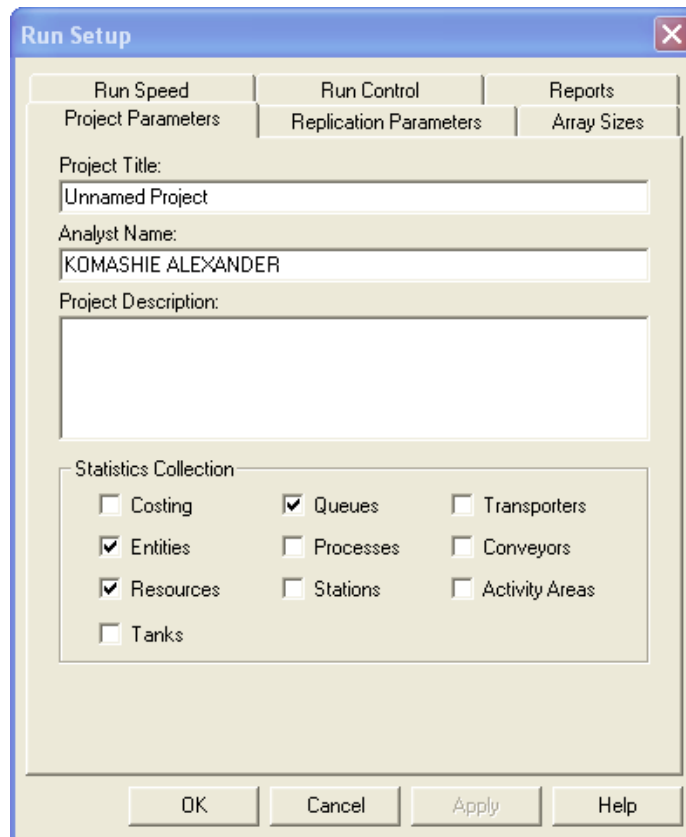
Figure 5.24: Run Setup dialog with Project Parameters tab displayed

The other tab we will look at is the *Project Parameters* tab. This tab provides general information about the simulation project such as "Project Title", "Analyst Name" and "Project description" as shown in figure 6.24. Additionally, it also enables you to choose which types of statistics may be collected. As shown, the entities, resources, and queues boxes have been checked hence Arena would only collect statistics on these objects during the simulation.

5.7.3 Running the model

Running the models in Arena is rather easy. This is done by clicking the Go button on the Standard toolbar as shown in figure 5.25. Alternatively, you may run the model by clicking Go in the Run menu or by pressing the F5 function key on the key board. The option *Check Model* in the Run menu, the F4 key or the check (✓) sign on the *Run Interaction* toolbar may be used to check the model for errors before running. However, if you begin to run the model without checking for errors, Arena will automatically do the checking before running the model. If there is an error in your

model, Arena will give a message with some possible reasons to help you fix the error. At times you may find it necessary to speedup the run. This can be done by using the Fast (Fast Forward) button as shown. Note however that when running in fast mode there will be no animations.
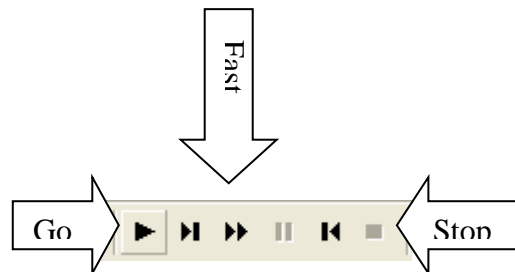
Figure 5.25: The run, fast and stop buttons

Once the model is without errors, it will begin to run and you can watch the entities moving from module to module as shown in figure 5.26. Notice that each of the modules has an animated counter. That to the right end of the Create module keeps track of the number of entities leaving that module. The counter below the process module keeps track of the number of entities in process at this module and the counter to the right end of the dispose module keeps track of how many entities have left the system through this module. Arena uses all of these variables to calculate its statistics.

Figure 5.26: The running model

You can choose to stop the run at any time using the Stop button shown in figure 5.25. If you do not stop the run, Arena will continue forever unless you have a terminating condition specified in one way or another. In our case we specified only one replication with length of 100 hours so the simulation will surely stop after 100 hours and by default, Arena will display the dialog shown in figure 5.27.
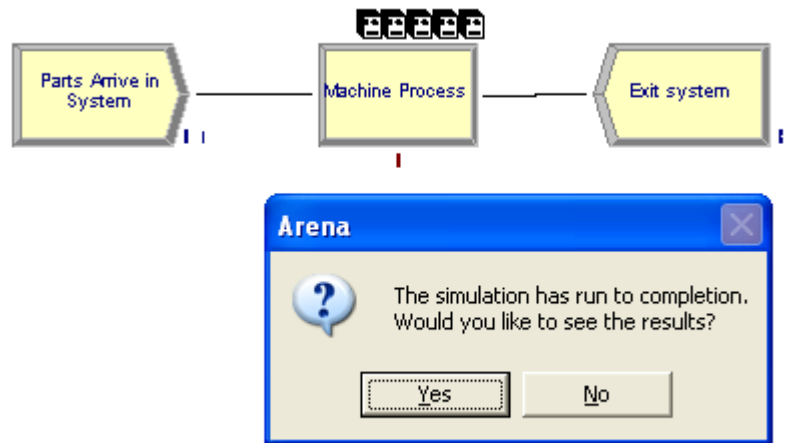
Figure 5.27: End of simulation run

5.7.4 Viewing the results

The dialog in figure 5.27 gives you the option to view the model report (this could be changed in the setup to display the report without prompting). Clicking yes will display either the view in figure 5.28 or 5.29 depending on which report type you have selected to display in run setup dialog. Let us now have a look at some parts of Arena's reports.

Figure 5.28: Summary report from simulation run

If you've been building the model along with us then just click on the "yes" button on the dialog to open the report. If the default report type has not been changed then Arena will display the "Category Overview" report as shown in figure 5.29. To change the report type, go back to the model, click on the "Run" menu and select Setup. Click on the Reports tab and then pull down the "Default Report" field. The second in the list (Category Overview) and the last (SIMAN Summary Report (.out file)) are the ones we are considering here.

The summary report is normally divided into different categories (e.g. tallies, discrete-change variables, counters and outputs), each one providing a specific type of statistic.

"Tally Variables" display the tallies recorded in your model. Tally statistics include entity and process costs and times.

"Discrete-Change Variables" include any statistic in the model that is time-weighted. (Time-weighted statistics "weight" the value of the variable by the amount

of time it remained at that value.) Included in this category are *Resource Number Busy*, *Number Scheduled* and *Utilisation* as well as *Number in Queue* statistics. These are also referred to as Time Persistent Statistics.

The **"**Outputs" section displays statistics for the final value of a given variable the model. Included in this category are costs of resource, total process costs and times and work in process information.

The "Counters" section displays statistics for any counters identified in your model. The number of entities into and out of the system is included in this category.

Note that there may be more or less categories of statistics depending on the types defined in your model.

The "Half Width" column shown in the report is the 95% Confidence Interval range around the average. This is included to help you determine the reliability of the results from your replication. This column may either be a value (real number), said to be "Insufficient" or "Correlated".

"Insufficient" means that there is insufficient data to accurately calculate the half width of the variable. This is because the formula used to calculate half width requires the samples to be normally distributed. That assumption may be violated if there is a small number (fewer than 320) of samples are recorded in the category. Running the simulation for a longer period of time should correct this.

"Correlated" also means that the data collected for the variable are not independently distributed. The formula used to calculate half width also requires the samples to be independently distributed. Data that is correlated (the value of one observation strongly influences the value of the next observation) results in an invalid confidence interval calculation. Running the simulation for a longer period of time should correct this as well.

If a value is returned in the Half Width category, this value may be interpreted by saying "in 95% of repeated trials, the sample mean would be reported as within the interval sample mean ± half width". The half width can be reduced by running the simulation for a longer period of time.
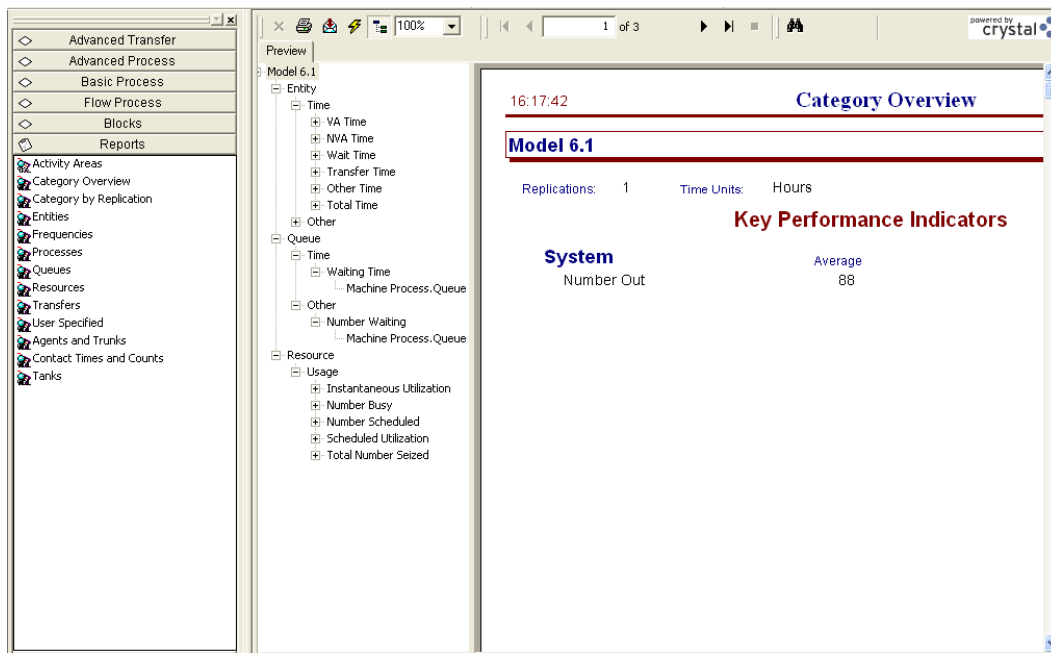
Figure 5.29: Category overview report from simulation run

The "Category Overview" report has more detailed information than the "Summary Report". As shown in figure 5.29, this displays a summary of the key performance indicators on the first page. It is however organized into the following sections: Key Performance Indicators, Activity Area, Conveyor, Entity, Process, Queue, Resource, Transporters, Station, Tank, and user specified. The statistics reported are a summary of values across all replications.

The information displayed in this report varies based on the number of replications executed and the type of statistic you decide to collect. It may be observed in the report tree structure that Arena makes available statistics only on Entities, Queues and Resources. This is because those are the only objects we specified for statistics collection as shown in the Run Setup dialog in figure 5.24. Notice also that Arena however displays all the various report categories in the report panel in the project bar.

To view any item in the report, you only need to click on the item in the Reports Panel or select the item in the reports tree structure. For example to view statistics on the entities in the model we clicked on the "Entity" item in the tree structure and it displayed statistics shown in figure 5.30 below.
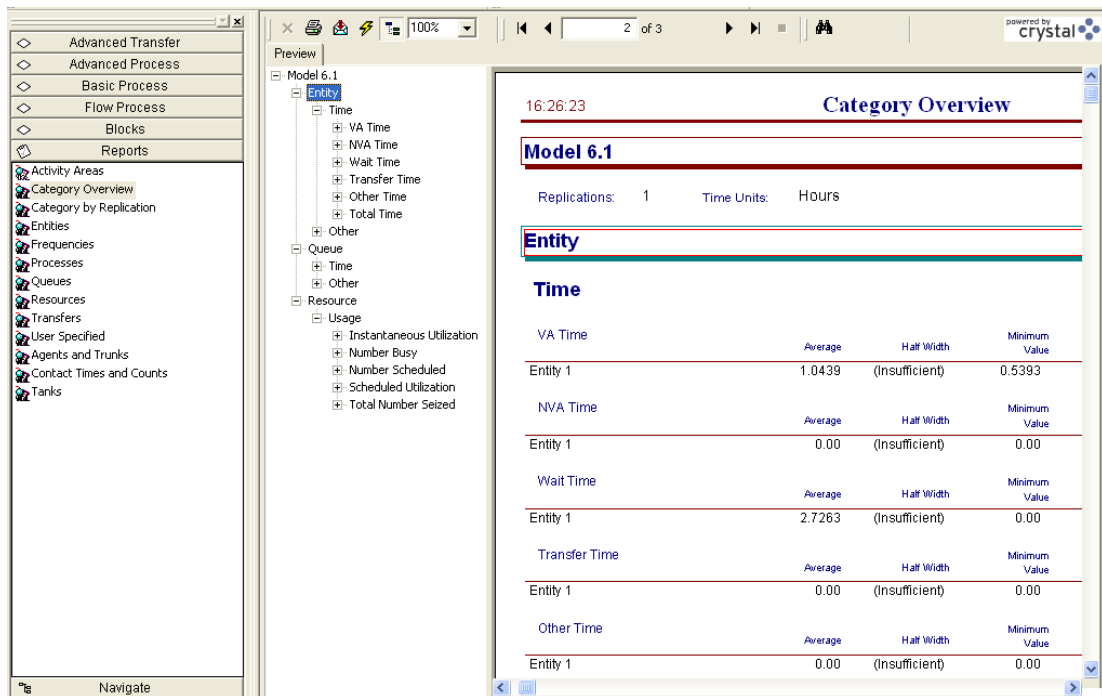
Figure 6.30: Category overview report showing Entity statistics

The explanations for the various columns of the report are the same as presented under the summary reports. Thus Half Width column for example may either have a value (real number), said to "Insufficient" or "Correlated" as explained above.

This chapter was mainly to introduce you to the fundamentals of building a simulation model using the Arena simulation software. The material covered here includes an introduction to the Arena software and its hierarchical structure, a tour of the Arena environment and some basic concepts and terminologies. We also explained that the building blocks in every Arena model are flowchart modules and data modules. The three basic flowchart modules in Arena were discussed in detail to prepare you for the simple modelling problem that followed. Finally in section 6.7 we looked at the basics of modelling in Arena by considering step by step, a simple three module system.

In the next chapter, we will continue to present detailed description of all the modules found in Arena's "Basic Process Panel". This will help you understand the various uses of the modules and to help you solve future modelling problems in Arena more easily.

CHAPTER 6

Simulation and Modelling Using Arena, the Basic Process Panel

This chapter covers:

1. The flowchart and data modules in the basic process panel.

2. Modelling basic systems

6.1   Introduction

In chapter 5, we introduced you to the Arena simulation software and took you through the fundamentals of modelling using this software.

For quite a while now I have trained students in the use of discrete event simulation using Arena. Our observation is that Arena is indeed a powerful and flexible tool, but students usually find it difficult to grasp the fullness of its power and to be able to use the tool for solving problems. This is the motivation for writing this chapter on the module by module approach to learning simulation with Arena.

It is anticipated that apart from teaching the student all possible uses of all modules in the *Basic Process* panel, this will also serve as a quick reference for students in solving problems that require the use of some of these modules.

Note that the *Create, Dispose and Process Modules* have been discussed in chapter 6 and are therefore not included in this chapter even though they are part of the *Basic Process* panel.

6.2   The Basic Process Panel

The *Basic Process* panel contains the most common modelling constructs that are very accessible, easy to use and with reasonable flexibility. This panel contains eight (8) flowchart modules and six (6) data modules. These modules are shown in table 7.1 below.

Table 6.1: Basic Process Panel Modules

	Flowchart Modules	Data Modules
1	Create	Entity
2	Dispose	Queue
3	Process	Resources
4	Decide	Variable
5	Batch	Schedule
6	Separate	Set
7	Assign	
8	Record	

6.2.1 Decide Module

The *Decide Module* allows for decision-making processes in the system. It includes options to make decisions based on one or more conditions (e.g., if entity type is Gold Card) or based on one or more probabilities (e.g., 75% true; 25% false). Conditions can be based on attribute values (e.g., Priority), variable values (e.g., Number Denied), the entity type, or an expression (e.g., NQ (ProcessA.Queue)). The module shape and dialog are shown in figure 6.1.
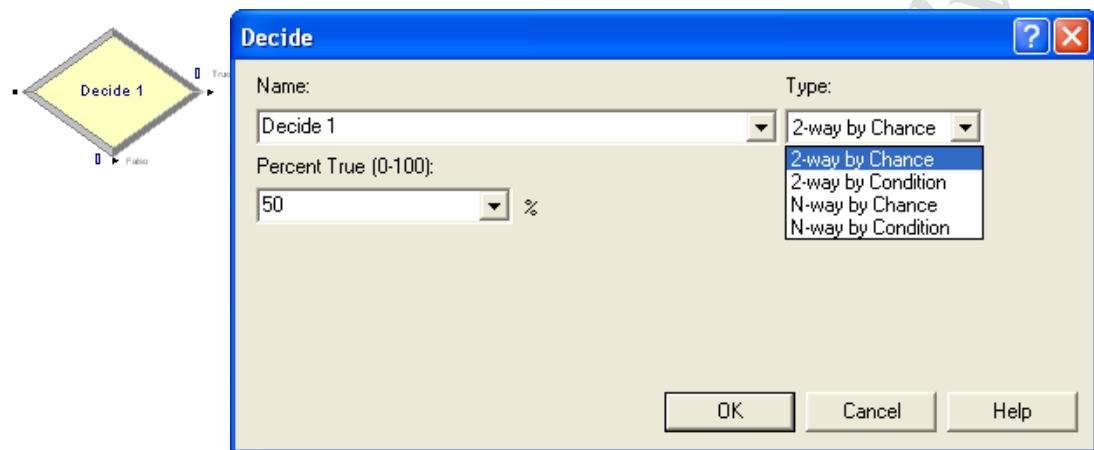
Figure 6.1: Decide Module shape and dialog

Note that the *Name* field in this module dialog serves the same purpose as we have already described in previous modules. Arena provides four (4) options in the *Type* field as shown. Thus the decision types possible with this module are *2-way by Chance, 2-way by Condition, N-way by Chance and N-way by Condition.* Before we look at these options in detail, it should be noted that the *Decide Module* has basically two exit points. The first which is to the right end of the module shape is the "True" exit whilst the second which is at the bottom end of the module shape is the "False" or "Else" exit. These are the only exits available when *Type* is either *2-way by Chance* or *2-way by Condition*. When *Type* is *N-way by Chance* or *N-way by Condition*, the will always be a number of a number of "True" exit points equal to the number of chance values or conditions specified. All of these will be lined up vertically at the right end of the module shape as shown in figure 6.2, but there will only be one "False" or "Else" exit which will always be the exit at the bottom end of the module shape. We

will now look at each of these options in detail to understand how and when to apply them.

The *2-way by Chance Type* is the basic and default option for Arena. This has the dialog view shown in figure 6.1. An example of this is to say that 50% of all entities that enter this module require inspection whilst the remaining 50% don't. You specify this by assigning the value 50 to the *Percent True* field as shown and this will tell Arena to send 50% of all entities that come into the module through the "True" exit and everything else goes out through the "False" or "Else" exit. Note that the *Percent True* value can be anything from 0 to 100.

The Type *N-way by Chance* is similar to the above except that you have to means of specifying more than just one chance or probability. When this is selected, the dialog view changes to that shown in figure 6.2. Clicking on the "Add" button on the *Decide* dialog displays the *Conditions* dialog in which the chance or probability value can be specified. Notice that there are three (3) exit points to the right of the module shape equal to the number of percentages or conditions specified. Thus counting from top, 10% of entities will leave through the first exit, 50% through the second exit, 15% through the third exit and 100 minus 75 (i.e. 10 +50 +15) will leave through the "False" or "Else" exit at the bottom of the module shape.
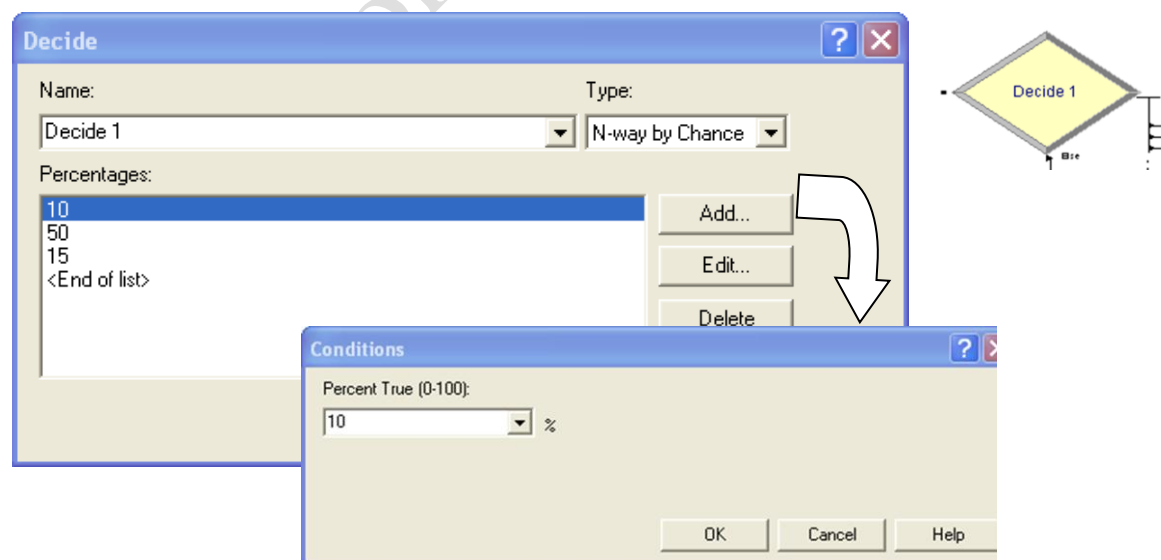
Figure 6.2: Decide Module shape and dialog with N-way by Chance

When the decide *Type* is *2-way by Condition*, the resulting dialog view is as shown in figure 6.3. In the *If* field, Arena displays a list of items with which a condition can be created. The list includes Variables, Attribute, Entity Type and Expression. When you select Entity Type in the *If* field, Arena would make all entity types defined in your model available in the *Name* field list for you to select from as shown in figure 7.3. If you select "Product A" then Arena will use the condition that "If Entity Type is named "Product A" then send it through the "True" exit, else send it through the "False" or "Else" exit.
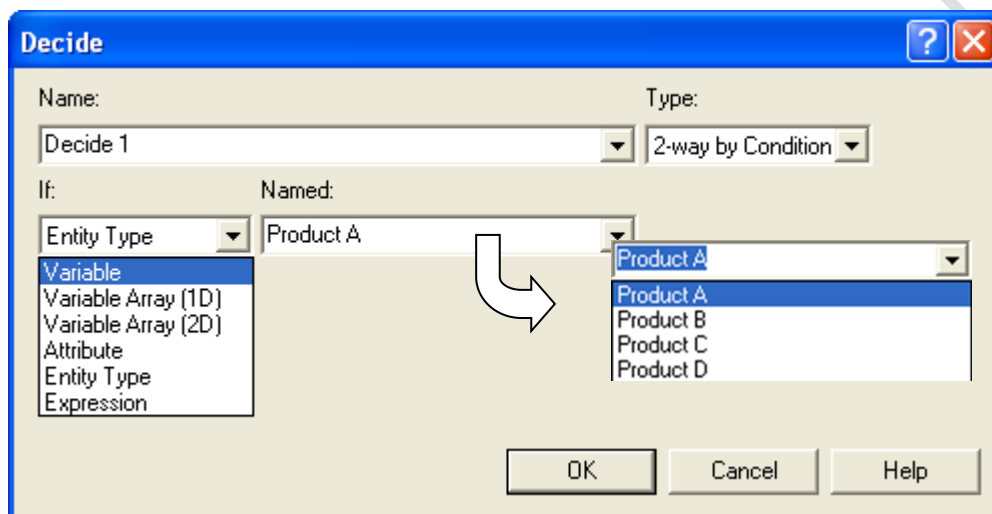
Figure 6.3: Decide Module dialog with 2-way by Condition

Selecting Variable in the *If* field, changes the dialog's view since additional parameters have to be specified. This view is shown in figure 6.4. This is exactly the same as when you select *Attribute* in the same field. Arena makes all variables or attributes defined in your model available in the *Named* field list for you to select from. There is also a means of specifying an evaluator (i.e. >, <, =, etc). The *Value* field requires an expression that will be either compared to the attribute or variable.
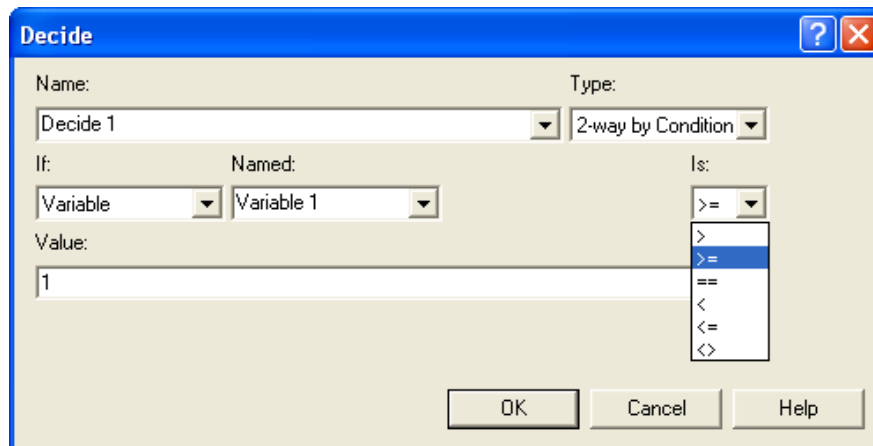
Figure 6.4: Decide Module dialog with 2-way by Condition – Variable view

When the condition is based on a *Variable Array (1D)*, a *Row* field is added as in figure 7.5. 1D means a one dimensional array which may be specified as *Variable1 (10)*, where "variable1" is the name of the variable and "10" is the array size. An example of this is having 10 different components in your system and wanting to keep track of the number of each component that has entered a process. One approach will be to define 10 different variables for each component type (i.e. from 1 through 10). An easier approach will be to use a one dimensional array (see the section on the Variable data module to learn how to define variables and arrays in Arena). We may define our variable array as "NumberOfCompents (10)". With this Arena will create 10 separate storage places (as 10 separate rows) for the number of components of each of the 10 components in your system. To use this in your model, you may use the following assign statements;

NumberOfComponents(ComponentType) = NumberOfComponents(ComponentType) + 1

Where ComponentType is a predefined attribute of the entities ranging from 1 through 10 depending of what type of component the entity represents (i.e. ComponentType = 1 for component 1, 2, for component 2 , 3 for component 3 etc) . When an entity arrives at the above statement, Arena will check the value of its ComponentType attribute and substitute that value in the above statement. For example if the ComponentType is 5, the expression becomes:

NumberOfComponents(5) = NumberOfComponents(5) + 1

Arena will thus check row number 5 of the "NumberOfComponents" array, add 1 to that value and use the result as the new value for that same row in the array. The same steps would be carried out for any value of "ComponentType" hence with one statement you can update the number of components for each component type irrespective of which component arrives at which time.

Note however that assignments as above are done in the Assign Module. We will talk about in section 6.2.7.

Let's now look at another example of using one dimensional variable array in a Decide Module as in figure 6.5. We have selected the one dimensional array named "NumberOfComponents" and row number equal to "ComponentType". Our desired evaluator is "greater than or equal to" (>=) and the test value is 300. This will instruct Arena to check if the value of the one dimensional array named NumberOfComponents at row number "ComponentType" is greater than or equal to 300. If this condition is true, then Arena will send the entity out through the "True" exit otherwise it is sent through the "False" or "Else" exit. Remember that "ComponentType" can be any integer from 1 through 10. What will happen in this module during the simulation run is that, an entity would be allowed to proceed in one direction (through the "True" exit) so long as the number of its component type is less than 300. As soon as it riches 300, Arena redirects all following entities of that type to a different exit ("False" or "Else" exit possibly to a different process).
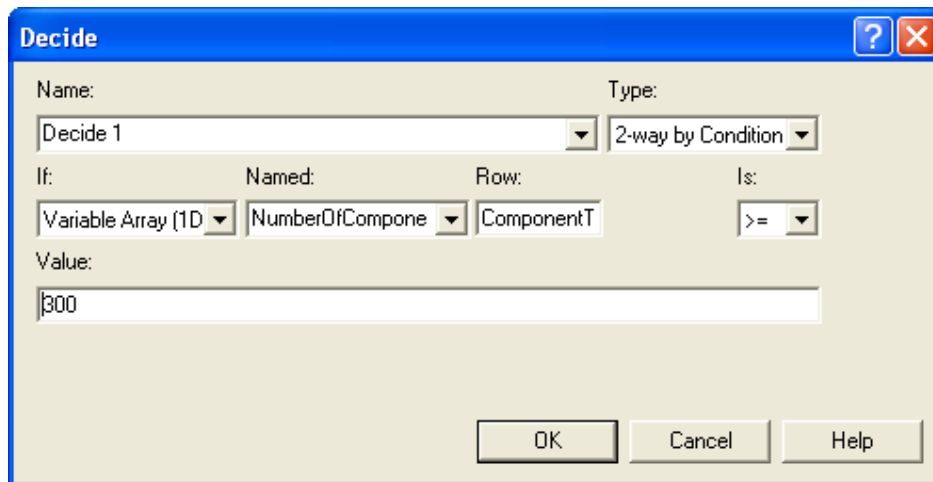
Figure 6.5: Decide Module dialog with 2-way by Condition – Variable Array (1D) view

The *Variable Array (2D)* option is very similar to *Variable Array (1D)*. The only difference is that it has an extra dimension to its array definition as shown in figure 7.6. Whilst *Variable Array (1D)* has only rows, *Variable Array (2D)* has rows and columns. These are defined as "Variable1 (Row, Column)" or "Variable1 (1, 1) meaning row 1 column 1 of the Variable1 two dimensional array.

To illustrate the use of this, consider figure 6.7. Assume a system that receives orders from customers with each order involving 10 different component types. In other to keep track of how many components of each type is in each order you may want to use the 2D variable array as shown. If there are for example 20 orders, then the variable array may be defined as:

NumberOfComponents (20, 10)

The values of the array may be assessed by using dynamic arguments like:

NumberOfComponents (OrderNumber, ComponentType)
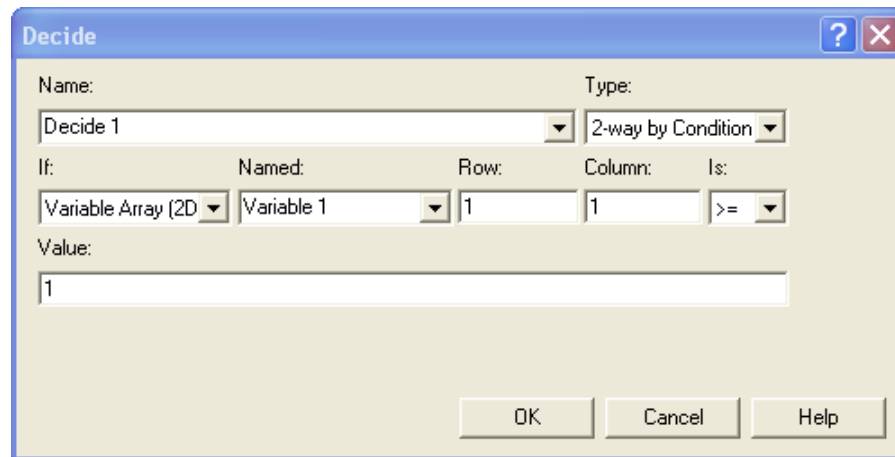
Figure 6.6: Decide Module dialog with 2-way by Condition – Variable Array (2D) view

Notice again that "OrderNumber" and "ComponentType" have to be pre-assigned attributes and both must have integer values.

Therefore, the condition specified in figure 7.7 requires checking if the value of the two dimensional array named NumberOfComponents at row number "OrderNumber" and column number "ComponentType" is greater than or equal to 300.
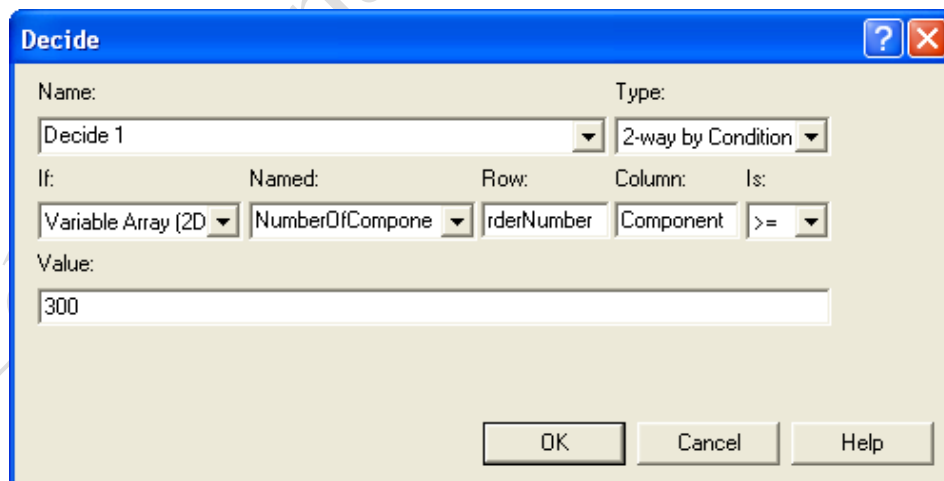
Figure 6.7: Decide Module dialog with 2-way by Condition – Variable Array (2D) example

When the condition is *Expression*, Arena displays the view in figure 7.8. With this, the value must also include the evaluator (e.g., Colour<>Red, <> means not equal to) and that will be evaluated as a single expression to determine if it is true or false.
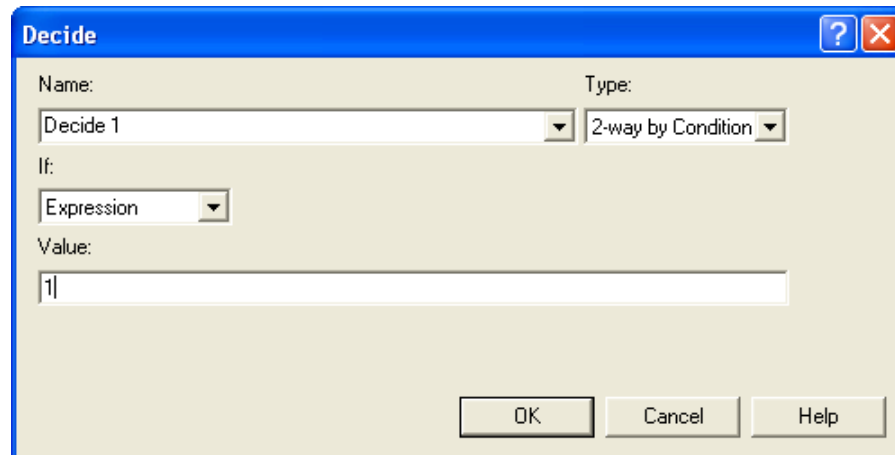
Figure 6.8: Decide Module dialog with 2-way by Condition – Expression view

The *N-way by Condition* option can be treated in the same way as we did the *N-way by Chance.* As shown in figure 6.9, the only difference is that we are now using multiple conditions instead of the multiple probabilities we used in *N-way by Chance.*

Notice here again that the number of exit points as shown on the module shape corresponds to the number of conditions specified in the dialog. The *If* field could also be any of the items shown in the list in figure 6.3. You can specify as many conditions as necessary by clicking on the "Add" button to display the "Conditions" dialog.
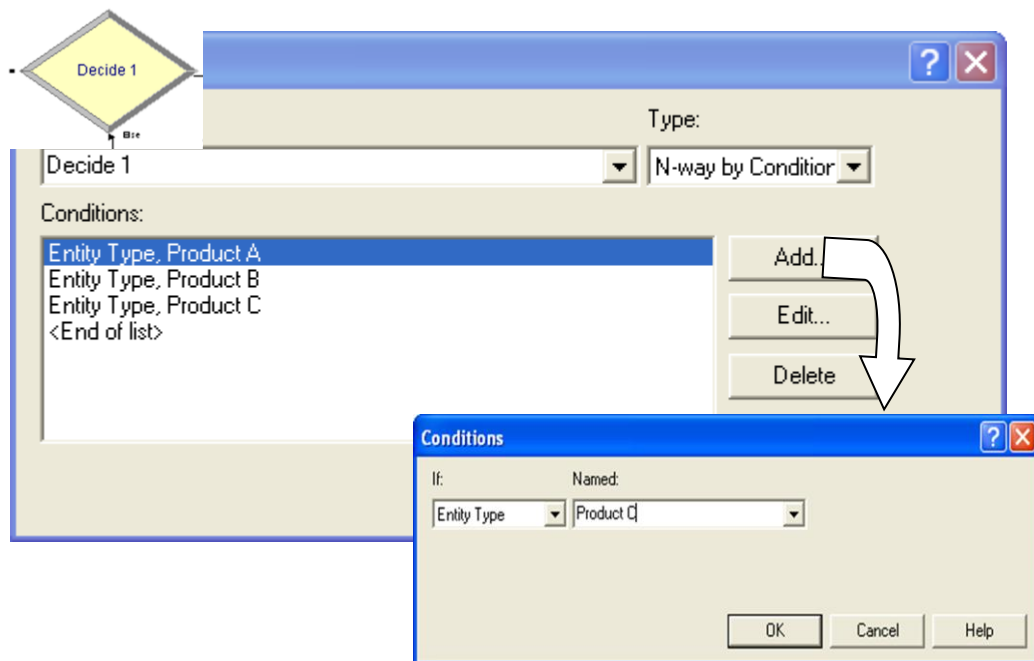
Figure 6.9: Decide Module dialog with N-way by Condition

6.2.2 Batch Module

The *Batch Module* is used for grouping or batching entities within the simulation model. Entities can be permanently or temporarily grouped in the simulation. Temporary batches must later be split using the *Separate* module (Section 6.2.3). Figure 6.10 shows the *Batch Module* shape and dialog box.
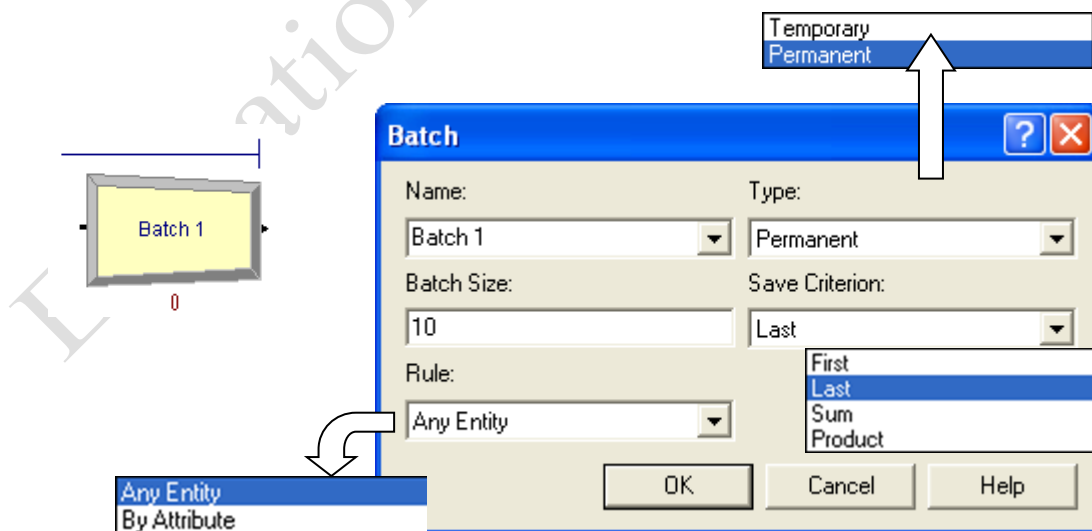
Figure 6.10: Batch Module shape and dialog

Arena requires the same parameters whether batches are permanent or temporary. Batches may be formed with any specified number of entering entities or may be matched together based on an attribute. When the batching *Rule* is by Attribute, the dialog view changes to what is shown in figure 6.11. Arena then adds an "Attributes" filed with a drop down list from which you can select. Entities arriving at the Batch module are placed in a queue until the required number of entities has accumulated. Once accumulated, a new representative entity is created. The batch size may be an integer or any expressions that evaluates to an integer.

The *Save Criterion* field is a method for assigning representative entity's user-defined attribute values.
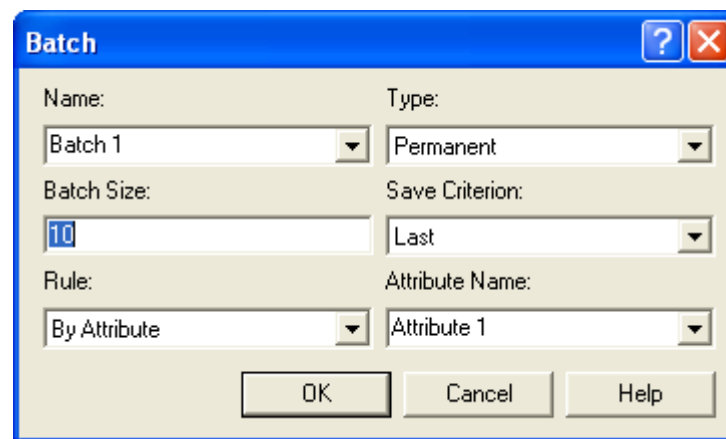
Figure 6.11: Batch Module dialog batch By Attribute Rule

6.2.3 Separate Module

This module can be used in only two ways. That is to either duplicate an incoming entity into multiple entities or to split a previously batched entity. The module shape and dialogue is as shown in figure 6.12. When "Type" is "Duplicate Original", Arena allows you to make duplicate or no copies of the incoming entity. That is no duplicate copies will be created when "# of Duplicates" is less than or equal to zero. Note otherwise that the total number of entities exiting the module will always be "# of Duplicates" plus one. The "Percent Cost to Duplicate (0-100)" field is best explained with the following example from the Arena help file.

If Cost to Duplicates is 50 and # of Duplicates is 2, 25% of the incoming entity's cost and time values will be allocated to each of the duplicates and the remaining 50% of the incoming entity's costs and times will be allocated to the original. Similarly, if the Cost to Duplicates is 90 and the # of Duplicates is 3, each duplicate will be allocated 30% of the incoming entities costs and times, while the remaining 10% will be retained by the original incoming entity.

When "Type" is "Split Existing Batch", the temporary representative entity that was formed is disposed and the original entities that formed the group are recovered. Note that to use this option, you might have previously created a temporary batch using the *Batch* module (section 6.1.2) else Arena will find nothing to split. The entities after splitting proceed sequentially from the module in the same order in which they originally were added to the batch. With the split batch option, Arena provides the "Member Attributes" field as shown in figure 6.12 (b). This field determines what attribute values of the representative entity should be passed on to the original entities after the batch is split. The three options as shown are to retain the original entity values, thus before they were batched, to retain all the values of the representative entity and to retain selected values. For the third option, Arena provides extra dialogs as shown in figure 6.12 (c) to enable you selected the specific attributes.
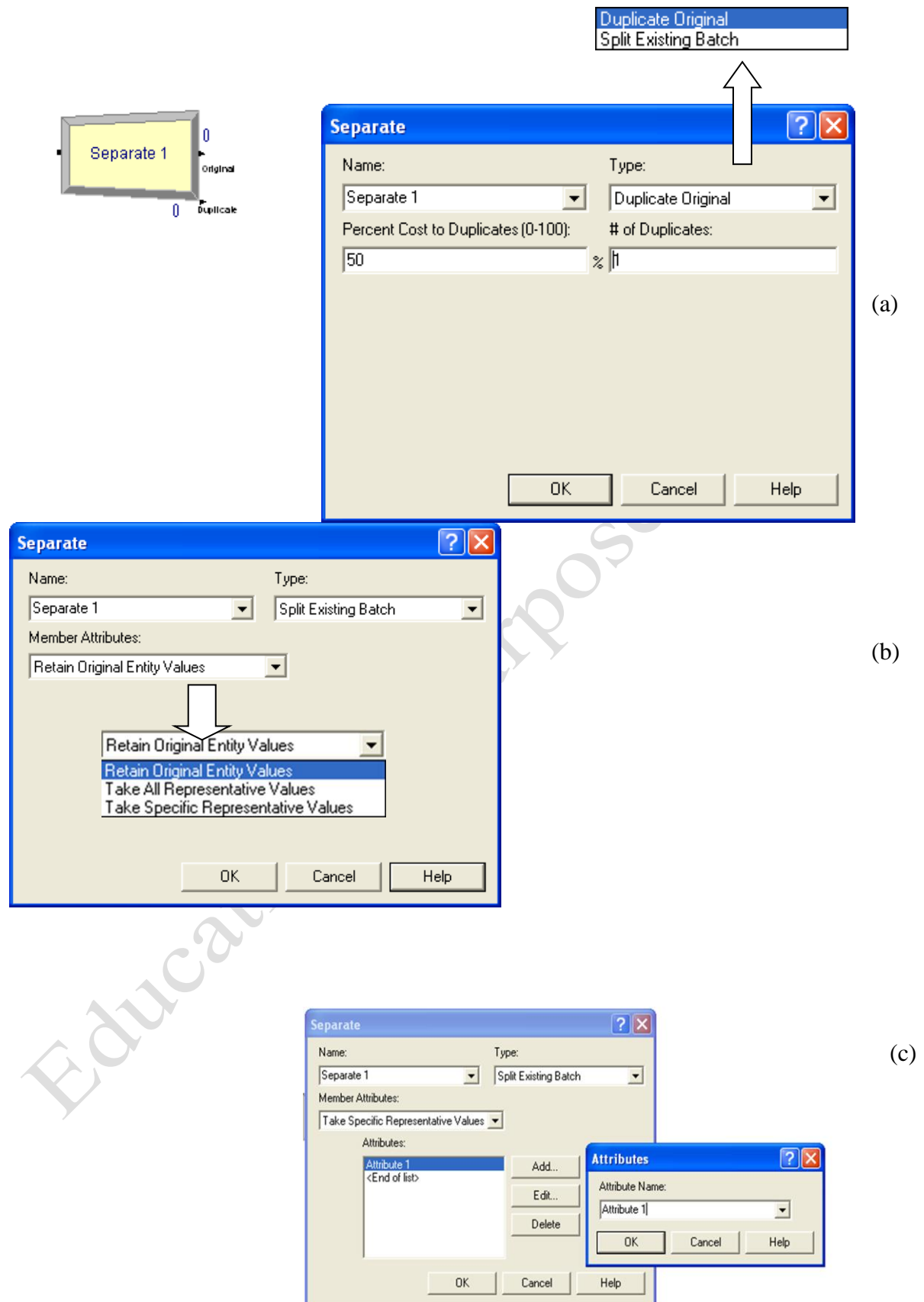
Figure 6.12: Separate Module shape and dialog

6.2.4 Assign Module

The purpose of this module is to assign new values to variables, entity attributes, entity types, entity pictures, or other system variables. Multiple assignments can be made with a single *Assign* module. This module can be used anywhere in the model where it is required define or reassign a new value to variables or attributes.

Figure 6.13 shows the module shape and dialog. Clicking on the "Add" button displays the "Assignments" dialog in which the values of the attributes or variables may be assigned. Notice that the "Type" field of the assignment dialog provides a list of all the possible assignments that can be made with the module. You may want to experiment with all of these types to find out how to use them. As in other modules, the "New Value" field can be a constant value or an expression.

Note that if there are multiple assignments, Arena performs the assignments in the order in which they appear in the list hence. This is important when assigning a value to a variable and using that variable in an expression within the same *Assign* module.
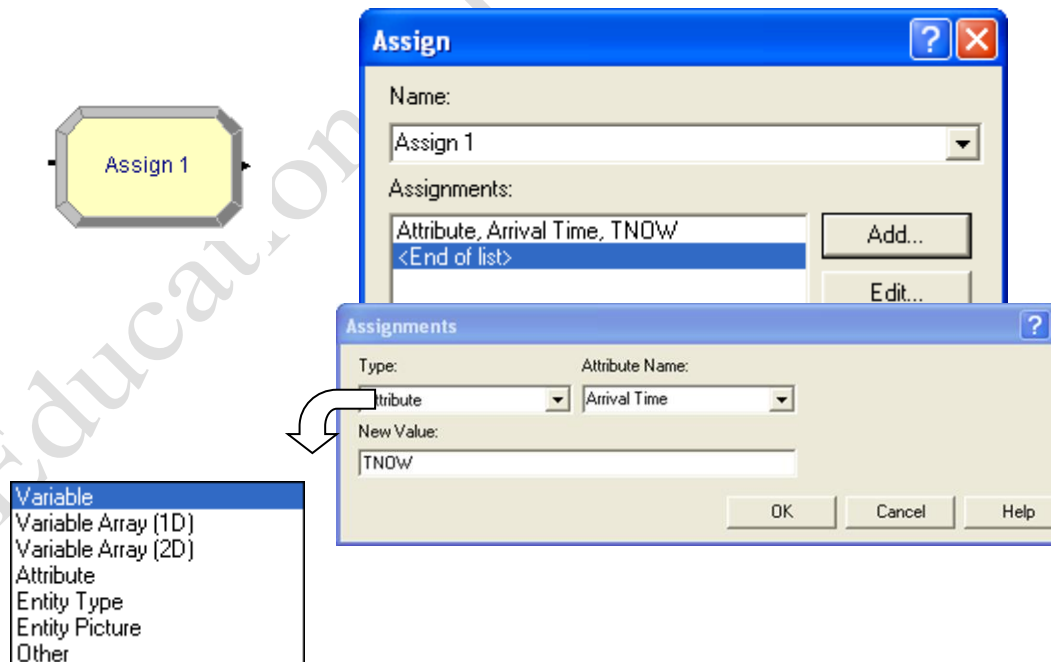
Figure 6.13: Assign Module shape and dialog

6.2.5 Record Module

The *Record* module is used to collect statistics in the simulation model. The module shape and dialog are shown in figure 6.14. The main part of the dialog is the "Type" field. As shown, there are five types of records; count, entity statistics, time interval, time between and expression. We will briefly explain these record types.

Count: use this type when you only need to count the number of entities going through the record module. Arena will increase this statistics by the value specified in the "Value" field each time an entity enters the module. A negative value will decrease the count. This can therefore be used to count the number of entities leaving a process or going into a process.

Entity Statistics: these statistics include VA Cost, NVA Cost, Wait Cost, Transfer Cost, Other Cost, Total Cost, VA Time, NVA Time, Wait Time, Transfer Time, Other Time and Total Time. Using this record type will keep record these statistics each time an entity enters the module.

Time Interval: this type helps record the time spent by an entity in a process or in the system. When using *Time Interval* to collect interval statistics, an attribute is required to hold the value of the start time of the required interval. For example to determine the time spent by an entity in a process, set the attribute to the current simulation time, TNOW (using the assign module), before the entity enters the process and then record the time interval after the process using the attributes value as the start time of the interval. What Arena does is to subtract the attributes value from the current simulation time at which the entity enters the record module.

Time Between: this option is used to track and record the time between entities entering the record module. An example of this would be to track the rate at which parts are entering into a process by putting the record module with type *Time Between* just before the process.

Expression: this is generally used to record the value of a specified expression. Each time an entity enters the record module, the expression would be evaluated and the result recorded for the entity.

You should realise that depending on which type you select, Arena provides more or less fields for you to specify the corresponding parameters. It would be good to try experimenting with all of these types to become familiar with their parameters.
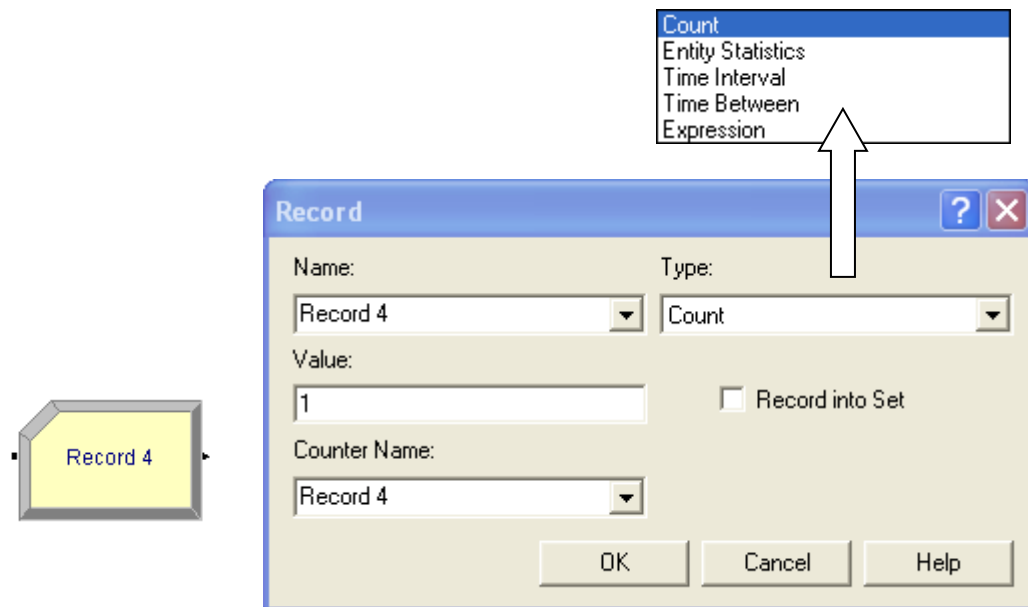
Figure 6.14: Record Module shape and dialog

6.2.6 Entity Module

The *Entity Module* is a data module. It is used to define the various types of entity in the model and their initial picture and cost and time values. The module and its parameters are shown in figure 6.15. When the *Entity Module* is selected by clicking on the icon shown, Arena typically displays all *Entities* that have been defined in the model and the initial values of every parameter. For example there are four *Entities* (Entity 1, Entity 2, Entity 3 and Entity 4) defined in the model from which figure 6.15 was taken Arena has a default list of *Entity* pictures that is displayed each time you click on the any row in the "Initial Picture" column. Initial costing information and holding costs are also defined for the entities. Each time you create an entity using the Create Module, Arena automatically updates this module by adding the last entity created to the list but you can also directly create a new entity here by double clicking the space just below the last row.
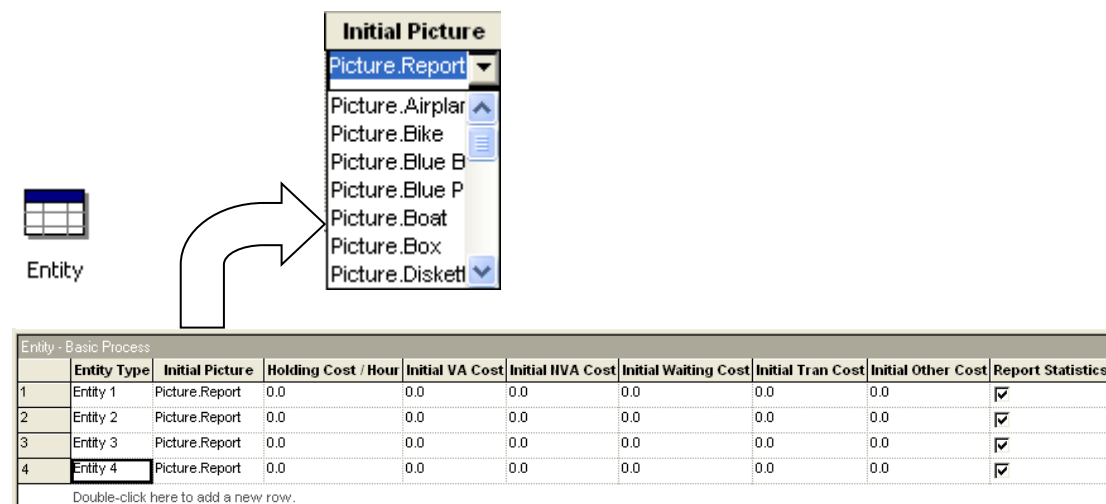
Figure 6.15: Entity Data Module Spreadsheet view

6.2.7 Queue Module

This is also a data module. It is used to define all *Queues* in the model and to change the ranking rule for members of a specified queue. The default ranking rule for all queues is First-In-First-Out (FIFO) unless otherwise specified in this module. There is an additional field that allows the queue to be defined as shared (not available in Arena Basic Edition). A shared queue is one that may be used in multiple places within the simulation model and can only be used for seizing resources.

You may add new *Queues* to this module by double-clicking below the last row. By default Arena gives the names Queue 1, Queue 2, Queue 3 and so on to the queues. However each time you add a process to your model that requires a resource Arena will automatically add its queue to the *Queue* data module with the name "ProcessName.Queue" (where "ProcessName" is the name of the specific process module). Similarly, if you define a queue in any flowchart module for example in a *Seize Module*, Arena will also add this to the list of queues in your *Queue* data module.

An important part of the *Queue* spreadsheet view shown in figure 6.16 is the "Type" column. As shown, the type of queue may be FIFO, Last-In-First-Out (LIFO), Lowest Attribute Value (LAV) or Highest Attribute Value (HAV).

FIFO and LIFO may be self explanatory. An example of using LAV may be when you are processing about four orders in your system at the same time and would like to give priority to the earlier orders. You may define an *Attribute* called "OrderNumber" which may have values of 1,2,3,4 etc up to the number of orders in your system. Now if you change your "Type" column to LAV, Arena adds another column called "Attribute Name" in which you can select the attribute whose value you want to use for your selection rule. In our case this is called "OrderNumber". Each time an entity enters "Machine Process.Queue" queue, Arena will check its "OrderNumber" attribute and will put those with the smallest value ahead of the queue. In this way you will be able to get the earliest orders through much quicker. Using the HAV "Type" is just the reverse of LAV.
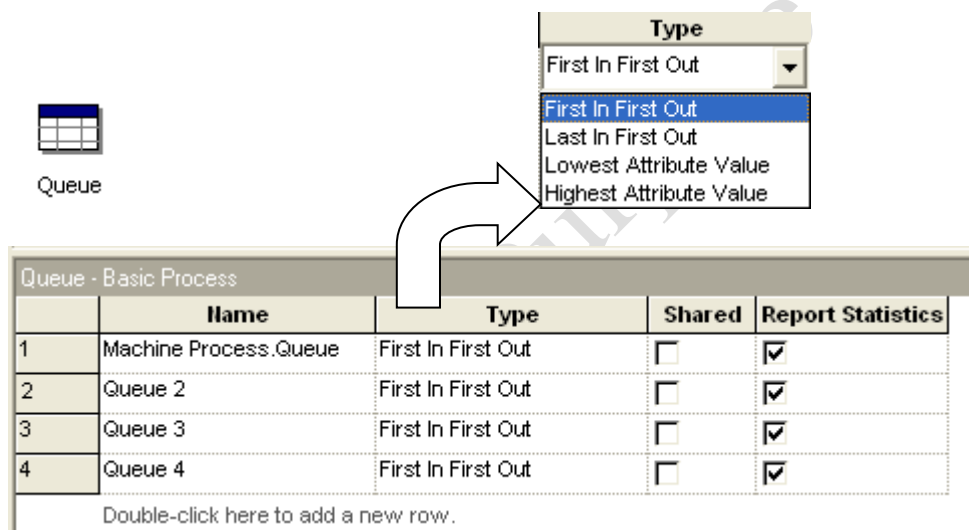
Figure 6.16: The Queue Data Module Spreadsheet view
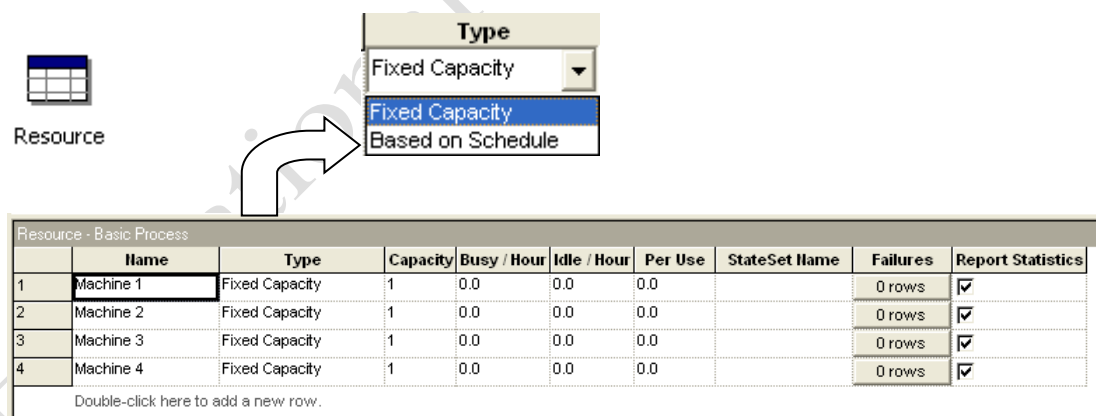
Figure 6.17: The Queue Data Module Spreadsheet view with Type LAV

6.2.8 Resource Module

This data module defines all the resources in the simulation model, and their costing information and availability. All resources may either have a fixed capacity that does not vary over the simulation run or may be based on a schedule defined in the schedule module. All the resource states and failure patterns may also be defined in this module. The *Resource Module* icon and spreadsheet view are shown in figure 6.18.

By default, Arena sets the "Type" column to "Fixed Capacity". This means that the resource will remain at the capacity specified in the "Capacity" column throughout the simulation. There is however a more flexible option as shown in the "Type" drop down menu in figure 6.18. If "Type" is set to "Based on Schedule", Arena adds two new columns for the "Schedule Name" and "Schedule Rule" as shown in figure 6.19. The former is the name of a specific schedule which Arena will apply to the specified resource and the latter is an instruction telling Arena what to do when the resource capacity change (decrease) is about to occur whilst the resource is busy. The three applicable rules *Wait, Ignore*, and *Preempt* as shown.

Figure 6.18: Resource Data Module Spreadsheet view

The *Wait* option will wait until the on-going process is completed and the entities release their units of the resource before starting the actual capacity decrease. Thus if for example a staff is supposed to go on break for 1 hour from 12 to 1pm but a customer arrives at 11:58am, this rule requires that the staff will wait and attend to the

customer completely but still take his or her full break afterwards. That is if the staff finishes with the customer at 12:15pm then his or her break will last until 1:15pm.

With *Ignore,* the resource starts the time duration of the schedule change or failure immediately, but allows the busy resource to finish processing the current entity before effecting the capacity change.

The *Preempt* option interrupts the currently-processing entity, changes the resource capacity and starts the time duration of the schedule change or failure immediately. The resource will resume processing the preempted entity as soon as the resource becomes available (after schedule change/failure).

Figure 6.19: Resource Data Module Spreadsheet view with Schedule

6.2.9 Variable Module

This data module, shown in figure 6.20, is used to define a variable's dimensions and initial values. Values of *Variables* can be referenced in other modules (e.g. the *Decide Module*, section 6.2.1), can be reassigned with the *Assign Module*, and can be used in any expression.

The three methods for manually editing the *Initial Values* of a *Variable Module* are the standard spreadsheet interface, the module dialog and by the two-dimensional spreadsheet interface.

To use the standard spreadsheet interface, first click on the module icon, then in the module spreadsheet, right-click on the *Initial Values* cell and select the *Edit via*

spreadsheet... menu item. If the variable is defined as a two-dimensional array, Arena will automatically provide cells corresponding to the specified numbers of rows and columns.

To use the two-dimensional (2D) spreadsheet interface, just click on the *Initial Values* cell in the module spreadsheet to display the spread sheet view. Note that to see a two-dimensional spreadsheet view, you need to define the number of rows and columns of the variable as shown for variable 4 in figure 6.20 (a).

To use the module dialog, select the module icon as before, then in the module spreadsheet, right-click on any cell and select the *Edit via dialog...* menu item. This displays the dialog view shown in figure 6.20 (b). Click on the "Add" button to add a new value for the variable. The values for two-dimensional arrays should be entered one column at a time. Array elements not explicitly assigned are assumed to have the last entered value.
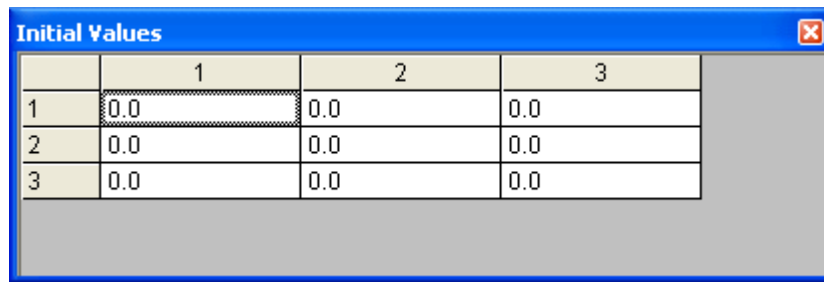
Ordinary Variable (e.g. Variable 1, 2)
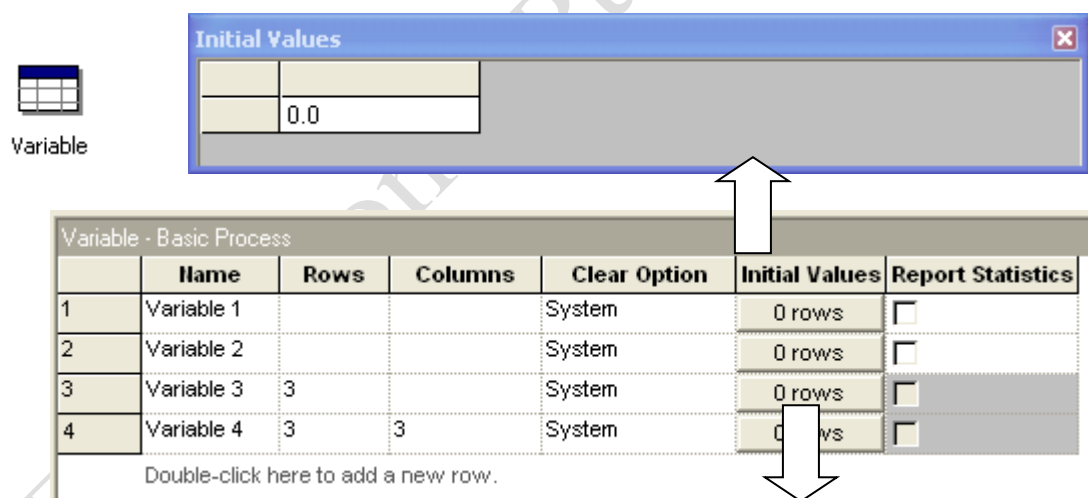
1D Variable Array (e.g. Variable 3)
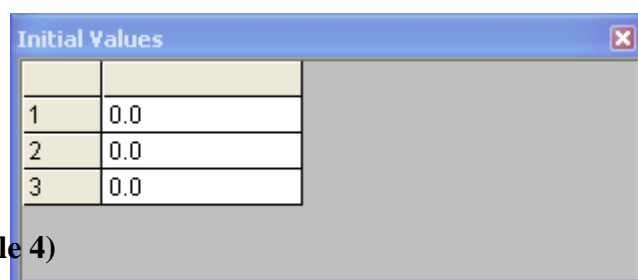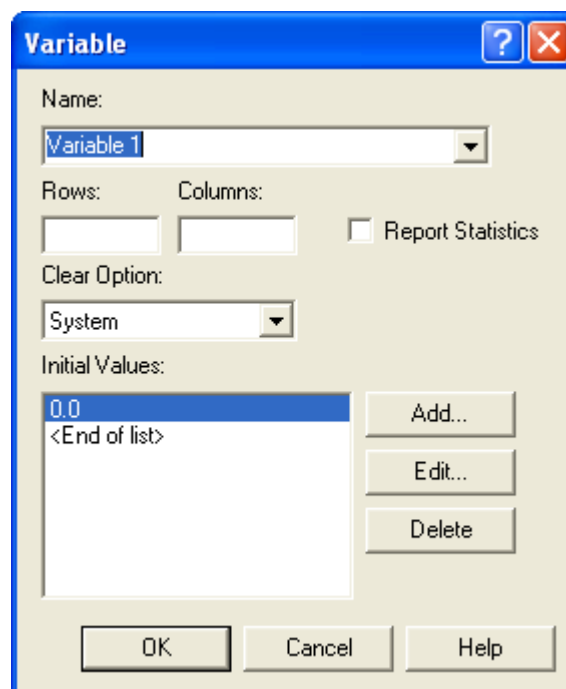
2D Variable Array (e.g. Variable 4)

(a)

(b)

Figure 6.20: Variable Data Module Spreadsheet view with snapshots of array

views

6.2.10 Schedule Module

The Schedule data module is normally used in conjunction with the Resource Module to define the availability of resources or with the Create Module to define an arrival schedule. A schedule may also be used and referenced to factor time delays based on the simulation time. This module is only used for duration formatted schedules. Calendar formatted schedules are defined by selecting the Calendar Schedules, Time Patterns command from the Edit menu. Figure 7.21 shows the

module icon and its spreadsheet view. As shown, the "Type" column can be capacity, arrival or other. Capacity schedules are used for resources whilst arrival schedules are used for scheduling arriving entities into the model. The "other" option is used for scheduling miscellaneous time delays or factors. Now to define the schedule itself, click on the durations cell to display the graphical schedule editor. Figure 6.21 (b) shows an example of a resource scheduled to have capacity of 1 for 12 hours and capacity of 2 for the next 12 hours of the day. It is important to note here that the number of hours available in a day as displayed in the schedule editor depends on the value given for the hours per day in the Run Setup dialog (see section 5.7.2).
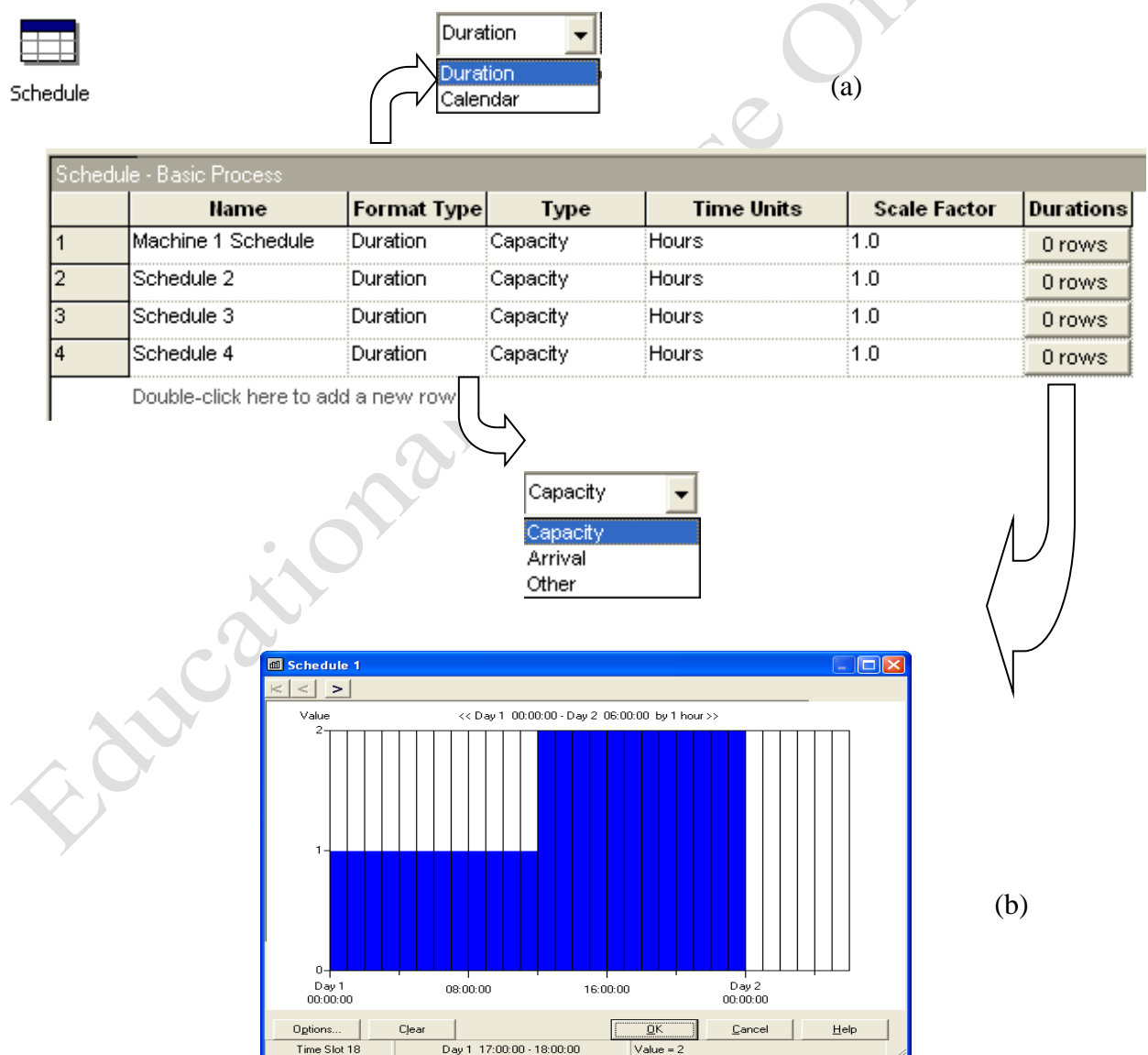
Figure 6.21: Schedule Data Module Spreadsheet view

6.2.11 Set Module

The final module we will look at in the basic process panel is the *Set* module. This data module's icon and spreadsheet view are shown in figure 6.22 below. It is mainly used to define various types of sets, including resource, counter, tally, entity type and entity picture. Resource sets can also be used in the Process (and Seize, Release, Enter and Leave of the Advanced Process and Advanced Transfer panels) modules. The types of sets that may be defined in this module are shown in the pop-up menu shown in figure 6.22. Counter and Tally sets can be used in the Record module. Other types of sets for example, Queue sets can also be defined but not in this module. To do these use the Advanced Set module in the Advanced Process Panel.
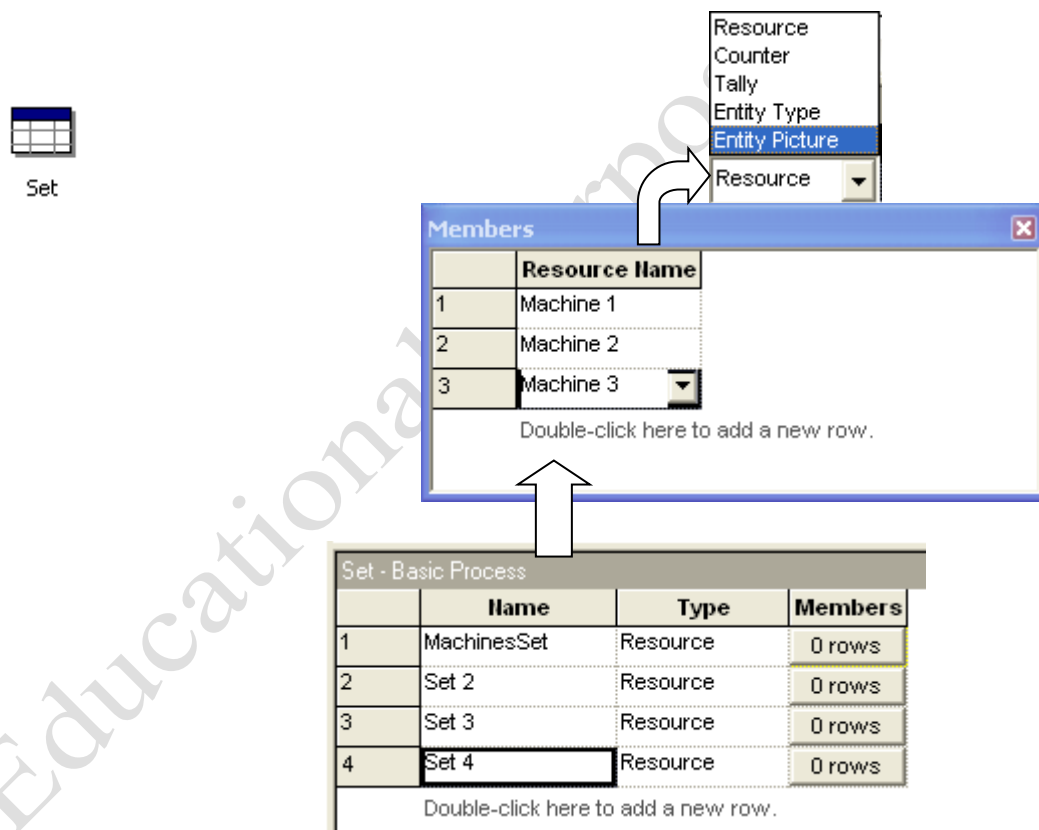
Figure 6.22: Set Data Module Spreadsheet view

In this chapter, we have been focusing on explaining in detail the main functions of all the flowchart and data modules in the basic process panel.

If you have understood the material in this chapter, you should become considerably familiar with the Decide, Batch, Separate, Assign and Record flowchart

Modules and the Entity, Queue, Resource, Variable, Schedule and Set data Modules. With these and the Create, Process and Dispose Modules discussed in chapter 6, you should be able to build models with considerable detail after a couple of examples.

There are several other Modules in the Advanced Process and Advanced Transfer panels. In the next chapter, we will take you through the process of modelling a reverse logistics system. This will further provide more examples of the use of some of the modules treated in this chapter and also introduce you to some new modules and concepts in Arena.

CHAPTER 7

Simulation and Modelling Using Arena (3):

Modelling a Typical Recycling and Reverse Logistics Problem

This chapter covers:

1. Issues surrounding systems modelling.

2. Developing a modelling approach for a typical simulation problem, an example in Reverse Logistics

3. Building and running a complete simulation model.

4. Reporting and analysing the results of a simulation run.

7.1   Introduction

We have so far laid enough foundation for tackling a real simulation problem. Since the objective of this course is to help you understand the use of simulation for modelling and analysing systems. Step by step we will now go through an example together.

It is important to realise that Arena's modelling concepts are always the same irrespective of the kind of system you are modelling. The only difference is that you need to understand how to interpret the elements of your particular system in order to know the modelling features you can use in Arena[5].

For example an *entity* in Arena is a generic concept. You need to understand that if you are modelling a banking system then your *entities* may be customers, data or financial transactions (or physical money). If you are modelling a manufacturing system, your *entities* may be parts, or if you are modelling a healthcare system, your *entities* may be patients. And since we are going to be modelling the return of products in our Reverse Logistics system, our *entities* will be returned products.

In section 7.2 I will present the problem definition and define the limits of our model. We will then continue to develop a modelling approach to our problem, defining what modules we may need and how much detail we intend to capture in our model. This will then lead us to building running and viewing the results of our basic model.

In section 7.2 we will enhance the model by remodelling the resources more realistically taking into account failures, schedules and resource states.

Section 7.3 adds further enhancements to the model for display purpose, introducing entity pictures, resource pictures, variables and plots. We will finally end

---

[5] I am not really biased over Arena. If you would like to use any other Discrete Event Simulation software package feel free to do so. The principles are the same.

the chapter with section 7.4 where we model entity transfers with the concepts of *Stations* and *Routes* and further enhance the model's animations.

7.2   A Reverse Logistics Problem

Dekker et al (2004) [1] observed that business activities of IBM, one of the major players in the electronic industry, involve several types of *"reverse"* product flows. They identified the following elements of the reverse logistic flow of IBM's business market;

1. **Main sources of return items**

   i. Products returned from expiring lease contracts – accounts for 35% of IBM's hardware sales

   ii. IBM product Take-Back programmes in several countries in North America.

   iii. Customers return used products for free or small fee

   iv. "Reverse" stream of new products which include retailers overstock and cancelled orders.

   v. Return of rotable (replaceable) spare parts – defective parts replaced in a customer's machine are sent back for repairs and possible reuse.

2. **Strategy**

   i. IBM setup a dedicated business unit in 1998 responsible for managing all product returns worldwide.

   ii. The goal was to manage the recyclability and reusability of returned items to maximise the total value recovered.

   iii. Operates 25 facilities worldwide where returns are collected, inspected and assigned to an appropriate recovery option.

3. **Process**

i. Remarkable equipment may be refurbished and put back into the market.

ii.   For this purpose, IBM operates nine (9) refurbishment centres worldwide, each dedicated to a specific product range.

iii.   Remanufactured equipment are normally internet or public auctioned.

iv.   Equipment that does not yield sufficient value as a whole is sent to a dismantling centre in order to recover valuable components such as hard-discs, cards, boards etc which can be reused

v.   The remaining return equipment is broken down into recyclable material fractions and sold to external recyclers.

vi.   In 2000, IBM reported the processing of 51,000t of used equipment, of which only a residual of 3.2% was landfilled.

7.3   Model 7.1: Modelling the Reverse Logistic Flow of an Electronic Company

Please note: The concept of this problem formulation is based on an illustrative case presented in Dekker et al (2004) (p.66-67). The system presented here is completely imaginary and has no relation with the operations of any real company. The aim here is to help you appreciate the issues involved in modelling a reverse logistics system.

Problem formulation

This system describes the operations of a refurbishment facility of an electronic firm as shown in figure 7.1.

The units that are returned to this facility are named Products A through D. All the products are collected and sorted at a separate facility outside the scope of this model. Product A arrives at a rate described by an exponential distribution with a mean of 6 (all time in minutes). These Products are transferred upon arrival to the Product A prep area where they are prepared for inspection. The prep process follows a TRIA (5, 7, 10) distribution. The product is then transferred to the inspection area.

Products B, C, D follow a similar process to Product A – but with different arrival rates and preparation (Prep) times, as indicated in figure 7.1. After their Prep processes, these Products are also sent to the inspection area.
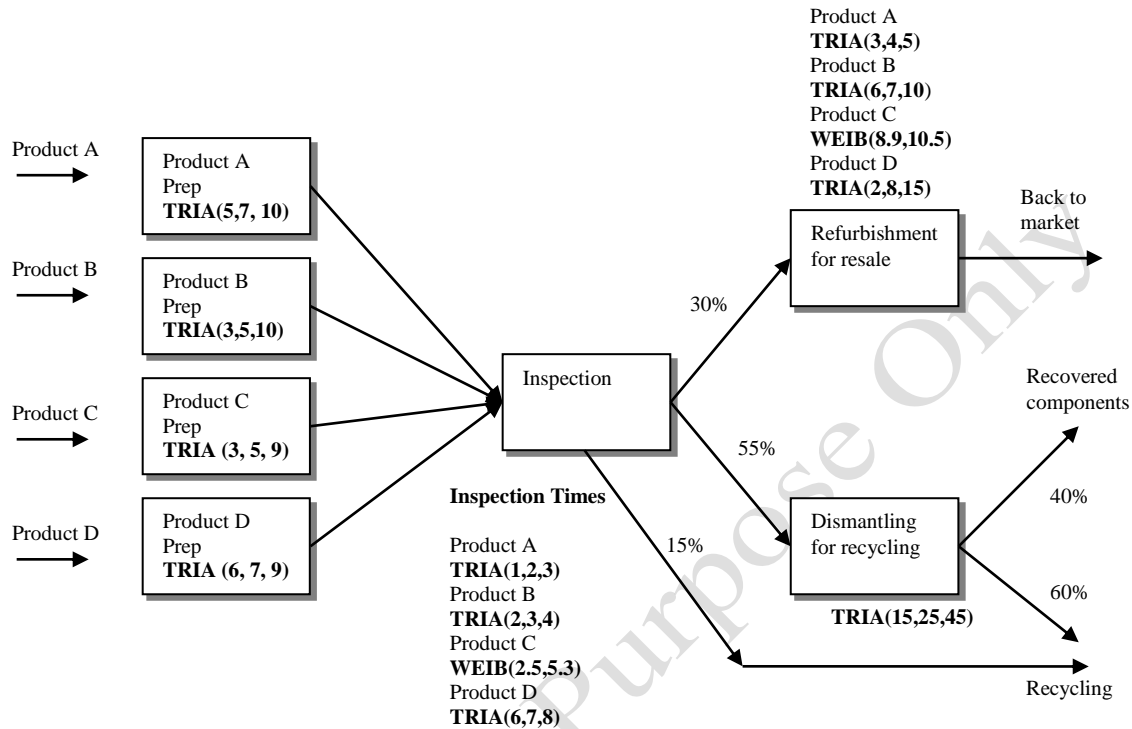
Figure 7.1 Returned products testing and refurbishment process

At the inspection area functional checks are performed on the products to check if they could be reused. The total process time for this operation depends on the product type: TRIA (1,2,3) for Product A, TRIA (2,3,4) for Product B, WEIB (2.5, 5.3) for Product C, TRIA (6,7,8) for Product D. After inspection 30% of Products are sent for refurbishment, 20% are sent for remanufacturing which is outside the scope of this model, 35% are sent for dismantling and subsequent recycling and the remaining 15% are rejected and disposed for recycling.

At the refurbishment area, products that are still salvageable are repaired using additional supply of parts and then returned into the market. The processing times in this area are TRIA (3,4,5), TRIA (6,7,10), WEIB (8.9,10.5) and TRIA (2,8,15) for Products A, B, C, and D respectively. Upon arriving in this area, the products have to wait for an appropriate part to be available. The parts are supplied in batches of 4 and come at the rate of twice a day.

At the dismantling area, all products are separated into individual components with those in good condition (40%) recovered for remanufacturing and the rest disposed for recycling. The dismantling process takes TRIA (15, 25, 45) and is irrespective of product type.

We want to collect statistics in each area on resource utilisation, number in queue, time in queue, and the cycle time (or total time in system) for refurbished parts, recovered components, and recycled parts. We will run the simulation for four consecutive 8-hour shifts, or 1,920 minutes.

7.2.1 The modelling approach

It must be repeated at this stage that using Arena to build a model is only one component of a simulation project. For a refresher of the discussion on the entire simulation project see chapters 4 to 6 of this book. The point to note here is that in the real world, there will not be a readily defined problem with data available or supplied as our example. In the real world often this information does not exist in the company and should painstakingly be collected. In fact input data collection, validation and verification can take up to 70% of the total simulation project time and effort.

The focus of this section is on the approach to developing and modelling a typical system. This is the stage where you as a modeller after understanding the problem and clearly stating your goals have to define your system, collect and analyse the data you require to specify your input parameters. At this stage you need to take a global view of your system and develop the best way to represent it. To do this you may need to segment your system into stations or sub models and decide on which Arena modules you might require.

Kelton et al (2010), recommend as a general approach when creating models to stay at the highest level possible for as long as you can until it is necessary to drop to a lower level. In this model, we will be using modules mainly from the Basic Process Panel and introduce new ones from other panels as we enhance the model. When you become familiar with Arena, you will find it important to stick to the advice of only highest level modules whenever possible.

In chapter 5, we described the hierarchical structure of Arena that allows the combination of modelling constructs from any level into a single simulation model. We have so far built very simple models using modules from only the Basic process panel. In this chapter we are going to introduce new modelling concepts that will require the use of the Advance process and Advanced Transfer panels.

Assuming all previous steps have been carried out, we will now break our system down into identifiable sections convenient for efficient modelling. The main aspects of our model will therefore be to

- Create arrival of products

- Send products through prep process

- Send products through inspection process

- Decide where each product goes after inspection

- Send part to refurbishment

- Send another part to Dismantling

- Dispose remaining part to Recycling

- Dispose to market after refurbishment

- Split products into components after dismantling

- Dispose recovered components after dismantling

- Dispose after dismantling to recycling

To do the above, we will require the following modules:

- Create (4)

- Assign (4)

- Process (7)

- Decide (2)

- Record (3)

- Separate (1)

- Dispose (3)

Each *Create* module will represent the arrival of each Product type. Each *Assign* module will also be used to assign *Attributes* to each product type. Four *Process* modules will represent the Prep process for each product type, one for the Inspection process, one for Refurbishment process and the last for the dismantling process. One *Decide* module will be used after the inspection process to split the products for refurbishment, dismantling and recycling. The other *Decide* module will be used to separate recovered components from those to be recycled after dismantling. Before each stream of products is disposed, we will use a Record module to collect statistics on time they spent in the system. Finally the three *Dispose* modules will be used to dispose products to market, recovered components and recycling.

Remember that arrival rates and Prep times are unique for each product type as shown in figure 7.1. We will use two attribute called *Inspection time* and *Refurbishment time* to assign the different times spent at the inspection and refurbishment processes for the various products types. The time for the dismantling process is constant for all products. We will call our resources Prep A through D, Inspector, RefTechnician and DisTechnician respectively for the Prep processes, inspection, refurbishment and dismantling. We are now ready to build our model.

7.2.2 Building the model

Start Arena and open a new model window (If you are unable to do this, refer to section 3.1 of Kelton et al (2010)). Place the required flowchart modules as mentioned above in the flowchart view of the model window.

Your model window should now look somewhat like figure 7.2. It was explained in chapter 5, how to place modules in the flowchart view by dragging and dropping. As you drag and drop your modules, Arena should be connecting them automatically if you have the auto-connect option on. To turn this off and on, select the objects menu and click on auto-connect.

We also explained that double clicking any module in the flowchart view will display the module's dialog in which its parameters can be updated. Remember that Arena simultaneously displays similar data for the selected module in the spreadsheet view showing the parameters of all modules of the same kind within the current model.
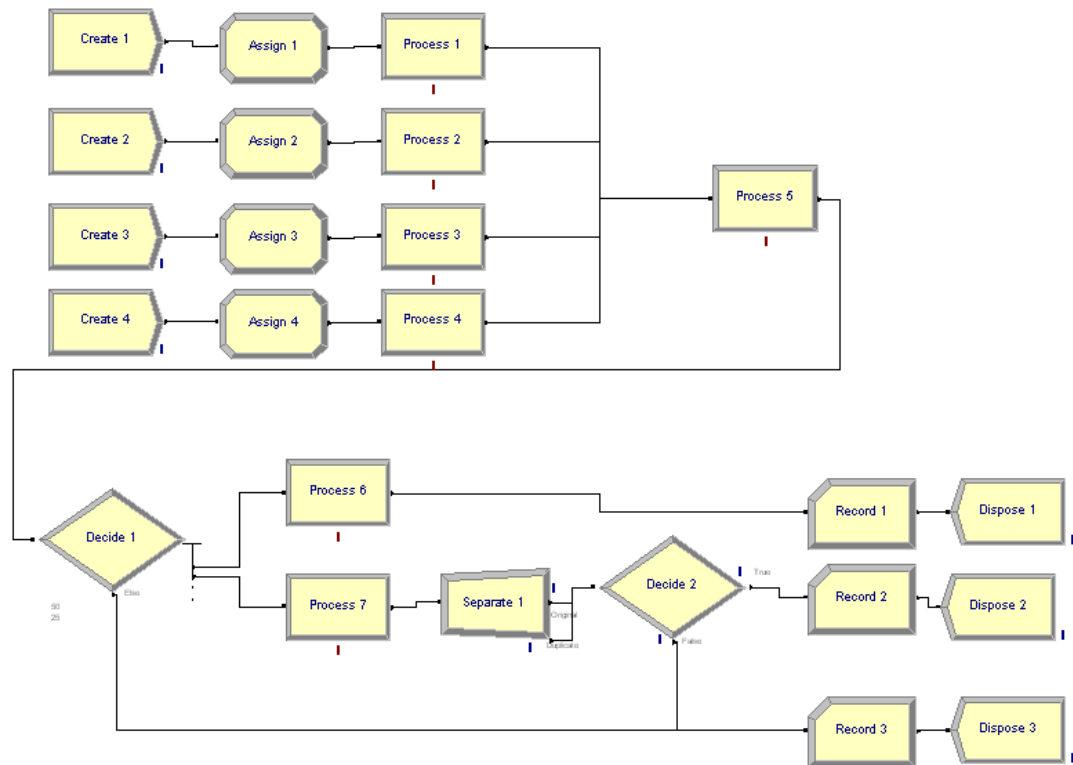
Figure 7.2: Modules placed in Model Window

Let us start by updating the *Create* modules, so double click the *Create 1* module to display its dialog. Set its name to *Create Product A* and Entity type to *Product A*. In the *time between arrivals* area, set the *type* to Random (Expo) with a mean value of 8 and select minutes for the *Units*. Let us assume for now that the products arrive in singles so we set the *Entities per Arrival* to 1. Leave the *Max Arrivals* to the default setting of *Infinite* and zero for the *First Creation*. After all that, your create dialog should look like figure 7.3.
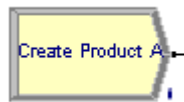
Figure 7.3: Completed Product A *Create* dialog box

Similarly, the *Create Modules* for Products B, C and D are shown in figure 8.4 below.
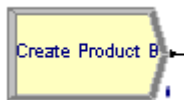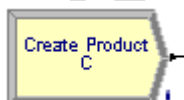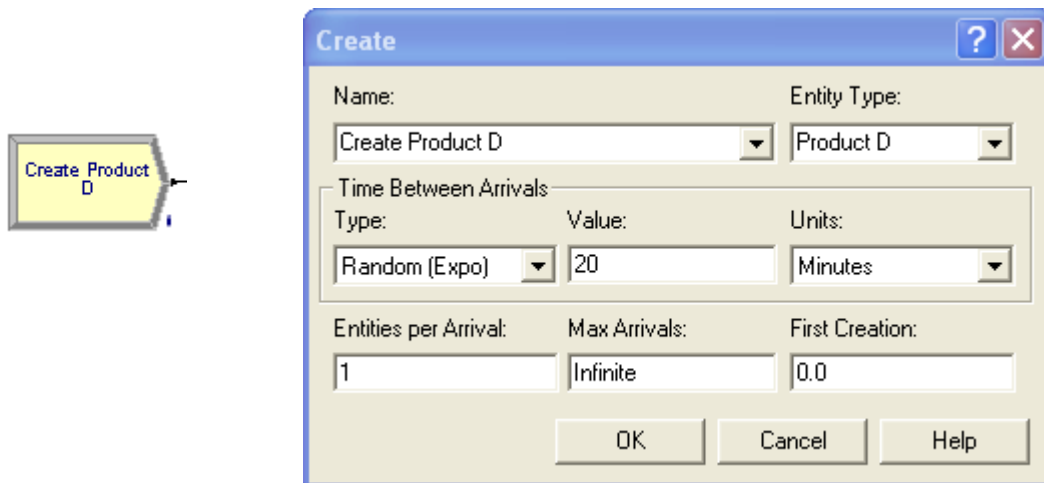
Figure 7.3: Completed Products A, B and C *Create* dialog boxes

Note that the main purpose of the *Create Module* is to provide a starting point for entities in a simulation model. For further details, refer to chapter 6. The above instances therefore are the starting points for all the products that come into our system. In the *Create Modules,* we specified the *Entity Types* to be *Products A through D.* This is not the only information we need about each product. We may also want to know what time the products arrived in our system, how much time that product will spend at inspection area etc. The information that are specific to each product are known as *Attributes* and are assigned to the entities by the *Assign Module* which we will discuss next.

We want to know the arrival time for each product. We also would like to know how much time that product spends at the inspection and refurbishment processes. For these, we define the following attributes, *Arrival Time, Inspection Time* and *Refurbishment Time*. Let's now open each of the *Assign Modules* and enter the information required. Double click the *Assign 1 Module* and enter information as shown in figure 7.4.
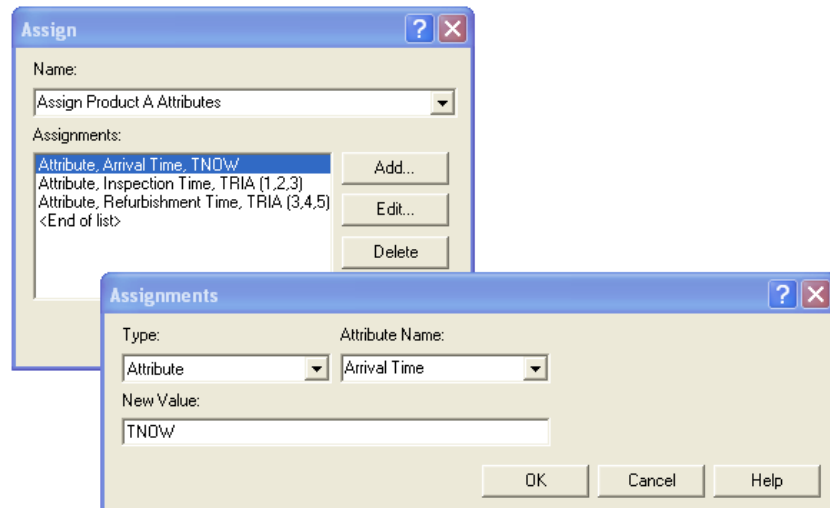
Figure 7.4: Assigning attributes to Product A
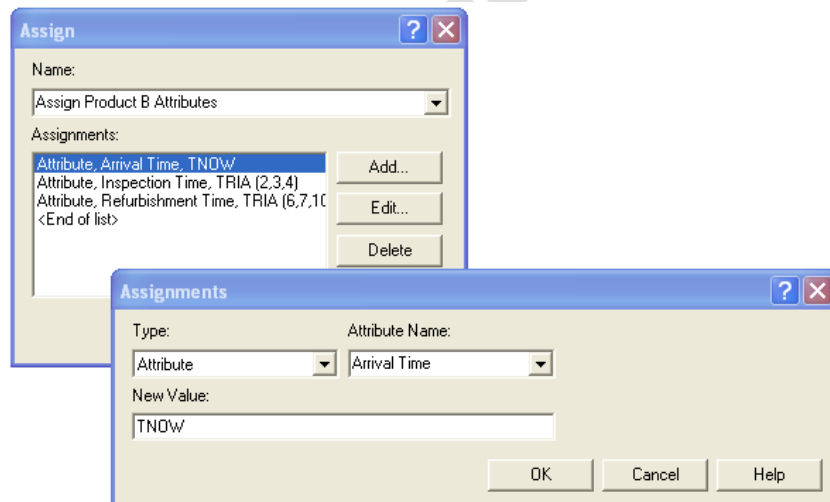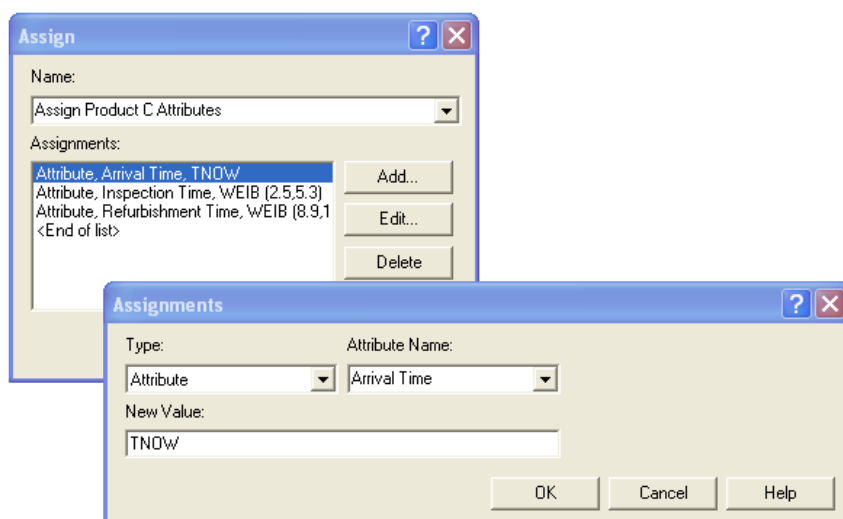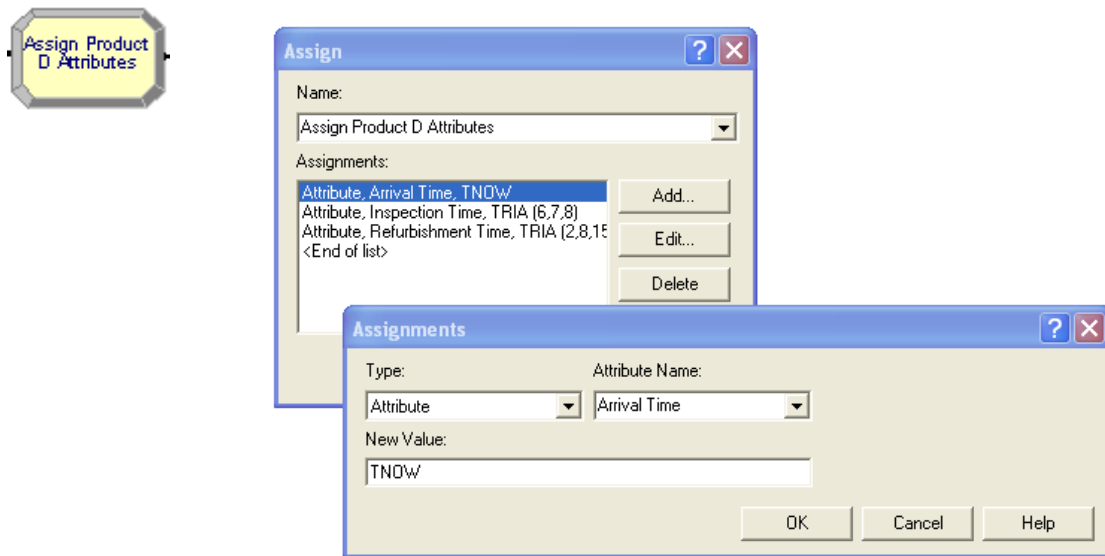
Figure 7.5: Completed Products A, B and C *Assign* dialog boxes

New assignments are added by clicking on the "Add" button which displays the assignments dialog as shown above. This module is used for assigning new values to variables, entity attributes, entity types, entity pictures, or other system variables. Multiple assignments can be made with a single Assign module. Following the above steps assign the corresponding attributes to Products B, C and D as shown in figure 7.5. TNOW is a standard Arena reserved variable that provides the current simulation time.

The next thing is to edit our *Process Modules* to update their parameters. We will start with the four *Prep Processes*. Double click on the *Process 1* module to open its dialog. The completed dialog is given in figure 7.6.
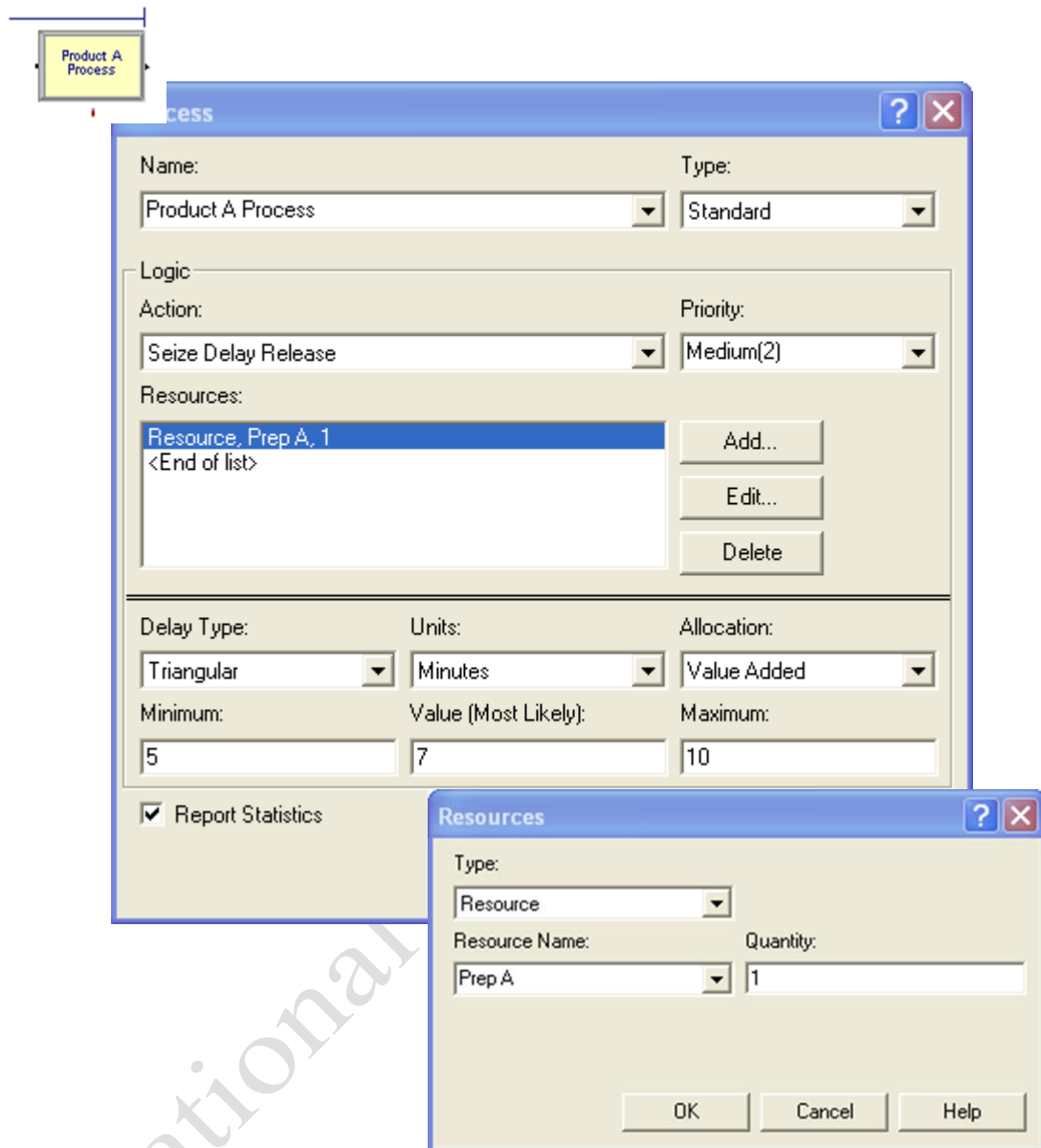
Figure 7.6: Completed Products A Prep Process dialog

We named a module *Product A Process*. We also used a standard process as you can see. We selected the *Seize-Delay-Release* action combination. This is because the products will normally seize the *Prep A* resource, delay the resource for the duration of the prep process and then release the resource before moving on to the next step in the process logic. The prep time for product A was given in the problem definition as a triangular distribution with parameters 5, 7 and 10. The time unit is minutes and the time allocation is value added. Leave the report statistics option checked so that Arena will provide reports on this process module.

New *Resources* are added by clicking on the "Add" button which displays the *Resources* dialog as shown above. This module is the main processing method is the simulation. It could contain a submodel by selecting the submodel option in the type combo box and perform various actions of seizing, delaying and releasing resources when the standard option is selected. For further details, on this refer to chapter 6.

In a similar procedure as above, update the remaining Prep Process modules. The completed modules are shown below.
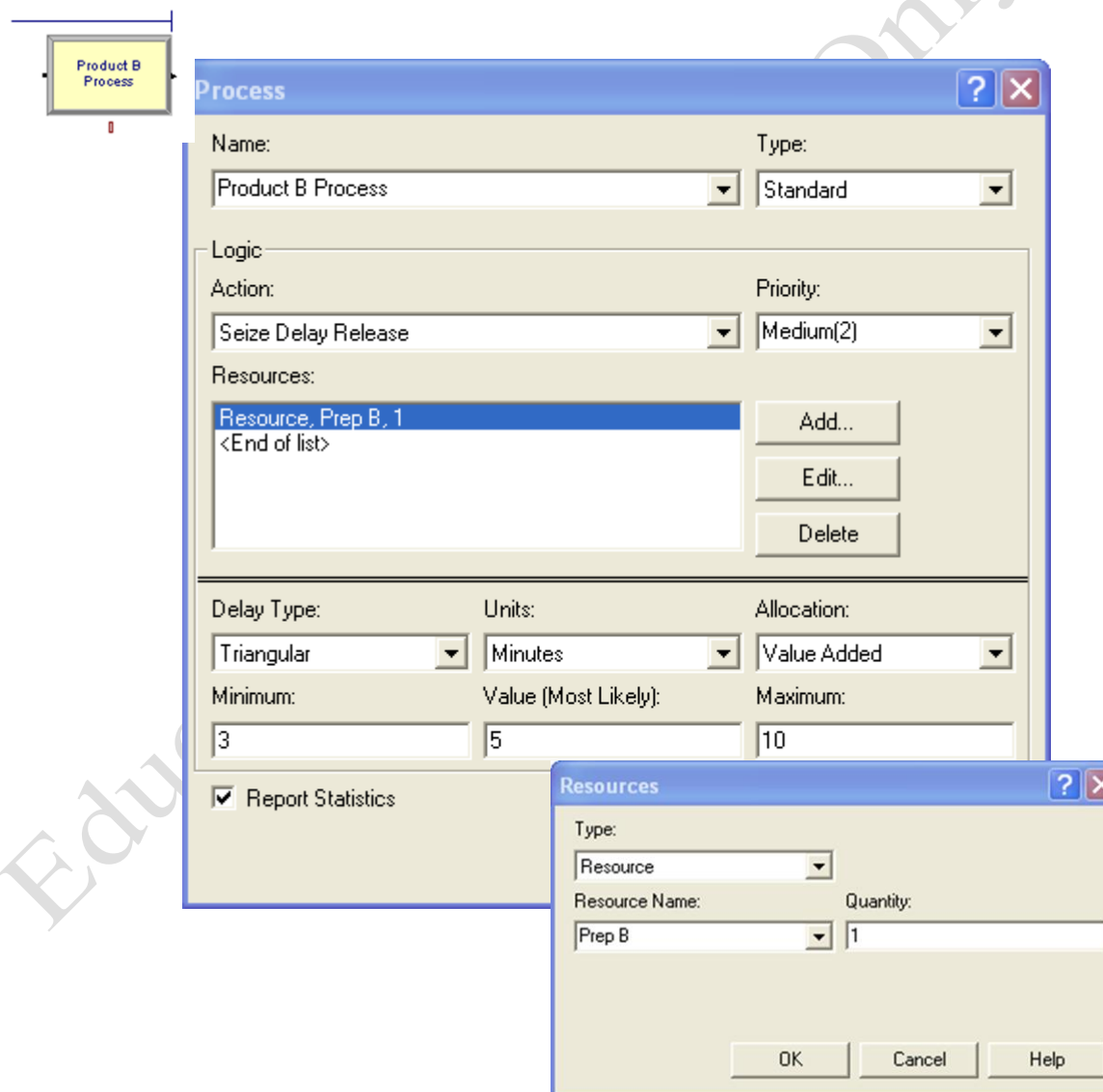
Figure 7.7: Completed Products B Prep Process dialog
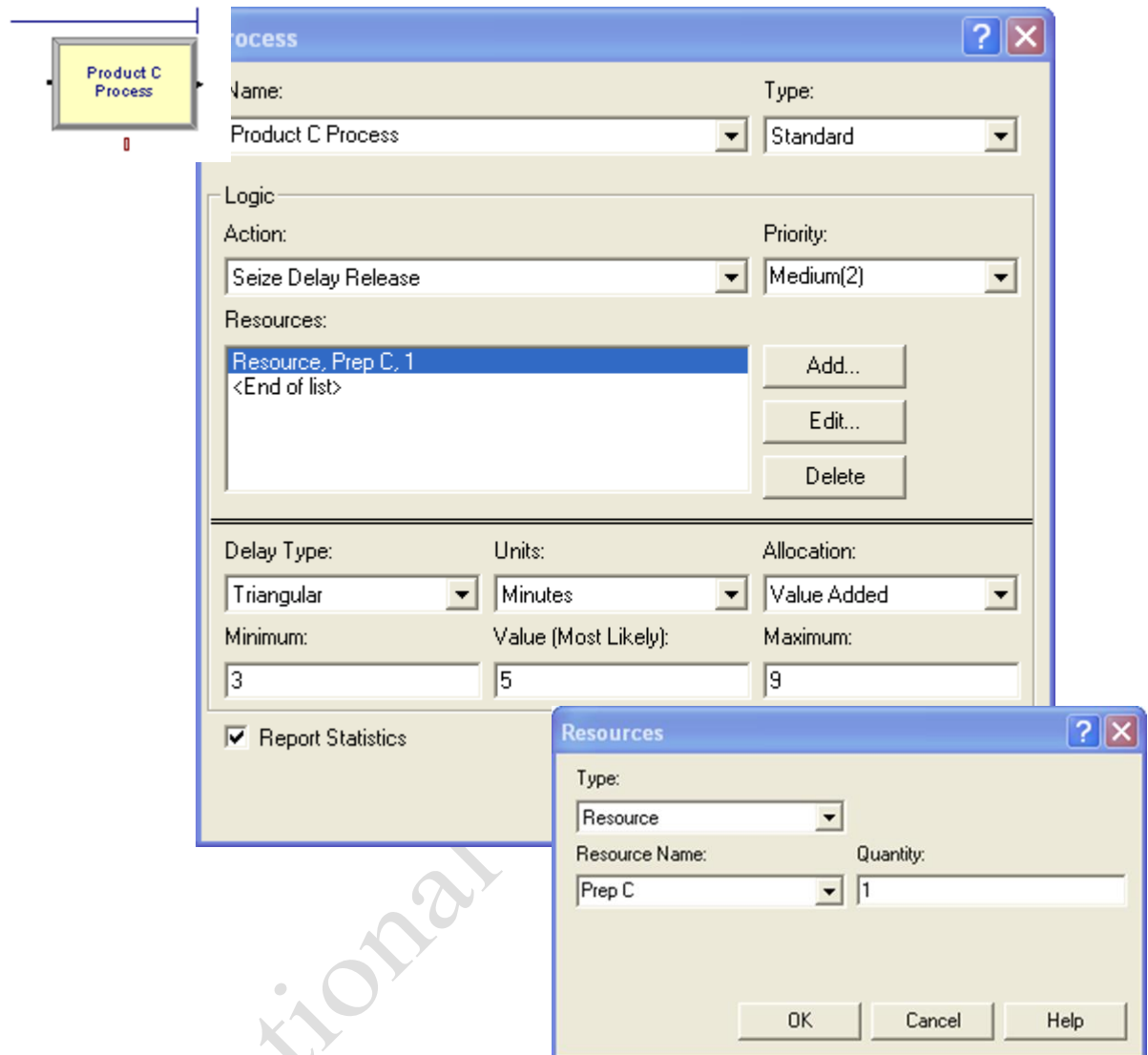
Figure 7.8: Completed Products C Prep Process dialog
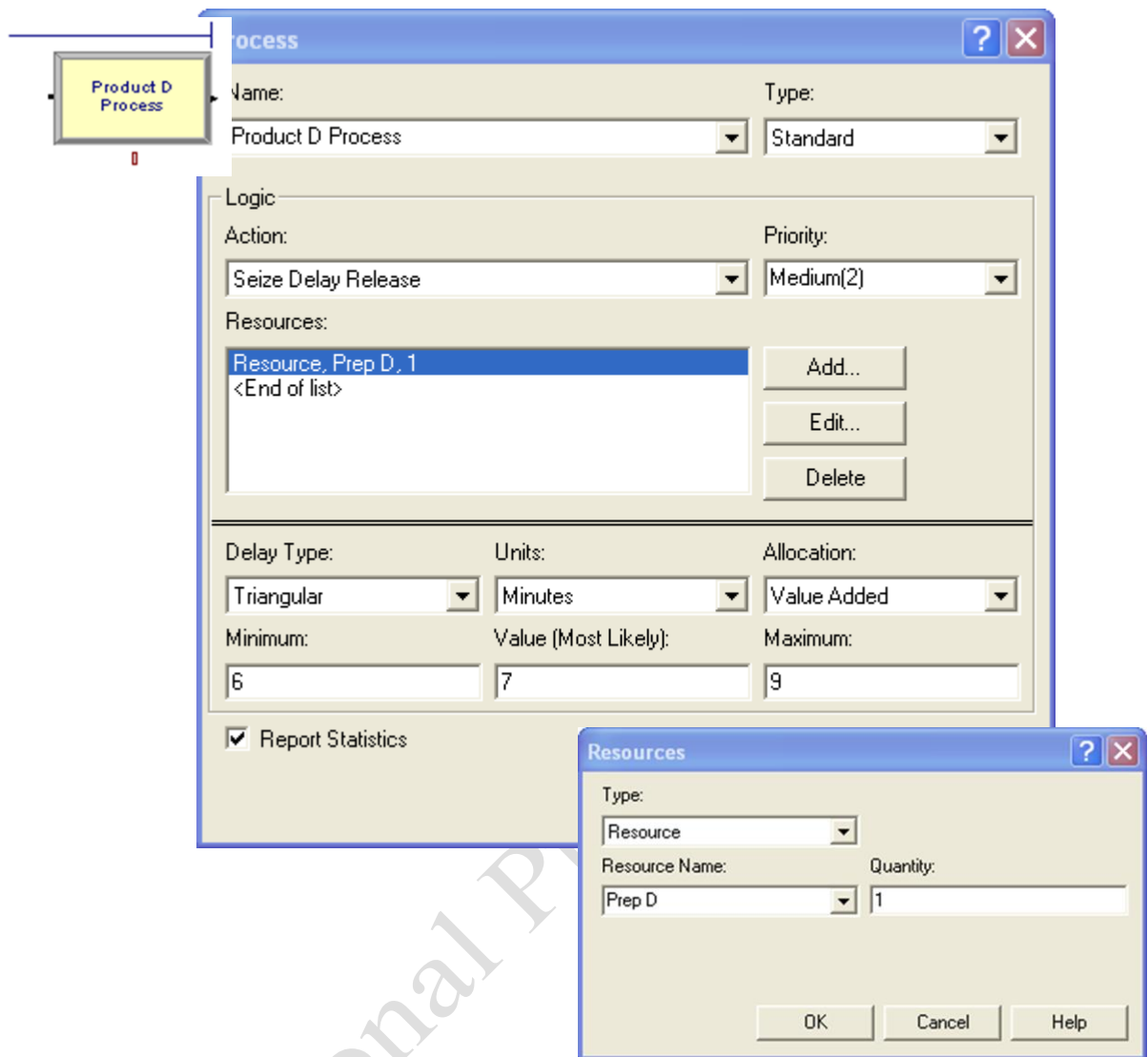
Figure 7.9 Completed Products D Prep Process dialog

The inspection process is very similar to the Prep processes except that it uses an expression for the delay instead of the triangular distribution type we have been using so far. This is the reason why we defined the inspection time attribute in the assign module. When you choose the delay type to be an expression, you can define any expression (sums or differences, products or rations of variables and attributes) and Arena will evaluate that and use the resulting value as the process time for the entity (product). In this case we only wanted to use a value we had pre-assigned to the entity's inspection time attribute.

The completed module is shown in figure 7.10.
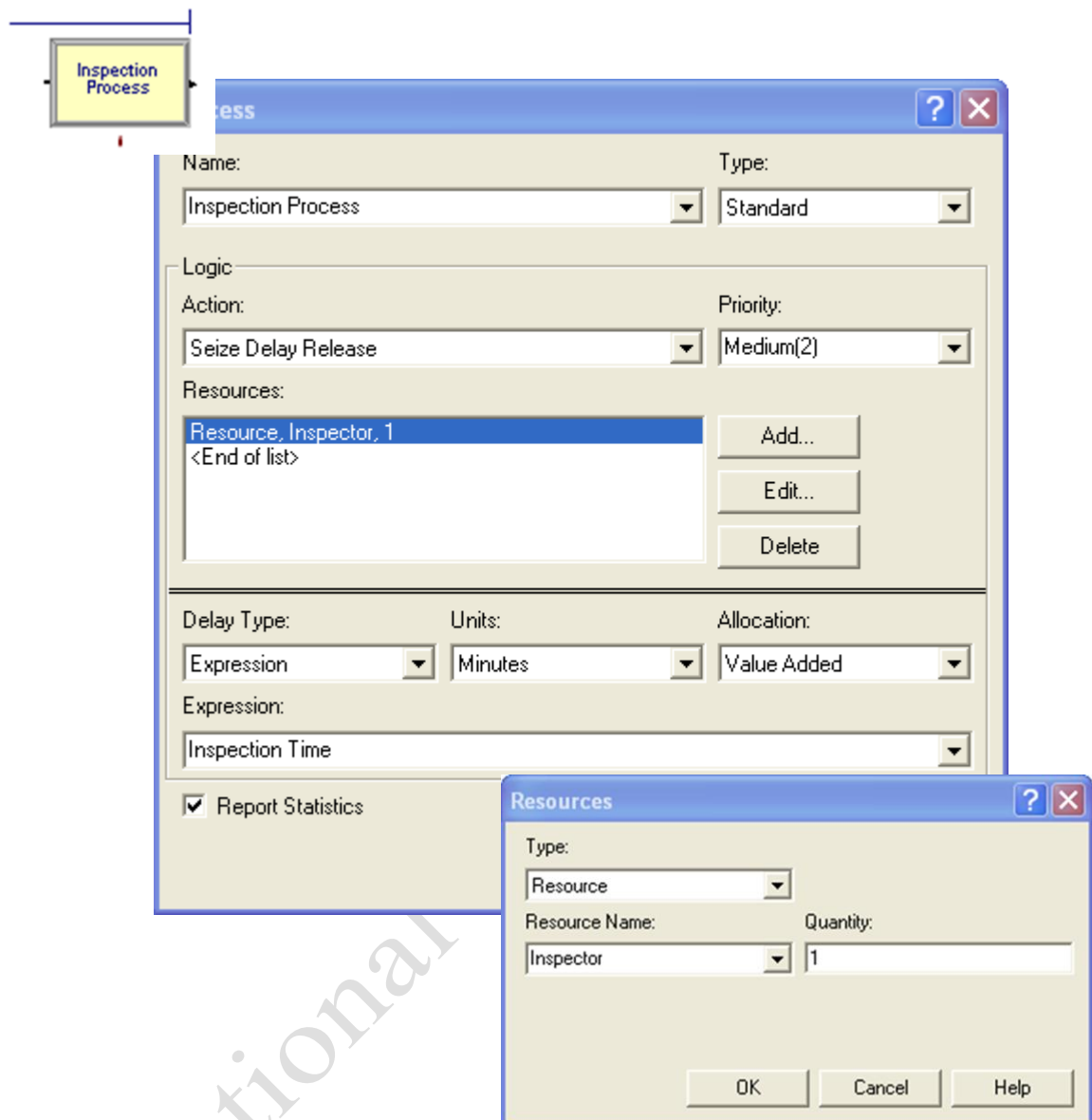
Figure 7.10: Completed Inspection Process dialog

Decisions or choices in Arena are modelled using the *Decide Module*. The problem definition states that following inspection, 30% of products are sent for refurbishment, 55% for dismantling and 15% for recycling. The *Decide Module* includes options for making decisions based on one more conditions or based on one or more probabilities. Since we have values given for the percentages, we will use the options based on personalities which Arena refers to as by chance.

Now double-click on the *Decide 1 Module* to display its dialogue. Since we have three possible outcome, we will select the *N-way by chance* option from type combo box in the decide dialog. The *Add* button displays another dialog in which you specify

the probability value. We will specify two values of 30% and 55%. Arena will automatically work out the difference and send the remaining 15% to another exit point. Notice that Arena provides a number of exit points from the *Decide Module* that is equal to the number of conditions you specify. The completed module is shown in figure 7.11.
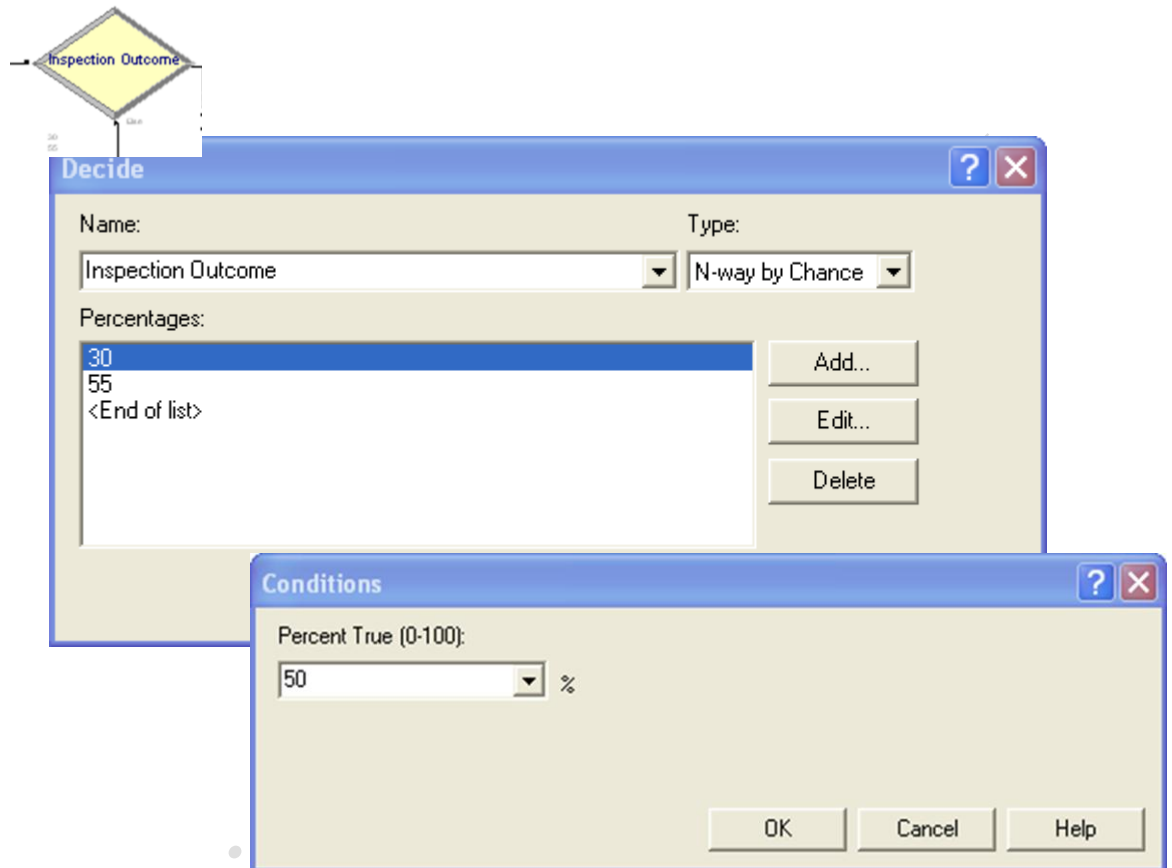

Figure 7.11: Completed Decide Module dialog

The next step in our process logic is to update the *Process 6 Module*. This is very similar to the inspection process module presented above except that we use the attribute *Refurbishment Time* instead of *Inspection Time.* The module name also changes to *Refurbishment process* and the resource to *RefTechnician*. The completed dialog should appear as shown below.
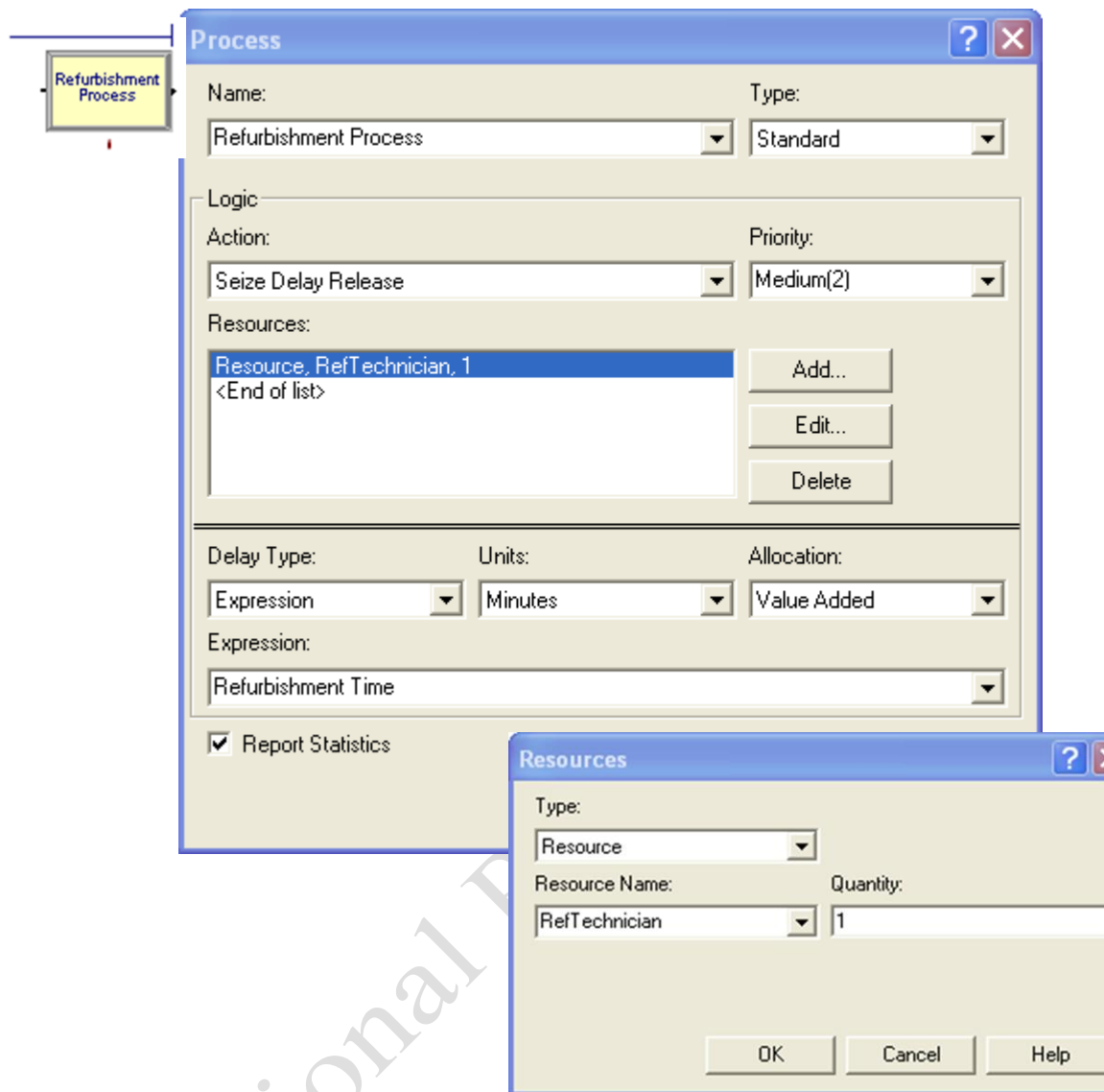
Figure 7.12: Completed Refurbishment Process dialog

Repeat the above steps to complete the *Process 7 Module.* Check with the completed dialog in figure 7.13 below.
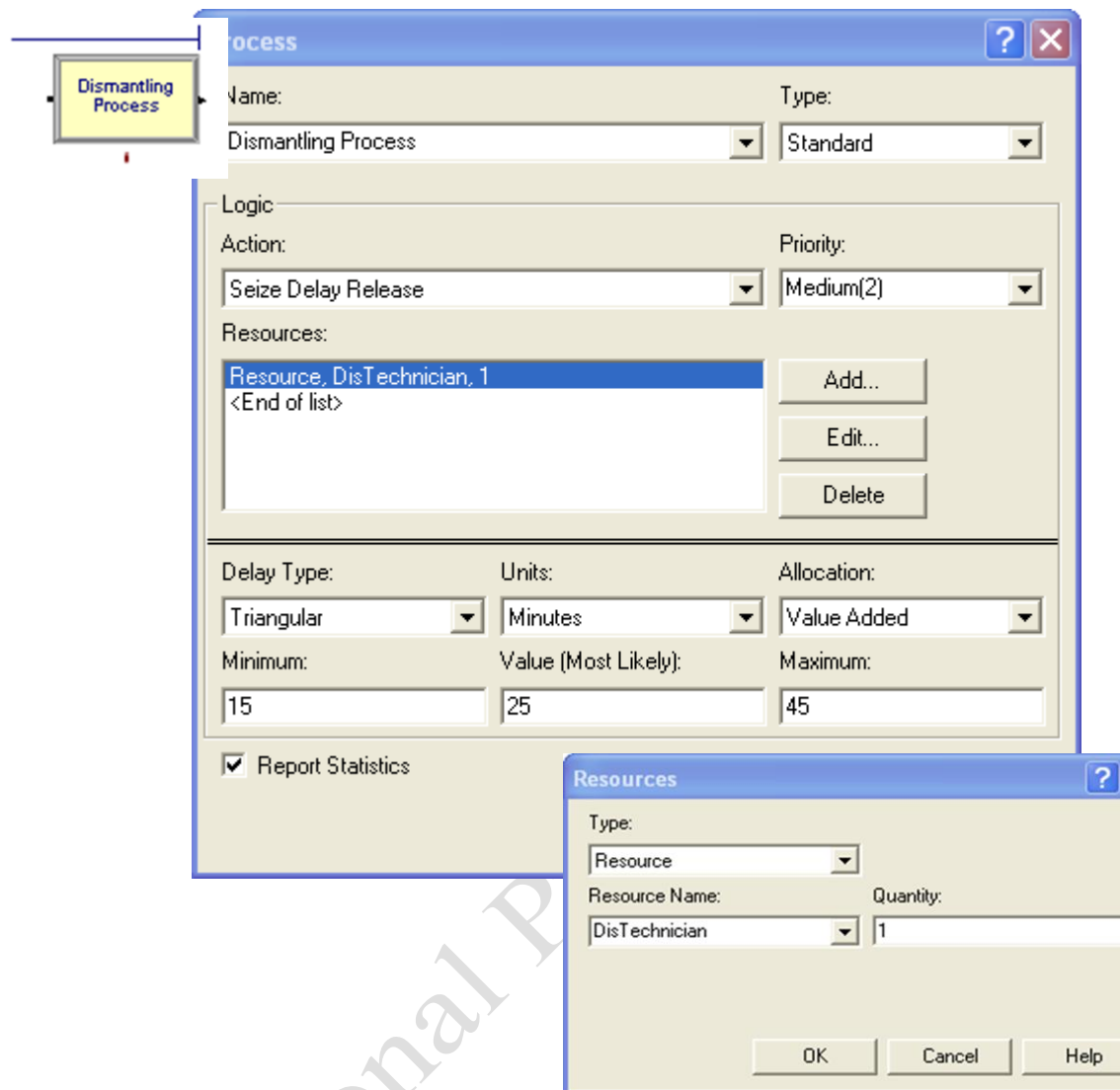
Figure 7.13: Completed Dismantling Process dialog

At the end of the dismantling process, the products will be turned into components. From the problem definition we realise that the number of components in each product varies and is represented by a triangular distribution of parameters, 6, 9 and 12. This is akin to assuming that there are a random number of components in each product. We model this by using a S*eparate Module* from the *Basic Process* panel.

Now double click on the *Separate 1 Module* to open its dialog box. Enter *Components* in the "Name" and leave the "Type" field to the default value (i.e. Duplicate Original). Also leave the "Percent Cost to Duplicate" field to its default value. The last thing is to enter the value TRIA (6,9,12) in the "Number to Duplicate" field. The completed dialog should be as shown in figure 8.14. You should realise that

the purpose of using this *Separate Module* is just to multiply the number of components leaving the dismantling process. That is you why you will observe that both exits of the module have been put together.
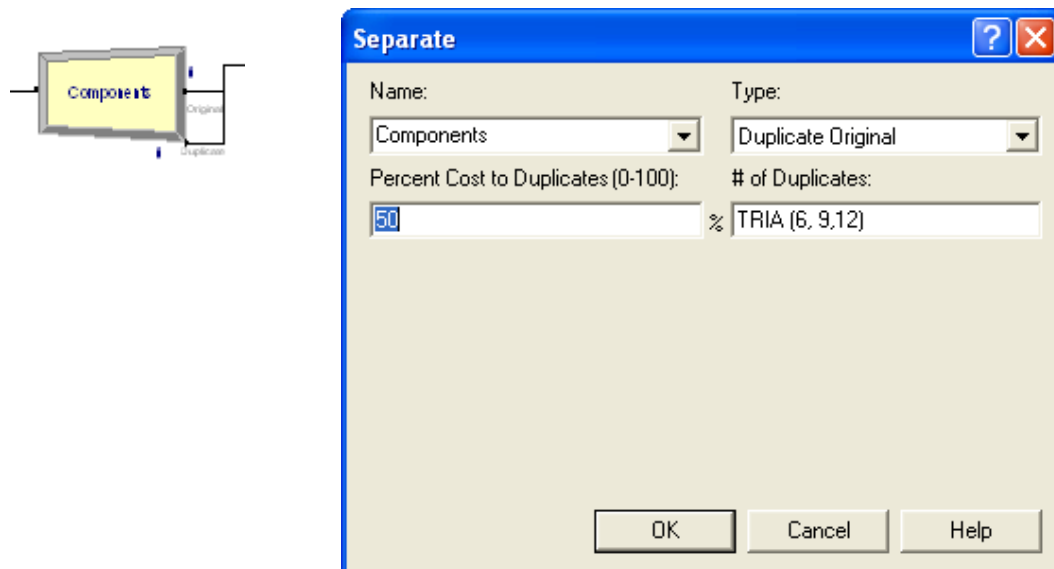
Figure 7.14: Completed Separate Module shape and dialog

There is another decision to be made after the dismantling process where 40% of components are recovered for remanufacturing and the remaining 60% sent to recycling. We will use another *Decide* Module and complete it as shown in figure 7.15. This has fewer values to specify since it's only 2-way by chance.
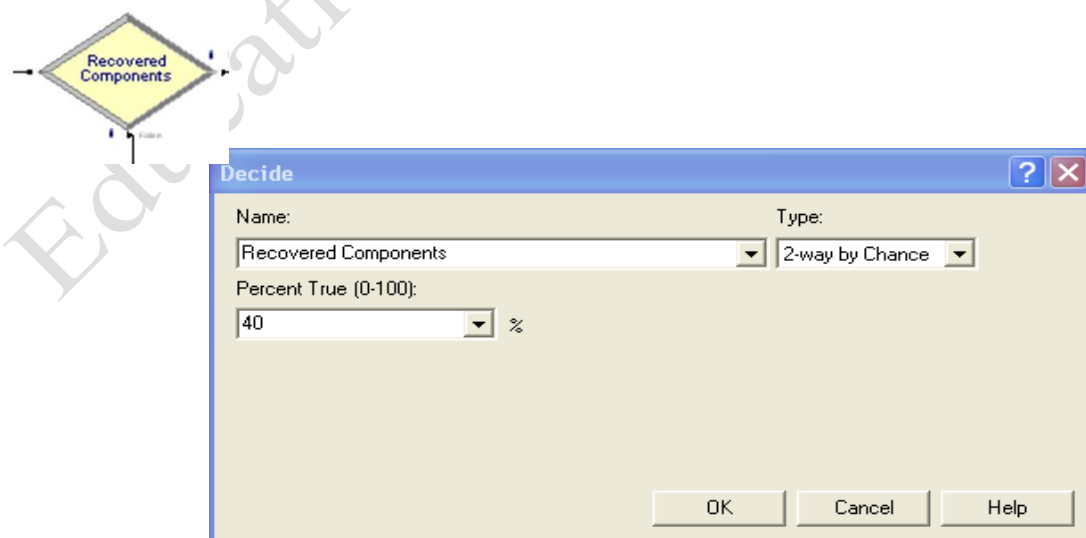
Figure 7.15: Completed Decide Module dialog

Having defined all the operations, we now need to update the Record and Dispose modules to finish.

Figure 7.16 shows the completed record dialog for recording the cycle times of products that have been refurbished products. In order to determine the cycle time which is the time from when the products arrive into our system to when they exit from the system, we use the *Time Interval* type from the record module. This is the main reason why we defined the *Arrival Time* attribute soon after the products were created. In this module, we selected the *Arrival Time* attribute as the reference for computing the cycle time (time interval). Arena makes this attribute available in the drop down list under Attribute Name in the record dialog because we had previously defined it. The cycle times observed for the entities will be recorded into a tally called "Record Refurbished".

Fill in the remaining record modules in the same way as above but with the names Record Recovered Components and Record Recycled. Notice that Arena automatically uses the module name you supply as the tally name.
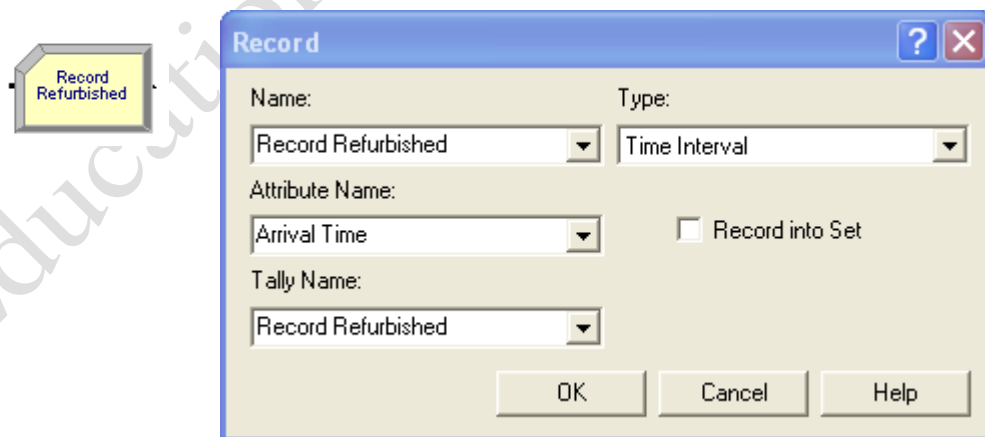
Figure 7.16: Completed Refurbished Products Record dialog

The final set of modules we have to fill in is the *Dispose Modules*. This module is intended as the ending point for entities in a simulation model. Entity statistics may be

recorded before the entity is disposed. Arena records entity statistics only when you check the box for Record Entity Statistics on the dispose dialog. Figure 8.17 shows the dispose dialog for products that are sent to the Market after refurbishment. The remaining modules are updated in the same way with the names send to remanufacturing for the recovered components and send to recycling for the rejected components.
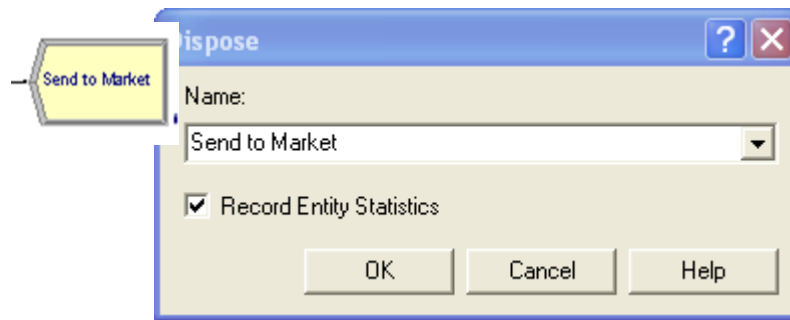

Figure 7.17: Completed Dispose dialog for refurbished products

The model can be run now but before we do so, there are a few things we need to specify as to how the model should run. One of these is to tell Arena when to stop the simulation. Without this the simulation will run forever because Arena doesn't know when to stop. This and other parameters necessary for the runtime behaviour of your simulation and information on generated report can be established by selecting "setup" from Arena's "run" menu. We will have a look at only two of the five tabs on the run setup dialog. Figure 7.18 shows the run setup dialog with the *Project Parameters* tab selected.
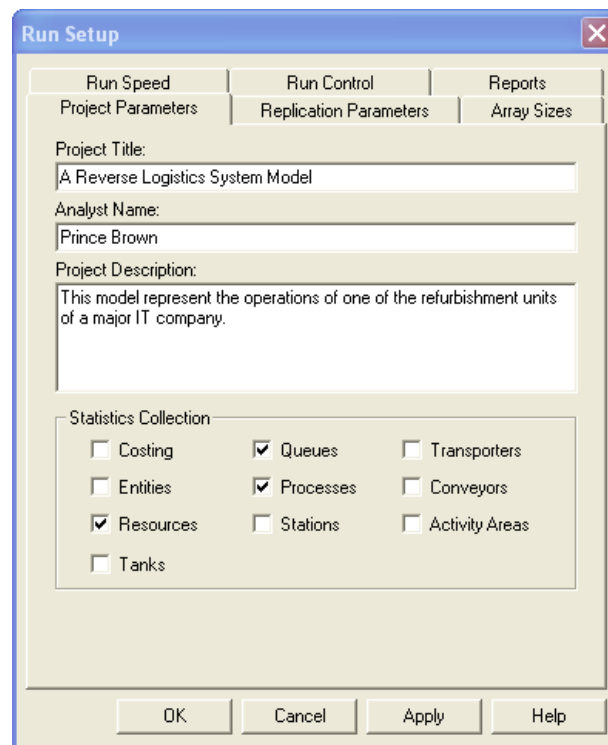
Figure 7.18: Project parameters in Run Setup dialog

This tab allows you to specify project information such as title, analyst name and project description. It also allows you to specify what aspects of your model you want to collect statistics on. We have checked resources, queues and processes for statistics collections. There will therefore be no statistics generated on all the other components of the model in the report that will be generated after the run.
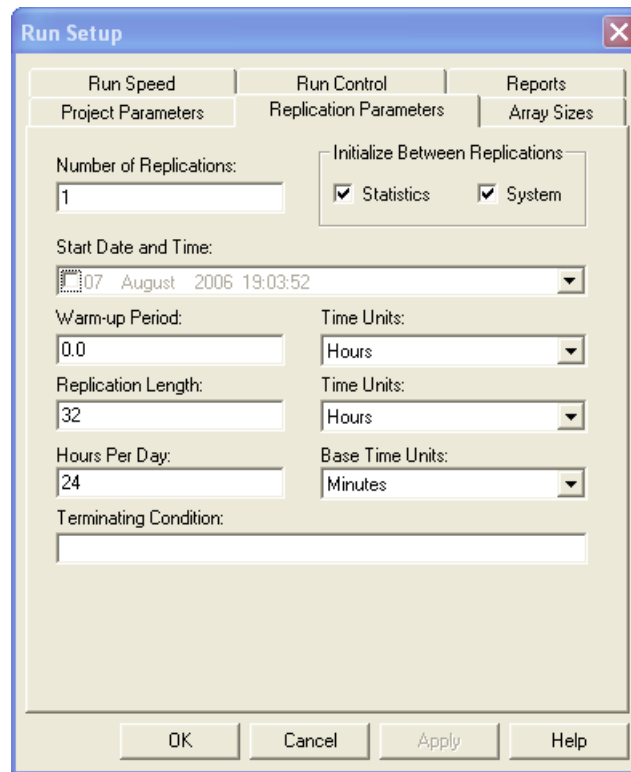
Figure 7.19: Replication parameters in Run Setup dialog

The other tab we will look at in the run setup dialog is the "Replication parameters" tab. This is displayed in figure 7.19 this is where we specify the run length for the simulation. Based on our system description, we have set the replication length to 32 hours (four consecutive 8-hour shifts). We also changed the base time units to minutes and left the remaining fields at the default setting.

There are four different types of products coming into our system and it would be nice to differentiate between them in the animation (the pictures that represent the products). We will do this by assigning different pictures to their entities using the entity data module. Bring up the *Basic Process* panel and click on the Entity data module. The first two columns of your spreadsheet view with the entity data module selected will look like figure 7.20. Arena automatically assigns an initial picture to every entity you create. In this case Arena assigns the *Picture.Report* picture to all the products which make them look the same during the simulation run.

Figure 7.20: Entity spreadsheet view with default initial pictures

Now when you click on the initial picture cell of the Product A, Arena gives you a drop down list of all entity pictures that are currently available for use. You can select different pictures from the list to represent each of your products. We used red, yellow, green and blow balls respectively to represent products A through D. See figure 7.21.

Figure 7.21: Entity spreadsheet view with updated initial pictures

After all these, your completed model should look somewhat like figure 7.22.
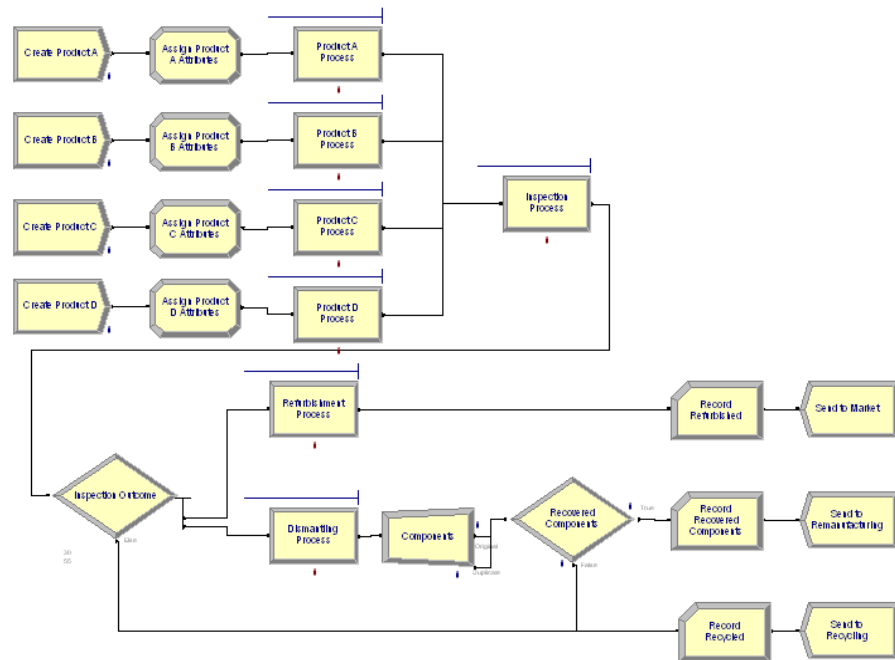
Figure 7.22: The complete model

7.2.3 Running the model

Arena cannot run a model with errors; hence the next task before running the model is to check for the errors in the model. This can be done by clicking on the check button (✓) on the Run Interaction toolbar, the check model command from the Run menu or by using the F4 key on the keyboard. If the model is without an error, Arena displays the message box in figure 7.23 otherwise you will receive an error message with find and edit buttons that will help you locate and fix the error. If there are no errors in your model, then you can run your simulation by clicking on the Go button (►) on the standard toolbar or just by pressing the F5 key on the keyboard or by selecting the Go command from the Run menu.

Figure 7.23: Arena message for successful model check

Figure 7.24 below is a snap shot from the running model[6].
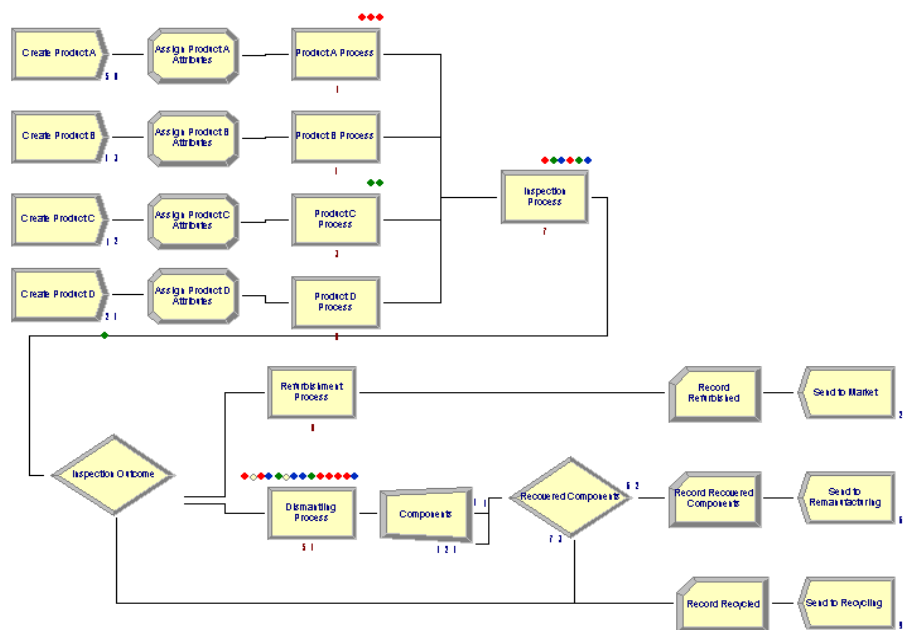
Figure 7.24: Snap shot of running model

---

[6] If you are running example 7.1 using Arena's academic/demo version you realise that a warning/error will pop up after a while declaring that *"you have exceeded 150 entities"* allowed for the academic/demo version of Arena. This is due to the fact that the model is generating too many entities i.e. exceeding the allowance for the demo version. The reason for this error could be due to the parts waiting behind resources to be processed or too many entities being generated. Do not worry - by changing the time between arrivals or reducing the processing time you can solve this at this stage. Later on you will be using the Balking example to monitor the number of entities behind queues and will be able to troubleshoot this sort of problem.

[This was intentional for you to experience this sort of error in Arena]

However, your individual assignments are designed in a way that you should not have this issue at all. Therefore, if you encounter this error whilst completing your individual assignment; this means that there is a mistake in your model. You need to detect and resolve this mistake in your modelling approach.

7.2.4 Viewing the results

For detailed discussions on the various parts of the Arena report refer to section 5.7.4. Notice here that Arena provides on the report view, the name of the project, number of replications simulated and the time units for all time values in the report. This time unit is taken from the "Base Time Unit" field on the "Replication Parameters" tab in the "Run Setup" dialog.

Figure 7.25 shows the Queue Summary data displayed in the simulation report. The report displays all Queues in the model and the time spent by products waiting in each Queue and the number of Entities waiting at each time. If you look at the average values for the waiting time in queue and number waiting in queue, you will notice that the dismantling process has a much longer waiting time and queue length than the other processes. This is obviously a source of concern; either the process doesn't have enough capacity to handle its work or there is a great deal of variability at this process.
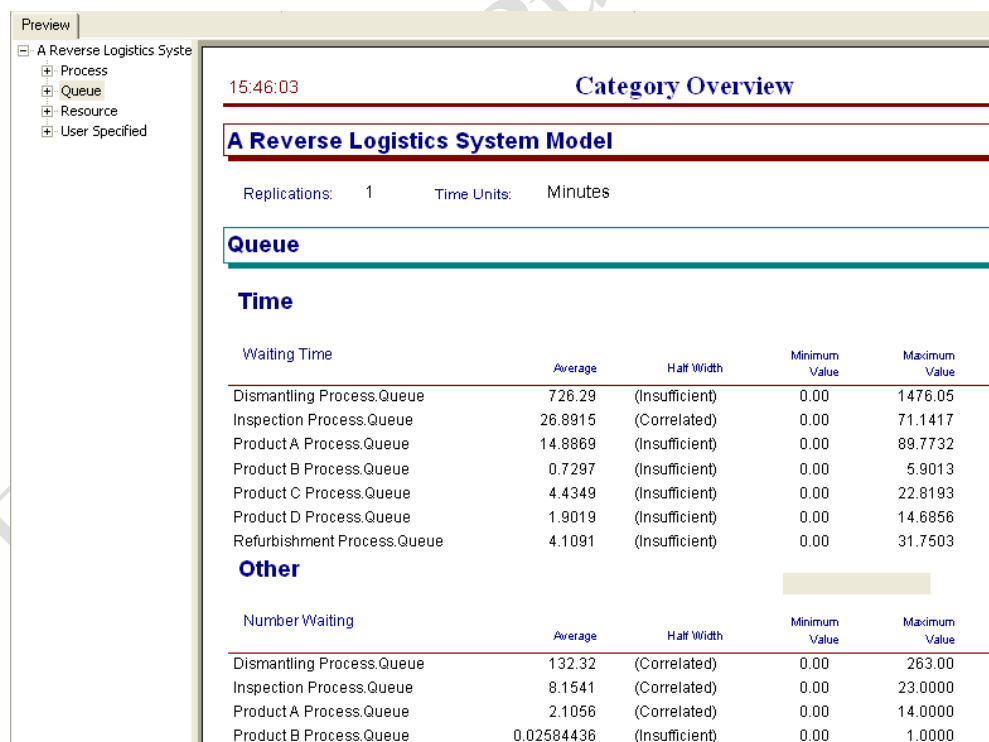
Figure 7.25: Simulation report displaying queue summary data

Remember again that having a result from your simulation model is not the end of the simulation process. Ideally, the next step assuming that your model has been successfully verified will be to validate the model by comparing the results with similar measures in the actual system. Simulation experts even admit for many reasons that true model validation is almost impossible.

Some reasons are that validation implies that the simulation behaves just like the real system, which may not even exist so it's impossible to tell. Even if the system exists, it may not be possible to capture all its complexities in the model hence there is bound to be some variation between model and real system data. An idealistic goal in validation is to ensure that the simulation is good enough so that it can be used to make decisions about the system.

Obviously, the difficulty in validation grows with the complexity of the system being modelled. Thus with our current model, it is pretty easy to validate by just cross checking with information given in the problem description.

Let us therefore assume that as part of this validation process you showed the above results to your manager or client with all the assumptions of running the model for 24 hours a day and only one resource at each process with no breaks during the 24 hours. Your manager's first response is that your assumptions were wrong and makes some suggestions for enhancing the model.

The next step will therefore be to enhance our model by making the necessary changes based on the new information received. This is the subject for the next section on Enhancing the Model.

7.3   Model 8.2: Enhancing the model

Your manager realises that the system actually operates two shifts a day and he suggests having three (3) technicians for the dismantling process during the first shift and four (4) for the second shift to see the impact on the queue at that process.

The manager also noted that there is a failure problem at the inspection process. An inspection device required by the inspector periodically breaks down. Historical data on these failures have shown that the mean uptime (time from the end of one

failure to onset of the next failure) is exponentially distributed with a mean of 180 minutes. The time to repair also follows an exponential distribution with a mean of 10 minutes.

In the next few sections, we will incorporate the above changes into our model with the introduction of some new Arena concepts.

7.3.1 Resource States

The need to model the failure and availability of resources requires an understanding of the concepts of *Resource States and Schedules* in Arena. We will explain the concept of *States* in this section and the *Schedules* in the next section.

Arena automatically defines four *Resourced States* namely, *Idle, Busy, Inactive and Failed.* Thus throughout the simulation period a resource can only be in one of these *States*. Arena keeps track of the time each resource in the system was in each of these *States* in order to report the required statistics. A resource is said to be *Idle* if none of its units has been seized by any entity. That is to say the resource is totally free, doing nothing. On the other hand, as soon as an entity seizes the resource its state is changed to *Busy*, because it is no more free. When the resource is not available to be used, for example a bank staff on break, Arena will set its state to *Inactive*. This is the case when a resource's capacity is reduced to zero (0). Finally, the state of the resource would be changed to *Failed* when he it is not available because of a breakdown.

When a failure occurs, Arena will make the entire resource unavailable and none of its defined capacity can be seized by any entity.

a. Resource schedules

Our initial assumption that the system works 24 hours a day was obviously not right. We will begin to implement the new changes by changing the "Hours per Day" field in the "Run Setup" dialog to 16 hours (Arena will prompt you that some calendar-related features require the hours per day to be 24. Ignore this) to correspond to the two 8 hour shifts in a day. We will also change the "Replication Length" to 10 and the "Time Units" to Days.

To schedule a resource means to define its availability. In Arena, you may start defining a resource's availability either in the Resource or Schedule data module. If you start in one, Arena will automatically make the name of the schedule available in the other. For more discussions on the Resource and Schedule data modules, refer to section 6.1.8 and 6.1.10 respectively.

If you built model 7-1 then open it now and click on the Resource data module in the Basic Process panel. This should display all the resources within the model in the spreadsheet view. We will schedule the dismantling process resource to have capacity of 3 for the first 8 hours and a capacity of 4 for the last 8 hours. Before that, change the "Type" column for the *DisTechnician* to "*Based on Schedule*", enter *Dismantling Schedule* for the "Schedule Name" and select *Ignore* for the "Schedule Rule" column. Your spreadsheet view should be looking as in figure 7.26 bellow. Recall that when the schedule rule is *Ignore*, the resource's capacity is decreased at the set time but the work being done on the current entity will be completed. Note also that these are not the only columns in this view. There are others as would be seen later.

	Name	Type	Capacity	Schedule Name	Schedule Rule
1	Prep A	Fixed Capacity	1	1	Wait
2	Prep B	Fixed Capacity	1	1	Wait
3	Prep C	Fixed Capacity	1	1	Wait
4	Prep D	Fixed Capacity	1	1	Wait
5	Inspector	Fixed Capacity	1	1	Wait
6	RefTechnician	Fixed Capacity	1	1	Wait
7	DisTechnician	Based on Schedule	Dismantling Schedule	Dismantling Schedule	Ignore

Double-click here to add a new row.

Figure 7.26: *DisTechnician* resource based on Schedule

We now need to define the details of the schedule. This could be done by using the spreadsheet schedule editor or by using a dialog option. We will be focusing on the former for now.

Select the *Schedule* data module in the *Basic Process* panel to display the *Schedule* spreadsheet view in the bottom of the screen. Double-click in the spreadsheet view to open a new schedule called *Schedule 1* by default. Click in the "Name" field and select *Dismantling Schedule* from the drop down list. You will have the view shown in figure 7.27.

Figure 7.27: The Schedule module spreadsheet view

Click again on the *Durations* field for that row and to display the Graphical Schedule Editor. The horizontal axis represents the simulation time. Notice that it displays only 16 hours in a day as we specified in the Run Setup dialog. The vertical axis also represents the resource capacity. This is filled in simply by clicking a required capacity and dragging horizontally over the period required. You also erase your selection by clicking on the zero capacity line and dragging horizontally. Figure 7.28 shows the editor filled in for capacity of 3 for the first 8 hours and 4 for the last 8 hours.
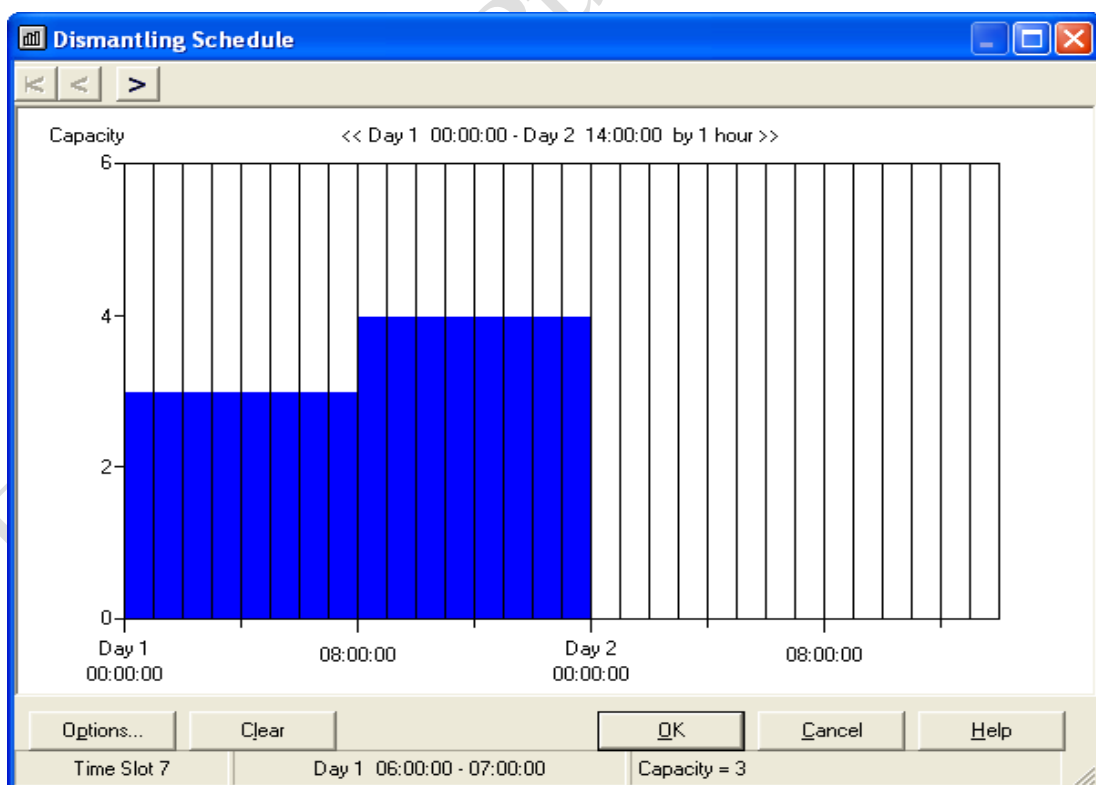
Figure 8.28: The Graphical Schedule Editor

b. Resource failures

Resource failures are defined in much the same way as we defined the resource schedules except that it does not provide a graphical interface. As mentioned in section 7.2.2 above, the complete resource data module's spreadsheet view is shown in figure 7.29.

	Name	Type	Capacity	Schedule Name	Schedule Rule	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Prep A	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
2	Prep B	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
3	Prep C	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
4	Prep D	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
5	Inspector	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
6	RefTechnician	Fixed Capacity	1		Wait	0.0	0.0	0.0		0 rows	✓
7	DisTechnician	Based on Schedule	Dismantling Schedule	Dismantling Schedule	Ignore	0.0	0.0	0.0		0 rows	✓

Figure 7.29: Complete resource spreadsheet view

We want to define the failure of the *Inspector* resource so click on the corresponding field for that resource in the "Failure" column to display its failures spreadsheet view. This displays the view shown in figure 7.30. Enter *Inspector Failure* in the "Failure Name" field and select *Wait* for the "Failure Rule". We choose *Wait* because when failure occurs, the device will complete work on the current entity (or product) before being taken out of service for repairs.

Figure 7.30: Resource failure spreadsheet view

The parameters for the failure are specified in the *Failure Data Module*. This module is found in the *Advanced Process Panel*. Click on the "Name" field and select *Inspector Failure.* Change the "Type" field to *Time* and the "Up Time" and "Down Time" values to EXPO(180) and EXPO(5) respectively. Set both time units to minutes. Your final view should look like figure 7.31.

Figure 7.31: Failure Data Module spreadsheet view

c. Model results

Table 7.1 shows a comparison of the Average Waiting Time in Queue and Average Number in Queue for Model 7-1 and Model 7-2. Recall also, the changes made to Model 7-1 as summarised in table 7.2. The impact of these changes is quite obvious. Generally, all differences may be attributed to the increase in the run length from 32 hours (in Model 7-1) to 160 hours (i.e. 16 hours x 10 days in Model 7-2). Particularly however, differences in results at the dismantling and inspection processes will be understood to be due to the increase in capacity and modelling of failure at those processes respectively.

We realise that the waiting time in queue at the dismantling process had reduced after the capacity was increased in Model 7-2. This does make sense since now more products can be dismantled than in the previous model. On the other hand, the waiting time in queue at the inspection process increased in the new model. This may also be well explained by the fact that the *Inspector* resource was not always available due to periodic failure.

Table 7.1: Queue data for Model 8-1 and Model 8-2

Result	Model 8-1	Model 8-2
Average Waiting Time in Queue		
Dismantling Process		
Inspection Process	726.29	504.94
Product A Process	26.89	39.23
Product B Process	14.89	38.23
Product C Process	0.73	0.75
Product D Process	0.43	3.66
Refurbishment Process	1.90	2.41
	4.10	14.11
Average Number Waiting in Queue		
Dismantling Process	132.32	69.88
Inspection Process	8.15	12.09
Product A Process	2.10	5.04
Product B Process	0.02	0.02
Product C Process	0.43	0.36
Product D Process	0.10	0.12
Refurbishment Process	0.36	1.72

Table 7.2: Difference in parameters for Model 7-1 and Model 7-2

Parameters	Model 7-1	Model 7-2
Hours per Day	24	16 (2 8hour shifts)
Replication Length	32	10 days
Failure at process	None	Inspection
Resource capacities:		(shift 1), 4(shift 2)
DisTechnician	1	1
Inspector	1	1
Prep A	1	1
Prep B	1	1
Prep C	1	1
Prep D	1	1
RefTechnician	1	

7.4   Model 8.3: Adding animations

An important part of a simulation model is the visual display. For presentation purposes, it is useful to make your model's animation look more like the real system before showing the model to decision makers. You may also find the animations very useful for the purpose of model verification (i.e. ensuring the model is working as expected). It helps to easily detect errors in the model logic.

In this section, we will modify Model 7-2 into Model 7-3 by adding animations for entities, resources and some dynamic plots. We will design the animations in a different environment from the model logic. Our complete animation captured during runtime is shown in figure 7.32.

Your final animation may not look exactly like ours since we will not take you through every detail of how we did it but just the main steps and leave you to try figuring out the rest yourself.
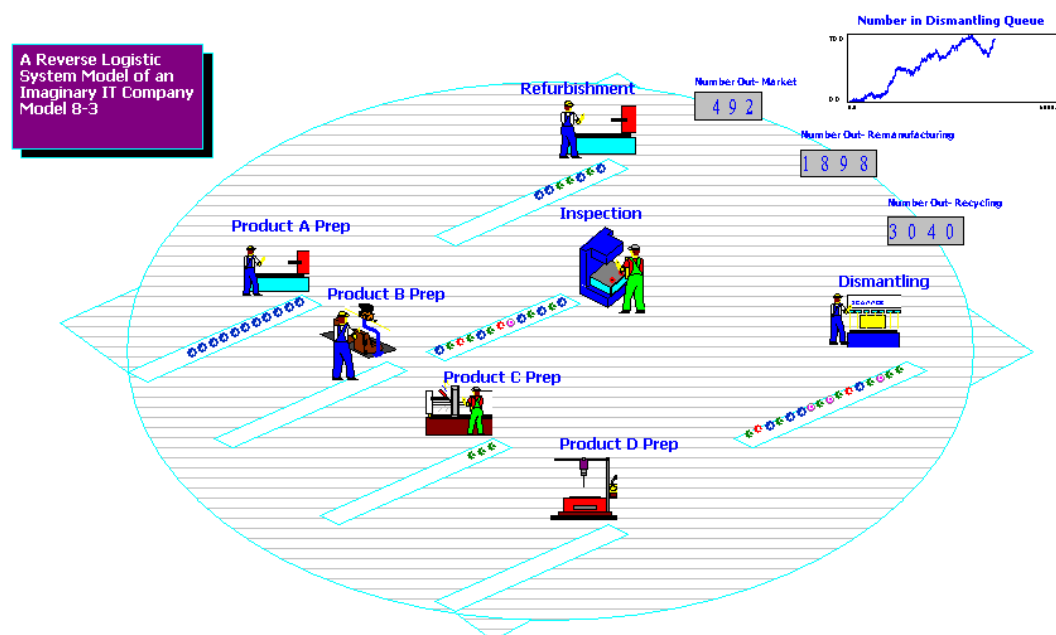

Figure 7.32: Final animation of Model 7-3

To start with, open Model 7-2 and scroll down to a blank Model Window. You may want to copy our style by laying an ellipse over a square as we deed. To do this make sure you have your "Draw" tool bar displayed. If not, right-click on any toolbar and choose "Draw" from the pop-up list. It looks like figure 7.33 bellow. Now try drawing the shapes using the "Polygon" and "Ellipse" buttons and changing the "Line Styles" and "Fill Patterns".
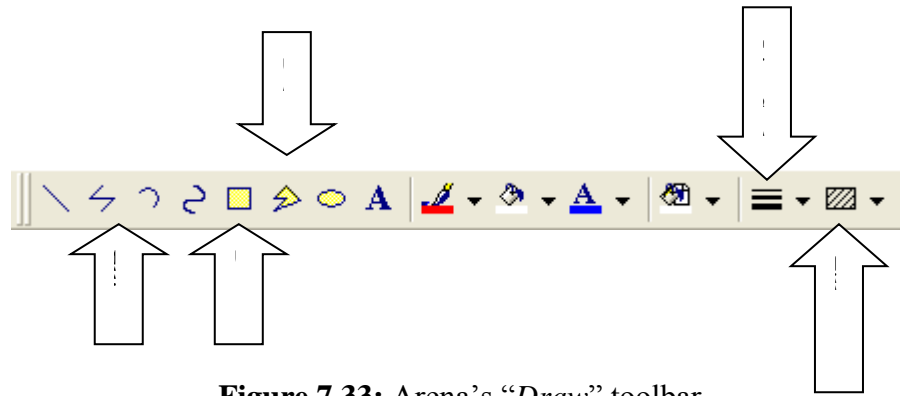
Figure 7.33: Arena's "*Draw*" toolbar

Before we start talking about entities and resources, let's quickly look at how to animate the queues in the model. You might have noticed by now that Arena automatically animates queues whenever you use a module that has an in-built queue for example the process module. When we want to animate the entire system as shown in figure 7.32, we need to be able to move the queues wherever we wish. Fortunately, Arena makes it possible to cut the queue objects from the modules and paste them where needed. That's exactly what we did as shown in figure 7.34. We cut the queue from the *Product A Process* module and pasted in our animation the way we want it.
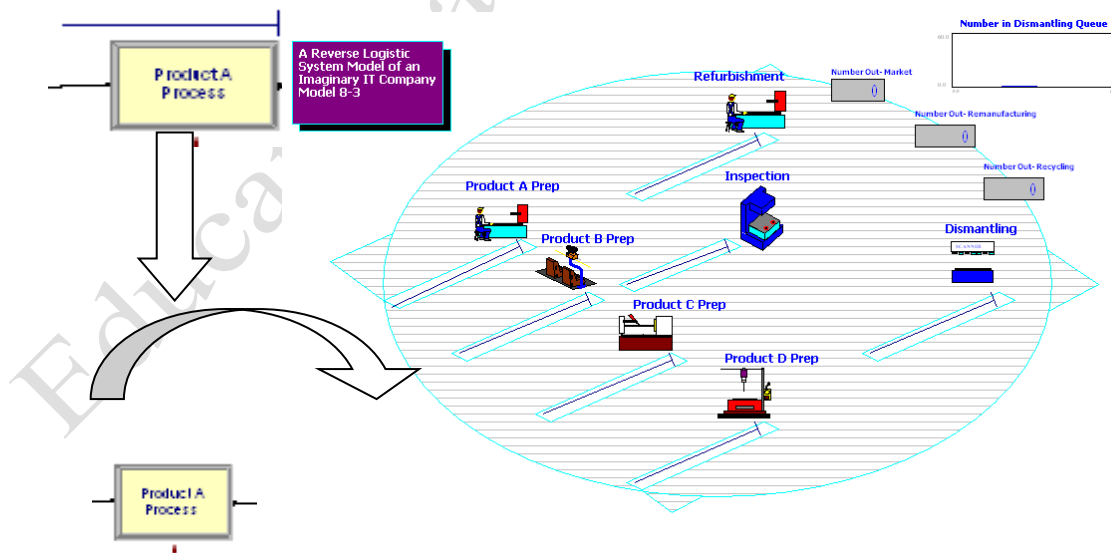
Figure 7.34: Queue animation by cutting and pasting

Once you paste your queue where needed, you may change its shape, length and orientation by clicking and dragging its handles. You may also make changes to its parameters by editing its dialog when you double click on it. This dialog is shown in figure 7.35.
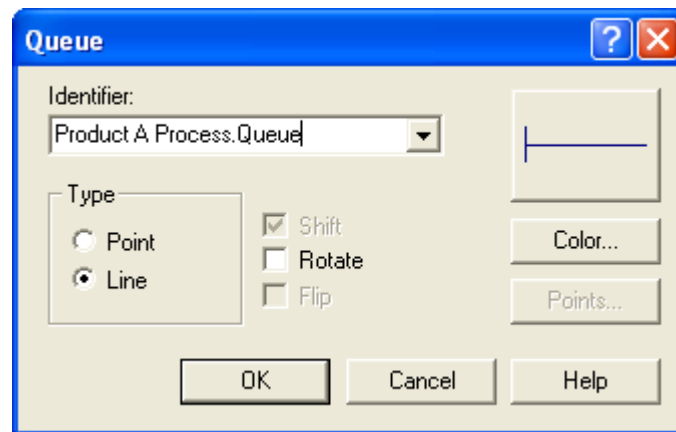

Figure 8.35: The Queue dialog

7.4.1 Changing entity pictures

We create and edit entity pictures in Arena's entity picture placement window accessed by selecting "Entity Pictures" from the Edit menu. A snap shot of this window is shown in figure 7.36 bellow. The left side of the window represents all the entity pictures currently available for use in a model whilst the right side represents one of Arena's picture library files (machines.backup.plb).

The "Add" button allows you to create your own entity picture. It basically provides a blank picture space for you and double-clicking this would then open a picture editor where you may create your entity picture. The "Copy" button also makes a copy of an existing picture which you make changes to, and yet preserve the original. The "Delete" button will only remove a selected entity picture from the list. What we have done is to represent our products A through D with coloured balls with the corresponding letters on them. To do this, we copied the existing balls in the list, double-clicked to open the picture editor and placed the letters on top of the balls.
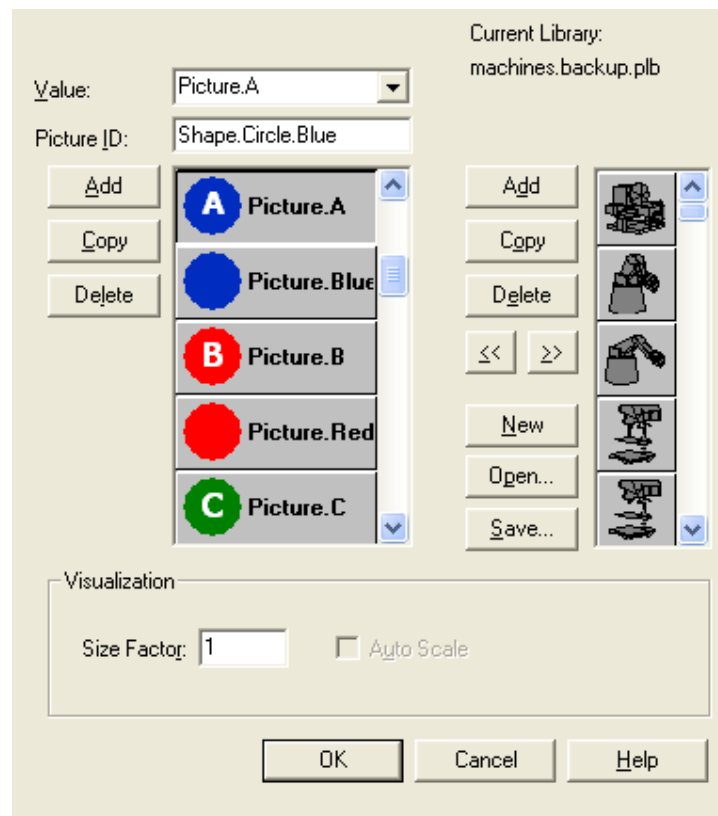
**Figure 8.36:** The Entity Picture Placement window

The corresponding buttons on the right hand side have the same functionality and entity pictures may be moved between both sides by clicking on the entity picture to move on one side and the destination location on the other side and then using the arrow buttons (<<,>>) to perform the move.

Arena Provides a picture ID and value or name which we changed to Product "A" etc. The names you specify here would be made available in the list of available pictures when you are assigning pictures in the *Assign Module*. You may also do the assignment by changing the initial picture in the *Entity data Module* to the name you gave to your entity. Now try to create your entities and assign them before we be begin to look at resource pictures.

7.4.2 Adding resource pictures

In order to add a resource picture to your animation in the "Animate" toolbar first click on the *Resource* button (). The *Resource Picture Placement* window (figure 7.37) will be displayed. This is similar to the entity picture placement widow we just discussed and the buttons have similar functionality. Recall our discussion in section 7.2.1 on the resource states. Arena allows you here to specify four different resource pictures to represent each of the four possible resource states (*Idle, Busy, Inactive and Failed*).

Arena will always make the list of all resources in your model available when you click on the identifier combo box as shown. Notice that you can move pictures from the library on your left to the states list by using the arrow buttons as in the entity placement window discussed.
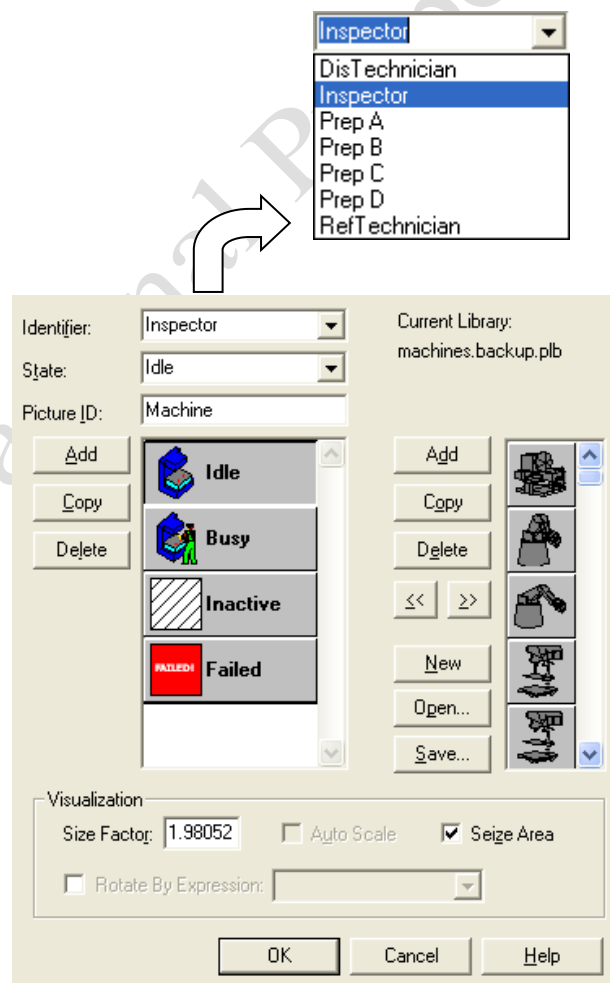
**Figure 7.37:** The Resource Picture Placement window

When you accept the resource picture by clicking the OK button, your pointer will be turned into a cross hair. Position this cross hair in the position where you want your resource to appear and click. This places the new resource in your animation. You may then drag its corner handles to resize it and click on it to change its position.

In the same way, add animations for all your resources arranging them in your animation environment as we did or in your own way.

7.4.3 Adding variables and plots

To complete this current model, we will now add some variables and plots to our animation. As shown in figure 7.32, we will add variables for the number of products going out into the market after refurbishment, number out to remanufacturing and number out to recycling. We will also add a plot for the number in queue at the dismantling process.

The *Dispose* module keeps track of the numbers of products going out and has a default animation of these variables next to the module shapes. Copy these variable animations from each of the three *Dispose* modules and paste them in your animation environment as we did if figure 7.32. You may resize the variable by highlighting it and dragging its handles. You may also reformat the text by double clicking on the variable to display its Variables dialog. An alternative and more general approach is to click the "Variable" button (🔢) on the "Animate" toolbar to open the "Variable" widow as shown in figure 8.38.
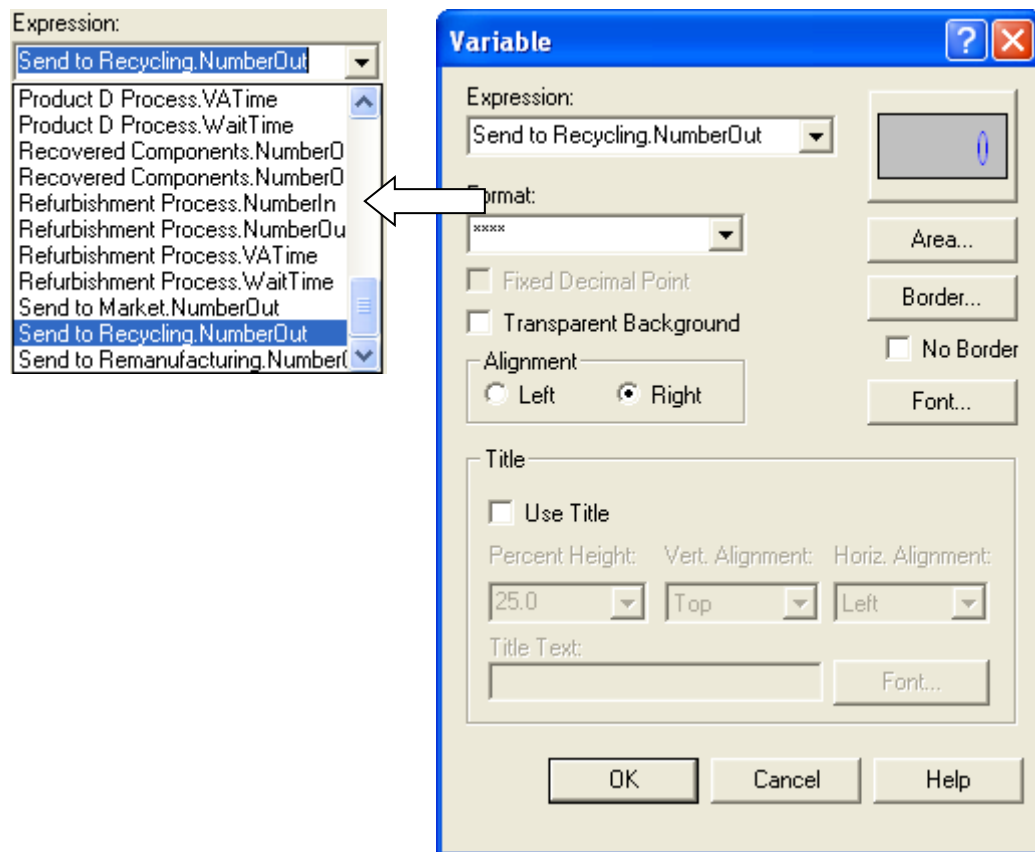
Figure 7.38: The Variable window

When you click on the expression field, Arena displays a list of all the expressions in your model that you may animate. You may also create your own expression by right-clicking the field and selecting the "Build Expression" option. This will open a new window where you may build the expression you desire.

Finally, let's add a plot for the number of products in queue at the dismantling process. Click the "Plot" button (🖼) on the "Animate" toolbar to display the plot window shown in figure 7.39. Clicking the "Add" button further opens the "Plot Expression" window which allows you to select or build the expression (s) you wish to plot. In our case we selected the number in queue for the dismantling process (i.e. NQ (Dismantling Process.Queue)) as shown.

We set the maximum to 60 hoping that the queue length will not be more than that. We set the "History Points" to 5000 and the time "Time Range" for our display to 9600 base time units (minutes in our model, check this in Run Setup dialog). When

you are done, click the OK button and with the cross hair pointer, click the desired location in your animation environment to display your plot (See figure 7.32).
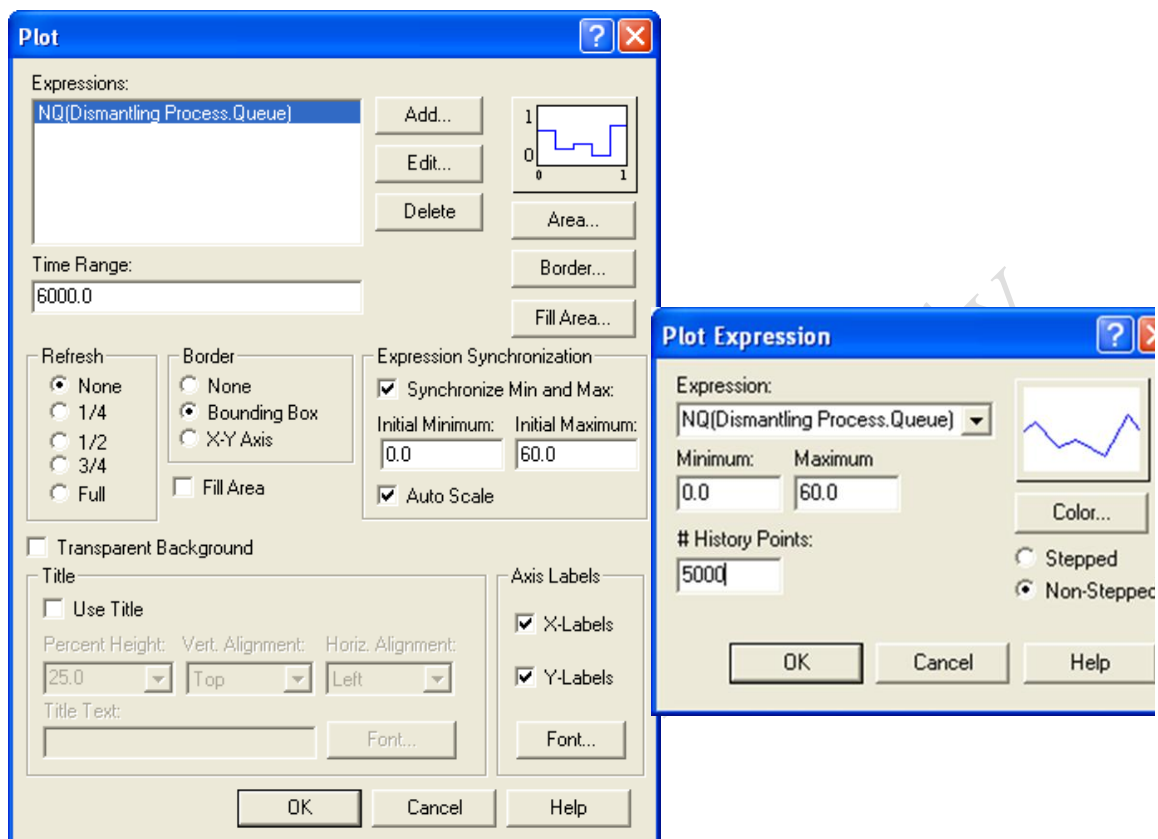
Figure 7.39: The Plot window

7.5   Model 7.4: Entity Transfers

We have so far been gradually building our imaginary reverse logistic model by trying to make it more and more realistic. Up until now, we have assumed that entities (or products) in our model move from point to point without time delay. That is to say they disappear from one point and appear at the other point. Well this obviously does not happen in any real world system but that is what happens when you connect your modules with the connector.

In this section, we will introduce two new Arena concepts that would enable us to model entity transfers more realistically by specifying the time it takes to transfer entities from point to point in the system. After discussing these new concepts of *Stations* and *Transfers,* we will then add further enhancements to Model 8-3 to create Model 8-4.

Let's assume here again that all entities take 5 minutes to move between various processes irrespective of the difference in distances.

7.5.1 Stations

Stations in Arena correspond to physical or logical locations in a system where processing occurs. Thus in our current example, all the Prep Processes, Inspection, Dismantling and Refurbishment locations may be referred to as stations. Stations may also be used to represent locations for product arrivals and departures as will be seen in this example. Each station in a model is assigned a unique name that can be referenced from any point in the model as a destination for entity transfer.

Arena provides a special flowchart module called *Station* for modelling this concept. This module may be used to define a single station as well as a set of stations. In this example however, we will only present the single station application.

Recall from section 7.2.1 that we initially divided the entire modelling problem into the following steps:

6. Create arrival of products

7. Send products through prep process

8. Send products through inspection process

9. Decide where each product goes after inspection

10. Send part to refurbishment

11. Send another part to Dismantling

12. Dispose remaining part to Recycling

13. Dispose to market after refurbishment

14. Split products into components after dismantling

15. Dispose recovered components after dismantling

16. Dispose after dismantling to recycling

From these we have derived the following stations to facilitate the transfer of entities in our model:

1. Product A arrival station

2. Product B arrival station

3. Product C arrival station

4. Product D arrival station

5. Prep A station

6. Prep B station

7. Prep C station

8. Prep D station

9. Inspection station

10. Refurbishment station

11. Dismantling station

12. market station

13. Remanufacturing station

14. Recycling station

With these we will be able to send any entity (or product) in to system to any of the stations by using the *Route* module and specifying the unique identifier (Name) of the station.

The *Station Module* shape and dialog is shown in figure 8.40 below.
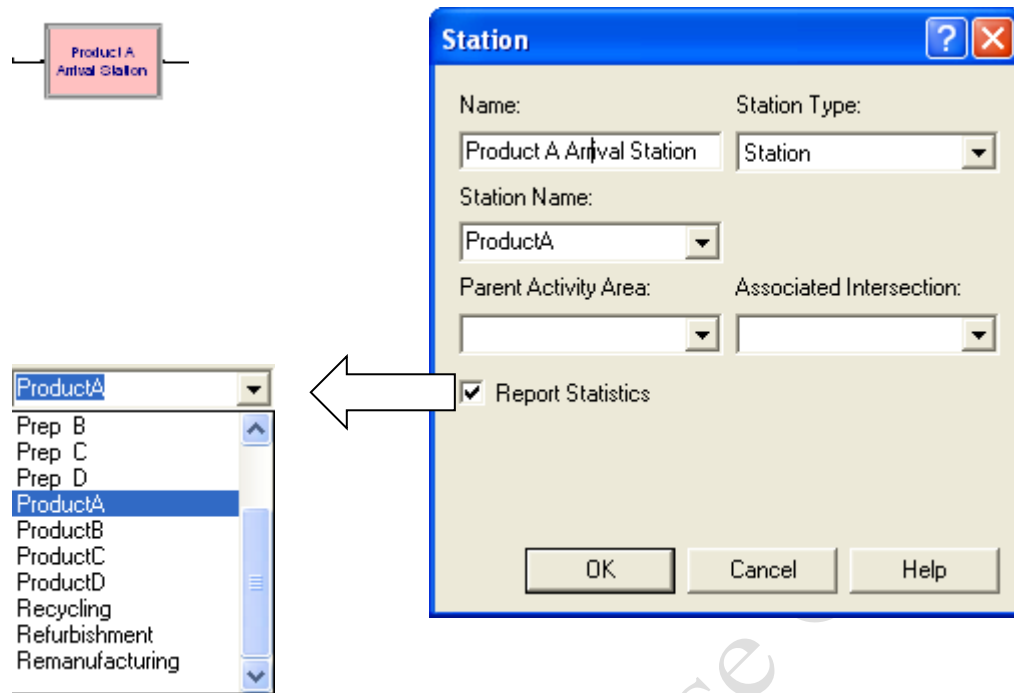
Figure 7.40: Station Module shape and dialog

7.5.2 Routes

The Route module transfers an entity to a specified station, or the next station in the station visitation sequence defined for the entity. A delay time to transfer to the next station may be defined.

When an entity enters the Route module, its Station attribute (Entity.Station) is set to the destination station. The entity is then sent to the destination station, using the route time specified.

The *Route Module* shape and dialog are shown in figure 7.41 below. The module's "Name", "Route Time" and "Units" fields are similar to those already discussed in other modules. When you click on the "Destination Type" field, Arena gives you a drop-down list of various ways of specifying destinations as can be seen in the figure. In this example, we will only use the *Station* option and this requires that we specify the name of the station in the "Station Name" field.
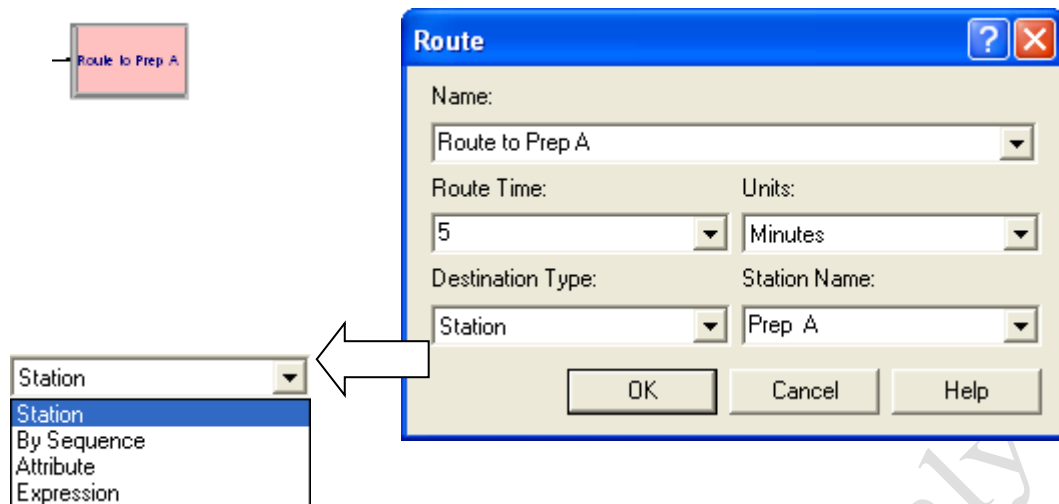
**Figure 7.41:** Route Module shape and dialog

Let us update Model 7-3. Note that *Station* and *Route* modules are found in the "Advanced Transfer Panel" in Arena. If you don't this panel, then attach it now by going to File menu, Template panel, Attach and look for the file *AdvancedTransfer.tpo*

Now open Model 7-3 and let's begin to modify it. Remove the connectors after each of the *Assign Modules* and add a *Station* and a *Route Module* to each. Before we begin to define our stations, you should note that our addition of stations and routes will affect both the model and the animation. For example if we want the animation to show the products arriving at some point before being sent to the Prep areas, then we should define that point as a station which we call in this case "Product A Arrival Station" for all product "As". Now double-click on the station module you have added to update its parameters. We gave this module the name "Product A arrival Station", set the "Station Name" field to "ProductA" and left all other fields to their default values. In a similar way, double –click on the route module and set its "Name" field to "Route to Prep A", "Route Time" field to 5, "Units" field to "Minutes", "Destination Type" field to Station (default value) and "Station Name" to Prep A. Your completed station and route modules should look like figure 7.42.
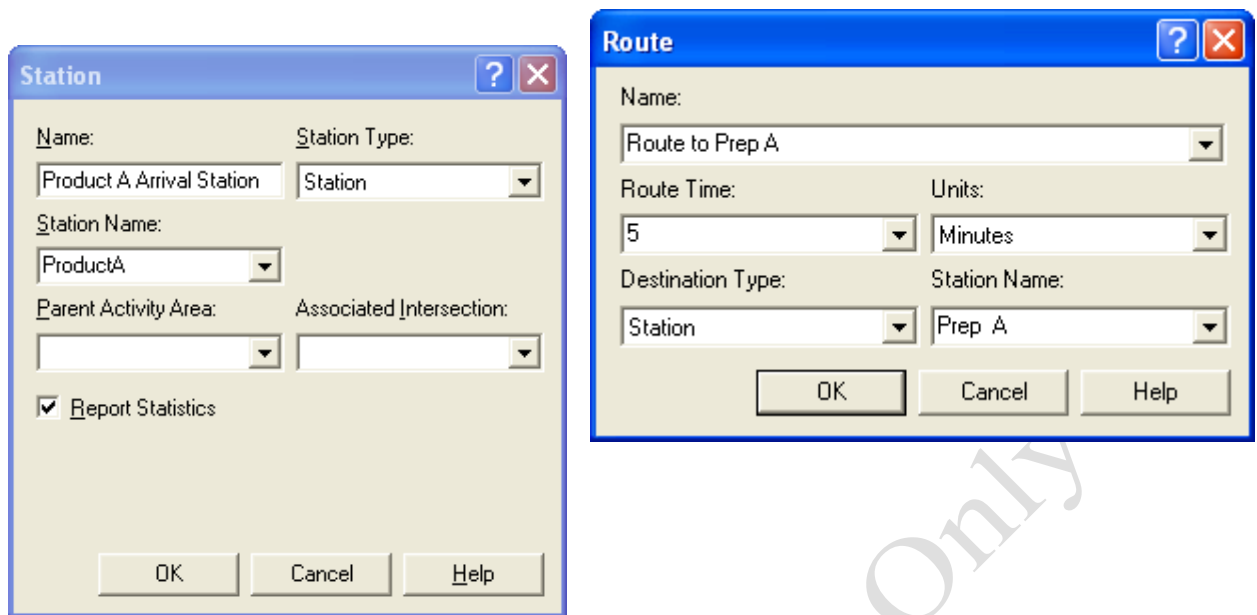
**Figure 7.42:** Station and Route dialogues for Product A

Continue to update the station and route modules you have added to the remaining assign modules as above. Remember use the product letters (B, C, and D) respectively in place of A when updating the remaining modules. When you are done with this part of your model, it should be looking like figure 7.43.
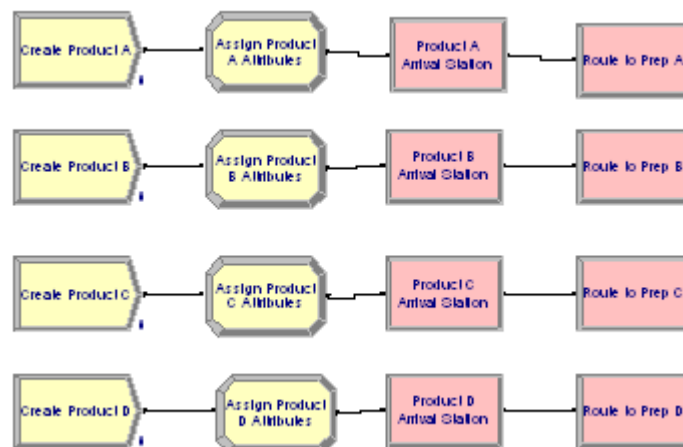

Figure 7.43: Product Arrival stations

We will next look at the Prep areas. We want to define each Prep area as a station since they are different physical locations. Hence we define four stations for Prep A through D. However, since all the products will go to the same inspection station after their Prep processes, we require only one route module to send all of them there. To start with, add a station module each to each of the product A through D processes and one route module to the right of the process modules. Update the station modules as before with names Prep A Station etc and station names as Prep A, Prep B etc. The only things we will change for the route module is the module name and destination station name. Thus "Route to inspection" and "Inspection" respectively. This part of your model when completed should look like figure 7.44.
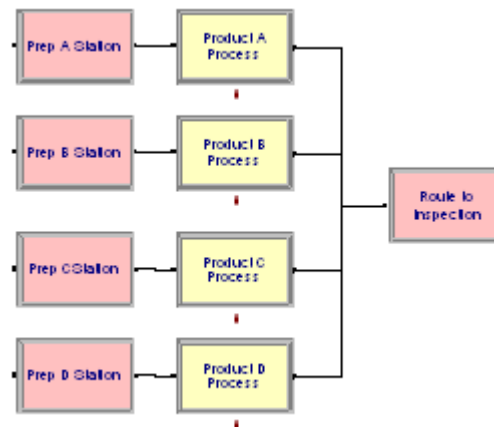

Figure 7.44: Prep stations

All we have done so far is to break our model down at various locations and add a station module to define the location and a route module to transfer the product from that location to another after processing is finished.

Thus the resulting logic for the inspection, refurbishment, and dismantling stations are shown in figures 7.45 and 7.46 respectively. What you should also note is that, as your define your stations, Arena keeps a list of them and would give you a drop down at any point you need to define a station or select a previously defined one.
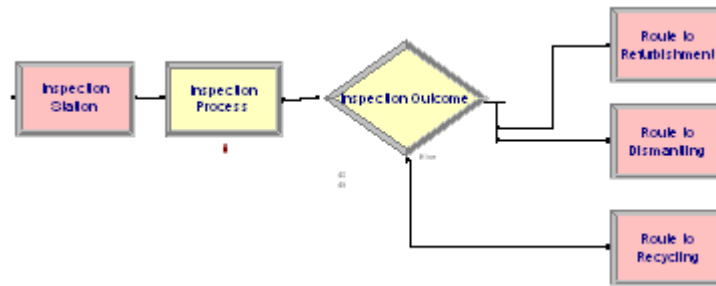
Figure 7.45: Inspection station
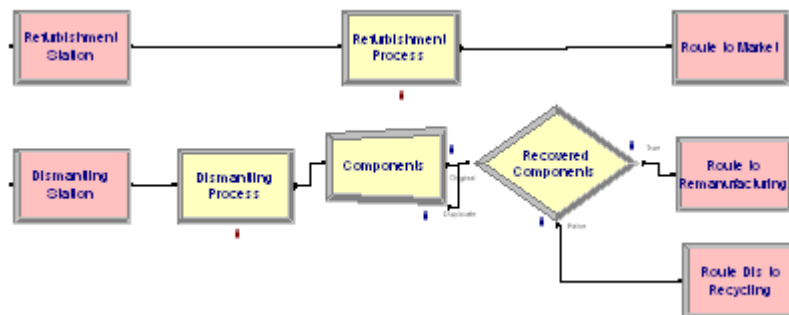
Figure 7.46: Refurbishment and Dismantling stations

You may realise by now that there are only three exit points in our model. That is the products are either sent to market, sent for remanufacturing or to recycling. We have again modelled these points as stations mainly for the sake of animation. We want to be able to see where the exits are located in our animation and the products moving there after processing. The logics for these are shown in figure 7.47.
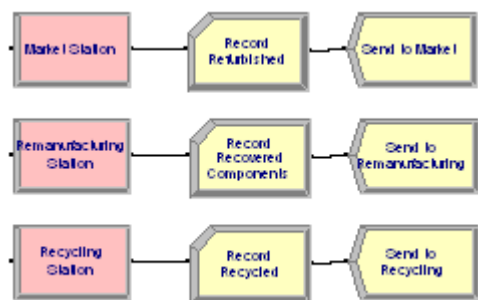

Figure 7.47: Market, Remanufacturing and Recycling stations

It may seem a lot of work adding all these station and transfer modules but it is also very important to make the animations look as realistic as possible. In a real project, it is more convincing for a client to see an animation that very closely depicts his or her system. We will now begin to update the model's animate in the next section.

7.5.3 Animation enhancement

Station animation is quite straight forward. You will need the "Animate Transfer" toolbar to be able to proceed. If it's not displayed in your project bar then, right-click on the toolbar and select the "Animate Transfer" icon from the pop-up list.

In order to animate a station, click on the station button (⊟) on the toolbar to display the station dialog as shown in figure 7.48. Click again on the "Identifier" field to display a list of all stations in your model, and then select the name of the particular station you want to animate at this instance. Leave the "Auto Connect" options to the default "None" value and click OK. You pointer will then turn into a cross hair. Click the desired location in your animation environment to place the station. This will have the shape shown in figure 7.47. Repeat this process to place animations of all the stations in the model in your environment.
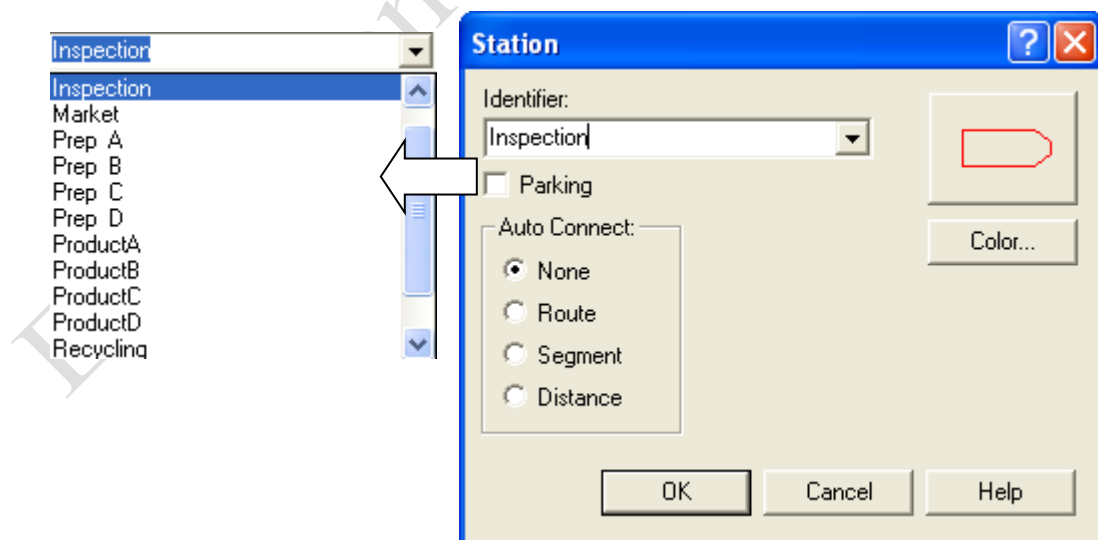
Figure 7.48: Animate Station dialog

After placing all your stations, the next thing will be to connect them with the route animations. Again this is quiet straight forward. Click on the route button ( ) on the "Animate Transfer" toolbar to display the route dialog shown in figure 8.49 below. For now let's just leave all the parameters to the default settings and click OK. With the resulting cross hair pointer, click on your starting station (e.g. Product A arrival station) then mover your pointer to the finishing station (e.g. Prep A station) and then click again. You should now see the path between the two stations which the entity in transit will follow. Note that this path can be redirected by clicking several points in a desired direction before finally clicking the finishing station.

Now, repeat the above process again to put a route between all your station animations. Note that you only need a route between stations for which you have specified an entity transfer in your model logic. It is also important to know that a route from say station "A" to station "B" is different from that from station "B" to station "A". That is if there are entities moving in both directions then you have to animate routes for both directions.
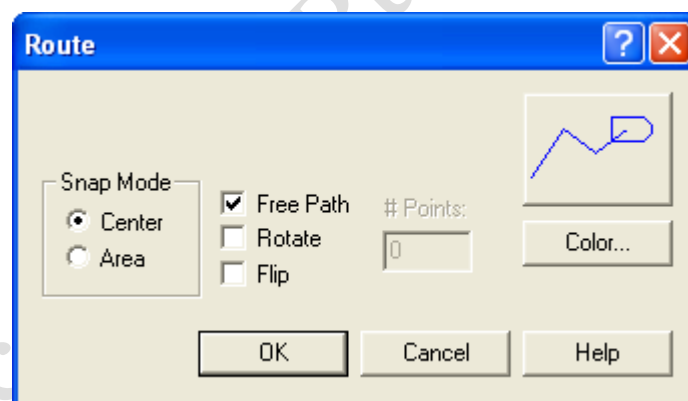
Figure 7.49: Animate Route dialog

When you have properly placed all your station and route animations then your final animation view should be looking something like figure 7.50. Figure 7.51 shows a run time snap shot of the completed model.
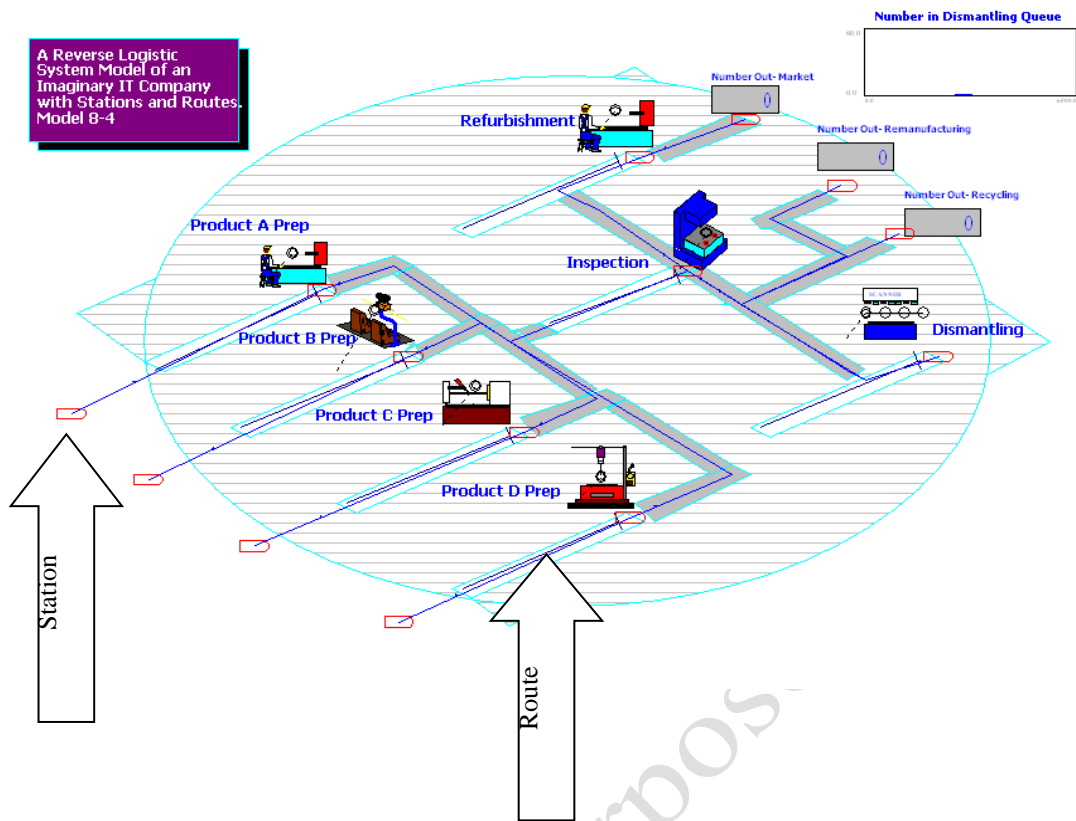
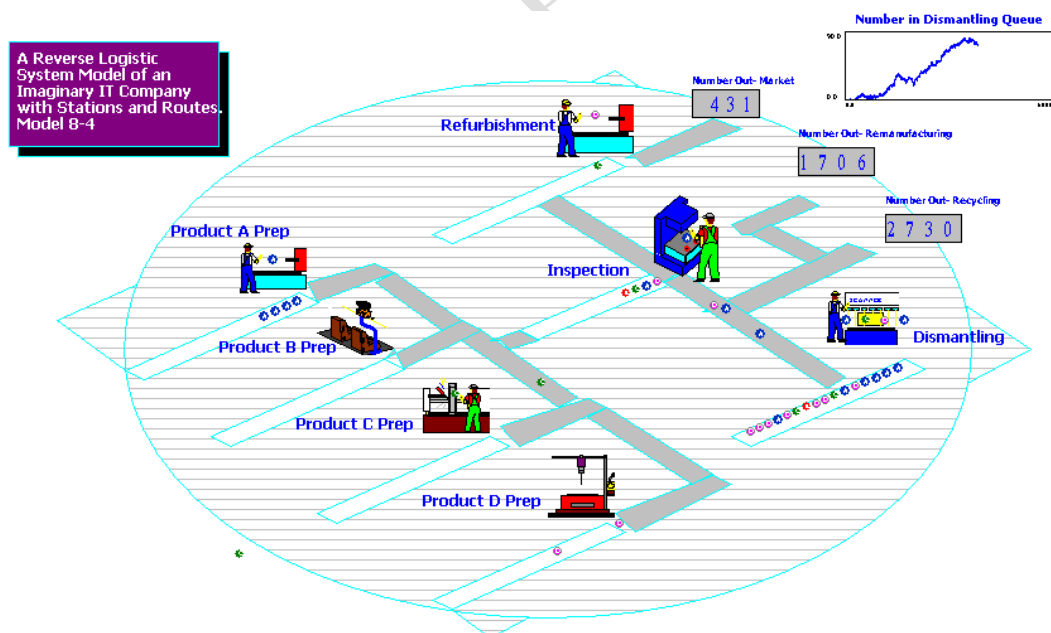**Figure 7.50:** Station and Route animations

Figure 7.51: Final running model

Be reminded again that there is more to simulation modelling than just using Arena. In this chapter, we have tried to take you through some of the key stages of

simulation but with more focus on the use of Arena. We identified a problem, formulated the problem, developed a modelling approach and went through the modelling process step by step. All the data required for this modelling work were provided but bear in mind that this would not be the case in a real life problem. You may have to identify what kind of data you will need to model your system and the work out how to collect such data.

There are many more features and concepts in Arena that cannot feasibly be covered in this course. However, if you have been following very closely and have taken in all the material in the last three chapters, then you should able to build models with considerable detail. I especially recommend that you follow the same line and bring more features such as *Transporters* and *Conveyors* into your models.

What you may have to do is to practice more, read more of the reference text and also consult the Arena help files in order to further develop your modelling skills.

In addition to this book I highly recommend my lecture notes on http://people.brunel.ac.uk/~emstaam website with extensive examples both from this book and other references. We also have some real world case studies on the site.

Chapter Reference

[1] Dekker, R., Flieschman, M., Inderfurth, K., and Van Wassenhove, L. (2004), Reverse Logistics for closed-loop supply chains, *Springer Verlag.*