INVESTIGATION OF AN OBJECT ORIENTED MODELING ENVIRONMENT FOR THE GENERATION OF SIMULATION MODELS

By

TERRENCE GILBERT BEAUMARIAGE

Bachelor of Science
Rochester Institute of Technology
Rochester, New York
May, 1984

Master of Science
Oklahoma State University
Stillwater, Oklahoma
May, 1987

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1990

INVESTIGATION OF AN OBJECT ORIENTED MODELING ENVIRONMENT FOR THE GENERATION OF SIMULATION MODELS

Thesis Approved:

Joe H. Mize
Thesis Advisor

M. P. Terrell

Kenneth E Case

Allen C. Schuermann

C. M. Bacon

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

Although I know that words will truly not be adequate, I am unfortunately limited to them as the means to express my appreciation to all of the individuals with whom I have interacted during my doctoral studies. First and most of all, I would like to thank my major professor, Dr. Joe H. Mize, for all of the time and effort that he invested in my research and education. Dr. Mize has certainly acted as my mentor, "a loyal friend and advisor". Without his guiding influence, I am certain that my graduate career would not have been as rewarding nor would this research activity have been possible. I hope that I can provide similar inspiration to students during my career in academia.

I would also like to thank each of the members of my committee members for their impact on my education and research. Dr. Charles M. Bacon introduced the concepts of artificial intelligence, expert systems, and alternative programming methodologies in a formalized manner and suggested ways to improve upon my original research proposal. Dr. Kenneth E. Case provided an example of what an excellent educator is, a goal that I shall strive for, along with guiding my interests into industrial applications of statistical and quality control concepts. Dr. Allen C.

Schuermann increased my knowledge in the area of real time programming and the internal operation of microcomputers, along with providing advice on the verification of the prototype Object Oriented Modeling system developed in this research. Dr. M. Palmer Terrell provided my first opportunity to work on funded research, got me to finally understand the concepts behind linear programming, and formally introduced multi-criteria decision making, which led to my use of an Analytic Hierarchy Process model as a simulation environment evaluation technique.

As a group, I would like to thank the faculty and staff of the Oklahoma State University (OSU) School of Industrial Engineering and Management (IE&M). The IE&M faculty consistently act as successful, professional role models for their graduate students to strive to emulate. They make the task of having a successful academic department and a challenging graduate program seem simple, when, in fact, it is not. The IE&M staff were a pleasure to interact with, truly were interested in the well-being of the students, and acted effectively as an extension of the faculty.

For assistance in accomplishing this research, I would specifically like to acknowledge the input provided by the individuals who, along with myself, generated the simulation evaluation pairwise weights. Chuda Basnet, Cem Karacal, Dr. Mize, and I spent the better part of three days in early August, 1989 working as a group through the evaluation

model. This involved understanding the model, critiquing the decision decomposition approaches, and providing the final weights used in the evaluation portion of this research. This input was critical because the participants needed to have a strong understanding of the topic of simulation along with both traditional and object oriented programming.

For their direct contributions to this research, I would like to thank those individuals who participated in the many discussions held in the Center for Computer Integrated Manufacturing (CIM Center). Chuda Basnet, Phil Farrington, and Cem Karacal were involved, along with Dr. Mize and myself, in each of these discussions. During various time periods, Dr. Jose Pablo Nuno, Dr. Manjunath Kamath, Dr. Silvanus Udoka, Laura Raiman, and David Pratt also participated in these meetings.

Of particular importance to my graduate studies were the many other graduate students at OSU. The level of camaraderie and constructive competition present among this group was exceptional. By working together on school tasks and socializing together outside the office, life was both more productive and enjoyable.

To this point, I have discussed the individuals having a personal impact on my graduate studies. Another group that must be mentioned and to whom I would like to express my sincere appreciation are those organizations providing

funding during my graduate career. First of all, three years of support were provided from the National Science Foundation (NSF) in the form of an NSF Graduate Fellowship. The AT&T Foundation provided significant financial and equipment support for the CIM Center at OSU, under whose auspices much of this work was performed and through which I received a research assistantship. The IE&M department directly provided support for the first year of graduate study, in the form of research and teaching assistantships. Additional support was received in the form of an Institute of Industrial Engineers Gilbreth Memorial Fellowship, the OSU Rapp Distinguished Graduate Fellowship, and an Alpha Pi Mu Fellowship.

Finally, I would like to acknowledge the support provided by members of my family. My wife, Kim, consistently encouraged my classroom and research efforts and prevented me from being discouraged throughout this time period. She, as an Industrial Engineering student, was a resource from which I could receive professional, as well as emotional, support. Lastly, I want to thank my parents for making me what I am today, for providing me with a strong sense of responsibility and ambition, for tolerating me as an active child and as a rebellious high school student, and for maintaining their faith in my potential. My family continued to believe in my ability even when I had trouble doing so.

TABLE OF CONTENTS

(*) Available in the School of Industrial Engineering and Management Library at Oklahoma State University.

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

The principal focus of this research is to explore the applicability and benefits of an Object Oriented Programming (OOP) environment for simulation model development. Reusable simulation objects within an OOP environment along with the procedures and software to guide their use will be developed and used to build a simulation model (as proof of concept). Appropriate measures for the evaluation of the effectiveness of this approach will be designed and used to compare this new development with traditional simulation approaches. Briefly, this research shall provide the conceptual development of an Object Oriented Modeling (OOM) environment, implement a usable prototype of this environment, and use the proposed effectiveness measures to compare the new environment with previously available approaches to simulation.

This research topic was chosen to further develop the author's skills within the interest areas of simulation, computer applications in manufacturing system development, and artificial intelligence. The literature applicable to simulation methodologies shows interest in several different directions including animation, menu driven model

development, and simulation modeling within an object oriented environment, all with the intention of improving the capabilities (explainability, ease of use, etc.) of simulation software. At this point the research on OOM described in the literature is fragmented and of a preliminary nature, but the number of articles and different topics and approaches discussed indicate that this is an area of great interest for simulation methodology development, both for improving the underlying paradigm used in simulation model development and for achieving advanced capabilities such as real time animation, interactive simulation, and graphical (programming free) model development. 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 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 V 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

STATEMENT OF THE PROBLEM

In the past, development of simulation models was an extremely time consuming activity. The modeler had to transfer the processing elements of simulation (linked lists, etc.) from algorithmic descriptions into a general purpose computer language. This involved a large amount of effort in the form of redundant software development and unique software design on the part of the simulation modeler. Once implemented and applied to the project of interest, these models (and the modeling effort represented) were seldom reusable. As the field developed further, simulation languages such as GASP IV, having generic code for timing, statistics collection, etc., were developed and made commercially available. Simulation models were written in general purpose computer languages and used generic functions and subroutines available in the specialized language to perform common simulation operations. The bulk of the simulation model development effort was spent in writing the code specific to the model of the system of interest. Although the modeling effort was reduced, the resulting models remained a single use effort.

More recent developments in the field of simulation are

higher level languages specialized for simulation and implementing a process orientation. These languages provide for a further decrease in the model development effort by providing standardized abstract modeling elements which are used as building blocks for models. These building blocks are specialized for a specific model by supplying parameter values describing the element's functions and activities within the model. For relatively straight forward modeling situations (those systems whose elements' activities coincide with the modeling elements available), this approach results in a much reduced level of modeling effort. The model building blocks are reusable and easily understood. These languages, with the addition of model building preprocessing software (graphical model definition) and model animation capabilities (TESS and CINEMA being well known examples), represent the current state of the art in simulation technology.

Along with the technical developments in simulation have come changes in the way in which the technique of simulation is used within business. In the past, simulation was used as a planning and diagnostic aid in manufacturing system design for relatively large, expensive projects. Part of the reason for this limited use was the requirement that a modeler have a significant level of expertise in the use of computers and general computer languages. Because of this, simulation modeling was an expensive and time consuming activity and individuals capable of developing

models were in short supply. As the computer skills and experience of manufacturing engineers improved, simulation models became a more common experimental tool used in smaller manufacturing projects. In addition to use as 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.

As simulation applications have changed, so has the nature of the factory floor. Where once a production system was made up of a relatively static design, current manufacturing systems are dynamic, constantly changing organisms containing a large number of detailed interactions. In addition, manufacturing systems in the future will be required to be reconfigurable to be responsive to dynamic changes in the environment. The effects of these trends on simulation modeling are the requirements that a simulation model be easily updated and highly modular (changes to a model should be localized).

In brief, the requirements of a simulation environment for advanced manufacturing systems are: high level of software reusability, software modularity, the ability to implement large, detailed models, low level of abstraction (modeling elements should relate to system elements in design form and simulation function), a graphical

interactive development environment, and ease of analysis of results.

The purpose of this research into Object Oriented Modeling is to show the applicability of OOP languages and concepts to simulation modeling and to measure and demonstrate the benefits derived from the use of OOP in the design and implementation of an OOM environment. Achievement of a prototype OOM environment should allow the researcher to illustrate the ability of the approach to fulfill many of the still unsatisfied needs of advanced manufacturing system simulation modeling.

CHAPTER III

BACKGROUND OF THE STUDY

This chapter presents a review of the research being performed in the areas of simulation methodology and environments. After a brief introduction, a general discussion on the research areas which hold promise for improving simulation methodology is presented. Next, sections describing specific approaches taken in menu driven simulation and simulation environments are described. Finally, the object oriented framework and diverse areas of applicability are discussed, along with the applicability of object oriented concepts to simulation methodology development.

Introduction

Models are descriptions of systems [Pritsker (1986)]. Models may be physical, mathematical, or graphical in nature and are primarily useful in describing, designing, and analyzing systems. Humans develop models to allow better communication of a system, to understand complex systems under study, to conceptualize and analyze systems which do not yet exist, etc. A simulation model is an abstract, mathematical model of a system of interest which is

dynamically "exercised" through the use of a computer [Zeigler (1976)]. Therefore, in order to perform a simulation modeling experiment, an analyst must be able to translate a representation of a system (a model) into information that a computer is able to understand and manipulate [Zeigler (1976)]. The main goal of simulation modeling is the development of a model that represents the real system correctly and with the appropriate amount detail to allow design and analysis experimental results to be extrapolated from the simulation model to the real system. Of course, related goals are present and are of great importance.

Given that we are in a world in which resources and time are finite quantities, a major goal in 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 viewpoint, we desire the ability to implement simulation models which satisfy current and future needs with a minimum cost. 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 the objective being the improvement of simulation modeling capabilities.

Simulation Methodology Research

As briefly stated in Chapter II, the research

developments in the area of simulation have gone through an extended evolutionary process. Early simulation modeling was performed through the use of general purpose computer languages. This method proved effective and illustrated the value of simulation modeling to the public and private sectors. Unfortunately, models were so expensive and difficult to design and maintain that simulation was a technique reserved for use within large scale, expensive projects. Rather than discard simulation, research into improving the systems available for simulation modeling (simulation languages, simulation environments, etc.) was undertaken. The research was driven from two directions: needs (the needs of simulation analysts, the complexity of new systems [manufacturing, vehicles, etc.], the limited resources available, etc.) and abilities (developments in the areas of computers, software, etc.).

The development of the most commonly available simulation languages (SLAM II [Pritsker and Associates, Inc., 1988], SIMAN [Systems Modeling Corp., 1988], etc.) was driven primarily by the need of simulation analysts for an easier, more efficient method of model translation and representation. As justification for this statement, consider that these languages, which have been available for roughly five to twenty years (in one form or another), were developed using standard hardware (time-sharing, mainframe computers) and software (Assembly code, Fortran) available for a long period of time (within the time frame of the

existence of computers). Also note, specifically considering the "SLAM family of languages" [Pritsker, 1986], that this development occurred in an evolutionary fashion within relatively constant hardware and software environments (time-sharing, batch simulation runs, Fortran language).

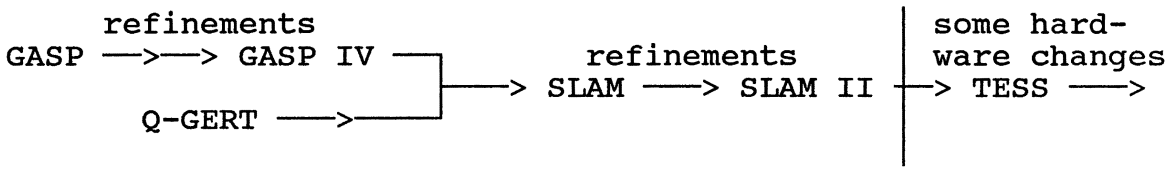


Figure 1. Incremental Improvement to a Simulation Language (in this case the "SLAM family")

This discussion is applicable until the late 1970's and early 1980's when the effects of the microcomputer and personal computer era were felt. Researchers in the hardware area worked to greatly extend the capabilities of computers and peripherals in facets directly impacting users. It was this hardware (and resulting software) research and development activity that really made the full capabilities of computers broadly available to the current and potential users of that period. These new abilities allow system software developers to conceive of and deliver greater functionality and ease of use which leads to the more productive application of computers. Consider the

following list of recent computer oriented topics discussed in the literature: graphics, animation, artificial intelligence concepts (sudden resurgence of interest), and software development environment concepts. A large number of literature sources discussing the application and benefits of these approaches to simulation methodology is available.

Being so closely related, it is appropriate to consider the impact of graphics and animation on simulation together. Wyvill (1985) presents the three basic ways in which graphics/animation can positively impact simulation:

" - To enhance the simulation results

- To facilitate the debugging and production of simulation programs

- To provide an interactive dialogue with a running simulation"

Systems implementing some or all of these improvements include, but are not limited to, ANDES and SIMSEA [Wyvill (1985)], TESS [Pritsker and Associates, Inc., 1988], Cinema [Systems Modeling Corp., 1988], and SIMFACTORY [CACI, Inc., 1988]. SIMSEA uses graphics to enhance the results of simulation by providing animated output of simulation results in the form of simple stick figures or simple icons. ANDES uses graphics capabilities to aid in debugging and program development. The underlying mechanism of the simulation along with the simulation results have been animated. The analyst is able to observe an executing

simulation model from different perspectives which include: the status of an entire model and the status of a specific model element. TESS (an acronym for The Extended Simulation Support System), a further improvement to the "SLAM family," "supports the model entry, simulation, statistical analysis, and result presentation tasks required in a simulation project" [Standridge, et al. (1987)]. TESS provides the analyst with the ability to interactively specify a model in the SLAM II language through graphics screens supporting icons and menu driven input. Simulation results (concurrent or playback mode) can be displayed through an animated model of the system. A newly released product, SLAMSYSTEM [Pritsker and Associates, Inc., 1988], combines the ability to build models through an interactive graphical model builder with the ability to produce graphical animation without programming as parts of an entire simulation environment. Among other features, the system has the ability to graphically display the simulation results from alternative models on a single graph. Cinema, associated with the SIMAN simulation language, [Kilgore and Healy (1987)] and SIMFACTORY from CACI provide capabilities which are roughly similar to those found in TESS.

The application of artificial intelligence concepts within simulation modeling has been discussed by Sathi et al. (1987), Khoshnevis et al. (1988), Khoshnevis and Austin (1987), Khoshnevis and Chen (1987) and (1986), Ford et al. (1987), and Murray and Sheppard (1987). The simulation

technology being developed at Carnegie Group, Inc. [Sathi et al. (1987)] "augments simulation expertise by infusing Artificial Intelligence techniques into the Simulation Life Cycle. It is a problem solving shell which uses simulation, statistical expertise, and domain specific knowledge" to assist in the solution of manufacturing system problems. This development employs several different embedded expert systems to assist the analyst in the most complicated and time consuming tasks (model building, model execution, and model analysis). The software systems described by Khoshnevis and various co-authors apply rule based and structured knowledge approaches (expert system and knowledge based system technology) to provide assistance to the simulation modeler in the development of systems dynamics and discrete event simulation models. These systems are structured in the form of preprocessor shells with an established simulation language (DYNAMO, SLAM II, and SIMNET) as the kernal. NATSIM is composed of a natural language processing system, a system analyzer, and a program generator. The natural language processing system, called PHRAN, takes a natural language description of a system as input, analyzes this input through a comparison to standard patterns in an associated knowledge base and produces a structured form of the input for use by the system analyzer (SA). "The SA uses an extensive knowledge base of system dynamics . . . to generate a language independent complete model description" [Khoshnevis et al. (1988)]. This

description is processed by a DYNAMO Program Generator to produce a model translated into the DYNAMO language. EZSIM, a discrete simulation modeling tool, "uses a combination of graphics and [a] menu driven user interface" along with a language specific knowledge base to act as a front end to an existing simulation language (SIMNET or SLAM in the current version) [Khoshnevis et al. (1988)]. The software systems discussed by Ford et al. (1987) and Murray and Sheppard (1987) have basically the same capabilities and structure as that described for the systems developed by Khoshnevis.

Reilly et al. (1985) have adopted Henrikson's (1983) conceptual framework for a simulation environment (discussed in detail later) and attempted to complement it through "emphasizing the role of AI techniques" within the architecture of the system. These AI techniques are knowledge based approaches implemented in LISP and OPS5 (a rule based expert system builder written in LISP) to assist the user in model building, model execution, and analysis of the simulation statistical results.

Another area of computer research impacting simulation methodology is the recent implementation and use of software development environments. A software development environment can be defined as a collection of tools that are well-integrated and interact synergistically in support of all phases of software development [Reilly et al. (1985)]. First developed for use on artificial intelligence platforms

(commonly known as LISP machines), these environments provide the user with immediate access to features such as a language supporting text editor (special features of the text editor may enhance use with a particular language), an interpreter (for quick checking of a piece of code), debugging aids linking the interpreter to an editor, and a compiler (for speed of operation on completed software). Two examples of these with which the author is familiar are the Golden Common LISP (GCLISP) [Gold Hill Computers (1987)] and Smalltalk/V environments [Digitalk, Inc. (1986)]. The editor in GCLISP keeps track of open parentheses (a major syntatic feature of LISP) to assist the programmer during the development of complex code. When the user completes a function, the GCLISP interpreter is used to check the syntax and to quickly verify the operation of the code. There is also on line help embedded within the environment. Smalltalk/V provides a window based approach to a software development environment. The user is able to access and modify all code available in the system, to add new code to the system, and to interactively test and debug new code. Smalltalk allows the user to suspend interaction with one activity, perform a task associated with another feature in the system, and resume the previously suspended activity. When one contrasts this with the previous software development activities: edit, compile, link, and repeat to correct errors, all performed in a non-integrated environment, the productivity benefits which can be gained

are apparent. The application of software environment concepts to the development of a complete simulation modeling (which is basically advanced software development) system is the direction in which most simulation research is leading (in an incremental fashion).

The key factor linking simulation methodology research together is the desire to improve the productivity and efficiency of the human modeler. This improvement can be considered to be felt over the long term in several areas. Most obvious is to facilitate an increase in the speed with which a simulation model for a specific system is developed, validated and verified, and put to use in the modeling study. Less obvious, but potentially more important are the results of simulation research which allow modelers to conceptualize and implement models of systems or elements of systems which, previously, had been either infeasible to model (due to complexity, expense, lack of understanding, etc.) or not even considered for model implementation.

Menu Driven Simulation Generators

As mentioned previously, one approach to simulation research has been to pursue the application of artificial intelligence, knowledge based systems, etc. to simulation software. This approach can result in the implementation of a complex environment (to be considered in the next section) or a system of lesser sophistication. The latter, which will be discussed here and shall be referred to as menu

driven simulation generation, is to develop a menu and/or graphics driven preprocessor supported through some type of knowledge base specific to a target simulation language to assist the user in the building of a simulation model. Endesfelder and Tempelmeier (1987) discuss their implementation of such a system for the generation of SIMAN simulation models. Called the SIMAN Module Processor (SMP), the system has a knowledge base composed of predefined, standardized modules of SIMAN code. "The SMP inputs predefined, filed modules, interprets them in relation to interactively specified data (e.g. problem-specific parameters) and produces a syntactically correct SIMAN simulation model" [Endesfelder and Tempelmeier (1987)]. The filed modules used by the SMP must be defined in a special syntax which is matched to the structure of SIMAN language elements and statements. "The SMP parses all program lines contained in a module [supplied from the knowledge base] and recognizes from the first small letter that an input is required. Capitals, numbers, and the special characters ',', ':', and ';' are adapted unchanged." The SMP allows the user to select a module for incorporation into a model from the modules available in the current library file. The SMP reads the module line by line and requests needed data (signaled by lower case text) by prompting the user. The user can select multiple modules and when the model is completed, the SIMAN simulation files are produced automatically. Another system having similar

characteristics, but specifically oriented to the modeling of flexible manufacturing systems has been described by Haddock (1988).

EZSIM, a system described by Khoshnevis and Chen (1987), is a simulation generator having SLAM (or SIMNET) as the target language. The system is written in Golden Common LISP and is composed of three principle segments: user interface, expert system, and program generator. The user interface uses a combination of menus and a natural language interface. An initial menu provides a choice of 11 nodes to include within a model. The user specifies the nodes which will be used in the model and the nodes which follow (entity flow direction) them. Upon completion of this input, the expert system analyzes the nodes chosen for the model through the use of its stored data and requests, either through a menu format or through a natural language interface, that the user provide missing information. Once the current model information is developed to the point that it passes the testing of the expert system, the program generator segment of EZSIM is initiated. This segment produces the SLAM source code file. Khoshnevis and Austin (1987) have also developed a similar system which analyzes user input for the generation of continuous simulation models in the DYNAMO language.

Oren and Aytac (1985) describe their implementation of a simulation generator titled MAGEST (Modeling Advisor for

GEST [General System Theory Implementor] programs). MAGEST has the capability to use two types of knowledge: 1) knowledge on the GEST language and 2) incremental knowledge obtained from user programs, which it uses to perform the following functions:

"- To assist the user to specify
  - Models,
  - Parameter sets, and
  - Experimentations

- To perform checks for
  - completeness,
  - correctness, and
  - compatibility, and

- To certify GEST programs which pass the above checks." [Oren and Aytac (1985)]

MAGEST is composed of an executive control routine, a model template generator, a certification and advisor program, and the GEST translator. The model template generator generates a template of the structure and keywords for a GEST program and additional information is added through the use of the MAGEST certification and advisor program. The result is passed to the GEST translator which produces the necessary statements in the SIMSCRIPT II.5 language for execution of the simulation model.

Simulation Environments

Also mentioned in the introductory section to this chapter was the concept of a software development environment. A simulation environment is defined as "a collection of tools that are well-integrated and interacting

synergistically in support of all phases of the modeling process" [Reilly et al. (1985)]. Henrikson (1983) presented his view of an integrated simulation environment in an article whose purpose was "to identify significant improvements that will be made in simulation software in the next 10 years." Henrikson states that "most of the current research in programming systems is being conducted in other problem contexts" and, therefore, simulationists "must look outside the discipline of simulation for most of our examples" for trends and features to implement in simulation systems. Henrikson proposes the architecture for a simulation environment that is illustrated in Figure 2. The software components of the proposed environment include the following:

1) Model Editor
2) Input Preparation Subsystem - distribution fitting, etc.
3) Statistics Collection Definition Facility - used to define how and what observations will be collected
4) Experimental Design Facility
5) Output Definition Facility
6) Program Editor - syntax directed editor for simulation source program
7) Compiler
8) Run-Time Support - interactive debugging, real time simulation monitoring

These components will operate within an integrated environment by interacting with the user at separate points through specialized formats or languages and through accessing complete data stored in a comprehensive knowledge base. Although such a system is not yet within reach, incremental research continues in all areas and will make this environment feasible at some time in the future.

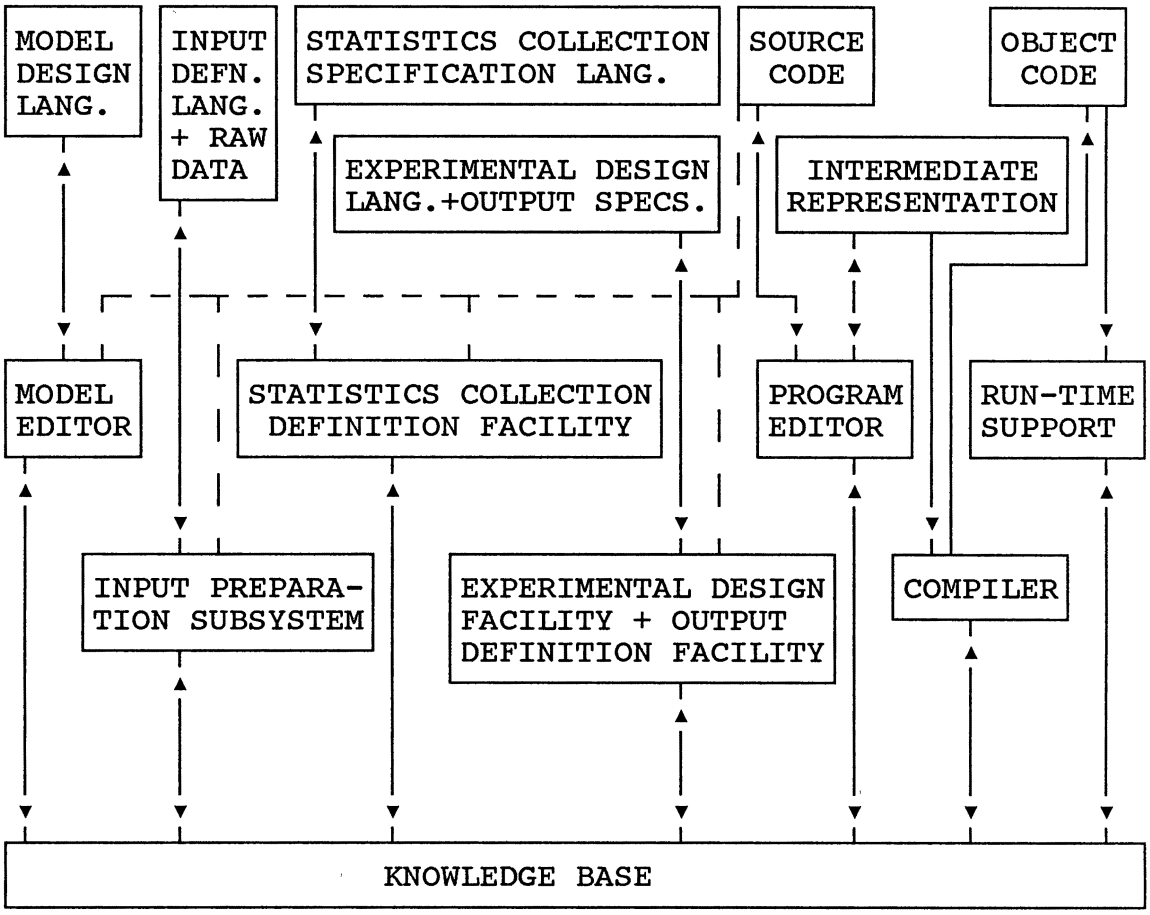


Figure 2. Henrikson's Proposed Simulation Environment [Henrikson (1983)]

Reilly et al. (1985) adopt and extend Henrikson's proposed environment by suggesting potential architectures to be implemented over a distributed processing architecture. They distill the many segments of the proposed environment down to four primary components: builder, model executor, record keeper, and results analyzer. From an initial implementation in a similar format on AT&T 3B2 computers, the researchers expect the

system to evolve into the form shown in Figure 3. The software systems shown within the computer elements of Figure 3 are: S - a graphics and data analysis system, GGC - multiple world view simulation package, SAS - statistical analysis software, LISP and PROLOG - symbolic processing languages, ASP - fully interactive language compiler, PSL/PSA - knowledge base development language.

An artificial intelligence approach to a simulation environment described by Sathi et al. (1987) appears to be the most complete environment implemented and operating to date. The system "is a problem solving shell which uses

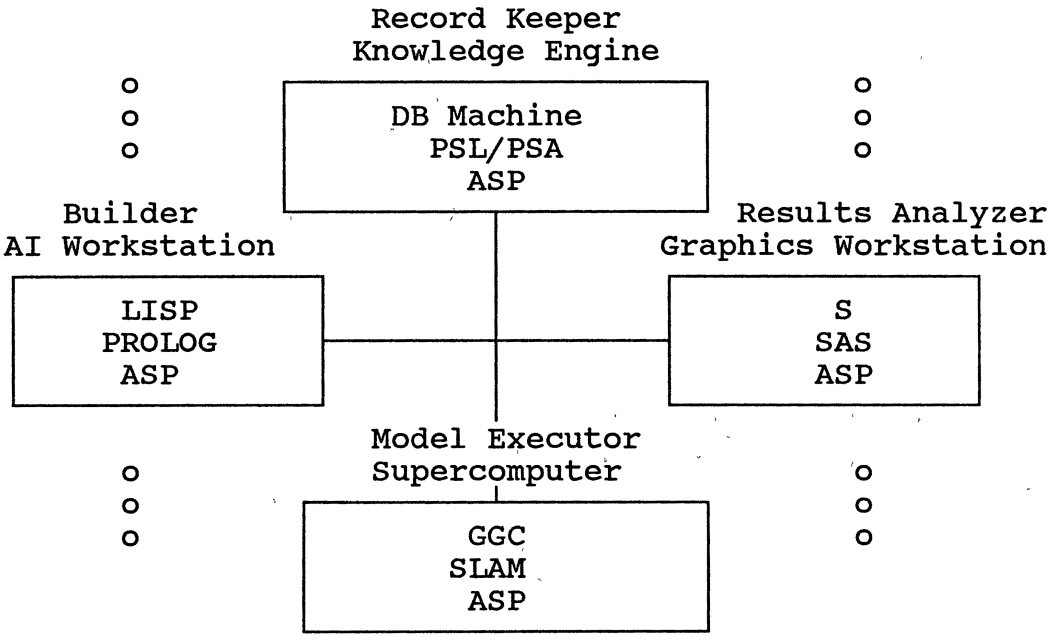


Figure 3. Future Architecture for a Simulation Environment, Proposed by Reilly et al. (1985), Multiple Versions of the Builder and Analyzer Implied

simulation, statistical expertise, and domain-specific knowledge to solve real world problems." The control architecture of the system has three major components which consist of: dynamic planner, embedded experts, and a suggested plan display. The dynamic planner functions to produce (through a rule-based generation process) and maintain a suggested plan for each model used for solving the simulation problem under consideration. The planner produces the plan at the beginning of model specification and dynamically updates the plan as changes are made or various steps in the plan are accomplished. The embedded experts include three basic systems: model building expert, model execution expert, and model analysis expert. These experts, which control the modeling activities, are hierarchically organized and communicate with each other to facilitate the performance of necessary actions and transmit the current model status. Figure 4 illustrates the structure of the experts and the responsibilities of each one. The suggested plan display provides the user with the ability to drive the simulation development procedure by choosing from the menu of suggested activities. A unique feature found in this environment is the automated analysis of simulation model results through the use of a knowledge based expert. This expert uses goals and constraints supplied by the user to evaluate the large quantity of data from a simulation and provide the user with a summary of the model's performance.

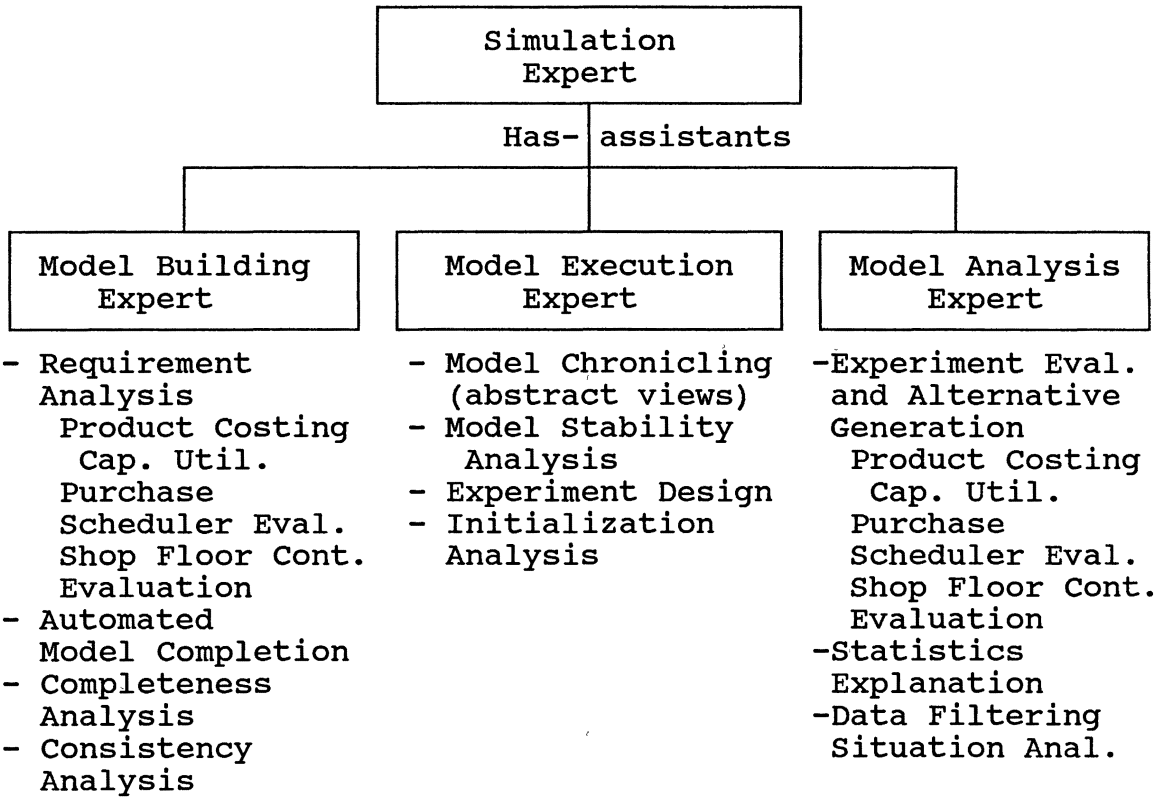


Figure 4. Expert Architecture for a Simulation Environment [Sathi et al. (1987)]

Object Oriented Programming:

A Brief Introduction

Due to the fact that OOP is such a new area in software development, it is appropriate to present a concise introduction to the major features included within the OOP paradigm. The principal idea associated with OOP is that all items (e.g., variables) in the system are treated as "objects." An object is a class or instance of a class and a particular class may have multiple different instances operating at any one time. The definition for a class

defines the data which may be stored within the class, the manner in which the data is stored, and the procedures which may perform operations on the data. "...the underlying notion of the object is to organize and store pieces of information relating to a single concept into a single location" [Shannon (1987)]. Smalltalk, 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, late binding, and inheritance.

Encapsulation means that an object's data and procedures are enclosed within a tight boundary, one which cannot be broken by other objects. An object may have within it several data storage locations. The values stored are only directly accessible by the procedures that have been defined as part of the object's class structure. All other access to this data (by other objects) is forced to occur through channels provided by procedures attached to the object itself.

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).

Binding refers to the process in which a procedure and the data on which it is to operate are related. 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 or late binding delays the binding process until the software is actually running. When an object receives a message (a procedure call), the OOP system searches the object's class to find the method to perform. This use of late binding gives OOP a great deal of flexibility in several ways. First, it is possible for the data type of a variable to change during run time. Another consideration is that different classes can have the same named procedures with different code found in each object. For example, the procedures to access the value of an element of an array and the character in a position of a string have the same name and very different software implementations, one for class Array and another for class String. Finally, the majority of classes defined in the OOP environment are independent of data type. An instance of the Array class can store many different types of objects at the same time.

The fourth feature, inheritance, provides for software

reusability. OOP classes are defined in a hierarchical tree structure. Because of inheritance, each class inherits the methods and data storage structure of all of its superclasses. Code which is identical for multiple classes in the same subtree is written and tested once and stored in one position in the tree. Also, consider that when it becomes time to change a particular method, it is only necessary to change the method once and all uses of the method will reflect the change.

Let us consider how these four features provide the benefits claimed for OOP. First, understandability is achieved because each object represents one concept and all data and methods which are part of the object function to implement characteristics of the concept (the object). A software object is the implementation of one complete concept and is, therefore, easier to grasp and implement. Modifiability is achieved because an object has all of the data and procedures associated with it tightly grouped together in one unit. When it becomes necessary or desirable to alter the data structure of an object, there is no need to search through all of the methods in the system because all methods which directly access the data structure and are designed to work with the specific data structure are defined as part of the object.

Reusability of code is achieved in two ways. The first way is through inheritance of code from superclasses to

subclasses. The second way is through the ability to include objects as components in further software development or as a building block in the definition of another class. The concept of a Software-IC, introduced by Cox (1986), illustrates this reusability in a conceptually simple manner. A Software-IC "is a package of programming effort that is independent of the specific job at hand and highly reusable in future jobs." "Programmers no longer build entire programs from raw materials, the bare statements and expressions of a programming language. Instead they produce reusable software components by assembling components of other programmers. These components are called Software-IC's to emphasize their similarity with the integrated silicon chip" [Cox (1986)].

Application of Object Oriented Concepts to Modeling

The single most important benefit which will be gained from the development of OOM is the ability for manufacturing people to think of modeling in terms of the objects to be modeled and their interactions. "Manufacturing-related people think of systems in terms of parts, machines or 'objects'; programming people think in terms of 'programs', 'data', etc." [Adiga, 1986]. "Conventional approaches to discrete simulation allow the developer a procedural level of modularity" while "Object oriented methodologies achieve an object level of modularity" [Ghaznavi-Collins and Thelen

(1988)]. Adelsberger et al. (1986) state:

> The philosophy of object-oriented programming is a simple one, and directly supports the simulation problem solving approach, especially for systems that deal with the explicit passage of time and/or changes of objects in time. This can be summarized as follows:
>
> (1) The user first creates or defines objects that correspond to real world objects, and represent modular components of the real world.
> (2) The behavior of the simulation model's objects describe the behavior of the real world objects and how these objects will behave/perform in response to various inputs.
> (3) Objects act on each other by passing messages describing both functional and relational actions. Messages passed between objects are carriers for all interaction between objects.
>
> ...The object oriented approach is especially valuable in that it provides a close correspondence between simulated objects and real world objects. ...a complex hierarchy of objects with inherited properties and behavior rivaling real world situations may be modeled.

Briefly, the development of a basic OOM system involves the programming of classes to represent simulation processing objects (which perform tasks to make the simulation run, i.e. time advance, next event triggering), simulation element objects (which provide system element specific event codes and element data storage), and simulation entity objects (which represent the routings and other data on items to be processed) [adapted from Nyen (1987)].

The literature relating research on the application of the object concept to simulation modeling is composed of two distinct classes. The first of these classes describes what shall be referred to as an OOM-like approach to simulation, while the second class consists of actual OOM systems. This

OOM-like approach to modeling is characterized by the development of a library of submodels which may be mildly altered and reused in multiple simulation models [Terrell and Bussey (1973), Terrell et al. (1975), Terrell and Chen (1977a, 1977b), Higdon (1988), Gordon et al. (1987), and Schroer and Tseng (1987)]. This approach was actually taken somewhat farther in several of the previously described menu driven modeling approaches in which standard modules (from a library) were altered by software rather than by the user in the process of model building. By grouping a set of modeling statements together to develop a high level component, we get the ability to treat this software component as an object (not quite equivalent to the OOP concept) and perform modeling with these modules from a higher level. Higdon (1988) describes this approach used in practice for the modeling of conveyors, AS/RS, and AGV's within the GPSS simulation language. Of course, this procedure does not allow the user to achieve some of the features of the OOM procedure (specifically inheritance or encapsulation and the associated benefits). Schroer and Tseng (1987) describe their implementation of three simulation modules in the GPSS language. The three modules include an assembly station segment, a manufacturing cell segment, and an inventory transfer segment. These modules are made specific by the assignment of parameters through matrix values and combined to form complete system models.

True OOM implementations having a range of features

have been described in a number of articles. Knapp (1987) describes a system called SimTalk, the Smalltalk Simulation Environment which "adds queueing support, statistics gathering, simulation oriented graphics, and an interactive user interface." Objects are simulated through the use of concurrent processes which have timing controlled through the application of semaphore operations. SimTalk, a class defined in the OOM environment, contains a simulated clock, a time queue (for time synchronization of multiple processes), and controls creating, suspending, resuming, and terminating processes. In order to model objects in this system, the user is required to define a subclass of the class SimTalkObject. The simulated activities of an object must be defined through a single method, "actions," which executes the appropriate event code when triggered by using a case structure. Bezivin (1987) describes another system named SimTalk which supports similar features and processes (the use of concurrent processes and semaphore sychronization operations) in distributed simulation environments by applying the TimeLock algorithm.

Researchers at Texas A&M University [Adelsberger et al. (1987)] describe the features available in the simulation environment under development. These features include:

- programming free object creation
- interactive system operation
- rich run time support having displays, experimental designs and statistical displays
- goal directed simulation
- graphic display during model building and simulation

The major segments of the environment are shown in Figure 5 and are listed as follows:

- Graphics Drivers
- Data Base Editor
- Intelligent Assistant
  - Menu driven
  - Graphics interface
  - System driven Natural Language Dialogue
  - Template Interface with defaults
  - Specification language input
  - Knowledge (Rule) based interpreter
  - Configuration Management
  - Knowledge Acquisition interface
- Conflict resolution and diagnostics
  - validation/consistency
  - own rule based database
- Run time Monitor
- Statistical Packages
- Goal Driven Experimental Design Driver
- Output Processing for Post Processing
- Interactive Help Environment
- Validation of Experimental Results
  [Adelsberger et al. (1986)]

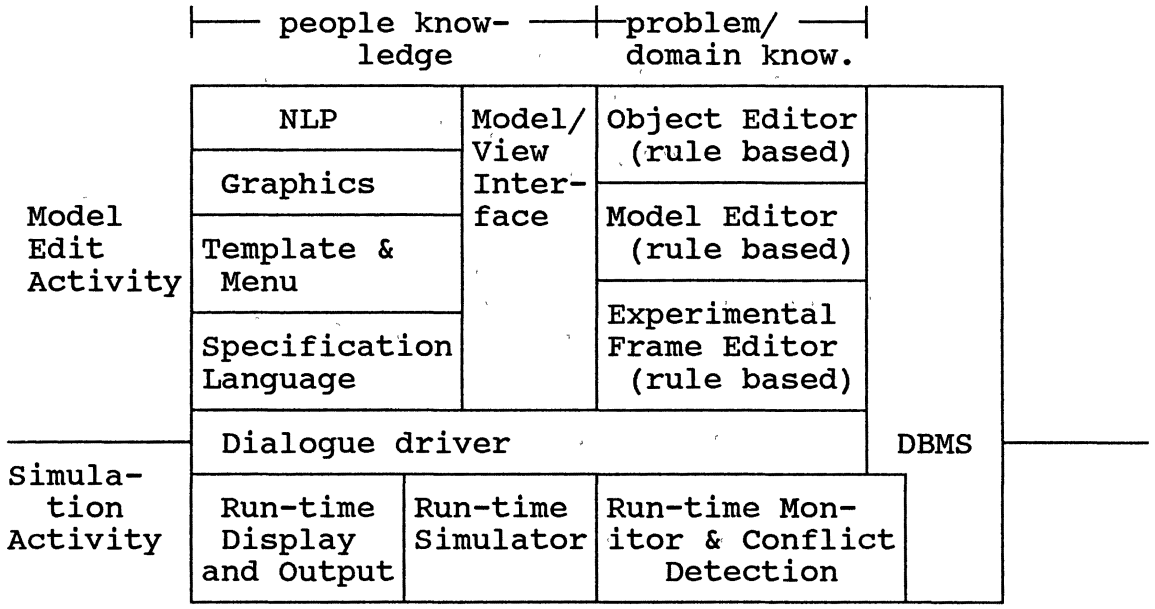


Figure 5. A Suggested Architecture for an Object Oriented Simulation Environment [Adelsberger et al. (1986)]

An application written in Smalltalk-80 [Ulgen and Thomasma (1987) and Thomasma and Ulgen (1987)] in OOM has been developed into a high level graphically supported simulation system. The system consists of the classes shown in the hierarchy in Figure 6.

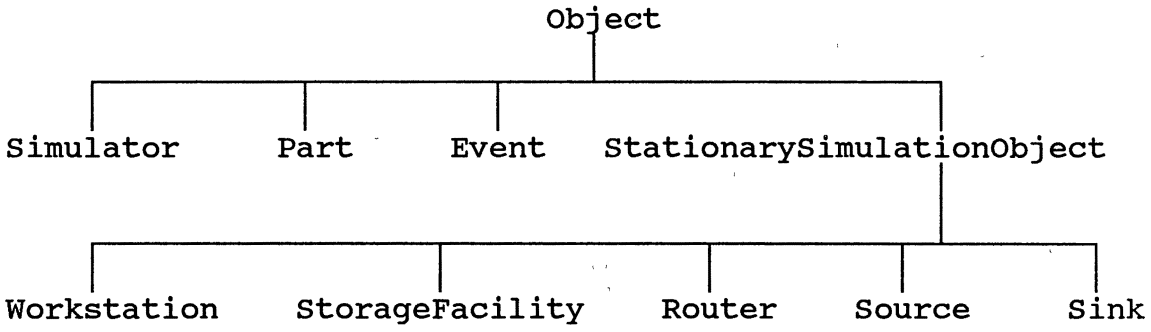


Figure 6. Simulation Classes for Ulgen and Thomasma's (1987) OOM System

The descriptions of the classes as provided by Ulgen and Thomasma (1987) are as follows:

> Class Simulator schedules the events of the simulation, initializes the simulation time, sets the speed of the simulation, and may produce the trace of the simulation. Class Event simply associates a time with something to be done. The remaining classes in the framework are designed to represent the real manufacturing system objects. Class Source represents the source point for parts in the system. Class Sink represents the point where the parts leave the system. Class Workstation represents processors in the system including machines, robots, servers, etc. Class StorageFacility describes objects such as buffer storages, conveyors, etc. Class Router represents part diverters and points where routing decisions are made. Finally, class Part represents workpieces in the system.

The user builds simulation models by interactively specifying parameter values for instances of the classes and linking these instances together and then running the simulation. The Workstations are prompted to begin an event method through a case structured "doit" method, similar to the previously described "actions" method used by Knapp (1987). In contrast to Knapp, Ulgen and Thomasma use a centrally controlled time advance procedure more closely akin to the typical approach used in traditional languages.

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)].

Simulation Evaluation Strategies

A limited number of references dealing with evaluation methods of simulation languages were found. Schriber (1987) provides a listing of desirable simulation software features including:

1) Model Input Flexibility
   a) Textual Definition
   b) Graphics Definition
   c) Digitizing
   d) CAD Interfacing
2) Supportive Syntax
3) Modularity
4) Modeling Flexibility
5) Modeling Conciseness
6) Macro Capability and Hierarchical Modeling

7) Material Handling Modules
8) Standard Statistics Generation
9) Data Analysis
10) Animation
11) Interactive Model Debugging
12) Micro/Mainframe Capability
13) Vendor Support
14) Reasonable Cost
15) Education [Schriber (1987)]

This list provides a good starting point from which to develop absolute or relative measures with which to compare simulation languages and environments.

Grant and Weiner (1987) provide a description of high level factors used to evaluate graphically animated simulation systems. These factors are specifically oriented to the graphics/animation capabilities of the systems considered. Wallace (1987) describes the development of a simulation model complexity measure called the control and transformation metric. This metric is concerned with the complexity of a specific model in comparison to another model within a particular world view. The metric is able to measure the complexity of a given model developed in different world views.

Banks and Carson (1984) provide an evaluation of five different modeling systems using fourteen features having a yes/no or low/medium/high scale. Some of the features included in the evaluation are: ease of learning, ease of conceptualizing a problem, and computer runtime. This provides a good basis for comparing environments on intangible characteristics. The evaluation table is

reproduced in Table 1 on page 37.

Approach in this Research

The OOM system which will be developed will have features highly similar to the basic aspects of the system implemented by Ulgen and Thomasma. One major difference is that event code execution will not be triggered from a case structured method. Rather, a general approach to directly linking simulation element objects will be used. In addition, hierarchically related object models for different equipment will be developed, in contrast to the general, abstract objects implemented by Ulgen and Thomasma.

Summary

This chapter has reviewed the many activities currently being undertaken in simulation methodology research. In addition, evaluation strategies proposed in the literature have been presented. Finally, the approach which will be taken in this research has been related to these items found in the literature. The next chapter presents specific goals and objectives for the development and evaluation of an Object Oriented Modeling system.

TABLE 1

COMPARISON OF LANGUAGES FOR DISCRETE EVENT SIMULATION [BANKS AND CARSON (1984)]

Criteria	Language				
	FORTRAN	GASP	SIMSCRIPT II.5	GPSS V	SLAM
Ease of learning	Good	Good	Good	Exc.	Exc.
Ease of conceptualizing a problem	Poor	Fair	Good	Exc.[a]	Exc.[a]
Systems oriented toward	None[b]	All	All	Queueing	All
Modeling approach					
Event-scheduling	No[b]	Yes	Yes	No	Yes
Process-interaction	No[b]	No	Yes	Yes	Yes
Continuous	No[b]	Yes	Yes	No	Yes
Support					
Random sampling built in	No[c]	Yes	Yes	No[d]	Yes
Statistics-gathering capability	Poor	Exc.	Exc.	Good	Exc.
List-processing capability	Poor	Good	Exc.	Fair[d]	Good
Ease of getting standard report	Poor	Exc.	Fair	Exc.	Exc.
Ease of designing special report	Fair	Good	Exc.	Poor[d]	Good
Debugging aids	Fair	Good	Exc.	Fair[d]	Good
Computer runtime	Exc.[e]	Good	Good	Poor[d]	Good
Documentation for learning language and for reference	V.Good	V.Good	Fair	V.Good	V.Good
Self-documenting code	Poor	Good	Good	Exc.	Good
Cost	Low[f]	Low	High	Low(GPSS/H,high)	Med.

[a] For queueing models, the block diagram (network) conceptualization is excellent.
[b] FORTRAN is not oriented toward system simulation. The programmer develops any desired orientation and takes any desired modeling approach.
[c] Several scientific subroutine libraries (e.g., IMSL) have FORTRAN routines for random variate generation.
[d] GPSS/H is much improved over GPSS V in these respects.
[e] FORTRAN will be fast assuming that the model is programmed in the most efficient manner.
[f] Usually available at most computer installations.

CHAPTER IV

GOALS, OBJECTIVES, AND ASSUMPTIONS OF THE RESEARCH

The overall goal of the research is to investigate an Object Oriented Modeling environment through the development of OOP classes and procedures for their use which result in a simulation environment that can be shown superior to currently available simulation methodology. To achieve this goal, the following objectives are proposed:

1. Object class development. A hierarchical organization of classes necessary for system simulation will be developed.

   The major functional classes within the OOM environment must be determined and described. Once the functions of these objects have been defined, the class hierarchy can be planned and implemented to take advantage of inheritance. Broad classes are (1) simulation processing objects, which function to accomplish the scheduling and initiation of events, collection of certain statistics (those not internally related to a particular object in the system, but related to overall system performance),

controlling output, etc., and (2) simulation element objects, which provide the ability to model and track an object's status, implement the event codes for specific objects modeled, calculate applicable internal statistics, etc. The simulation element objects which will be implemented shall consist of those needed to model a real system chosen specifically as a prototype development target.

Further, model building procedures for the use of available objects within a simulation model must be specified and tested for compatibility with the simulation object designs.

2. Develop measures which allow the comparison of pertinent aspects of modeling environments.

In order to judge the impact of the new paradigm of OOM, a determination of the important features in modern simulation environments must be made. Using this information, valid measures relating to ease of modeling, degrees of detail, etc. will be developed. These measures will probably take both intangible (non-numeric, qualitative) and tangible (numeric, quantitative) forms. An example of an intangible measure would be the degree of abstraction required in building the simulation model. A tangible measure might be the amount of time it takes for a working model to be

developed.

Another manner in which the analysis procedure may be designed is through the use of the structured, multicriteria technique known as the Analytic Hierarchy Process (AHP). Building upon the significant features determined previously, the decision process of choosing the "best" simulation environment shall be modeled hierarchically and solved through the AHP weighting process.

3. Evaluate the effectiveness of the new simulation environment.

In order to measure the benefits of an OOM approach, a comparison between the prototype OOM system and traditional approaches to simulation will be made through the application of the environment measures of performance. During the prototype stage, at which this evaluation will be performed, a large portion of this step shall be composed of a convincing analysis in the form of a logically consistent argument. Additionally, rating comparisons shall be gathered from knowledgeable individuals and used within the previously developed decision model during the application of the AHP analysis procedure.

4. Explore ways to expand the functionality of the developed environment. Conceptualize a

comprehensive framework for conducting a long-term research program to bring about the fruition of the OOM environment.

The author views this research as one portion of an on-going program composed of multiple contiguous phases. Each phase will build additional features and understanding onto the foundation provided by previous phases. The fulfillment of this objective will provide future directions for further research.

The principle assumption made in this project is that the research of an OOM environment shall be oriented towards the simulation of manufacturing systems. This is not intended to imply that knowledge gained here will not be applicable to other systems, on the contrary, it quite probably will be broadly applicable. However, this research project will specifically consider OOM applied to discrete part manufacturing.

CHAPTER V

RESEARCH PLAN AND PROCEDURES

To achieve the goals and objectives outlined in Chapter IV of this research proposal, the research will be performed through several different chronologically ordered phases as presented below.

Phase I

Conceptual and functional specification of the object classes needed to implement an OOM environment of sufficient magnitude to evaluate the effectiveness of the approach. As a target system, a portion of the manufacturing operations of an electronics manufacturer, specifically the electronics kitshop, has been chosen. Figure 7 presents a diagram of the physical layout of this system. Components enter the system as "selects", which are directly applied to kits; bulk parts, which are preformed prior to inclusion in kits; and reeled parts, which are sequenced before being applied to kits. Kits which exit the kitshop are composed of the appropriate grouping of selects, preformed bulk parts, and sequenced reels. The work stations include one sequencing machine, ten kitting stations, and fifteen preform operation stations. There are WIP storage locations for

selects, preformed bulk parts, partially completed kits, and sequenced reels. Approximately ten different kits are produced within the kitshop and processing times for three representative kit types have been generated.

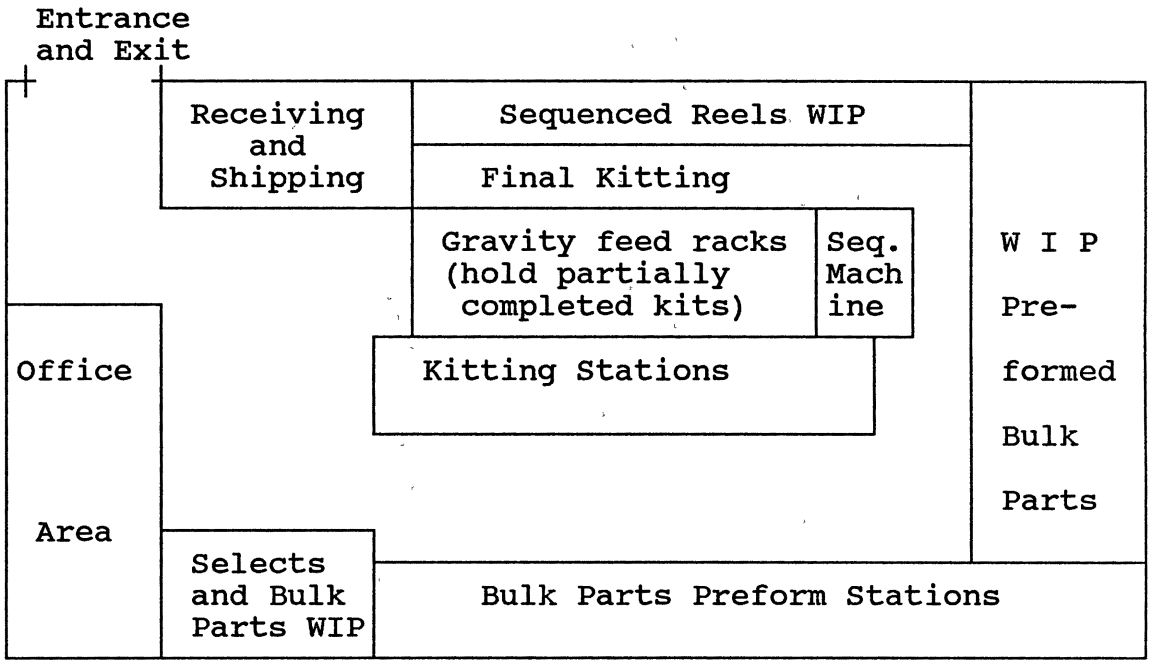


Figure 7. Electronics Kitshop Diagram

Phase II

Determination of the object linking and model building procedures based upon the functional specifications from Phase I. Alterations of Phase I results may occur.

Phase III

Implementation of the Phase I functional design within the general software environment (Smalltalk) with consideration given to the message passing capabilities needed for the linking procedures.

Phase IV

Application of the developed classes within a simulation model. This shall result in the demonstration of the achievement of one portion of the overall research goal, the implementation of an OOM environment.

Phase V

Conceptualization and formal development of criteria by which to measure the features of simulation environments. In order to compare environments in Phase VI of this research, measures of performance allowing valid comparisons of simulation environments must be designed and tested. Features which would generally be called intangible shall also be considered. Each of these features shall be used in the AHP hierarchy development which will also be completed within this phase.

Phase VI

Application of the developed criteria in the measurement and comparison of the new environment and other commonly used environments (which will be selected prior to

measurement and comparison). During this phase, the measures designed in Phase V will be applied to the simulation environments chosen for the study. This analysis shall be composed of two parts, (1) a coherent argument providing a logically consistent comparison of the environments and (2) the completion of the AHP analysis for the hierarchical model developed in Phase V. Conclusions drawn from this comparison should allow the researcher to determine the benefits and disbenefits of an object oriented approach to simulation modeling.

Phase VII

Development of the long term framework providing future directions for this area of research. At the conclusion of the previous phases, a prototype OOM environment will have been achieved. In order to gain the full benefits of the OOP paradigm, additional functionality should be added in the future. By providing a planned approach to the improvement of the OOM system, this increase in functionality can be made in a coherent and efficient manner.

CHAPTER VI

DEVELOPMENT OF A PROTOTYPE OBJECT ORIENTED MODELING (OOM) ENVIRONMENT

This chapter presents the steps taken in the design and implementation of an OOM environment and describes the features and capabilities of the resulting implementation.

Conceptual Design of a Prototype Object Oriented Modeling (OOM) Environment

Introduction

The development of a prototype simulation modeling system in an Object Oriented Programming (OOP) environment involves the design and implementation of a system composed of two broad classes of objects. These two classifications of objects are simulation processing objects and simulation element objects. Simulation processing objects are abstract objects providing the software functions which allow the background simulation processing tasks, such as: time advance, event triggering, entity creation, list processing, etc., to be performed. Simulation element objects, which provide the reusable simulation model building blocks, are

implemented in such a way that their actions model the activities of actual elements making up the system of interest. The following sections provide a detailed discussion of both of these types of objects.

Prior to pursuing this discussion, it is appropriate to define certain concepts relating to the human component of the OOM environment. It is feasible to conceive of three types of human interaction with the OOM environment: 1) Model Developer, 2) Class Developer, and 3) Environment Controller. The Model Developer is the person who will use the already implemented simulation classes in the construction of simulation models. It is assumed that the Model Developer is familiar with the system of interest, the basic concepts behind the technique of simulation, and the manner in which OOM models are constructed. The Class Developer is the title for an individual who has the privilege and responsibility of extending the modeling environment through the definition of new simulation element object classes. This person must have a significant level of knowledge on the Smalltalk environment and language and on the inter-simulation element communication procedures used. Finally, the Environment Controller has "software quality control" responsibility. The need for implementation of new simulation element classes must first be approved by this individual and, upon completion, the conceptual and software implementations must be approved before the new class(es) are used. The Environment

Controller must have broad knowledge of the system within which OOM is used and of the Smalltalk language and environment. As might be expected, the boundary lines separating these individuals and tasks are not rigidly drawn, but remain flexible. It is quite feasible that all of these tasks could reside within the responsibilities of one person or each task could be handled collectively by a group of people, depending upon the size of the organization and the extent of simulation modeling activities. In the following discussion, these terms and concepts will be used when the human interaction is mentioned.

Overall Structure

The overall structure of the simulation classes which shall be added to the OOP hierarchical tree is represented in Figure 8. Note that the simulation classes are grouped together within the class tree under a placeholding class called SimObject. This class serves as a top level location for the provision of global simulation functions and information storage locations. A major objective in the design of the simulation system classes is to develop the classes and communication procedures between classes which will allow instances of classes to be generally reusable and system models to be reconfigurable. This reusable type of design is needed to take advantage of the OOP benefits of modularity and the loose coupling between objects due to message passing. In order to achieve this type of design,

the 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

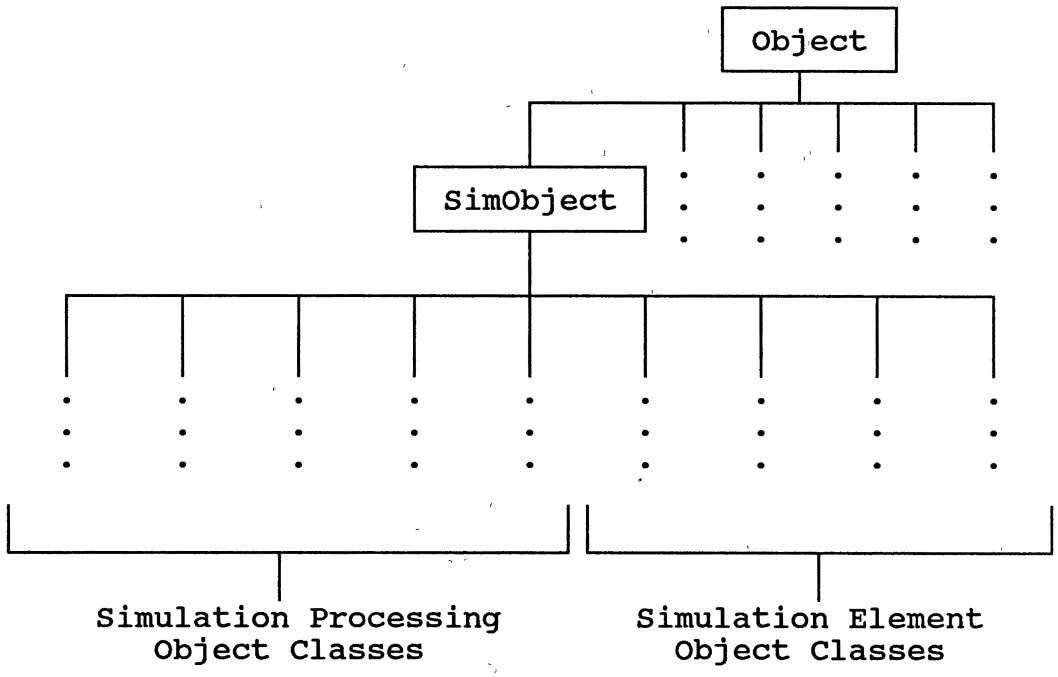


Figure 8. High Level Structure of the OOP Simulation Subtree

newly created instances of previously defined classes. The methods which are defined for the classes will be written in such a manner that this generally specified linking information is accessed through the instance variable locations or through responses to message requests. This approach to class design is referred to as the "basis for reusability".

Simulation Processing Objects

The simulation processing objects make up one portion of the simulation subtree (see Figure 9). The classes defining these objects are implemented by combining instances of other classes found within the software environment through the development of appropriate procedures. These procedures link the functions and methods of the other classes to provide the features needed within the new classes.

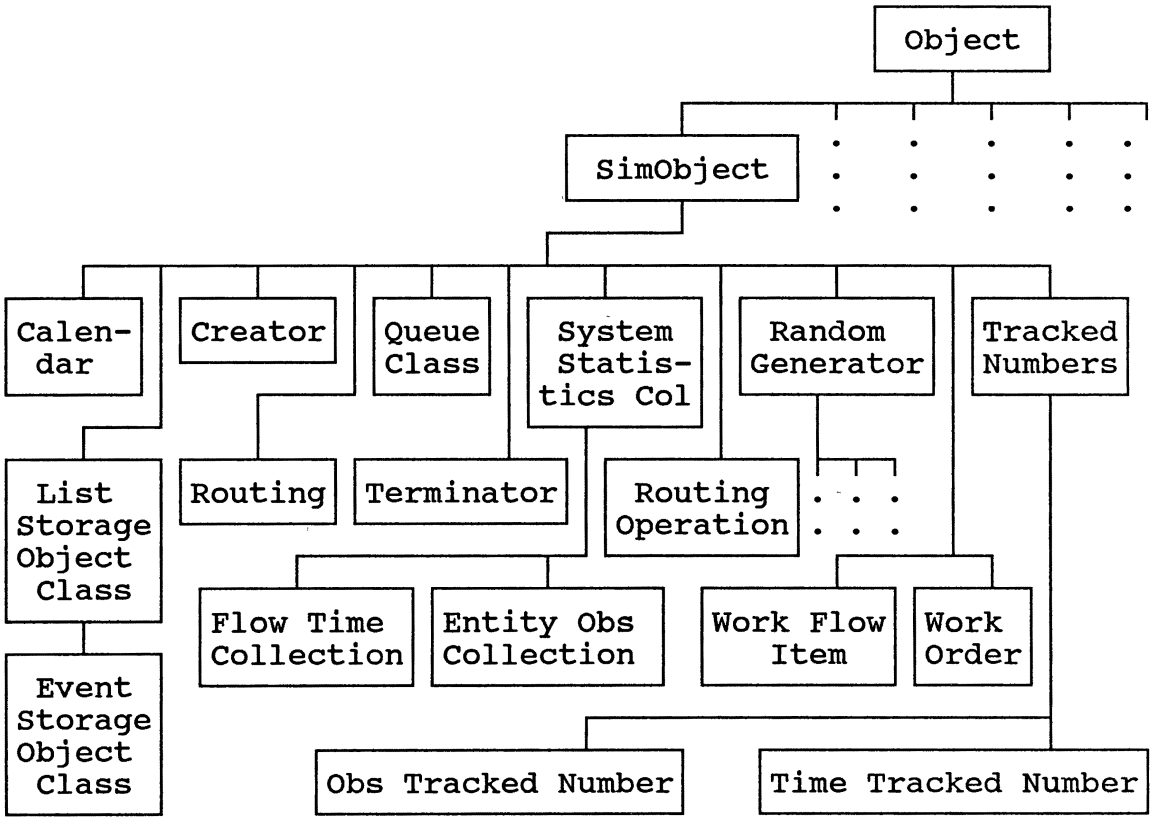


Figure 9. A Diagram of the Structure of the Simulation Processing Classes

The principal class present among the processing objects is the class called the "Calendar" class. The Calendar class provides several capabilities/methods which include the ability to:

1) Control the addition of events to and removal of events from the pending event list.

2) Provide the means with which events are triggered at their scheduled times.

3) Update the simulation world time as events continue to be processed.

4) Provide the simulation world time to requesting objects.

5) Trigger all objects in the simulation model software system to produce output at specified times and at the end of the simulation execution.

6) Trigger all objects in the system to clear their statistics data locations (to remove transient effects from the simulation results).

7) Maintain statistics on the event list.

The information maintained within the Calendar object includes the current simulated time, the event list pointers, event list statistical data, and the list of objects in the simulation model. The Calendar object provides the structure through which all communication between other system level objects within the simulated system occurs. All system level objects within the OOM model communicate <u>indirectly</u> with each other and <u>directly</u> with the Calendar object. Sublevel objects contained within the system level objects communicate hierarchically with the objects which contain them and with the objects that they

contain (communication between objects is discussed in Section 6.2). The Calendar object acts as the controller of system level interaction for the entire model run.

Calendar Class Summary:

Function:
The object within the simulation model which acts as the controller of the activities occurring within the model.

Data storage:
- Current simulated time value
- Event list pointers and information
- Event list statistical information
- System element list

Actions:
- Maintain and update the event list
- Clear system statistics at desired times
- Trigger production of output
- Collection of event list statistics

Another class definition needed is the Creator class. Instances of the Creator class have one capability, the ability to create a new instance of an entity or work flow item class or to trigger a group of creations by the Work Order Class. An instance of the Creator class schedules the creation activity on the calendar and performs the creation activity when the event is initiated by the calendar. After creating a new work flow item object(s) and scheduling the next creation event, the Creator passes the work flow item(s) on to the next object (based on routing information) in the system.

Creator Class Summary:

Function:
This class provides the manner in which all types of entities or work flow items (objects processed through the system) can be created and delivered to the system.

Data storage:
- Intercreation interval
- Type of object to create or message to execute

Actions:
- Creation of entities at appropriate times
- Scheduling of the next creation
- Transferring entities to the simulated system

Work flow items enter the simulation system from Creator instances and exit the system through Terminator instances. The Terminator class defines objects which provide a sink for work flow items passing through and exiting the system. In addition to performing this function, instances of the Terminator class collect data on the time work flow items spend in the system.

Terminator Class Summary:

Function:
This class provides a structure for the removal of work flow items from the system along with total flow time data collection.

Data storage:
- Statistical information locations

Actions:
- Accept the arrival of work flow items and process their information
- Terminate work flow items or entities from the simulation

Another simulation processing class needed within the OOM system is the Queue class. 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, to notify an

associated Work In Process (WIP) Aggregator (discussed below) of its changes, and to collect and output statistics on its own activities. When a new simulation element class needing queueing features must 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.

Queue Class Summary:

Function:
Provide the complete implementation of a "queue" within a single, reusable building block. The queue is a passive object intended to be incorporated as an internal component of other active objects.

Data storage:
- Queue maintenance data
- Reference to the optional queue aggregator (see below)
- Statistical information

Actions:
- Addition and removal of objects to/from the queue, according to the specified queue discipline
- Notification to the queue aggregator of changes
- Collection of queue size and work flow item time in queue statistics

Two classes needed as building blocks to support the functions of other classes are the List Storage class and the Event Storage class. The List Storage class provides the structure for the building blocks which are used to construct the linked list portion of the Queue class. This class provides support for successor and predecessor pointers and a pointer to the object which is being stored. The Event Storage class inherits the features of the List Storage class and adds the ability to store an event code and an event time. Obviously, instances of this class are

used to construct the event list within the Calendar object.

List Storage Class Summary:

Function:
To provide a subcomponent or building block for the construction of linked lists (queues).

Data storage:
- Linked list pointers
- Stored object pointer
- Time of entry to queue marker

Actions:
- Set and return pointer and time of entry values

Event Storage Class Summary (in addition to above):

Function:
Provide subcomponent support for construction of a scheduled event list ordered on the event time.

Data storage:
- Event initiation code
- Scheduled event time

Actions:
- Set and return event code and time

Another group of classes provided as building blocks (like the Queue) useful for simulation element construction are the Tracked Number, Observation Tracked Number, and Time Tracked Number classes. These classes are used to collect all statistics in the simulation environment and provide the ability to collect observation or time based data, calculate statistics from these observations, and print the statistics in a standard output format.

Tracked Number Class Summary:

Function:
Provide inheritable data storage and methods for the Observation and Time Tracked Number classes.

Data storage:
- Current value

- Cumulated value and cumulated squared values.
- Minimum and maximum values
- Time of last initialization

Actions:
- Set initial values
- Return current value

Observation Tracked Number Class Summary (in addition to above):

Function:
Add to Tracked Number features to allow observation based data to be collected and processed.

Data storage:
- Number of values collected

Actions:
- Update all statistics collection locations based on a new observation.
- Calculate statistics on current observations
- Print statistics according to a standard output format

Time Tracked Number Class Summary (in addition to above):

Function:
Add to Tracked Number features to allow time based data to be collected and processed.

Data storage:
- Time of the last change
- Number of value changes made

Actions:
- Update all statistics collection locations based on a new value and the current time.
- Calculate statistics on current observations
- Print statistics according to a standard output format

The generation of random numbers requires the definition of a subtree providing the class definitions needed for several probability distributions. This subtree, shown in Figure 10, is composed of a root class, Random Generator, which has the ability to store a seed value and to generate the zero - one uniform random variables commonly needed to produce samples from typical probability

distributions, along with several subclasses. In addition to features needed to support random variate generation by its subclasses, the Random Generator class has methods which allow it to generate samples from probability distributions when provided with the distribution specific parameter values as arguments within messages. As subclasses to Random Generator, classes with the methods and instance variable storage locations (for parameters) needed for the generation of samples from specific distributions such as Exponential, Normal, Uniform, etc., are defined. These classes use the features inherited from the Random

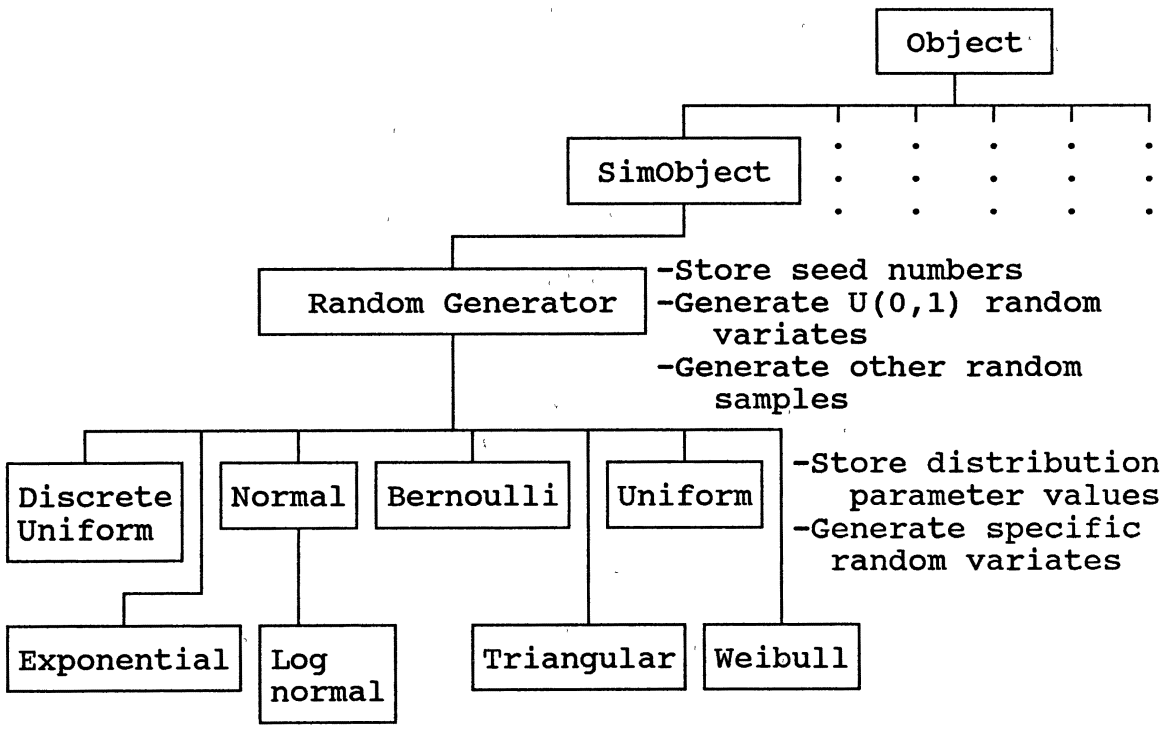


Figure 10. The Random Generator Subtree

Generator class and augment them with features specific to their distribution, providing an alternate method for the generation of samples (as opposed to methods attached to the Random Generator class).

Random Generator Class Summary:

Function:
Provide a mechanism through which samples from probability distributions may be generated.

Data storage:
- Current seed value
- Parameter values as required by specific distributions implemented (among the subclasses)

Actions:
- Set and return seed and parameter values (among the subclasses)
- Generation of a random sample from a distribution (at both Random Generator class and subclasses)

Another group of classes needed within the simulation processing objects is the System Statistics Collection classes. These classes provide the ability to perform the collection of overall system data during simulation execution. There are two types of system statistics which need to be collected in simulation, 1) flow times between two points and 2) specific observations of entity attributes. As such, two collection classes are implemented to allow these statistics to be gathered. One class provides for the marking of work flow items and later collection of flow time observations and the second class functions to collect specific observations.

System Statistics Collection Class Summary:

Function:
Provide a mechanism through which overall system statistical observations may be made.

Data storage:
- Collected information on the observations
- Model connections linking information

Actions:
- Gather observations as entities pass through
- Print results as required

The Work Flow Item class is a simulation processing object class which is needed to provide a structure for the representation of work flow item types and attributes, specification of routing data, and provision for flow time marking. Instances of this class are passive objects which provide required data in response to queries from active system objects.

Work Flow Item Class Summary:

Function:
Provides for the representation of parts (flow items to be processed) or entities and their data flowing through a simulated system.

Data storage:
- Work flow item creation times and flow time markers
- Work flow item routing and processing time information
- Work flow item type and work order designation

Actions:
- The ability to set and return the various internally stored values.

Two classes highly related to the work flow items are the Routing and Routing Operation classes. The Routing class defines the structure and capabilities needed for the representation of processing routings which are attached to

instances of the Work Flow Item class. The Routing Operation class defines building blocks which are combined into a routing.

Routing Class Summary:

Function:
Provide for the representation of routing information attached to work flow items.

Data storage:
- Routing operations

Actions:
- Provide copies of itself
- Add new operations including operation location, processing and setup times
- Return and remove the first operation on the operation list

Routing Operation Class Summary:

Function:
Provide for the storage of information on a single routing operation within the routing.

Data storage:
- Simulation processing object access code
- Processing and setup time generation code

Actions:
- Set and retrieve access code and processing and setup time generation codes

The Work Order Class is a class of objects designed to operate in conjunction with the Creator class in the creation of complete jobs for processing systems. As such, instances of the Work Order Class are instantiated with the information needed to allow the creation of all Work Flow Items which typically are released as part of a single work order. A Creator object triggers the method attached to the Work Order Class which creates new Work Flow Item instances and initializes the item labels, creation time, and routing

information for each of the new Work Flow Items. The new Work Flow Items are returned to the Creator object which releases them into the system.

Work Order Class Summary:

Function:
Provide for the instantiation and initialization of all Work Flow Items needed to make up a complete Work Order.

Data storage:
- Labels, routings, and processing time specifiers for each Work Flow Item
- Current Work Order number (at the class level)

Actions:
- Create new Work Flow Items
- Initialize new Work Flow Items according to the data storage
- Return items to the Creation object

The OOM classes discussed to this point are of a highly abstract nature and represent the objects or concepts which must be explicitly accomplished to make simulation work. The simulation element objects, covered in the next section, round out the capabilities of the OOM environment by representing the concrete elements present in the system(s) of interest.

Simulation Element Objects

The simulation element classes are a group of subclasses of the simulation root class, SimObject (see Figure 11). Once again, this allows the classes to inherit features defined at the SimObject level and needed commonly among all of the simulation classes (available through inheritance). Each simulation element class is set up to

have all event and internal processing methods (load, unload, measure of performance calculation, etc.) implemented as part of the class definition along with the

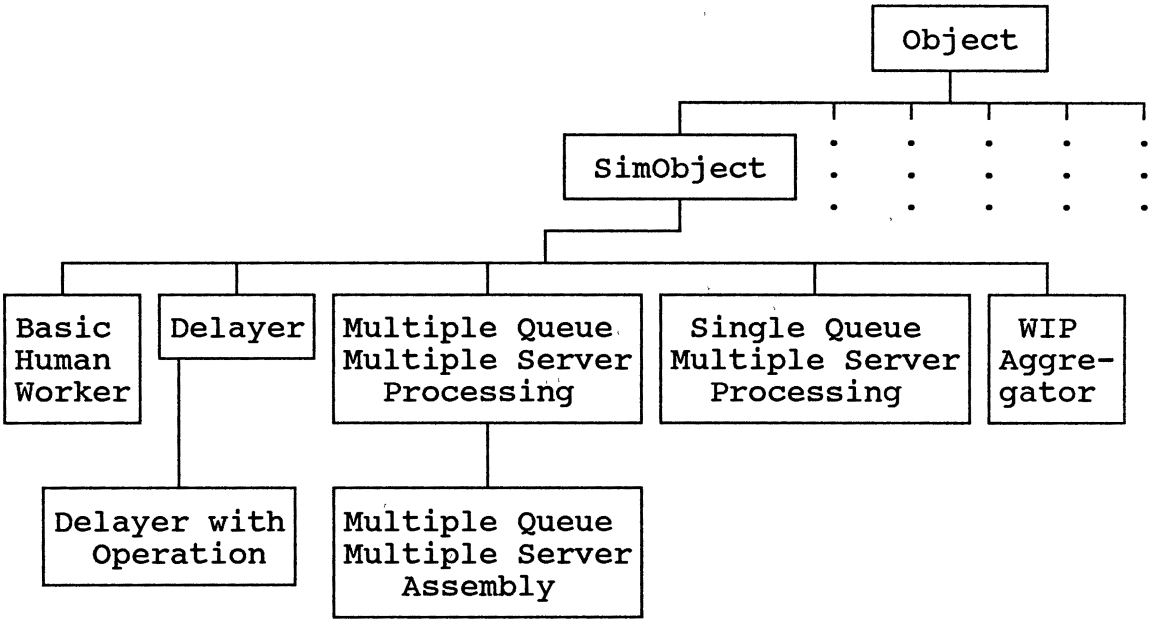


Figure 11. A Diagram of the Structure of the Simulation Element Classes (as Developed for the Target System)

appropriate internal instance variables needed to keep track of the statistical data and state of the object. Instances of the simulation element classes are used as building blocks in the construction of the simulation model. As mentioned previously, the simulation element classes which have been developed are those determined necessary to implement a model of the chosen target system. This target

system is an electronics kitshop (described in Chapter 5). Figure 12 represents a rough sketch of the layout of the kitshop.

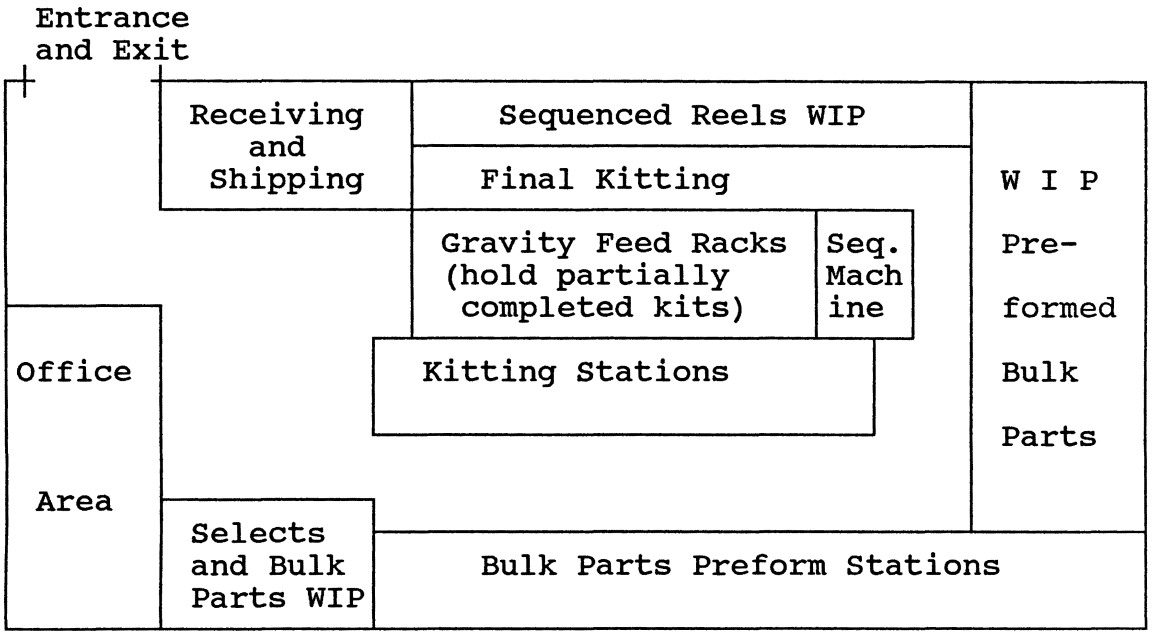


Figure 12. Electronics Kitshop Diagram (Repeat of Figure 7)

The major components of this system (through which kits are processed) are as follows:

Receiving -Incoming selects and bulk parts are verified and paper work is generated and the parts enter the Selects (plastic tubed chips) and Bulk Parts WIP locations. Incoming HICS (hybrid integrated circuits) are sent to the Sequenced Reels WIP location.

WIP Storage locations - The WIP locations within the kitshop system function primarily as centralized queueing centers. The Selects and Bulk Parts WIP location acts as a queueing system for the kitting stations and the bulk parts preform stations. The

Preformed Bulk Parts WIP provides queueing for the kitting stations and the sequencing machine, while the Gravity feed racks and the Sequenced reels and HICS WIP locations are queueing systems for the final kitting operation.

Sequencing machine - This operation uses reeled parts waiting in the Selects and Bulk Parts WIP location and produces sequenced reels, which queue into the Sequenced Reels WIP location.

Bulk parts preform - The bulk preform stations process parts waiting in the Selects and Bulk Parts WIP and transfer the result to the Preformed Bulk Parts WIP.

Kitting stations - Kitting stations combine selects and preformed bulk parts into partially completed kits which enter the gravity feed racks.

Final kitting - This operation combines partial kits, sequenced reels, and HICS into the completed outgoing kit.

Shipping - Shipping is an auditing and data collection point before parts enter the manufacturing system.

The simulation model representation of this system requires several different types of objects including work orders and parts (with special representation of each of the different part types), single queue - multiple server processing stations, multiple queue - multiple server assembly stations, multiple queue - multiple server processing stations, WIP aggregators, and humans, plus the simulation processing objects (described previously), including a calendar, creators, terminators, and system statistic collectors (as desired).

The workorders and parts in the simulation are actually instances of class Work Flow Item with specific instance variable values. Workorders (different from the Work Order

Class) are a special type of work flow item which function as the basic product unit around which other consumed work flow items are aggregated. Consider that in the system being modeled, multiple parts and part types are combined to result in a single unit, the complete workorder kit. In order to facilitate the required processing, the workorder objects provide focal work flow items around which disjoint kitting operations will be linked. The workorder objects are routed in order through each station or processor at which an assembly operation occurs. Upon exit from an assembly point, the workorder object is routed further in the system (it represents the assembled collection) while work flow items representing parts which were assembled onto the workorder are routed to specific Terminator objects. Note that this allows flow time statistics to be collected on the assembled work flow items as well as the entire workorder. Other work flow items in the system include selects, bulk parts, and reeled parts. Each of these different part types is represented by work flow item objects with the appropriate instance variable values and transfer through the system as mentioned previously.

Workorder and parts summary:

Function:
Instances of the Work Flow Item class which represent workorders and parts in the simulated system. The workorder type of object (a member of the Work Flow Item class with special instance variable values) guides the overall routing of the output item(s) of interest through multiple assembly points.

Data storage:
- Assembly and processing points routing for the

finished product
- Part type label and time of system arrival

Actions:
- Provides access to the information on the routing
- Responds to requests for flow time statistics gathering and observation statistics gathering

The bulk parts preform stations are set up as multiple servers processing work flow items from a single queue of waiting items. Therefore, an OOM class which allows this type of station (single queue, multiple server processing station) to be represented was implemented. The station is able to accept the arrival of new work flow items, determine an available server from among those allocated, schedule the service operation, and transfer the part to the next processing station. In addition, the station must keep statistics on its operation and provide for their output as requested.

Single Queue, Multiple Server Processing Station Summary:

Function:
To represent a single queue, multiple server station within a simulation model.

Data storage:
- Number of servers allocated
- Status and statistics on each parallel server allocated
- Maintain a reference to the internal queue

Actions:
- Accept a new work flow item
- Schedule processing of work flow items
- Transfer work flow items to other objects upon completion of processing
- Collect utilization statistics and produce output

The kitting and final kitting operations within the kitshop system are set up as multiple servers which assemble

work flow items (of the same work order) from multiple queues. The OOM class used to represent this station must be able to accept the arrival of new work flow items, place the items in the appropriate queue based on item type, match items from among the different queues, determine an available server from among those allocated, schedule the service operation, and transfer the parts to the next processing station. Statistics on these activities must be kept and produced as output when required.

Multiple Queue, Multiple Server Assembly Station Summary:

Function:
Provide for the simulation representation of an assembly station having multiple queues and servers.

Data storage:
- Number of servers allocated
- Number of queues allocated
- Status and statistics on each parallel server allocated
- Maintain references to the internal queues

Actions:
- Accept new work flow items and segregate by type
- Match work flow items from queues based on work order number
- Schedule processing of work flow items
- Transfer work flow items to other objects upon completion of processing
- Collect utilization statistics and produce output

The receiving and shipping functions are primarily information processing activities performed by one or two individuals on two separate queues of work orders. As an appropriate object to model this situation, a multiple queue - multiple server processing station representation is required. The OOM class used to represent this station must be able to accept the arrival of new work flow items, place

the items in the appropriate queue based on entry to or exit from the system, determine an available server from among those allocated and the queue from which items should be served, schedule the service operation, and transfer the parts to the next processing station. For exiting work flow items, the next step is to transfer to an instance of the Terminator class. Entering items are sent to the next processing station on their particular routing.

Multiple Queue, Multiple Server Processing Station Summary:

Function:
Provide for the simulation representation of a processing station having multiple input ports (queues), multiple output ports, and multiple servers.

Data storage:
- Number of servers allocated
- Status and statistics on each parallel server allocated
- Maintain references to the internal queues

Actions:
- Accept new work flow items and segregate to the appropriate queue
- Determine the queue from which items should be removed for processing
- Schedule processing of work flow items
- Transfer work flow items to other objects (either into or out of the system) upon completion of processing
- Collect utilization statistics and produce output

One of the most difficult aspects of the system which has been chosen for prototype modeling is that the WIP is stored in centralized locations and, yet, is waiting for service from different assembly or processing stations. This is an example of a physical grouping of material into centralized locations and a logical grouping of material within the same physical location into separate queues. As

was discussed, the work stations already described shall have direct access and control of their own queues, however, a WIP Aggregator class has been implemented to gather statistics on the queued material as it appears in the physical system. The queues associated with a specific WIP Aggregator have pointers to the WIP Aggregator with which they are associated. When changes occur in a queue, the queue notifies its specific WIP Aggregator object which then collects statistics on all queues associated with it. This setup provides the capability to represent as a unit the simulated information from a centralized WIP storage location (in the real system) modeled as a distributed WIP storage system (in the simulation model).

WIP Aggregator Class Summary:

Function:
This object provides the capability to track the aggregate contents of multiple work flow item queues which occupy the same physical location in the real system.

Data storage:
- Statistical information

Actions:
- Collection of information from associated queues
- Output of results

One class needed for the specific conditions of the target system is the Delayer class. This class allows an unlimited number of work flow items to delay for a specified (on the routing) period of time. Basically, work flow items enter a Delayer instance and are scheduled to arrive at their next destination in some amount of delay time

generated as a random variable. The Delayer With Operation class provides the ability to perform some operation on the work flow item in addition to providing a delay capability.

Delayer Class Summary:

Function:
This class provides the capability to delay a work flow item for a specified (by the processing time on the routing) period of time.

Data storage:
- Number of delayed work flow items

Actions:
- Transfer work flow items through a delay
- Print statistics on the number of delayed items

Delayer With Operation Class Summary (in addition to above):

Function:
Add the ability of performing an operation to the Delayer class.

Data storage:
- Operation specification context

Actions:
- Perform operation on work flow items

A simple class representing the cycle of rest and work exhibited by the human workers in the system completes the list of simulation element classes. The Basic Human Worker class defines an object which switches between active and inactive using times based on statistical distributions specified by the model builder. The class provides the ability to signal the work station when switching its status and responds to status queries and is used as a subcomponent of top level system elements.

Basic Human Worker Class Summary:

Functions:
Represents a simple active/inactive human.

Data storage:
- Machine or station which contains the object as a subcomponent and index of the object within the station implementation (for multi-server stations)
- Status information

Actions:
- Switching between active/inactive
- Notifying the containing object of a status switch
- Responding to status queries

Simulation Model Operation

The major classes needed to develop a working simulation for the target system have been described. Figure 13 illustrates a large portion of the entire simulation class subtree. The following sections describe, in high level terms, the manner in which the objects will cooperate with one another during the simulation activity.

Time Advancement. The Calendar object in an OOP simulation system handles time advancement. Time advance occurs by having the calendar object loop through a portion of a method to find the next event on the calendar. This event initiation method then sets the new value of the current time instance variable and triggers the next event to occur by executing the event initiation 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).

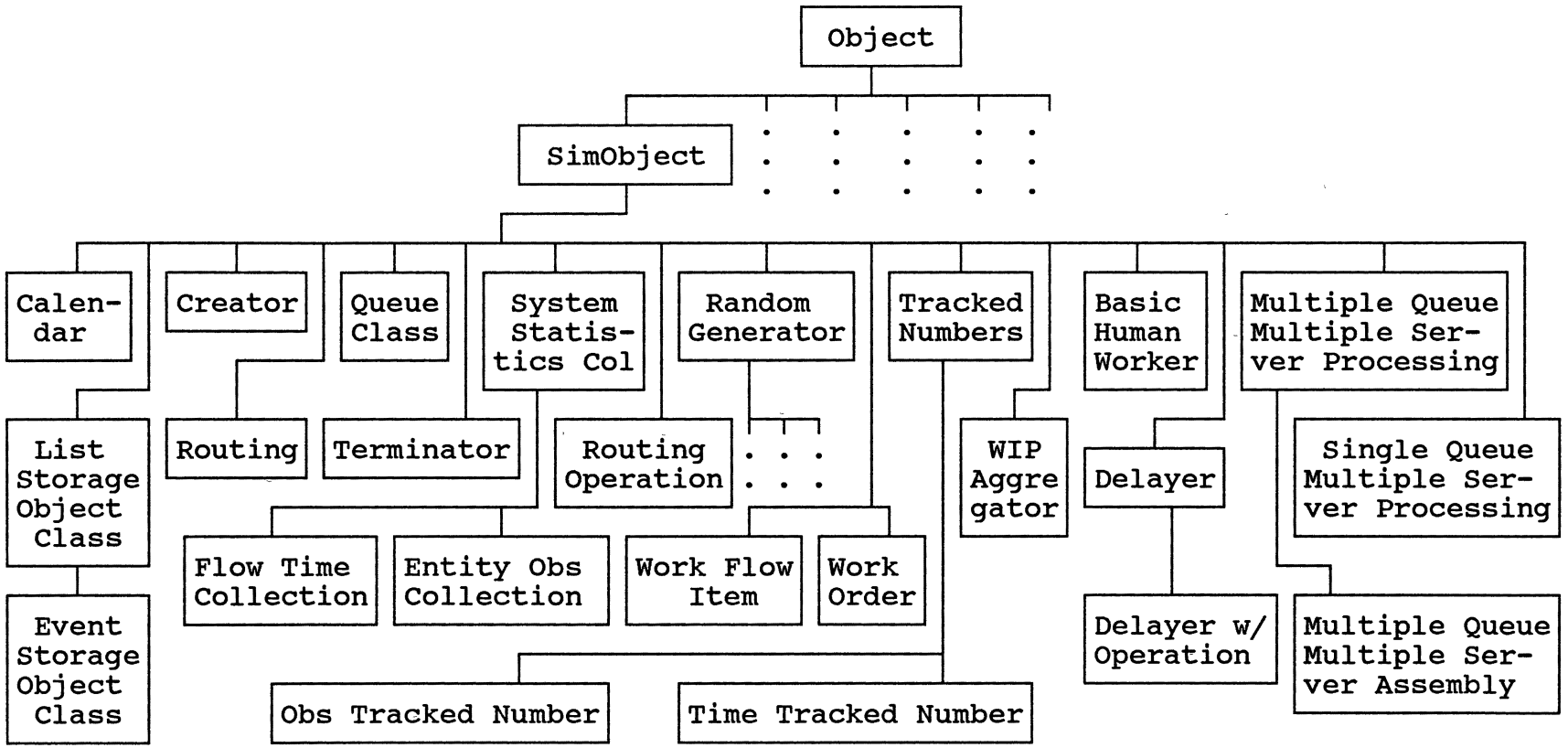


Figure 13. The Complete Prototype OOM Simulation Environment

Entity Creation and Flow. As mentioned earlier, a special class of object, the Creator object, is used to implement new instances of entities and trigger their arrival to the simulation element instances which are part of the model. This creation will be initiated when a creation event on the calendar list becomes the next event to be processed. Each instance of the Creator class is instantiated with the ability to create one specific type of object or group of objects. The calendar uses the information stored on the event list to tell the appropriate Creator object when to create an entity. The Creator object will create a new instance of a Work Flow Item (or group of Work Flow Items) and use its internal routing information to schedule the arrival event with the correct time delay on the calendar. Another creation event can be scheduled as part of the same method. During normal time advance operation (either with zero or nonzero time advance), the calendar object will arrange for the arrival event to be processed at the correct simulation model component object.

The travel of Work Flow Items through the model is completely controlled by the routing information contained within instance variables attached to the Work Flow Item object. When a departure event at an element of the model is processed, it retrieves the routing information provided by the routing attached to the Work Flow Item object. An arrival event to the next simulation model element listed on the routing is sent to the Calendar object for scheduling on

the event list. This arrival event contains the pointer to the Work Flow Item object which is being transferred. In general, the flow of entities among the elements in the simulation model is performed through controlled use of the relationship information stored in the routing that is a portion of the Work Flow Item object.

Event Initiation and Scheduling. Events are initiated by the calendar object while it continues to monitor the event list. The calendar uses the information on the list to access the correct method attached to the element object and pass along the Work Flow Item identifier. As part of its activities, the simulation element object might pass an event creation message (to schedule the end of service) to the calendar object to add the appropriate event to its list. When the calendar method has completed its event list addition, control returns to the simulation element instance method, from which control will return to the calendar 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 calendar object, processed through all needed methods, and finally execution control is returned to the calendar object which then retrieves the next event on the calendar list.

Statistics Collection. There are two areas of

statistics collection which need to be addressed: 1) simulation element activity and 2) entity measures. Statistics collection for simulation element objects in the model is performed by the objects themselves. Statistics collection is either handled by a separate method attached to each simulation element object or by statements within event methods attached to the object. The method or statements are executed by the other methods attached to the simulation element to update statistics at each change in the simulation element status. This interaction between the methods is designed into the simulation element objects when the object itself is designed, not when it is included in the simulation model by the user. At the end of the simulation, the calendar object instructs each simulation element to access the values of the statistics instance variables, perform calculations to result in the output statistics of interest (average utilization, utilization standard deviation, etc.) and print these output statistics according to the format specified by the output method attached to the simulation element class.

Statistics collection for the entities (system statistics) is be handled by having Systems Statistics Collection objects (mentioned previously) as part of the simulation model. These objects are designed in such a way that they address one observation of interest on each Work Flow Item which they process. These classes are able to retrieve the specific observation of interest from the Work

Flow Item object. Entity arrival and departure from these objects are handled similarly to previous descriptions (see Entity Creation and Flow).

Summary

This discussion may lead one to the conclusion that an OOP 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 should be 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.

OOM Simulation Object Linking and Model Building Procedures

Introduction

The description of OOM classes provided in the previous section was made from a perspective internal to the classes and largely ignored the manner in which instances of classes will connect with one another. In order to build simulation

models, the objects in the simulation environment must communicate together in a way that supports generalized linking and the techniques used to provide this linkage must be understood.

The Structure of Object Oriented Models

A simulation model imbedded in an OOM environment is made up of a group of interconnected objects which work together to simulate the activities of the physical system modeled. These objects may represent machines, work items, queues, etc. Because one of the major objectives in the development of an OOM environment is to support the (desirable) reusability of simulation elements, a model structure must be designed which allows 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 linkages between system components are vertical (there are no horizontal links between subtrees in a hierarchical system). The first feature allows a hierarchical structure to be used, and the second feature supports reusability of simulation objects.

An illustration of this approach and the reasoning behind it is exhibited through the use of the hypothetical system pictured in Figure 14 and the corresponding OOM simulation model for the system illustrated in Figure 15.

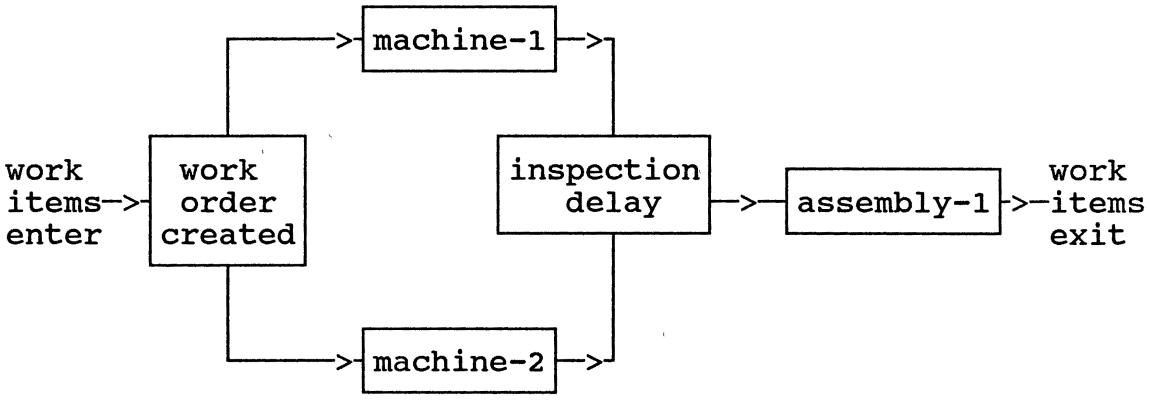
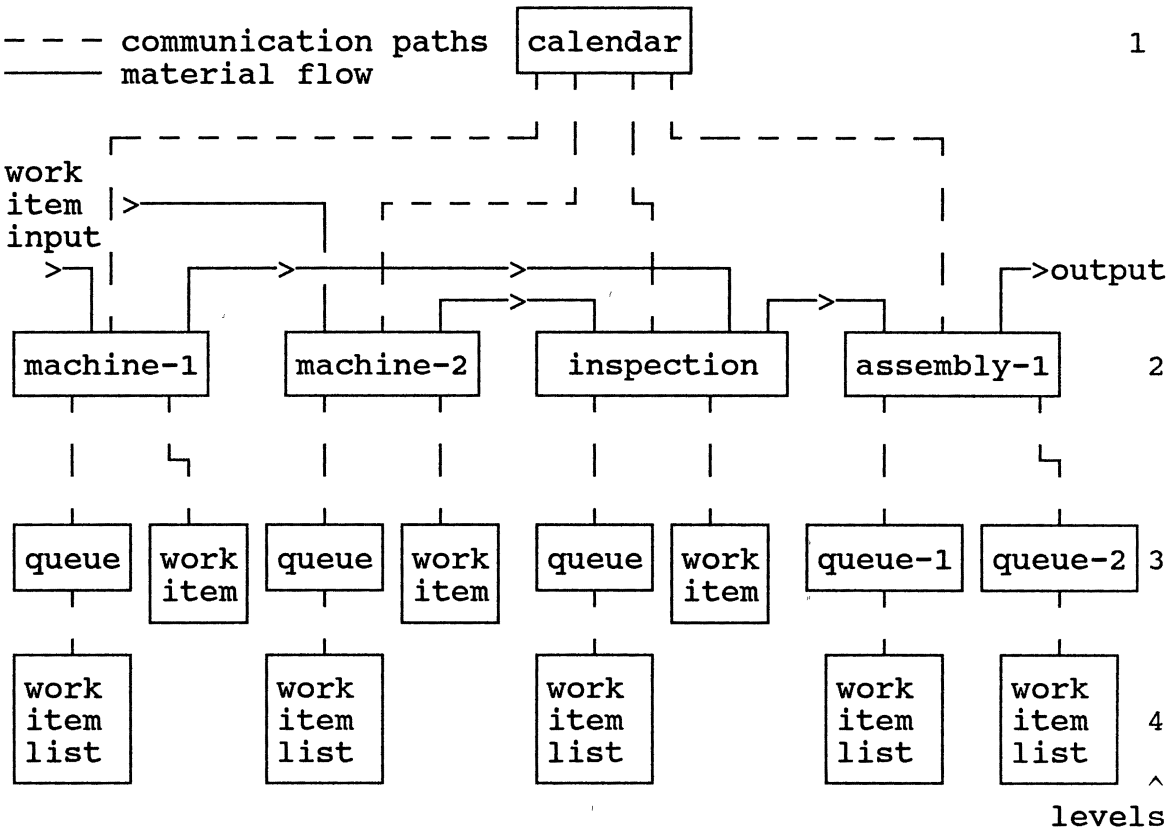


Figure 14. Hypothetical Physical System Targeted for Simulation Modeling

The illustrative system shown in Figure 14 is a simple system consisting of two processing machines, an inspection station, and an assembly station. Work items enter at machines-1 and -2, are processed, are transferred to the inspection station, and then are transferred to the assembly station. The work items are matched and assembled at the assembly station after which they exit the system.

Basically, this system should be representable in a simulation model by four separate top level simulation



Additional Communication - All objects in the simulation system can communicate (send and receive messages) with the calendar without regard to intervening model levels.

Figure 15. Potential Organization of the Major Elements of an OOM Model for the Hypothetical System (Work flow items and other peripheral items are not shown.)

element objects: 1) machine-1, 2) machine-2, 3) an inspection station, and 4) an assembly station. These four elements are shown in Figure 15 at level 2. The operation of the simulation model is as follows: Level 1 in Figure 15 contains the calendar instance for the model. The calendar maintains control of the execution of the simulation model

and coordinates the interaction of level 2 model elements. It receives messages to schedule event occurrences from the objects located at level 2 (the system element level) and initiates messages (from information on the event calendar) to trigger the start of event processing by the objects in level 2. The objects in level 2 interact directly with those in level 3 which are subcomponents of the corresponding level 2 objects. Level 3 objects maintain direct links to those objects in level 4. Direct communication from a particular object is limited to other objects that exist either one level higher or one level lower in the same subtree of the model 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 calendar is accessible to all components of the simulation system.) The calendar acts as the controller for communication between system element level (level 2) objects in the model. The two major types of communication occurring in the model, communication between elements and communication within elements (between elements and subelements), are discussed in later sections.

Let us consider in more detail the construction of the simulation model as illustrated in Figure 15. Machine-1 (on the left hand side) has communication paths linking it to the calendar, which is above it in the model hierarchy, and

to its subcomponents, the queue and current work item, which are below it in the model hierarchy. Notice that the hierarchical orientation of machine-1 to the calendar and to its subcomponents makes machine-1 capable of functioning independently of other components (present in other model subtrees) in the simulation system. The queue (a component of machine-1) has communcation paths linking it to machine-1 and to its internal work item list. In a similar manner, all objects are linked to other objects dependent on their location in the hierarchy. This type of linkage makes objects strictly dependent on the presence of a specific set of other objects in the system (those with which direct interaction must occur) and completely unaffected by the presence or absence of objects outside this specific set. In the case of machine-1, the set of required objects consists of the calendar, the internal queue, and the current work item. The design of a simulation class ensures that internal objects (queue, etc.) are available because these internal objects are set up whenever a new instance of the simulation class is created.

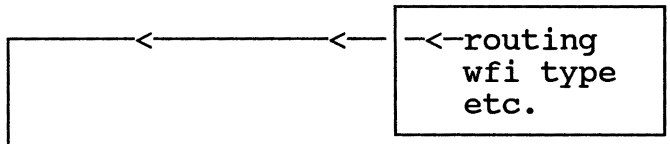
Element Level Object Linking (Communication Between Elements)

The communication between objects representing elements of the modeled system is principally driven through the scheduling of event occurrences on the event list in the

calendar. The transfer of work flow items between these top level elements is specified by the routings defined for the work flow items. As work flow items are processed through the simulated system, the elements in the model retrieve transfer information from the routing contained within the work flow item. This routing information provides a sequential list of all of the objects (physical system elements along with system statistics collection elements) that a work flow item must visit, the event code needed to trigger the transfer of the work flow items between elements, and the specification of the setup time, processing time, etc. at each element. This structure is illustrated in Figure 16 (based on the example situation illustrated in Figures 14 and 15).

In this conceptualization, system element objects exist in the model as separate "entities", with incomplete linking among themselves. This incomplete linking is completed by the information specified in the routing of the work flow item. The elements of the system accept new work flow items, process the items through the execution of internal activities (internal to the elements themselves) and the scheduling of internal events (which require time advance and occur through the calendar object), and schedule external events, such as work flow item transfer (which may or may not require time advance), on the event list. By scheduling and initiating events, the calendar supervises many intra-element (those requiring time advance) and all

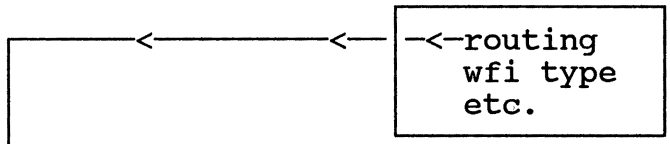
One type of work flow item and its routing:



Routing seq num	Simulation element	Entry link event code	Proc time specifier	Setup time specifier	etc.
1	machine-1	part-arriv	proc time	setup time	
2	inspection	part-arriv	proc time	setup time	
3	assembly-1	part-arv-1	proc time	setup time	
.	.	.	.	.	.
.	.	.	.	.	.

Work stations present in the simulated system:

Another type of work flow item and its routing:



Routing seq num	Simulation element	Entry link event code	Proc time specifier	Setup time specifier	etc.
1	machine-2	part-arriv	proc time	setup time	
2	inspection	part-arriv	proc time	setup time	
3	assembly-1	part-arv-2	proc time	setup time	
.	.	.	.	.	.
.	.	.	.	.	.

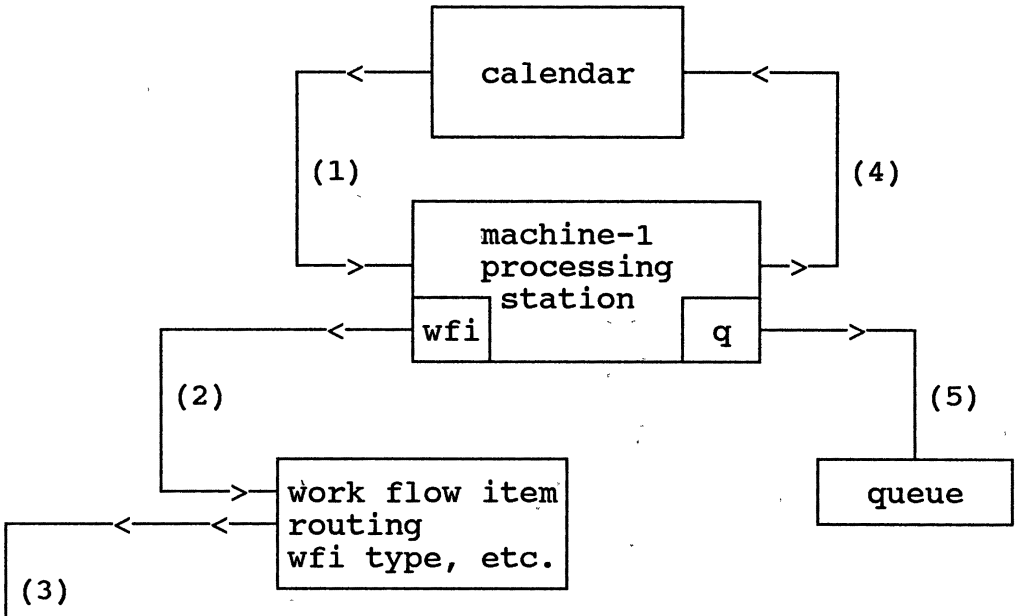
Figure 16. A Pictorial Representation of the Relationship Between Work Stations and Between Work Flow Items and Work Stations, as Provided by Routing Information

inter-element activities at level 2 in the model hierarchy.

Subelement Level Object Linking (Communication within Elements)

Moving down a level in the simulation model hierarchy, model elements themselves (level 2 in Figure 15) are constructed of multiple subelements (objects at level 3 in Figure 15) which are linked together to construct the model element by the instance methods defined for the class. As an example, consider a machine composed of a queue and several status variables. A work flow item arriving to the machine will enter service if the machine is idle or enter the queue if the machine is busy. The machine instance methods directly check the status variables and send work flow items either directly to the machine or to the queue if the machine is busy. As activities occur at the element level (such as the work flow item arrival just discussed), messages are sent to the subelements to perform functions (work flow item storage or retrieval by the queue) as part of the representation of the complete element's actions. Drawing on a portion of the example used in the previous explanation, Figure 17 (based on Figures 14, 15, and 16) illustrates this concept of hierarchical communication linkages.

Assume, for the sake of discussion, (1) that the processing station (machine-1) has just completed (through



Routing seq num	Simulation element	Entry link event code	Proc time specifier	Setup time specifier	etc.
2	inspection	part-arriv	proc time	setup time	
3	assembly-1	part-arv-1	proc time	setup time	
.	.	.	.	.	.
.	.	.	.	.	.

Figure 17. Communication Methods to Subelements

time advance) the processing of a work flow item. The processing station class passes a message (2) to its work flow item subelement in order to get the transfer information from its routing. The work flow item retrieves the needed information (3) from the routing (which is a subelement of the work flow item) and provides it as a response to the processing station request. Using this information, the processing station object transfers the work flow item to its next station by scheduling the arrival

event on the calendar (4) (as described previously). The processing station methods would next check the internal queue(s) for waiting jobs (5). All communication between an element and its subelements occurs in a similar hierarchical manner within the OOM environment.

Construction of OOM Based Simulation Models

There are several steps to follow in the construction of a simulation model in the prototype Smalltalk OOM environment.

These are as follows:

1) Set up temporary variables which will provide element level (human interaction level) symbols for the element level objects used in the system.

2) Set up a new Calendar class instance.

3) Create instances of classes as needed for the representation of the physical system being modeled and set the temporary variables to point to these instances. Also create instances for terminators and system statistics collection objects.

4) Set up Creator instances for each work flow item type or work order type which will be traveling through the system. This involves specifying (1) the work order and the routings (in terms of the temporary variables mentioned above) through all objects (including statistics collection, material handling, etc.) which the work flow items will be visiting (in sequential order) and (2) processing times at each location.

5) Set up the list of system elements to include all objects in the system from which output is desired.

6) Schedule any special initial events onto the event list. These might include intermediate results output, clearing of statistics at special times, initial work flow item arrivals, etc.

7) Start the execution of the simulation model by

messaging the calendar object.

As an example of this procedure, consider the pseudo-code model implementation shown in Figure 18 for the system shown in Figure 14. Each of the specific actions described above is illustrated in this figure. Note that during step 3 of Figure 18 each top level simulation element is automatically set up with the appropriate internal components.

Steps

```
1    Local variables (calendar, machine1, machine2,
                      inspection, assembly,
                      creat1, term1, term2,
                      workOrder routing1, routing2)
2    calendar = Calendar new
3    machine1 = Simple machine new
     machine2 = Simple machine new
     inspection = Simple machine new
     assembly = Assembly station (2 queue) new
     term1 = Terminator new (Final assembly term)
     term2 = Terminator new (Assembled WFI term)
4    routing1 = Routing new with operations:
                (machine1, 'a processing time', etc.)
                (inspection, 'a processing time', etc.)
                (assembly at queue 1, 'a processing time',
                  etc.)
                (term1)
     routing2 = Routing new with operations:
                (machine2, 'a processing time', etc.)
                (inspection, 'a processing time', etc.)
                (assembly at queue 2, etc.)
                (term2)
     workOrder = WorkOrder new ('part 1', routing1)
                               ('part 2', routing2)
     creat1 = Creator new (workOrder)
              time-between-creations ('a time specifier')
5    calendar set system elements: (calendar, machine1,
       machine2, inspection, assembly, term1, term2)
6    calendar schedule  (creat1 create) at  0
     calendar schedule  (calendar end)  at  480
7    calendar event processor
```

Figure 18. Pseudo-code OOM Simulation Model

As described, following the seven model development steps results in the creation of all needed model objects, in the specification of linking information (routings), and in setting initial events and beginning simulation processing. Once processing is completed, output is generated by each system level object according to the internally defined methods.

As a final, concrete illustration of this process, the actual Smalltalk implementation of the example simulation model is as follows:

```
"
   Step 1: Set up top level instance variables.
"
|calendar machine1 machine2
 inspection assembly
 creat1 term1 term2
 workOrder routing1 routing2|

"
   Step 2: Set up a new Calendar class instance.
"
calendar:= Calendar new.
"
   Step 3: Set up instances of classes to represent the
           physical system being modeled.  Create instances
           for terminators.
"
machine1:= SQueueMServerProc newWithName: 'Machine 1
                                           Station'
                             andSize: 1.
machine2:= SQueueMServerProc newWithName: 'Machine 2
                                           Station'
                             andSize: 1.
inspection:= SQueueMServerProc newWithName: 'Inspection
                                            Station'
                              andSize: 1.
assembly:= MQueueMServerAssem newWithName: 'Assembly
                                           Station'
                              andServers: 1
                              andQueues: 1.
term1:= Terminator newWithName: 'Final Assembly Terminator'.
term2:= Terminator newWithName: 'Assembled WFIs Terminator'.
```

```
"
   Step 4: Set up routings and creators.
"
routing1:= Routing new.
routing1 addOperation: machine1 key: nil
            processingTime: [:rg | rg uniformHigh: 5 low: 3]
            setupTime: nil;
         addOperation: inspection key: nil
            processingTime: [:rg | rg uniformHigh: 2 low: 1]
            setupTime: nil;
         addOperation: assembly key: 'workOrderQueue'
            processingTime: [:rg | rg uniformHigh: 2 low: 1]
            setupTime: nil;
         addOperation: term1 key: nil.

routing2:= Routing new.
routing2 addOperation: machine2 key: nil
            processingTime: [:rg | rg uniformHigh: 4 low: 2]
            setupTime: nil;
         addOperation: inspection key: nil
            processingTime: [:rg | rg uniformHigh: 2 low: 1]
            setupTime: nil;
         addOperation: assembly key: 1
            processingTime: [:rg | rg uniformHigh: 2 low: 1]
            setupTime: nil;
         addOperation: term2 key: nil.


workOrder:= WorkOrder newWorkOrderType: 'Work Order 1'.
WorkOrder setWorkOrderNumber: 1.
workOrder addComponentWFI:'part 1' andCWFIRouting: routing1;
          addComponentWFI:'part 2' andCWFIRouting: routing2.

creat1:= WOCreator newWithWorkOrder: workOrder
                   timeBetweenCreationsGenerator:
                      (Uniform newHigh: 8 low: 3).
"
   Step 5: Set the list of system elements to provide for
           output from all important items.
"
calendar addToListOfSystemElements: machine1;
         addToListOfSystemElements: machine2;
         addToListOfSystemElements: inspection;
         addToListOfSystemElements: assembly;
         addToListOfSystemElements: term1;
         addToListOfSystemElements: term2.
"
   Step 6: Schedule initial events (WFI arrival and end of
           simulation execution).
"
calendar schedule: [creat1 create] at: 0.
calendar schedule: [calendar end] at: 480.
```

```
"
   Step 7: Start the simulation model executing.
"
calendar eventInitiator.
```

The output produced as the result of executing the simulation model is structured to present all information on each object as a coherent unit. Figure 19 contains the output of one run of the simulation model just presented. If we look at the object named Machine 1 Station, for example, we see that the information provided includes station processing times statistics, station utilization statistics, and internal queue statistics (length and time in queue). A similar output format is followed for each object in the system with output structured to consider its particular composition (number of servers, number of queues, etc.).

Summary

This discussion has provided a basis for the choice of a hierarchical orientation for the construction of OOM simulation models. In addition, it has illustrated the ease with which OOM simulation models may be constructed, a product of the hierarchical orientation. A simple example system was used as a basis for discussion throughout the section concluding with the construction and execution of an OOM model for the example.

```
Calendar Statistics
-------------------

  Event List Length Information
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
  12.0634  1.1332     9.0000    1.0000   16.0000        2500

                          <<< O >>>

Machine 1 Station  (a Single Queue, Multiple Server ...
-------------------------------------------------------

  Processing Times Information
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        87     3.9595  0.5700    3.8554    3.0204    4.9619

  Cell upper                                          Percentage
    limit                                              of obser.
    [---------+---------+---------+---------+---------+
    [
3.00[                                                    0.0000
    [
3.20[*****                                               0.1149
    [
3.40[*****                                               0.1149
    [
3.60[****                                                0.0805
    [
3.80[**                                                  0.0460
    [
4.00[*********                                           0.1954
    [
4.20[***                                                 0.0690
    [
4.40[******                                              0.1379
    [
4.60[**                                                  0.0460
    [
4.80[*****                                               0.1034
    [
5.00[****                                                0.0920
    [
    [                                                    0.0000
```

Figure 19. One Set of Simulation Output for the OOM Simulation Model of the Example System

```
  Cell upper                                                  Percentage
    limit                                                      of obser.
    [---------+---------+---------+---------+---------+
    [
3.00[                                                            0.0000
    [
3.40[***********                                                 0.2299
    [
3.80[******                                                      0.1264
    [
4.20[*************                                               0.2644
    [
4.60[*********                                                   0.1839
    [
5.00[*********                                                   0.1954
    [
    [                                                            0.0000
```

Utilization Information
Time of initialization = 0.00
Current Time = 480

Avg Value	Std Dev	Curr Value	Min Value	Max Value	No. Changes
0.7177	0.4501	0.0000	0.0000	1.0000	91

Queue Length Statistics
Time of initialization = 0.00
Current Time = 480

Avg Value	Std Dev	Curr Value	Min Value	Max Value	No. Changes
0.8880	1.3993	0.0000	0.0000	6.0000	91

Time In Queue Statistics
Time of initialization = 0.00
Current Time = 480

Total Obs.	Avg Obs.	Std Dev	Last Obs.	Min Obs.	Max Obs.
45	9.4724	7.6877	1.9348	0.0817	28.5310

Figure 19. (Continued)

```
  Cell upper                                                  Percentage
    limit                                                      of obser.
    [---------+---------+---------+---------+---------+
    [
4.00[***************                                            0.3111
    [
5.00[                                                          0.0000
    [
6.00[**                                                        0.0444
    [
7.00[**                                                        0.0444
    [
8.00[***                                                       0.0667
    [
9.00[***                                                       0.0667
    [
10.0[****                                                      0.0889
    [
11.0[**                                                        0.0444
    [
12.0[                                                          0.0000
    [
13.0[**                                                        0.0444
    [
14.0[**                                                        0.0444
    [
    [*************                                             0.2444
```

<<< O >>>

Machine 2 Station (a Single Queue, Multiple Server ...

Processing Times Information

Time of initialization = 0.00
Current Time = 480

Total Obs.	Avg Obs.	Std Dev	Last Obs.	Min Obs.	Max Obs.
87	2.9893	0.6079	2.0346	2.0346	3.9698

Utilization Information

Time of initialization = 0.00
Current Time = 480

Avg Value	Std Dev	Curr Value	Min Value	Max Value	No. Changes
0.5418	0.4982	0.0000	0.0000	1.0000	125

Figure 19. (Continued)

```
  Queue Length Statistics
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.5571  1.1899     0.0000    0.0000    5.0000          57

  Time In Queue Statistics
       Time of initialization = 0.00
       Current Time           = 480
                                                    Total Obs.  Avg Obs
---------- ---------- ------- ---------  --------  --------
        28     9.5506  7.8853    0.6096    0.0027   28.0752

                          <<< O >>>

Inspection Station  (a Single Queue, Multiple Server ...
-------------------------------------------------------

  Processing Times Information
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
       174     1.4815  0.2988    1.7842    1.0017    1.9962

  Utilization Information
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.5358  0.4987     1.0000    0.0000    1.0000         214

  Queue Length Statistics
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.1081  0.3168     0.0000    0.0000    2.0000         139

  Time In Queue Statistics
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        69     0.7523  0.4151    0.4826    0.0026    1.7148

                          <<< O >>>
```

Figure 19. (Continued)

```
Assembly Station  (a Multiple Queue, Multiple Server ...
-----------------------------------------------------

  Processing Times Information
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        86     1.4960  0.2906    1.3149    1.0006    1.9909

  Utilization Information
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.2680  0.4429     0.0000    0.0000    1.0000         173

  Workorder Queue Information

  Queue Length Statistics
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.0164  0.1269     0.0000    0.0000    1.0000         173

  Time In Queue Statistics
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        86     0.0914  0.3721    0.0000    0.0000    1.9330

  Queue Number 1 Statistics

  Queue Length Statistics
       Time of initialization = 0.00
       Current Time           = 480
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.5393  0.7547     1.0000    0.0000    3.0000         174

  Time In Queue Statistics
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        86     2.9955  2.6445    1.3033    0.0000   11.8705

                          <<< O >>>
```

Figure 19. (Continued)

```
Final Assembly Terminator  (a Terminator Object)
-------------------------------------------------

  Time In System Statistics
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        86    13.0306  7.8713    6.9090    6.2509   36.1407

  Cell upper                                          Percentage
    limit                                              of obser.
     [---------+---------+---------+---------+---------+
     [
5.00[                                                    0.0000
     [
7.00[******                                              0.1279
     [
9.00[**********************                              0.4535
     [
11.0[*                                                   0.0233
     [
13.0[*                                                   0.0349
     [
15.0[*                                                   0.0233
     [
17.0[****                                                0.0930
     [
19.0[*                                                   0.0349
     [
21.0[**                                                  0.0465
     [
23.0[*                                                   0.0349
     [
25.0[*                                                   0.0349
     [
     [****                                               0.0930

                          <<< O >>>

Assembled WFIs Terminator  (a Terminator Object)
-------------------------------------------------

  Time In System Statistics
       Time of initialization = 0.00
       Current Time           = 480
Total Obs.  Avg Obs.  Std Dev Last Obs.  Min Obs.  Max Obs.
---------- ---------- ------- ---------  --------  --------
        86    13.0306  7.8713    6.9090    6.2509   36.1407

                          <<< O >>>
```

Figure 19. (Concluded)

Smalltalk Class Implementation

In the first section of this chapter, the different OOP classes needed within the prototype OOM system were described in terms of the functions each must support to effectively simulate the operation of objects in the system. In the second section, the concepts used to design the simulation model structure and specification methods were described. In addition, the impact of the application of these concepts on the simulation model top level appearance was illustrated through the complete development of a simulation model of a simple system. For the sake of brevity, this section shall discuss in some detail the implementation of several representative simulation processing objects, including, Calendar, Work Flow Item, Routing, Random Generator, and one simulation element object, the Multiple-Queue, Multiple-Server Processing Station. This discussion is intended to guide the reader to the development of a basic understanding of the structure of the simulation software. The Smalltalk implementations of each of these classes are available for detailed examination in Appendix A along with all of the other class implementations.

Any OOP class definition provides information on four specific elements about the class. These include: 1) Class variable names, 2) Instance variable names, 3) Class methods, and 4) Instance methods. Class variables are data

storage locations which are allocated once and are associated with a class. Instance variables are data storage locations which are allocated uniquely for each instance of a class. Instances of the same class will have the same instance variable allocations, but, most probably, will have different values stored in their own locations. Class 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. Instance 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. The remaining text in this section discusses in detail each of these four aspects of the five classes discussed.

Specific Simulation Processing Objects

As mentioned previously, the Calendar object in a simulation run acts as the central processor or controller for the operation of the dynamic simulation model. As such, understanding of the implementation of the Calendar class is an important conceptual and operational requirement. The definition of the Calendar class provides the following breakout:

Data storage

Class variable names:

--

Instance variable names:

calendarHead
calendarTail
currentTime
debug
listLength
listOfSystemElements

The instance variables, calendarHead and calendarTail, provide references to the first entry and last entry in the event list, which is implemented as a doubly linked list. currentTime is a storage location for the value of the current simulated time. The variable debug takes the values true and false. When debug is set to true, the simulation operation will halt after the execution of each event. This feature allows the simulation analyst to use Smalltalk inspector windows to completely debug the operation of an object or model. listLength is a storage location which tracks statistics on the length of the event calendar. listOfSystemElements stores references to elements in the simulation model in an OrderedCollection instance (a class definition already available in Smalltalk). When the simulation is completed, this list of elements is used to prompt each object in the system for output.

Software methods (boldface items are arguments):

Class methods:

1) new

2) newEndTime: **aTime**

The class method new creates a new instance of the Calendar class (with all instance data storage allocated) and calls the initialize instance method (see below) to set initial values. It then returns the pointer to the new, initialized Calendar instance as the result of its operation. The newEndTime: method performs a similar activity with the addition of automatically scheduling the end of simulation event at the requested time.

Instance methods:

1) addToListOfSystemElements: **newElement**
2) arrayExecute: **anArray**
3) clearStatistics
4) contextExecute: **aContext**
5) end
6) eventInitiator
7) getTime
8) initialize
9) output
10) printResults
11) removeEvent: **anEvent**
12) schedule: **anObject** at: **intervalTime**
13) setDebug: **aBoolean**
14) setListOfSystemElements: **anOrderedCollection**
15) setTime: **newTime**

The addToListOfSystemElements: method provides the calendar object with the ability to add new objects to the system element list. The two methods arrayExecute: and contextExecute: handle the two types of event specification methods (events are specified either in the form of an Array or as a Context, two classes in Smalltalk). clearStatistics is a method which is typically scheduled to execute at some specific time (to remove the effects of a system warm up period) by the model developer. The method collects

references to all statistics collection objects in the simulation model and requests each of these objects to reinitialize itself. The end method removes all events from the calendar and causes the simulation execution to end. This method can also be scheduled by the model developer to execute at some simulated future time. eventInitiator is the method which controls the execution of events during the simulation run. It repeatedly loops through the process of removing the first event from the calendar and causing it to execute. When no further events are on the calendar, this method calls for simulation execution output. getTime simply returns the current value of simulated time to the calling object. initialize sets initial calendar instance variable values during the calendar instance creation activity. output prompts each system simulation element listed in the model element list (the instance variable listOfSystemElements) to produce statistical results on its activities. This method operates by assuming that classes used in the simulation model have been constructed with a class specific printResults method which will print the appropriate results for each object. The printResults method outputs statistics on the calendar's operation during the simulation execution. The removeEvent method uses an event specifier (array or context format) to exactly match and remove an event from the calendar list. When an event is to be placed on the calendar list, the method schedule:at: is used. Arguments needed are the event

specifier and an interval of time until the event should occur. The setDebug: method sets the value of the debug instance variable used to control the occurrence of inspection halts in simulation execution. setListOfSystemElements: and setTime: are lower level methods used for model experimentation and not typically executed during a standard model run. From this discussion, one can see that these capabilities provide for a full featured basis for the major simulation component, the calendar. The Smalltalk code, fully commented, provides the maximum amount of detail and is available in Appendix A.

Dropping down from the high level calendar object, one of the lower level simulation processing objects is the Work Flow Item class. Instances of this class represent work items in the simulated system and contain the data and methods needed to emulate their passive operation. The definition of the Work Flow Item class provides the following breakout:

Data storage

Class variable names:

--

Instance variable names:

creationTime
wfiLabel
workOrderType
workOrderNumber
routing
flowTimeMarkers

The creationTime instance variable is set equal to the simulated time of creation for each Work Flow Item instance allocated. The storage of this value allows the time in system statistics to be collected for each work flow item passing through the system. The wfiLabel storage location is merely a character string label used to specify the type of item represented by the work flow item instance. workOrderType is an instance variable which stores a string indicating the type of work order with which a particular work flow item is associated. workOrderNumber is an integer set during the work order creation operation and is provided to allow for matched assembly of work flow items from the same work order. The workOrderNumber is unique to the group of work flow items from each work order created during a simulation execution. The routing instance variable stores a reference to the routing used by the work flow item to guide its progress through the simulated system. Note that the Routing class will be discussed in detail next. flowTimeMarkers is a storage location pointing to a Dictionary instance (a general Smalltalk class). This dictionary functions to allow subsystem flow times to be monitored and collected. As a work flow item passes through a flow time collection object (another simulation class) for the first time, it is sent a message to add a flowTimeMarkers entry with the key being the flow time collection object itself and the storage value being the current simulated time. On the second pass through the flow

time collection object this marked time entry is removed and used to calculate an observation of subsystem flow time.

Software methods (boldface items are arguments):

Class methods:

1) new

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

Instance methods:

1) > **aWFI**
2) flowTimeMark: **anObject**
3) getCreationTime
4) getFlowTimeMark: **anObject**
5) getNextAccessCode: **anObject**
6) getProcessingTime
7) getSetupTime
8) getWFILabel
9) getWorkOrderNumber
10) getWorkOrderType
11) initialize
12) operationCompleted
13) routingEmpty
14) setRouting: **aRoutingObject**
15) setWFILabel: **aString**
16) setWorkOrderNumber: **aNumber**
17) setWorkOrderType: **aString**

The > instance method is used when a work flow item is placed into a queue. At this point in the OOM system development the FIFO queue discipline is the only one supported. This method returns a Boolean false, which forces a newly entering work flow item to be placed at the end of the current queue. flowTimeMark: is the method used by flow time collection objects during the first pass a

work flow item makes through the collection object. As mentioned previously, it provides for the addition of a marked time dictionary entry in the flowTimeMarkers instance variable. getFlowTimeMark: is the method which handles the second pass of a work flow item through a flow time collection object. It retrieves the time marker, removes the entry from the time marker dictionary, and returns the time marker to the flow time collection object. getCreationTime simply returns the value of the creationTime instance location to the calling object. The method getNextAccessCode: is used by a simulation element object, which has just finished processing a work flow item, to retrieve the designator for the next processing location for the work flow item. This designator is used to set up the arrival event, on the calendar, of the work flow item to the simulation model element. getProcessingTime returns to the calling object, generally an element object, a Context which specifies the processing time in the form of some random variable distribution. The element object will use this Context to generate a specific processing time value. In a similar manner, the getSetupTime instance method operates to provide a specific setup time value for an element object to use. The methods getWFILabel, getWorkOrderNumber, and getWorkOrderType return the string and numeric values for the previously described instance variables. initialize is used to set the value for the creationTime instance variable and to set up a new, empty Dictionary instance in the

flowTimeMarkers instance variable. The operationCompleted method is used by element objects to prompt a work flow item to remove the first routing operation from its routing. Note that this first routing operation is the one which refers to the element object currently controlling the work flow item. routingEmpty is a method which tests for an empty routing list and returns a Boolean true or false. setRouting:, setWFILabel:, setWorkOrderNumber:, and setWorkOrderType: are all methods which set the work flow item's appropriate instance variable values to the passed argument.

Mentioned many times previously, the Routing class defines the structure of routing objects which specify the path a work flow item will take through a simulated system and which are subcomponents of work flow items. The components of the class specification are:

Data storage

Class variable names:

--

Instance variable names:

listOfOperations

The only instance variable for this class, listOfOperations, is an OrderedCollection instance which stores each routing operation of a work flow item. Each routing operation contained in this list specifies the element object name and access code, processing time

distribution, and setup time distribution.

Software methods (boldface items are arguments):

Class methods:

1) new

The new method functions to allocate memory for a new routing instance and to initialize the listOfOperations instance variable to an empty OrderedCollection.

Instance methods:

1) addOperation: **anObject** key: **aKeyValue**
2) addOperation: **anObject** key: **aKeyValue**
   processingTime: **aContext** setupTime: **aContext**
3) addOperation: **aContext** processingTime: **aContext**
   setupTime: **aContext**
4) at: **aNumber**
5) copyOperation: **aRoutingOperation**
   processingTime: **aContext** setupTime: **aContext**
6) copyRouting
7) removeFirst
8) setListOfOperations: anOrderedCollection

The first three methods listed, addOperation:key:, addOperation:key:processingTime:setupTime:, and addOperation:processingTime:setupTime: function to create a new operation specification and add this operation to the routing list. The addOperation:key: method creates a new operation with the element object name and access code, but without processing and setup time specifiers. The addOperation:key:processingTime:setupTime: method creates a similar operation specification but with processing and setup time specifiers. Both of these two methods use the object reference and key passed as arguments to prompt a simulation model object for the appropriate arrival event

initiation access code. In this way, the model developer is not required to know the form of the arrival event code for each simulation element object because the element itself will supply the information in response to a standard message. In contrast to these two methods, the first argument for the addOperation:processingTime:setupTime method is required to be the completed access code for the appropriate simulation model object. The at: method returns the routing operation located at the argument specified position in the routing list.
copyOperation:processingTime:setupTime: is an internally used method (called from copyRouting) which simply copies the contents of a routing operation into another memory location. The copyRouting method is used to completely copy an entire routing. This method is used when a work order is created and the routings for each of the parts in the work order must be recreated and attached to the work flow items. removeFirst is used to completely remove the first operation from a routing and is typically executed when a simulation system element has completed processing of a work flow item. setListOfOperations: is an environment development method used to set the listOfOperations instance variable to a completed OrderedCollection list (for experimentation during model development).

One capability alluded to and critical for stochastic simulation is fulfilled through the Random Generator class. The Random Generator class forms the root of a subtree in

the class hierarchy which provides for the generation of random variable observations. The structure of this class is as follows:

Data storage

Class variable names:

LastSeed

Default seed numbers for random variable observation generation are themselves generated through a separate linear congruential generator. The value of the LastSeed class variable itself acts as the seed for simulation element seed number generation. In this manner, the model developer does not need to specify seed values for each of the various simulation model components. As new seeds are generated and assigned, the value of LastSeed is assigned to the most recent one generated.

Instance variable names:

seed

The seed instance variable is used by a Random Generator instance as the basis for the random number stream that it can produce. The storage allocation is also inherited by descendents of the Random Generator class which use it in a similar manner.

Software methods (boldface items are arguments):

Class methods:

getLastSeed
new
new: **aSeedValue**

setLastSeed: **aSeedValue**

The getLastSeed and setLastSeed: class methods accomplish the activities of retrieving and setting the LastSeed value for the requestor. The new and new: methods provide for the allocation of a new Random Generator instance. With the new method, the seed value for the newly created instance is generated using the LastSeed value. With the new: method, the seed value for the Random Generator instance is provided as an argument.

Instance methods:

1) bernoulliMean:
2) discreteUniformHigh: **aNumber** low: **aNumber**
3) exponentialLambda: **aNumber**
4) initializeSeed
5) lognormalMu: **aNumber** sigma: **aNumber**
6) normalMu: **aNumber** sigma: **aNumber**
7) setSeed: **aSeedValue**
8) triangularHigh: **aNumber** low: **aNumber** mode: **aNumber**
9) uniformHigh: **aNumber** low: **aNumber**
10) weibullAlpha: **aNumber** beta: **aNumber**
11) zeroOneUniformRV

The methods bernoulliMean:, discreteUniformHigh:low:, exponentialLambda:, lognormalMu:sigma:, normalMu:sigma: triangularHigh:low:mode:, uniformHigh:low:, and weibullAlpha:beta: all provide an instance of the Random Generator class with the ability (via the inverse transform method) to generate an observation from the particular distribution. Notice that the necessary parameter values must be supplied as part of the message. initializeSeed is the method used when a new Random Generator instance seed value is to be generated from the class variable, LastSeed. setSeed: is used when a specific seed value has been

supplied as an argument. zeroOneUniformRV uses a linear congruential generator and a seed value to generate a (0,1) uniform random variable. This method is typically called by the previously mentioned random number generation methods during the random variable generation process. Note that this method is also available to instances of subclasses of Random Generator, which are also able to produce random variable observations.

Specific Simulation Element Object

One of the moderately complex simulation element objects implemented is the Multiple-Queue, Multiple-Server Processing Station (MQMSPS) class. This class emulates a system component which has one or more prioritized queues of parts which wait for simple processing (ie. no assembly or matching takes place). In addition, there can be one or more identical, parallel servers which have the ability to take breaks or break down according to some distribution. Each of these servers is represented by an instance of another type of class, the Basic Human Worker class. The class structure is:

Data storage

Class variable names:

--

Instance variable names:

queues
name
randomGenerator

```
inputCode
partsBeingWorkedOn
workerStatus
busyStatus
procTimes
numberOfServers
numberOfQueues
endOfServiceEvents
```

As discussed, the instance variables for a simulation element object provide the data and status storage which allows the element object to simulate the desired system component. queues is the instance variable location which stores the OrderedCollection list of the one or more queues associated with the station. name contains the character string which will be used to identify results output produced after the simulation execution. The variable randomGenerator stores the reference to the Random Generator instance used by the simulation element for processing time observation generation. inputCode is initialized at MQMSPS creation to the partially completed arrival event access code. For the MQMSPS class, this access code is an array with the first element being the instance itself and the second element being the method designator #partArrival:withPart:withCallingObj. Notice that when the event is actually placed on the calendar that a work flow item and a calling object will be provided to complete the event execution request. partsBeingWorkedOn is another OrderedCollection instance which maintains the reference to work flow items undergoing processing (busy server) at each of the multiple servers or to "nil" when a particular server is idle. workerStatus is an OrderedCollection instance in

which each element is a reference to the Basic Human Worker instance representing the corresponding parallel server. This data allows for communication from the top level MQMSPS object to its Basic Human Worker subcomponents. The instance variable busyStatus is a statistics collection location (a Time Tracked Number) which is used to collect the utilization information for the station. procTimes is another statistics collection instance variable (an Observation Tracked Number) which is used to collect as observations the processing times for all work flow items handled by the MQMSPS instance. numberOfServers and numberOfQueues are simply numbers indicating the number of servers and queues allocated to an instance of the MQMSPS class. endOfServiceEvents is an OrderedCollection, with each element corresponding to a parallel server, of all end of service events currently on the calendar. This information is used to halt processing of work flow items when a parallel server goes inactive (such as for human rest activities or machine breakdowns). When an activity has been halted, the remaining processing time is stored in this instance variable.

Software methods (boldface items are arguments):

Class methods:

1) newSeed: **aSeedValue** withName: **aString** andServers: **aNumber** andQueues: **aNumber**
2) newWithName: **aString** andServers: **aNumber** andQueues: **aNumber**

The newSeed:withName:andServers:andQueues: is a method which sets up a new instance of the MQMSPS class by using

the information provided in the arguments. This information includes a specific seed value for the object's random generator instance, an identifier string, and the numbers of servers and queues. newWithName:andServers:andQueues: is a similar method except the seed value for the random generator is produced using the LastSeed value at the Random Generator class.

Instance methods:

1) checkEvent: **aServerNumber**
2) checkQueue: **aServerNumber**
3) getFirstPart
4) initializeSeed: **aSeedValue** andServers: **aNumber** andQueues: **aNumber**
5) initializeServers: **aNumber** andQueues: **aNumber**
6) partArrival: **aQueueKey** withPart: **aWFI** withCallingObj: **anObject**
7) partDeparture: **aServerNumber**
8) printResults
9) returnLinksOnKey: **aQueueKey**
10) setName: **aString**
11) setWIPAggregator: **aWIPAgg** atQueueNumber: **aQueueKey**

The checkEvent: method is used by Basic Human Worker instances within the MQMSPS to signal the station that they are transitioning from active to inactive. This results in the removal from the calendar of a pending end of service event and calculation and storage of the remaining processing time. In the meantime, the Basic Human Worker has scheduled an inactive to active transition time on the calendar and the checkqueue: method is used to signal the MQMSPS that the server is returning to active status. This method either restarts a halted process, pulls a new work flow item from the queues, or leaves the server idle. The getFirstPart method is used to correctly remove work

flow items from the prioritized queues and returns the removed work flow item's reference to the calling method. The two methods, initializeSeed:andServers:andQueues: and initializeServers:andQueues:, are called by the previously mentioned class methods to correctly initialize a new MQMSPS instance. partArrival:withPart:withCallingObj: is the arrival event method. Note that the first parameter in the message is the queue key. This value tells the MQMSPS instance which prioritized queue the arriving part must enter. partDeparture: is the end of service event method and the argument is the number of the server which has completed processing. Notice that this information is entered on the event list by the MQMSPS object at the time service is initiated. printResults is the method needed for each simulation element object. This version of the method has been design specifically for instances of the MQMSPS class and prints out all of the statistical results of a simulation execution. The method, returnLinksOnKey:, is accessed during the routing construction phase of model execution. This method returns the appropriate arrival event access code to the calling location for inclusion in a work flow item routing. setName: merely sets the value of the name instance variable equal to the string argument contained in the message. setWIPAggregator:atQueueNumber: allows the model developer to associate each of the one or more queues in a MQMSPS instance with a WIP Aggregator instance. The function of a WIP Aggregator is to combine

the time based and observation based statistical characteristics of one or more queues into one object. This allows the model developer to treat one system WIP location as a composite of queues for different stations and still get information on the modeled WIP location as one unit.

Summary

This coverage of representative classes from the Smalltalk prototype simulation environment was intended to provide a basis with which the reader might peruse and understand the code listed in Appendix A. The next section illustrates the use of the developed classes in the construction of a simulation model for the target system (see chapters V and VI).

Target System Simulation

Model Representation

As mentioned several times previously, the target system for OOM model development is an electronics manufacturer kitting operation. The diagram for the system can be seen in Figures 7 and 12. Items which are processed through the system enter as a collection of parts (selects, bulk, and reels) and paper work which must be prepared and checked for the assembly operation. Figure 20 is how a simulation modeler would view the system as a collection of separate, yet interacting, objects. The numbers 1 through 9 have been added to the figure to provide links with the discussion of the simulation element objects needed to model

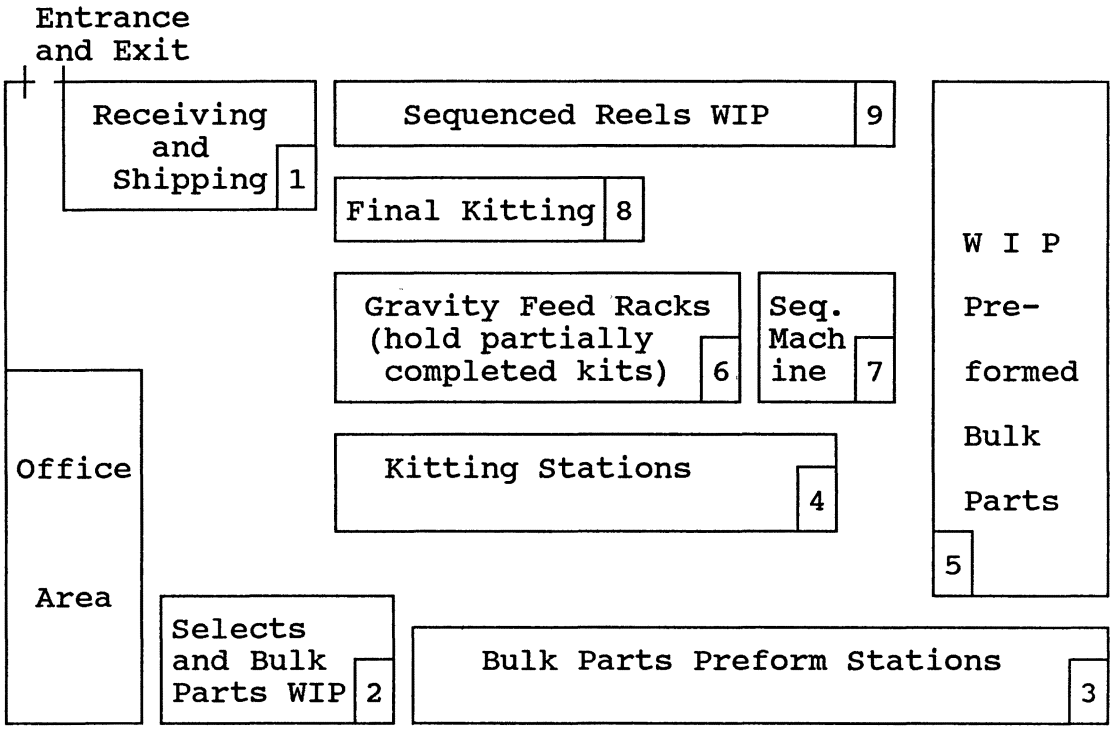


Figure 20. Object Oriented Electronics Kitshop Diagram (Note that the office area is not considered as part of the manufacturing model)

the system. The system element objects are as follows:

1) Receiving and Shipping - Because there are two queues of items to be processed (incoming and outgoing) and an assembly operation is not performed, the service activity of this station shall be represented by a Multiple-Queue, Multiple-Server Processing Station. In addition, the amount of WIP located in this area (contents of the two queues) shall be tracked through the use of a WIP Aggregator.

2) Selects and Bulk Parts WIP - Parts in this area are waiting for processing by two separate stations, kitting and

bulk parts preform. Therefore, a WIP Aggregator shall be used to collect data on the amount of WIP contained in the location (in the two separate simulation queues).

3) Bulk Parts Preform Stations - These stations are parallel servers performing a processing (not assembly) activity. There is one queue of work flow items waiting for processing (associated with 2 above). These characteristics allow the stations to be represented through the use of a Single-Queue, Multiple-Server Processing Station.

4) Kitting Stations - These work stations perform the majority of the kitting operation. At these stations, three components of the kit, preformed bulk parts, selects, and work order paperwork, are matched together and checked for completeness. In order to represent this portion of the system, a Multiple-Queue, Multiple-Server Assembly Station shall be used. This simulation element object provides for the "assembly" of work flow items in the simulation by matching work order numbers.

5) WIP for Preformed Bulk Parts - As another WIP location containing multiple queues (unsequenced reels and preformed bulk parts) this system object shall be represented in the model through the use of a WIP Aggregator.

6) Gravity Feed Racks - This WIP location contains partially completed kits waiting in one queue for the final

kitting operation. For the sake of uniformity, this queue shall also be represented in the model as a WIP Aggregator.

7) Sequencing Machine - There is one sequencing machine in the system. This processing station takes reels of unsequenced axially leaded components and combines the components in the correct order to yield sequenced reels ready for the insertion activity (outside the boundaries of this target system). A Single-Queue, Multiple-Server Processing Station provides the ability to simulate this system activity.

8) Final Kitting - In this operation, a human worker combines the partial kits from the gravity feed racks with the waiting sequenced reels to result in the fully completed kit. This operation requires the ability to match work order items from two queues, therefore, a Multiple-Queue, Multiple-Server Assembly Station shall be used.

9) Sequenced Reels WIP - This WIP location is associated with one queue in the system, that of the sequenced reels waiting for final kitting. A WIP Aggregator shall again be used in the simulation model.

In developing the simulation model for this system, this basic set of ten objects shall be augmented with the necessary simulation processing objects to result in the complete simulation model. Note that this set of element objects simulate the activities in the kitting system while

the simulation processing objects will allow for the creation, termination, and routing of work flow items.

Let us now consider the simulation processing objects which must be added to complete the model. First of all, there must be a way to generate arrivals to the system. This function shall be provided through the use of a creator for each work order type processed through the facility (3 representative types). Next, of course, we must provide for the termination of work flow items by including Terminators in the model. In order to provide information on each work order type, a Work Order instance must be created along with a Routing instance for each work flow item that is part of the work order. Finally, a Calendar object must be provided for the simulation model.

In the case of the target system, the following types and quantities of processing objects are needed:

1) Calendar - One required.

2) Work Order Creator - One for each work order type, three total.

3) Work Order - One for each work order type, three total.

4) Routing - One for each work flow item type for each work order type. With the work flow item types of: work order paperwork, bulk parts, selects, and reels for each work order type, twelve are necessary.

5) Terminator - One for each work flow item type in the system, four total.

The complete OOM representation is illustrated in Figure 21.

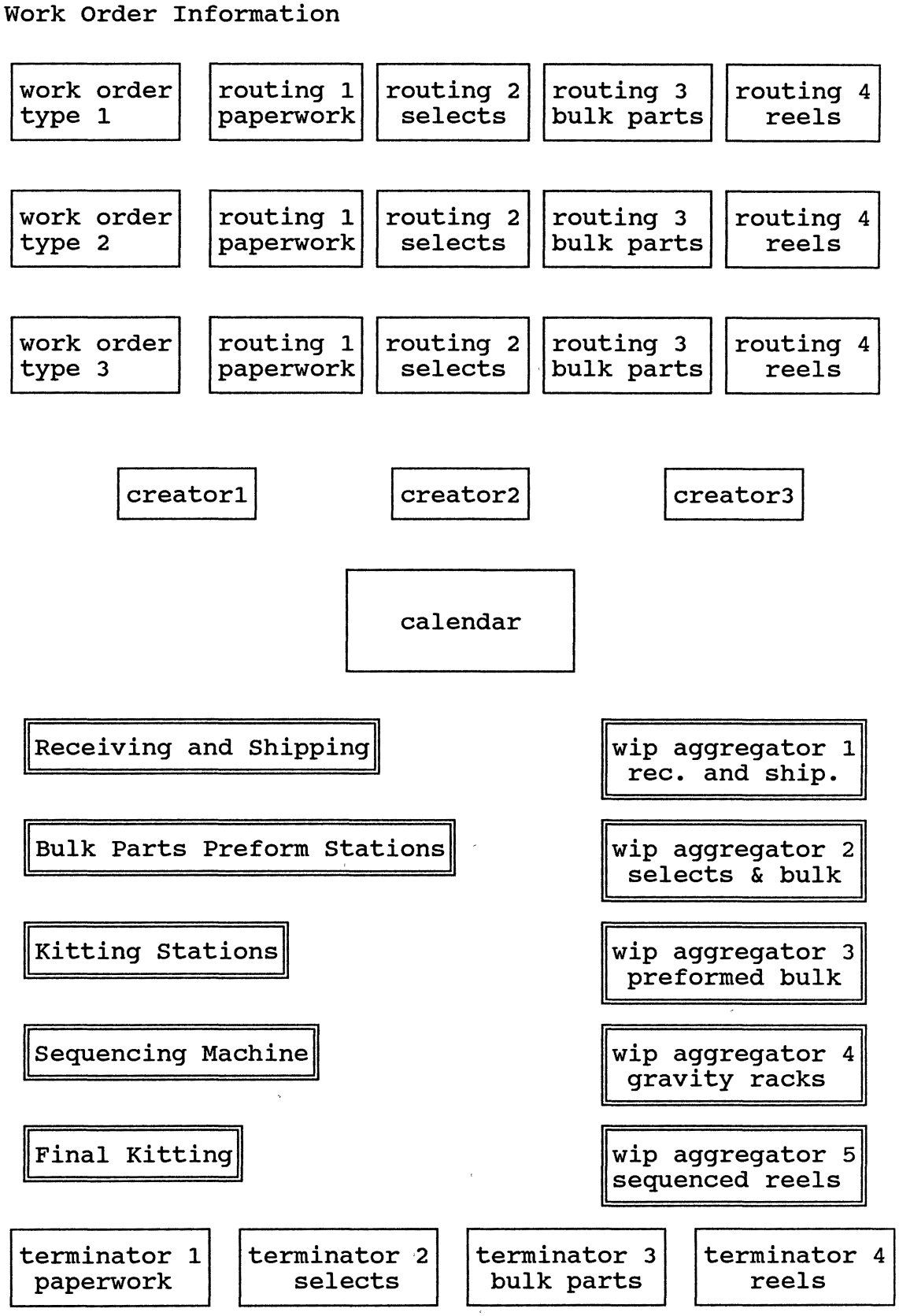


Figure 21. Complete Target System OOM representation

Using this information along with work order routing (depicted in Figure 22) and processing times information,

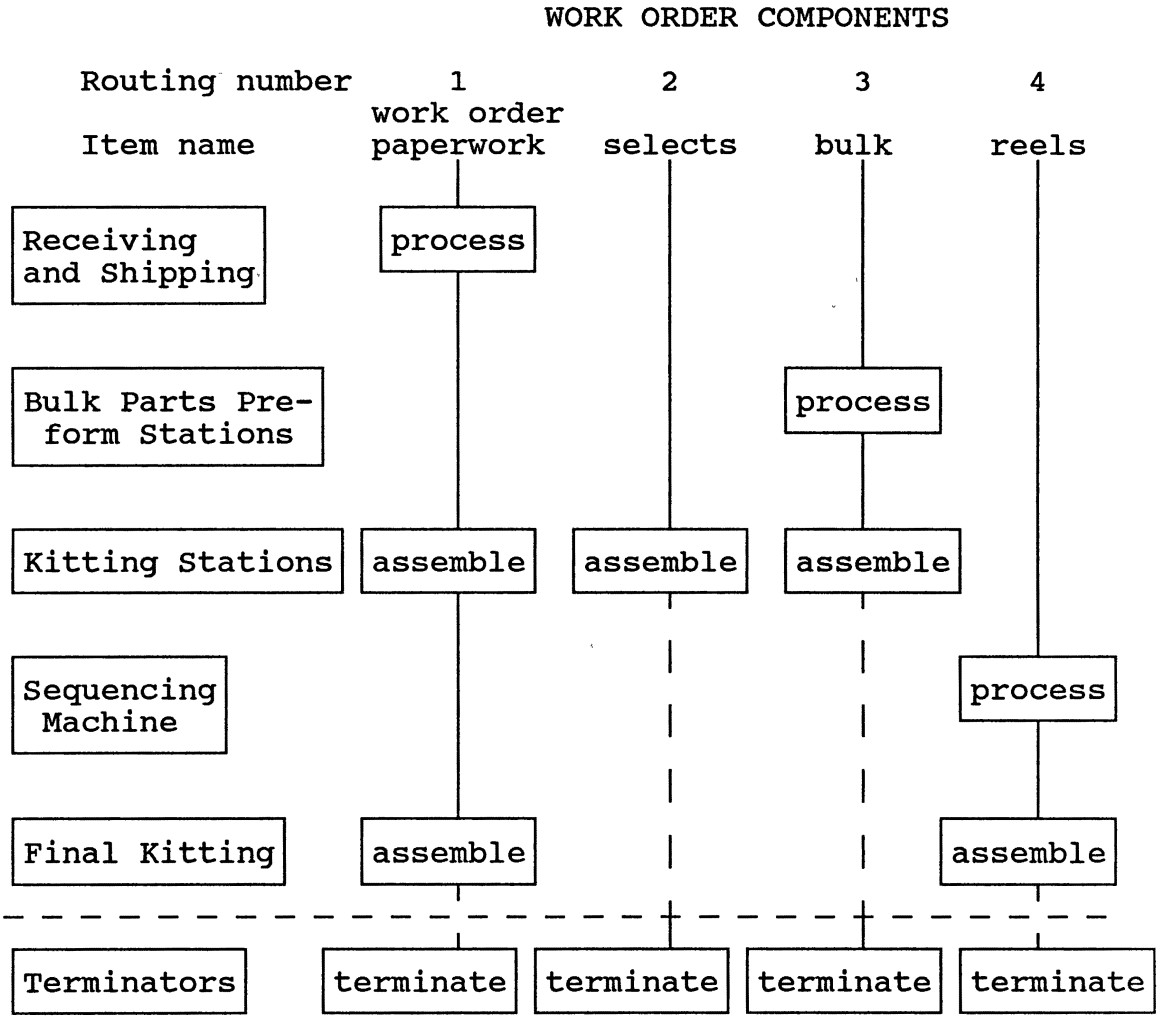


Figure 22. Work Order Routings

and the OOM development steps, the following OOM model implementation was created (note that time units are stated in hours):

```
"
 Electronics Kitshop Simulation Model
   Step 1: Set up top level instance variables.
"
|calendar

 creator1  "Creator for Type 1 Work Orders"
 creator2. "Creator for Type 2 Work Orders"
 creator3  "Creator for Type 3 Work Orders"

 workOrder1 w1r1 w1r2 w1r3 w1r4 "Info for Type 1 WO"
 workOrder2 w2r1 w2r2 w2r3 w2r4 "Info for Type 2 WO"
 workOrder3 w3r1 w3r2 w3r3 w3r4 "Info for Type 3 WO"

 woTerm       "Terminator for work order paperwork"
 selectsTerm  "Terminator for selects"
 bulkTerm     "Terminator for bulk parts"
 reelsTerm    "Terminator for reels"

 wipAgg1  "WIP Aggregator for Receiving and Shipping Queues"
 wipAgg2  "WIP Aggregator for Selects and Bulk Parts WIP"
 wipAgg3  "WIP Aggregator for Preformed Bulk Parts WIP"
 wipAgg4  "WIP Aggregator for Gravity Feed Racks"
 wipAgg5  "WIP Aggregator for Sequenced Reels WIP"

 m1  "MQMSP Station representing Receiving and Shipping"
 m2  "SQMSP Station representing Bulk Parts Preform"
 m3  "MQMSA Station representing Kitting"
 m4  "SQMSP Station representing Sequencing"
 m5  "MQMSA Station representing Final Kitting"
|
"
   Step 2: Set up a new Calendar class instance.
"
calendar:= Calendar new.
"
   Step 3: Set up instances of classes to represent the
           physical system being modeled.  Create instances
           for terminators.
"
wipAgg1:= WIPAggregator newWithName: 'Rec. and Ship WIP'.

wipAgg2:= WIPAggregator newWithName: 'Selects and Bulk WIP'.

wipAgg3:= WIPAggregator newWithName: 'Bulk Preform WIP'.

wipAgg4:= WIPAggregator newWithName: 'Gravity Racks WIP'.

wipAgg5:= WIPAggregator newWithName: 'Sequenced Reels WIP'.

m1:= MQueueMServerProc newWithName: 'Receiving and Shipping'
        andServers: 1 andQueues: 2.
m1 setWIPAggregator: wipAgg1 atQueueNumber: 1.
m1 setWIPAggregator: wipAgg1 atQueueNumber: 2.
```

```
m2:= SQueueMServerProc newWithName: 'Bulk parts preform'
        andSize: 6.
m2 setWIPAggregator: wipAgg2.

m3:= MQueueMServerAssem newWithName: 'Kitting Stations'
        andServers: 5 andQueues: 2.
m3 setWIPAggregator: wipAgg2 atQueueKey: 1.
m3 setWIPAggregator: wipAgg3 atQueueKey: 2.

m4:= SQueueMServerProc newWithName: 'Sequencing Station'
        andSize: 1.
m4 setWIPAggregator: wipAgg3.

m5:= MQueueMServerAssem newWithName: 'Final Kitting'
        andServers: 1 andQueues: 1.
m5 setWIPAggregator: wipAgg4 atQueueKey: 'workOrderQueue'.
m5 setWIPAggregator: wipAgg5 atQueueKey: 1.

bulkTerm:= Terminator newWithName: 'Bulk Parts Terminator'.

selectsTerm:= Terminator newWithName: 'Selects Terminator'.

reelsTerm:= Terminator newWithName: 'Reels Terminator'.

woTerm:= Terminator newWithName: 'WorkOrders Terminator'.
"
   Step 4: Set up routings and creators.

   Note that each work order is set up with routings for
   work order paperwork, selects, bulk, and reels in that
   order.
   ***First work order type***
"
workOrder1:= WorkOrder newWorkOrderType: 'WO Type 1'.

w1r1:= Routing new.  "Work Order Paperwork Routing"
w1r1
 addOperation: m1 key: 1
   processingTime: [:rg | rg triangularHigh: 0.5 low: 0.167
                           mode: 0.25]
   setupTime: [];
 addOperation: m3 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 3.1 low: 0.6
                           mode: 1.2]
   setupTime: [];
 addOperation: m5 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 0.7 low: 0.167
                           mode: 0.5]
   setupTime: [];
 addOperation: m1 key: 2
   processingTime: [:rg | rg triangularHigh: 0.33 low: 0
                           mode: 0.167]
   setupTime: [];
 addOperation: woTerm key: nil.
```

```
w1r2:= Routing new.  "Selects Routing"
w1r2
 addOperation: m3 key: 1;
 addOperation: selectsTerm key: nil.

w1r3:= Routing new.  "Bulk Routing"
w1r3
 addOperation: m2 key: nil
   processingTime: [:rg | rg triangularHigh: 3.0 low: 1.0
                           mode: 2.0]
   setupTime: [];
 addOperation: m3 key: 2;
 addOperation: bulkTerm key: nil.

w1r4:= Routing new.  "Reels Routing"
w1r4
 addOperation: m4 key: nil
   processingTime: [:rg | rg triangularHigh: 0.6 low: 0.3
                           mode: 0.4]
   setupTime: [];
 addOperation: m5 key: 1;
 addOperation: reelsTerm key: nil.

workOrder1
   addComponentWFI: 'workOrder' andCWFIRouting: w1r1;
   addComponentWFI: 'selects'   andCWFIRouting: w1r2;
   addComponentWFI: 'bulk'      andCWFIRouting: w1r3;
   addComponentWFI: 'reels'     andCWFIRouting: w1r4.

creator1:= WOCreator newWithWorkOrder: workOrder1
   timeBetweenCreationsGenerator:
   (Uniform newHigh: 2.0 low: 0).
"
   ***Second work order type***
"
workOrder2:= WorkOrder newWorkOrderType: 'WO Type 2'.

w2r1:= Routing new.  "Work Order Paperwork Routing"
w2r1
 addOperation: m1 key: 1
   processingTime: [:rg | rg triangularHigh: 0.5 low: 0.167
                           mode: 0.25]
   setupTime: [];
 addOperation: m3 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 3.1 low: 0.7
                           mode: 1.4]
   setupTime: [];
 addOperation: m5 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 0.8 low: 0.26
                           mode: 0.55]
   setupTime: [];
```

```
 addOperation: m1 key: 2
   processingTime: [:rg | rg triangularHigh: 0.33 low: 0
                           mode: 0.167]
   setupTime: [];
 addOperation: woTerm key: nil.

w2r2:= Routing new.  "Selects Routing"
w2r2
 addOperation: m3 key: 1;
 addOperation: selectsTerm key: nil.

w2r3:= Routing new.  "Bulk Routing"
w2r3
 addOperation: m2 key: nil
   processingTime: [:rg | rg triangularHigh: 5.0 low: 1.8
                           mode: 2.6]
   setupTime: [];
 addOperation: m3 key: 2;
 addOperation: bulkTerm key: nil.

w2r4:= Routing new.  "Reels Routing"
w2r4
 addOperation: m4 key: nil
   processingTime: [:rg | rg triangularHigh: 0.45 low: 0.2
                           mode: 0.27]
   setupTime: [];
 addOperation: m5 key: 1;
 addOperation: reelsTerm key: nil.

workOrder2
   addComponentWFI: 'workOrder' andCWFIRouting: w2r1;
   addComponentWFI: 'selects'   andCWFIRouting: w2r2;
   addComponentWFI: 'bulk'      andCWFIRouting: w2r3;
   addComponentWFI: 'reels'     andCWFIRouting: w2r4.

creator2:= WOCreator newWithWorkOrder: workOrder2
   timeBetweenCreationsGenerator:
   (Uniform newHigh: 3.0 low: 1.0).
"
   ***Third work order type***
"
workOrder3:= WorkOrder newWorkOrderType: 'WO Type 3'.

w3r1:= Routing new.  "Work Order Paperwork Routing"
w3r1
 addOperation: m1 key: 1
   processingTime: [:rg | rg triangularHigh: 0.5 low: 0.167
                           mode: 0.25]
   setupTime: [];

 addOperation: m3 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 3.8 low: 0.9
                           mode: 1.5]
   setupTime: [];
```

```
 addOperation: m5 key: 'workOrderQueue'
   processingTime: [:rg | rg triangularHigh: 0.5 low: 0.15
                           mode: 0.3]
   setupTime: [];
 addOperation: m1 key: 2
   processingTime: [:rg | rg triangularHigh: 0.33 low: 0
                           mode: 0.167]
   setupTime: [];
 addOperation: woTerm key: nil.

w3r2:= Routing new.  "Selects Routing"
w3r2
 addOperation: m3 key: 1;
 addOperation: selectsTerm key: nil.

w3r3:= Routing new.  "Bulk Routing"
w3r3
 addOperation: m2 key: nil
   processingTime: [:rg | rg triangularHigh: 5.0 low: 1.4
                           mode: 3.0]
   setupTime: [];
 addOperation: m3 key: 2;
 addOperation: bulkTerm key: nil.

w3r4:= Routing new.  "Reels Routing"
w3r4
 addOperation: m4 key: nil
   processingTime: [:rg | rg triangularHigh: 0.65 low: 0.3
                           mode: 0.5]
   setupTime: [];
 addOperation: m5 key: 1;
 addOperation: reelsTerm key: nil.

workOrder3
   addComponentWFI: 'workOrder' andCWFIRouting: w3r1;
   addComponentWFI: 'selects'   andCWFIRouting: w3r2;
   addComponentWFI: 'bulk'      andCWFIRouting: w3r3;
   addComponentWFI: 'reels'     andCWFIRouting: w3r4.

creator3:= WOCreator newWithWorkOrder: workOrder3
   timeBetweenCreationsGenerator:
   (Uniform newHigh: 9.0 low: 1.0).
"
   Step 5: Set the list of system elements to provide for
           output from all important items.
"
calendar addToListOfSystemElements: m1;
         addToListOfSystemElements: m2;
         addToListOfSystemElements: m3;
         addToListOfSystemElements: m4;
         addToListOfSystemElements: m5;
         addToListOfSystemElements: wipAgg1;
         addToListOfSystemElements: wipAgg2;
         addToListOfSystemElements: wipAgg3;
```

```
          addToListOfSystemElements: wipAgg4;
          addToListOfSystemElements: wipAgg5;
          addToListOfSystemElements: woTerm;
          addToListOfSystemElements: selectsTerm;
          addToListOfSystemElements: bulkTerm;
          addToListOfSystemElements: reelsTerm.
"
   Step 6: Schedule initial events (WFI arrival, clear
           statistics, and end of simulation execution.
"
calendar schedule: [creator1 create] at: 0;
          schedule: [creator2 create] at: 0;
          schedule: [creator3 create] at: 0;
          schedule: [calendar clearStatistics] at: 90;
          schedule: [calendar end] at: 360.
"
   Step 7: Start the simulation model execution.
"
calendar eventInitiator.
```

The results of executing this model in the Smalltalk V environment is shown in Figure 23. Note that as the purpose of the target system model development was proof of concept on a real world example, an analysis of the results of this model was not performed. On the other hand, 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.

The validation process for the OOM conceptual approach involved the development and validation of two separate OOM models, the first one being for the standard M/M/1 queueing

system and the second one being for a simplified representation of the target system. The analysis presented in Appendix B validates the OOM M/M/1 queueing model by comparing the results of several simulation runs to the analytical solution for the M/M/1 queueing system. The validation of the simplified target system OOM model was performed by comparing the results of the OOM model to the results of a model of the same system in a commonly used simulation language, SLAM II. The simplified target system was designed so that it would be completely representable in the network portion of SLAM and would have a high face validity in both simulation representations. These OOM and SLAM models were executed and an analysis of the results was performed. This analysis involved the use of t tests to compare the values of key measures of performance from both models. The comparisons indicate that the results of the OOM execution are not distinguishable from those of the SLAM model. This successfully validates the conceptual organization of the OOM prototype environment for the generation of discrete simulation models of manufacturing systems. The complete comparison and additional discussion is contained in Appendix B.

```
Calendar Statistics
-------------------
  Event List Length Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
  41.0603  4.3184    42.0000   31.0000   50.0000       27160
                        <<< O >>>

Receiving and Shipping  (a Multiple Queue, Multiple Server
                         Processing Object)
-------------------------------------------------------------
  Processing Times Information
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       891     0.2355  0.1026     0.4236    0.0149    0.4848
     Utilization Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.7761  0.4169     1.0000    0.0000    1.0000         429
     Queue Number 1 Statistics
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.4301  0.6637     2.0000    0.0000    4.0000         809
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       403     0.2878  0.2508     0.0821    0.0010    1.3225
     Queue Number 2 Statistics
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   2.4360  2.6557     1.0000    0.0000   13.0000         786
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       392     1.6758  1.5267     0.5892    0.0076    7.3455
                        <<< O >>>
```

Figure 23. Target System Simulation Model Output

```
Bulk parts preform  (a Single Queue, Multiple Server
                     Processing Object)
-------------------------------------------------------------
     Processing Times Information
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- --------- -------  ---------  --------  --------
       449    2.4319  0.7587     1.7425    1.1248    4.9235
     Utilization Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   4.0251  2.0703     6.0000    0.0000    6.0000        1758
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.3126  0.7057     0.0000    0.0000    5.0000         361
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- --------- -------  ---------  --------  --------
       180    0.4689  0.4426     0.0093    0.0009    2.3535
                          <<< O >>>

Kitting Stations  (a Multiple Queue, Multiple Server
                   Assembly Object)
-------------------------------------------------------------
     Processing Times Information
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- --------- -------  ---------  --------  --------
       446    1.7014  0.5472     1.9649    0.7127    3.5233
     Utilization Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   2.8208  1.6646     3.0000    0.0000    5.0000        1717
     Workorder Queue Information
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   4.1262  1.3307     3.0000    1.0000    9.0000         893
```

Figure 23. (Continued)

```
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       446     2.4980  1.0148     3.0048    0.3735    6.1190
     Queue Number 1 Statistics
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   5.1655  1.5366     6.0000    2.0000   11.0000         896
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       446     3.1244  1.0029     3.4188    1.1248    7.0219
     Queue Number 2 Statistics
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.0676  0.2912     0.0000    0.0000    2.0000         893
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       446     0.0409  0.1475     0.0000    0.0000    1.3581
                        <<< O >>>

Sequencing Station  (a Single Queue, Multiple Server
                     Processing Object)
-------------------------------------------------------------
     Processing Times Information
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       447     0.3983  0.0872     0.5571    0.2098    0.6106
     Utilization Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.6579  0.4744     1.0000    0.0000    1.0000         457
```

Figure 23. (Continued)

```
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.6492  0.8662     2.0000    0.0000    5.0000         667
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       332     0.5276  0.3988     0.0664    0.0031    1.9580
                        <<< O >>>

Final Kitting  (a Multiple Queue, Multiple Server Assembly
                Object)
-------------------------------------------------------------
     Processing Times Information
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     0.4731  0.1193     0.5433    0.1857    0.7631
     Utilization Information
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.7804  0.4140     1.0000    0.0000    1.0000        1119
     Workorder Queue Information
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   2.5440  2.3314     3.0000    0.0000    9.0000         894
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     1.5378  1.2918     1.0874    0.0000    4.7321
     Queue Number 1 Statistics
     Queue Length Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   9.6732  2.5441     9.0000    5.0000   16.0000         893
```

Figure 23. (Continued)

```
     Time In Queue Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     5.8480  1.7123     6.4547    2.1160   11.7644
                        <<< O >>>

Rec. and Ship WIP  (a WIP Aggregator Object)
-------------------------------------------
     WIP Size Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   2.8661  2.7668     3.0000    0.0000   13.0000        1594
     Time In WIP Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       795     0.9722  1.2894     0.0821    0.0010    7.3455
                        <<< O >>>

Selects and Bulk WIP  (a WIP Aggregator Object)
----------------------------------------------
     WIP Size Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   5.4782  2.0977     6.0000    2.0000   16.0000        1256
     Time In WIP Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       626     2.3609  1.4891     0.0093    0.0009    7.0219
                        <<< O >>>

Bulk Preform WIP  (a WIP Aggregator Object)
------------------------------------------
     WIP Size Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   0.7168  0.9445     2.0000    0.0000    5.0000        1559
```

Figure 23. (Continued)

```
     Time In WIP Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       778     0.2486  0.3718     0.0664    0.0000    1.9580
                        <<< O >>>

Gravity Racks WIP  (a WIP Aggregator Object)
-------------------------------------------
     WIP Size Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   2.5440  2.3314     3.0000    0.0000    9.0000         894
     Time In WIP Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     1.5378  1.2918     1.0874    0.0000    4.7321
                        <<< O >>>

Sequenced Reels WIP  (a WIP Aggregator Object)
---------------------------------------------
     WIP Size Statistics
       Time of initialization = 90.00
       Current Time           = 360
Avg Value Std Dev Curr Value Min Value Max Value No. Changes
--------- ------- ---------- --------- --------- -----------
   9.6732  2.5441     9.0000    5.0000   16.0000         893
     Time In WIP Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     5.8480  1.7123     6.4547    2.1160   11.7644
                        <<< O >>>

WorkOrders Terminator  (a Terminator Object)
-------------------------------------------
     Time In System Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     8.9428  2.6758     7.6722    3.5861   16.1805
                        <<< O >>>
```

Figure 23. (Continued)

```
Selects Terminator  (a Terminator Object)
-----------------------------------------
     Time In System Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       448     5.1737  1.2679     6.2658    2.6510    9.1533
                          <<< O >>>

Bulk Parts Terminator  (a Terminator Object)
--------------------------------------------
     Time In System Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       448     5.1737  1.2679     6.2658    2.6510    9.1533
                          <<< O >>>

Reels Terminator  (a Terminator Object)
---------------------------------------
     Time In System Statistics
       Time of initialization = 90.00
       Current Time           = 360
Total Obs.  Avg Obs.  Std Dev  Last Obs.  Min Obs.  Max Obs.
---------- ---------- -------  ---------  --------  --------
       445     7.2739  1.8192     7.1673    3.5040   13.0759

                          <<< O >>>
```

Figure 23. (Concluded)

CHAPTER VII

SIMULATION ENVIRONMENT EVALUATION PROCEDURES

This chapter describes the simulation environment evaluation approach developed and utilized as part of the research. This includes sections on the criteria developed and/or chosen, discussion of these criteria in relation to the environments considered, and the design and execution of the Analytic Hierarchy Process [Saaty, 1988] decision model.

Introduction

As the simulation evaluation strategies discovered during the literature review were analyzed, the researcher was struck by an encompassing theme present in all of these resources. This theme can be basically summarized through the following two steps:

1) (a) Evaluate simulation systems through tangible considerations (measurable, quantitative characteristics) as a group of disjoint criteria,
   (b) Evaluate simulation systems through intangible considerations (non-measurable, qualitative) as a group of disjoint criteria, or
   (c) Evaluate simulation systems through both tangible

and intangible considerations as a group of disjoint criteria.

2) Upon completing one of choices 1a, 1b, or 1c, the result shall be a group of distinct, possibly conflicting, conclusions based on the different criteria considered. The final conclusion (choice of a simulation strategy to pursue) is then made through the analyst's intuitive, unstructured combination of the multiple conclusions.

This approach to simulation system evaluation is deficient. A significant problem with the approach is that the limited structure for the evaluation process dead ends with a group of conclusions (typically conflicting) rather than a single, final conclusion. An important segment of the comparison and decision making process, namely, the combination of the multiple conclusions into one final conclusion is left as an undefined and unstructured, and therefore, unrepeatable process. Through the use of the Analytic Hierarchy Process (AHP) as a method for structuring the multi-criteria decision problem, it is shown that this shortcoming can be rectified.

In addition to the decision structure, a set of alternatives providing feasible solutions to the problem must also be determined. In this case, due to the nature of the research, which is to develop a model or procedure for the comparison of simulation environments, a representative

set of environments shall be considered. This set shall be composed of 1) traditional, special purpose simulation systems and 2) OOP simulation systems. We may consider systems such as SLAM and SIMAN as examples of the first alternative. The second alternative shall be represented by the OOP simulation system developed as a portion of this research.

This portion of the research strives to evaluate simulation systems through the use of both tangible and intangible features within an organizing structure made up of an Analytic Hierarchy Process decision model.

Simulation System Evaluation Criteria

As with any situation in which a comparison between elements of a set of alternatives is required, a group of criteria and any necessary constraints must be determined. In the context considered here, in which a specific situation (a system to model, a set of hardware to utilize, etc.) is not part of the comparison process, it is inappropriate to factor specific constraints into the comparison. Therefore, the main focus of this section is to discuss the criteria which are significant in measuring the suitability of simulation systems.

The first step taken in developing a list of appropriate criteria was to address the various publications

in the area of simulation evaluation. For the most part, criteria from these sources consisted of intangible features but several publications addressed tangible measurements. In a preliminary, consolidated format, the low level criteria considered in one or more of these sources are:

General aspects

- Modeling flexibility
- Simulation language learning time
- Ease of model development
- Managed model complexity
- Easily understood simulation models
- Self documenting code
- Modeling in multiple levels of detail
- Reusable model code
- Simple model modification
- Incremental model implementation
- Similarity between models and systems of interest
- The availability of flexible, easy-to-use modules for modeling transporters, AGVS, conveyors, AS/RS, cranes, and robots
- Modeling approaches supported (event, process, continuous)
- Debugging aids (interactive debugger, on-line help)
- Standard output reports but allow for tailored reports
- Support for high quality graphical displays
- Model execution speed
- Size of simulation model allowed

Statistical aspects

- Ability to model probability distributions, large variety of standard distributions
- Allow distributions based on observed shop floor data
- Multiple stream random number generator
- Allow for multiple independent replications (different random numbers starting from the same state)
- Warm up period provisions

As one can observe, this list provides a rather complete collection of features which might be considered when comparing simulation systems. These criteria

form the nucleus around which the AHP model (discussed in the next section) has been developed.

After using these criteria as a starting point and remaining aware of the desire to structure an AHP model, it was determined that the appropriate overriding simulation evaluation aspects with which we are concerned are the following: 1) Simulation modeler effectiveness, 2) Usefulness and value of the simulation model, 3) Simulation environment performance considerations, and 4) Simulation language developer effectiveness. Thinking about each of these in turn starting with simulation modeler effectiveness, we see that this area is impacted by a significant number of the low level criteria from the previous list including:

- Modeling flexibility
- Simulation language learning time
- Ease of model development
- Managed model complexity
- Modeling in multiple levels of detail
- Reusable model code
- Simple model modification
- Incremental model implementation
- Similarity between models and systems of interest
- The availability of flexible, easy-to-use modules for modeling transporters, AGVS, conveyors, AS/RS, cranes, and robots
- Modeling approaches supported (event, process, continuous)
- Debugging aids (interactive debugger, on-line help)
- Standard reports but allow for tailored reports
- Ability to model probability distributions, large variety of standard distributions
- Allow distributions based on observed shop floor data
- Multiple stream random number generator
- Allow for multiple independent replications (different random numbers starting from the same state)
- Warm up period provisions

These characteristics have an impact on the amount of effort the modeler must expend, on the validity of the modeler's models, and on the ability to correctly ascertain system measures of performance.

The second top level aspect, the usefulness and value of the simulation model, is also impacted by many of the same low level criteria in the list:

- Modeling flexibility
- Managed model complexity
- Easily understood simulation models
- Self documenting code
- Modeling in multiple levels of detail
- Reusable model code
- Simple model modification
- Incremental model implementation
- Similarity between models and systems of interest
- Modeling approaches supported (event, process, continuous)
- Standard reports but allow for tailored reports
- Support for high quality graphical displays
- Ability to model probability distributions, large variety of standard distributions
- Allow distributions based on observed shop floor data

These criteria impact the usefulness of the simulation model by making model alteration and reuse simpler and less error prone, by allowing the models to be easier to maintain by successive analysts and to explain and sell to decision makers, and by improving the validity of completed models.

The third top level aspect, simulation environment performance, are covered by a much smaller set of listed criteria made up by:

- Model execution speed
- Size of simulation model allowed

We note that these criteria are concerned with the highly

tangible characteristics most commonly used in simulation environment comparisons.

The fourth and last top level aspect, the simulation language developer effectiveness, is one which is not commonly considered, and yet, is critical to the future success of a particular simulation system. Consider for a moment the dynamic character of any piece of software and it is apparent that the effectiveness of the software developer is of great importance. Both traditional and OOM simulation environments are changing software systems which receive new abilities through the efforts of the simulation language developers. Lower level criteria (not mentioned in the previous list) which impact this consideration include:

- Base language features
- Software modularity
- Software reusability

The four top level considerations just described, 1) Simulation modeler effectiveness, 2) Usefulness and value of the simulation model, 3) Simulation environment performance considerations, and 4) Simulation language developer effectiveness, form the basis for an AHP decision model developed and described in the next section of this chapter. By separating these characteristics into more manageable pieces and addressing the lower level characteristics (listed roughly before) directly to the simulation alternatives, a structured evaluation approach is achieved.

In addition to these criteria used in the AHP model, several novel approaches to the task of tangible model comparison were considered before being discarded as infeasible. One approach discussed was to create three separate systems of interest, small, medium, and large, and to record for comparison the discrete steps or amount of time taken while modeling each of the systems in the two simulation language alternatives. This evaluation approach was discarded as being invalid for several reasons. First, it would be necessary to have a fully fleshed out OOM system for the modeling exercise. Only in this manner could a fair comparison to an established language be made. Secondly, in order to carry out the experiment in a valid manner, a group of modelers equally familiar with both evaluation alternatives would be necessary. This was impossible to accomplish in any reasonable time limit (less than 3 years) due to the fact that an overwhelming majority of current simulation practitioners have a background in procedural languages and traditional simulation environments (SLAM, SIMAN, GPSS, etc.). Finally, a comparison between simulation approaches based on some small number of test cases (systems of modeling interest) would be weakly defensible at best, because it would be difficult to avoid choosing a test case which was not easier to model in one language than another. For these three reasons, this approach, although intuitively attractive, was discarded as being presently unmanageable and invalid.

Analytic Hierarchy Process (AHP)

Decision Model Development

The Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) is a methodology developed by Dr. T. L. Saaty in the mid-1970's as a "multi-objective, multi-criterion, decision making system employing a pairwise comparison procedure to arrive at a scale of preferences among sets of alternatives" [Saaty and Ramanujam (1983)]. The methodology deals with complex decision problems by providing a systematic approach to performing the required mental processes through the modeling of any problem as a hierarchy of interrelated elements. Applications of the methodology have appeared in several fields: economics and planning, energy policy making, health, conflict resolution, etc.

The AHP procedure is presented in detail through a general description and through its use in the simulation environment evaluation in a later section of this chapter.

The AHP is made up of four steps: [Zahedi (1986)]

1) The decision hierarchy must be set up by breaking the decision problem into a hierarchy of interrelated decision elements.

2) The input data which is made up of pairwise comparisons of the decision elements must be determined.

3) The eigenvalues and eigenvectors of the matrices are used to estimate the relative weights of the decision elements.

4) The relative weights of the decision elements are aggregated to result in a set of ratings for decision alternatives.

During step 1, the decision problem must be broken down into a hierarchy of interrelated decision elements.

> At the top of the hierarchy lies the most macro decision objective, such as the objective of making the best decision (or selecting the best alternative). The lower levels of the hierarchy contain attributes (objectives) which contribute to the quality of the decision. Details of these attributes increase at the lower levels of the hierarchy. The last levels of the hierarchy contain decision alternatives or selection choices. [Zahedi (1986)]

A generalized hierarchical structure as described is illustrated in Figure 24.

Once the complete hierarchy model is defined, the analyst may proceed with step 2 of the AHP which is the determination of the pairwise comparison matrices.

> The judgment phase of the AHP requires the following scale of absolute values (not ordinals) to express judgments in making paired comparisons: 1, equal [weight]; 3, moderate; 5, strong; 7, very strong; 9, extreme; 2, 4, 6, 8 for compromise; reciprocals for the inverse comparison; and decimal refinements between, if it is desirable to obtain a predetermined set of final priorities. [Saaty (1987)]

During the judgment phase, the analyst must carry out the comparisons.

> The elements in the second level are arranged in a matrix, and judgements are elicited as to the relative importance of each criterion when compared to every other criterion on that level. [Saaty (1987)]

The generalized question here is as follows: For the best problem solution, which second level criterion is considered more important and in how strong a manner? In particular,

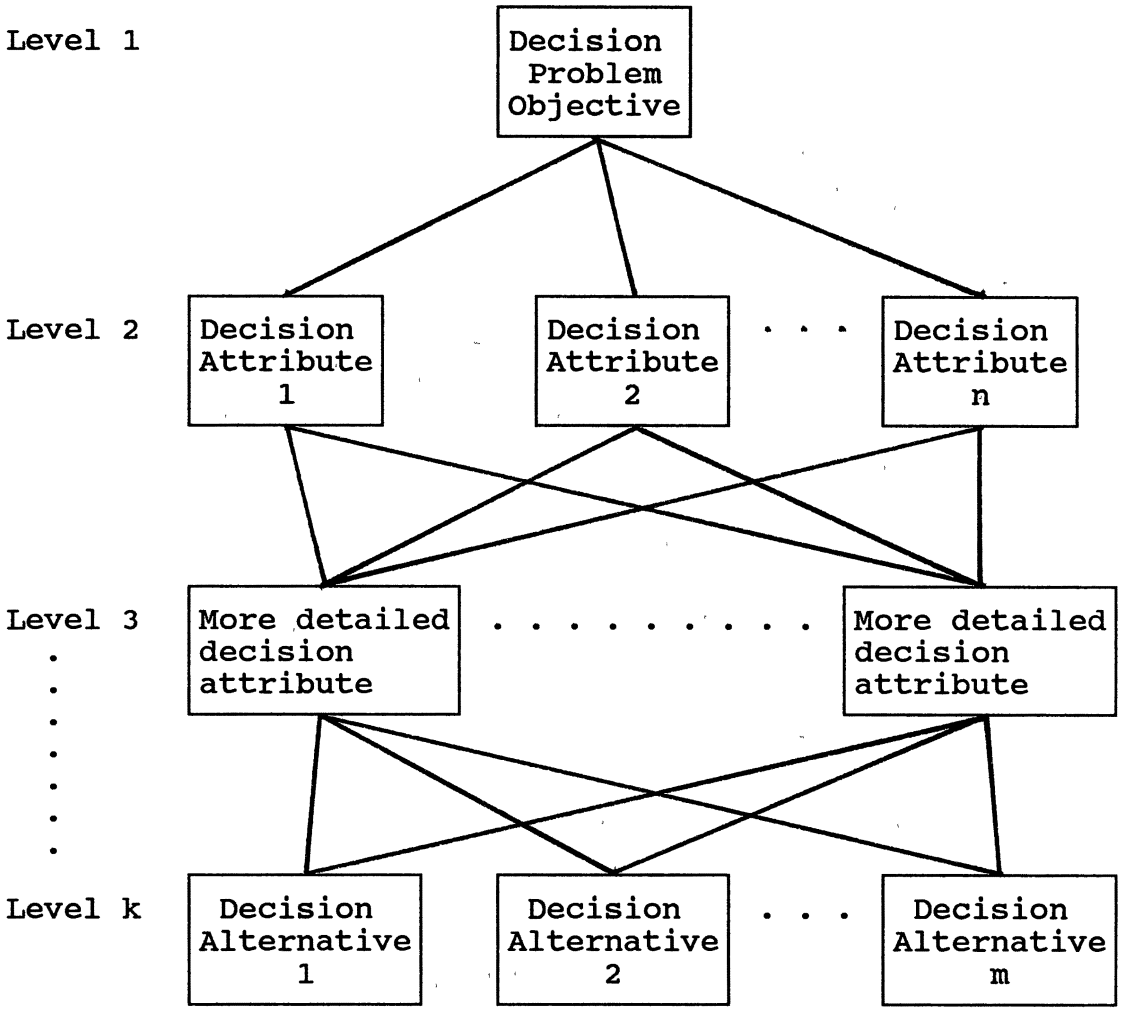


Figure 24. Standard Format of an AHP Decision Model [adapted from Zahedi (1986)]

for the evaluation of a simulation environment: For the best choice of a simulation system, which criterion is more important and how strongly do we favor it? In creating the pairwise comparison matrices, the following procedure is used:

> A criterion X represented on the left is compared with respect to a criterion Y represented on the top of the matrix. If X is more important than Y, then a

> numerical value greater that one is used in the (X,Y) position. If Y is more important than X, then the reciprocal of this value is used. The reciprocal of whatever value is entered in the (X,Y) position automatically is entered in the (Y,X) position. [Saaty (1987)].

In a like manner, the elements in the next level down in the hierarchy are subjected to pairwise comparisons. At each level, there will be n(i-1) (the number of elements at level i-1) pairwise comparison matrices with n(i) rows and columns each (fewer if an incomplete hierarchy is being evaluated). Each time the analyst is trying to answer (through the appropriate assignment of relative weights) the question "How important is this element at level i versus this other element at level i in satisfying or facilitating the element at level i-1 for which the comparison matrix is now being built?"

Step 3 of the AHP consists of solving the pairwise comparison matrices for the eigenvalues and eigenvectors in order to estimate the relative weights of the decision elements.

> The paired comparisons produce a ratio scale of weights of the relative importance or priorities of the criteria. Ratio scales are a strong class of numbers whose ratios remain the same when each of them is multiplied by a constant. ... Ratio scales make it possible not only to rank alternatives, but also to allocate resources in proportion to the values in an appropriate fashion. [Saaty (1987)]

The manner in which the ratio scales are derived from the pairwise comparison matrices is as follows:

> The argument for the solution methodology [Saaty] is as follows. If the evaluator could know the actual relative weights of n elements (at one level of the

hierarchy with respect to one level higher), the matrix of pairwise comparisons would be:

$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & \cdots & n \\ 1 & w(1)/w(1) & w(1)/w(2) & w(1)/w(3) & \cdots & w(1)/w(n) \\ 2 & w(2)/w(1) & w(2)/w(2) & w(2)/w(3) & \cdots & w(2)/w(n) \\ 3 & w(3)/w(1) & w(3)/w(2) & w(3)/w(3) & \cdots & w(3)/w(n) \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ n & w(n)/w(1) & w(n)/w(2) & w(n)/w(3) & \cdots & w(n)/w(n) \end{array}$$

In this case, the relative weights could be trivially obtained from each one of the n rows of matrix A. In other words, matrix A has rank 1; and the following holds:

$$A * W = n * W$$

where $W = (w(1), w(2), \ldots, w(n))T$ is the vector of actual relative weights, and n is the number of elements. In matrix algebra, n and W are called the eigenvalue and the right eigenvector of matrix A.

AHP posits that the evaluator does not know W and, therefore, is not able to produce the pairwise relative weights of matrix A accurately. Thus, the observed matrix A contains inconsistencies. The estimation of W (denoted as $\hat{W}$) could be obtained similarly to [the above equation] from:

$$\hat{A} * \hat{W} = \text{lambda(max)} * \hat{W},$$

where $\hat{A}$ is the observed matrix of pairwise comparisons, lambda(max) is the largest eigenvalue of $\hat{A}$, and $\hat{W}$ is its right eigenvector. $\hat{W}$ constitutes the estimation of W. [Zahedi (1986)]

Lambda(max) may be considered the estimation of n in [the above equation]. Saaty has shown that lambda(max) is always greater than or equal to n. The closer the value of computed lambda(max) is to n, the more consistent are the observed values of $\hat{A}$. This property has led to the construction of the consistency index (CI) as:

$$CI = (\text{lambda(max)} - n) / (n - 1)$$

and of the consistency ratio (CR) as:

$$CR = (CI / ACI) * 100,$$

where ACI is the average index of randomly generated weights [for a matrix of similar size]. As a rule of thumb, a CR value of 10 percent or less is considered acceptable. Otherwise, it is recommended that $\hat{A}$ be re-observed to resolve inconsistencies in pairwise comparisons. [Zahedi (1986)]

The estimation of W can be achieved through several different methods, of which one is:

> Divide the elements of each column by the sum of that column (i.e., normalize the column) and then add the elements in each resulting row and divide this sum by the number of elements on the row. This is a process of averaging over the normalized columns. [Saaty (1988)]

The final step of the AHP is the aggregation of the relative weights into measures of the solution alternatives. These final priorities are attained by weighting the relative values through the hierarchy and summing the totals for each decision alternative and normalizing the results (to sum to 1). More formally,

> the composite relative weight vector of elements at the kth level with respect to that of the first level may be computed from:

$$C[1,k] = \prod_{i=2}^{k} B(i),$$

> where C[1,k] is the vector of composite weights of elements at level k with respect to the element on level 1, and B(i) is the n(i-1) by n(1) matrix with rows consisting of estimated $\hat{W}$ vectors. n(i) represents the number of elements at level i and is the same as n in [previous equation] but is subscripted to show that it belongs to level i. [Zahedi (1986)]

The benefits of the AHP applied to decision problems in general, and simulation environment evaluation in

particular, are several. The hierarchical approach to modeling decision problems can be a beneficial procedure because it ensures that all decision elements are explicitly considered. The modeling process allows the analyst to fully refine the problem situation. In addition, the developed model allows the problem considerations to be effectively communicated to others affected by the analysis.

AHP Simulation Evaluation Decision Model

The development of the AHP model was an extremely time consuming and highly thought-intensive process. In general, the researcher came up with a preliminary structure and utilized the knowledge of the evaluation group to validate and adjust the model structure and linkages. The evaluation group consisted of four individuals, Dr. Joe H. Mize (Regents Professor in Industrial Engineering), Cem Karacal (Ph.D. candidate in Industrial Engineering), Chuda Basnet (Ph.D. candidate in Industrial Engineering), and the author. It should be pointed out that the decision hierarchy design was a thoroughly iterative process. In fact, the final decision model presented in this section represents a significant revision of an earlier hierarchy. This prototype hierarchy had 4 levels and was in the prioritization stage when the evaluation group determined that there was a need to reconsider the node definitions in the third level due to perceived lateral dependency among nodes. The reevaluation and alteration was performed using

this prototype hierarchy as the design foundation. Upon completion, the final hierarchy represented a significant change over the prototype hierarchy, including a reduction in the number of level 2 nodes and the addition of another intermediate attribute level. This adjustment was made with extreme care being taken to avoid any lateral dependency between nodes in the levels. Also, the redesign was effective because it took advantage of the partially completed first prioritization session, which indicated the location of problems in the prototype hierarchy when difficulties in setting pairwise evaluations were encountered.

The final AHP decision model developed for the comparison of simulation environments uses the four top level attributes listed in the second section of Chapter VII: 1) Simulation modeler effectiveness, 2) Usefulness and value of the simulation model, 3) Simulation environment performance considerations, and 4) Simulation language developer effectiveness, as the partition for level 2 in the hierarchy. Level 3 of the AHP model is composed of thirteen detailed attributes relating directly to level 2 above and to level 4, composed of 20 highly detailed attributes, below. The attributes listed in levels 3 and 4 were distilled and/or synthesized from those discussed in Section 2 of this chapter. The linkages between levels 2, 3, and 4 have been specified only when a lower level node has a possibility of affecting the achievement of an upper level

characteristic. Level 5 in the AHP model is composed of the two decision alternatives, traditional simulation systems and OOP simulation systems. Figure 25 graphically depicts the structure of the decision model.

The definitions of each of the nodes in the decision model and the discussion of the linkages between the nodes are as follows:

Level 1 - Problem Statement

1-1) Simulation Approach. The problem area that we are concerned with is the choice of the best simulation approach. The viewpoint from which this decision shall be made is that of a combined simulation system user and developer. In addition, the pairwise comparison weights for this decision problem are assigned from the viewpoint of a company which has a committed, long term effort to utilize simulation as a system planning tool (This coincides with the new orientation of simulation models as multiple use efforts).

Level 2 - Major Considerations

2-1) Simulation language developer effectiveness in simulation language extension. This important criterion in the decision problem addresses the ability of developers to extend the simulation language capabilities through the addition of significant new features. This person or task within the structure of the simulation system involves the

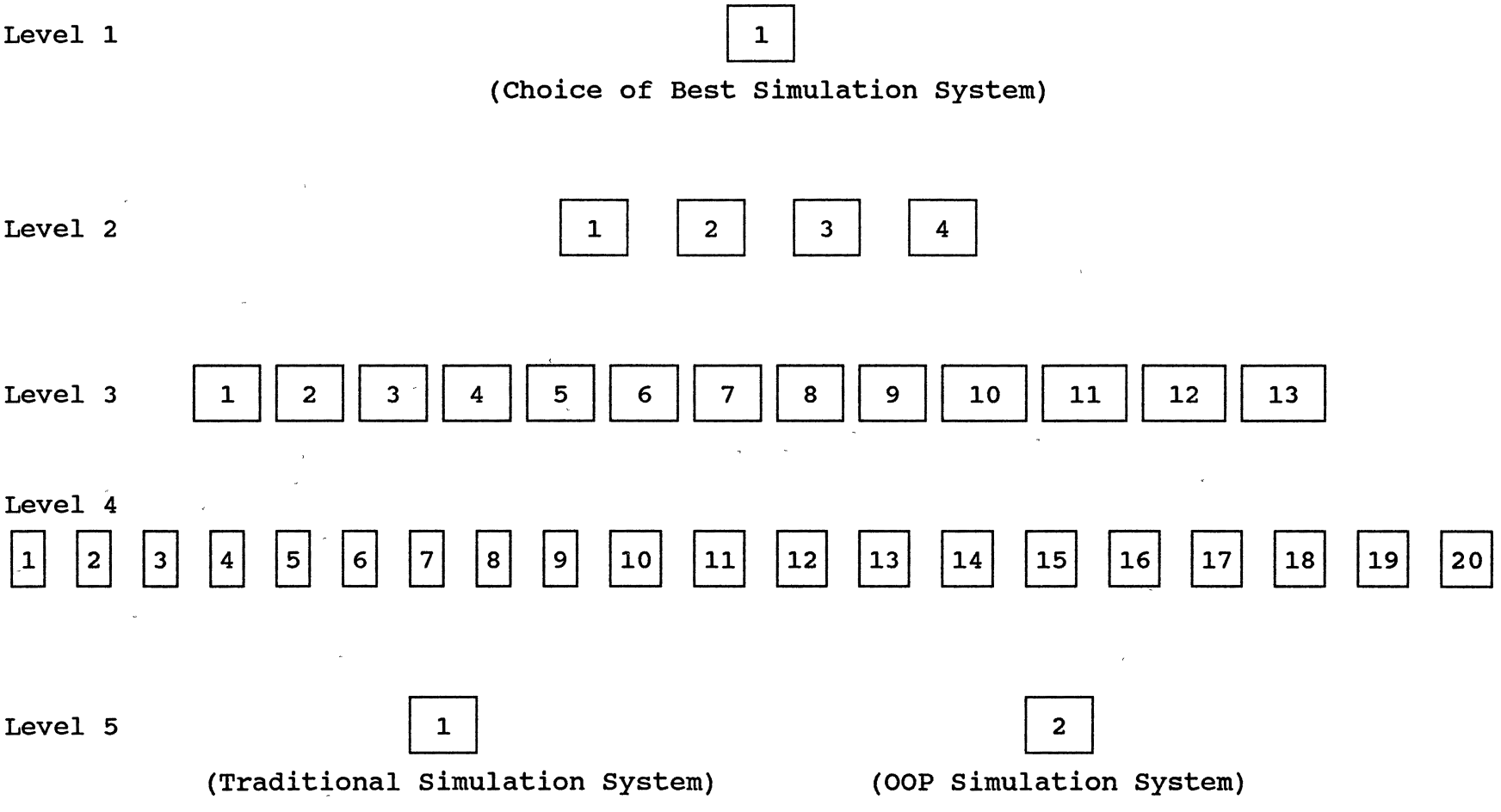


Figure 25. AHP Simulation Language Comparison Model

implementation of significant changes and additions to the entire software system, not merely software development for the creation of a specific new simulation model. Level 3 attributes are evaluated pairwise in their impact on increasing the effectiveness of the simulation language developer.

Upward links - 1-1

As could be anticipated, each of the nodes at level 2, 2-1 through 2-4, will be linked to the single top level node, 1-1, to allow the interrelationships accounted for in the lower linkages to factor into the final decision.

2-2) Simulation model developer effectiveness/Model development effort. This area in the decision problem is concerned with the effectiveness of the efforts of simulation model developers (how effective are their efforts). The person or task associated with simulation model development is involved in the use of currently available constructs and the implementation of reasonably simple new constructs (new base code) within the development of useful simulation models. Level 3 attributes are evaluated pairwise in their impact on increasing the model developer effectiveness and/or decreasing the model development effort required for a given system.

Upward links - 1-1

2-3) Model effectiveness. This area of the decision problem concerns the ability of a developed model to fulfill

its various needs (provide measures of system performance, facilitate presentation of results, meet future modeling extensions, alteration, and reuse needs, etc.) within the system planning, implementation, and operation task. Level 3 attributes are evaluated pairwise on their importance toward increasing model effectiveness.

Upward links - 1-1

2-4) Performance considerations. This system aspect addresses basic hardware related performance measures (memory size, execution speed, etc.). Level 3 attributes impacting this criterion are compared pairwise on how important they are in system performance as a whole.

Upward links - 1-1

Level 3 - Detailed Criteria

3-1) Full featured base language. This criterion considers the features of the base language in providing a foundation from which to build software. The pairwise comparison of features is completed based on their relative importance for inclusion in a base language.

Upward links - 2-1,2-2

This node has upward links to two of the nodes in level 2, namely simulation language developer effectiveness and simulation model developer effectiveness, because both of these tasks involve some effort in base language coding.

3-2) Software life cycle management and change control features. This aspect considers the impact of specific features in level 4 on the ability to control software changes throughout the growth of a software system. Specifically, 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. Attributes in level 4 are evaluated against one another in their ability to increase the manageability of software changes.

Upward links - 2-1

Primary, long-term changes to the simulation system are the purview of the simulation language developer. Only through this task do environment enhancements become formalized and widely available. Abilities in software change management, therefore, have an effect on language development and negligible effect on the other nodes in level 2.

3-3) Development support environment. This criterion refers to the type of environment in which simulation language and model development is performed. Items which are considered to have a positive impact include debugging abilities, code libraries, data structure support, etc. The items in level 4 which are linked to this node will be evaluated pairwise on their ability to support base language software implementation.

Upward links - 2-1,2-2

Support of base language software development affects the ability of developers and modelers in accomplishing their tasks, which to a greater and lesser degree, respectively, involve the implementation of new code in the base language.

3-4) Extension and reuse of software development efforts. This criterion gathers together level 4 features which affect the ability to extend and re-apply previously developed base language software. This attribute greatly increases the value of developed software.

Upward links - 2-1,2-2

Again, the level 2 nodes which include base language software development aspects are the language and model developer effectiveness. This ability has no addressable effect on either model effectiveness or performance considerations.

3-5) Simulation language knowledge/learning effort required. Attributes present in level 4 which impact the amount of knowledge needed to use a simulation system (and the effort required to learn the system) are addressed within this characteristic.

Upward links - 2-2

This consideration has appreciable impact on the effectiveness of the simulation modeler and no impact on the other level 2 criteria.

3-6) Simulation language features. Different simulation language abilities (i.e., debugging support, statistical support, etc.) possess different levels of importance in supporting simulation modeling. This category allows the importance of various available features to be interrelated to one another and related to higher level effectiveness. The features to which this node is connected in level 4 are compared pairwise on their importance in providing a complete simulation environment.

Upward links - 2-2,2-3

Improved simulation language features, represented by this node in the decision model, can influence both the modeler and model effectiveness.

3-7) Ability to communicate model structure and features. Of importance in simulation modeling is the ability for humans to exchange thoughts on the structure and features of a model. Level 4 attributes are compared pairwise on their ability to improve the communicability of simulation models.

Upward links - 2-2,2-3

Simulation modeler effectiveness and model effectiveness are both improved by an increased ability to communicate the structure of a simulation model.

3-8) Amount of modeling abstraction required/Degree of correspondence to the real system. Model abstraction refers to the degree to which the representation of the system (the

simulation model) is conceptually removed from the actual system. As model abstraction increases, the degree of model correspondence to the real system decreases. This node relates specific features (at lower levels) having an effect on the abstraction required in the modeling process.

Upward links - 2-2,2-3

Because the impact of reductions in modeling abstraction is to improve the effectiveness of modelers and models, this criterion is linked to nodes 2 and 3 in the second level.

3-9) Model extension, alteration, and reuse. This criterion provides for the comparison of items which impact the ability of specific models or portions of models to be used through a change process (extension = minor change, alteration = moderate change, reuse = significant change).

Upward links - 2-2,2-3

This important characteristic obviously has considerable impact on the effectiveness of both the modeler and model effectiveness.

3-10) Provision for high level combination/Model complexity management. 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.

Upward links - 2-2,2-3

As provisions for system groupings and the conceptual ease with which they may be used increases, both modeler and model effectiveness improve.

3-11) Size of model supported. This memory characteristic considers how efficiently computer memory is used in simulation model representation and execution. The viewpoint addresses the relative size of models which may exist within a basic PC.

Upward links - 2-4

This attribute links to the performance considerations criterion in level 2 and directly to the two simulation alternatives in level 5.

3-12) Basic memory requirements. Another computer memory consideration, this characteristic addresses the amount of memory needed to run the simulation environment for the smallest of models.

Upward links - 2-4

Again, this attribute links to the performance considerations criterion in level 2 and directly to the two simulation alternatives in level 5.

3-13) Execution speed. Another performance consideration is the execution time required for simulation models of a particular system.

Upward links - 2-4

Again, this attribute links to the performance

considerations criterion in level 2 and directly to the two simulation alternatives in level 5.

Level 4 - Simulation System Attributes

4-1) Graphics/User interface capability. For the extension of capabilities into new features, particularly for simulation environment enhancements, support for graphics or enhanced interfaces within the base language is important.

Upward links - 3-1,3-3

In addition to being a base language feature (and linking to node 3-1), this capability also has an effect on the development support environment (therefore, the link to node 3-3).

4-2) Ease of learning the base language. This attribute is important to consider because both developers and modelers shall be working to some degree in the base language.

Upward links - 3-1,3-2,3-3,3-4

Linkages to level 3 are: base language features (3-1), software change management (3-2), development support environment (3-3), and software extension and reuse (3-4).

4-3) Integrated software toolset (prototyping, language debugging, etc.). This attribute addresses the type of environment provided by the base language for software development.

Upward links - 3-1,3-2,3-3,3-4,3-9

Level 3 linkages are: base language features (3-1), software change management (3-2), development support environment (3-3), software extension and reuse (3-4), and model extension, alteration, and reuse (3-9).

4-4) Access to stand alone code libraries. The ability to develop and use complex data types and related software in the base language as stand alone units is an important feature. This feature is comparable to software primitive libraries (in procedural languages) and object oriented classes (in OOP languages).

Upward links - 3-1,3-2,3-3,3-4,3-5,3-9,3-10

Level 3 linkages are: base language features (3-1), software change management (3-2), development support environment (3-3), software extension and reuse (3-4), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-5) Code reusability. This attribute refers to the ability to incorporate portions of already developed base language software within a stand alone unit. The attribute is comparable to the inheritance feature in OOP systems.

Upward links - 3-2,3-3,3-4,3-5,3-9

The ability to reuse code in this manner influences software change management (3-2), development support environment (3-3), software extension and reuse (3-4),

simulation language knowledge/learning effort required (3-5), and model extension, alteration, and reuse (3-9).

4-6) Software modularity. An important attribute for both base language and model software is addressed by this node, software modularity. As the modularity of developed software in the base language and modeling language changes, many characteristics in level 3 are impacted (defined by the links).

Upward links - 3-2,3-3,3-4,3-5,3-9,3-10

The impact of software modularity is seen in the level 3 nodes: software change management (3-2), development support environment (3-3), software extension and reuse (3-4), simulation language knowledge/learning effort required (3-5), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-7) High level model language. The presence of a high level model specification method (either iconic or brief textual) has become the standard for model implementation. Therefore, this attribute, the conceptual level of typical simulation model specification, must be accounted for in any evaluation.

Upward links - 3-5,3-6,3-7,3-8,3-9,3-10

Characteristics affected by the presence of a high level specification language are: simulation language knowledge/learning effort required (3-5), basic simulation

language abilities (3-6), ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-8) Structured model development approach. The ability to implement simulation models in a structured manner is a positive feature. By increasing the structure of the model development process, the consistency of resulting simulation models is increased.

Upward links - 3-5,3-6,3-7,3-8,3-9

Influences of structured model specification include decreased simulation language knowledge/learning effort required (3-5), improved basic simulation language abilities (3-6), an improved ability to communicate model structure (3-7), reduced modeling abstraction (3-8), and increased model extension, alteration, and reuse capacity (3-9).

4-9) Output provisions. This simulation system attribute refers to the level of simulation environment support for both standard and special results output.

Upward links - 3-5,3-6,3-7

This attribute influences the amount of simulation language knowledge/learning effort required (3-5) and the ability to communicate model structure and features (3-7), and is one basic simulation language ability (3-6).

4-10) Model debugging support/verification. This attribute addresses the features provided for model debugging and verification and the degree of effectiveness achieved by these features.

Upward links - 3-5,3-6,3-9,3-10

Simulation environment characteristics which this attribute affects include: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-11) Statistical support. Obviously, provisions within a simulation environment for the generation and use of random numbers are necessary and important. This attribute considers the level of random number (distributions, separate streams, clearing, etc.) support provided by the environment.

Upward links - 3-5,3-6

The impact of this simulation attribute is seen at level 3 in the decision hierarchy in the two characteristics: simulation language knowledge/learning effort required (3-5) and basic simulation language abilities (3-6).

4-12) Incorporation of special code implementation and "packaging" within models/Extension of high level constructs. This environment feature refers to the ease

with which new base language coding can be included in a simulation model and how well (conceptually) the new base code links to the rest of the model.

Upward links - 3-4,3-5,3-6,3-7,3-8,3-9,3-10

The ability to package new base language code seamlessly into models and extend already present high level constructs is important for the achievement of a number of characteristics at level 3 of the decision model. These are: software extension and reuse (3-4), simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-13) Specialized component support at a high level. In addition to supporting model development through high level constructs, an important consideration is the presence of a full complement of high level language features. This attribute specifically considers the simulation of typically "difficult" equipment (material handling, AGVS, conveyors, etc.)

Upward links - 3-5,3-6,3-7,3-8,3-9

This attribute has impact in the level 3 characteristics: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), ability to communicate model structure (3-7),

modeling abstraction required/degree of model correspondence to the real system (3-8), and model extension, alteration, and reuse (3-9).

4-14) Provisions for different levels of modeling detail. In certain modeling situations it may be appropriate to model portions of the system of interest with a high level of detail and other portions in an aggregate manner. This attribute refers to the conceptual and actual ability to achieve this goal within the simulation system.

Upward links - 3-5,3-6,3-7,3-8,3-9,3-10

Level 3 characteristics where this attribute has an impact include: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-15) Access to model code/On-line documentation. Another environment attribute which is considered in this evaluation is the ability to access the source code (or some type of highly detailed documentation) for the simulation environment.

Upward links - 3-5,3-6

Affected level 3 criteria include the amount of simulation language knowledge/learning effort required (3-5)

and basic simulation language abilities (3-6).

4-16) Modeling approaches supported. Most important in a simulation system is the presence of a process oriented specification mode (high level representation). Of additional importance is the ability to perform modeling in terms of the other two world views, continuous and discrete event. This attribute in the decision model accounts for this ability.

Upward links - 3-5,3-6,3-8

The ability to model systems using multiple orientations affects level 3 characteristics: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), and modeling abstraction required/degree of model correspondence to the real system (3-8).

4-17) Model code readability. Although not a primary concern itself, the readability or understandability of a simulation model representation scheme influences a number of aspects in simulation environment effectiveness.

Upward links - 3-5,3-6,3-7

Model language readability has influence in these level 3 criteria: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), and ability to communicate model structure (3-7).

4-18) Information and decision processes modules. Of recent interest in simulation modeling is support for

structured and non-structured decision support model components and centralized model database features. This attribute considers these types of features which may be supported by a simulation system.

Upward links - 3-5,3-6,3-7,3-8,3-9,3-10

The availability of these types of features impacts the following level 3 nodes: simulation language knowledge/learning effort required (3-5), basic simulation language abilities (3-6), ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), model extension, alteration, and reuse (3-9), and provision for high level combination/model complexity management (3-10).

4-19) Validation: Model operation correspondence to the real system. Another consideration in the evaluation of modeling systems addresses the enhancement of the model validation process through the degree of model operation correspondence to the real system. As model operation becomes conceptually closer to that of the real system, a number of criteria in level 3 are positively affected.

Upward links - 3-7,3-8,3-10

The level 3 nodes affected are: ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), and provision for high level combination/model complexity management (3-10).

4-20) Physical component representation correspondence. In the same manner as model operation correspondence, model representation correspondence to the real system can improve model validation and understanding. This attribute refers to the degree of correspondence between the real system and model representation (i.e., one-to-one relationship between modeling elements and real system elements).

Upward links - 3-7,3-8,3-10

The level 3 nodes affected are: ability to communicate model structure (3-7), modeling abstraction required/degree of model correspondence to the real system (3-8), and provision for high level combination/model complexity management (3-10).

Level 5 - Solution Alternatives

5-1) Traditional, special purpose simulation systems. This solution alternative represents the standard simulation system typically used in discrete event modeling, of which, a number of commercial systems are available.

Upward links - all at level 4 plus 3-11, 3-12, and 3-13

5-2) OOP simulation system. This solution alternative represents the new OOP simulation system, for which the prototype system was developed.

Upward links - all at level 4 plus 3-11, 3-12, and 3-13

Summary

This section has presented a brief introduction to the Analytic Hierarchy Process and fully described the structure of the AHP simulation environment comparison model. In the next section, the pairwise comparison matrices determined during the AHP model evaluation phase are shown along with the results of the weight composition process. In addition to this AHP evaluation, an evaluation discussion of the two simulation alternatives is also presented.

Evaluation of Modeling Environments

AHP Decision Model Analysis

In order to complete the analysis of the AHP simulation environment comparison model, a group of four individuals (the author, his major advisor, and two other doctoral students) experienced in simulation, worked through the prioritization process for the entire decision model. During this weighting process, the participants were careful to thoroughly discuss the criteria or attributes being considered and to agree on the assigned weights. In addition, upon completion and entry into a previously prepared spreadsheet, each matrix was addressed to ensure that consistent weights had been assigned. Two of the weighting matrices were reevaluated due to an excessive level of inconsistency. The pairwise priority matrices determined in this manner are presented in Tables 2 through 39.

The prepared AHP calculation spreadsheets calculate the priorities from each of the completed matrices (in addition to checking matrix consistency). These priority vectors were then combined into the appropriate matrices which were multiplied together to yield the solution alternatives priority vector which is listed in Table 40. The AHP calculation spreadsheets and the weight composition spreadsheet are contained in Appendix C.

TABLE 2

NODE 1-1 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

1-1 Simulation Approach
Lower level connections:
1) Simulation language developer effectiveness in simulation language extension, 2-1
2) Simulation model developer effectiveness/Model development effort, 2-2
3) Model effectiveness, 2-3
4) Performance considerations, 2-4

Pairwise weights

Row \ Col	1	2	3	4
1	1.000	0.250	0.200	5.000
2	4.000	1.000	0.500	7.000
3	5.000	2.000	1.000	9.000
4	0.200	0.143	0.111	1.000

TABLE 3

NODE 2-1 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

2-1 Simulation language developer effectiveness in simulation language extension
Lower level connections:
1) Full featured base language, 3-1
2) Software life cycle management and change control features, 3-2
3) Development support environment, 3-3
4) Extension and reuse of software development efforts, 3-4

Pairwise weights

Row \ Col	1	2	3	4
1	1.000	0.333	0.200	0.333
2	3.000	1.000	0.333	0.500
3	5.000	3.000	1.000	1.000
4	3.000	2.000	1.000	1.000

TABLE 4

NODE 2-2 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

2-2 Simulation model developer effectiveness/Model development effort
Lower connections:
1) Full featured base language, 3-1
2) Development support environment, 3-3
3) Extention and reuse of software development efforts, 3-4
4) Simulation language knowledge/learning effort required, 3-5
5) Simulation language features, 3-6
6) Ability to communicate model structure and features, 3-7
7) Amount of modeling abstraction required/Degree of correspondence to the real system, 3-8
8) Model extension, alteration, and reuse, 3-9
9) Provision for high level combination/Model complexity management, 3-10

Pairwise weights

Row \ Col	1	2	3	4	5	6	7	8	9
1	1.000	0.200	0.333	0.167	0.143	0.143	0.111	0.200	0.250
2	5.000	1.000	2.000	0.333	0.200	0.200	0.143	0.500	0.500
3	3.000	0.500	1.000	0.200	0.200	0.200	0.143	0.250	0.333
4	6.000	3.000	5.000	1.000	1.000	0.500	0.333	2.000	3.000
5	7.000	5.000	5.000	1.000	1.000	0.333	0.200	2.000	4.000
6	7.000	5.000	5.000	2.000	3.000	1.000	0.333	2.000	4.000
7	9.000	7.000	7.000	3.000	5.000	3.000	1.000	5.000	7.000
8	5.000	2.000	4.000	0.500	0.500	0.500	0.200	1.000	3.000
9	4.000	2.000	3.000	0.333	0.250	0.250	0.143	0.333	1.000

TABLE 5

NODE 2-3 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

2-3 Model effectiveness
Lower level connections:
1) Simulation language features, 3-6
2) Ability to communicate model structure and features, 3-7
3) Amount of modeling abstraction required/Degree of correspondence to the real system, 3-8
4) Model extension, alteration, and reuse, 3-9
5) Provision for high level combination/Model complexity management, 3-10

Pairwise weights

Row \ Col	1	2	3	4	5
1	1.000	0.143	0.143	0.143	0.333
2	7.000	1.000	0.500	1.000	5.000
3	7.000	2.000	1.000	2.000	5.000
4	7.000	1.000	0.500	1.000	4.000
5	3.000	0.200	0.200	0.250	1.000

TABLE 6

NODE 2-4 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

2-4 Performance considerations
Lower level connections:
1) Size of model supported, 3-11
2) Basic memory requirements, 3-12
3) Execution speed, 3-13

Pairwise weights

Row \ Col	1	2	3
1	1.000	5.000	3.000
2	0.200	1.000	0.333
3	0.333	3.000	1.000

TABLE 7

NODE 3-1 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-1 Full featured base language
Lower level connections:
1) Graphics/User interface capability, 4-1
2) Ease of learning the base language, 4-2
3) Integrated software toolset, 4-3
4) Access to stand alone code libraries, 4-4

Pairwise weights

Row \ Col	1	2	3	4
1	1.000	3.000	0.250	0.200
2	0.333	1.000	0.125	0.143
3	4.000	8.000	1.000	0.333
4	5.000	7.000	3.000	1.000

TABLE 8

NODE 3-2 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-2 Software life cycle management and change control features
Lower level connections:
1) Ease of learning the base language, 4-2
2) Integrated software toolset, 4-3
3) Access to stand alone code libraries, 4-4
4) Code reusability, 4-5
5) Software modularity, 4-6

Pairwise weights

Row \ Col	1	2	3	4	5
1	1.000	0.111	0.200	0.333	0.143
2	9.000	1.000	5.000	7.000	4.000
3	5.000	0.200	1.000	3.000	0.250
4	3.000	0.143	0.333	1.000	0.167
5	7.000	0.250	4.000	6.000	1.000

TABLE 9

NODE 3-3 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-3 Development support environment
Lower level connections:
1) Graphics/User interface capabilities, 4-1
2) Ease of learning the base language, 4-2
3) Integrated software toolset, 4-3
4) Access to stand alone code libraries, 4-4
5) Code reusability, 4-5
6) Software modularity, 4-6

Pairwise weights

Row \ Col	1	2	3	4	5	6
1	1.000	2.000	0.111	0.143	0.250	0.200
2	0.500	1.000	0.111	0.167	0.250	0.200
3	9.000	9.000	1.000	5.000	7.000	5.000
4	7.000	6.000	0.200	1.000	3.000	1.000
5	4.000	4.000	0.143	0.333	1.000	0.333
6	5.000	5.000	0.200	1.000	3.000	1.000

TABLE 10

NODE 3-4 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-4 Extension and reuse of software development efforts
Lower level connections:
1) Ease of learning the base language, 4-2
2) Integrated software toolset, 4-3
3) Access to stand alone code libraries, 4-4
4) Code reusability, 4-5
5) Software modularity, 4-6
6) Incorporation of special code implementation and "packaging" within models/Extension of high level constructs, 4-12

Pairwise weights

Row \ Col	1	2	3	4	5	6
1	1.000	0.143	0.143	0.167	0.200	0.167
2	7.000	1.000	0.200	0.333	0.333	0.200
3	7.000	5.000	1.000	5.000	3.000	2.000
4	6.000	3.000	0.200	1.000	1.000	0.500
5	5.000	3.000	0.333	1.000	1.000	1.000
6	6.000	5.000	0.500	2.000	1.000	1.000

TABLE 11

NODE 3-5 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-5 Simulation language knowledge/learning effort required
Lower level connections:
1) Access to stand alone code libraries, 4-4
2) Code reusability, 4-5
3) Software modularity, 4-6
4) High level model language, 4-7
5) Structured model development approach, 4-8
6) Output provisions, 4-9
7) Model debugging support/verification, 4-10
8) Statistical support, 4-11
9) Incorporation of special code implementation and "packaging" within models/Extension of high level constructs, 4-12
10) Specialized component support at a high level, 4-13
11) Provision for different levels of modeling detail, 4-14
12) Access to model code/On-line documentation, 4-15
13) Modeling approaches supported, 4-16
14) Model code readability, 4-17
15) Information and decision processes modules, 4-18

Pairwise weights

Row \ Col	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1.00	3.00	1.00	0.11	0.14	0.33	0.20	0.33	1.00	0.14	0.33	0.33	0.50	0.25	0.50
2	0.33	1.00	1.00	0.11	0.14	0.33	0.20	0.20	3.00	0.14	0.50	0.25	0.33	0.20	0.25
3	1.00	1.00	1.00	0.11	0.14	0.33	0.20	0.20	3.00	0.14	0.50	0.25	0.33	0.25	0.20
4	9.00	9.00	9.00	1.00	3.00	5.00	3.00	5.00	9.00	4.00	7.00	7.00	5.00	5.00	3.00
5	7.00	7.00	7.00	0.33	1.00	3.00	1.00	3.00	5.00	0.33	3.00	3.00	3.00	3.00	0.25
6	3.00	3.00	3.00	0.20	0.33	1.00	0.50	1.00	3.00	0.20	0.33	0.33	0.33	0.33	0.20
7	5.00	5.00	5.00	0.33	1.00	2.00	1.00	3.00	5.00	0.20	1.00	0.50	0.33	0.33	0.20
8	3.00	5.00	5.00	0.20	0.33	1.00	0.33	1.00	3.00	0.20	0.33	0.33	0.33	0.33	0.20
9	1.00	0.33	0.33	0.11	0.20	0.33	0.20	0.33	1.00	0.14	0.25	0.20	0.20	0.20	0.14
10	7.00	7.00	7.00	0.25	3.00	5.00	5.00	5.00	7.00	1.00	5.00	4.00	5.00	5.00	2.00
11	3.00	2.00	2.00	0.14	0.33	3.00	1.00	3.00	4.00	0.20	1.00	0.33	0.33	0.33	0.20
12	3.00	4.00	4.00	0.14	0.33	3.00	2.00	3.00	5.00	0.25	3.00	1.00	1.00	0.50	0.33
13	2.00	3.00	3.00	0.20	0.33	3.00	3.00	3.00	5.00	0.20	3.00	1.00	1.00	0.50	0.20
14	4.00	5.00	4.00	0.20	0.33	3.00	3.00	3.00	5.00	0.20	3.00	2.00	2.00	1.00	0.20
15	2.00	4.00	5.00	0.33	4.00	5.00	5.00	5.00	7.00	0.50	5.00	3.00	5.00	5.00	1.00

TABLE 12

NODE 3-6 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-6 Simulation language features

Lower level connections:

1) High level model language, 4-7
2) Structured model development approach, 4-8
3) Output provisions, 4-9
4) Model debugging support/verification, 4-10
5) Statistical support, 4-11
6) Incorporation of special code implementation and "packaging" within models/Extension of high level constructs, 4-12
7) Specialized component support at a high level, 4-13
8) Provisions for different levels of modeling detail, 4-14
9) Access to model code/On-line documentation, 4-15
10) Modeling approaches supported, 4-16
11) Model code readability, 4-17
12) Information and decision processes modules, 4-18

Pairwise weights

Row \ Col	1	2	3	4	5	6	7	8	9	10	11	12
1	1.00	7.00	3.00	1.00	3.00	7.00	3.00	5.00	4.00	4.00	7.00	5.00
2	0.14	1.00	0.33	0.20	0.20	3.00	0.25	1.00	0.50	0.20	1.00	0.33
3	0.33	3.00	1.00	0.33	1.00	5.00	0.50	5.00	3.00	0.50	2.00	2.00
4	1.00	5.00	3.00	1.00	3.00	5.00	3.00	5.00	5.00	2.00	4.00	3.00
5	0.33	5.00	1.00	0.33	1.00	4.00	3.00	5.00	5.00	2.00	5.00	3.00
6	0.14	0.33	0.20	0.20	0.25	1.00	0.20	0.25	0.20	0.14	0.20	0.25
7	0.33	4.00	2.00	0.33	0.33	5.00	1.00	3.00	2.00	0.33	5.00	0.50
8	0.20	1.00	0.20	0.20	0.20	4.00	0.33	1.00	0.33	0.20	0.33	0.20
9	0.25	2.00	0.33	0.20	0.20	5.00	0.50	3.00	1.00	0.20	0.50	0.20
10	0.25	5.00	2.00	0.50	0.50	7.00	3.00	5.00	5.00	1.00	5.00	3.00
11	0.14	1.00	0.50	0.25	0.20	5.00	0.20	3.00	2.00	0.20	1.00	0.33
12	0.20	3.00	0.50	0.33	0.33	4.00	2.00	5.00	5.00	0.33	3.00	1.00

TABLE 13

NODE 3-7 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-7 Ability to communicate model structure and features
Lower level connections:
1) High level model language, 4-7
2) Structured model development approach, 4-8
3) Output provisions, 4-9
4) Incorporation of special code implementation and "packaging" within models/Extension of high level constructs, 4-12
5) Specialized component support at a high level, 4-13
6) Provisions for different levels of modeling detail, 4-14
7) Model code readability, 4-17
8) Information and decision processes modules, 4-18
9) Validation: Model operation correspondence to the real system, 4-19
10) Physical component representation correspondence, 4-20

Pairwise weights

Row \ Col	1	2	3	4	5	6	7	8	9	10
1	1.00	6.00	7.00	5.00	3.00	7.00	3.00	2.00	1.00	2.00
2	0.17	1.00	2.00	0.33	0.20	2.00	0.33	0.20	0.14	0.17
3	0.14	0.50	1.00	0.20	0.20	0.50	0.33	0.25	0.17	0.20
4	0.20	3.00	5.00	1.00	0.33	3.00	0.50	0.33	0.14	0.17
5	0.33	5.00	5.00	3.00	1.00	5.00	2.00	1.00	0.33	0.50
6	0.14	0.50	2.00	0.33	0.20	1.00	0.33	0.20	0.14	0.17
7	0.33	3.00	3.00	2.00	0.50	3.00	1.00	0.25	0.14	0.20
8	0.50	5.00	4.00	3.00	1.00	5.00	4.00	1.00	0.33	0.50
9	1.00	7.00	6.00	7.00	3.00	7.00	7.00	3.00	1.00	2.00
10	0.50	6.00	5.00	6.00	2.00	6.00	5.00	2.00	0.50	1.00

TABLE 14

NODE 3-8 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-8 Amount of modeling abstraction/Degree of correspondence to the real system
Lower level connections:
1) High level model language, 4-7
2) Structured model development approach, 4-8
3) Incorporation of special code implementation and "packaging" within models /Extension of high level constructs, 4-12
4) Specialized component support at high level, 4-13
5) Provisions for different levels of modeling detail, 4-14
6) Modeling approaches supported, 4-16
7) Information and decision processes modules, 4-18
8) Validation: Model operation correspondence to the real system, 4-19
9) Physical component representation correspondence, 4-20

Pairwise weights

Row \ Col	1	2	3	4	5	6	7	8	9
1	1.000	7.000	6.000	2.000	5.000	3.000	1.000	0.333	0.333
2	0.143	1.000	0.500	0.167	0.333	0.200	0.143	0.125	0.143
3	0.167	2.000	1.000	0.200	0.500	3.000	0.200	0.143	0.200
4	0.500	6.000	5.000	1.000	5.000	1.000	0.250	0.167	0.200
5	0.200	3.000	2.000	0.200	1.000	0.333	0.200	0.143	0.200
6	0.333	5.000	0.333	1.000	3.000	1.000	0.333	0.200	0.200
7	1.000	7.000	5.000	4.000	5.000	3.000	1.000	0.333	1.000
8	3.000	8.000	7.000	6.000	7.000	5.000	3.000	1.000	2.000
9	3.000	7.000	5.000	5.000	5.000	5.000	1.000	0.500	1.000

TABLE 15

NODE 3-9 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-9 Model extension, alteration, and reuse
Lower level connections:
1) Integrated software toolset, 4-3
2) Access to stand alone code libraries, 4-4
3) Code reusability, 4-5
4) Software modularity, 4-6
5) High level model language, 4-7
6) Structured model development approach, 4-8
7) Model debugging support/verification, 4-10
8) Incorporation of special code implementation and "packaging" within models/ Extension of high level constructs, 4-12
9) Special component support at a high level, 4-13
10) Provisions for different levels of modeling detail, 4-14
11) Information and decision processes modules, 4-18

Pairwise weights

Col / Row	1	2	3	4	5	6	7	8	9	10	11
1	1.00	0.33	0.33	0.20	0.14	0.13	0.20	0.20	0.20	0.20	0.14
2	3.00	1.00	0.33	0.20	0.14	0.14	0.33	0.33	0.25	0.20	0.14
3	3.00	3.00	1.00	0.20	0.14	0.14	0.33	0.20	0.20	0.33	0.20
4	5.00	5.00	5.00	1.00	0.33	0.50	2.00	3.00	0.50	3.00	0.33
5	7.00	7.00	7.00	3.00	1.00	2.00	4.00	3.00	3.00	5.00	2.00
6	8.00	7.00	7.00	2.00	0.50	1.00	3.00	2.00	0.33	0.33	0.20
7	5.00	3.00	3.00	0.50	0.25	0.33	1.00	0.33	0.20	0.33	0.20
8	5.00	3.00	5.00	0.33	0.33	0.50	3.00	1.00	0.33	1.00	0.20
9	5.00	4.00	5.00	2.00	0.33	3.00	5.00	3.00	1.00	3.00	0.50
10	5.00	5.00	3.00	0.33	0.20	3.00	3.00	1.00	0.33	1.00	0.25
11	7.00	7.00	5.00	3.00	0.50	5.00	5.00	5.00	2.00	4.00	1.00

TABLE 16

NODE 3-10 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-10 Provision for high level combination/Model complexity management
Lower level connections:
1) Access to stand alone code libraries, 4-4
2) Software modularity, 4-6
3) High level model language, 4-7
4) Model debugging support/verification, 4-10
5) Incorporation of special code implementation and "packaging" within models/ Extension of high level constructs, 4-12
6) Provisions for different levels of modeling detail, 4-14
7) Information and decision processes modules, 4-18
8) Validation: Model operation correspondence to the real system, 4-19
9) Physical component representation correspondence, 4-20

Pairwise weights

Col / Row	1	2	3	4	5	6	7	8	9
1	1.000	0.200	0.200	0.333	0.333	0.200	0.250	0.333	0.333
2	5.000	1.000	1.000	3.000	3.000	1.000	3.000	5.000	3.000
3	5.000	1.000	1.000	5.000	2.000	0.333	0.500	3.000	1.000
4	3.000	0.333	0.200	1.000	0.200	0.143	0.200	0.333	0.250
5	3.000	0.333	0.500	5.000	1.000	0.333	0.333	1.000	0.500
6	5.000	1.000	3.000	7.000	3.000	1.000	3.000	4.000	3.000
7	4.000	0.333	2.000	5.000	3.000	0.333	1.000	5.000	5.000
8	3.000	0.200	0.333	3.000	1.000	0.250	0.200	1.000	1.000
9	3.000	0.333	1.000	4.000	2.000	0.333	0.200	1.000	1.000

TABLE 17

NODE 3-11 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-11 Size of model supported
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	9.000
2	0.111	1.000

TABLE 18

NODE 3-12 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-12 Basic memory requirements
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	5.000
2	0.200	1.000

TABLE 19

NODE 3-13 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

3-13 Execution speed
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	7.000
2	0.143	1.000

TABLE 20

NODE 4-1 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-1 Graphics / User interface capabilities
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.143
2	7.000	1.000

TABLE 21

NODE 4-2 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-2 Ease of learning the base language
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	5.000
2	0.200	1.000

TABLE 22

NODE 4-3 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-3 Integrated software toolset
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.111
2	9.000	1.000

TABLE 23

NODE 4-4 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-4 Access to stand alone code libraries
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Col / Row	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 24

NODE 4-5 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-5 Code reusability
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Col / Row	1	2
1	1.000	0.143
2	7.000	1.000

TABLE 25

NODE 4-6 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-6 Software modularity
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Col / Row	1	2
1	1.000	0.111
2	9.000	1.000

TABLE 26

NODE 4-7 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-7 High level model language
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Col / Row	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 27

NODE 4-8 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-8 Structured model development approach
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 28

NODE 4-9 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-9 Output provisions
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	1.000
2	1.000	1.000

TABLE 29

NODE 4-10 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-10 Model debugging support/verification
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.333
2	3.000	1.000

TABLE 30

NODE 4-11 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-11 Statistical support
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	1.000
2	1.000	1.000

TABLE 31

NODE 4-12 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-12 Incorporation of special code implementation and "packaging" within models/Extension of high level constructs
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 32

NODE 4-13 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-13 Specialized component support at a high level
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	1.000
2	1.000	1.000

TABLE 33

NODE 4-14 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-14 Provision for different levels of modeling detail
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 34

NODE 4-15 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-15 Access to model code/On-line documentation
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 35

NODE 4-16 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-16 Modeling approaches supported
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	3.000
2	0.333	1.000

TABLE 36

NODE 4-17 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-17 Model code readability
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.333
2	3.000	1.000

TABLE 37

NODE 4-18 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-18 Information and decision processes modules
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 38

NODE 4-19 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-19 Validation: Model operation correspondence to real system
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Row \ Col	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 39

NODE 4-20 LOWER LEVEL CONNECTIONS PAIRWISE COMPARISONS

4-20 Physical component representation correspondence
Lower level connections:
1) Traditional, special purpose simulation systems, 5-1
2) OOP simulation system, 5-2

Pairwise weights

Col / Row	1	2
1	1.000	0.200
2	5.000	1.000

TABLE 40

SIMULATION EVALUATION FINAL PRIORITIES

	Weight
Traditional simulation system	0.242
OOP simulation system	0.758

As can be seen from the final priority vector listed in Table 40, the results of the AHP comparison procedure indicate that an OOP simulation system is preferable to the traditional simulation systems which currently dominate modeling activities.

Verbal Environment Comparison

For the sake of continuity, the verbal comparison of the two simulation approaches shall be pursued using a similar top level breakdown as was developed for the AHP model. Namely, the major topics in this discussion are: simulation language developer effectiveness, simulation modeler effectiveness, model effectiveness, and performance considerations.

The difference between software development in traditional simulation environments and an OOP simulation environment is caused entirely by the difference between the new OOP languages and the older procedural languages. As mentioned in Chapter III, Object Oriented Programming has the features of encapsulation, message passing, dynamic binding, and inheritance. These features positively influence software development in OOP environments as compared to procedural environments in several ways. First, understandability of classes is improved because they represent the data and method implementations of a coherent concept rather than the loose combination of multiple procedural routines. Secondly, the four features of OOP improve the ease with which already developed software systems can be maintained and modified. By encapsulating the data and methods which use the data, internal class implementations can be altered while instances of the class retain the same message passing relationships to other objects in a software system. Finally, base language code is reusable through inheritance (definition of new

subclasses) and through the use of instances of a class as an internal component of new classes.

In addition to the impact of these key features, the base language development support environment made up of code testing capabilities, debugging windows, integrated editing and compiling along with graphics capabilities and well developed code libraries (a fleshed out class hierarchy) significantly improve the ease and speed with which a complex software application can be conceptualized, implemented, and tested (and revised and maintained, in later versions).

The impact of OOM on simulation modeler effectiveness is probably the most important characteristic uncovered in this research. On the negative side, it was determined that only after a modeler has a thorough understanding of the simulation class library and the general structure of the OOP language is the individual capable of developing models with any amount of speed and reliability. However, when one considers that a certain level of learning is required to become adept at modeling in the traditional simulation environments, we conclude that this is only a mild drawback.

The most significant positive benefit for the simulation modeler is the new correspondence between simulation modeling objects and real system objects. From experience, one recognizes that one of the most difficult aspects of traditional simulation model development is the level of creativeness which must be utilized. Rather than working with

one-to-one relationships (model to real system), the modeler is faced with the task of modeling real system objects as a conglomeration of simulation language building blocks which represent separate activities or characteristics from objects in the system. Within OOM, the modeler is able to construct simulation models from a group of high level building blocks, each of which are a software representation of a full, coherent system object. Also, because in traditional languages new high level constructs cannot be implemented by the modeler, a degree of creativity is required when modeling complex systems. Because the actions of traditional constructs do not always agree with the activities of objects in a real system, the modeler is often obliged to work around the restrictions by creating a complex model network or dropping down to base language coding (thereby causing problems in validation, model communication, etc.).

Another benefit of OOM is that objects in a model do not have direct connections with one another. Rather, as described earlier, the linkages between objects are defined by the structure of the routings for work flow items passing through the system model. In traditional simulation languages, model building blocks are linked together in order to provide for the routing of the entities transiting the system. This also results in direct relationships between the various model components. Because of the existence of these direct relationships, and because traditional model constructs only represent a portion of system objects, all of the

building blocks in traditional environments have strict interrelationship rules which must be followed for their appropriate combination. In OOM, because objects must be and are designed as self sufficient entities, these interconnection rules are non-existent. Rather, the modeler must simply be sure to use class instances which have the internal characteristics needed to model the system components of interest.

Another problem with the use of the high level portion of traditional simulation environments is that although these languages may be extremely well documented, not all of the characteristics of high level constructs may be presented in the documentation. Where differences in modeler assumptions or understanding and language implementations occur, there is the distinct possibility of modeling or results interpretation errors. In the case of OOM, in addition to the ability to thoroughly document a simulation class, a modeler always retains the ability to peruse the software representation of a class in order to obtain an exact understanding of the object's operation.

Due to the fact that models are more communicable, modeler effort is reduced when previously written models, possibly authored by another individual, must be understood, reused, and/or updated. As object oriented models are easier to understand regarding both components and linkages, the model learning effort is reduced and the degree of uncertainty is decreased. For models written in traditional environments,

a greater amount of time and effort must be expended toward understanding archived models and the degree of uncertainty may still be significant.

Model effectiveness is also affected significantly by OOM characteristics. OOM models are much easier to alter and maintain than models in traditional languages. This is due to three characteristics of OOM. First, the components of OOM models are encapsulated and do not share memory allocation. Therefore, there is little concern for the duplication of object parameter values (entity file numbers, etc.) that there is in traditional modeling. Secondly, because object oriented models are constructed of independent objects and are devoid of direct interobject linkages, new model components can be added or deleted and routings quickly changed without effecting the model structure significantly. Finally, because the OOM models themselves are significantly easier to understand, the effort required to understand available model files is less than that required for comparable traditional simulation models.

As discussed previously, OOM models are implemented in a manner which improves their communicability. In addition to improving the simulation modeler effectiveness, this characteristic improves the model effectiveness as well because it increases the ease with which model results can be related to the real system, and thereby, "sold" to decision makers.

From the standpoint of performance considerations, the OOM environment is not as capable as the traditional simulation environment. First of all, the amount of time needed to execute an OOM model is roughly six times longer. This statement is made from experience in the validation activities in which a SLAM model and an OOM model of the same system were executed repeatedly on the same hardware. Reasons for this characteristic are: 1) OOP languages are not typically as efficient in their execution as procedural languages and 2) A significant amount of memory allocation and deallocation (object creation and garbage collection) is involved in the execution of OOM models. Another deficiency of the OOM system is that it is not as efficient in its use of computer memory as the traditional systems. The object oriented environment was not able to contain the same size model (large numbers of work flow items) in the given memory. In addition, the base amount of memory necessary for the prototype system is higher than for traditional modeling environments. These attributes are not perceived as strong drawbacks because of the continued increases in computer processing speeds and the continued reductions in the cost of computer memory.

From this discussion, which is based on the researcher's experience with procedural languages, traditional simulation languages, object oriented programming, and the prototype OOM environment, the conclusion is reached that OOM is the next step in the continuing progression towards improved simulation

capabilities.

Summary

There are several important outcomes of this chapter. First, an AHP model structure appropriate for the comparison of simulation environments has been described. Also, the pairwise comparison matrices determination and the weight composition process and results have been presented. Next, a textual comparison of the two simulation alternatives was performed. The final result of the comparison activities is the conclusion that an Object Oriented Modeling approach to simulation is superior to traditional simulation environments.

CHAPTER VIII

FUTURE RESEARCH DIRECTIONS IN OOM

This chapter describes the development plan which has been created to facilitate the coherent expansion of object oriented simulation capabilities. The material covered includes the research and development activities which are planned and a time phased plan indicating the order and timing for these efforts.

Introduction

The determination of the conceptual organization of an OOM environment and the implementation of a prototype OOM system were two significant phases in this research project. However, these two steps in themselves are not sufficient to ensure the continuation and success of this modeling paradigm. The prototype OOM system is just that, a prototype system. This prototype contains only the basic conceptual and implementational structure necessary to establish an object oriented, discrete event simulation capability. It is the intention of the researcher to use this initial structure as the conceptual core around which additional simulation modeling and data management

capabilities will be added. The remainder of this chapter has been divided into discussions of appropriate avenues of effort and the time phased research plan.

Appropriate Areas for Research and Environment Extension

The following areas are those top level groupings which the researcher feels should be addressed in improvements to the developed OOM simulation system.

1) Random number generation features
2) Simulation element and processing object classes and class abilities
3) Measures of performance capabilities
4) Model data management
5) Support for continuous simulation
6) Improved modeler interface capabilities, both for model input and results output

These are listed roughly in order of increasing conceptual and implementation difficulty as perceived by the author. A discussion on each of these areas is presented in the rest of this section.

Although the complete structure for random number generation has been designed and the capability for generating a significant number of distributions has been provided in the prototype system, several distributions have not been supplied. The generating methods for these distributions, which include the Poisson and Erlang, and the method for generating observations from a user supplied

discrete probability function are not currently present. With a limited amount of effort the necessary class and method definitions will be added to the system.

As completed, the OOM prototype system has certain basic simulation element and processing object classes. An important, necessary step in expanding the environment features is to improve this class library by: 1) Enhancing the operation of current classes and 2) Adding additional classes. Enhancements which will be added to existing classes include:

- Provision for the handling of more queue ordering disciplines by the QueueObject class
- Provision for multiple executions of the same model
- Completing the implementation of the setup time specification and use by simulation element objects, including monitoring of idle, busy, and setup time statistics
- Provision for the storage and manipulation of attribute values associated with instances of the Work Flow Item class
- Provision for information access links between simulation element objects and information flow objects (see below)
- Provision for alternate processing locations specified in the Routing class (for systems where alternate routings may be applicable)
- Provision for the interaction between simulation element objects representing processing stations and simulation element objects representing material handling entities (e.g., a central material handling robot interacting with the several machines in a manufacturing cell)
- Event scheduling and initiation based on day and time tracking (e.g., Monday, 10:03am)
- Provision for dynamic system operation based on day

and time of day (e.g., work order arrivals occur at a different rate during the day rather than at night)

The benefits of these improvements will be to allow more complete and correct system modeling without dropping into general purpose language coding and without "tricking" a certain system operation within a network representation. Additional classes which will be added to the simulation subtree are numerous and include:

- Classes representing the basic types of material handling equipment such as fork lifts, automatic guided vehicle systems, material handling robots, conveyors, etc.
- Classes which are used to group interacting sets of items together (e.g., machine and material handling objects) to allow complete coordination of mutual activities
- Classes representing information flow elements which interact with simulated machine elements for work flow item processing
- Classes representing inspection stations having the ability to arrange rework routings for rejected work flow items
- Processing classes providing the capability to perform decision processes regarding parameters of simulated system components

Another portion of changes which will be performed on the OOM system addresses performance issues. Specifically, the execution time of object oriented models is long when compared to that for traditional systems. Although the author does not feel that slow execution time makes OOM infeasible, it appears advisable to attempt to improve execution time. From OOM experimentation, it is obvious that a large amount of overhead memory management is occurring due to allocation and deallocation of memory for

transient objects (queue storage locations and work flow items). A potential remedy which will be pursued is to "pre-allocate" space for a modeler specified number of these objects, retain this space on a "next available storage location" list, and, thereby, virtually eliminate this repetitive activity. The effect of the implementation of this remedy should be to significantly reduce model execution times.

Another area which will be addressed in further OOM research is the ability to have additional performance measures monitored during simulation execution. A principle measure which will be researched and possibly implemented is the tracking of cost data associated with storing, servicing, or processing work flow items. By having a library of simulation objects with cost data for specific machines attached, the simulation analyst or decision maker can determine the economic impact of routing options and work schedule changes. A second characteristic of systems to be considered is the quality of product which has been processed through multiple operations at multiple stations. As with cost data, it would be desirable to have a library of simulation objects with output quality modeling information for specific machines attached. Using these simulation objects, a designer or manufacturing planner will be able to determine the ability of a certain manufacturing plan or manufacturing system to fulfill specified output quality requirements.

From the discussion in the last paragraph, the reader can observe that one long range goal for this OOM environment is to develop a simulation system which answers questions in a number of previously unaddressed areas. In order to meet this capability, a large amount of data shall need to be input, tracked, maintained and analyzed. In order to facilitate this data management without forcing potentially cumbersome links to external data base packages, an internal object oriented data management capability will be developed. This data management will be designed and implemented with the specific objective of assisting the simulation modeler and analyst in the intelligent use and maintenance of data which is directly applicable to simulation modeling endeavors. By providing current system status, cost, and quality information in an organized format and having background links from this information to specific simulated service or manufacturing system components, the objective of simulation directed data management can be achieved.

The current OOM system provides the ability to perform discrete event simulation on systems of independent software objects. One of the enhancements mentioned above involves the grouping of sets of top level objects together to allow non-independent object interactions to be carried out successfully. Using this as a conceptual basis for event or timing synchronization, a further OOM system research area is the development of a continuous simulation capability

(either differential or difference equation based continuous simulation) which can co-exist with discrete event modeling elements.

The prototype OOM system has the input and output methods commonly available in traditional simulation systems, namely model text files for input and results text files for output. At the prototype level, this was sufficient for all modeling tasks because the OOM system capabilities are limited and easily understood by modelers. Once the improvements to the OOM system just mentioned are added, without providing user assisting interface capabilities the environment will have the potential to overwhelm all but the most dedicated user. In order to avoid this significant problem, it will be necessary to greatly improve upon the basic windowing system provided with Smalltalk by applying it specifically to the simulation modeling and analysis functions. Requirements in this area consist of:

- Iconic model and work flow item routing construction possibly driven from a 2-D location grid
- Menu driven specification of model parameters (e.g., choice of processing and setup time distributions)
- Graphic window presentation of simulation results (e.g., graphing queue length or queue waiting time versus time)

In addition to the "incremental" interface improvements, a significant need exists for a top level simulation executive controller which will provide both runtime and model development support and integrate the data management

functions mentioned previously. The incremental improvements just mentioned will be integrated with other model development and simulation runtime features within this controller. A runtime interface improvement will be a top level runtime executive which provides the ability to halt the execution of a model and view the status of system components in a structured manner. Research and experimentation will also address the feasibility of providing system optimization features within the runtime executive controller. Also integrated, a structured on-line help capability will utilize methods and data attached to classes in the simulation library to provide information upon request to the developer or modeler. This information might include class usage recommendations, class capability descriptions, and/or class operation documentation.

The following section presents a first cut at an obviously dynamic timed research plan.

Phased Research Plan

The activities described in the previous section are grouped according to the following classifications:

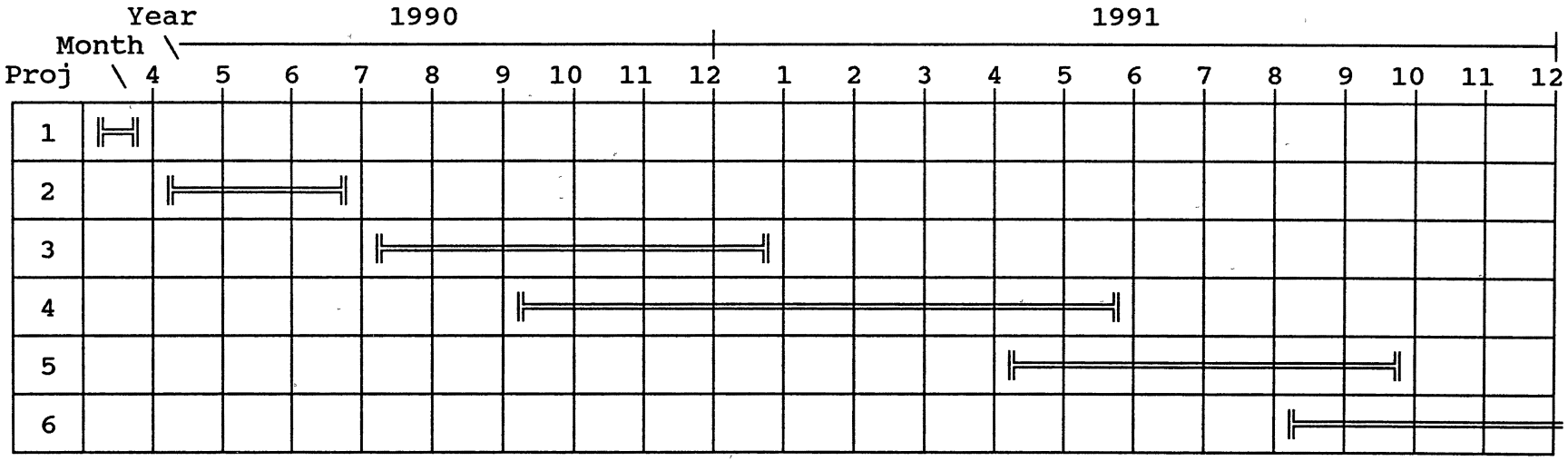
1) Random number generation features
2) Simulation element and processing object classes and class abilities
3) Measures of performance capabilities
4) Model data management
5) Support for continuous simulation

6) Improved modeler interface capabilities, both for model input and results output

The timing plan for these further research activities is illustrated in Figure 26 and addresses each of these six groups and any possible interaction and dependency. Random number generation enhancement (1) has no interaction with the other efforts and will be, therefore, pursued separately. This project is scheduled to be started and completed in one month (as shown on Figure 26), April 1990.

The extension of simulation element and processing object classes and class abilities will involve adjustments to currently available classes and the definition of new classes within the conceptual context already defined for the OOM environment. Because the conceptual basis for OOM will not be altered, this activity is relatively independent of the other research projects. It is scheduled to consume approximately three months of work (5/90 - 7/90, inclusive).

The next two projects, measures of performance improvements and model data management features, are interrelated and the research and developmental effort for the two will overlap in timing significantly. Measures of performance improvements will involve two months of up front work to provide the conceptual basis for the implementation of the discussed concepts. After this approach is determined, the model data management project will begin and the implementation for both projects will use the requirements of measures of performance improvements to



Start and end of a research activity are specified by the symbols ⊩ and ⊣, respectively.

Figure 26. Time Phased Plan for Further Research

drive some of the concepts and implemented capabilities of the data management effort.

Towards the end of the fourth project (data management), the continuous simulation capability development will begin. This overlap will be necessary because the interaction between objects within a continuously simulated system will be significantly greater than among objects in a discrete event system. Approximately six months of time will be consumed in the design and implementation of this simulation capability.

A very long task timed to begin upon completion of the previous five is the construction of a window and graphics based simulation user interface. This interface will integrate all previously defined software features together within a single application providing guidance through model construction, data management, and results interpretation. Because this is anticipated to be of extremely long duration, Figure 26 shows this project continuing on beyond the end of 1991.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

The goal of this research was to investigate the feasibility and benefits of an Object Oriented Modeling environment. To achieve this goal, four research objectives were established (also see Chapter IV). The first part of this chapter discusses the conclusions from this research in the context of these objectives.

The first of the research objectives was to develop a set of object oriented classes which provide the ability to generate simulation models. In order to fulfill this objective, several tasks had to be performed. First, the methods of simulation model operation and representation had to be conceptualized. Next, this conceptual organization had to be applied to the implementation of object oriented classes providing the ability to create a model of a demonstration target system. The resulting simulation system was then tested against an accepted standard to provide validation of the operation of the software. Finally, information from the target system was combined with the simulation classes to result in a completed simulation model, 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. The AHP provides a theoretical and practical framework for decomposing decision problems involving multiple quantitative and qualitative criteria into manageable units. For the problem of interest, namely comparison of simulation environments, a rather large AHP model was created. This involved the determination of an appropriate scheme for decision process decomposition along with the linkages between elements in the decision model. Thus the second research objective was successfully completed.

The third objective of the research, evaluation of an Object Oriented Modeling system for simulation modeling, was performed using two approaches. The first approach involved the application of the AHP decision model in the comparison of the new OOM system to traditional simulation systems. 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. These comparisons were manipulated to result in a final set of weights indicating the preferable simulation approach. The conclusion

resulting from this AHP model application is that a simulation environment constructed following the object oriented programming paradigm represents an improvement over the currently available traditional simulation systems. The second approach to the simulation environment problem was to use the developed criteria as the basis for a logical, textual comparison of the two simulation alternatives. The conclusion from this unstructured, multicriteria discussion agrees with the results from the AHP analysis, thus successfully completing the third research objective.

The fourth and final research objective was to create a plan stating the timing and areas of activity for a sequence of project phases leading to an expanded OOM environment. The developed prototype environment provides the basic features required to be considered a viable simulation modeling alternative. Only by increasing the available features into new areas will the OOM approach gain acceptance and wide use. Included within the developed framework for expansion are improvements in random number generation, increases in the number and scope of reusable classes for modeling, measures of performance enhancements, features for simulation model data management, incorporation of continuous simulation, and development of a top level user interface and execution controller. This phased research plan represents the accomplishment of the fourth research objective.

The final recommendations from this research are simple and to the point:

1) Current simulation modeling software is excellent and has met and continues to meet the needs of many applications, however, object oriented programming provides a framework for simulation software implementation which allows improvements in the accomplishment of these traditional modeling tasks.

2) Future modeling environments will need to support interaction across a broad range of users and developers and provide a significant level of functionality.

Characteristics and features of these future environments must significantly support the efforts of the manufacturing engineer, who may have limited knowledge of simulation and the simulation environment, and the efforts of the simulation model and simulation environment developers, who will have more complete technical knowledge of simulation and the simulation environment. Simulation environments must begin to allow for the creation of models composed of multiple levels (physical processes, information processes, decision processes, etc.), along with acting as a "simulation workbench" supporting the analyst. The AHP analysis conducted as part of this research shows that an Object Oriented Modeling approach to simulation provides a robust environment which is able to achieve this expanded modeler functionality while providing a framework within which significant software modifications can be performed.

3) Simulation modeling within an object oriented implementation should be pursued both by simulation package developers and simulation system users.

BIBLIOGRAPHY

Abed, S. Y., T. A. Barta, and K. L. McRoberts. "A Qualitative Comparison of Three Simulation Languages: GPSS/H, SLAM, SIMSCRIPT." Computers and Industrial Engineering, Vol. 9, No. 1, pp. 35-43.

Abed, S. Y., T. A. Barta, and K. L. McRoberts. "A Quanitative Comparison of Three Simulation Languages: GPSS/H, SLAM, SIMSCRIPT." Computers and Industrial Engineering, Vol. 9, No. 1, pp. 45-66.

Adelsberger, H. H., U. W. Pooch, R. E. Shannon, 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 Modelling 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.

Alexander, J. M. and Saaty, T. L., The Forward and Backward Processes of Conflict Analysis, Behavioral Science, 1977, Vol. 22, pages 87-98.

Alexander, J. M. and Saaty, T. L., Stability Analysis of the Forward-Backward Process: Northern Ireland Case Study, Behavioral Science, 1977, Vol. 22, pages 375-382.

Armstrong, F. B. and S. Sumner. "The Project Approach to Simulation Language Comparison." Proceedings of the 1988 Winter Simulation Conference, Atlanta, GA, 1988, pp. 636-645.

Banks, J. and J. S. Carson. Discrete-Event System Simulation. Prentice-Hall: Englewood Cliffs, NJ, 1984.

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.

CACI, Inc. Promotional material, Industrial Engineering, November, 1988, pp. 1.

Cammarata, S., B. Gates, and J. Rothenberg. "Dependencies and Graphical Interfaces in Object-Oriented Simulation Languages." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 507-517.

Corbin, M. J. and G. F. Butler. "Object oriented simulation in FORTRAN." Tools for the Simulation Profession, 1989, The Society for Computer Simulation: San Diego, CA, 1988, pp. 29-43.

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

Cox, B. and B. Hunt. "Objects, Icons, and Software-IC's." Byte, August, 1986, pp. 161-176.

Curry, G. L., B. L. Deuermeyer, and R. M. Feldman. "cdf/BOSS: A Language and MicroComputer Implementation for Discrete Simulation." Computers & Industrial Engineering, Vol. 15, Nos. 1-4, pp. 104-112.

Digitalk, Inc. Smalltalk/V Tutorial and Programming Handbook. Digitalk, Inc.: Los Angeles, CA, 1986.

Doman, A. "Object-Prolog: Dynamic Object-oriented Representation of Knowledge." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 83-88.

Elmaghraby, A. S. and V. Jagannathan. "An Expert System for Simulationists." Artificial Intelligence, Graphics, and Simulation, The Society for Computer Simulation: San Diego, CA, 1985, pp. 106-109.

Endesfelder, T. and H. Tempelmeier. "The SIMAN Module Processor - A Flexible Software Tool for the Generation of SIMAN Simulation Models." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation: San Diego, CA, 1987, pp. 38-43.

Eversheim, W. "Graphic Interactive Simulation for the Planning of Manufacturing Systems." Journal of Manufacturing Systems, Vol. 6, No. 2, pp. 151-156.

Farnsworth, K. D., V. B. Norman, and T. A. Norman. "Integrated Software for Manufacturing Simulation." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 195-201.

Ford, D. R., B. J. Schroer, and R. Daughtrey. "An Intelligent Modeling System for Simulation Manufacturing Processes." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 525-529.

Ghaznavi-Collins, I. and D. Thelen. "An Object Oriented Approach toward System Architecture Simulation." AI Papers, 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.

Gold Hill Computers. Golden Common LISP Operating Guide, Version 1.1. Gold Hill Computers: Cambridge, MA, 1987.

Goldberg, A. SMALLTALK-80: The Interactive Programming Environment. Addison-Wesley: Reading, MA, 1984.

Gordon, R. F., E. A. MacNair, K. J. Gordon, and J. F. Kurose. "A Visual Programming Approach to Manufacturing Modeling." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 465-471.

Grant, J. W. and S. A. Weiner. "Factors to Consider in Choosing a Graphically Animated Simulation System." Simulation: Modeling Manufacturing & Service Systems, Industrial Engineering and Management Press: Norcross, GA, 1987.

Griesmeyer, J. M. "Generalized Simulation Environment for Factory Systems." Tools for the Simulation Profession 1989, The Society for Computer Simulation: San Diego, CA, 1988, pp. 20-29.

Haddock, J. "A Simulation Generator for Flexible Manufacturing Systems Design and Control." IIE Transactions, Vol. 20, No. 1, pp. 22-31.

Henriksen, J. O. "The Integrated Simulation Environment (Simulation Software of the 1990's)." Operations Research, Vol. 31, No. 6, 1983, pp. 1053-1072.

Higdon, J. "Planning a New Material Handling System." Industrial Engineering, November, 1988, pp. 55-59.

Hilton, M. L. "A Multi-level Event Scheduling Mechanism for Supporting Intelligent Objects." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 127-130.

Hitchens, M. W. "Simulation: The Key to Automation Without Risk." CAD/CAM Technology, Vol. 3, No. 3, pp. 15-17.

Kachitvichyanukul, V. "Simulation Environment of the 1990's (Panel)." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 455-460.

Kaehler, T. and D. Patterson. "A Small Taste of Smalltalk." Byte, August, 1986, pp. 145-159.

Karian, Z. A. "Software Review: GPSS/PC." Byte, October, 1985, pp. 295-301.

Khoshnevis, B. and A. Chen. "An Automated Simulation Modeling System based on AI Techniques." Simulation and AI, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation: San Diego, CA, 1987, pp. 87-91.

Khoshnevis, B. and A. Chen. "An Expert Simulation Model Builder." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation: San Diego, CA, 1986, pp. 129-132.

Khoshnevis, B. and A. Chen. "EZSIM: An Automated Simulation Model Building System." Working Paper, Department of Industrial and Systems Engineering, University of Southern California.

Khoshnevis, B. and W. M. Austin. "An Intelligent Interface for System Dynamics Modeling." Simulation and AI, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation: San Diego, CA, 1987, pp. 81-86.

Khoshnevis, B., W. M. Austin, A. Chen, and Q. Chen. "Intelligent Simulation Environments for Systems Modeling." 1988 International Industrial Engineering Conference Proceedings, Institute of Industrial Engineers, Orlando, FL, 1988, pp. 25-29.

Kilgore, R. A. and K. J. Healy. "Animation Design with Cinema." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 261-268.

Kimbler, D. L. and B. A. Watford. "Simulation Program Generators: A Functional Perspective." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 133-136.

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

King, C. U., S. S. Adams, and E. L. Fisher. "Representation of Manufacturing Entities." Intelligent Manufacturing: Proceedings from the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston, SC, 1987, pp. 77-91.

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

Kreutzer, W. "A Modellers's Workbench - Simulation based on the Desktop Methphor." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 43-48.

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

Kumar, A. and S. Y. W. Su. "Object Manipulation in an Object-Oriented Semantic Association Model (OSAM*)." Manufacturing International '88, Atlanta, GA, 1988.

Law, A. M. and S. W. Haider. "Selecting Simulation Software for Manufacturing Applications: Practical Guidlines & Software Survey." Industrial Engineering, May, 1989, pp. 33-46.

Law, A. M. and W. D. Kelton. Simulation Modeling and Analysis. McGraw-Hill: New York, NY, 1982.

McGinnis, L. F. and M. Goetschalckx. "An Engineering Workstation for Computer Aided Engineering of Material Handling Systems." Manufacturing International '88, Atlanta, GA, 1988, pp. 137-142.

Meyer, B. "Bidding Farewell to Globals." Journal of Object-Oriented Programming, Vol. 1, No. 3, pp 73-76.

Miller, J. A., O. R. Weyrich, Jr., and D. Suen. "A Software Engineering Oriented Comparison of Simulation Languages." Tools for the Simulation Profession 1989, The Society for Computer Simulation: San Diego, CA, 1988, pp. 97-104.

Moseng, B. "The Process Planner's Work Place Today and Tommorrow." Robotics & Computer-Integrated Manufacturing, Vol. 1, No. 3/4, pp. 237-244.

Murray, K. J. and S. V. Sheppard. "Automatic Model Synthesis: Using Automatic Programming and Expert Systems Techniques toward Simulation Modeling." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 534-543.

Myler, H. R. "Object-Oriented Training Simulation." AI Papers, 1988, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation: San Diego, CA, 1988, pp. 156-160.

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.

O'Keefe, R. M. "What is Visual Interactive Simulation? (and is there a Methodology for doing it right?)." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 461- 464.

Oren, T. I., B. P. Zeigler, and M. S. Elzas, editors. Simulation and Model-Based Methodologies: An Integrative View. Springer-Verlag: Berlin, West Germany, 1984.

Oren, T., and K. Aytac. "Architecture of MAGEST: A Knowledge-based Modeling and Simulation System." Simulation in Research and Development, Elsevier Science Publishers: North-Holland, 1985, pp. 99-109.

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

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

Pritsker and Associates, Inc. Promotional material, Industrial Engineering, November, 1988, pp. 16a-16d.

Reilly, K., W. T. Jones, and P. Dey. "The Simulation Environment Concept Artificial Intelligence Perspectives." Artificial Intelligence and Simulation, The Society for Computer Simulation: San Diego, CA, 1985, pp. 29-34.

Rembold, U. and P. Levi. "The Factory of the 90s." Computers in Mechanical Engineering, March/April, 1988, pp. 26-28 and May/June, 1988, pp. 30-37.

Robinson, J. T. and P. J. Otaduy. "An Object-oriented Simulation Package for Power Plants." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 55-58.

Saaty, T. L., Concepts, Theory, and Techniques: Rank Generation, Preservation, and Reversal in the Analytic Hierarchy Decision Process, Decision Sciences, 1987, Vol. 18, pages 157-177.

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

Saaty, T. L. and Ramanujam, V., An Objective Approach to Faculty Promotion and Tenure by the Analytic Hierarchy Process, Research in Higher Education, 1983, Vol. 18, Number 3, pages 311-331.

Sathi, N., M. Fox, V. Baskaran, and J. Bouer. "An Artificial Intelligence Approach to the Simulation Life Cycle." A Technical Brief, Carnegie Group, Inc., Pittsburgh, PA, 1987.

Schriber, T. J. "The Nature and Role of Simulation in the Design of Manufacturing Systems." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation: San Diego, CA, 1987, pp. 5-18.

Schroer, B. J. and F. T. Tseng. "Modeling Complex Manufacturing Systems using Simulation." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 677-682.

Shannon, R. E. "Models and Artificial Intelligence." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 16-24.

Shannon, R. E. Systems Simulation: The Art and Science. Prentice-Hall: Englewood Cliffs, NJ, 1975.

Software Survey: Simulation. The Engineering Software Report, October, 1986, pp. 3-7.

Standridge, C. R., A. A. B. Pritsker, and C. W. Stein. "A Tutorial in TESS(tm): The Extended Simulation Support System." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 238-246.

Stauffer, R. N. "Graphic Simulation Answers Preproduction Questions." CAD/CAM Technology, Vol. 3, No. 3, pp. 11-17.

Stroustrup, B. The C++ Programming Language. Addison-Wesley: Reading, MA, 1986.

Systems Modeling Corp. Promotional material, Industrial Engineering, January, 1989, back cover.

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

Terrell, M. P. and T. Chen. "The Development and Application of a Utility Simulation Program for Design and Analysis of Complex Conveyor Systems Using SIMSCRIPT II.5." Abstracts for the International Symposium on Extremal and Systems Analysis, September, 1977a, pp. 166-167.

Terrell, M. P. and T. Chen. "The Study and Development of a Utility Simulation Model for Conveyor System Analysis Using SIMSCRIPT II.5." Proceedings of the First International Conference on Mathematical Modeling, August, 1977b, pp. 669-680.

Terrell, M. P. and L. Bussey. "A Model for Analyzing Closed-Loop Conveyor Systems with Multiple Work Stations." Proceedings of the 1973 Winter Simulation Conference, San Francisco, CA, 1973.

Terrell, M. P., R. Gourley, and T. Chen. "The Development of a General Purpose Conveyor Systems Simulation Model Utilizing a Modular Format." Proceedings of the 1975 Summer Computer Simulation Conference, San Francisco, CA, 1975.

Tesler, L. "Programming Experiences." Byte, August, 1986, pp. 195-206.

Thomas, D. "The Time/Space Requirements of Object-Oriented Programs." Journal of Object Oriented Programming, March/April, 1989, pp. 71-73.

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.

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

Unger, B., A. Dewar, J. Cleary, and G. Birtwistle. "A Distributed Software Prototyping and Simulation Environment: JADE." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation: San Diego, CA, 1986, pp. 63-71.

Vesterager, J., K. E. Wichmann, R. E. Young, and J. Heide. "Simulation Uses in CIM Development." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation: San Diego, CA, 1987, pp. 95-100.

Wales, F. J. and P. A. Luker. "An Environment for Discrete Event Simulation." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation: San Diego, CA, 1986, pp. 58-62.

Wallace, J. C. "The Control and Transformation Metric: Toward the Measurement of Simulation Model Complexity." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 597-603.

Wilson, R. "Object-oriented languages reorient programming techniques." Computer Design, November 1, 1987, pp. 52-62.

Wyvill, B. "Current Trends in Graphics and Animation." Artificial Intelligence, Graphics, and Simulation, The Society for Computer Simulation: San Diego, CA, 1985, pp. 49-53.

Zahedi, F., The Analytic Hierarchy Process - A Survey of the Method and its Applications, Interfaces, July-August 1986, Vol. 16, Number 4, pages 96-108.

Zeigler, B. P. Multifacetted Modelling and Discrete Event Simulation. Academic Press: London, 1984.

Zeigler, B. P. Theory of Modelling and Simulation. Robert E. Krieger Publishing Co.: Malabar, FL, 1976.

VITA

Terrence Gilbert Beaumariage

Candidate for the Degree of

Doctor of Philosophy

Thesis: INVESTIGATION OF AN OBJECT ORIENTED MODELING ENVIRONMENT FOR THE GENERATION OF SIMULATION MODELS

Major Field: Industrial Engineering and Management

Biographical:

Personal Data:
Born in Washington, Pennsylvania, July 10, 1961, the son of Gilbert and Karoline Beaumariage.

Education:
High School - Canon-McMillan Senior High, Canonsburg, Pennsylvania. Graduated as Salutatorian, May, 1979.
Undergraduate - Rochester Institute of Technology, Rochester, New York. Received a Bachelor of Science degree with Highest Honors in Industrial Engineering, May 19, 1984.
Graduate - Oklahoma State University, Stillwater, Oklahoma. Received a Master of Science degree in Industrial Engineering, May, 1987. Completed requirements for the Doctor of Philosophy (majoring in Industrial Engineering) in May, 1990.

Professional Experience:
Assistant Professor, Department of Industrial and Management Systems Engineering, Arizona State University, August, 1989 to present.
Teaching Assistant, School of Industrial Engineering and Management, Oklahoma State University, September, 1988 to December, 1988 and September, 1984 to May, 1985.
Research Assistant, School of Industrial Engineering and Management, Oklahoma State University, September, 1984 to August, 1989.

Professional Experience (continued):

Material Handling Engineer, Telex Computer Products, Tulsa, Oklahoma, May, 1987 to August, 1987.

Manufacturing Engineer, IBM Entry Systems Division, Austin, Texas, May 1985 to August 1985.

Manufacturing Engineer, IBM Federal Systems Division, Owego, New York, December 1983 to February 1984.

Cost Engineer, IBM Federal Systems Division, Owego, New York, June, 1983 to August, 1983 and December, 1982 to February, 1983.

Assistant Engineer, O'Donnell and Associates, Inc., Pittsburgh, Pennsylvania, June, 1982 to August, 1982.

Assistant Engineer, Rochester Products Division of General Motors, Rochester, New York, December, 1981 to February, 1982 and June, 1981 to August, 1981.

Professional Activities:

Institute of Industrial Engineers
American Society for Quality Control (Certified Quality Engineer)
National Society for Professional Engineers
Arizona Society for Professional Engineers
American Society for Engineering Education
Society for Computer Simulation
Registered Engineering Intern (Oklahoma)

Honor Societies and Awards:

Alpha Pi Mu Industrial Engineering Honor Society (Arizona State University Chapter Faculty Advisor)
Phi Kappa Phi Honor Society
Tau Beta Pi Engineering Honor Society
National Science Foundation Graduate Fellowship
Alpha Pi Mu Scholarship
IIE Gilbreth Memorial Fellowship
RIT Outstanding Undergraduate Scholar
National Merit Scholar