APPLICATION OF THE OBJECT-ORIENTED PARADIGM TO THE MODELING OF A CONSTANT SPEED, DISCRETELY SPACED, RECIRCULATING CONVEYOR SYSTEM

By

SALOUA SMAOUI

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE

July, 1992

APPLICATION OF THE OBJECT-ORIENTED PARADIGM
TO THE MODELING OF A CONSTANT SPEED,
DISCRETELY SPACED, RECIRCULATING
CONVEYOR SYSTEM

Thesis Approved:

Manjunath Kamath
Thesis Adviser

Joe H. Mize

M. P. Lenell

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGEMENTS

As I conclude this research work I would like to thank each of the members of my committee for their impact on my education and research. My major advisor, Dr. Manjunath Kamath for his intelligent guidance, Dr. Palmer Terrell for encouraging me to choose such a challenging research work, and Dr. Joe Mize not only for his guidance and advice throughout this study, but also for the support, understanding and friendship he has offered me during my difficult times. I would also like to thank Dr. Mize and Dr. Kamath for the giving me the opportunity to be part of the CIM center at Oklahoma State University eventhough I was doing my own research. I would like to extend my special thanks to Hemant Bhuskute for his great technical help.

I would like to thank the Scientific Mission of Tunisia for their continuous financial support throughout my academic career. I also would like to extend my appreciation to the School of Industrial Engineering at Oklahoma State University for offering me a teaching assistantship during my graduate studies.

I would like to express my deepest appreciation to all my friends, especially the ones in Stillwater for holding my hand and walking with me through the darkness during my hardship.

I would like to thank my sisters Sihem, Leila, Rakia, and their families, my brother Imed for their unending support and belief in my capabilities, and the family's friend Noureddine for his great and loyal friendship to the family.

Last, but not least, I would like to dedicate my modest work to the three most precious people in my life: my brother Ahmed for being such a great second father and the point where we all converge to stay as united as possible; my mother Aicha for her

love, support, and understanding; and my father and my best friend Mohammed Salah whom I lost unexpectedly during this study. Dad I owe you all what I am and all what I will be. I have big faith in God that eventhough I cannot see you, I know that there is a big proud smile on your face as I am finishing this work that meant the most to you. I miss you dad very, very much!.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

With the fast evolution in manufacturing processes, needs, and objectives, the integration of computers in today's industrial environment emerged smoothly and rapidly. The computer had to be exploited to the maximum. It was found to be the savior of the modern manufacturing. In fact, it has been used in a variety of tasks. From data storage to fully automated process monitoring, the computer showed an extraordinary flexibility in solving many engineering problems and concerns. Computer Integrated Manufacturing (CIM) systems, although still conceptual, seem to be the next stage in the manufacturing evolution.

Pritsker defines computer simulation as the process of designing a mathematical-logical model of a real system and experimenting with this model on a computer (Pritsker 1986). Another definition of simulation by Mize and Cox (as referenced in Turner, Mize, and Case 1986) states that "simulation is the process of conducting experiments on a model of a system in lieu of, either (1) direct experimentation with the system itself or, (2) direct analytical solution of some problem associated with the system". In order to understand these definitions we need to explain the terms, system and model.

A system is a collection of interdependent elements which work cooperatively for the purpose of achieving a common goal. Frequently, a system is characterized by random, but statistically predictable, behavior. A model is a representation of a system. If the model is expressed mathematically as a set of logical and functional relationships, it is referred to as an abstract model. A computer based model is an abstract model

implemented on a computer upon which experiments are conducted for the purpose of generating information useful in making decisions. Both definitions above agree that simulation allows drawing conclusions about the system, without building it disturbing it, or destroying it. Thus, a simulation model is very useful in both design and analysis of a manufacturing system. In addition to being a manufacturing system planning and design tool, simulation is currently being used for production planning and shop floor scheduling. This involves testing a variety of input conditions on up-to-date factory models for satisfactory output results.

A simulation of a manufacturing system, or even one part of it, can be a very challenging and complex task. In fact, the external factors that can influence such a system, and consequently any contingent decision, are enormous. Figure 1 is an illustration of a manufacturing system and its standing in the environment. Manufacturing systems in the future have to be reconfigurable to be responsive to dynamic changes in the environment. Simulation modeling should be easily updated and highly modular (changes to a model should be localized) (Beaumariage, 1990).

With the recognition of the importance of computer systems in improving manufacturing productivity, there is a pressing need for good software modeling approaches to support efficient design and control of manufacturing systems. Software design concepts based on Object-Oriented Programming (OOP) are emerging as powerful techniques for developing large scale software systems. This research presents important features of object-oriented computing and the relevance of such an approach in modeling and developing software for manufacturing systems such as a constant speed, discretely spaced recirculating conveyor system.

The evaluation and comparison of OOM features (through the design and implementation of a prototype OOM system) to traditional modeling approaches should provide greater impetus for the development of commercial OOM capabilities and for

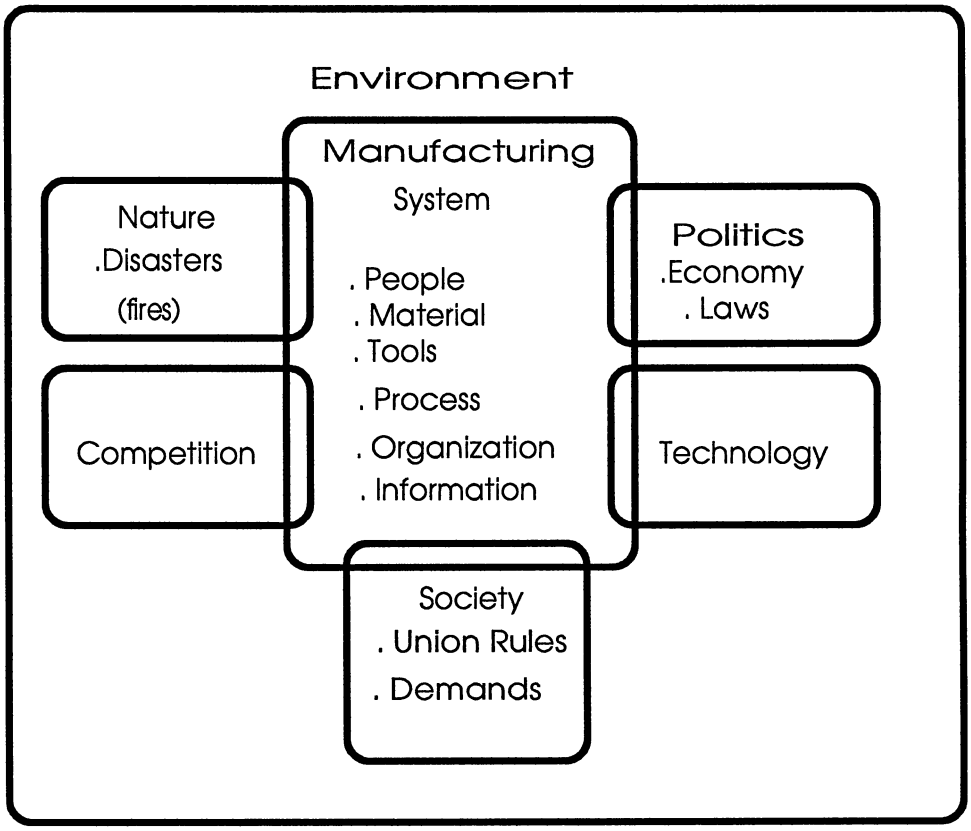

Figure 1. A Manufacturing System

simulation practitioners to pursue the use of the new and beneficial approaches to modeling.

The availability of the advanced development environment present in the Smalltalk-80 programming system in conjunction with the application oriented discussions pursued in the Center for Computer Integrated Manufacturing result in a favorable environment within which to pursue this research activity.

CHAPTER II

PROBLEM STATEMENT

Introduction

So far, one of the biggest drawbacks of simulation modeling is that models are typically constructed as single use models. That is, once used for its original purpose, a particular model is rarely used again. When a new problem is encountered, a new model is generated from the beginning even though it may include elements contained in earlier models. The cost of this approach, measured in both dollars and hours, causes many to question the value of using simulation to model large complex systems (Pratt, 1992). One can ask why is simulation still gaining popularity as a decision making tool in today's highly complex systems. The answer is that it is clear that it is not due to the increased power of simulation but due to the increased consensus on the inappropriateness of analytical tools and increased computer literacy among decision makers. Analytical models employ techniques from stochastic processes and queueing theory to study system performance. They are generally the most efficient method of investigation if they are applicable. They frequently yield explicit information about the functional form of the relationships among system variables and, under some circumstances, indicate whether a unique optimal solution exists. Unfortunately, some real systems of interest are so complex that formulating and solving an exact analytical model is either extremely difficult or impossible (Pratt, 1992).

Simulation modeling overcomes many of the disadvantages inherent in analytical modeling. Unfortunately, as stated above, one of the big disadvantages of simulation is

the high expenditure of time and money. Therefore, in order for simulation to continue its popularity (by reducing the time and money required by its implementation) it has to address some requirements such as:

. High level of software reusability,

. Software modularity,

. Ability to implement urgent, detailed models,

. Low degree of abstraction,

. A graphical interactive development environment, and

. Ease of analysis of results.

These requirements are currently being sought by the so called revolutionary approaches to simulation. These approaches are based on new programming paradigms and knowledge representation methods along with new perspectives in viewing and analyzing systems (Karacal, 1991). Within this class, the major interest areas are: Object-Oriented Programming (OOP), Logic Programming and Expert Systems, Distributed Simulation, and Knowledge Based Simulation (KBS).

Research Motivation and Goal

The purpose of this research is to illustrate the applicability of a revolutionary approach to modeling manufacturing systems through the development of a prototype environment to model a constant speed, discretely spaced, recirculating conveyor system. A comparison between the traditional and revolutionary approach to simulation will be presented. Since, an object-oriented modeling (OOM) environment is under development within OSU' s Center for Computer integrated Manufacturing, the revolutionary approach to simulation will be studied through the area of OOP. The

traditional approach will be represented by the simulation language SLAM II. The comparison will be illustrated using an Analytic Hierarchy Process (AHP) model.

Description of the Target System

A particular conveyor system was selected to initiate the design and development of the simulation model. The recirculating conveyor system originally chosen to initiate this development is a sub-floor towline conveyor that moves finished goods to several unload and load centers in a 500,000 square feet warehouse. The towline is a sub-floor towline made up of an endless chain running in the floor with hooks spaced every twenty-one feet along the chain. At any time a hook may or may not be pulling a cart along with it. There are 380 carts in the system at all times (Terrell, 1977).

Figure 2 shows the floor plan diagram of the conveyor system. The conveyor delivers the manufactured goods to storage or rail and truck docks. Incoming goods at the rail and truck docks are delivered to storage. The main conveyor loop is indicated by the dashed line. This loop is 5600 ft. long and the speed of the towline is 70 ft. per minute. There are 20 destinations around the loop and a cart may be programmed to any of these destinations. Carts are programmed manually by moving the magnet tipped probes at the front of each cart. The carts will always arrive at their destinations via the shortest route. The non-powered spurs at the end of the storage aisles constitute the various unload and load stations and a cart arriving at one of these as its destination is side-tracked into one of the non-powered spurs and a following cart pushes it deeper into the spur.

A loaded cart joins the load station queue and waits on the unloading facility. Carts are unloaded on a first come first-served basis. Upon completion of unloading, the unloaded cart may be placed on the towline and assigned a new destination. Sometimes it may be desirable to retain a certain number of empty carts in an unload and

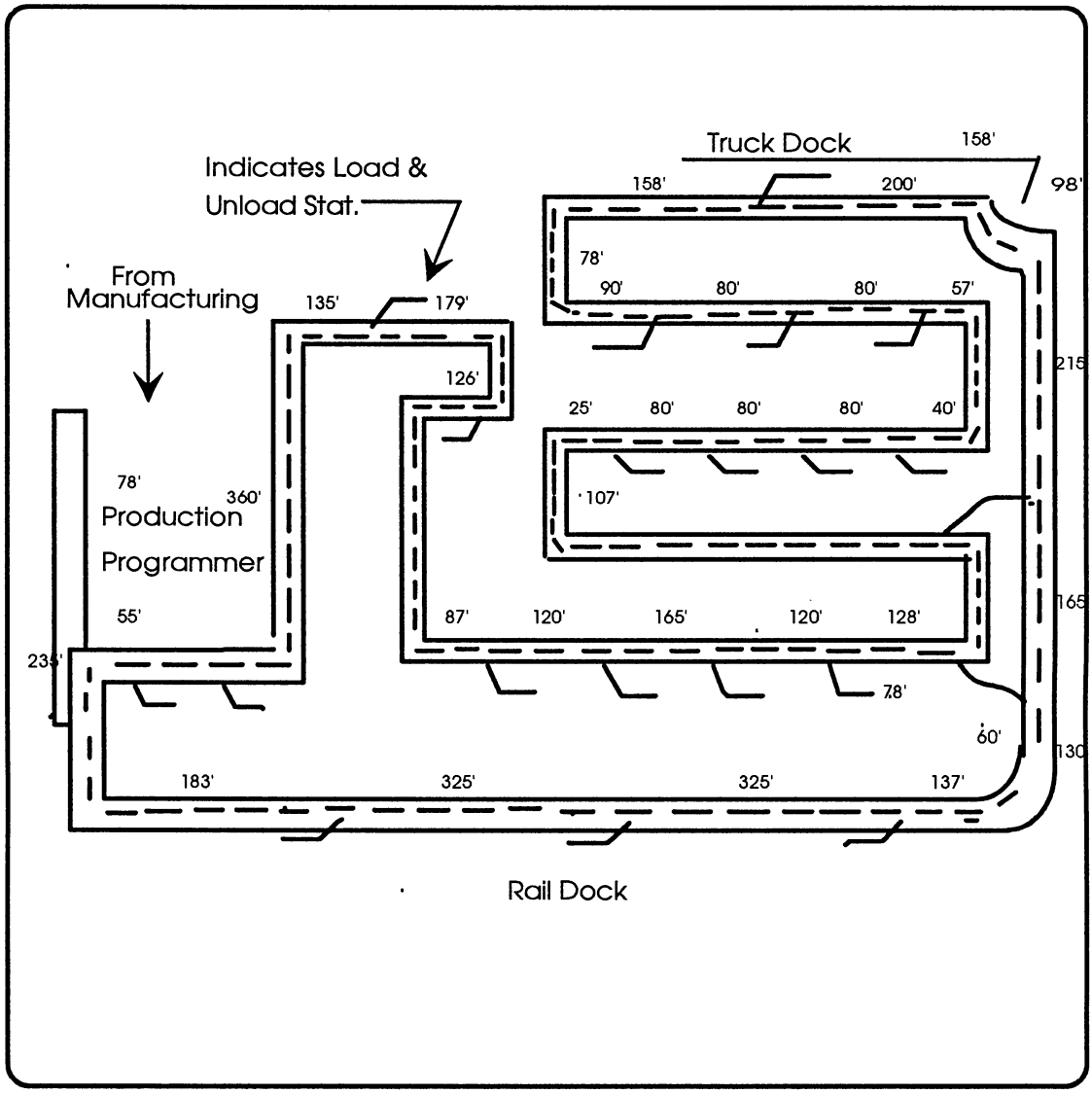

Figure 2. Conveyor System Floor Plan

load station. If this policy is adopted, an unloaded cart will be returned to the towline only when the empty cart queue is full.

All carts except those retained at unload and load stations will be moving on the towline. At any time, a cart may be loaded or unloaded. Finished goods from the manufacturing area are dispatched to a destination in the warehouse by the production programmer. Carts are loaded and unloaded by forklift trucks. After unloading a cart, the forklift operator places the empty cart on the towline and programs it to a new destination. An order for dispatching goods from a workstation other than the production programming workstation is generated according to an arrival distribution. The forklift operator executes this paper order and programs the loaded cart to the required destination.

Transfer sections with bypasses are provided to enable a cart to avoid traveling the entire distance on the main loop to arrive at a destination. There are decision points in the system where the cart has to decide between the main loop and a bypass for its subsequent movement. Bypasses also allow recirculation of empty carts in the system.

For this study, the system described above will be reduced to a single-loop conveyor system with bypasses. However, with the concepts and nature of OOP languages (reuse and extensions), the model can be easily extended to include all the features and details of the system: multi-loop and multi-floor conveyor systems (a short description of some possible extensions has been included in section VIII). The system just described is therefore composed of : (1) a towline made up of chain and hooks, which form the main loop, (2) carts in the system, (3) unload and load stations, and (4) bypasses. These system objects will be built individually, then the messages which make up the language of interactions among them will be designed to build the overall simulation program. Figure 3 shows the conveyor system that will be modeled in this study.

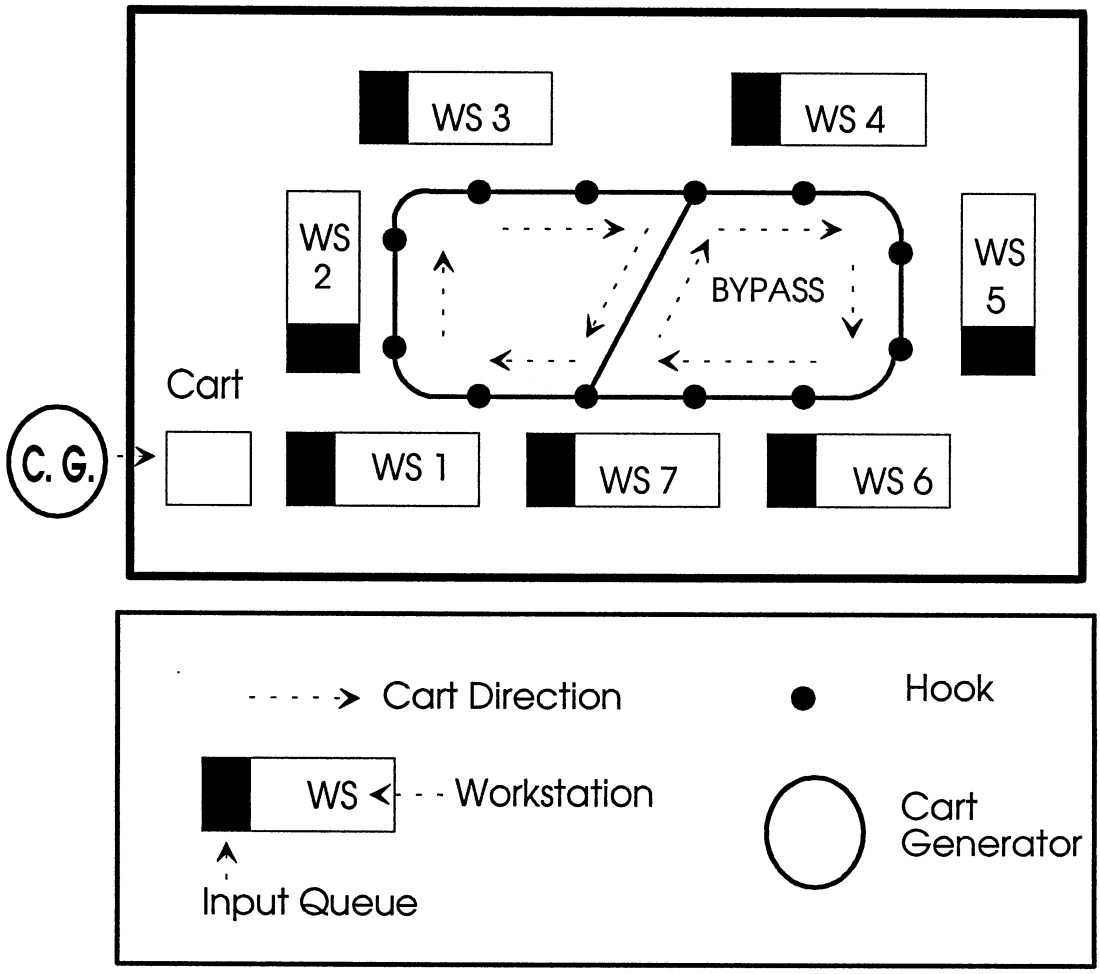

Figure 3. A Single Loop Conveyor System Selected for Experimentation

CHAPTER III

INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING AND MODELING

The research being performed in the area of simulation methodology and environments is very well detailed by Beaumariage (1990). He also presents a general discussion on the research areas which hold promise for improving simulation methodology. A review of the research being performed in the area of the OOP paradigm and a presentation of the literature relating research on the application of the object concept to simulation modeling is presented in this chapter. Also, a discussion of the applicability of Object-Oriented Programming languages and concepts to simulation modeling will be presented. The various benefits of this approach to simulation will also be illustrated.

Introduction

Given that we are in a world in which resources and time are finite quantities, a major advantage of simulation is that the development of a model and its translation into computer terms can be performed in an efficient manner. Considering this from a life cycle cost view point, we desire the ability to implement simulation models which satisfy current and future needs with a minimum cost. Therefore, the most sought objective of the new approaches to simulation is the ability of simulation models to be easily updated and highly modular (changes to a model should be localized) (Beaumariage, 1990). This

will eliminate the cost of generating a new model from the beginning once a new problem is encountered. As stated in Chapter II, OOM is a major area of this revolutionary approach to simulation modeling.

Because of this desire and because of the information on complex systems which can be gained, simulation methodology is an area experiencing continuing research activity with one of the objectives being the improvement of simulation modeling capabilities.

Object-Oriented Programming (OOP)

The principal idea associated with OOP is that all items (e.g., variables) in the system are treated as "objects". An object is either a "class" or an instance of a class. A class is a software module which provides a complete definition of the capabilities of its members. These capabilities are either provided by the procedures and data storage contained within the immediate class definition or inherited from other class definitions to which this class is related. Moreover, a class in OOP is defined by specifying its four specific elements:

1. Class variable names: these are locations which are allocated once and are associated with the class.
2. Instance variable names: these are data storage locations which are allocated uniquely for each instance of a class. Each object maintains it own internal state. That is instances of the same class will have the same instance variable allocations, but, most probably, will have different values stored in their own locations.
3. Class methods: these methods are methods available to the class itself. These methods typically manipulate class variables and provide for the creation of new instances of a class.
4. Instance methods: these methods are methods available to instances of a class. These methods will have direct access to the data associated with the

class instance receiving a message. Other instances from the same class are unaffected by variable value changes made during an instance method execution.

Smalltalk-80, the original and purest OOP language, contains four key concepts which result in making systems understandable, modifiable, and reusable (Wilson, 1987). These concepts are: encapsulation, message passing, inheritance, and dynamic binding.

Encapsulation

Encapsulation means that an object's data and procedures are enclosed within a tight boundary, one which cannot be broken by other objects. Encapsulation of properties of an object is a side-effect of implementation of an object in OOP. In the OOP paradigm objects are tightly encapsulated. The only way one can access the data is through the relevant predefined operations. Encapsulation restricts the effects of change by wrapping the data in a shield or a wall of code. All access to the data is handled by procedures that were put there to control access to the data. It also makes the objects relatively independent of their environment. The implication is that objects can be designed and tested as stand-alone units without the knowledge of any particular application. Another important side-effect of encapsulation leads to an effective enforcement effect. A person who uses encapsulation may manipulate the objects only through the operations that are defined. Direct manipulation of data in storage is not allowed.

Message Passing

Message passing is a necessary result of encapsulation. It is the only way in which objects can communicate with each other because the data stored within an object

is not shared or available to the procedures of other objects. In order for one object to affect the internal condition of another object, the first object must tell the second object to use one of its (the second object) procedures on itself. This is performed by sending a message (somewhat comparable with procedure calling).

Inheritance

Objects belong to classes. Objects that have things in common are abstracted into a "class". The subclasses inherit all the instance variables, methods of the super class. This is the concept of inheritance. The subclasses may add their own methods and variables that are appropriate to the more specialized objects. Also, one can override a general method by adding another method with the same name at the specialized level.

Dynamic Binding

Binding is more than what most programmers call linking. It is the process where operators and operands of different types, provided by suppliers, are functionally integrated by the consumers of code (Adiga, 1989). Traditional languages use early binding, in which binding is determined by the programmer and is performed when the code is written. Declaring variables to be integer, real, logical, etc., is an example of the type of early binding done in traditional programming. Dynamic binding (or late, or delayed binding) occurs generally while the program is running. This means that the decision as to which a compiled procedure will be invoked by a given procedure is not made until run-time. For example consider the line of code active Robot move-part-8772. This will execute a method (procedure) that will cause a robot to move a part. The robot is told to move a part, and it will pick an appropriate procedure to do its job.

The decision is only known at execution time. This use of late binding gives OOP a great deal of flexibility.

Benefits of Object-Oriented Programming

The domain of object-oriented programming offers many attractive features to model elements of a manufacturing system. Some of the obvious correspondences between the two domains follow.

The manufacturing system considered in this study consists of objects such as workstations (workers), carts, and hooks in a towline. The state variables of these objects change in response to events, such as the completion of the unloading of a cart; these events occur at discrete points in time. There is a natural one-to-one correspondence between physical objects in a factory and instances of software objects that represent them. The encapsulation within software objects of local data (instance variables) represent the state of the physical object and procedures (methods) for updating the state variables provides modularity of software.

The use of the object-oriented programming paradigm of "send messages to objects" in place of "procedure calls with parameters" is a convenient way to represent a real-world event. For example, when the message getProcessedAtLocation is sent to the object Cart, the Cart object will change its status from empty to loaded. The same message can be sent to WorkStation object, which will change its status from busy to idle. This emphasizes the fact that the same event is experienced by several objects of different classes, and these objects react in ways appropriate to their individual natures.

Inheritance of methods and instance variables by use of hierarchical structure permits the addition of complexity and functionality to simple objects as necessary by creating subclasses of existing classes. This feature is particularly useful in constructing special-purpose simulations for research purposes because it avoids unnecessary

complexity, permitting run-time efficiency, and also avoids confusing the researcher (Adiga, 1988).

Finally, because the objects contain their own functionality, intelligence can be built directly into this functionality using the techniques of Artificial Intelligence.

Disadvantages of OOP

OOP is no panacea! There are a few irritants that are integral to the approach. The first one is that the productivity improvements through reusability starts only after a library has been constructed. It also means that one must learn the library well before doing any serious programming. This makes the need for good documentation (something most programmers hate) very important. The cost of implementing the concept of inheritance is high in terms of space for small applications that do not take full advantage of the library.

It should also be noted that calling a procedure or subroutine is still faster than sending a message (Retting, 1987). An early study by Cox (1986) indicated that message passing is between 2 to 70 times slower than procedure calling. As mentioned earlier, another disadvantage is that languages such as Smalltalk-80 demand extensive machine resources. The run-time cost is more, hence is costly in space for small applications. But if this is a major factor in an application, extensions of conventional languages such as C offer other options for more efficient implementations (Adiga, 1986).

Application of Object-Oriented Concepts to Modeling

Many concepts of the OOP paradigm have their origins in SIMULA (Dahl and Nygaar, 1966). Although SIMULA never achieved a large popularity (especially in the

United States), many of the concepts (instance, class, etc.) introduced in SIMULA formed the foundation of OOP languages such as Smalltalk-80. So, it is not surprising that OOP languages are good platforms for discrete event simulation.

True OOM implementations having a range of features have been described in a number of articles. Knapp (1987) describes a system called SimTalk, which is a discrete event simulation environment implemented in Smalltalk-80. SimTalk adds queuing support, statistics collection, simulation graphics and interactive user interface to the features that already exist in Smalltalk-80 (multiple process support, interactive programming, graphics, etc.). A class called SimTalk provides central communication and maintains the time queue and simulated clock. Another class, SimTalkObject, is used to present the classes of objects to be simulated. There are a large number of other classes in SimTalk that include random number generators, probability distributions, statistics collectors, statistics analysis, etc. Bezivin (1987) describes another system named SimTalk which supports similar features and processes (the use of concurrent processes and semaphore synchronization operations) in distributed simulation environments by applying the TimeLock algorithm.

Researchers at Texas A&M University (Adelsberger et al., 1986) describe the features available in a simulation environment under development at their university. These features include:

. Graphical object creation along with a natural language interface aided by an intelligent assistant.

. Simulation model as well as experiments are treated as objects.

. Interactive user interface.

. Run time model modification and display, automatic experimental designs and statistical display.

. Consistency and completeness checks on model, experiments, and objects.

. Goal directed simulation.

. Selection of various abstraction levels of the simulation model and/or experiment.

Figure 4 graphically represents the proposed architecture for such a system.

Ulgen and Thomasma (1987) implemented an object-oriented simulation system in Smalltalk-80. In this system, class simulator handles the initialization of time and event scheduling. A class called Event associates a time with something to be done. Since the system is manufacturing system simulation oriented, the other classes in the system are designed to represent manufacturing entities such as work parts, work stations, storage facilities, etc. Figure 5 shows the classes developed in this system.

A SIMULA based simulation system (Nyen, 1987) was developed in the Norwegian Institute of Technology. This system defines three major object groups for the simulation of manufacturing systems: Resource Objects, Entity Objects, and Stationary Objects. In addition to the simulation kernel which actually executes the simulation, five other segments are defined that interface the user to the simulation system. The intelligent front and back end modules that carry out the actual user interface are graphical and interactive.

Among several, some other object-oriented simulation systems are: a distributed simulation system (Bezivin, 1987), a C++ based object library for parallel simulation (Abrams, 1988), an interactive simulator for VLSI design implemented in Smalltalk (Van der Meulen, 1989). Also, other simulation systems developed include: a system to provide performance models for computer systems (Pazirandeh and Becker, 1987), a computer system architecture modeling system (Ghaznavi-Collins and Thelen, 1988), a simulator for a defense related autonomous land vehicle (Glicksman, 1986), and a manufacturing OOM system (Nyen, 1987).

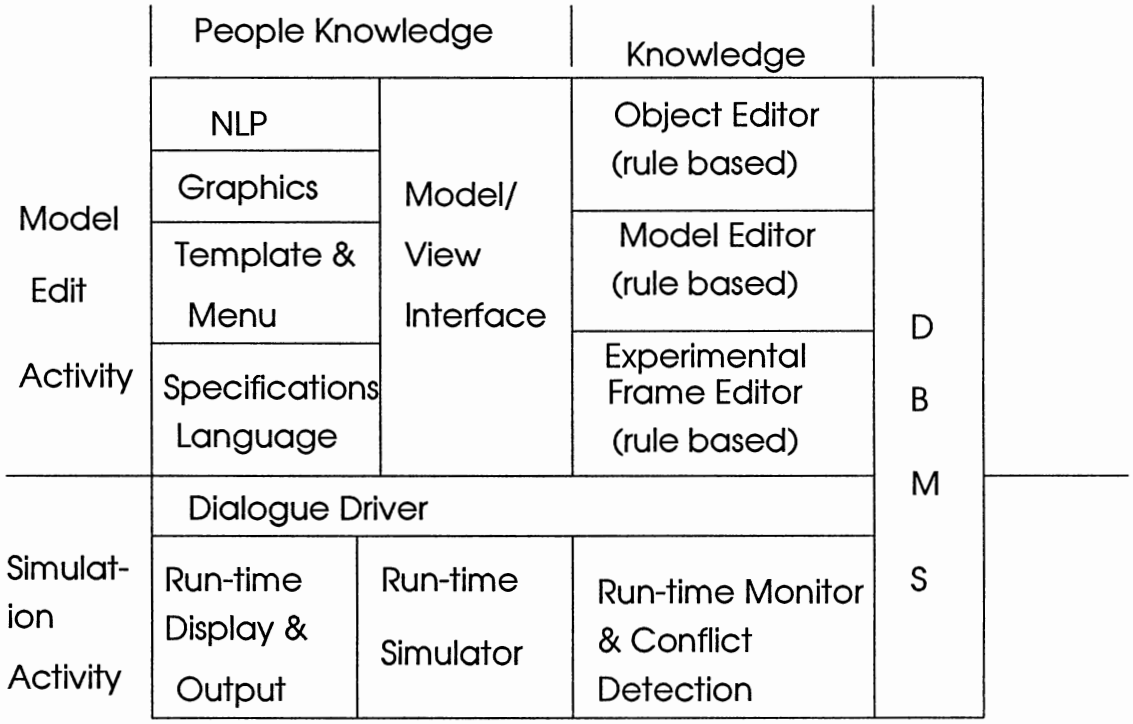

Figure 4. A Suggested Architecture for an Object-Oriented Simulation Environment (Adelsberger et al., 1986)

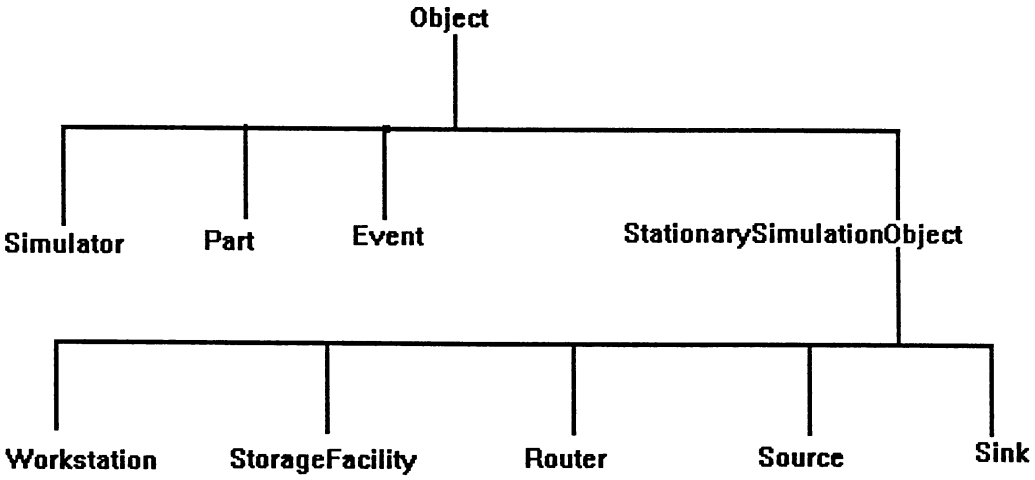

Figure 5. Simulation Classes for Ulgen and Thomasma's (1987) OOM System

CHAPTER IV

GOALS AND OBJECTIVES OF THE RESEARCH

The main objective of this research is to develop, validate, and document utility "plug-in" modular component computer simulation models which may be used to interpret and synthesize the operating characteristics of various types of complex recirculating conveyor systems, using a revolutionary approach to simulation such as Object-Oriented Programming. The author should be able to illustrate the benefits and disbenefits of this approach relative to the currently available simulation methodology (in this case SLAM). This effort is to be completed by accomplishing the following series of sub-objectives:

1. Develop a library of reusable software: A hierarchical organization of classes necessary for the conveyor system at hand will be developed. This is accomplished by identifying the objects appearing in the problem at hand. Once the necessary objects are created, they should be classified according to their similarities and differences. Once the functions have been defined, the class hierarchy can be planned and implemented to take advantage of inheritance.
2. Evaluate the value and quality of the developed prototype: This will require the definition of tangible and intangible benefits of this new modeling methodology over traditional approaches. This objective will be achieved through the application of the Analytic Hierarchy Process (AHP) to evaluate various aspects, benefits, and disbenefits of the developed OO simulation system.

3. Explore ways to expand the model developed to accommodate all the complexities that a conveyor system can incorporate: For this study, only the most important features of the system will be modeled. However, ways to expand the functionality and application of the classes developed by introducing the level of details desired will be explored. Bordiga et al. (1985) stated that the proper way in dealing with complex systems with high need for details such as manufacturing systems is to abstract out the most important ones and introduce the others in successive phases of a refinement process. This concept could be used in an Object-Oriented Paradigm which supports an incremental style of development.

CHAPTER V

RESEARCH PLAN AND PROCEDURES

To achieve the goals and objectives outlined in chapter IV above, the research will be performed through several chronologically ordered phases as presented below.

Phase I:

Examining and analyzing the functional components that are common to representative existing recirculating conveyor systems: The particular conveyor system selected to initiate the design and development of this simulation program was explained in detail in chapter II.

Phase II:

Specifying the types of interfacing that can occur between the functional components of a recirculating conveyor system: Determination of the object linking and model building procedures based upon the functional specifications from Phase I.

Phase III:

Developing and encoding within the general software environment (Smalltalk-80) a series of modular elements to represent the functioning of the objects and message passing among them previously described.

Phase IV:

Incorporating the software simulation objects developed into flexible utility simulation systems that can be utilized to "build" complex conveyor system simulations.

Phase V:

Use of the AHP model to compare the new environment and another commonly used environment (SLAM II): Conclusions drawn from this comparison should allow the researcher to determine the benefits and disbenefits of an object-oriented programming environment.

Phase VI:

Documenting the software objects, including instructions for their use in developing modular utility simulation models of larger scale complex conveyor systems: Ways to expand the model developed to accommodate all the complexities that the conveyor system described above can incorporate will be explored.

Phase VII:

Summarize results and prepare final format: At the end of phase VII the research results will be summarized and documented. This phase represents the summary of the research activities and the presentation of results in final form.

CHAPTER VI

DEVELOPMENT OF A PROTOTYPE OBJECT-ORIENTED MODELING (OOM) ENVIRONMENT FOR A CONVEYOR SYSTEM

This chapter presents the steps taken in the design and implementation of an OOM environment developed for the system at hand. Illustration of the features and capabilities of the resulting implementation will also be presented.

Conceptual Design and Implementation

This section describes the approach to the design of the software library, conveyor system, its structure, and some of the techniques used to enforce the design guidelines in implementing the objects.

Relevance of Object-Oriented (OO) Paradigm to the Model at Hand

The OO paradigm has been exploited for modeling manufacturing systems by several researchers (Adiga and Gadre, 1990; King and Fisher, 1986; Sanderson et al., 1991). Adiga and Gadre (1990) describe modeling of a flexible manufacturing system. Their emphasis is on the modeling methodology and its translation into software using OOP. Adiga and Glassey (1986) present a conceptual design of a software library for simulation of semiconductor manufacturing systems. They have identified three goals in their research as (1) ease of assembling special purpose simulation models, (2) ease of

modification of object library and (3) the run time efficiency of the model assembled from the library of objects. The first two goals directly lead to reusability. Sanderson et al. (1991) describe design and implementation of a Hierarchical Simulation Language, which is interpreter based and hence offers certain advantages and disadvantages of portability and modifiability (during program execution) (Bhuskute et al., 1992).

Reuse, extension, and maintenance of software objects are the main productivity benefits sought from our adoption of object-oriented programming technology. Software reusability is a goal of great promise. Designing for reusability includes identification of object behaviors that are reusable in more than one context. Also, class definitions must be written in such a way that the system object interconnection information can be supplied as parameters, routings, or values of instance variables to newly created instances of previously defined classes. The methods which are defined for the classes are written in such a manner that this generally specified linking information is accessed through instance variable locations or through responses to message requests. Designing for maintenance involves designing objects to be independent of others. A set of goals were formulated for the library of software objects needed to be built:

1. The first goal is to make it easy to assemble special purpose simulation models, customized for individual research questions. This can be accomplished by designing a library of reusable software objects. With the right library of software objects, we expect that the work of designing simulations would be one of choosing and interconnecting objects of interest and linking those objects with code of one's own research strategy related to the problem at hand.
2. The second goal was that the library should be easily modified and extended and that parts of it could be reused in other contexts. Achievement of this objective depends to a large extent on identifying the proper conceptual framework for the library of objects and on the use of design principles that capitalized on the strengths of the object-oriented programming paradigm.

3. The third, and last important goal is that the library must be easy to understand, both the individual objects and the way they work together.

Smalltalk-80 was chosen to be the language of implementation for the simulation model. This choice is due to the fact that Smalltalk-80 is one of the purest OOP languages. Many of the object-oriented characteristics can be traced to SIMULA 1 language (Meyer, 1988). Simula though popular among the academia in Europe and throughout the world, has never gained widespread use in the commercial environment (Kreutzer, 1986). Smalltalk-80 added the message passing paradigm creating a programming style well known as OOP (Kreutzer 1986, page 194; Meyer 1988, page 437). The concepts underlying OOP can be easily extended to simulation modeling (King and Fisher 1986; Mize et al. 1989; Thomasama and Ulgen 1988; Ulgen et al. 1989). For details on the language Smalltalk-80 and OOP, readers may refer to Goldberg and Robson (1989) or Cox (1987). The concepts object, class, message, and method form the basis of programming in Smalltalk-80. The methodology for using Smalltalk-80 consists of:

1. Identifying the objects appearing in the problem.
2. Classifying the objects according to their similarities and differences.
3. Designing messages which make up the language of interactions among objects.
4. Implementing methods which are the algorithms that carry out the interaction among the objects.

Design of Approach: A Conceptual Framework

This section describes briefly the conceptual framework used for the discrete event simulation of the conveyor system. In discrete event simulation, the time sequence of real-world events is reproduced by the model; the state of the simulated system

changes only at the discrete times when events occur. After the state update has been computed, the simulation clock is advanced to the time of the next event.

In Figure 6, the author illustrates an object diagram representing the topmost structure of the system. Here the author asserts the existence of the objects named in the earlier discussion of the system' s description: (1) a towline (conveyor loop) made up of chain and hooks, which form the main loop, (2) the carts in the system, (3) the unload and load stations, and (4) the bypasses. The author has explicitly chosen not to show any other relationship among the objects at this level. To do so during this phase of analysis would be premature, since assertions of internal relationships generally denote design decisions. These system objects will be built individually, and then the messages which make up the language of interactions among them will be designed.

Design of the Object Hierarchy

It is important at this stage in the software development life cycle to identify reusable software components that can be used to build the system at hand, so that one can build as much of the conveyor system as practical from existing components rather than creating entirely new ones.

Actively looking for reusable software components that are relevant to a new system is a very important activity in any development. This process is facilitated by rich class libraries that are typically available for object-based and object-oriented programming languages. However, these classes cannot be applied directly, because they are domain dependent. Instead they must be tailored so that they express the vocabulary of the conveyor system. After, studying the object library made available by ParcPlace and OSU' s CIM Center research team, the author developed a set of classes that proved relevant to the target system. The classes were developed by combining some classes

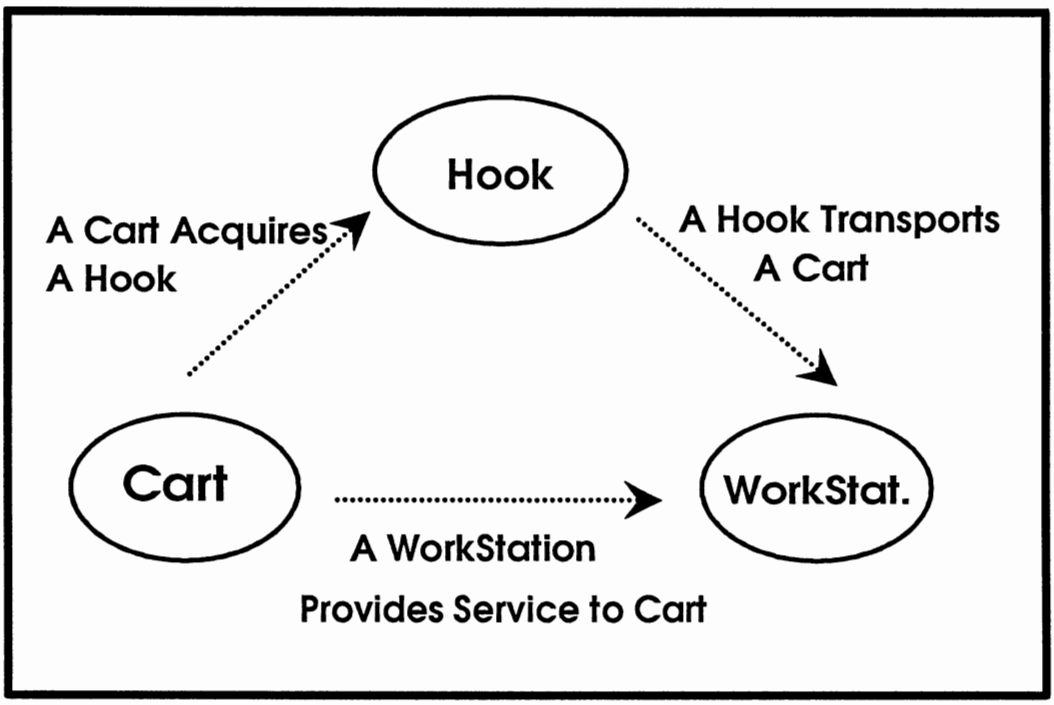

Figure 6. Conveyor System Object Diagram

found within the software environment through the development of the appropriate procedures. The Simulation Class library developed for the model at hand consists of four categories:

1. Simulation Classes
2. Target System Simulation Elements
3. Class ConvModel
4. User Interface Classes

Simulation Classes. These classes collectively provide a basic framework for discrete system simulation and statistics collection. The OOM classes discussed in this section are of a highly abstract nature and represent the objects or concepts which must be explicitly accomplished to make simulation work. Some of these classes, such as Simulation, DelayedEvent, TrackedNumber, ObsTrackedNumber, Probability Distribution, and RandomNumberGenerator, are as described in Goldberg and Robson (1989). Figure 7 depicts the class hierarchy of these objects.

Simulation: The purpose of class Simulation is to manage the topology of simulation objects and to schedule actions to occur according to simulated time. The event_Queue, instance of class Simulation maintains a reference to a collection of SimObjects, to the current simulated time and to a queue of events waiting to be invoked. The unit of time appropriate to the simulation is saved in an instance variable called the Sim_Clock and represented as a floating-point number. The unit might be in milliseconds, minutes, days, etc. A simulation advances time by checking the event_Queue to determine when the next event is scheduled to take place. If the event_Queue is empty, then the simulation terminates.

Smalltalk-80 message protocol provides a mechanism for defining arrival schedules of simulation objects, managing scheduling of processes, and controlling the execution of simulation. This subclass is further refined to define subclass ConveyorSimulation.

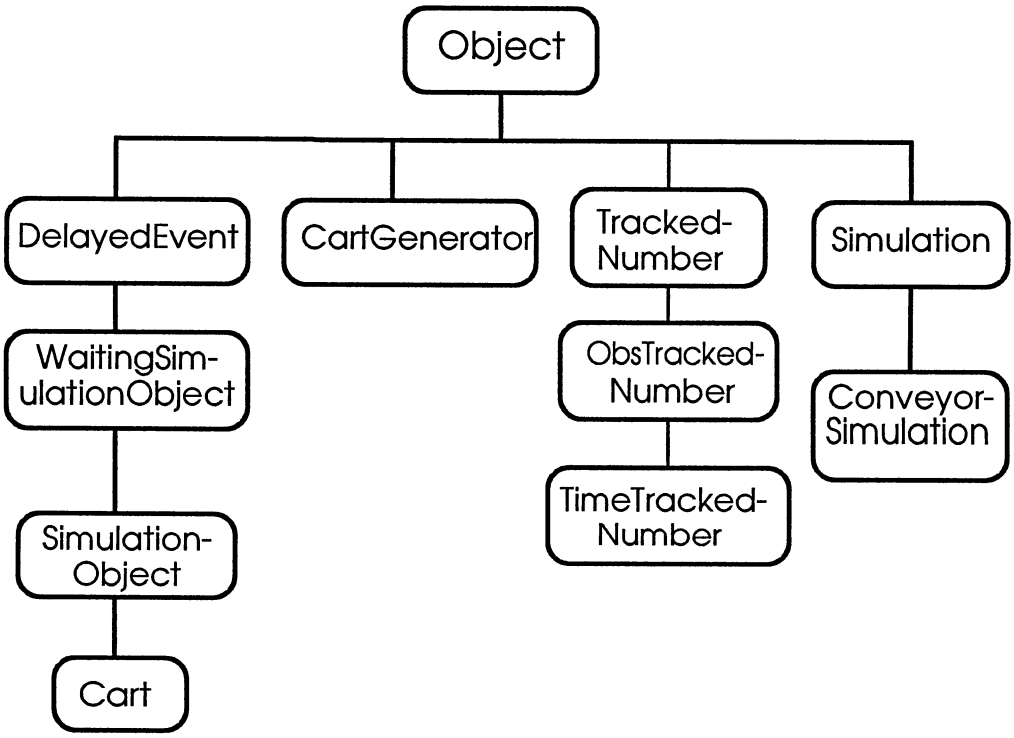

Figure 7. A Diagram of the Structure of the Simulation Classes

With its level of abstraction the Simulation object provides the structure through which all communication between other system level objects within the simulated system occurs. All system objects within the OOM at different level of abstraction communicate indirectly with each other and directly with the Simulation Object. Therefore, the Simulation object acts as the controller of system level of interaction for the entire model run.

DelayedEvent: This is an active process, which when delayed for a specific amount of time, is placed on the queue sorted with respect to the resumption time. This class models the simulation entity and is an abstract class. It has to be further refined to faithfully represent the system being modeled. The message protocol provides mechanisms for entering the entities in the simulation, for executing the tasks corresponding to the life cycle of the entity, and for terminating these entities.

Tracked Number: This class is defined as an abstract class. Its instance cannot be used directly, but subclasses are further refined as needed. The tracked number is a repository for statistics collected on a particular variable.

ObsTrackedNumber: This class collects statistics on observations. The message protocol consists of methods for clearing statistics, collecting observations, calculation of resulting statistics, and for printing the results.

Probability Distribution Classes and RandomNumberGenerator: Class RandomNumberGenerator provides a stream of random numbers required for simulation, whereas the probability distribution classes implement a variety of random variate generators. For brevity reasons these classes were not included in figure 7.

CartGenerator: Another class definition needed is the CartGenerator class. This class provides a behavior similar to a 'Create' node in SLAM II (Pritsker, 1986). It creates SimulationObject instances and enters them into the simulation based on their arrival distributions. The creation activity of theses instances is performed when the event is initiated. The SimulationObject instances are then passed on to the next object. This

class is the same as the WorkFlowGenerator Class developed by the Computer Integrated Manufacturing Center at Oklahoma State University.

Target System Simulation Element Objects. The simulation element objects, covered in this section, round out the capabilities of the OOM environment by representing the concrete elements present in the system of interest. Figure 8 depicts the hierarchy of these classes. Instances of the simulation element classes are used as building blocks in the construction of the simulation model. The simulation element classes which have been developed are found necessary to implement a model of the chosen target system. This target system is a Constant Speed, Discretely Spaced, Recirculating Conveyor System. Figure 9 represents a rough sketch of the Conveyor System.

The simulation model representation of this system requires several different types of objects including the following:

Workstation: A loaded cart joins the unload station queue and waits on the unloading facility. At the workstation, carts are unloaded on a first come - first served basis. On completion of unloading, the unloaded cart may be placed on the towline and assigned a new destination supplied by the workstation. Sometimes, it may be desirable to retain a certain maximum number of empty carts in an unload and load station, so that if an order to load goods arrives at the station it can be executed immediately using an empty cart available at the station. If this policy is adopted, an unloaded cart will be returned to the towline only when the empty cart queue is full. Therefore, Workstation class is set up to process work flow items from a single queue of waiting items. The Workstation must be able to accept the arrival of work flow items, determine if it can provide service if it is idle, schedule the service operation, and transfer the item to the next processing workstation. In addition, the Workstation class must keep statistics on its service and provide for their output as requested.

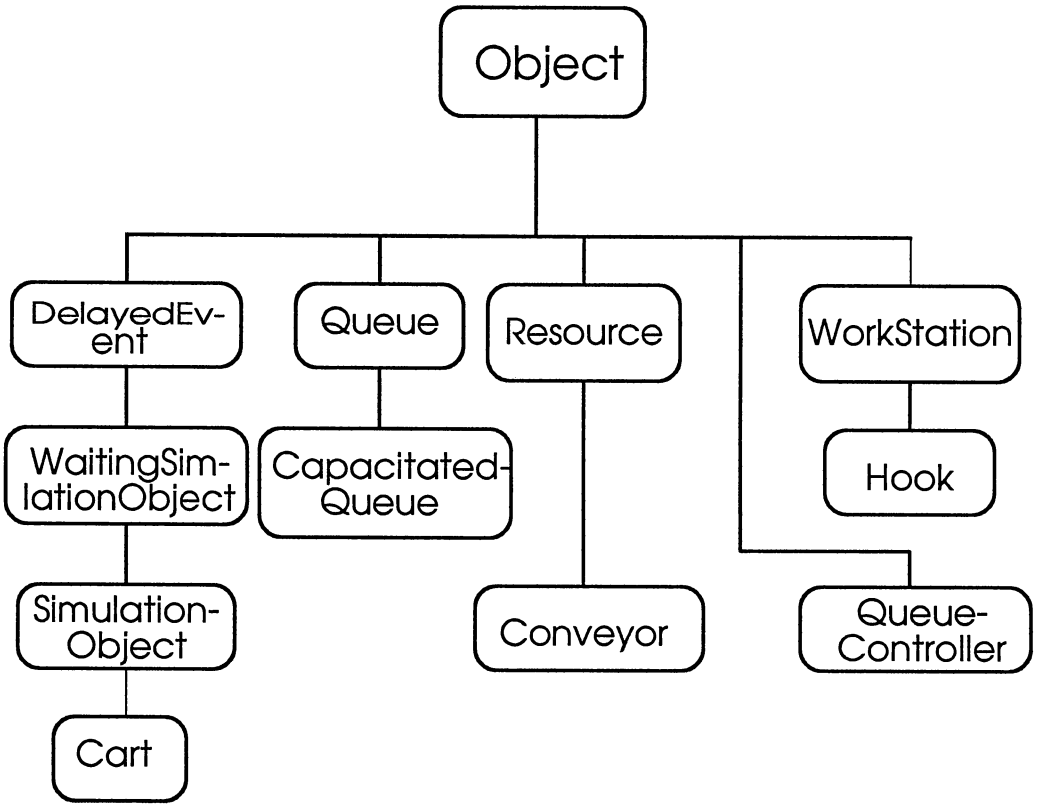

Figure 8. A Diagram of the Structure of the Simulation Target System Objects

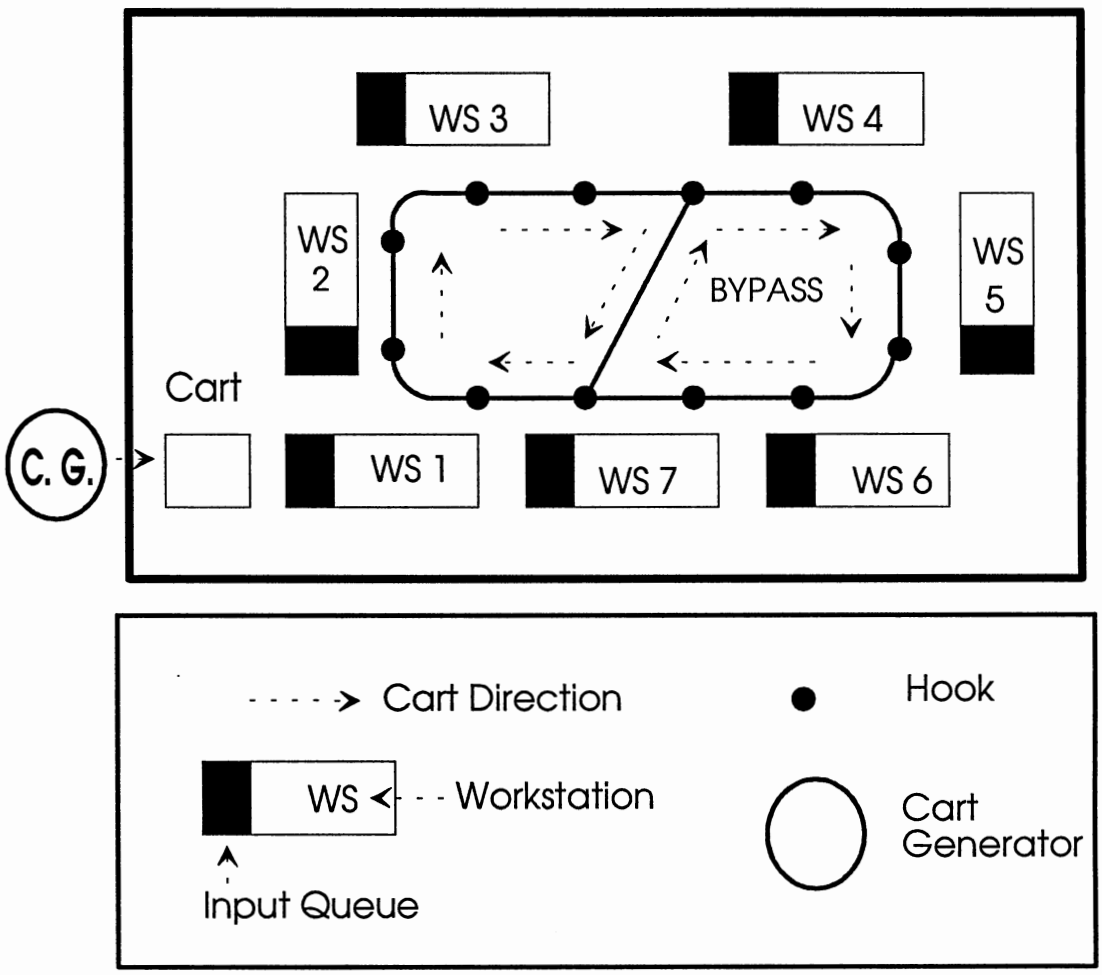

Figure 9. A Diagram of the Target Conveyor System

It is important at this point to define the different types of WorkStation instances incorporated in the system and the type of service they provide. The system incorporates five types of workstations based on the operation performed on an arriving cart (loading or unloading). The types of unload and load stations that can be handled by the program are as follows:

. Unload and load station with common service facility.
. Load station with assigned load time.
. Load station without assigned load time.
. Unload station.
. On-line unload station.

A special type of load station is the production programming and load station and the object component bypass. Each type workstation is described below in more details.
- Production Programming and Load Station (PPLS): In a production warehouse system, goods from the manufacturing plant are loaded at PPLS to be transported to various locations in the warehouse. In general, PPLS is a central location from which goods are transported to various unload stations in the system. The load and unload activities at other stations may be considered as being initiated from PPLS. A PPLS is generally served by an auxiliary power line which brings the empty carts from the main conveyor to the loading area and delivers the loaded carts from the loading area to the main conveyor line.
- Unload and Load Station With Common Service Facility (UALSC): Both unload and load activities occur at UALSC. A loaded cart arriving at UALSC is unloaded and assigned a new destination. An unloaded cart will be retained at UALSC if the empty cart queue at the load station is not full. An empty cart arriving at UALSC is transferred to the empty cart queue for being loaded.

All loaded carts arriving at UALSC join the station queue at first. If there is no other cart that is being unloaded or loaded, unloading on the newly arrived cart begins

immediately. Otherwise, the newly arrived cart waits in the queue until all the preceding carts are unloaded or loaded.

A certain maximum number of empty carts may be retained at the load station to facilitate the availability of an empty cart for loading when a load order arrives. This desired maximum number specifies the capacity of the empty cart queue. Load action is initiated when a load order exists and an empty cart is available, provided the service facility is free. A loaded cart is assigned a new destination and transferred to the conveyor line. In UALSC, only one service facility is available for both unloading and loading. Carts are unloaded and loaded following a first-come first-served (fcfs) policy.

- Load Station With Assigned Load Time (LSWAT): An empty cart arriving at LSWAT joins the station queue first. Loading on the newly arrived car begins immediately if there in no preceding cart at the station, provided a load order is available. When there are empty carts in the station already, loading on the newly arrived cart is scheduled when loading is completed on all the preceding carts. Load action will be initiated only when a load order exists. The load orders are generated by the production programmer or some other source in the real system, and are initiated in the simulation program by a user defined distribution function.

- Load Station Without Assigned Load Time (LODST): The difference between LSWAT arises in the mode of operation. LSWAT performs loading only when a load order exists whereas LODST performs loading immediately after an empty cart arrives provided the loading facility is free. A load order is not explicitly required in the case of load station without assigned load time. It is implied that an empty cart arrival at LODST brings a load order with it. In both LSWAT and LODST, carts are loaded on a first come first served basis. A loaded cart is assigned a new destination and placed on the conveyor chain.

- Unload Station (ULST): A loaded cart arriving at an unload station joins the station queue first. Unloading begins immediately, if there is no other cart in the station queue. If there is one or more carts in the station queue, unloading is scheduled on the new arrival after all the preceding carts are unloaded. The unloaded cart is assigned a new destination and placed on the conveyor chain.

- OnLine Unload Station (OLUNLOD): This is a special type of unload station where goods are unloaded as the carts move along the conveyor chain. A special type of unloading facility enables this operation. In certain warehouse applications, the unloading device used for this type of operation employs a lever mechanism by which a moveable unloading arm sweeps across the load and pushes it off the cart. It is also possible to accomplish this type of unload using an overhead crane facility that can move up and down and in the horizontal direction (Terrell, 1977). Several other unload facilities of this type can be conceived. A few types are already in use and some may be developed in the future. Regardless of the type of equipment used, a station that performs unloading while the cart is moving on the conveyor is classified as on-line unload station.

- Bypass (BYPASS): Bypass sections are provided for recirculating carts in the system along the shortest route. If the station queue of a load or unload station is full, a cart arriving at this facility recirculates and returns to the station after recirculating once via a shorter route. If there is no bypass in the system, a cart has to travel the entire loop before it arrives at the facility again. Bypass sections accommodate one cart at a time.

Hook/Conveyor: The Hook object is a subclass of the Workstation class. However, this object will be used as the material handler to allow the movement of the object Cart (to be defined below) between the different workstations. All the hooks in the system will make up the Conveyor object. Hooks in the Conveyor object will be discretely and uniformly spaced. The Conveyor object will allow the movement of the hooks at a constant speed specified by the user. The conveyor instance will also schedule the arrival

of hooks to the workstations. Hooks in front of the workstation' s input queues will check if they can "deliver carts" for service. At the same time hooks in front of the workstation' s output queues will check if they can "accept carts" to transport them to their desired destination.

Cart: This object is actually an instance of SimulationObject with specific instance variable values. All carts except those retained at unload and load stations will be moving on the towline. At any time, a cart may be loaded or unloaded. After, a cart is unloaded, it is placed on the towline and sent to its new destination. The destination information is contained within instance variables carried by the cart. The cart's next destination is acquired from the workstation at which it is currently being processed. An arrival event to the next simulation model element specified by the destination attached to the Cart instance is sent to the simulation object for scheduling on the event list.

Queue: This class provides one building block which may be used to construct specific simulation element classes. The class is defined with the procedures to store other objects within an ordered linked list, to remove objects from the front of the queue, to search the queue for specific objects. When a new simulation element class needing queuing features is to be defined, the class developer simply uses an instance of the queue class as a component of the new simulation element and programs the correct internal interaction mechanism.

QueueController: This is a subclass of object. Class QueueController creates a controller through which a workstation communicates with its input and output queues. The QueueController is created automatically when the WorkStation object is created.

Class ConvModel. This class (ConvModel) is responsible for providing a template that holds information about the entire simulation model. ConvModel allows the user to create, modify, add or delete portions of the model. Before running an actual simulation, ConvModel undergoes a consistency check for the newly created model and then based on the user information it instantiates all the objects required for the execution

of the simulation. This arrangement separates model definition from the simulation entities and enables the user to initiate a new simulation run by using the information available in the templates provided by the ConvModel.

User Interface Classes. These classes collectively provide a variety of user interface items for system model definition, modification, and simulation experimentation with the conveyor system.

ConvView: This class offers the main menu for the modeling and simulation environment. As depicted in Figure 10, the Conveyor Simulation Launcher view provides a user with a list of options such as Workstation Definition.

WorkStationDefintionView: This class offers a user interface for workstation definition. As depicted in Figure 11, the workstation definition view provides the user the workstation database for quick workstation definition or reconfiguration. The pop-up menus available in the workstation definition view provide the user a mechanism to enter the required input parameters for the workstation.

ProbabilityDefinitionView: This class offers the user interface for the workstation' s probability definition. As depicted in Figure 12, the probability definition view provides the user with database to specify the probabilities that a certain workstation uses when specifying the next destination for the Cart instance.

ConveyorDefinitionView: This class offers the user interface for conveyor definition. As depicted in Figure 13, the ConveyorDefinitionView provides the user with a database to specify the required input parameters to initialize the Conveyor ConveyorDefinitionView.

ConvExperimentView: This class offers a user interface for defining the experimental simulation parameters such as, simulation termination time, initial seed for the random number stream, "Trace on" time, "Statistics clear" time, and histograms.

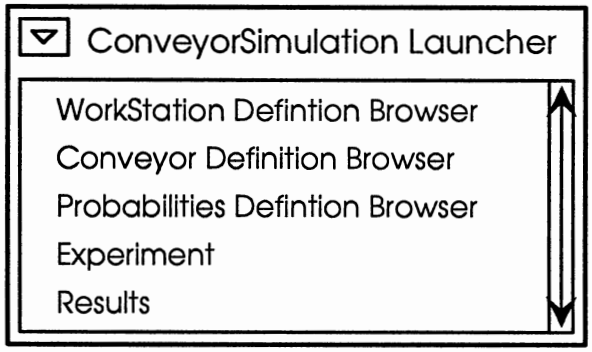

Figure 10. Conveyor Simulation Launcher View

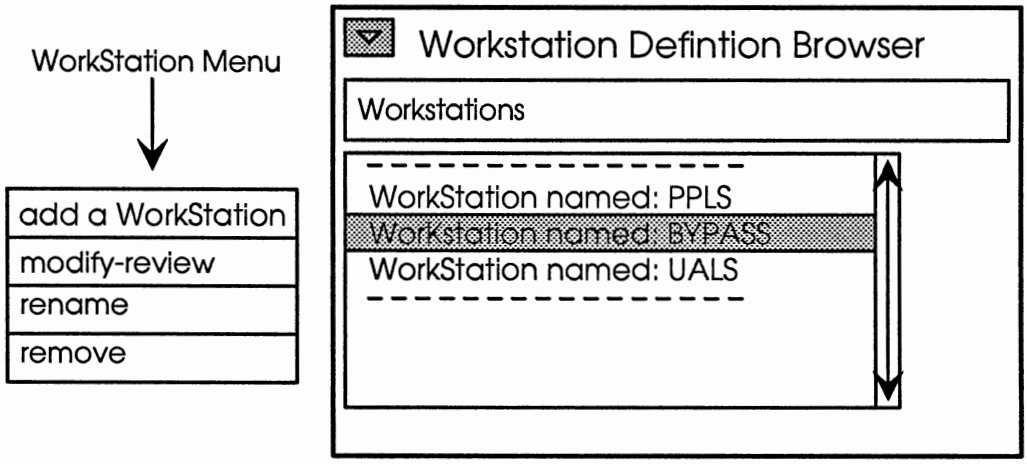

Figure 11. WorkStation Definition View

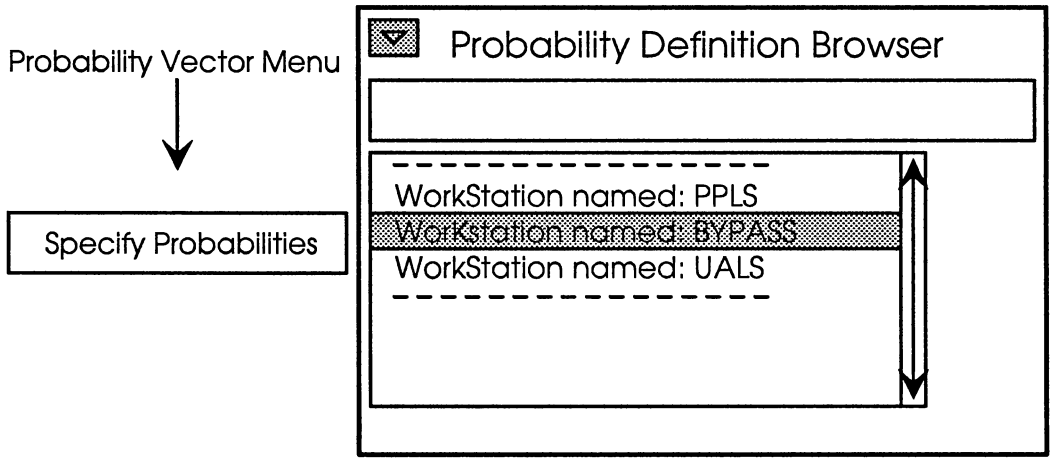

Figure 12. Probability Definition View

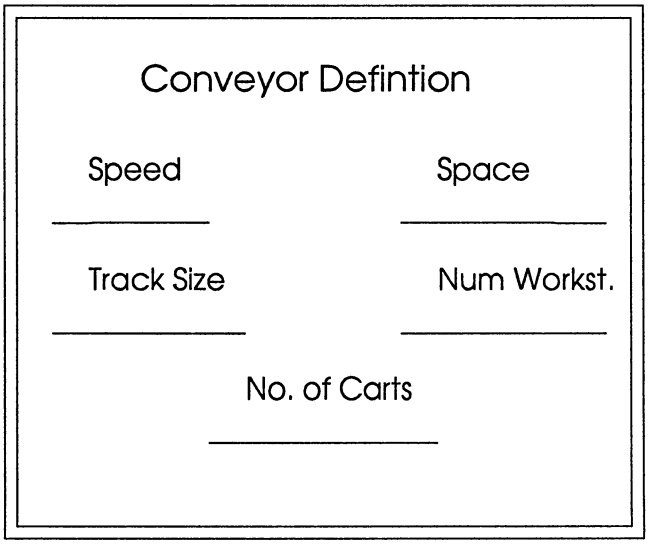

Figure 13. Conveyor Definition View

ConvResultView: This class offers user interface for depicting the simulation results. The result view provides a summary report of all the statistics collected and trace output for a simulation run.

Simulation Model Operation

At this point the major classes needed to develop a working simulation for the target system have been described. The following section describes the manner in which the objects will cooperate with one another during the simulation activity.

Time Advancement

The Simulation object in an OOP simulation system handles time advancement. Time advance occurs by having the Simulation object loop through a portion of a method to find the next event on the eventQueue. This event initialization method then sets the new value of the current time instance variable and triggers the next event to occur by executing the event initialization code retrieved from the event list. This sequence of activities is performed repeatedly until no further events are scheduled or the specified simulation run length has been achieved (designated by the end of execution event).

Entity Creation and Flow

The CartGenerator object, as mentioned earlier, is used to implement new instances of entities and trigger their arrival to the simulation element instances which are part of the model. The creation will be initiated when the message startUp invokes the message defineArrivalSchedule which schedules the arrival of instances of Cart. Each time the message proceed is sent to simulation, a Cart instance enters or exists.

The travel of Cart instances through the model is completely controlled by the simulation control framework, sometimes referred to as the "life cycle" of the entity. This framework involves the sequence of messages startUp, tasks, and finishUp. When the Cart instance first arrives at the simulation, it is sent the message startUp. The category task consists of messages the modeler can use in specifying the entity's sequence of actions as it flows through the system. The Cart instance might be held for an increment of time (holdFor:). It will also acquire access to another simulation object that is playing the role of a resource (acquire: ofResource:) such as the element object Workstation. Finally when the Cart instance carries out all its required tasks, it is sent the message finishUp. This message will signal to the simulation class that the receiver is done with its tasks and ready to exit the simulation.

Two separate processes are created in the model that will repeatedly schedule specific sequences of actions according to specific arrival distributions.

The first process is the scheduling of the arrival of hooks to the input and output queues of each workstation where each hook checks if it can deliver or pick up a cart.

The second process is the scheduling of the arrival of work orders to the loading workstations.

Event Initiation and Scheduling

The simulation approach adopted in this study is the process-oriented approach. A process is defined as a sequence of events, together with a set of actions accompanying each event. Thus the (potentially infinite) sequence of arrival events, together with the associated creation of new entities, can be considered as a process. The behavior of a system can be represented by a set of processes whose event sequences, when merged, contain all events that occur in the system. Somewhere behind the scenes there is the clock which is advanced from event to event, and the equivalent of an event list showing

what is scheduled to happen and when. However, the entries on that list are now processes, ordered according to the time of the next events in their respective sequences (Mitrani, 1982). In the process approach, the Simulation object itself has a control framework similar to that described for the SimulationObject (Cart). The response to startUp is to make the simulation the currently active one and then define the simulation objects and arrival schedule. The inner loop of scheduled activity is given as the response to the message proceed. Whenever the simulation receives the message proceed, it checks the reference count of processes by sending the message readyToContinue. If the reference count is not zero, then there are still processes active for the current simulated time. Then the system-wide processor, Processor, is asked to yield control and let these processes proceed. If the reference count is zero, then the event queue is checked. If it is not empty, the next event is removed from the event queue, time is changed, and the delayed process is resumed. When the simulation method has completed its event list addition, control returns to the simulation element instance methods, from which control will return to the simulation event initiation method. Basically, what happens is that a hierarchy of messages to different methods is established. Execution is returned to methods in reverse order when a method which makes no call to another method is encountered. Each event is initiated by the simulation object, processed through all needed methods, and finally execution control is returned to the simulation object which then retrieves the next event on the eventQueue.

Summary

This discussion may lead one to the conclusion that an OO simulation system will be a very complex package. This perception is not really correct. Actually, the interaction, which will be handled by the OOP environment, is the complex part. By using the inheritance and encapsulation features in the OOP environment, the

development of the software needed is much easier than would typically be the case in a traditional computer language. Once the basic units are developed (a library of simulation element objects and the set of simulation process elements) and standard procedures for element interactions are determined, the design and use of simulation models within the OOM environment should be relatively straightforward and efficient.

OO Simulation Object Linking

Introduction

In the previous section the description of the classes was made from a perspective internal to the classes without consideration of the way these classes communicate with one another. The building of simulation models is essentially developing the communication between these objects to fit one's purpose. The design of the interaction between these objects should support generalized linking and the techniques used to provide this linkage must be understood.

The Structure of Object-Oriented Models

The first step of object-oriented design which relies on the identification of the topmost classes and objects is completed. Next, design decisions regarding the semantics of each of these abstractions, as well as their relationships must be accomplished. Also some mechanisms that exploit the commonalty among these objects to simplify our overall design must be developed.

Since, the first goal of building the library of software objects is to make it easy to assemble special purpose simulation models, customized for individual research questions, designing the OOP environment should have as a main goal the design for

reusability of simulation elements. This could be accomplished by designing the model structure to allow the separately developed simulation objects to exist and function correctly together in any simulation model. A hierarchical organization (for the communication links between objects in a model) of simulation model objects is proposed based on the following characteristics: 1) the "stand alone" nature of objects allows an object to be linked to a set of necessary (for correct functioning) objects and to be unaffected by the presence or absence of other objects in the system and 2) a hierarchical organization assumes that linkage (i.e., methods and variables inherited) between system components are vertical (there are no horizontal links between subtrees in hierarchical system) (Beaumariage, 1990). The first feature allows a hierarchical structure to be used, and the second feature supports reusability of simulation objects. Also, direct communication from a particular object is limited to other objects that exist either one level higher or one level below in the same subtree structure. Interaction between objects separated by more than one hierarchical level or on the same level of the hierarchy occurs indirectly through an intermediate object or controller. The only relaxation of these restrictions is that the Simulation class acts as the controller for communication between different system levels.

Smalltalk-80 Class Implementation

This section discusses in some detail the implementation of several representative simulation objects developed for the system at hand. This section however, only provides the reader with a basic understanding for the structure of the simulation software. The Smalltalk-80 implementations of each of these classes are available for detailed examination in Appendix A.

As mentioned in detail in chapter III, a class in OOP is defined by specifying its four specific elements: 1) class variable names, 2) instance variable names, 3) class

methods, and 4) instance methods. The author will mention all the instance and class methods but will only explain in detail the important ones.

Simulation Element Objects

For the reason of brevity , the simulation classes will not be explained here. The description of theses classes is the same as the one included in Goldberg and Robson (1989). The section below will discuss in detail each of the target system simulation element objects:

ConveyorSimulation. This class is a subclass of the class Simulation. In addition to the capabilities inherited from its superclass, ConveyorSimulation class will have procedures designed specially for the system at hand.

Data storage

Class variable names:

Instance variable names:

cartGenerator
outputStream
conveyor

The instance variable cartGenerator stores references to different cartGenerator elements in the simulation model in an OrderedCollection instance (a class definition already available in Smalltalk-80). The instance outputStream will be used to designate the outputStream of the simulation. Finally, the instance variable conveyor will be used to store the object Conveyor.

Software methods

Class methods:

1. examplecg

This method is used to develop an example for an experiment run. The model developer will not have to type in all the input parameters through the user interface every time he/she desires to run an experiment.

Instance methods:

1. addCartGenearator
2. cartGenerator add: aGenerator.
3. Cart initialize
4. finishUp
5. clearStatisticsAt: aTime
6. printResultsOn: aStream

The CartGenerator object is added to the simulation by using the method addCartGenerator. Different types of cartGenerator(s) will be created and then stored in the cartGenerator storage location by sending the message cartGenerator add: aGenerator. The method initialize will initialize the Cart object so that "time in" statistics could be collected for individual job types. This method will also allow the specification of the outputStream where the results will be written. In addition to emptying the event queue, the finishUp message will allow the printing of all the statistics collected during the simulation run execution (these statistics are the ones specified by the user) by invoking the message printResultsOn: aStream. The method clearStatisticsAt: aTime, is a method which is typically scheduled to execute at some specified time (to remove the effects of a simulation warm up period) by the model developer.

Cart. For the system at hand, the author created a subclass of SimulationObject. This class will inherit all the capabilities of its superclass in addition to the features needed for this study.

Data Storage

Class variable names:

1. CartUtilization
2. Count

The class variable CartUtilization is used to store the time persistent statistics such as the utilization of the cart in a TimeTrackedNumber instance (a class definition already available in Smalltalk-80). Count is a location to store the serial number of the Cart instance created in a Dictionary instance (a class already available in Smalltalk-80).

Instance variable names:

1. name
2. serial
3. entryTime
4. queueEntryTime
5. currentPosition
6. currentWorkStation
7. destination
8. destinationPosition
9. bypass
10. tempDestination
11. status
12. done
13. county

The instance variable name will store the name of the Cart instance created. Each cart instance created will have a number attached to it to identify it from other instances. This number is stored in the instance variable serial. The entryTime instance variable is set equal to the creation time of each Cart instance created. The storage of this value allows the time in system statistics to be collected for each cart passing through the system. The instance variable queueEntryTime is set equal to the time the Cart instance joins a queue of a workstation. The currentPosition instance variable is set equal to the current position of the Cart instance. This position is the same as the position of the workstation the Cart instance is currently getting serviced at. The currentWorkStation is set equal to the name of the workstation the cart instance is currently being processed at. The destination is the instance variable that stores the next workstation for the cart. The destinationPosition is the position of the cart' s destination. The instance variable bypass stores the object named BYPASS. Bypasses are typically transfer lines that will allow a cart to recirculate in the conveyor loop always via a shorter route. When the cart

instance chooses to move along a bypass to reach its destination, its original destination is stored in a temporary storage location set equal to the instance variable tempDestination. The instance variable status will store the status of the cart instance which will have a value of one if the cart is carrying a load order or zero if the cart is empty. The instance variable done will be either equal to 'true' if the cart instance is done with all its tasks or 'false' if the cart is not done with its tasks. Since the system is a recirculating conveyor system this value will always have the value 'false' which allows the cart instance to keep recirculating in the system till the simulation is terminated.

Software methods

Class methods

1. name: aName
2. initialize
3. returnInstanceWithSerial: aSerial

The initialize method will initialize the class variable mentioned above to their initial instances. The last method returnInstanceWithSerial: aSerial will return the cart object with the serial number aSerial.

Instance methods:

1. initialize
2. initName: aName
3. accessing methods
4. initialize-release methods
5. pause
6. resume
7. startUp
8. tasks
9. checkShortestRouteFrom: aWorkStation
10. completeOperationsAt aLocation
11. finishUp

The first method, initialize, will initialize the cart instances as required by the system. The instance variable currentWorkStation of all the created carts will be initialized to PPLS since, all the carts will start at this workstation with an empty status. For the sake of brevity, the author included all the methods that request and return the

values of an object instance variables under the accessing category. The methods under this category will allow the access of all the instance variables in addition to the resourceNeeded by the Cart instance and the amountNeeded of this resource. Moreover, all the methods initializing these instance variables are grouped under the initialize-release category. The Cart instance in carrying its activities will follow the simulation framework referred to as the "life cycle of the object" which involves the sequence startUp_tasks_finishUp. When the Cart instance first arrives to the simulation it is sent the message startUp which will set its entryTime to the Simulation active time and its initial position to the PPLS workstation. Within the same method the Cart instance is sent the method tasks. The method tasks will specify the sequence of actions the Cart instance has to perform. First the Cart instance has to acquire a workstation (acquireResource) and then request service. If this request is successful, the Cart instance is sent the method completeOperationsAtLocation: aLocation. At this point the Cart instance will be serviced and assigned a new destination by the WorkStation. At the end of this method the cart will be put in the output queue of the workstation waiting for a hook to be available to carry it to its next destination. In order to allow the recirculation of Cart instances the instance variable done is never set to true during the simulation. Therefore, the tasks block of actions is repeatedly performed till the simulation is terminated. Figure 14 illustrates the message flow diagram for the Cart object. The message flow diagram is simply a network in which the nodes represent objects and the arcs connecting them represent messages. The arcs are numbered sequentially to show the order in which messages are sent.

WorkStation. This class embodies the behavior of a work station or machine. It has three queues (input, output, and empty), a processor and a queue controller.

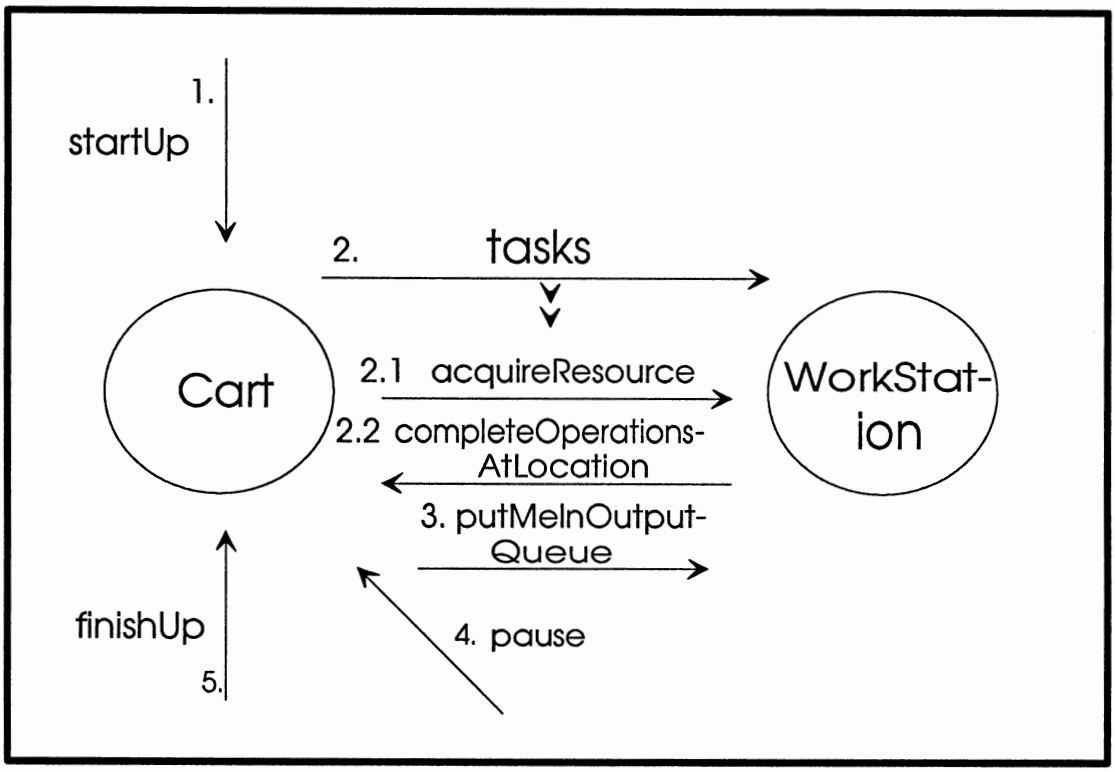

Figure 14. Message Flow Diagram for the Cart Object

Data Storage

Class variable names:

1. Destinations

This Destinations class variable is a storage location pointing to a Dictionary instance (a general Smalltalk-80 class). This dictionary functions to allow the storage of the pointers to the different workstations that could exist in the system (the pointers will be the workstation names).

Instance variable names:

1. name
2. wsType
3. enterPosition
4. exitPosition
5. processTime
6. wsProcessingTimes
7. probabilities
8. wsQueueController
9. wsAmountAvailable
10. loadCount

The WorkStation' s type will be carried by the instance variable wsType. The instance variables enterPosition and exitPosition define respectively the position of the input queue and the output queue of the WorkStation instance. The instance variable processTime will be set equal to the time it takes a specific WorkStation instance to service a cart. The instance variable probabilities is a storage location pointing to a Dictionary instance. The locations in the Dictionary instance will carry the values of the probabilities with which a WorkStation instance sends a Cart instance to another WorkStation instance. The instance variable wsQueueController is an instance of the object QueueController. The wsAmountAvailable is an integer set either to one when the WorkStation instance is idle or zero otherwise.

Software methods

Class methods:

1. generateAndLoadOrders
2. returnInstanceWithName: aName

The method generateAndLoadOrders will allow repeated scheduling of the arrival of load orders according to a user specified probability distribution. Once the load order is scheduled it should check if there is any empty cart available to start loading (through the method tryToLoad) and schedule the next arrival of a load order. The last method returnInstanceWithName: aName will allow the WorkStation class to return the Workstation instance with name aName.

Instance methods:

1. initializeWithName: aString andAmount: aNumber exitPosition: ExitPositon enterPosition: aEnterPosition probabilities: aVector processTime: aTime type: aType
2. accessing methods
3. getaRandomNumber
4. getNextDestination
5. provideServiceTo: aCart
6. provideServices
7. release: anAmount
8. tryToLoad
9. printOn: aStream
10. printResultsOn: aStream

The first method will initialize all the instance variables of the WorkStation instance. All the methods requesting the values of the instance variables were grouped under the accessing category. In order for the WorkStation instance to assign a new destination for the Cart instance it will get a random number by sending to itself the method getaRandomNumber. The random number returned from this message will be checked against the probabilities with which a WorkStation instance sends Cart instances to other WorkStation instances. The destination for the Cart instance will be the WorkStation instance in the DestinationList that satisfied a match between the

probability requirements and the random number generated. This will be done within the method getNextDestination. When a cart acquires a resource (WorkStation instance), it will ask it to provideService to it. In response to the message provideServiceTo: aCart, the WorkStation instance delays the cart for the required amount of time (which simulates the processing of the Cart at the WorkStation if the machine is idle). A busy WorkStation enqueues the Cart requesting its service. Once the cart is serviced the WorkStation instance should be made available (incrementing the number of wsAmountAvailable by the amount that was acquired by the Cart instance). This is done by sending the message release: anAmount to WorkStation instance. The method tryToLoad is invoked whenever a load order is generated to check if loading is possible (e.g. is there an empty cart available ?). The last two methods are the methods needed to print statistics on the WorkStation's utilization and its instance queues. The output form of the WorkStation instance is defined through the method printOn: aStream.

Hook. Another important object in the simulation model is the Hook object. This object will be created as a subclass of WorkStation object. In addition, Hook instances will be specified by their positions and numbers in the conveyor loop. At any time in the simulation the model developer should be able to identify the status and the position of the Hook instance. These attributes of the Hook instance will be stored in its instance variables. The next class to be considered is the class conveyor. This class will allow a meaningful place to put methods dealing with all the Hook instances.

Data storage

Class variable names

Instance variable names

space
track
trackSize

inputPos
outputPos
numWorkStations

The instance variable space is set equal to the distance between two consecutive Hook instances. The track instance variable is a storage location pointing to a Dictionary that functions to allow the storage of all the Hook instances created. Therefore, the trackSize will always be equal to the number of Hook instances needed to be created in the system. The inputPos is a storage location pointing to a Dictionary instance. This Dictionary will store the positions of the input queues of the WorkStation instances in the system. This explanation applies to the outputPos instance variable except that the Dictionary stores the positions of the output queues of the WorkStation instances. The numWorkStations is set equal to the number of WorkStations instances that will be created in the system.

Software methods

Class methods:

1. new

The new method functions to allocate memory space for the representation of a new Conveyor. In addition, it sends a message to the new Conveyor instance to initialize itself through the use of the initialize instance method.

Instance methods:

1. initialize
2. accessing methods
3. updateHookPositions
4. move
5. acceptCarts
6. deliverCarts

The initialize method will allow the storage of the Hook instances in the track (this is the same thing as a conveyor loop). At this point a reference point is selected on the track and Hook instances are assigned numbers sequentially starting form one. The hookNumber (defined above) is the same as the hookPosition when no movement has

occurred yet. All the Hook instances start with an idle status. Accessing methods will be used to return the values of the instance variables at any simulated time in the simulation run. The instance method updateHookPositions1: aNumber functions to update the Hook instances positions after the conveyor has moved through a certain number of hook spaces equal to aNumber. The Hook instances numbers (hookNumber) are not changed, only their positions are changed (hookPosition). Cart instances will be transported to their destinations by sending the message move: aNumber. This method will schedule the arrival of Hook instances to the WorkStation instances repeatedly after a certain number of hook spaces. At this point all the Hook instances in front of the input and output queues of the WorkStation instances will check if they can deliver carts to the WorkStations or accept to transport carts from their current workstation to their destinations. In the method acceptCarts, the Conveyor checks the status of all the Hook instances in front of the workstation's output queues. If a Hook instance is is not carrying a cart it checks if there is a cart waiting for transportation. If a cart is waiting in the output queue of the workstation, the Hook instance removes it and adds it to its input queue. The Hook instance will always check whether or not the destination of the Cart instance lies along the shortest route. After checking if the Hook instances can transport carts to their destinations, the Conveyor checks if the Hook instances in front of the input queues of the workstations can deliver the carts they are carrying. If there is space in the workstation's input queue, the cart leaves the hook and joins the input queue for service at the workstation. It is at this point and only at this point that the Cart instance is resumed to carry out the block of actions in its method tasks one more time. If the workstation does not have input space the cart is recirculated through the shortest route before it tries to acquire the workstation again. Figure 15 illustrates the message diagram for this object.

For the sake of brevity, and because its description will not help the reader or give him/her more insight in understanding the simulation model, the class

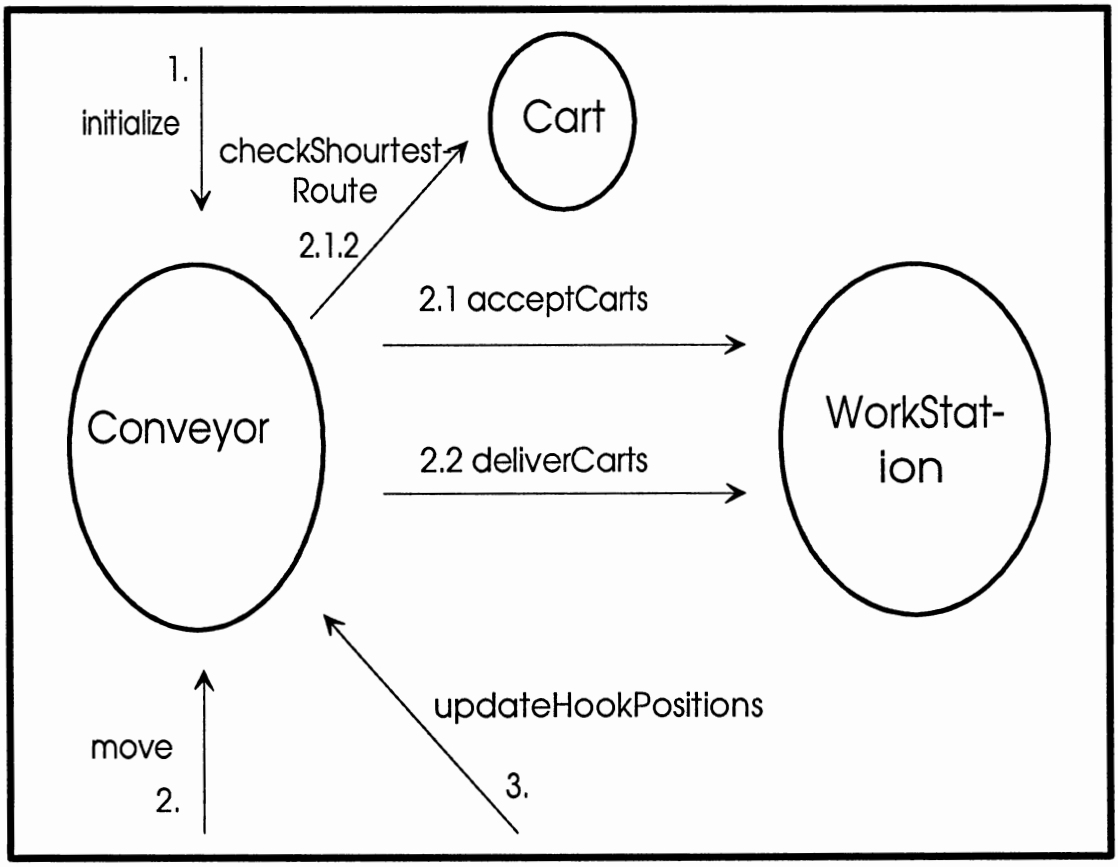

Figure 15. Message Diagram for the Conveyor Object

QueueController will not be explained here. Detailed description will however, be included in Appendix A.

Summary

This section discussed in some detail the implementation of several representative simulation processing and element objects. This coverage is intended to provide the reader a basic understanding of the code listed in Appendix A.

Target System Simulation Model Representation:

An Illustrative Example

An Illustrative Example

In this section an example model is executed. The following tables include the parameters specification of the different workstations and their probability vectors and the conveyor object of the example model.

TABLE 1

WORKSTATION PARAMETERS SPECIFICATION

Facility Name	Type	Process Time	Exit Position	Enter Position
UALSC	1	Normal (4, 0.5)	8	7
LSWAT	2	Normal (4, 0.5)	24	23
LODST	3	Normal (4, 0.5)	38	37
OLUNLOD	4	0	43	42
ULST	5	Normal (3, 0.4)	32	31
BYPASS	6	8	44	17
PPLS	7	Normal (4, 0.5)	5	1

TABLE 2

CONTINUED WORKSTATION PARAMETERS SPECIFICATION

Facility Name	Input Queue Cap.	Output Queue Cap.	Empty Queue Cap.	Prob. Vector *
UALS	5	5	2	(0.0, 0.2, 0.1, 0.3, 0.3, 0.0, 0.1)
LSWAT	5	5	2	(0.2, 0.0, 0.0, 0.4, 0.4, 0.0, 0.0)
LODST	5	5	2	(0.2,0.0,0.0, 0.4, 0.4, 0.0, 0.0)
OLUNLOD	10	10	0	(0.2, 0.3, 0.3, 0.0, 0.0, 0.0, 0.2)
ULST	5	5	0	(0.2, 0.3, 0.3, 0.0, 0.0, 0.0, 0.2)
BYPASS	5	5	0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
PPLS	20	20	0	(0.4, 0.0, 0.0, 0.3, 0.3, 0.0, 0.0)

* The vector entries follow the order of the facility name column. That is for the Prob. Vector of the UALSC the 0.2 in the second entry represents the probability from which the UALSC will send a cart to the workstation LSWAT.

TABLE 3

CONVEYOR PARAMETERS SPECIFICATION

Conveyor		
Speed	Space	No. Hooks
1 ft/min	10 ft	48

The model will have 20 carts and a simulation end time of 700 minutes. The results of executing this model in the Smalltalk-80 environment is shown in Figure 16.

Verification and Validation

Verification of the developed simulation software and validation of the OOM conceptual approach to simulation model generation have been addressed. Verification of the object-oriented simulation software was performed through the close scrutiny and testing (debugging and tracing) of the simulation classes during the software implementation phase. An additional measure of modeling construct verification was achieved through the successful completion of the validation process. During the construction of the simulation program, a systematic series of steps were used to validate the program. The objective of validation was to establish that each system object responded in a logical manner like an actual system.

The first phase of the program validation was accomplished during the early stages of construction of software objects. The different situations which arise at each module were first carefully examined and exhaustively enumerated. The system logic was then simulated using Lotus 123 to insure its correctness before the coding of the computer program.

The second phase of validation consisted of implementing each software object by itself first to make sure that it was doing what it was intended to do by going to the Workspace window and inspecting the values of all its class and instance variables. The implementation of the software objects was also done with an incremental style of development. That is, object capabilities were added incrementally to make sure that every feature added to the object was functioning correctly. Several cases of software object inputs (arrival distributions, processing times distributions, etc.) were deterministically programmed and the output was analyzed to insure the correct functioning of the software object under most situations.

A third phase of validation consisted of testing the relationship between objects to verify that the program responds like a real system and exhibits the same logical

Simulation initiated at 11:58:36 am, 24 April 1992
Simulation Experimentation Seed : 1234 Simulation End Time : 700
Statistics Cleared At : 0 Traced From : 0

Statistics for WorkStation: BYPASS

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	5	0.0228571	0.149448	0.0	1.0	0.0
Proc.Time:	3	5.33333	4.6188	0.0	8	8
Input Queue:						
Length:	5	0.0	0.0	0.0	1.0	0.0
Wait Time:	3	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	5	0.00571429	0.0753766	0.0	1.0	0.0
Wait Time:	3	1.33333	1.1547	0.0	2.0	2.0
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for WorkStation: LODST

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	7	0.0172491	0.130198	0.0	1.0	0.0
Proc.Time:	4	3.01859	2.02027	0.0	4.26975	3.95331
Input Queue:						
Length:	7	0.0	0.0	0.0	1.0	0.0
Wait Time:	4	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	7	0.025608	0.157963	0.0	1.0	0.0
Wait Time:	4	4.48141	2.99291	0.0	6.14868	6.04669
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for WorkStation: LSWAT

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	3	0.00614973	0.0781787	0.0	1.0	0.0
Proc.Time:	2	2.05385	2.90458	0.0	4.10769	4.10769
Input Queue:						
Length:	3	0.0	0.0	0.0	1.0	0.0
Wait Time:	2	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	3	0.00813599	0.089832	0.0	1.0	0.0
Wait Time:	2	2.8476	4.02711	0.0	5.69519	5.69519
empty Queue:						
Length:	3	2.81634e-4	0.0167796	0.0	1.0	0.0
Wait Time:	2	0.0985718	0.139402	0.0	0.197144	0.197144

Figure 16. Result View

Statistics for WorkStation: OLUNLOD

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	23	0.0	0.0	0.0	1.0	0.0
Proc.Time:	12	0.0	0.0	0.0	0.0	0
Input Queue:						
Length:	23	0.0	0.0	0.0	1.0	0.0
Wait Time:	12	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	23	0.157143	0.363935	0.0	1.0	0.0
Wait Time:	12	9.16667	2.88675	0.0	10.0	10.0
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for WorkStation: PPLS

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	50	0.14322	0.350297	0.0	1.0	0.0
Proc.Time:	25	4.01015	0.518814	3.05248	5.20612	4.0428
Input Queue:						
Length:	49	1.10393	3.62384	0.0	19.0	0.0
Wait Time:	25	30.9101	26.5978	0.0	76.638	0.0
Output Queue:						
Length:	51	1.86713	3.33182	0.0	12.0	0.0
Wait Time:	26	50.2689	38.4304	0.0	119.3	35.9572
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for WorkStation: UALSC

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	17	0.0844311	0.278033	0.0	1.0	0.0
Proc.Time:	15	3.94012	1.12848	0.0	4.64781	4.50024
Input Queue:						
Length:	17	0.0	0.0	0.0	1.0	0.0
Wait Time:	9	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	17	0.0298546	0.170186	0.0	1.0	0.0
Wait Time:	9	2.32203	1.91283	0.0	5.49976	5.49976
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Figure 16. (continued)

Statistics for WorkStation: ULST

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	19	0.0378806	0.190908	0.0	1.0	0.0
Proc.Time:	10	2.65165	0.967777	0.0	3.34706	3.14901
Input Queue:						
Length:	19	0.0	0.0	0.0	1.0	0.0
Wait Time:	10	0.0	0.0	0.0	0.0	0.0
Output Queue:						
Length:	19	0.0906908	0.287169	0.0	1.0	0.0
Wait Time:	10	6.34836	2.24589	0.0	7.58289	6.85101
empty Queue:						
Length:	1	0.0	0.0	0.0	0.0	0.0
Wait Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #1

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.9	0.3	0.0	1.0	1.0
Proc.Time:	3	120.0	190.788	0.0	340	340

Statistics for Hook: Hook #2

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.928571	0.257539	0.0	1.0	1.0
Proc.Time:	2	185.0	261.63	0.0	370	370

Statistics for Hook: Hook #3

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	5	0.614286	0.486764	0.0	1.0	0.0
Proc.Time:	3	263.333	229.42	0.0	420	420

Statistics for Hook: Hook #4

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.942857	0.232115	0.0	1.0	1.0
Proc.Time:	3	116.667	132.035	0.0	260	90

Statistics for Hook: Hook #5

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	5	0.614286	0.486764	0.0	1.0	0.0
Proc.Time:	3	263.333	229.42	0.0	420	420

Figure 16. (continued)

Statistics for Hook: Hook #6

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	2	0.314286	0.464231	0.0	1.0	1.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #7

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #8

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	2	0.342857	0.474664	0.0	1.0	1.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #9

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #10

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #11

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #12

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Figure 16. (continued)

Statistics for Hook: Hook #13

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #14

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #15

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #16

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #17

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	3	0.0714286	0.257539	0.0	1.0	0.0
Proc.Time:	2	215.0	304.056	0.0	430	430

Statistics for Hook: Hook #18

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #19

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	3	0.0714286	0.257539	0.0	1.0	0.0
Proc.Time:	2	215.0	304.056	0.0	430	430

Figure 16. (continued)

Statistics for Hook: Hook #20

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	2	0.514286	0.499796	0.0	1.0	1.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #21

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #22

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.528571	0.499183	0.0	1.0	1.0
Proc.Time:	2	130.0	183.848	0.0	260	260

Statistics for Hook: Hook #23

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #24

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #25

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #26

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Figure 16. (continued)

Statistics for Hook: Hook #27

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #28

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #29

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #30

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #31

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #32

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Statistics for Hook: Hook #33

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	1	0.0	0.0	0.0	0.0	0.0
Proc.Time:	1	0.0	0.0	0.0	0.0	0.0

Figure 16. (continued)

Statistics for Hook: Hook #34

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.7	0.458258	0.0	1.0	1.0
Proc.Time:	2	130.0	183.848	0.0	260	260

Statistics for Hook: Hook #35

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.7	0.458258	0.0	1.0	1.0
Proc.Time:	3	103.333	137.961	0.0	260	50

Statistics for Hook: Hook #36

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	10	0.685714	0.464231	0.0	1.0	1.0
Proc.Time:	5	68.0	92.5743	0.0	230	40

Statistics for Hook: Hook #37

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.742857	0.437059	0.0	1.0	1.0
Proc.Time:	2	130.0	183.848	0.0	260	260

Statistics for Hook: Hook #38

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.757143	0.428809	0.0	1.0	1.0
Proc.Time:	2	185.0	261.63	0.0	370	370

Statistics for Hook: Hook #39

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	5	0.157143	0.363935	0.0	1.0	0.0
Proc.Time:	3	36.6667	47.2582	0.0	90	90

Statistics for Hook: Hook #40

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.771429	0.419913	0.0	1.0	1.0
Proc.Time:	3	243.333	210.792	0.0	370	360

Figure 16. (continued)

Statistics for Hook: Hook #41

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	5	0.157143	0.363935	0.0	1.0	0.0
Proc.Time:	3	36.6667	47.2582	0.0	90	90

Statistics for Hook: Hook #42

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	10	0.728571	0.444697	0.0	1.0	1.0
Proc.Time:	5	116.0	143.631	0.0	310	20

Statistics for Hook: Hook #43

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.828571	0.376883	0.0	1.0	1.0
Proc.Time:	2	185.0	261.63	0.0	370	370

Statistics for Hook: Hook #44

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.842857	0.363935	0.0	1.0	1.0
Proc.Time:	2	185.0	261.63	0.0	370	370

Statistics for Hook: Hook #45

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.842857	0.363935	0.0	1.0	1.0
Proc.Time:	3	243.333	210.792	0.0	370	360

Statistics for Hook: Hook #46

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	6	0.857143	0.349927	0.0	1.0	1.0
Proc.Time:	3	103.333	137.961	0.0	260	50

Statistics for Hook: Hook #47

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.885714	0.318158	0.0	1.0	1.0
Proc.Time:	2	185.0	261.63	0.0	370	370

Statistics for Hook: Hook #48

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
Utilization:	4	0.9	0.3	0.0	1.0	1.0
Proc.Time:	2	130.0	183.848	0.0	260	260

Average Utilization Of All The Hooks Is: 0.343155

Figure 16. (continued)

Utilization for Carts:

	No.Obs.	Average	Std.Dev	Minimum	Maximum	Current
cart serial # 1	3	0.9	0.3	0.0	1.0	1.0
cart serial # 2	3	0.422959	0.494029	0.0	1.0	1.0
cart serial # 3	3	0.887884	0.315509	0.0	1.0	1.0
cart serial # 4	2	0.567674	0.495399	0.0	1.0	0.0
cart serial # 5	4	0.569842	0.495098	0.0	1.0	0.0
cart serial # 6	2	0.427681	0.494742	0.0	1.0	0.0
cart serial # 7	2	0.593815	0.49112	0.0	1.0	0.0
cart serial # 8	3	0.87437	0.331432	0.0	1.0	1.0
cart serial # 9	3	0.769483	0.421164	0.0	1.0	1.0
cart serial # 10	2	0.621124	0.485107	0.0	1.0	0.0
cart serial # 11	2	0.627972	0.483346	0.0	1.0	0.0
cart serial # 12	7	0.666488	0.471467	0.0	1.0	1.0
cart serial # 13	5	0.366326	0.4818	0.0	1.0	0.0
cart serial # 14	3	0.738271	0.439576	0.0	1.0	1.0
cart serial # 15	4	0.38305	0.48613	0.0	1.0	0.0
cart serial # 16	2	0.669854	0.470265	0.0	1.0	0.0
cart serial # 17	2	0.521381	0.499543	0.0	1.0	0.0
cart serial # 18	6	0.594847	0.490922	0.0	1.0	0.0
cart serial # 19	3	0.809408	0.392768	0.0	1.0	1.0
cart serial # 20	2	0.54766	0.497723	0.0	1.0	0.0

Average Utilization Of All The Carts Is: 0.658669

Simulation Ended at #(24 April 1992 12:04:10 pm)

Figure 16. (continued)

relationship between the interface and functional objects as in the actual system. In order to accomplish this, the types of event occurrences were exhaustively enumerated and the logic was traced by hand for each type of event.

The fourth phase of validation was accomplished by building a small hypothetical conveyor system simulation model that consisted of at least one of each of the system objects. A series of trial simulation runs were made. In addition, several model inputs were deterministically programmed and the output was analyzed to insure the correctness of logic under all situations. Using the simulation trace-on feature available in Smalltalk-80 (see Figure 17), the flow of carts through the system was checked to insure that they follow the proper sequence.

Summary

The last few sections describe the procedure employed to validate the conceptual organization of the OOM prototype environment for the generation of discrete simulation models of a constant speed, discretely spaced recirculating conveyor system. In the next chapter, the OOM environment will be evaluated through the use of both tangible and intangible features within an organization structure made up of an Analytical Hierarchy Process decision model.

```
31.3618
Cart cart serial# 8 released 5
Cart cart serial# 8 going to WorkStation named: ULST
cart  is at: PPLS
Cart cart serial# 9 processing time required at  WorkStation named: PPLS  3.85135
Cart cart serial# 9 obtained WorkStation named: PPLS
35.2132
Cart cart serial# 9 released 5
Cart cart serial# 9 going to WorkStation named: OLUNLOD
cart  is at: PPLS
Cart cart serial# 10 processing time required at  WorkStation named: PPLS  5.20612
Cart cart serial# 10 obtained WorkStation named: PPLS
39.3904
40.0
Cart cart serial# 4 obtained aHook2
a Conveyor all the hook positions are updated at 40.0
40.4193
Cart cart serial# 10 released 5
Cart cart serial# 10 going to WorkStation named: OLUNLOD
cart  is at: PPLS
Cart cart serial# 11 processing time required at  WorkStation named: PPLS  4.9199
Cart cart serial# 11 obtained WorkStation named: PPLS
45.3392
Cart cart serial# 11 released 5
Cart cart serial# 11 going to WorkStation named: OLUNLOD
cart  is at: PPLS
Cart cart serial# 12 processing time required at  WorkStation named: PPLS  3.91146
Cart cart serial# 12 obtained WorkStation named: PPLS
```

Figure 17. Simulation Trace

CHAPTER VII

EVALUATION OF THE APPROACH THROUGH ANALYTIC HIERARCHY PROCESS

This chapter summarizes the Analytic Hierarchy Process used to compare the proposed modeling approach against conventional simulation paradigm.

Analytic Hierarchy Process

This chapter describes the application of the Analytical Hierarchy Process (AHP) to evaluate various aspects of the developed framework and the methodology against the conventional simulation paradigm. A detailed explanation of the AHP process will not be covered here and can be obtained from Saaty (1988). In simple terms, AHP is a multi-criteria decision methodology that utilizes structured pair wise comparisons among similar aspects of alternatives to reach a scale of preference. A more comprehensive application of AHP for simulation environment evaluation purposes can also be seen from the study done by Beaumariage (1990). In his study Beaumariage compares object-oriented simulation environments against traditional environments such as SLAM II and SIMAN. This study also provides a summarized guideline of the AHP application process. Therefore, the process of developing an AHP model will not be covered here, but can be obtained from one of the resources mentioned above.

The preliminary AHP model developed by the author was discussed, critiqued, and iterated on by an AHP study group that consisted of Hemant Bhuskute, Manoj Duse, Jagannath Gharpure (three PHD candidates in Industrial Engineering and

Management at Oklahoma State University), and the author. The group started by developing the levels, the major aspects, and the criteria in terms of a set of nodes. The next task was defining the linkages between these nodes. Finally, the group assigned weights to the preference matrices resulting from the above two tasks. The following paragraphs summarize the resulting level, major aspects, criteria and weight matrices of the group's study.

Level 1 : Definition of the Problem

1.1 - Simulation Paradigm. The goal of this analysis is the choice of the best means of conceptualizing and representing the system in terms of a simulation model along with underlying structures.

Level 2 : Major Aspects

2.1 - Model Effectiveness. This aspect is concerned with the ability of the model to represent critical aspects of the real system. This includes the ability of the model to manage future modeling extensions, alterations, and reuse needs, etc.

2.2 - Model Developer's Capability and Modeling Effort. This aspect is concerned with the capabilities of the model developer and the effectiveness of his/her efforts. The person or task associated with simulation model development is involved in the use of currently available constructs (new base code) within the development of useful simulation models.

2.3 - Performance Considerations. This system aspect addresses basic hardware related performance measures.

Level 3 : Criteria Considered

3.1 - Model Reusability. This criteria mainly addresses the ability of specific models or portions of models to be used through a change process. The capability of

developing models with different levels of detail without major model overhauls is also part of the flexibility. This node links to 2.1 and 2.2.

3.2 - Change Management Capability. Because our concept of a simulation environment is that of a growing, changing system, we must consider software change management to be an important capability to allow for simulation model reconfigurability. This node is linked to 2.1 and 2.2.

3.3 - Software Modularity. This is the simulation paradigm's ability to represent physical, information, and control components of the system under study in a modular fashion. This criteria increases the validity of the model, thereby promoting the credibility of the whole simulation study (Karacal, 1991). This node links to 2.1 and 2.2.

3.4 - Output Provisions. This simulation criteria reefers to the degree of the simulation model's support for both standard and special results output from a simulation run. This criteria has a strong influence on model effectiveness and therefore is linked to node 2.1.

3.5 - Model Debugging Support/Verification. This criteria addresses the features provided for model debugging and verification and the degree of effectiveness achieved by these features. This node links to 2.2.

3.6 - Graphics/User Interface Capability. This criteria is very important for simulation environment enhancements. This node is linked to 2.1

3.7 - Execution Speed. The CPU time required to run the simulation model represents the execution speed. This criteria is linked to 2.3.

3.8 - Simulation Language Knowledge/Ease of Learning Effort Required. This addresses the amount of knowledge needed to use a simulation system and the effort required to learn the system. This node is linked to 2.2.

3.9 - Basic Memory Requirements. This criteria is concerned with computer memory requirements. This characteristic addresses the amount of memory needed to

run the simulation environment for the smallest of models. This node is linked to node 2.3.

3.10. - Model Representation Correspondence to the Real System. This aspect evaluates how accurately the real system can be expressed in the model. As the degree of model correspondence to the real system decreases, the degree of model abstraction increases. Model abstraction refers to the degree to which the representation of the system (the simulation model) is conceptually removed from the actual system. This node is linked to 2.1 and 2.2.

3.11. - Modeling Flexibility. This criterion addresses the kinds of features for development of higher level constructs (the grouping of model portions in a way that supports the conceptual grouping of a system) that are available and the manner in which new constructs fit in with the normal simulation model specification mode.

Level 4 - Alternative Simulation Paradigms

4.1 - Traditional, Special Purpose Simulation Systems. This alternative represents the standard simulation system typically used in discrete event modeling. Simulation systems under this category are commercially available. This node is linked to all the nodes at level 3.

4.2 - OOP Simulation System. This alternative represents the new OOP simulation system, for which the prototype system was developed. This node is linked to all the nodes at level 3.

Figure 18 shows the AHP hierarchical diagram. Tables 4 through 18 show the original weights of the AHP matrices agreed upon by the study group.

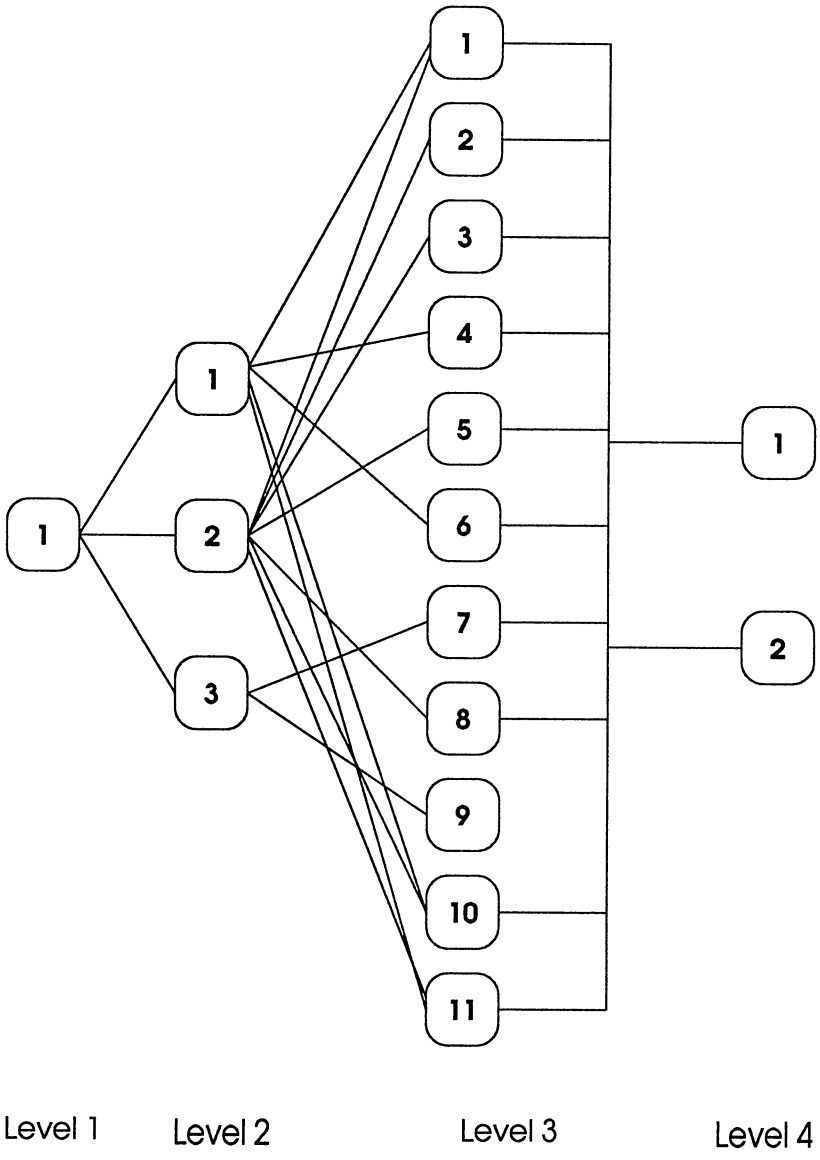

Figure 18. AHP Simulation Language Comparison Model

TABLE 4

NODE 1.1

NODE 1.1 - Simulation Paradigm

Links from Lower Level

1) Node 2.1 - Model effectiveness
2) Node 2.2 - Model developer's potency and modeling effort
3) Node 2.3 - Performance considerations

Original weights

Col	1	2	3
Row			
1	1.000	7.000	9.000
2	0.143	1.000	5.000
3	0.111	0.200	1.000

TABLE 5

NODE 2.1

NODE 2.1 - Model Effectiveness

Links from Lower Level

1) Node 3.1 - Model reusability
2) Node 3.4 - Output provisions
3) Node 3.6 - Graphics/User interface capability
4) Node 3.10-Model representation correspondence to the real system
5) Node 3.11-Model flexibility

Original weights

Col	1	2	3	4	5
Row					
1	1.000	0.200	1.000	0.200	0.143
2	5.000	1.000	5.000	3.000	1.000
3	1.000	0.200	1.000	0.200	0.167
4	5.000	0.333	5.000	1.000	4.000
5	7.000	1.000	6.000	0.250	1.000

TABLE 6

NODE 2.2

NODE 2.2 - Model Developer's Potency and Modeling Effort

Links from Lower Level

1) Node 3.1 - Model reusability
2) Node 3.2 - Change management capability
3) Node 3 3 - Software modularity
4) Node 3.5 - Model debugging support/Verification
5) Node 3.8 - Simulation language knowledge/Ease of learning effort required
6) Node 3.10-Model representation correspondence to the real system
7) Node 3.11-Modeling flexibility

Original weights

Row \ Col	1	2	3	4	5	6	7
1	1.000	3.000	2.000	0.250	0.167	0.333	1.000
2	0.333	1.000	2.000	0.250	0.500	0.500	0.333
3	0.500	0.500	1.000	0.250	0.333	1.000	0.500
4	4.000	4.000	4.000	1.000	1.000	3.000	2.000
5	6.000	2.000	3.000	1.000	1.000	3.000	2.000
6	3.000	2.000	1.000	0.333	0.333	1.000	1.000
7	1.000	3.000	2.000	0.500	0.500	1.000	1.000

TABLE 7

NODE 2.3

NODE 2.3 - Performance Considerations

Links from Lower Level

1) Node 3.7 - Execution speed
2) Node 3.9 - Basic memory requirements

Original weights

Row	Col 1	2
1	1.000	0.125
2	8.000	1.000

TABLE 8

NODE 3.1

NODE 3.1 - Model Reusability

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.143
2	7.000	1.000

TABLE 9

NODE 3.2

NODE 3.2 - Change Management Capability

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.200
2	5.000	1.000

TABLE 10

NODE 3.3

NODE 3.3 - Software Modularity

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.333
2	3.000	1.000

TABLE 11

NODE 3.4

NODE 3.4 - Output Provisions

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.125
2	8.000	1.000

TABLE 12

NODE 3.5

NODE 3.5 - Model Debugging Support/Verification

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.125
2	8.000	1.000

TABLE 13

NODE 3.6

NODE 3.6 - Graphics/User interface capability

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col	1	2
	1	1.000	0.167
	2	6.000	1.000

TABLE 14

NODE 3.7

NODE 3.7 - Execution Speed

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col	1	2
	1	1.000	6.000
	2	0.167	1.000

TABLE 15

NODE 3.8

NODE 3.8 - Simulation Language K knowledge/Ease of Learning Effort Required

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	5.000
2	0.200	1.000

TABLE 16

NODE 3.9

NODE 3.9 - Basic Memory Requirements

Links from Lower Level

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	7.000
2	0.143	1.000

TABLE 17

NODE 3.10

NODE 3.10 - Model Representation Correspondence to the Real System
Links from Lower Level
1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.111
2	9.000	1.000

TABLE 18

NODE 3.11

NODE 3.11 - Modeling Flexibility
Links from Lower Level
1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.143
2	7.000	1.000

The next step in the AHP procedure was the calculation of the relative weights of the decision elements. Appropriate spreadsheets were developed for this purpose. These spreadsheets calculated the priorities for each of the above matrices along with matrix consistencies (Appendix C). After, checking the consistencies, these relative weights were aggregated through a series of matrix calculations to yield a solution to the problem. Table 19 shows the resulting final weights.

TABLE 19

FINAL WEIGHTS

Final Solution Weights	
Traditional, special purpose simulation systems:	0.225
OOP simulation systems:	0.775

Summary

The results of final weights obtained from the AHP evaluation clearly indicate that the OOP simulation systems are superior to the traditional special purpose simulation systems in terms of the aspects and criteria considered in the AHP study. The conclusion reached in this study is in agreement with Beaumariage's (1990) and Karacal's (1991) results, which were obtained using a different set of factors.

CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

This chapter summarizes the conclusions reached, research contributions, and recommendations of this study.

Conclusions

The main objective of this research was to develop, validate, and document utility "plug-in" modular component computer simulation models which may be used to interpret and synthesize the operating characteristics of various types of complex recirculating conveyor systems, using an Object-Oriented Modeling environment. To achieve this objective, three sub-objectives had to be accomplished.

The first objective of this research was to develop a library of reusable software objects which would provide the ability to generate the simulation model desired. In order to fulfill this objective, several tasks had to be performed. First, the functional components that are common to representative existing recirculating conveyor systems were examined and analyzed. Next, the types of interfacing that can occur between the functional components of a recirculating conveyor system were specified. Then, a series of modular elements to represent the functioning of the objects and message passing among them were developed and encoded within the general software environment Smalltalk-80. Finally, the software simulation objects were incorporated into flexible utility simulation systems that can be utilized to "build" complex conveyor system simulations, thus successfully completing the first research objective.

The second objective of the research was to develop an approach which would allow the comparison of modeling environments. In order to accomplish this, criteria for comparing simulation modeling environments were developed. Using these criteria, the decision problem, choosing the best simulation environment was addressed through the application of the Analytic Hierarchy Process. For the problem of interest, namely comparison of simulation environments, an AHP model was developed. This involved the determination of an appropriate scheme for decision process decomposition along with the linkages between elements in the decision model. This model will allow the comparison of the new OOM system to traditional simulation systems. This will also provide an evaluation of the quality of the developed prototype. In order to complete this evaluation, a group of simulationists experienced in both of the alternatives provided the many pairwise comparisons required by the Analytic Hierarchy Process and the developed model. The comparisons were then manipulated to result in a final set of weights concluding the preferable simulation approach. The conclusion from the AHP evaluation was that an object-oriented modeling environment is superior to the traditional simulation modeling environment. This completed the second objective successfully.

The third objective of this research was to explore ways to expand the model developed to accommodate all the complexities that a conveyor system can incorporate. Given the restricted time frame of completion of this work, the author was not able to implement some of the features incorporated in the existing conveyor system. This includes having multiple loops, multiple floors, and multiple bypasses. A detailed description of how to incorporate these features in the developed model is presented in the following section. This represents the accomplishment of the third research objective.

The final conclusions from this research are :

1) Most simulation languages available today are well suited for developing a model of a system from scratch. These languages are excellent and have met and continue to meet the needs of many applications. However, the analyst generally concentrates on the model building activities of a specific scenario. Major changes in the system model such as changes in the structure of the system and their impact on the system performance cannot be easily handled. Thus, the traditional languages facilitate model reusability only to a limited degree. Object-Oriented programming provides a framework for simulation software implementation of a highly reusable modeling environment.

2) In general, object-oriented programming offers a lot of improvements in the accomplishment of the traditional modeling tasks (these have been discussed in the previous chapters). This should encourage simulation package developers and simulation system users to pursue simulation modeling within an object-oriented implementation.

Recommendations For Future Implementations

Developing software as complex as the Conveyor System model requires a lot of research, insight, and especially time. It was clear from the start that a complete model can never be achieved at this stage. But, all along the development of the Conveyor System model, serious considerations were anticipated to make the model manage the future changes and reconfigurations without any major changes.

As described in Chapter VI, the model developed will only accommodate a single loop conveyor system. Bypass sections may or may not exist depending upon the need of the particular system. Taking the model a step further to accommodate multiple loops and multiple floors will be illustrated below:

- Multi-loop Conveyor System: A conveyor which contains more than one closed loop in a one-floor configuration is termed as multi-loop conveyor system. Such a system includes transfer sections that deliver goods from one loop to another loop. A typical multi-loop conveyor system is shown in Figure 19. A transfer section is different from a bypass (description in chapter VI). A transfer section is used for moving carts from one conveyor loop to another whereas a bypass is used for providing shorter routes

between sections on the same loop. A transfer section does not form a closed loop and in this respect it is similar to a bypass. In this study it is assumed that a transfer section like a bypass accommodates only one cart at a time. In order to accommodate this extra feature, the modeler should add to all the classes developed, an instance variable that would store the loop number. One also needs to make the transfer section a subclass of WorkStation class (bypass is also a subclass of workstation). A cart needing to be transferred to another loop will have the transfer section position as its destination. Once the cart reaches the transfer section it leaves the hook and joins the transfer section's input queue. At that point the cart's instance variable specifying its loop number should be updated. Once the cart is delivered to the transfer section it will be held for an amount of time equal to the time to travel the transfer section. At the end of this time the cart should be in the next loop where it waits in the output queue of the transfer section for a hook to be available to take it to its next destination in the current loop. Since the different loops will have different hook numbers and speeds, the system designer should have the ability to specify the number of loops desired and their parameters. Therefore, depending on the loop number stored in the instance variable carried by the cart, the program logic will use the parameters (number of hooks, conveyor loop speed, etc.) specified for that conveyor loop. This is easily done due to the fact that the conveyor loop parameters are not hard coded in the program logic.

- Multi-floor Conveyor System: A multi-floor conveyor system contains more than one floor and goods are transported from one floor to another floor through an elevator. The elevator carries only one cart at a time. It operates on a first-come first-served basis. A typical multi-floor conveyor system is shown in Figure 20. Using the same concept as in the case of the transfer section, the modeler needs to add to all the classes an instance variable specifying the floor number. The modeler should also specify the elevator object as a subclass of the WorkStation object. A cart needing to go to another floor will

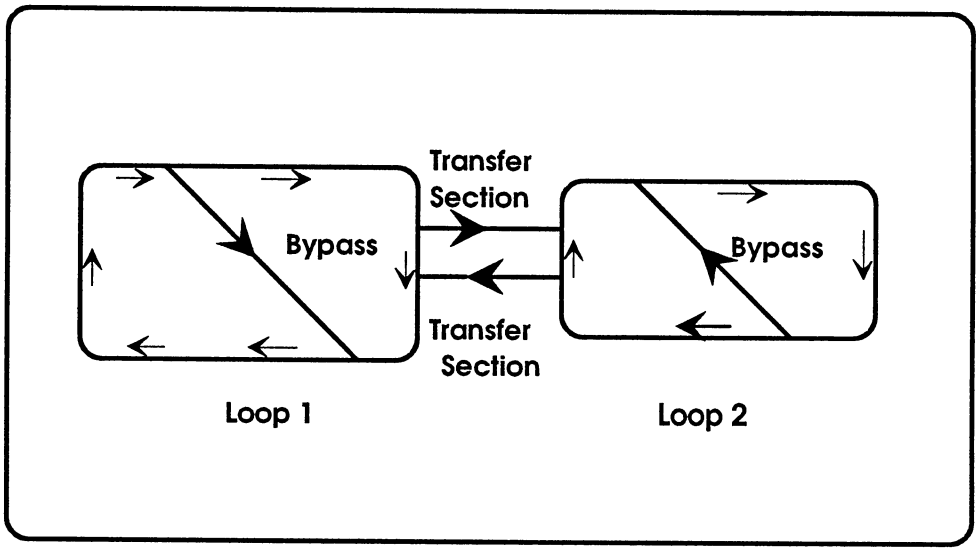

Figure 19. A Multi-loop Conveyor System

first be delivered by the hook at the elevator position. After, joining the elevator (in the same way a cart joins a workstation) the cart is held for an amount of time equal to the number of floors to be traveled times the travel time between two consecutive floors. At the end of this specified time the cart leaves the elevator and joins the output queue of the elevator station waiting for a hook available to carry it to its next destination. One of the system's input parameters will be the number of floors desired by the system designer.

- Currently, the transfer sections and bypasses accommodate only one cart at a time. Another option would be to make both of them subclasses of WorkStation with multiple-servers. The number of servers should be equal to the number of carts the user wants the transfer section and the bypass to accommodate at one time. This could be done by setting the instance variable wsAmount (workstation amount) to this desired number.

- A major area of improvement in the model developed is the user interface implementation. The user should be able to get desired statistics in a histogram form.

- Other options that could be added to the model for greater flexibility are unreliable workstations and hook failures.

- Currently, the destination of the carts is carried by the workstations and they are randomly decided by each workstation. Another option could be to make the routing of the carts through the system fixed and carried by the carts.

Contributions

The main contribution of this research to the body of simulation knowledge is the development of an object-oriented simulation model of a constant speed, discretely spaced, recirculating conveyor system. The evaluation of the quality of the model developed through the application of the Analytic Hierarchy Process provided more insights on the benefits and disbenefits of OOM over the traditional approaches to simulation.

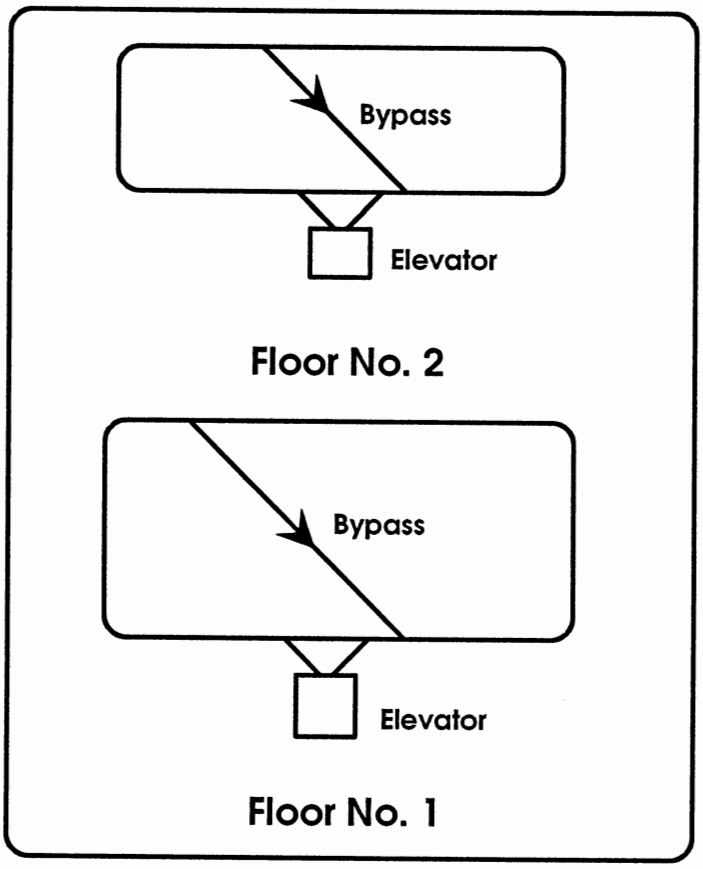

Figure 20. A Mulit-floor Conveyor System

BIBLIOGRAPHY

Abrams, M. "The Object Library for Parallel Simulation (OLPS)." Proceedings of The 1988 Winter Simulation Conference, San Diego, CA, 1988, pp. 210-219.

Adelsberger, H. H., U. W. Pooch, R. E. Ahannon, and G. N. Williams. "Rule Based Object Oriented Simulation Systems." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 107-112.

Adiga, S., "Software Modeling of Manufacturing Systems: A Case for an Object-Oriented Programming Approach." Working Paper, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA, 1986.

Adiga, S., "Software Modeling of Manufacturing Systems: A Case for an Object-Oriented Programming Approach." Annals of Operations Research, 1989, Vol. 17, pp. 363-378.

Adiga, S. and M. Gadre, "Object-Oriented Software Modeling of a Flexible Manufacturing System." Journal of Intelligent and Robotics Systems, ,1990, Vol. 3, pp. 147-165.

Adiga, S., Glassey, C. R. "Berkeley Library of Objects for Control and Simulation of Manufacturing (BLOCS/M)." Working Paper, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA, 1986.

Beaumariage, T. G. "Investigation of an Object Oriented Modeling Environment for the Generation of Simulation Models." Unpublished Ph.D. Dissertation, School of IE and Management, Oklahoma State University, Stillwater, OK, 1990.

Bezivin, J. "Timelock: A Concurrent Simulation Technique and its Description in Smalltalk-80." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 503-506.

Bhuskute, H., M. Duse, J. Gharpure, D. Pratt, M. Kamath, J.H. Mize. "Design and Implementation of a Highly Reusable Modeling and Simulation Framework for Discrete Part Manufacturing Systems." Center for Computer Integrated Manufacturing, Working Paper Series: CIM-WPS-92-HB1. Oklahoma State University, 1992.

Bordiga, A., D. Greenspan and J. Mylopous, "Knowledge Representation as the Basis for Requirements Specification." Computer, 1985, Vol. 4, pp. 82-91.

Cox, B. J., "Message/Object Programming: An Evolutionary Change in Programming Technology." IEEE Software, 1984, pp 50-61.

Cox, B. J., Object-Oriented Programming: An Evolutionary Approach, Addison-Wesley, Reading, MA, 1986.

Dahl, O. J. and K. Nygaar. "Simula-An Algol Based Simulation Language." Communications of the ACM, 1966, Vol. 9, pp. 123-145.

Ghaznavi-Collins, I. and D. Thelen. "An Object Oriented Approach toward System Architecture Simulation." AI Paper, 1988, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation, San Diego, CA, 1988, pp. 103-107.

Glicksman, J. "A Simulator Environment for an Autonomous Land Vehicle." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 53-57.

Goldberg, A. and D. Robson. Smalltalk-80: The Language, Addison-Wesley, Reading, MA, 1989.

Karacal, S. C. "The Development of an Integrative Structure for Discrete Event Simulation, Object Oriented Modeling and Embedded Decision Processes." Unpublished Ph.D. Dissertation School of Industrial Engineering and Management, Oklahoma State University, Stillwater OK, 1991.

King, C.U. and E.L. Fisher. "Object-Oriented Shop-Floor Design, Simulation, and Evaluation." Proceedings of the 1986 Fall Industrial Engineering Conference, Institute of Industrial Engineers, Norcross, GA, 1986, pp. 131-137.

Knapp, V. E. "The Smalltlak Simulation Environment, Part II." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 146-151.

Kreutzer, W., System Simulation: Programming Styles and Languages, Addison-Wesley, Reading, MA, 1986.

Meyer, B. Object-Oriented Software Construction, Prentice Hall International (UK) Ltd., Hertforshire, Great Britain, 1988.

Mitrani, I. Simulation Techniques for Discrete Event Systems, Cambridge University Press, Cambridge, 1982.

Mize, J.H., T.G. Beaumariage, S.C. Karacal. "Systems Modeling Using Object-Oriented Programming." Proceedings of the 1989 Spring Conference Institute of Industrial Engineers, Norcross, GA, 1989, pp. 13-18.

Nyen, P. A. "A Comprehensive Environment to Object-Oriented Simulation of Manufacturing Systems." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation, San Diego, CA, 1987, pp. 21-25.

Pazirandeh, M. and J. Becker. "Objet-Oriented Performance Models with Knowledge-Based Diagnostics." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 518-524.

Pratt, B. D. "Development of a Methodology for Hybrid Metamodeling of Hierarchical Manufacturing Systems Within a Simulation Framework." Unpublished Ph.D. Dissertation School of Industrial Engineering and Management, Oklahoma State University, Stillwater OK, 1992.

Pritsker, A. A. B. Introduction to Simulation and SLAM II. Third edition. John Wiley & Sons, New York, 1986.

Retting, M. "Using Smalltalk to Implement Frames." AI Expert, March 1987, pp. 15-18.

Saaty, T. L. Decision Making: The Analytic Hierarchy Process. RWS Publications, Pitsburg, PA, 1988.

Sanderson D.P., R. Sharma, R. Rozin, and S. Treu. "The Hierarchical Simulation Language HSL: A Versatile Tool for Process-Oriented Simulation." ACM Transactions on Modeling and Computer Simulation, 1991, pp. 113-153.

Shaw, M. "Abstraction Techniques in Modern Programming Languages. " IEEE Software, October 1984, pp. 10-26.

Terrell, M. P., Principle Investigator. NSF Final Report Grant GK-43583, "Modular System Analysis and Design for Utility Simulation of Constant Speed, Discretely Spaced, Recirculating Conveyor System.", May 30, 1977.

Turner, W. C., J. H. Mize, and K. E, Case. Introduction to Industrial and Systems Engineering, New Jersey: Prentice-Hall, Inc., 1986.

Thomasma, T. and O. M. Ulgen. "Modeling of a Manufacturing Cell Using a Graphical Simulation System Based on Smalltalk-80." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 683-691.

Thomasma, T. and O. M. Ulgen. "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing." Proceedings of the 1988 Winter Simulation Conference, IEEE, Piscataway, NJ, 1988, pp. 254-262.

Ulgen, O. M. and T. Thomasma. "A Graphical Simulation System in Smalltalk-80." Simulation in CIM and Artificial Intelligence Simulation Environments, The Society for Computer Simulation: San Diego, CA, 1987, pp. 53-58.

Ulgen, O. M. and T. Thomasma, Y. Mao. "Object-Oriented Toolkits for Simulation Program Generators." Proceedings of the 1989 Winter Simulation Conference, IEEE Piscataway, NJ, 1989, pp. 593-600.

Van der Meulen, P. "Development of an Interactive Simulator in Smalltalk." Journal of Object Oriented Programming, January/February 1989, Vol. 1, No 5, pp.28-51.

Wilson, R. "Object-Oriented Languages Reorient Programming Techniques." Computer Design, November 1, 1987, pp. 52-62.

APPENDICES

APPENDIX A

SMALLTALK-80 CLASSES DEFINED

Introduction

This appendix contains listings of Smalltalk-80 code beginning on the next page. The listings are relevant portions of new and/or modified classes and methods that were used for the conveyor system.

The listings are separated by class. Within each class, there is a header section followed by listings of methods. The header section contains the class hierarchy specification as well as the names of all instance and class variables. A comment segment concludes the header section.

The methods are divided into groups of related methods. This grouping is arbitrary but provides insight as to the general intend of the methods in the group. The group headers are designated by the character string "!*classname* methodsFor: *groupname*". The last grouping under a method (if listed) is the group for class methods. These methods are used by the class rather than instances of the class. A good example of their use is the creation of a new instance.

Methods listings always start with the method name including any incoming parameters. The names are free form except that a colon is used to separate the parameter(s) from the name. The code itself follows Smalltalk-80 convention. Any text within the method enclosed by quotation marks is a comment. All methods terminate with an exclamation point (!).

SmallTalk-80 Code For Class: QueueController

```
Object subclass: #QueueController
        instanceVariableNames: 'inputQueue outputQueue emptyQueue loadQueue '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation'!
QueueController comment:
'Class QueueController creates a controller through which a machine
(resource) communicates with its input and output queues.  The queue
controller is created automatically when the machine is created.'!

!QueueController methodsFor: 'initialize - release'!

emptyQueueCapacity: aNumber
        "The input queue has limited capacity"

        emptyQueue := CapacitatedQueue capacity: aNumber.

        "output queues are adjacent to the workstation, and do not need reservation. a cart directly
        moves into them, without worrying about other competition"
        emptyQueue addWithoutReservation: aCart!

putInOutput: aCart
        "output queues are adjacent to the workstation, and do not need reservation.
        Cart' s directly move into them, without worrying about other competition"
        outputQueue addWithoutReservation: aCart!

remove: aCart
        "Remove the cart from the output queue"
        ^outputQueue remove: aCart! !

!QueueController methodsFor: 'accessing'!

addToEmptyQueue: aCart
        emptyQueue add: aCart.
        ^self!

addToInputQueue: aCart
        "add a cart to the input queue"
        inputQueue add: aCart.
        ^self!

emptyQueueCapacity
        ^emptyQueue capacity!

emptyQueueEmpty
        ^emptyQueue isEmpty.!

emptyQueuequeueLength
        "Answer the length of the queue"
        ^emptyQueue  queueLength!
```

```
emptyQueueRemoveNext: aCart
        ^emptyQueue remove: aCart!

inputQueueCapacity
        ^inputQueue capacity!

inputQueueEmpty
        ^inputQueue isEmpty.!

inputQueueNextAmount
        "Answer the amount needed for the next item in queue"
        ^inputQueue next amountNeeded!

inputQueuequeueLength
        "Answer the length of the queue"
        ^inputQueue  queueLength!

inputQueueRemoveNext
        |qdisc |
        self error: 'Obsolete code'.
        qdisc := inputQueue whatIsQueueDiscipline.
        ^inputQueue remove: (self perform: qdisc)!
next
        "Answer the next item to be processed from the input  queue"
        ^inputQueue next!

next1
        "Answer the next item to be processed from the input  queue"
        ^outputQueue next!

nextEmpty
        "Answer the next item to be processed from the input  queue"
        ^emptyQueue next!

outputQueueCapacity
        ^outputQueue capacity!

outputQueueEmpty2
        ^outputQueue isEmpty.!

setEmptyQDiscipline: aQDiscipline
        "set the discipline of the empty queue for this controller"
        emptyQueue setQDiscipline: aQDiscipline.
        ^self!

setInputQDiscipline: aQDiscipline
        "set the discipline of the input queue for this controller"
        inputQueue setQDiscipline: aQDiscipline.
        ^self!

setOutputQDiscipline: aQDiscipline
        "set the discipline of the output queue for this controller"
        outputQueue setQDiscipline: aQDiscipline.
        ^self! !
```

```
!QueueController methodsFor: 'testing'!

emptyQueueHasSpace
        "check if the empty queue is at its capacity "
        ^emptyQueue hasSpace!

inputHasSpace
        "Does the input queue have space?"
        ^inputQueue hasSpace!

inputHasSpace: availableServers
        "Does the input queue have space, considering so may servers are available?"
        ^inputQueue hasSpace: availableServers!

outputHasSpace
        "Does the output queue have space?"
        ^outputQueue hasSpace! !

!QueueController methodsFor: 'statistics'!

printResultsOn: aStream
        "print the length of queue and time in queue statistics for this queue controller"
        aStream nextPutAll: 'Input Queue:'.
        inputQueue printResultsOn: aStream.
        aStream nextPutAll: 'Output Queue:'.
        outputQueue printResultsOn: aStream.
        aStream nextPutAll: 'empty Queue:'.
        emptyQueue printResultsOn: aStream!
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
QueueController class
        instanceVariableNames: "!
!QueueController class methodsFor: 'instance creation'!

new
        "create a new instance of QueueController"
        ^ super new initialize! !
```

SmallTalk-80 Code For Class: WorkStation

```
Object subclass: #WorkStation
        instanceVariableNames: 'wsAmountAvailable wsQueueController wsProcessingTimes
wsUtilization waitingForInputQ blockingWFI blocked cart enterPosition wsType exitPosition name
probabilities processTime loadCount '
        classVariableNames: 'Destinations LoadCount '
        poolDictionaries: "
        category: 'ConveyorSimulation'!

WorkStation comment:
'Class WorkStation is the class which represents a delayer to an object
being processed.  This class and its subclasses are used to represent
machine resources in the system.'!

!WorkStation methodsFor: 'accessing'!
```

```
amountAvailable
        "return the workstation amount available"
        ^wsAmountAvailable!

destinationPosition
        "get the position for the next destination of the cart"
        | adestination |
        adestination := WorkStation returnInstanceWithName: WorkStation getNextDestination.
        ^adestination enterPosition!

destinationPosition: aWorkStation
        " get the position for the next destination of the cart from the workstation"
        | adestination  |
        adestination := WorkStation returnInstanceWithName: (WorkStation getNextDestination:
                        aWorkStation).
        ^ (adestination enterPosition)!

destinations
        "Answer the destinations stored in the class variables"
        | adestinationList |
        adestinationList := WorkStation DestinationList.
        ^adestinationList!

emptyQueueCapacity
        "Answer the length of the queue"
        ^wsQueueController emptyQueueCapacity!

enterPosition
        "answer the enter position of the workstation"
        ^enterPosition!

enterPosition: aEnterPosition
        "set the workstation enter position to aEnterPosition"
        enterPosition := aEnterPosition!

exitPosition
        "answer the exit position of the workstation"
        ^exitPosition!

exitPosition: aExitPosition
        "set the exit position of the workstation to aExitPosition"
        exitPosition := aExitPosition!

getaRandomNumber
        "return a random number from the Random number generator class"
        | aRandom |
        aRandom := Random new.
        ^aRandom next!

getNextDestination: aWorkStation
        "get the next destination of the cart from the workstation depending on certain probabilities of
        routing"
        | r s aConveyor wsNumber |
```

```
        aConveyor := ConveyorSimulation active conveyor.
        wsNumber := aConveyor numWorkStations.
        r := aWorkStation getaRandomNumber.
        s := 0.
        1 to: wsNumber do:
                [:i |
                s := (aWorkStation probabilitiesVector at: i)
                                        + s.
                r <= s ifTrue: [^WorkStation DestinationList at: i]]!

inputQueueCapacity
        "Answer the length of the queue"
        ^wsQueueController  inputQueueCapacity!

loadCount
        "Answer the loadCount that is the number of orders to be loaded at a workstation"
        ^loadCount!

name
"Answer the name"
        ^name!

name: aString
        " set workstation name"
        name := aString.!
next2
        "Answer the next cart in the output queue"
        ^wsQueueController next1.!

outputQueueCapacity
        "Answer the length of the queue"
        ^wsQueueController outputQueueCapacity!

probabilities
        "return the probability vector for the destination routing"
        ^probabilities!

probabilitiesVector
        ^probabilities!

processTime
        "Answer the process time"
        ^processTime!

queueLength
        "Answer the length of the queue"
        ^wsQueueController  queueLength!

setName: aString
        " set workstation name"
        name := aString.!

setQDisciplineTo: aQDiscipline
        "set the queues discipline of the workstation"
```

```
        wsQueueController setBothQDisciplines: aQDiscipline!

type
        "Answer the type of the workstation"
        ^wsType!

type: aType
        "set the type of the workstation"
        wsType := aType! !

!WorkStation methodsFor: 'statistics'!

printResultsHookOn: aStream with: aNumber
        "print statistics for the Hook1 instances created"
        aStream cr; cr; nextPutAll: 'Statistics for Hook: ' , name , ' #' , aNumber printString; cr;
nextPutAll: '              No.Obs. Average  Std.Dev  Minimum  Maximum  Current'; cr.
        70 timesRepeat: [aStream nextPut: $-].
        aStream cr.
        aStream nextPutAll: 'Utilization: '.
        wsUtilization printResultsOn: aStream.
        aStream cr; nextPutAll: 'Proc.Time: '.
        wsProcessingTimes printResultsOn: aStream.
        aStream cr.
        70 timesRepeat: ["wsQueueController printResultsOn: aStream."
                aStream nextPut: $-].
        aStream cr!

printResultsOn: aStream
        "print statistics for the workstation"
        aStream cr; cr; nextPutAll: 'Statistics for WorkStation: ' , name; cr; nextPutAll: '
No.Obs. Average  Std.Dev  Minimum  Maximum  Current'; cr.
        70 timesRepeat: [aStream nextPut: $-].
        aStream cr.
        aStream nextPutAll: 'Utilization: '.
        wsUtilization printResultsOn: aStream.
        aStream cr; nextPutAll: 'Proc.Time: '.
        wsProcessingTimes printResultsOn: aStream.
        aStream cr.
        wsQueueController printResultsOn: aStream.
        70 timesRepeat: [aStream nextPut: $-].
        aStream cr!

processTime1: aTime
        "set the process time of the workstations"
        wsProcessingTimes equals: aTime!

!WorkStation methodsFor: 'task language'!

generateAndLoadOrders
        "This method is used to schedule the generation of the subsequent load
        orders and try to load the current ones if possible"
        | aConveyor aWS |
        aConveyor := ConveyorSimulation active.
        1 to: aConveyor numWorkStations do:
```

```
                [:i |
                aWS := self returnInstanceWithName: (WorkStation DestinationList at: i).
                aWS type = 1
                        ifTrue:
                                [self tryToLoad.
                                aWS loadCount: loadCount + 1].
                aWS type = 2
                        ifTrue:
                                [self tryToLoad.
                                aWS loadCount: loadCount + 1].
                aWS type = 3
                        ifTrue:
                                [self tryToLoad.
                                aWS loadCount: loadCount + 1].].
        ConveyorSimulation active schedule: [self generateAndLoadOrders]
                after: (Normal mean: 20 deviation: 4)!

produce: anAmount
        "anAmount of the workstation resource is available"
        wsAmountAvailable := wsAmountAvailable + anAmount.
        self provideServices!

provideServices
        "provide workstation resources to the next job in queue"
        | waitingWFI |
        [wsQueueController inputQueueEmpty not
                and: [cart := wsQueueController next.
                        cart amountNeeded <= self amountAvailable]]
                whileTrue:
                        [waitingWFI := wsQueueController inputQueueRemove: cart.
                        wsAmountAvailable := wsAmountAvailable - waitingWFI amountNeeded.
                        wsUtilization equals: wsUtilization value + waitingWFI amountNeeded.
                        waitingWFI resume]!

provideServiceTo: aCart
        "This wfi needs to be serviced. Put into the queue, and provide a server if possible"
        wsQueueController addToInputQueue: aCart.        "SimScript cr; nextPutAll:        aCart
printString, ' needs service at: ',
        self    printString."
        self provideServices.
        aCart pause!

release: anAmount
        "Some cart has released anAmount of the workstation resource"
        wsUtilization equals: wsUtilization value - anAmount.
        self produce: anAmount! !

!WorkStation methodsFor: 'initialize-release'!

initializeWithName: aString andAmount: aNumber exitPosition: aExitPosition enterPosition:
aEnterPosition probabilities: aVector processTime: aTime
        "Initialize the instance vars"
        name := aString.
        exitPosition := aExitPosition.
```

```
        enterPosition := aEnterPosition.
        wsAmountAvailable := aNumber.
        wsQueueController := QueueController new.
        wsProcessingTimes := ObsTrackedNumber new.
        processTime := aTime.
        wsUtilization := TimeTrackedNumber new.
        waitingForInputQ := OrderedCollection new.
        probabilities := aVector.
        blocked := false.
        loadCount := 0!

initializeWithName: aString andAmount: aNumber exitPosition: aExitPosition enterPosition:
aEnterPosition probabilities: aVector processTime: aTime type: aType
        "Initialize the instance vars"
        name := aString.
        exitPosition := aExitPosition.
        enterPosition := aEnterPosition.
        wsAmountAvailable := aNumber.
        wsQueueController := QueueController new.
        wsProcessingTimes := ObsTrackedNumber new.
        processTime := aTime.
        wsUtilization := TimeTrackedNumber new.
        waitingForInputQ := OrderedCollection new.
        probabilities := aVector.
        blocked := false.
        loadCount := 0.
        wsType := aType!

loadCount: aCount
        "set the load count to aCount"
        loadCount := aCount!

processTime: aTime
        "set the process time to aTime"
        processTime := aTime.
        ^self! !

!WorkStation methodsFor: 'printing'!

printOn: aStream
        "Express the work station in printable characters"
        aStream nextPutAll: 'WorkStation named: ' , name! !

!WorkStation methodsFor: 'queue capacity'!

emptyQueueAddCart: aCart
        "add a cart to the empty queue"
        wsQueueController putInEmptyQueue: aCart.!

emptyQueueCapacity: aNumber
        "Change the empty queue to a definite capacity"
        wsQueueController emptyQueueCapacity: aNumber!
        inputQueueAddCart: aCart
        "add a cart to the empty queue"
```

```
        wsQueueController addToInputQueue: aCart!

inputQueueCapacity: aNumber
"Change the input queue to a definite capacity"
wsQueueController inputQueueCapacity: aNumber!

loadQueueCapacity: aNumber
        "Change the load queue to a definite capacity"
        wsQueueController loadQueueCapacity: aNumber!

outputQueueCapacity: aNumber
        "Change the output queue to a definite capacity"
        wsQueueController outputQueueCapacity: aNumber!

outputQueueEmpty
        "Check if the output queue is empty or not"
        ^wsQueueController outputQueueEmpty2!

putMeInOutputQueue: aCart
        "output queues are adjacent to the workstation,
        and do not need reservation. Cart's directly move into them, without
        worrying about other competition, if there is a space. If there is no space, they are blocked"
        | true5 |
        true5 := self hasOutputSpace.
        true5 = True
                ifTrue: [wsQueueController putInOutput: aCart]
                ifFalse:
                        ["There is no place for this wfi. the workStation is blocked"
                        SimScript cr; nextPutAll: aCart printString , 'blocked the ' , self printString.
                        blocked := true.
                        blockingWFI := aCart.
                        aCart pause.
                        blocked := false.
                        wsQueueController putInOutput: aCart]!

remove2: aCart
        "The cart needs to be removed from the output queue of the workstation"
        blocked ifTrue: [blockingWFI resume].
        ^wsQueueController remove: aCart!

removeEmptyCart
        "The cart needs to be removed from the output queue of the workstation"
        ^wsQueueController emptyQueueRemoveNext! !

!WorkStation methodsFor: 'simulation control'!

holdFor: aTimeDelay
        "Schedule a delay of this period"
        Simulation active delayFor: aTimeDelay! !

!WorkStation methodsFor: 'testing'!

hasEmptyQueueSpace
        "Is there place in the output queue?"
```

```
        ^wsQueueController emptyQueueHasSpace!

hasInputSpace
        "There is space in the workStation if the input queue has space or if there is a server available"
        ^wsQueueController inputHasSpace!

hasOutputSpace
        "Is there place in the output queue?"
        ^wsQueueController outputHasSpace!

isBlocked
        "Answer if the workStation is blocked"
        ^blocked!

!WorkStation methodsFor: 'decisions'!

tryToLoad
        "Allow service if there is a loading order waiting for service"
        | aCart |
        wsQueueController emptyQueueEmpty not
                ifTrue:
                        [cart := wsQueueController nextEmpty.
                        aCart := wsQueueController emptyQueueRemoveNext: cart.
                        SimScript cr; nextPutAll: self printString , ' satsified a laodOrder  ' , self
printString. aCart resume]! !
!WorkStation methodsFor: 'comparing'!

<= another
        "work stations compare by name"
        ^name <= another name! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

WorkStation class
        instanceVariableNames: ''!

!WorkStation class methodsFor: 'instance creation'!

DestinationList
        "return the destination list"
        ^Destinations!
newWithName: aString andAmount: aNumber exitPosition: aExitPosition enterPosition: aEnterPosition
probabilities: aVector processTime: aTime type: aType
        "initialize instance vars"
        ^(self new)
                initializeWithName: aString
                andAmount: aNumber
                exitPosition: aExitPosition
                enterPosition: aEnterPosition
                probabilities: aVector
                processTime: aTime
                type: aType; yourself!

probabilityVector
"^probabilities"!
```

```
setDestinations: aDestDectionary
        "set the destination list to the list of workstations input by the user through
        the user interface"
        Destinations := aDestDectionary! !

!WorkStation class methodsFor: 'schedulingOrders'!

generateAndLoadOrders
        "This method is used to schedule the generation of the subsequent load
        orders and try to load the current ones if possible"
        | aConveyor aWS |
        aConveyor := ConveyorSimulation active conveyor.
        1 to: aConveyor numWorkStations do:
                [:i |
                aWS := self returnInstanceWithName: (WorkStation DestinationList at: i).
                aWS type = 1
                        ifTrue:
                                [aWS tryToLoad.
                                aWS loadCount: aWS loadCount + 1].
                aWS type = 2
                        ifTrue:
                                [aWS tryToLoad.
                                aWS loadCount: aWS loadCount + 1].
                aWS type = 3
                        ifTrue:
                                [aWS tryToLoad.
                                aWS loadCount: aWS loadCount + 1].].
        SimScript cr; nextPutAll: self printString , ' generated ordersAt:      ' , self printString.
        ConveyorSimulation active schedule: [self generateAndLoadOrders]
                after: (Normal mean: 10 deviation: 2) next! !

!WorkStation class methodsFor: 'getAnInstance'!

returnInstanceWithName: aName
        "return the WorkStation instance with the name aName"
        WorkStation allInstances.
        ^WorkStation allInstances detect: [:each | each name = aName]! !
```

SmallTalk-80 Code For Class: Hook1

```
WorkStation subclass: #Hook1
        instanceVariableNames: 'hookPosition hookNumber cartHeld hookStatus hookUtilization '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation'!

!Hook1 methodsFor: 'accessing'!

cartHeld
        "Return the cart held in the hook"
        ^cartHeld!

hookNumber
```

```
        "Return the hook number of the hook instance"
        ^hookNumber!

hookPosition
        "Return the hook position of the hook instance"
        ^hookPosition!

hookStatus
        "Return the hook status of the hook"
        ^hookStatus! !

!Hook1 methodsFor: 'initialize-release'!

cartDispose
        "After delivering the cart to its destination the storage location of the cart
        held is changed to empty and the hookStatus is changed to idle"
        cartHeld := nil.
        hookStatus := 0!

cartHeld: aCart
        "Store the cart to be transported in the storage location cartHeld."
        cartHeld := aCart!

hookNumber: aNumber
        "Set the hook number to aNumber"
        hookNumber := aNumber!

hookPosition: aPosition
        "set the hook position to aPosition"
        hookPosition := aPosition!

hookProcess: aTime
        "Set the process time of the hook to aHook"
        wsProcessingTimes equals: aTime!

hookStatus: aStatus
        "Set the hook status of the hook to aStatus"
        hookStatus := aStatus!

initializeWithName: aString andAmount: aNumber hookNumber: aNumber1 hookPosition: aNumber2
hookStatus: aStatus
        "Initialize the instance vars"
        name := aString.
        wsAmountAvailable := aNumber.
        wsQueueController := QueueController new.
        wsProcessingTimes := ObsTrackedNumber new.
        wsUtilization := TimeTrackedNumber new.
        hookNumber := aNumber1.
        hookPosition := aNumber2.
        hookStatus := aStatus.
        wsQueueController inputQueueCapacity: 5; outputQueueCapacity: 5!

uitlization1: aNumber
        "calculating the workstation utilization"
```

```
        wsUtilization equals: wsUtilization value + aNumber!

uitlization: aNumber
        "Calculating the workstation utilization"
        wsUtilization equals: wsUtilization value - 1! !

!Hook1 methodsFor: 'task language'!

provideServices
        "provide a Hook1 instance to the next job in queue"
        | waiting cart |
        [wsQueueController inputQueueEmpty not
                and: [cart := wsQueueController next.
                        cart amountNeeded <= wsAmountAvailable]]
                whileTrue:
                        [waiting := wsQueueController inputQueueRemove: cart.
                        wsAmountAvailable := wsAmountAvailable - waiting amountNeeded.
                        waiting resume]!

provideServiceTo: aCart
        "This cart needs to be serviced. Put into the queue, and provide a hook if possible"
        wsQueueController addToInputQueue: aCart.
        self provideServices.
        aCart pause.
        wsUtilization equals: wsUtilization value + aCart amountNeeded!
release: anAmount at: aLocation
        "release anAmount of the hook at aLocation"
        wsUtilization equals: wsUtilization value - anAmount.
        "SimScript cr; nextPutAll: name, ' is at: ', aLocation name."
        self hookStatus: 0.
        self produce: anAmount! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Hook1 class
        instanceVariableNames: ''!

!Hook1 class methodsFor: 'instance creation'!

newWithName: aString andAmount: aNumber hookNumber: aNumber1 hookPosition: aNumber2
hookStatus: aStatus
        "Create a new hook at this location"
        ^self new
                initializeWithName: aString
                andAmount: aNumber
                hookNumber: aNumber1
                hookPosition: aNumber2
                hookStatus: aStatus! !
```

SmallTalk-80 Code For Class: CartGenerator

```
Object subclass: #CartGenerator
        instanceVariableNames: 'name firstTime distribution lastTime totalCount initialCount county '
        classVariableNames: ''
```

```
        poolDictionaries: ''
        category: 'ConveyorSimulation'!
CartGenerator comment:
'Class WorkFlowGenerator creates supply-driven workflowitems into the
system.  This works like a create node in SLAM.'!

!CartGenerator methodsFor: 'accessing'!

name
        "Answer the name of the job that it generates"
        ^name!

totalCount
        "Answer the totalCount of the job that it generates"
        ^totalCount! !

!CartGenerator methodsFor: 'initialize-release'!

firstTime: time1 lastTime: time2
        "Initialize the first time of creation, last time of creation, initial number
        of carts created, and the maximum number of carts to be created"
        firstTime := time1.
        lastTime := time2.
        ^self!

firstTime: time1 lastTime: time2 initialCount: count1 totalCount: count2
        "Initialize the first time of creation, last time of creation, initial number
        of carts created, and the maximum number of the Carts to be created"
        firstTime := time1.
        lastTime := time2.
        initialCount := count1.
        totalCount := count2.
        ^self!

initialize
        "Default values of the first time, last time of creation, and the initial count and final count of the
Carts to be created. The system is initialized to have 20 Carts"
        firstTime := 0.0.
        lastTime := 1000000000.
        totalCount := 19.
        initialCount := 0.
        name := 'Cart'.
        ^self!

name: aName distribution: aDistribution
        "Set the name and the distribution according to which the Cart instances will be created with"
        name := aName.
        distribution := aDistribution.
        ^self!

totalCount: count
        "Initialize the initial number of carts created, and the maximum number of the Carts to be
created"
        | aCount |
```

```
        aCount := count asNumber - 1.
        initialCount := 0.
        totalCount := aCount .
        ^self! !

!CartGenerator methodsFor: 'scheduling'!

scheduleArrival
        "This is the message to schedule the creation of entities by this Cart generator."
        | sim aBlock |
        aBlock := [(Cart name: name) startUp].
        sim := Simulation active.
        sim
                newProcessFor:
                        [| count |
                        1 to: initialCount do: [sim newProcessFor: aBlock copy].
                        sim delayUntil: firstTime.
                        sim newProcessFor: aBlock copy.
                        count := 0.
                        [sim time < lastTime & count < self totalCount]
                                whileTrue:
                                        [count := count + 1.
                                        sim delayFor: distribution next.
                                        sim newProcessFor: aBlock copy]]! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
CartGenerator class
        instanceVariableNames: ''!

!CartGenerator class methodsFor: 'instance creation'!

name: aName distribution: aDistribution
        "Create an instance of CartGenerator"
        | anObject |
        anObject := self new.
        anObject name: aName distribution: aDistribution.
        ^anObject!

new
^super new initialize! !
```

SmallTalk-80 Code For Class: Cart

```
SimulationObject subclass: #Cart
        instanceVariableNames: 'name entryTime currentPosition workStation queueEntryTime
currentWorkStation destination bypass tempDestination destinationPosition status serial done county
cartUtilization '
        classVariableNames: 'CartUtilization Count EntryTime MaterialHandling TimeInSystem '
        poolDictionaries: ''
        category: 'ConveyorSimulation'!

!Cart methodsFor: 'initialize-release'!

cartUtilization: aNumber
```

```
self destination name = bypass name
ifFalse: [ self status = 1
ifTrue: [cartUtilization equals: cartUtilization value - 1]
ifFalse: [cartUtilization equals: cartUtilization value + 1 ]].!

initialize
"The workflowitem starts from the buffer 'storage'"
currentPosition := 1.
currentWorkStation := 'WS1'.
status := 0.
done := false.
county := 0.
bypass := WorkStation returnInstanceWithName: 'BYPASS'.
super initialize!

initName: aName
"The workflowitem is given a name"
self  initialize.
self name: aName.
  cartUtilization := TimeTrackedNumber new.
(Count includesKey: aName) ifTrue: [serial := Count at: aName put: ((Count at: aName) + 1)]
ifFalse: [serial := Count at: aName put: 1].
SimScript cr; nextPutAll: self printString, ' created'! !

!Cart methodsFor: 'accessing'!

amountNeeded
"return the amount needed by a cart"
^amountNeeded!

bypass
        ^bypass!

county
        ^county!

currentProcessTime
        "Answer the expected value of the current process time"
        ^currentWorkStation processTime mean!

currentWorkStation
        "Answer the name of the current workstation  of the Cart   "
        ^currentWorkStation!

currentWorkStation: aName
        "set the name of the current workstation  of the Cart   "
        currentWorkStation := aName!

destination
        "Answer the  destination of the Cart"
        ^destination!

destination: aWorkStation
        "Set  the  destination of the Cart"
```

```
        destination := aWorkStation!

done
        ^done!

name: aName
        name := aName.
        ^self!

position
        "Answer the current position of the Cart   "
        ^currentPosition!

position: aPosition
        "Answer the current position of the Cart   "
        currentPosition := aPosition.!

queueEntryTime
        ^queueEntryTime!

resourceNeeded
        ^resourceNeeded!

resourceNeeded: aResource
        resourceNeeded := aResource!
serial
        ^serial!

status
        ^status!

status: aStatus
        status := aStatus.
        ^self!

tempDestination
        ^tempDestination!

tempDestination: aDestination
        tempDestination := aDestination!

timeStamp
        " put time stamp to the entry time to the queue"
        queueEntryTime := Simulation active time! !

!Cart methodsFor: 'simulation control'!

holdFor: aTimeDelay
        Simulation active delayCart: self for: aTimeDelay!

pause
        "Pause the process of this cart"
        Simulation active stopProcess.
        resumptionSemaphore wait!
```

```
resume
        "Resume the process of this work flow item"
        Simulation active startProcess.
        resumptionSemaphore signal! !

!Cart methodsFor: 'testing'!

checkShortestRouteFrom: aWorkStation
        "This method is to check if the shourtest route lies along a bypass"
        | aDestination aDestinationPosition bypassPosition exit1 exit2 |
        aDestination := self destination.
        self currentWorkStation  = bypass name
                ifFalse: [aDestinationPosition := aDestination enterPosition.
        bypassPosition := bypass enterPosition.
exit1 := aDestination exitPosition.
exit2 := bypass exitPosition.
        aDestinationPosition > bypassPosition  & (exit1< exit2)
                ifFalse: [ self position > aDestinationPosition & (self position  < bypassPosition)
                        ifTrue:[tempDestination := destination.
                        destination := bypass. SimScript cr; nextPutAll: self printString , ' changed
destination to bypass ' , destination printString.]
                        ifFalse: [self position < aDestinationPosition & (exit1 > exit2)
                                        ifTrue: [tempDestination := destination.
                        destination := bypass. SimScript cr; nextPutAll: self printString , ' changed
destination to bypass ' , destination printString.].]]].! !

!Cart methodsFor: 'task language'!

acquireResource: aResource
        "the cart should try to acquire a resource "
        resourceNeeded := aResource.
        amountNeeded := 1.
        aResource provideServiceTo: self!

completeOperationsAtLocation1
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location.  This is done at the UALSC"
        | aWorkStation destname aStatus true2 true3 |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        aStatus := self status.
        aStatus = 0
                ifTrue: [aWorkStation loadCount > 0
                                ifTrue:
                                        [self getProcessedAtLocation.
                                aWorkStation loadCount: aWorkStation loadCount - 1. self status: 1. ]
                                ifFalse:
                                        [true2 := aWorkStation hasEmptyQueueSpace.
                                                true2 = True
                                                ifTrue:
```

```
                                                    [aWorkStation emptyQueueAddCart: self.
self pause.  self getProcessedAtLocation. self status: 1.  aWorkStation loadCount: aWorkStation
loadCount - 1.
        SimScript cr; nextPutAll: self printString , ' joined the emptyQueue ' , currentPosition
printString.]]]
                ifFalse:
                        [self getProcessedAtLocation.
                        self status: 0.
                        aWorkStation loadCount > 0
                                ifTrue:
                                        [self getProcessedAtLocation.
                                        aWorkStation loadCount: aWorkStation loadCount - 1.
                                        self status: 1. ]
                                ifFalse: [true3 := aWorkStation hasEmptyQueueSpace.
                                                        true3 = True
                                                ifTrue:
                                                        [aWorkStation emptyQueueAddCart: self.
SimScript cr; nextPutAll: self printString , ' joined the emptyQueue ' , currentPosition printString.
                                                        self pause.
                                                        "self acquireResource: aWorkStation."
                                                        self getProcessedAtLocation.
                                                        self status: 1.  aWorkStation loadCount:
aWorkStation loadCount - 1.  SimScript cr; nextPutAll: self printString , ' satisfied an order ' ,
currentPosition printString.]]].
        county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

completeOperationsAtLocation2
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location. This is done at the LSWAT"
        | aWorkStation destname aStatus true3 |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        aStatus := self status.
        aStatus = 0 ifTrue:  [   " [aWorkStation loadCount > 0
                        ifTrue:
                                [self getProcessedAtLocation.
                                aWorkStation loadCount: aWorkStation loadCount - 1.
                                self status: 1. ]
                        ifFalse: ["    true3 := aWorkStation hasEmptyQueueSpace.
                                                        true3 = True
                                        ifTrue:
                                                [aWorkStation emptyQueueAddCart: self.
                                                self pause.

                                                self getProcessedAtLocation.
                                                aWorkStation loadCount: aWorkStation loadCount -
1. self status: 1.
```

```
        SimScript cr; nextPutAll: self printString , ' satisfied a load order ' , currentPosition printString]]
.       county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

completeOperationsAtLocation3
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location. This subroutine is for the LODST"
        | aWorkStation destname aStatus |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        aStatus := self status.
        aStatus = 0
                        ifTrue: [self getProcessedAtLocation.
                                "aWorkStation loadCount: aWorkStation loadCount - 1."
                                self status: 1. ].
        county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

completeOperationsAtLocation4
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location. This subroutine is for the OLUNLOD WS9"
        | aWorkStation destname aStatus |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        aStatus := self status.
        "aStatus = 1
                        ifTrue: ["self getProcessedAtLocation. self status: 0.
        county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

completeOperationsAtLocation5
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location. This subroutine is for the ULST"
        | aWorkStation destname aStatus |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
```

```
        aStatus := self status.
        aStatus = 1
                        ifTrue: [self getProcessedAtLocation.
                                aWorkStation loadCount: aWorkStation loadCount - 1.
                                self status: 0. ].
        county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

completeOperationsAtLocation6
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location. This subroutine is for the BYPASS"
        | aWorkStation  |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        self getProcessedAtLocation. county := county + 1.
        destination := self tempDestination.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!
completeOperationsAtLocation8
        "The wfi has arrived at a resource, and acquired it. Now complete all the
        operations that are assigned at this location.  This is done at the PPLS workstation"
        | aWorkStation  destname  |
        aWorkStation := destination.
        self currentWorkStation: destination name.
        self position: destination exitPosition.
        self getProcessedAtLocation.
        county := county + 1.
        destname := aWorkStation getNextDestination: aWorkStation.
        destination := WorkStation returnInstanceWithName: destname.
        self status: 1.
        aWorkStation putMeInOutputQueue: self.
        SimScript cr; nextPutAll: self printString , ' released ' , currentPosition printString.
        SimScript cr; nextPutAll: self printString , ' going to ' , destination printString.
        self releaseResource!

finishUp
super finishUp!

getProcessedAtLocation
        "The wfi has arrived at a resource, and acquired it. Now complete the current operation"
        | resource time  aType |
        resource := WorkStation returnInstanceWithName: self currentWorkStation.
        aType := resource type.
                aType = 4
                        ifTrue: [time := 0.]
                        ifFalse: [ time := resource processTime next. ].
```

```
        self cartUtilization: self status.
        SimScript cr; nextPutAll: self printString , ' processing time required at  ' , resource printString ,
' ' , time printString.
        SimScript cr; nextPutAll: self printString , ' obtained ' , resource printString.
 resource processTime1: time.
        self holdFor: time!

release: aHook
        "release the hook when the cart gets to its destination at aPosition"
        aHook hookStatus: 0.!

releaseResource
        "release the hook when the cart gets to its destination at aPosition"
        SimScript cr; nextPutAll: name , '  is at: ' , currentWorkStation.
        resourceNeeded release: amountNeeded.
        amountNeeded := 0!

startUp
        "The initial start up message for the work flow item"
        entryTime := Simulation active time.
        self position: 5.
        self currentWorkStation: 'PPLS'.
        county := 0.
        self tasks.
        self finishUp!

tasks
        | aWorkStation aType |
        [done]
                whileFalse:
                [aWorkStation := WorkStation returnInstanceWithName: self currentWorkStation.
                        county = 0
                                ifTrue:
                                        [destination := aWorkStation.
                                        destinationPosition := destination enterPosition]
                                ifFalse: [aWorkStation := destination].
                        aType := aWorkStation type.
                        aType = 1
                                ifTrue:
                                        [self acquireResource: aWorkStation.
                                        self completeOperationsAtLocation1].
                        aType = 2
                                ifTrue:
                                        [self acquireResource: aWorkStation.
                                        self completeOperationsAtLocation2].
                        aType = 3
                                ifTrue:
                                        [self acquireResource: aWorkStation.
                                        self completeOperationsAtLocation3].
                        aType = 4
                                ifTrue:
                                        [self acquireResource: aWorkStation.
                                        self completeOperationsAtLocation4].
                        aType = 5
```

```
                        ifTrue:
                                [self acquireResource: aWorkStation.
                                self completeOperationsAtLocation5].
                aType = 6
                        ifTrue:
                                [self acquireResource: aWorkStation.
                                self completeOperationsAtLocation6].
                aType = 7
                        ifTrue:
                                [self acquireResource: aWorkStation.
                                self completeOperationsAtLocation7].
                self pause]!

!Cart methodsFor: 'printing'!

printOn: aStream
"Printable form of this cart"
aStream nextPutAll: 'Cart ', name, ' serial# ', serial printString!
printResultsOn: aStream
aStream cr.
" nextPutAll: 'Utilization for : ' , name,' serial #',serial printString; cr;
        nextPutAll: '                No.Obs.  Average   Std.Dev   Minimum   Maximum   Current'; cr.
        70 timesRepeat: [aStream nextPut: $-]. aStream cr."
aStream
        nextPutAll: '' , name , ' serial # ',serial printString.
        cartUtilization printResultsOn: aStream.
         aStream cr.
        70 timesRepeat: [aStream nextPut: $-]. aStream cr.! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Cart class
        instanceVariableNames: ''!

!Cart class methodsFor: 'instance creation'!

name: aName
| anObject |
        anObject := super new initName: aName.
^anObject!

!Cart class methodsFor: 'initialize-release'!

initialize
CartUtilization := TimeTrackedNumber new.
Count := Dictionary new.! !

!Cart class methodsFor: 'getAnInstance'!

count
        ^Count!

returnInstanceWithSerial: aSerial
        Cart allInstances.
        ^Cart allInstances detect: [:each | each serial = aSerial]! !
```

SmallTalk-80 Code For Class: ConveyorSimulation

```
Simulation subclass: #ConveyorSimulation
        instanceVariableNames: 'cartGenerator outputStream conveyor '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation'!
ConveyorSimulation comment:
'The actual Simulation model is represented by this object. '!

!ConveyorSimulation methodsFor: 'initialize'!

addCartGenerator: aGenerator
        "Include a new CartGenerator in the set. CartGenerator
        merely send Cart Items into the simulation with time between
        arrivals given by a distribution"

        cartGenerator add: aGenerator!
addConveyor: aConveyor
        "Include a new Conveyor in the set."
        conveyor := aConveyor!

conveyor
        "Return the known conveyor in the set."
        ^conveyor!

initialize
        "Cart is initialized so that time in system statistics could be collected
        for individual job types. CartGenerator sends work through the
        system. outputStream is where the simulation results are written."
        Cart initialize.
        cartGenerator := OrderedCollection new.
        outputStream _ TextStream on: (String new: 512).
        super initialize! !

!ConveyorSimulation methodsFor: 'simulation control'!

defineArrivalSchedule
        "The CartGenerator needs to schedule the first and subsequent arrivals of its Cart Items"
        cartGenerator do: [:creator | creator scheduleArrival].
        super defineArrivalSchedule!

delayCart: cart for: timeDelay
        "This method is to delay the evaluation of the next block of action of the Cart object until the
        simulation time reaches the appropriate simulated time."
        self delayCart: cart until: currentTime + timeDelay!

delayCart: cart until: aTime
        "This method is to delay the evaluation of the next block of action of the Cart object until the
        simulation time reaches the appropriate simulated time"
        cart resumptionTime: aTime.
        eventQueue add: cart.
```

```
        cart pause!

finishUp
        "We need to empty out the event queue."
        super finishUp.
        self printResultsOn: outputStream!

startUp
        "start up of simulation run"
        | dateAndTime |
        dateAndTime := Time dateAndTimeNow.
        outputStream nextPutAll: 'Simulation initiated at ' , (dateAndTime at: 2)        printString , ',  ' ,
(dateAndTime at: 1) printString.
        super startUp!

time: aTime
        "This message is for debugging purposes only"
        currentTime := aTime! !

!ConveyorSimulation methodsFor: 'accessing'!

includesPart: partName
        "Answer if the partName is used in this simulation"
        cartGenerator do: [:cart | cart name = partName ifTrue: [^true]].
        ^false!

outputStream
        "Answer the current outputStream"
        ^outputStream!

outputStream: aStream
        "Designate aStream as the outputStream of the simulation"
        outputStream _ aStream! !

!ConveyorSimulation methodsFor: 'statistics'!

clearStatisticsAt: aTime
        "Method to clear all the collected statistics at a specified time"
        self
                schedule:
                        [ObsTrackedNumber allInstances do: [:aNumber | aNumber clearStatistics].
                        TimeTrackedNumber allInstances do: [:aNumber | aNumber clearStatistics].
                        TrackedNumberWithCollection allInstances do: [:aNumber | aNumber
clearStatistics].
                        TrackedNumberWithHistogram allInstances do: [:aNumber | aNumber
clearStatistics]] at: aTime!

printResultsOn: aStream
        "print the collected statistics for the workstations, hooks, and carts"
        | aConveyor workStation aCart hooks aGenerator |
        aConveyor := ConveyorSimulation active conveyor.
        workStation := Dictionary new.
        1 to: aConveyor numWorkStations do: [:i | workStation at: i put: (WorkStation
        returnInstanceWithName: (WorkStation DestinationList at: i))].
```

```
        1 to: aConveyor numWorkStations do: [:i | (workStation at: i) printResultsOn: aStream].
        hooks := Dictionary new.
        1 to: aConveyor trackSize do: [:i | hooks at: i put: (aConveyor track at: i)].
        1 to: aConveyor trackSize do: [:i | (hooks at: i)
                printResultsHookOn: aStream with: (hooks at: i) hookNumber].
        aStream cr; cr; nextPutAll: 'Utilization for Carts:'; cr; cr; nextPutAll: '                No.Obs.
        Average   Std.Dev   Minimum   Maximum   Current'; cr.
        70 timesRepeat: [aStream nextPut: $-].
        aStream cr.
aGenerator := cartGenerator at: 1.
        1 to: (( aGenerator totalCount) + 1) do:
                [:i |
                aCart := Cart returnInstanceWithSerial: i.
                aCart printResultsOn: aStream].
aStream cr; nextPutAll: 'Simulation Ended at '; nextPutAll: Date dateAndTimeNow printString; cr! !

!ConveyorSimulation methodsFor: 'tracing'!

traceOff
"Switch the trace on"

SimScript := DummyTextStream new!

traceOn
"Switch the trace on"
SimScript := TextStream on: (String new: 512)!

traceOnAt:  aTime
"Switch the trace on"
Simulation active  schedule: [
SimScript := TextStream on: (String new: 512)] at: aTime! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConveyorSimulation class
        instanceVariableNames: ''!

!ConveyorSimulation class methodsFor: 'instance creation'!

new

" Method to create a new instance of ConveyorSimulation "
^super new initialize! !

!ConveyorSimulation class methodsFor: 'examples'!

examplecg: aTime
        "This example does not use BOM or assembly station, and is not consistent any more with the
simulation codes"
        | aVector aWorkStation bVector bWorkStation gVector gWorkStation hVector hWorkStation
        iVector iWorkStation sim aConveyor aCart jVector jWorkStation dVector dWorkStation fVector
        fWorkStation |
        aCart := CartGenerator name: 'cart' distribution: (Deterministic value: 0).
        aVector := Dictionary new.
```

```
aVector at: 1 put: 0; at: 2 put: 0.3; at: 3 put: 0.3; at: 4 put: 0; at: 5 put: 0.2; at: 6 put: 0; at: 7 put:
0.2.
aWorkStation := WorkStation new.
aWorkStation initializeWithName: 'PPLS'
        andAmount: 1
        exitPosition: 5
        enterPosition: 1
        probabilities: aVector
        processTime: (Normal mean: 4 deviation: 0.5)
        type: 9; emptyQueueCapacity: 0; inputQueueCapacity: 20; outputQueueCapacity: 20.
bVector := Dictionary new.
bVector at: 1 put: 0.4; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0.3; at: 5 put: 0; at: 6 put: 0.3; at: 7 put:
0.
bWorkStation := WorkStation new.
bWorkStation initializeWithName: 'WS2'
        andAmount: 1
        exitPosition: 9
        enterPosition: 8
        probabilities: bVector
        processTime: (Normal mean: 4 deviation: 0.5)
        type: 1; emptyQueueCapacity: 5; inputQueueCapacity: 10; outputQueueCapacity: 20.
dVector := Dictionary new.
dVector at: 1 put: 0.6; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0; at: 5 put: 0; at: 6 put: 0.4; at: 7 put: 0.
dWorkStation := WorkStation new.
dWorkStation initializeWithName: 'WS3'
        andAmount: 1
        exitPosition: 15
        enterPosition: 14
        probabilities: dVector
        processTime: (Normal mean: 3 deviation: 0.4)
        type: 2; emptyQueueCapacity: 5; inputQueueCapacity: 5; outputQueueCapacity: 20.
fVector := Dictionary new.
fVector at: 1 put: 0; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0; at: 5 put: 0.5; at: 6 put: 0; at: 7 put: 0.5.
fWorkStation := WorkStation new.
fWorkStation initializeWithName: 'WS4'
        andAmount: 1
        exitPosition: 23
        enterPosition: 22
        probabilities: fVector
        processTime: (Normal mean: 4 deviation: 0.5)
        type: 3; emptyQueueCapacity: 5; inputQueueCapacity: 5; outputQueueCapacity: 20.
gVector := Dictionary new.
gVector at: 1 put: 0.5; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0.5; at: 5  put: 0; at: 6 put: 0; at: 7 put:
0.
gWorkStation := WorkStation new.
gWorkStation initializeWithName: 'WS5'
        andAmount: 1
        exitPosition: 31
        enterPosition: 30
        probabilities: gVector
        processTime: (Normal mean: 3 deviation: 0.4)
        type: 5; emptyQueueCapacity: 0; inputQueueCapacity: 5; outputQueueCapacity: 20.
hVector := Dictionary new.
```

```
        hVector at: 1 put: 0; at: 2 put: 0.25; at: 3 put: 0.25; at: 4 put: 0; at: 5 put: 0.25; at: 6 put: 0; at: 7
put: 0.25.
        hWorkStation := WorkStation new.
        hWorkStation initializeWithName: 'WS6'
                andAmount: 1
                exitPosition: 37
                enterPosition: 36
                probabilities: hVector
                processTime: (Normal mean: 4 deviation: 0.5)
                type: 5; emptyQueueCapacity: 5; inputQueueCapacity: 5; outputQueueCapacity: 20.
        iVector := Dictionary new.
        iVector at: 1 put: 0.3; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0.7; at: 5 put: 0; at: 6 put: 0; at: 7 put: 0.
        iWorkStation := WorkStation new.
        iWorkStation initializeWithName: 'WS7'
                andAmount: 1
                exitPosition: 44
                enterPosition: 43
                probabilities: iVector
                processTime: (Normal mean: 4 deviation: 0.5)
                type: 6; emptyQueueCapacity: 0; inputQueueCapacity: 10; outputQueueCapacity: 10.
        jVector := Dictionary new.
        jVector at: 1 put: 0; at: 2 put: 0; at: 3 put: 0; at: 4 put: 0; at: 5 put: 0; at: 6 put: 0; at: 7 put: 0.
        jWorkStation := WorkStation new.
        jWorkStation initializeWithName: 'BYPASS'
                andAmount: 1
                exitPosition: 42
                enterPosition: 17
                probabilities: jVector
                processTime: (Deterministic value: 8)
                type: 7; emptyQueueCapacity: 0; inputQueueCapacity: 111; outputQueueCapacity: 111.
        aConveyor := Conveyor new.
        aConveyor space: 10; speed: 1; trackSize: 48; numWorkStations: 8.
        sim := ConveyorSimulation new.
        sim activate.
        sim outputStream: Transcript.
        sim addCartGenerator: aCart.
        sim addConveyor: aConveyor.
        sim traceOn.
        sim startUp.
        sim schedule: [aConveyor move: 1] after: 10.
        sim schedule: [WorkStation generateAndLoadOrders]
                after: (Normal mean: 20 deviation: 4) next.
        sim clearStatisticsAt: 0.0.
        [sim time < aTime]
                whileTrue: [sim proceed].
        sim finishUp.
        Transcript endEntry! !
```

SmallTalk-80 Code For Class: Queue

```
Object subclass: #Queue
        instanceVariableNames: 'queue queueDiscipline queueLength timeInQueue entryTime '
        classVariableNames: ''
```

```
        poolDictionaries: ''
        category: 'ConveyorSimulation'!
Queue comment:
'This is a Queue object with
queueDiscipline : Describes the Queue displine (LIFO, FIFO, ... etc.) ( Default : FIFO)'!

!Queue methodsFor: 'removing'!
remove: anObject
"remove a particular item from the queue"
queueLength equals: (queueLength value - 1).
timeInQueue equals:  ( Simulation active time - anObject queueEntryTime).
^queue remove: anObject .!

removeFirst
"Answer the first item in the queue, and remove it"
|job|
job := queue removeFirst.
queueLength equals: (queueLength value - 1).
timeInQueue equals:  ( Simulation active time - job queueEntryTime).
^job!

removeLast
"Answer the first item in the queue, and remove it "
|job|
job := queue removeLast.
queueLength equals: (queueLength value - 1).
timeInQueue equals:  ( Simulation active time - job queueEntryTime).
^job! !

!Queue methodsFor: 'initialize-release'!

initialize
"Set up the statistics collection object, the queue itself is an ordered collection"
queueLength := TimeTrackedNumber new.
timeInQueue := ObsTrackedNumber new.
queue := OrderedCollection new.
^self! !

!Queue methodsFor: 'adding'!

add: aJob
"Add to the ordered collection, and collect statistics"
aJob timeStamp.
queue add: aJob.
queueLength equals: (queueLength value + 1).!

!Queue methodsFor: 'testing'!

hasSpace
"Unless overridden, queues have unlimited capacity"
^true!

isEmpty
"Answer whether the queue is empty"
```

```
^queue isEmpty!

!Queue methodsFor: 'statistics'!
display
        "display results"
        timeInQueue display!

printResultsOn: aStream
        "answer the statistics on the stream in a formatted style"
        aStream cr; nextPutAll: 'Length:    '.
        queueLength printResultsOn: aStream.
        aStream cr; nextPutAll: 'Wait Time: '.
        timeInQueue printResultsOn: aStream.
        aStream cr!

results
        "Answer the statistics in an array of size 2"
        | stats |
        stats := Array new: 2.
        stats at: 1 put: queueLength results; at: 2 put: timeInQueue results.
        ^stats! !

!Queue methodsFor: 'accessing'!

first
        "First item in the queue"
        ^queue first!

last
        "Answer the last item in the queue"
        ^queue last!

queueLength
        "Answer the current queue length"
        ^queueLength value! !

!Queue methodsFor: 'task language'!

fifo
        "Implement FIFO discipline"
        ^queue first!

lifo
        "Implement LIFO discipline"
        ^queue last!

next
^self perform: queueDiscipline!

setQDiscipline: aQDiscipline
        "set queue discipline"
        queueDiscipline := aQDiscipline.
        ^self!
```

```
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Queue class
	instanceVariableNames: ''!

!Queue class methodsFor: 'instance creation'!
new
" create a new queue "
^super new initialize! !
```

SmallTalk-80 Code For Class: CapacitatedQueue

```
Queue subclass: #CapacitatedQueue
	instanceVariableNames: 'capacity '
	classVariableNames: ''
	poolDictionaries: ''
	category: 'ConveyorSimulation'!

!CapacitatedQueue methodsFor: 'testing'!

hasSpace
	"Answer if there is enough place in the queue"
	queueLength value < capacity
		ifTrue: [^True]
		ifFalse: [^False]! !

!CapacitatedQueue methodsFor: 'initialize-release'!

capacity
	"initialize the instance vars of superclasses and set the capacity"
	super initialize.
	^capacity!

initCapacity: anAmount
	"initialize the instance vars of superclasses and set the capacity"
	super initialize.
	capacity := anAmount! !

!CapacitatedQueue methodsFor: 'adding'!

add: aJob
	"This job had reserved a place. Now it has entered. Add to the queue."
	queueLength value > capacity ifTrue: [self error: 'Queue capacity exceeded'].
	super add: aJob! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

CapacitatedQueue class
	instanceVariableNames: ''!

!CapacitatedQueue class methodsFor: 'instance creation'!

capacity: amount
"Create a capacitated queue with capacity amount"
```

```
^super new initCapacity: amount! !
```

SmallTalk-80 Code For Class: Conveyor

```
Resource subclass: #Conveyor
        instanceVariableNames: 'space speed trackSize track inputPos outputPos numWorkStations
hookUtilization '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation'!

!Conveyor methodsFor: 'initialize-release'!

initializeWith: anArray
        "This is to initialize the instance vars of the Conveyor class from the array
        defined through the user input. This will also store all the Hook1 instances
        in the conveyor loop."
        | aWS1 aPos1 aWS2 aPos2 |
        space := (anArray at: 2) asNumber.
        speed := (anArray at: 1) asNumber.
        trackSize := (anArray at: 3) asNumber.
        numWorkStations := (anArray at: 4) asNumber.
        track := Dictionary new.
        inputPos := Dictionary new.
        outputPos := Dictionary new.
        1 to: trackSize do: [:i | track at: i put: (Hook1
                                newWithName: 'Hook' andAmount: 1 hookNumber: i hookPosition: i
                                hookStatus: 0)].
        1 to: numWorkStations do:
                [:i | aWS1 := WorkStation returnInstanceWithName: (WorkStation DestinationList at:i).
                aPos1 := aWS1 enterPosition.
                inputPos at: i put: aPos1].
        1 to: numWorkStations do:
                [:i | aWS2 := WorkStation returnInstanceWithName: (WorkStation DestinationList at:i).
                aPos2 := aWS2 exitPosition.
                outputPos at: i put: aPos2]!

numWorkStations: aNumber
        "This is to initialize the number of workstations needed in the conveyor loop."
        numWorkStations := aNumber!

space: aSpace
        "initialize the space between two consecutive hooks"
        space := aSpace!

speed: aSpeed
        "initialize the speed of the conveyor loop"
        speed := aSpeed!

trackSize: aSize
        "initialize the track size that is the number of Hook1 instances desired in the conveyor system."
        trackSize := aSize! !
```

```
!Conveyor methodsFor: 'accessing'!

numWorkStations
        "return the number of workstations desired in the system"
        ^numWorkStations!

space
        "return the space between two hooks in the conveyor loop."
        ^space!
speed
        "return the speed of the conveyor loop."
        ^speed!

track
        "return the track object that is all the hooks stored in the track (conveyor loop)"
        ^track!

trackSize
        "return the number of hooks in the conveyor system"
        ^trackSize! !

!Conveyor methodsFor: 'task language'!

move: aNumber
 "updates all the hook positions at a regular interval of time"

self acceptCarts.
self deliverCarts.
self updateHookPositions1: aNumber.
ConveyorSimulation active schedule: [ self move: aNumber] after: 10.!

release: aHook at: aPosition
 "release the hook when the cart gets to its destination at aPosition"
 "nothing to do"
"SimScript cr; nextPutAll: name, '   is at: ', aPosition name."!

updateHookPositions1: numHookSpaces
        "after moving a numHookSpaces we have to update the hook positions. The hook numbers stay
        the same only their positions in the conveyor loop change."
        | hook tempTrack |
        SimScript cr; nextPutAll: self printString , ' all the hook positions are updated at ' ,
        ConveyorSimulation active time printString.
        [numHookSpaces > self trackSize]
                whileTrue: [numHookSpaces = numHookSpaces - self trackSize].
        tempTrack := Dictionary new.
        1 to: numHookSpaces do: [:i | tempTrack at: i put: (track at: self trackSize - numHookSpaces +
i)].
        self trackSize to: 1 by: -1
                do: [:i | i > numHookSpaces
                                ifTrue: [track at: i put: (track at: i - numHookSpaces)]
                                ifFalse: [track at: i put: (tempTrack at: i)]].
        1 to: self trackSize do:
                [:i | hook := self returnHookAvailableAtPosition: i.
                hook hookPosition + numHookSpaces > self trackSize
```

```
                    ifTrue: [hook hookPosition: hook hookPosition + numHookSpaces - self
trackSize]
                    ifFalse: [hook hookPosition: hook hookPosition + numHookSpaces]]! !

!Conveyor methodsFor: 'testing'!

acceptCarts
        "This method is to allow the hook to grap a cart from the output queue of        the workstation
        and transport it to its next destination"
        | aHook aWS1 cart |
        1 to: numWorkStations do:
                [:i | aHook := track at: (outputPos at: i).
                aHook hookStatus = 0
                        ifTrue:
                                [aWS1 := WorkStation returnInstanceWithName: (WorkStation
                        DestinationList at: i).
                                aWS1 outputQueueEmpty not
                                        ifTrue:
                                                [cart := aWS1 next2.
                                                aHook cartHeld: (aWS1 remove2: cart).
                                                cart checkShortestRouteFrom: aWS1.
                                                aHook hookStatus: 1.
                                                aHook uitlization1: aHook hookStatus.
                                                SimScript cr; nextPutAll: cart printString , ' obtained
aHook' , aHook hookNumber printString]]]!

deliverCarts
        "This method is to allow the hook to deliver a cart to the input queue of the workstation and
        allow it to get serviced"
        | aHook aWS1 aWS2 aCart true3 pos1 pos2 travelTime |
        1 to: numWorkStations do:
                [:i | aHook := track at: (inputPos at: i).
                aHook hookStatus = 1
                        ifTrue: [aWS1 := WorkStation returnInstanceWithName: (WorkStation
DestinationList at: i).
                                aWS2 := aHook cartHeld destination.
                                true3 := aWS1 hasInputSpace.
                                (aWS2 name = aWS1 name and: [true3 = True])
                                        ifTrue:  [aCart := aHook cartHeld.
                                                aHook cartDispose.
                                                aHook hookStatus: 0.
                                                aHook uitlization: aHook hookStatus.
                                                pos1 := (WorkStation returnInstanceWithName:
aCart currentWorkStation) exitPosition.
                                                pos2 := aWS2 enterPosition.
                                                travelTime := (pos2 - pos1 * (self space / self
speed)) abs.
                                                aHook hookProcess: travelTime.
                                                SimScript cr; nextPutAll: aCart printString , 'left the
Hook' , aHook hookNumber printString. aCart resume]]]!

isHookAvailableAtPosition: aPosition
        "This method is to check if the hook at a certain position is idle. If it is then        it can provide
        service to a cart waiting for service"
```

```
        (track at: aPosition) hookStatus = 0
                ifTrue: [^True]
                ifFalse: [^False]!

returnHookAvailableAtPosition: aPosition
        "return a Hook instance at a position"
        ^track at: aPosition! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Conveyor class
        instanceVariableNames: "!

!Conveyor class methodsFor: 'instance creation'!

newWith: anArray
        "Create an instance of conveyor by using the data provided by the user through the user
        interface"
        ^super new initializeWith: anArray! !
Cart initialize!
```

SmallTalk-80 Code For Class: ConExperimentView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:13:25 pm'!

View subclass: #ConExperimentView
        instanceVariableNames: "
        classVariableNames: "
        poolDictionaries: "
        category: 'ConveyorSimulation Interface'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConExperimentView class
        instanceVariableNames: "!

!ConExperimentView class methodsFor: 'instance creation'!

openOn: convModel
        "Open a dialog view to ask question about the experiment"

        | theModel top doneModel ctrl1 ctrl2 ctrl3 ctrl4 listView listWrapper plug done launchWindow
height origin |
        "convModel resetExperiment."
        doneModel := ValueHolder with: false.
        theModel := ValueHolder with: (Array new: 6).
        (top := DialogView model: doneModel) leftIndent: 10; addVerticalSpace: 3; addTextLabel:
'SIMULATION EXPERIMENT'.
        top yPosition: 30; leftIndent: 10; rightIndent: 130; addTextLabel: 'Seed for the experiment';
yPosition: 30; leftIndent: 150; rightIndent: 270; addTextLabel: 'Termination time'.
        top yPosition: 50; leftIndent: 10; rightIndent: 130.
        ctrl1 := top addTextFieldOn: ((PluggableAdaptor on: theModel)
                                        collectionIndex: 1) initially: ".
        top yPosition: 50; leftIndent: 150; rightIndent: 270.
        ctrl2 := top addTextFieldOn: ((PluggableAdaptor on: theModel)
```

```
                                        collectionIndex: 2) initially: ''.
        top yPosition: 80; leftIndent: 10; rightIndent: 130; addTextLabel: 'Clear statistics at'; yPosition:
80; leftIndent: 150; rightIndent: 270; addTextLabel: 'Trace on at'.
        top yPosition: 100; leftIndent: 10; rightIndent: 130.
        ctrl3 := top addTextFieldOn: ((PluggableAdaptor on: theModel)
                                        collectionIndex: 3) initially: ''.
        top yPosition: 100; leftIndent: 150; rightIndent: 270.
        ctrl4 := top addTextFieldOn: ((PluggableAdaptor on: theModel)
                                        collectionIndex: 4) initially: ''.
        done := LabeledBooleanView new model: (plug := (PluggableAdaptor on: doneModel)
                                                        selectValue: true).
        done beVisual: 'DONE ' asComposedText.
        done controller beTriggerOnUp.
        top addVerticalSpace: 4; leftIndent: 10; rightIndent: 270; addWrapper: ((BorderedWrapper on:
done)
                        inset: 2)
                atX: 0.5.
        plug
                putBlock:
                        [:m :v |
                        ctrl1 accept.
                        ctrl2 accept.
                        ctrl3 accept.
                        ctrl4 accept.
                        m value: v].
        ctrl1 crBlock: [].
        ctrl2 crBlock: [].
        ctrl3 crBlock: [].
        ctrl4 crBlock: [].
        top width: 280.
        launchWindow := ScheduledControllers activeController view.
        height := launchWindow extent y.
        origin := launchWindow globalOrigin + (50 @ (height + 150)).
        top openAt: origin.
        convModel carryOutExperimentWith: theModel value.

!ConExperimentView class methodsFor: 'printing'!

printString
^'Experiment'! !
```

SmallTalk-80 Code For Class: ConResultView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:13:38 pm'!

View subclass: #ConResultView
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation Interface'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConResultView class
```

```
        instanceVariableNames: "!

!ConResultView class methodsFor: 'instance creation'!

openOn: conveyorModel
        "Display the results"
        | menu action |
        menu := PopUpMenu labels: 'Show Statistics\Show Histograms\Show Trace ' withCRs lines: #(2
).
        action := menu startUp.
        action = 0 ifFalse: [action = 1
                        ifTrue: [conveyorModel showStatistics]
                        ifFalse: [action = 2
                                        ifTrue: [conveyorModel showHistograms]
                                        ifFalse: [conveyorModel showTrace]]]! !

!ConResultView class methodsFor: 'printing'!

printString
^'Results'! !
```

SmallTalk-80 Code For Class: ConveyorDefinitionView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:13:52 pm'!

View subclass: #ConveyorDefinitionView
        instanceVariableNames: "
        classVariableNames: "
        poolDictionaries: "
        category: 'ConveyorSimulation Interface'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConveyorDefinitionView class
        instanceVariableNames: "!

!ConveyorDefinitionView class methodsFor: 'comparing'!

<= another
^self name <= another name! !

!ConveyorDefinitionView class methodsFor: 'instance creation'!

openOn: convModel
        | theModel top    doneModel ctrl1 plug done ctrl2 launchWindow height origin ctrl3 ctrl4 ctrl5 |
        doneModel := ValueHolder with: false.
        theModel := ValueHolder with: (Array new: 5).
        (top := DialogView model: doneModel) addVerticalSpace: 3; addTextLabel: 'Conveyor
Definition'.
        top yPosition: 30; leftIndent: 10; rightIndent: 130; addTextLabel: 'Speed'; yPosition: 30;
leftIndent: 150; rightIndent: 270; addTextLabel: 'Space'.
        top yPosition: 50; leftIndent: 10; rightIndent: 130.
        ctrl1 := top addTextFieldOn: ((PluggableAdaptor on: theModel) collectionIndex: 1) initially: ".
top yPosition: 50; leftIndent: 150; rightIndent: 270.
```

```
        ctrl2 := top addTextFieldOn: ((PluggableAdaptor on: theModel) collectionIndex: 2) initially: ''.
        top yPosition: 80; leftIndent: 10; rightIndent: 130; addTextLabel: 'TackSize'; yPosition: 80;
leftIndent: 150; rightIndent: 270; addTextLabel: 'Num WorkStat'.
        top yPosition: 100; leftIndent: 10; rightIndent: 130.
ctrl3 := top addTextFieldOn: ((PluggableAdaptor on: theModel) collectionIndex: 3) initially: ''.
        top yPosition: 100; leftIndent: 150; rightIndent: 270.
        ctrl4 := top addTextFieldOn: ((PluggableAdaptor on: theModel) collectionIndex: 4) initially: ''.
top yPosition: 120; leftIndent: 10; rightIndent: 130; addTextLabel: 'No. of Carts'.
        top yPosition: 140; leftIndent: 10; rightIndent: 130.
        ctrl5 := top addTextFieldOn: ((PluggableAdaptor on: theModel) collectionIndex: 5) initially: ''.
        done := LabeledBooleanView new model: (plug := (PluggableAdaptor on: doneModel)
                                                        selectValue: true).
        done beVisual: 'DONE ' asComposedText.
        done controller beTriggerOnUp.
        top addVerticalSpace: 4; leftIndent: 10; rightIndent: 270; addWrapper: ((BorderedWrapper on:
done)
                        inset: 2)
                atX: 0.5.
        plug
                putBlock:
                        [:m :v |
                        ctrl1 accept.
                        ctrl2 accept.
                        ctrl3 accept.
                        ctrl4 accept.
                        ctrl5 accept.
                        m value: v].
        ctrl1 crBlock: [].
        ctrl2 crBlock: [].
        ctrl3 crBlock: [].
        ctrl4 crBlock: [].
        ctrl5 crBlock: [].
        top width: 280.
        launchWindow := ScheduledControllers activeController view.
        height := launchWindow extent y.
        origin :=  launchWindow globalOrigin + (50 @ (height + 150)).
        top openAt: origin.
        convModel informSimulationWith: theModel value.! !

!ConveyorDefinitionView class methodsFor: 'printing'!

printString
^'Conveyor Definition Browser'! !
```

SmallTalk-80 Code For Class: ConveyorView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:14:16 pm'!

View subclass: #ConveyorView
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation Interface'!
```

```
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConveyorView class
	instanceVariableNames: ''!

!ConveyorView class methodsFor: 'instance creation'!

open
	| window   topView launchWindow  origin height   convModel |
	convModel := ConvModel new.
	window := ScheduledWindow new.
	window label: 'ConveyorSimulation Launcher'.
	window minimumSize: 200 @ 100.
	topView := CompositePart new.
	topView add: (LookPreferences edgeDecorator on: (SelectionInListView
						on: convModel
						printItems: true
						oneItem: false
						aspect: #viewName
						change: #viewChange:
						list: #viewList
						menu: nil
						initialSelection: #viewName))
		borderedIn: (0@0 extent: 1@1).
	window component: topView.
	launchWindow := ScheduledControllers activeController view.
	height := launchWindow extent y.
	origin :=  launchWindow globalOrigin + (50 @ (height + 10)).
		window openAt: origin! !
```

SmallTalk-80 Code For Class: ConvModel

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:14:28 pm'!

Model subclass: #ConvModel
	instanceVariableNames: 'workStationSelection workStationMenu currentObject viewSelection
processTimeSelection wsSelection histogramSelection histograms histogramNameList
histogramStationList workstations arrivalDistributions probabilityMenu probabilities aConveyorVector
probabilitySelection providedHistogramList processTimereset processTimeselection processTime '
	classVariableNames: 'TextMenu '
	poolDictionaries: ''
	category: 'ConveyorSimulation Interface'!

!ConvModel methodsFor: 'launching views'!

viewChange: aView
	"The user wants to open a view"
	aView isNil
		ifTrue: [viewSelection := nil]
		ifFalse: [viewSelection := aView. aView openOn: self]!

viewList
	| list |
	list := OrderedCollection new: 5.
```

```
list add:  WorkStationDefinitionView; add: ConveyorDefinitionView; add: ProbabilitiesDefinitionView;
add:  ConExperimentView; add: ConResultView.
^list!

viewName
^viewSelection! !

!ConvModel methodsFor: 'initialize-release'!

initialize
"Initialize the model"
workstations := Dictionary new.! !

!ConvModel methodsFor: 'workstation definition'!

add: aWorkStation
"Add a workstation to the workstation list"
workstations at: aWorkStation name put: aWorkStation.!

addWorkStation
        "Add a new workstation to the  work station list of this model"
        self addWorkStation: #workStation!

addWorkStation: aWorkStation
        "Add a new workstation to the workstation list of this model"
        | description in out   em exit enter tipe   aVector   workst |
aVector := Dictionary new.
workst := WorkStation new.
processTimereset := false.
currentObject := workst.
currentObject := WorkStation newWithName: 'current' andAmount: 1 exitPosition: 1 enterPosition: 1
probabilities: aVector processTime: 0 type: 1.
self add: currentObject.
        description := DialogView getWorkStationWithDefault: #('WorkStationName' 1 1 0 0 3 0 0 0)
at: self dialogDisplayPoint model: self.
        (description at: 1)
                = '' ifFalse:
                        ["currentObject name: (description at: 1)."
                        workstations do: [:workstation | workstation name = (description at: 1)
                                        ifTrue:
                                                [(PopUpMenu labels: 'WorkStation with that name
already exists') startUp. ^self]]].
(description at: 1) =''
ifFalse:[self  renameWorkStationNamed: 'current' to: (description at: 1).
                        in := (description at: 2) asNumber.
                        currentObject inputQueueCapacity: in.
                        out := (description at: 3) asNumber.
                        currentObject outputQueueCapacity: out.
                        em := (description at: 4) asNumber.
                        tipe := (description at: 5) asNumber.
                        currentObject type: tipe.
                        currentObject emptyQueueCapacity: em.
                        exit := (description at: 6) asNumber.
                        currentObject exitPosition: exit.
```

```
                    enter := (description at: 7) asNumber.
                    currentObject enterPosition: enter.].
        self changed: #workStationName!
dialogDisplayPoint
        "Place to show a dialog view"
        | window height origin |
        window := ScheduledControllers activeController view.
        height := window extent y.
        origin := window globalOrigin + (0 @ (height / 2)) + (170 @ 60).
        origin x: origin x rounded.
        origin y: origin y rounded.
        ^origin!

initialProcessTime
"Answer the process time distribution of the  current operation"
self workStationName isNil
ifTrue: [^nil]
ifFalse: [^self workStationName processTime]!

modifyWorkStation
        "Add a new plant to the plant list of this model"
        | description oldDescription |
        currentObject := workStationSelection.
        oldDescription := Array new: 7.
        oldDescription at: 1 put: currentObject name.
        oldDescription at: 2 put: currentObject inputQueueCapacity.
        oldDescription at: 3 put: currentObject outputQueueCapacity.
        oldDescription at: 4 put: currentObject emptyQueueCapacity.
        oldDescription at: 5 put: currentObject type.
        oldDescription at: 6 put: currentObject exitPosition.
        oldDescription at: 7 put: currentObject enterPosition.
                    description := DialogView
                getWorkStationWithDefault: oldDescription at: self dialogDisplayPoint model: self.
        (description at: 1) = ''
                ifFalse: [self renameWorkStationNamed: currentObject name to: (description at: 1).
                    currentObject inputQueueCapacity: (description at: 2) asNumber.
                    currentObject outputQueueCapacity: (description at: 3) asNumber.
                    currentObject emptyQueueCapacity: (description at: 4) asNumber.
                    currentObject type: (description at: 5) asNumber.
                    currentObject exitPosition: (description at: 6) asNumber.
                    currentObject enterPosition: (description at: 7) asNumber.
        currentObject := nil.
        self changed: #workStationName!

probabilitiesList
"Answer the list of objects for which probabilities are needed"
| list |
^ self wsList!

processTime
"Answer the name of the process time distribution  currently selected"
^processTimeselection!

processTimeChange: aDist
```

```
        "The user of the operation definition dialog view has changed the selection of the distribution"
        aDist isNil
                ifTrue: [processTimeselection := nil. processTimereset := false]
                ifFalse:
                        [processTimereset
                                ifTrue: [processTimereset := false]
                                ifFalse: [aDist getParameters.
                                        currentObject  processTime: aDist].
                                        processTimeselection := aDist]!

processTimeList
        "Answer the list of  process time for the current operations "
        ^currentObject processTimeList!

removeWorkStation
        "the user wants to remove a workStation from the workStation list "
        | workstation |
        workstation := workStationSelection.
        workStationSelection := nil.
workstations removeKey: workstation  name.
        self changed: #workStationName!

renameWorkStation
        "the user wants to rename a workStation from the workStation list. "
        | aName  |
        aName := DialogView request: 'New name for the workStation? '.
        aName = '' ifTrue: [^self].
        self renameWorkStationNamed: workStationSelection name to: aName.
        self changed: #workStationName!

renameWorkStationNamed: oldName to: aName
        "the user wants to rename a workCenter from the workCenter list. "
        | name  workStation |
        oldName = aName ifFalse: [workStation := workstations at: oldName.
        name := aName.
        [workstations includesKey: name]
        whileTrue: [name := (DialogView request: 'This name is already used. Provide another name') ].
        workstations removeKey: oldName.
        workStation name: name.
        workstations at: name put: workStation]!

workStationChange: aWorkStation
        "The user of the workstation definition window has changed the selection of the workstation.
        Inform the dependent lists"
        aWorkStation isNil
                ifTrue: [workStationSelection := nil]
                ifFalse: [workStationSelection := aWorkStation].
        self changed: #text!

workStationMenu
        "Answer an ActionMenu of operations on workStations that is to be displayed
        when the operate menu button is pressed."
        workStationSelection isNil
                ifTrue: [workStationMenu _ ActionMenu labels: 'add a work station' withCRs
```

```
                    lines: #() selectors: #(#addWorkStation)]
                ifFalse: [(workStationSelection isKindOf: WorkStation)
                    ifTrue: [workStationMenu _ ActionMenu
                    labels: 'add a workStation\modify-review\remove\rename' withCRs lines: #(2 )
selectors: #(#addWorkStation #modifyWorkStation #removeWorkStation #renameWorkStation )]
        ifFalse: [workStationMenu _ ActionMenu
                    labels: 'add a workStation\modify-review\remove\rename' withCRs lines: #(2 )
selectors: #(#addWorkStation  #modifyWorkStation #removeWorkStation #renameWorkStation )]].
        ^workStationMenu!

workStationName
        "Answer the list of workstations defined so far"
        ^workStationSelection!

wsList
        "Answer the list of workstations defined so far"
        ^workstations  values asSortedCollection.

!ConvModel methodsFor: 'accessing'!

conveyorVector
        ^aConveyorVector!

workstation
        |nameWorkStation |
        workstations isEmpty
                ifFalse: [nameWorkStation := DialogView request: 'Name of workstation'.
                ^workstations at: nameWorkStation put: (WorkStation new name: nameWorkStation)].
                ^workstations values asOrderedCollection at: 1! !

!ConvModel methodsFor: 'information'!

informSimulationWith: anArray
aConveyorVector := anArray.

!ConvModel methodsFor: 'probability definition'!

probabilityMenu
        "Answer an ActionMenu of operations on workStations that is to be displayed
        when the operate menu button is pressed."
        (workStationSelection isKindOf: WorkStation) ifTrue: [ probabilityMenu _ ActionMenu
                                                labels: 'Specify Probabilities' withCRs
                                                lines: #(2 )
                                                selectors: #(#specifyProbabilities)].
        ^probabilityMenu!

specifyProbabilities
        | aList aSize anObject  aVector anArray bList |
        aList := workstations values asSortedCollection.
        aSize := aList size.
        anObject := workStationSelection.
        anArray := Array new: aSize.
        aVector := DialogView getProbabilitiesWithDefault: anArray
        at: self dialogDisplayPoint model: self.
```

```
1 to: aSize do: [: il ( anObject probabilities) at: i put: (aVector at: i) asNumber].
bList := Dictionary new.
1 to: (aList size) do: [:i | bList at: i put: ((aList  at: i ) name)].
WorkStation setDestinations: bList.! !

!ConvModel methodsFor: 'experimenting'!

carryOutExperimentWith: anArray
| aCart sim bArray aConveyor aGen |
sim := ConveyorSimulation new.
bArray := self conveyorVector.
aGen := CartGenerator new.
aGen totalCount: (bArray at: 5).
aCart :=aGen name: 'cart' distribution: (Deterministic value: 0).
bArray := self conveyorVector.
aConveyor := Conveyor newWith: bArray.
sim activate.
sim addConveyor: aConveyor.
sim outputStream: Transcript.
sim addCartGenerator:  aCart.
sim startUp .
sim clearStatisticsAt: (anArray at: 3) asNumber.
sim schedule: [aConveyor move: 1] after: 10.
sim schedule: [WorkStation generateAndLoadOrders]
                after: (Normal mean: 10 deviation: 2) next.
        Cursor execute showWhile: [[sim time < (anArray at: 2) asNumber]
                whileTrue: [sim proceed]].
        sim finishUp.
        Transcript endEntry! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ConvModel class
        instanceVariableNames: "!

!ConvModel class methodsFor: 'instance creation'!

new
"Create an instance of the sim model"
^super new initialize! !

!ConvModel class methodsFor: 'initialize-release'!

initialize
"The menu to appear in the code view"
TextMenu :=
        ActionMenu
                        labelList: #((again undo) (copy cut paste)  (accept cancel))
                        selectors: #(again undo copySelection cut paste  accept cancel)! !

ConvModel initialize!
```

SmallTalk-80 Code For Class: ProbabilitiesDefintionView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:48:37 pm'!
```

```
View subclass: #ProbabilitiesDefinitionView
	instanceVariableNames: 'topView '
	classVariableNames: ''
	poolDictionaries: ''
	category: 'ConveyorSimulation Interface'!

!ProbabilitiesDefinitionView methodsFor: 'initialize-release'!

openOn: aModel
	"Open a workstation definition window on the model."
	| window buttonOffset launchWindow height origin |
	buttonOffset := LabeledBooleanView defaultHeight negated.
	window := ScheduledWindow new.
	window label: 'Probability Definition Browser'.
	window minimumSize: 300 @ 150.
	topView := CompositePart new.
	topView add: self borderedIn: ((LayoutFrame new) leftOffset: 0.0; topFraction: 0.0;
rightFraction: 1.0; bottomFraction: 0.0 offset: buttonOffset negated + 4).
	topView add: (LookPreferences edgeDecorator on: (SelectionInListView
					on: aModel
					printItems: true
					oneItem: false
					aspect: #workStationName
					change: #workStationChange:
					list: #wsList
					menu: #probabilityMenu
					initialSelection: #workStationName))
			in: ((LayoutFrame new) leftFraction: 0.0; leftOffset: 0.0; topFraction: 0.0 offset:
buttonOffset negated + 4; rightFraction: 0.66; bottomFraction: 1 offset: buttonOffset).
	window component: topView.
	launchWindow := ScheduledControllers activeController view.
	height := launchWindow extent y.
	origin := launchWindow globalOrigin + (0@ (height + 5)).
	window openAt: origin! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ProbabilitiesDefinitionView class
	instanceVariableNames: ''!

!ProbabilitiesDefinitionView class methodsFor: 'printing'!

printString
^'Probabilities Definition Browser'! !

!ProbabilitiesDefinitionView class methodsFor: 'comparing'!

<= another
^self name <= another name! !

!ProbabilitiesDefinitionView class methodsFor: 'instance creation'!

openOn: aModel
	"Open a plant definition window on a SimModel"
```

```
        | view |
        view := self new initialize.
        view model: aModel.
        view openOn: aModel! !
```

SmallTalk-80 Code For Class: StringListView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:48:53 pm'!

SelectionInListView subclass: #StringInListView
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation Interface'!


!StringInListView methodsFor: 'list access'!

displayableLinesFrom: anArray
        "Answer a collection of displayable lines from anArray."

        ^printItems
                ifTrue: [anArray collect:
                                        [:each | each  copyUpTo: Character cr]]
                ifFalse: [anArray]! !
```

SmallTalk-80 Code For Class: WorkStationDefintionView

```
'From Objectworks(r)\Smalltalk, Release 4 of 25 February 1991 on 21 April 1992 at 12:49:14 pm'!

View subclass: #WorkStationDefinitionView
        instanceVariableNames: 'codeView boxView topView buttonSelection '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ConveyorSimulation Interface'!

!WorkStationDefinitionView methodsFor: 'initialize-release'!

openOn: aModel
        "Open a workstation definition window on the model."
        "WorkStationDefinitionView openOn: ConveyorModel new"
        | window buttonOffset launchWindow height origin |
        buttonOffset := LabeledBooleanView defaultHeight negated.
        window := ScheduledWindow new.
        window label: 'Workstation Definition Browser'.
        window minimumSize: 300 @ 150.
        topView := CompositePart new.
        topView add: self borderedIn: ((LayoutFrame new) leftOffset: 0.0; topFraction: 0.0;
rightFraction: 1.0; bottomFraction: 0.0 offset: buttonOffset negated + 4).
        topView add: (LookPreferences edgeDecorator on: (SelectionInListView
                                        on: aModel
                                        printItems: true
                                        oneItem: false
                                        aspect: #workStationName
                                        change: #workStationChange:
```

```
                                        list: #wsList
                                        menu: #workStationMenu
                                        initialSelection: #workStationName))
                        in: ((LayoutFrame new) leftFraction: 0.0; leftOffset: 0.0; topFraction: 0.0 offset:
buttonOffset negated + 4; rightFraction: 0.66; bottomFraction: 1 offset: buttonOffset).
        window component: topView.
        launchWindow := ScheduledControllers activeController view.
        height := launchWindow extent y.
        origin := launchWindow globalOrigin + (0@ (height + 5)).
        window openAt: origin! !

!WorkStationDefinitionView methodsFor: 'displaying'!

displayOn: aGraphicsContext

"Create or refresh the view. All the subview take care of
 themselves. The function of this message is to write the heading
 of the lists"

| aRectangle cellLabel width height xCoord yCoord |

aRectangle :=  Rectangle origin: Point zero extent: self container bounds extent.
cellLabel  := 'Workstations' asText allBold asComposedText.
width := cellLabel width.
height := cellLabel height.
xCoord :=   (aRectangle width * 0.33 - width) / 2 .
yCoord :=  (aRectangle height  - height )/ 2 -1.
cellLabel displayOn: aGraphicsContext  at: (xCoord @yCoord).! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

WorkStationDefinitionView class
        instanceVariableNames: ''!

!WorkStationDefinitionView class methodsFor: 'printing'!

printString
^'WorkStation Definition Browser'! !

!WorkStationDefinitionView class methodsFor: 'instance creation'!

openOn: aModel
        "Open a workstation definition window on a ConveyorModel"
        | view |
        view := self new initialize.
        view model: aModel.
        view openOn: aModel! !

!WorkStationDefinitionView class methodsFor: 'comparing'!

<= another
^self name <= another name! !
```

APPENDIX B

USER MANUAL

A single-loop conveyor system is the simplest conveyor system configuration. The system does not have an elevator or a transfer section. However, the system can have bypass sections for providing shorter routes between stations on the conveyor loop. The following appendix will provide the user with information that will guide him/her in entering the conveyor system model parameters. This information will be divided in three sections: (1) workstation parameters specification, (2) conveyor parameters specification , and (3) probability vector specifications. Theses three sections are represented by three browsers in the user interface.

Workstation parameters specification: Each workstation has to be defined by the following parameters:

- Name: The name of the workstation is specified as desired by the user except in the case of the Production Programming and Load Station (PPLS) and the ByPass section (BYPASS). If a Production Programming and Load Station is to be included in the model it has to have the name PPLS. A ByPass section should also have the name BYPASS.

- Input/Output/Empty queues capacities: The only restrictions on the queue capacities is in the case of the PPLS workstation. Both the input and output queues should have capacities equal to the number of carts included in the model.

- Type: The model accommodates seven types of workstations. The following table illustrates the numbers assigned for the different types of workstations:

TABLE 20

WORKSTATION TYPES

Workstation Name	Type
UALSC	1
LSWAT	2
LODST	3
OLUNLOD	4
ULST	5
BYPASS	6
PPLS	7

- Process time: The workstation browser provides a list of probability distributions that the user can use.

- Exit/Enter positions: Each workstation will have an entering position where carts get delivered by the hooks (this position is the same as the input queue position) and an exiting position where carts get picked up by the hooks (this position is the same as the output queue position). The user has to enter a number between 1 and the number of hooks included in the model. The exit position is always smaller than the entering position of the workstation (this is explained in details in Chapter VI).

Conveyor Parameters Specification: Each conveyor will have the following parameters:

- Speed: The user can enter any speed with any units desired for the conveyor system.

- Space: The user should enter the space desired between two consecutive hooks. The unit of the space should match the unit of the speed. For example if the speed is specified in ft/min., the space should be specified in ft.

- track size: The user should enter the maximum number of hooks to be included in the conveyor loop. This number when multiplied by the space specified will result in the length of the conveyor loop.

- No. of Carts: The user should enter the number of carts to be accommodated by the model.

Probability Vector Specification: As explained in chapter VI, each workstation will have a probability vector that will specify the probability with which a workstation sends a specific cart to another workstation. The probability vector for the BYPASS section should be the vector zero. Also each other workstation will have a zero probability of sending a cart to the BYPASS. The probability vector for a specific workstation has to have zero in the entry of this specific workstation; since a workstation does not send carts to itself. The user has to make sure that the sum of all the probabilities add up to one.

APPENDIX C

AHP CALCULATIONS

AHP Matrix Calculation Set 1

NODE 1.1 - Simulation Paradigm

Links from Lower Level:

1) Node 2.1 - Model effectiveness
2) Node 2.2 - Model developer's potency and modeling effort
3) Node 2.3 - Performance considerations

Original weights

Row \ Col	1	2	3
1	1.000	7.000	9.000
2	0.143	1.000	5.000
3	0.111	0.200	1.000

Normalized weights

Row \ Col	1	2	3	Weights
1	0.797	0.853	0.600	0.750
2	0.114	0.122	0.333	0.189
3	0.089	0.024	0.066	0.059

Estimation of matrix consistency:

Lambda max:	3.210
Consistency index:	0.105
Consistency ratio:	0.181

AHP Matrix Calculation Set 2

NODE 2.1 - Model Effectiveness

Links from Lower Level:

1) Node 3.1 - Model reusability
2) Node 3.4 - Output provisions
3) Node 3.6 - Graphics/User interface capability
4) Node 3.10-Model representation correspondence to the real system
5) Node 3.11-Model flexibility

Original weights

Row \ Col	1	2	3	4	5
1	1.000	0.200	1.000	0.200	0.143
2	5.000	1.000	5.000	3.000	1.000
3	1.000	0.200	1.000	0.200	0.167
4	5.000	0.333	5.000	1.000	4.000
5	7.000	1.000	6.000	0.250	1.000

Normalized columns

Row \ Col	1	2	3	4	5	Weights
1	0.053	0.073	0.055	0.043	0.023	0.049
2	0.263	0.366	0.277	0.645	0.158	0.342
3	0.053	0.073	0.055	0.043	0.026	0.050
4	0.263	0.121	0.277	0.215	0.633	0.302
5	0.368	0.365	0.333	0.053	0.158	0.255

Estimation of matrix consistency:

Lambda max:	5.637
Consistency index:	0.159
Consistency ratio:	0.142

AHP Matrix Calculation Set 3

NODE 2.2 - Model Developer's Potency and Modeling Effort

Links from Lower Level:

1) Node 3.1 - Model reusability
2) Node 3.2 - Change management capability
3) Node 3 3 - Software modularity
4) Node 3.5 - Model debugging support/Verification
5) Node 3.8 - Simulation language knowledge/Ease of learning effort required
6) Node 3.10-Model representation correspondence to the real system
7) Node 3.11-Modeling flexibility

Original weights

Row \ Col	1	2	3	4	5	6	7
1	1.000	3.000	2.000	0.250	0.167	0.333	1.000
2	0.333	1.000	2.000	0.250	0.500	0.500	0.333
3	0.500	0.500	1.000	0.250	0.333	1.000	0.500
4	4.000	4.000	4.000	1.000	1.000	3.000	2.000
5	6.000	2.000	3.000	1.000	1.000	3.000	2.000
6	3.000	2.000	1.000	0.333	0.333	1.000	1.000
7	1.000	3.000	2.000	0.500	0.500	1.000	1.000

Normalized Columns

Row \ Col	1	2	3	4	5	6	7	Weights
1	0.063	0.193	0.133	0.069	0.043	0.033	0.127	0.094
2	0.021	0.064	0.133	0.069	0.130	0.051	0.042	0.073
3	0.031	0.032	0.067	0.069	0.086	0.101	0.063	0.064
4	0.252	0.258	0.266	0.279	0.260	0.305	0.255	0.027
5	0.378	0.129	0.200	0.279	0.260	0.305	0.255	0.258
6	0.189	0.129	0.067	0.093	0.086	0.101	0.127	0.113
7	0.063	0.193	0.133	0.139	0.130	0.101	0.127	0.126

Estimation of matrix consistency:

Lambda max:	7.590
Consistency index:	0.098
Consistency ratio:	0.074

AHP Matrix Calculation Set 4

NODE 2.3 - Performance Considerations

Links from Lower Level:

1) Node 3.7 - Execution speed
2) Node 3.9 - Basic memory requirements

Original weights

Row	Col	1	2
	1	1.000	0.125
	2	8.000	1.000

Normalized Columns

Row	Col	1	2	Weights
	1	0.111	0.111	0.111
	2	0.889	0.889	0.889

AHP Matrix Calculation Set 5

NODE 3.1 - Model Reusability

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Col	1	2
Row		
1	1.000	0.143
2	7.000	1.000

Normalized Columns

Col	1	2	Weigths
Row			
1	0.125	0.125	0.125
2	0.875	0.875	0.875

AHP Matrix Calculation Set 6

NODE 3.2 - Change Management Capability

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.167	0.167	0.167
2	0.833	0.833	0.833

AHP Matrix Calculation Set 7

NODE 3.3 - Software Modularity

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.333
2	3.000	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.250	0.250	0.250
2	0.750	0.750	0.750

AHP Matrix Calculation Set 8

NODE 3.4 - Output Provisions

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.125
2	8.000	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.111	0.111	0.111
2	0.888	0.888	0.888

AHP Matrix Calculation Set 9

NODE 3.5 - Model Debugging Support/Verification

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.125
2	8.000	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.111	0.111	0.111
2	0.888	0.888	0.888

AHP Matrix Calculation Set 10

NODE 3.6 - Graphics/User interface capability

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row	Col 1	2
1	1.000	0.167
2	6.000	1.000

Normalized Columns

Row	Col 1	2	Weights
1	0.143	0.143	0.143
2	0.857	0.857	0.857

AHP Matrix Calculation Set 11

NODE 3.7 - Execution Speed

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	6.000
2	0.167	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.857	0.857	0.857
2	0.143	0.143	0.143

AHP Matrix Calculation Set 12

NODE 3.8 - Simulation Language K knowledge/Ease of Learning Effort Required

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	5.000
2	0.200	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.833	0.833	0.833
2	0.167	0.167	0.167

AHP Matrix Calculation Set 13

NODE 3.9 - Basic Memory Requirements

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	7.000
2	0.143	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.875	0.875	0.875
2	0.125	0.125	0.125

AHP Matrix Calculation Set 14

NODE 3.10 - Model Representation Correspondence to the Real System

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.111
2	9.000	1.000

Normalized Columns

Row \ Col	1	2	Weights
1	0.100	0.100	0.100
2	0.900	0.900	0.900

AHP Matrix Calculation Set 15

NODE 3.11 - Modeling Flexibility

Links from Lower Level:

1) Node 4.1 - Traditional, special purpose simulation systems
2) Node 4.2 - OOP simulation systems

Original weights

Row \ Col	1	2
1	1.000	0.143
2	7.000	1.000

Original weights

Row \ Col	1	2	Weights
1	0.125	0.125	0.125
2	0.875	0.875	0.875

Matrix 1

	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11
4.1	.125	.167	.250	.111	.111	.143	.875	.833	.875	.100	.125
4.2	.875	.833	.750	.888	.888	.875	.143	.167	.125	.900	.875

Matrix 2

	2.1	2.2	2.3
3.1	0.049	0.094	
3.2		0.073	
3.3		0.064	
3.4	0.342		
3.5		0.027	
3.6	0.05		
3.7			0.111
3.8		0.258	
3.9			0.889
3.10	0.302	0.113	
3.11	0.255	0.126	

Matrix 3

	1.1
2.1	0.750
2.2	0.189
2.3	0.059

Matrix 4 = (Matrix 1) * (Matrix 2)

	2.1	2.2	2.3
4.1	.113	.284	.875
4.2	.885	.470	.127

Matrix 4 = (Matrix 4) * (Matrix 3)

	1.1
4.1	0.225
4.2	0.775

VITA

Saloua Smaoui

Candidate for the Degree of

Master of Science

Thesis: APPLICATION OF AN OBJECT-ORIENTED PARADIGM TO THE MODELING OF A CONSTANT SPEED, DISCRETELY SPACED, RECIRCULATING CONVEYOR SYSTEM

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Gafsa, Tunisia, September 26, 1967, the daughter of Mr. and Mrs. Mohamed Salah Smaoui.

Education: Graduated from Houssine Bouzaine Primary School, June 1978; Lycee-Mixte Gafsa High School: Baccalaureate, June 1986; Oklahoma State University: Bachelor of Science in Industrial Engineering, May 1990; completed requirements for the Master of Science in Industrial Engineering in July, 1992.

Professional Experience: Teaching Assistance, 1990-1992, School of Industrial Engineering, Oklahoma State University; Research Engineer, summers 1988/1990, Companyie De Phosphates Gafsa, Tunisia.