12-1992

An Integrated Simulation Model Development Environment for Slam II Using Object-Oriented Paradigm

Rizvan Erol

AN INTEGRATED SIMULATION MODEL DEVELOPMENT ENVIRONMENT FOR SLAM II USING OBJECT-ORIENTED PARADIGM

by

Rizvan Erol

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Industrial Engineering

Western Michigan University
Kalamazoo, Michigan
December 1992

AN INTEGRATED SIMULATION MODEL DEVELOPMENT ENVIRONMENT FOR SLAM II USING OBJECT-ORIENTED PARADIGM

Rizvan Erol, M.S.

Western Michigan University, 1992

An integrated simulation model development environment was implemented to assist the modeler by automating certain activities of simulation modeling. The system included interactive model definition, experimental design, automatic simulation program generation in SLAM II. Object-oriented paradigm at software development stage was extensively used to conceptualize the structure, and rules of the SLAM II language in order to generate efficient, and modular program code. The present system targeted modeling of various probabilistic inventory control system problems. The remarkable advantages of the system were rapid model development time, and achieving reliable program code without requiring any knowledge in SLAM II. Object-oriented programming was very promising, and effective programming paradigm in system development cycle.

At the final stage, Response Surface Methodology (RSM) in conjunction with the steepest-ascent method was used to find the optimum inventory policy minimizing the average cost per unit of time based on the simulation output. Comparing the RSM results to those of the deterministic relaxation of the probabilistic inventory model demonstrated that RSM is an efficient tool for optimization in simulation.

ACKNOWLEDGMENTS

I should admit that I received significant contribution in the completion of this study from my professors. First, to my advisor, Dr. Tarun Gupta, I appreciate his guidance and support throughout the study. Without spending almost every weekend in his office, it would have not been possible to achieve the goals of this study. Also, I would like to thank my committee members Dr. Kailash Bafna, and Dr. Richard Munsterman for their review, and suggestions in preparing the final report. I thank Dr. Franklin Wolf for his constructive suggestions in this study.

Finally, I would thank the Turkish Government for supporting my graduate study at Western Michigan University. To all my family who continuously provided support and encouragement, I owe a debt of gratitude.

Rizvan Erol

Order Number 1351252

An integrated simulation model development environment for SLAM II using object-oriented paradigm

Erol, Rizvan, M.S.

Western Michigan University, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

TABLE OF CONTENTS

Table of Contents--Continued

CHAPTER

Table of Contents--Continued

CHAPTER

LIST OF TABLES

LIST OF FIGURES

List of Figures--Continued

List of Figures--Continued

NOMENCLATURE

h = Holding Cost ($ per unit per unit of time)

p = Backlog Cost ($ per unit per unit of time)

K = Setup Cost ($ per setup)

Q = Order Quantity (unit)

S = Reorder Level (unit)

s = Reorder Point (unit)

t = Review Period (unit of time)

r = Production, or Supply Rate (unit per unit of time)

t_w = Expected Waiting Time for Customer (unit of time)

t_c = Cycle length (unit of time)

L = Lead time (unit of time)

a = demand (depletion) rate (units per unit of time)

p_r = customer renege probability

p_{ij} = Probability of Substituting ith product by jth product

q_{ij} = Multiplication Factor between ith product and jth product

CHAPTER I

INTRODUCTION

Statement of the Problem

Simulation modeling has become the most powerful modeling tool in manufacturing systems area after the recent dramatic advancements in computer hardware, and software technology. A survey conducted by the Bell Labs showed that 80% of existing visual interactive modeling environments were dedicated to manufacturing problems, and among them simulation was the most preferred modeling tool (Haddock & O'Keefe, 1990). Paul (1991) sees the growing number of existing simulation software avalaible to the modelers as an indication of greater demand, and interest toward simulation.

Growing popularity of simulation can be attributed to the following factors: (a) introduction of computer assisted simulation environments with faster model execution speed and graphics facilities; (b) greater responsiveness required from current manufacturing systems due to the dynamics of business environment; (c) increasing need for modeling tools for systems with stochastic behavior and state-dependent decision mechanisms; (d) fewer rules to follow in simulation, and greater flexibility given to the modeler by simulation, and (e) the ease of interpretation of simulation results by decision-makers.

Today's manufacturing systems have more sophisticated and dynamic nature than the ones in the past. In some cases where analytical models happen to be inadequate or oversimplified, simulation may become the only tool to represent these

systems in greater detail with as few assumptions as possible (Chaharbaghi, 1990). Applications of simulation in manufacturing are broader due to its global and less restricted approach to systems modeling.

Recent dramatic improvements in both number and quality of simulation model development environments have played the major role in boosting the popularity and applicability of simulation to broader range of application fields. However, simulation studies still take considerable amount of time, and require a team of experts from various fields (e.g., application domain knowledge, computer programming, statistical analysis) (Balci & Nance, 1987). It is obvious that the ease with which complex systems can be modeled depends on the capabilities and features of the simulation software employed by the modeler. Many researchers have been trying to shorten the elapsed time during a simulation modeling study through artificial intelligence and expert systems methodologies.

Program coding holds the largest portion of the total elapsed time to finish a simulation study with its every aspect. Law and Haider (1989) estimated that program coding takes about 30-40% of the total simulation project time. This is why, program generators were the first examples of expert simulation systems introduced in the literature due to the significance of coding within problem solution time.

Although high-level programming languages such as FORTRAN, C and Pascal have the greatest flexibility in writing program code, a large amount of time in coding, debugging, and verifying the code has to be allocated. The lack of extensibility of the code written in these languages limits possible future modifications. The major motivation for the introduction of the first general-purpose simulation languages such as SLAM II and SIMAN was to overcome these difficulties in using high-level programming languages. These languages have built-in functions, nodes, and control

statements to perform frequently used operations for simulation entities, data collection, and reporting results. Unfortunately, computer programming and model development expertise are still required to employ any of the simulation languages, may be to a lesser degree (Law & Haider, 1989; Shannon, Mayer, & Adelsberger, 1985).

Crookes (1987) estimated that 70% of a simulation code could be generated with the use of generic models, and program generators. A generic family of model development classes based on a simulation language, for example SLAM II, can be used to generate simulation program code for various set of application domains (Ulgen, Thomasma, & Mao, 1989). Program generators can increase the efficiency of the modeler allowing him, or her to put more attention on the intellectual activities of simulation modeling. O'Keefe (1986) described program generators as Intelligent Front Ends (IFE) to existing simulation languages. Going one step further, modeler might need more diversified support from an assisting simulation system beyond program generation to automate the other tasks of simulation modeling (i.e., experimental design, statistical analysis, intelligent reasoning on the results, optimization). A trend that current simulation software systems have been mainly designed to automate simulation modeling activities was indicated by Paul (1991) in his extensive review of existing simulation software in the market.

In this study an integrated simulation model development environment based on the SLAM II simulation language using the object-oriented paradigm of the C++ language was implemented. Inventory control systems was the application domain of the proposed system. The system is basically a modular integration of some of simulation modeling tasks within a single computer system. These are: (a) experimental design, (b) program generator, (c) model execution, and (d) optimization of simulation

output. The system was developed in such a way that additional application domains could be added with little additional effort in the future.

Inventory control systems hold a major research interest in manufacturing systems area. An efficient inventory control system requires that its optimal control strategy and performance measures be established so that the total inventory cost can be minimized. Many analytical models subject to some pre-defined assumptions have been introduced in the literature to find the optimal inventory policy for a given inventory problem type (Koulamas, 1990; Park, 1989). Most mathematical models appeal to only a particular inventory problem type. The economic order quantity (EOQ) model is the simplest inventory model to find the optimum policy. However, as inventory problems become more and more complex with the inclusion of random customer arrivals, demand size, multiple items, and complex decision mechanisms, assumptions made by the analytical inventory models become unrealistic, for instance very small constant depletion rate, deterministic demand pattern. Moreover, establishing the mathematical relationships between system parameters may become an impossible task. At this point, the developed simulation modeling system became a powerful tool to analyze the effectiveness of complex inventory control mechanisms, and policies.

In general three types of inventory control mechanisms were considered in this study. These are as follows:

1. Continuous review (S,s) model: where S, and s are reorder level and reorder point in units respectively. Inventory level is reviewed after each inventory transaction (i.e., customer arrivals). When the inventory level is less than or equal to the reorder point, a new order is placed.

2. Periodic review model (S,s,t): t is time period to review inventory level. It has a similar decision-mechanism to the one in case 1 with one exception that review of inventory is performed after each t time units.

3. Production model with continuous review: In this case when the inventory level is less than or equal to s (reorder point), manufacturer starts production at a pre-defined production rate until the inventory level is equal to the S level.

Inventory systems having multiple items subject to the same, or different control mechanisms can be analyzed within a single simulation model using the modeling framework. Substitution of an inventory item by another can be modeled for cases that customer order may be met from other products in the system when there is a shortage in inventory on hand. Substitution mechanism causes an interaction between individual inventory levels of products in the system.

User-selected random distributions can be assigned to customer arrivals, demand size, lead time, and order processing time. The system also allows the user to define price breaks, and various backlog policies. The user can create various types of inventory models ranging from single-item inventory model to multi-item inventory model with complex decision mechanisms by simply changing appropriate values, and options enabled by the system.

As emphasized, automatic program generation alone cannot cover all aspects of a simulation study. The same emphasis should be also given to post-simulation analysis after the execution of simulation model to compare a set of alternatives. At this stage appropriate use of statistical tools becomes crucial due to existence of variations in system performance variables. Since simulation modeling, in some way, is to perform experiments on a given system by changing the levels of controllable factors, experimental design stage prior to model execution affects the statistical

evaluation of post-simulation analysis results. To address this issue the developed modeling system provided also a user-interactive computer program for experimental design. Experimental design were required in two-aspects of post-simulation analysis. These are:

1. Establishing regression meta-model to explore the relationship between controllable (independent) factors (i.e., reorder point (s), reorder level (S), time period (t), and production rate), and system performance variables(e.g., total inventory cost, average holding level, average profit, and etc.). Sensitivity analysis can also be performed using a regression model within a pre-defined solution region.

2. Finding the optimum inventory policy (i.e., the best levels of controllable factors). First-order model design (full or fractional orthogonal design) and second-order model (central composite design) designs were required in using Response Surface Methodology (RSM) to search for optimum policy. It should be noted that since the true total inventory cost function is unknown, existing analytical algorithms such as non-linear programming could not be employed here directly. However, certain analytical inventory models were used to narrow down the potential search region in optimization by simplifying stochastic simulation model into its deterministic equivalent. The accuracy of this estimation depends on both the complexity of the simulation model and the analytical model selected. Narrowing down the search region resulted in fewer simulation runs, and less computer CPU time. RSM was used in conjunction with steepest-ascent search method in optimization stage to perform additional simulation runs along the steepest path to improve the response variable (i.e., total cost). A detailed discussion of RSM and other optimization methods in simulation is provided in Chapter II.

Objectives

The increasing need for expert simulation systems is well emphasized by many authors (O'Keefe, 1986; Paul, 1991). The main objective in this study was to design an integrated simulation model development environment in which the user could develop and run simulation model within a short time. Other supportive objectives were as follows:

1. Design of user-friendly interactive model input system.
2. Design of efficient model retrieval, storage, updating mechanism for models.
3. Design of a reliable simulation program generator.
4. Design of experimental design module to create efficient, and correct design matrix for a given problem.
5. Application of RSM at optimization stage, and evaluation of its efficiency.
6. Comparison of the results calculated using appropriate analytical model with the ones using RSM.
7. Assessment of advantages of using object-oriented paradigm in the computer system development, and its potential contribution in further esearch.
8. Assessment and discussion of advantages of using the proposed model development system.

Organization of the Study

Basically in this study, major steps to achieve the proposed objectives were experimental design, model building, program generation, and execution, and finally optimization of the simulation output using RSM. A brief description of each remaining chapters in this report is as follows:

In Chapter II, a review of the literature on simulation support systems, related analytical, and stochastic inventory models, and optimization methods in simulation is introduced. Main issues in this chapter are: (a) current trend and examples in computer-aided simulation modeling systems; (b) object-oriented programming as a new approach to software development; (c) shortcomings of analytical models for inventory theory, and the need for simulation as alternative tool; and (d) important optimization methods in simulation, comparison of RSM with other methods, and future directions in RSM.

In Chapter III, detailed outline of the inventory systems that can be modeled using the proposed system is made regarding modeling flexibility, modeling options, and problem parameters.

In Chapter IV, the integrated model development environment is fully explained. The details of the implemented class hierarchies, the file organization, and the code generation process in the computer program are discussed.

In Chapter V, a modeling session example demonstrating most of the features of the developed system is introduced. A brief discussion of the example model results is also provided.

In Chapter VI, first, the guidelines on how to use RSM in optimization of inventory simulation models are presented. Then, an optimization example for a continuous review inventory model is introduced. At the end of this chapter, performance assessment of RSM is provided.

In Chapter VII, further research suggestions in the subject are made, specifically integrating the system with a statistical package, and addition of more application domains.

Finally, Chapter VIII introduces the conclusions and the findings of this study.

CHAPTER II

REVIEW OF THE RELATED LITERATURE

Simulation Support Systems

Increasing demand to simulation as alternative decision-making tool has encouraged many researchers to improve simulation modeling methodology in its every aspects. Simulation projects usually take lengthy time, and require the collaboration of experts from various fields (Shannon et al., 1985). Model building and program coding activities take the biggest share within total elapsed time during a simulation project (Balci & Nance, 1987). As Law and Haider (1989) estimated, program coding alone could be about 30-40% of the total time. General-purpose simulation languages took the first steps toward making simulation affordable, and efficient tool to more people. Fifth-generation simulation languages have taken the matter one step further by adding design, and model building activities to the ongoing automation trend (Shannon et al., 1985).

Besides the time factor in simulation, another prominent reality is that coming up with the right model, and code, and the correct interpretation of the simulation output could be at risk even after days of work, if there are any misuse of concepts, and logical errors in the program code. Flitman and Hurrion (1987) suggested combining the knowledge and expertise of people from various backgrounds within an expert simulation system framework to ease the problem. In this regard, intelligently designed program generators can shorten the coding time drastically to minimum levels (Crookes, 1987; Paul, 1991).

Due to the dynamic nature of the current manufacturing systems (e.g., FMS), the modeler is subject to revise the model occasionally. Rapid retrieval, changing, and reusing of previously developed models are essential to increase the responsiveness of simulation models in this sense (O'Keefe, 1986). A rapid model prototyping environment called (SDME) by Balci and Nance (1987) was introduced for UNIX-based Sun workstations. Multitasking and powerful window management of the Sun workstations improved the efficiency of the modeler in terms of modeling time. This system used both the top-down model definition and bottom-up model specification approaches simultaneously in order to get as much as information from the user during data collection. The user was directed to break down his model into logical and hierarchical objects, and attach some attributes to them at model building stage. This approach is an efficient way of constructing rapid prototypes when there is no exact specification about the problem in the beginning. Later, as more data is collected, the user could build more complex and specific models on the prototypes.

As far as domain-dependency is concerned, program generators can be either domain-dependent or domain-independent. Domain-independent generators use activity cycles, casual diagrams, or intermediate model description languages as model definition mechanism (Ulgen & Thomasma, 1989). In Activity Cycle Diagram (ACD) approach the user is required to specify the life cycle of each entity in the system such as customers, and parts starting from source to sink including decision points. AUTOSIM used ACD to collect data, and generate code in Pascal using the library containing Pascal routines for simulation (Paul & Chew, 1987). In ACD approach, some logical errors may occur if there exists misidentifying or missing some simulation entities which are not physical (e.g., control entities). On the other hand, it gives more flexibility to the user as compared to domain-dependent program generators.

A domain-independent simulation framework called SPIF using a natural language interface to interact with the user was introduced for discrete-event simulation by Doukidis (1987). English-like model description languages are commonly designed to create an intermediate file to be converted either to a high-level programming language, or to a well-known simulation language subsequently. These languages can ease writing code, but the user may be still required to learn some rules to build models to a lesser degree. In this study, a model description language was also utilized to store model-input data in such a form that can be interpretable to the program generator.

Another model generator (PASSIM) using a description language accommodated three different modeling frameworks in terms of the level of experience, and knowledge of the user (Shearn, 1990). Advanced users were able to add their own Pascal code to be linked with the main code. Pascal was also preferred language in another program generator for queuing models due to its ability to allocate dynamic memory to hold temporary entities during run time (Raczynski, 1990). The generator facilitated a block-diagram editor for model entry, and translated the blocks into corresponding PASSION code (Pascal-based simulation libraries).

Flexible manufacturing systems have been a commonplace problem domain to some program generators reported in the literature due to its significance in manufacturing area, and complex modeling nature. An intelligent program generator for SIMAN was developed in PROLOG by Haddock and O'Keefe (1990). Arrival patterns, description of machine centers, part types, batch size, and part sequences were some of the system parameters introduced to the user. Statistical tests, and confidence intervals could be done automatically by this system in post-simulation analysis. Another generator for FMS generating code in SLAM II provided the similar

modeling parameters along with a menu-driven data collection facility (Co & Chen, 1988). Both generators are data-driven which is commonly used approach in domain-dependent program generators.

The profound effect of object-oriented paradigm (OOP) in software development productivity has proved itself extensively in simulation support systems. Physical objects (e.g. machines, resources, entities), and conceptual objects (e.g. control, decision objects) can be modeled within hierarchy of classes in OOP languages. Ability to reuse existing classes, and derive new classes from earlier ones can boost modeling efficiency in great deal. Also, OOP concepts are quite helpful in simplifying the complexity of systems in more natural way resulting in less confusion.

OOP has been replacing procedural programming as the dominant software paradigm in designing simulation modeling systems today. Simulation languages ModSim (Herring, 1990), SmarterSim (Ulgen et al., 1989), model development environments DEVS-Scheme (Zeigler, 1987), SIMBIOS (Guasch & Luker, 1990) were all object-oriented. A fundamental feature of these systems is to support modular, and hierarchical modeling approach that is to build a complete simulation model from earlier hierarchical submodels rather than starting from scratch. In this study OOP was also the programming paradigm to design the integrated simulation system.

It is obvious that more expert systems to enhance simulation will be available to wider range of users in the future. Paul (1991) predicted three major trends in research on simulation modeling environments: (1) ease of use and flexibility, (2) rapid model formulation, and (3) inclusion of expert statistical systems. Significant improvements have been achieved in first two items so far. However, most of the modeling environments fail to provide assistance for statistical design and validation of

simulation models. Automation of optimization process in simulation has not taken enough attention in existing systems. Integration of analytical tools with advanced simulation systems can improve the optimization process in simulation by narrowing down the search region, and validation of results (Sabuncuoglu & Hommertzheim, 1989).

Related Inventory Models

Problem of determining optimum policy for a given inventory problem type has been investigated extensively in operations research literature. Classical inventory models including the economic order quantity (EOQ) model make some assumptions (e.g., constant demand, no lead time, and so on) that may not fit to real-life inventory problems (Datta & Pal, 1990).

Park (1989) introduced a constant renege rate ρ, that a customer waiting in the line do not wait longer than a duration which is also exponential distribution function with parameter ρ. He assumed that lead time was known, and constant, and more importantly only one customer order can wait at the maximum. An iterative process that converged with probability of one was performed after an initial estimate for average demand per cycle to find an optimum solution. Probabilistic renege rate ρ was appeared to be sensitive to the average inventory cost.

Analytical solution to single supplier, multiple-item inventory model with constant demand rates was introduced by Kumar and Arora (1990). All items were subject to a common inter-order time, which was decision variable, constant demand rate, and lead time. This model is applicable to small stores having single supplier for certain items. They indicated that introducing probabilistic demand to the system increased the complexity drastically. Their suggestion was to use mean demand rates

for the constant demand rate in the model, and keep higher safety-stock levels to absorb the randomness in demand.

Datta and Pal (1990) used inventory-level-dependent demand rate for single-item inventory model assuming that demand may increase or decrease depending upon the inventory level on hand. Proposed demand rate $R(i)$ as a function of inventory level i was

$$\begin{aligned} R(i) &= \alpha i^{\beta}, & i \geq S_0 \\ &= D, & 0 \leq i \leq S_0, \end{aligned}$$

where $\alpha > 0$ and $0 < \beta < 1$ are scale, and shape parameters, S_0 is the inventory level after which demand rate is assumed to be constant (i.e., D). Their two prominent assumption were zero lead time, and no shortage.

Optimization in Simulation

Optimization of simulation output is much more complex, and time consuming than using analytical models, and it can be regarded as ultimate goal for simulation studies. Since there exists random error in response variables, the term optimization should be referred as optimum-seeking procedure for simulation studies (Safizadeh, 1990). In general, k controllable factors $X_1, X_2, \ldots, X_k$, and m response variables $y_1, y_2, \ldots, y_m$ produced by the simulation model are considered, and objective is to find combination of controllable factors such that it maximizes or minimizes the response variables. In this study only single response variable optimization methods were investigated.

Major characteristic of optimization in simulation is the lack of knowledge about the true response function that could be investigated directly. Basically it is assumed that the response function can be estimated within a region through regression, or meta-models based on simulation output under different combinations of controllable factors. Many techniques to search the optimum from the simple random search to RSM have been introduced in the literature. However, there are some trade-offs in selecting the appropriate method for a given problem type. When simulation model execution time is very large, selection of optimization method becomes more important due to the limited number of runs. As pointed out by Smith (1973a), the performance of the selected method depends upon the following factors: (a) number of controllable factors (i.e., independent variables), (b) number of available computer runs, (c) existence of local optima, (d) size of random error (i.e., statistical variation in response), (e) distance of starting point from the true optimum, and (f) significance of interaction between controllable factors.

In the following sections, the most used optimization methods for simulation are discussed. RSM was particularly investigated in more detail as it was employed in this study.

Response Surface Methodology (RSM)

RSM was first developed to find optimum operating conditions in the chemical industry in the 1950s (Myers, Khuri, & Carter, 1989). Box and Wilson (1951) first developed the principles of RSM. RSM is based on an assumption that a response variable y as a function of k controllable variables $X_1, X_2, \ldots, X_k$ can be approximated within some pre-defined region by a polynomial function. As a matter of fact, this as-

sumption is based on the Taylor's series approximation of a function. The two prominent polynomial models are the first-order model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k \qquad (1)$$

and the second-order model

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i X_i + \sum_{i=1}^{k} \beta_{ii} X_i^2 + \sum_{i=1}^{k} \sum_{i<j}^{k} \beta_{ij} X_i X_j \quad (2)$$

Generally the least-squares method is employed to estimate the true coefficients $\beta_1, \beta_2, \ldots, \beta_k$ in the model without bias, and with minimum variance in error.

Issues in Experimental Design

Selection of experimental design parameters (i.e., number of runs, number of levels, number of replications) affects the quality of estimation of the response surface in terms of statistical significance, and reliability. Orthogonal designs can reduce the number of runs needed. They also prevent confounding effects of individual factors resulting in lesser error variance. This is because each column in an orthogonal design matrix is independent from each other due to the fact that the summation of cross-product of any two columns is always zero. When two levels exist for each controllable factor, the required number of runs for full factorial orthogonal design would be 2^k. In some cases where k is very large, a 2^{k-1} fractional factorial experiment could be satisfactory to make statistically significant estimates of regression coefficients (Safizadeh, 1990).

In actual simulation studies number of controllable factors could be very large resulting in need for too many runs. However, in general only a small portion of controllable factors have a significant effect on the response. Considering this fact, group

screening methods can be employed to detect the significant factors by putting factors into groups, and treating these groups as single factor. Cochran and Chang (1990) implemented a two-stage group screening method to find optimal settings for a flight simulation. Only two of the six original factors in the beginning were selected to be used in performing the steepest-ascent method as a result of the two-stage group screening. As a potential drawback, in group screening some significant factors may be excluded when factor effects within a group are neutralized by each other. Therefore, forming groups requires some pre-knowledge about factors (Mauro & Smith, 1982).

The success of RSM depends on the variance reduction method used, and the quality of estimation of the gradient (i.e., β's). The presence of error variance in response makes optimization difficult for simulation (Safizadeh, 1990). Making longer runs, using expected value of response, and steady state values free the optimization process from variance and bias to a certain degree. Several methods, and guidelines called variance reduction techniques have been introduced to address this problem. Most widely used method is to have common pseudo-random numbers for each cell in the design (Law & Kelton, 1991, p. 613). Other methods include antithetic variates, control variates, indirect estimation, and so on. Use of common pseudo-random numbers is the easiest one as it does not require sophisticated statistical covariance calculations, and adjustments.

Second-order model design is constructed at the final stage of optimization in simulation to find optimum settings. Rotatable designs for a second-order model can keep the error variance of predicted value $\hat{y}$ at some point x that are equal distant from the center of the design (Montgomery, 1984, p. 462). Furthermore, rotatability of a design implies that the error variance is not a function of direction. Orthogonal

design for the first-order model, and central composite design (CCD) for the second-order model satisfy this important property (Myers et al., 1989; Safizadeh, 1990). However, the distance from center of the design to other points is an important parameter to make CCD rotatable (Biles & Ozmen, 1987).

Steepest ascent method RSM is used in conjunction with the steepest-ascent method to find the optimum settings of controllable factors. Procedure starts with identifying the search region, in other words determining the upper, and lower limits for each factor. At the outset fairly small region should be selected so that parameters for gradient search can be estimated more accurately (Box & Draper, 1987). In summary the following steps are carried out in the method of steepest ascent (Myers, 1976):

1. Fit a first-order regression model (see Equation 1) in some restricted region of the controllable factors $X_1, X_2, \ldots, X_k$.

2. Locate a path of steepest ascent based on parameter estimates in Step 1.

3. Perform simulation runs along the path until no additional improvement (i.e., increase, or decrease) in response is evident.

4. Steps 1, 2, and 3 are repeated within new regions until first-order model is inadequate (i.e., F-test fails). If model is inadequate, proceed Step 5.

5. If interaction and second-order terms in model become significant (i.e., a curvature in response is evident), then use central composite design to build a second-order model (see Equation 2).

6. Perform canonical ridge analysis to find the optimum levels of factors, or use partial derivations to find the optimum if the second-order model function is convex.

In Step 1 if there are k factors, a simple 2^k factorial design can be used to estimate the coefficients of the following $(k+1)$ dimensional hyperplane.

$$y_i = b_0 + \sum_{j=1}^{n} b_j X_{ij} + e_i \qquad (3)$$

The experimenter wishes to advance from the center of the design r units such that maximum increase in response is obtained. If factors are coded, with design center being (0 , 0, . . . , 0), the experimenter wishes to find the values of $(x_1, x_2, \ldots, x_k)$ which maximize

$$b_0 + \sum_{i=1}^{k} x_i$$

with subject to the constraint

$$\sum_{i=1}^{k} x_i^2 = r^2$$

Using the Lagrange multipliers for the restricted maximization, we define the function

$$Q(x_1, x_2, \ldots, x_k) = b_0 + \sum_{i=1}^{k} b_i x_i - \lambda\left(\sum_{x_i=1}^{k} x_i^2 - r^2\right) \qquad (5)$$

where λ is the Langrange multiplier.

Equating the partial derivations of (5) to zero

$$\frac{\partial Q(x_1, x_2, \ldots x_k)}{\partial x_i} = 0 \qquad (j = 1,2,\ldots k)$$

and

$$\frac{\partial Q(x_1, x_2, \ldots, x_k)}{\partial \lambda} = 0$$

we obtain the following

$$x_i = \frac{b_i}{2\lambda} \qquad (i = 1, 2, \ldots, k). \qquad (6)$$

$$\lambda = \frac{b_i}{2x_i} \qquad (7)$$

Rather than selecting the value of λ corresponding to a particular r, a reasonable increment in one of the factors is selected by the experimenter based on his experience to calculate increment values for other factors using Equation 6. New experiments are performed with these increments until no improvement in response is evident.

As mentioned, first-order models are likely to be inadequate when the interaction, and higher polynomial degree terms (e.g., quadratic effects) become significant eventually. Then, second-order model (see Equation 2) is built for the final step. The following two methods can be followed to find the optimum settings based on the second-order model function:

1. Partial Derivations: Equating the partial derivations of the response function, we get the following equations to find stationary points.

$$\frac{\partial \hat{y}}{\partial x_1} = \frac{\partial \hat{y}}{\partial x_2} = \cdots = \frac{\partial \hat{y}}{\partial x_k} = 0$$

At this point, a stationary point can represent (a) a point of maximum response, (b) a point of minimum response, or (c) a saddle point (i.e., inflection point) (Montgomery, 1984, p. 453). A saddle point is a stationary point which does not respond to a local optimum (minimum, or maximum).

2. Ridge Analysis: It is a canonical search method that is basically cutting the response surface by equally distanced circles, and recording the change in response.

In order to decide whether a stationary point is a local optimum, or a saddle point, we should transform the second-order model to the canonical form shown in Equation (8) in the new coordinate system, with the origin at the stationary point, and the axes of the new coordinate system being parallel to those of original fitted regression function.

$$\hat{y} = \hat{y}_0 + \mu_1 w_1^2 + \mu_2 w_2^2 + \cdots + \mu_k w_k^2 \qquad (8)$$

where the $\{w_i\}$ are the transformed independent variables and the $\{\mu_i\}$ are constants.

Based on the sign, and magnitude of the $\{\mu_i\}$, the following conclusions are made in terms of the nature of the response function (Montgomery, 1984, p. 455).

1. If all the $\{\mu_i\}$ are positive, then the stationary point is a minimum.
2. If all the $\{\mu_i\}$ are negative, then the stationary point is a maximum.
3. If the $\{\mu_i\}$ have different signs, the the stationary point is a saddle point.

Other Optimization Methods in Simulation

Random Search

In this method, values of controllable factors are chosen randomly within a search region, and then simulation is run at these points. The best value among them is claimed to be optimum (Farrell, 1977; Smith, 1973b). There is no guarantee for finding the optimum in this method. Some variations also exist such as intensifying the search within a certain region after some pre-runs. It is not a structured method, but easy to use. As number of runs increases better estimate for optimum can be made.

Coordinate Search

Only one variable at a time is changed starting with an initial point until no more improvement in response is evident. The search is continued with another factor until available number of runs are exhausted. If interactions between factors are significant this methods may fail as the search path is always parallel to the axis (Farrell, 1977; Smith, 1973b).

Pattern Search

The search starts with an initial point (b_0), and another point (b_1) to find a pattern ($b_1 - b_0$) for the search. Simulation is run along the pattern as long as there is improvement in response. Another pattern is determined for further runs.

Perturbation Analysis (PA)

Main idea in PA is to estimate derivatives of factors with respect to the response in single run. Infinitesimal PA is used for continuous variables while finite PA is for discrete variables such as buffer size (Suri & Leung, 1989). Infinitesimal PA assumes that perturbations in the control variables do not change the order of events in simulation. In most simulation models with state-dependent events, and multiple types of customers, it is impossible to satisfy this condition (Meketon, 1987; Safizadeh, 1990). Therefore, PA applications are limited to only queuing networks based on up to date literature (Wilson, 1987), for example M/M/1 queue (Suri & Leung, 1989), M/G/1 queue (Suri & Zazanis, 1988), and tandem queue networks (Ho & Cao, 1983).

Likelihood-Ratio Methods

They are limited to applications with Markov chains, and Poisson process. Currently, they have been applied to regenerative simulation analysis (Wilson, 1987).

Frequency Domain Methods

These methods require careful indexing of simulation generated observations together with sinusoidal variation of selected factors according to time index. Such variation could be difficult to arrange especially for discrete input variables (Wilson, 1987).

Comparison of the Methods

Smith (1973a) compared random search, coordinate search, and response surface methodology along with steepest ascent method based on some performance measures for pre-known true response function. His findings were:

1. As the number of runs was increased, RSM was better than the others.
2. When the number of runs was small, the random search was the best.
3. The steepest ascent methods required fewer runs to estimate the search direction.
4. Coordinate search performed poorly.

Mathematical and statistical foundations of RSM are more clear, and completely developed as compared to the other methods (Wilson, 1987). Safizadeh (1990) concluded that RSM in conjunction with gradient search methods was the best optimization method for simulation due to its practical dealings with statistical variance, and optimization. Although perturbation analysis gives estimates of derivatives

of response function in single run, and makes single-run optimization possible, its application to complex simulation models is limited. PA is still in its fancy, more theoretical validity research is needed to strength its foundations of its applications in wider range of problems (Safizadeh, 1990).

The success of random search, coordinate search methods depends upon characteristics of the response surface. They cannot guarantee the optimum solution even though there is a single optima.

CHAPTER III

PROBLEM DOMAIN: INVENTORY CONTROL SYSTEMS

Characteristics of the Investigated Inventory Problems

Inventory Control Mechanisms

As modeling flexibility, three types of control mechanisms can be attributed to a particular product in the model. If inventory model is a multi-product model, different types of control mechanisms can be assigned to products. The following is the description of these three control mechanisms:

1. Continuous review (S, s): Inventory level is reviewed after each transaction (i.e., change in the level) to place a new order when the inventory level is less than or equal to the reorder point. Order quantity Q for a new order is calculated from

$$Q = S - I + D_L \qquad (8)$$

where I, D_L are inventory level on hand, and expected demand during lead time respectively. In general, D_L

$$D_L = a.L, \qquad a = \frac{1}{t_a} d\ , \qquad (9)$$

where t_a is average time between customers arrivals in unit of time, d is average demand size for a customer, a average demand rate, and L average lead time. Inventory level I may take negative values to represent the amount of backordered units. The following assumptions are made regarding this control mechanism: (a) order quantity arrives as whole at once after a certain deterministic, or probabilistic lead time, and (b) another order cannot be placed before the earlier one arrives.

In Figure 1 the diagram of the inventory level as a function of time is depicted for continuous review model with backlog permitted, where t_c is cycle length.

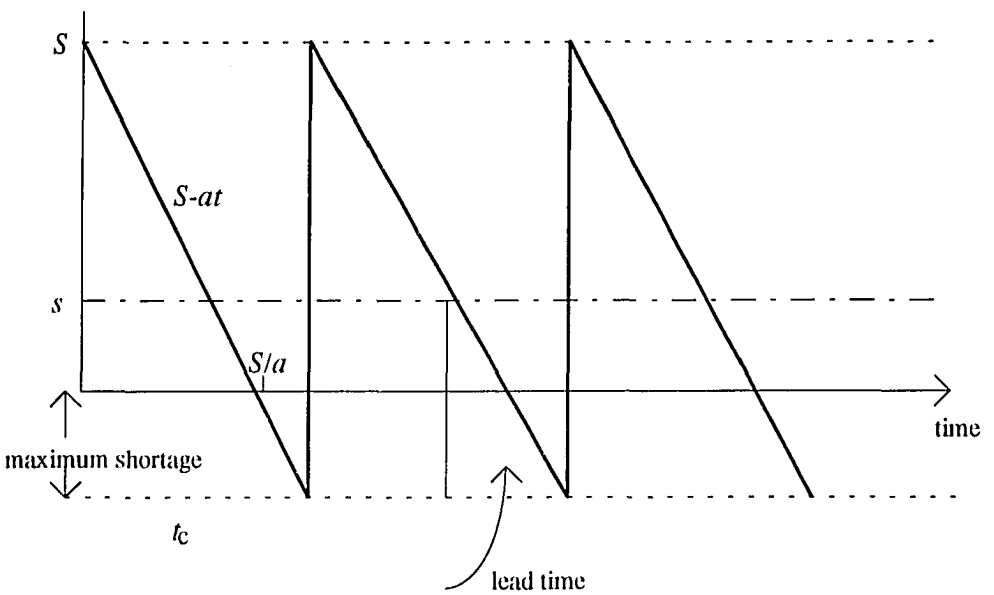

Figure 1. Diagram of the Continuous Review Model With Backlog Permitted.

2. Periodic review: It has a similar decision mechanism to the one in continuous review with one exception that inventory level is reviewed after each t time period. Equations 8, and 9 are still valid for periodic review. This review system can be applicable to products which are not very valuable, and not critical.

3. Production model with continuous review: When the inventory level is less than or equal to reorder point, product is produced, or supplied by the vendor at a constant rate r until inventory level reaches to the reorder level (S). A deterministic, or probabilistic lead time may exist prior to the start of production.

As seen in Figure 2 inventory level increases steadily at a constant rate along with a probabilistic demand. One should note that there is an obvious condition that production rate r has to be greater than demand rate a, otherwise an infinite customer queue occurs in the system.

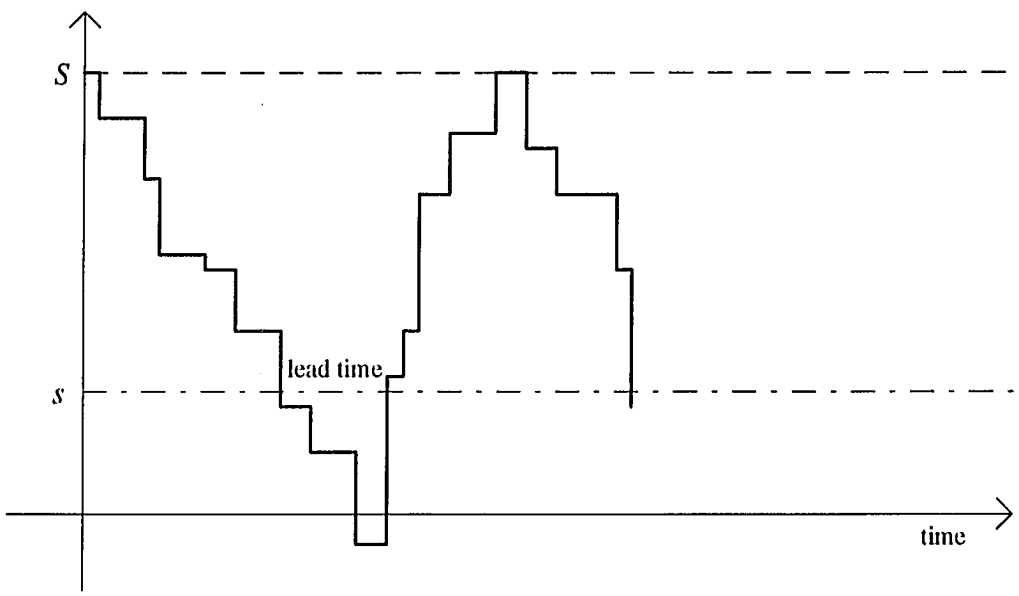

Figure 2. Diagram of the Production Inventory Model.

Cost Elements

Five types of inventory costs associated with each product can be defined in a given inventory problem. All unit costs are assumed to be constant, and independent from quantity. These cost elements are:

1. Holding cost (h): The cost of maintaining inventory which covers the costs of capital tied up, insurance, warehouse space, and so on. It is measured in $ per unit per unit of time.

2. Backlog cost (p): The cost of backordering customer due to not having inventory on hand per unit per unit of time.

3. Setup (ordering) cost (K): the cost of placing an order, or starting new production per setup.

4. Cancellation cost (Cc): This cost occurs when a customer cancels the order based on the pre-defined backlog policy due to the shortage on hand. Profit losses are covered in this cost element, and measured as $ per unit.

5. Review cost (C_r): A unit cost per review may be included in the model to consider costs that occur during review process of inventory level such as labor, equipment cost to review, and so on.

Customer Arrivals, Demand Pattern, and Lead Time

Customer Arrivals

Time between customer arrivals can be either deterministic (i.e., constant), or based on a user-selected random distribution function such as exponential, uniform, and normal distribution. Furthermore, each product in the model can have its own arrival pattern. Basic assumptions made for customer arrivals were: (a) it is

independent from inventory level, and (b) it is independent from the queue length of backlogged customer orders.

Demand Size

Deterministic, or probabilistic demand size for a customer arriving to the system exists for each product in the model. It is assumed that demand size is independent from inventory level, and customer queue length. User-defined empirical distribution can be alternative choice when any known standard distribution does not fit the past data for demand size.

Figure 3 shows the graph of a discrete-distribution having k possibilities. Note that the sum of probabilities must be one (i.e., $p_1 + p_2 + \ldots + p_k = 1$)

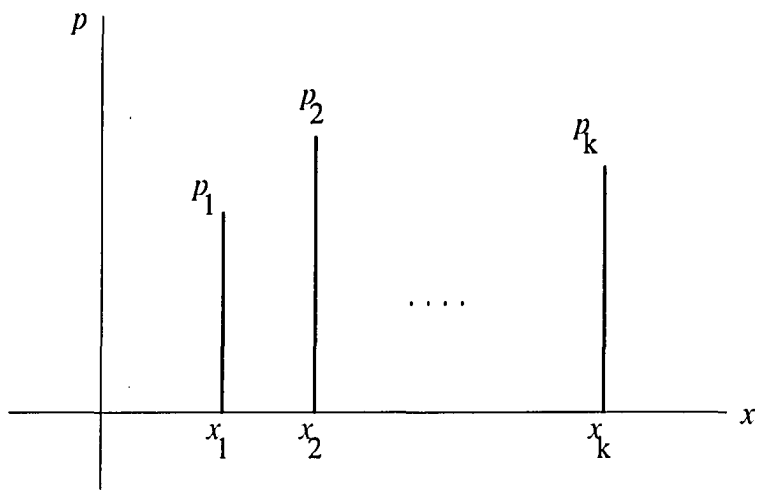

Figure 3. User-Defined Empirical Distribution Function.

Lead Time

Deterministic, or probabilistic lead times prior to order arrival, and the start of production are enabled.

Price Breaks

Selling price p_s to customer, and purchase price c (or, manufacturing cost) associated with a particular product can have price breaks, or single constant unit price. Up to 20 price breaks can be specified for any of the products in the model. The price breaks should be entered in the following format:

$$0 \le Q < q_1$$
$$q_1 \le Q < q_2$$
$$\cdots$$
$$q_{k-1} \le Q$$

where q_1,and q_2 are the lower, and the upper limits of the second price break, and k is the number of price breaks.

Backlog Policy

What action should be taken when there is a shortage in inventory on hand is outlined through a user-defined backlog policy. Three options are available to the user in this respect:

1. Constant probability: Customer leaves the system without waiting at a constant probability (p_r) when backorder occurs. Two extreme cases can be modeled using this option:

(a) Customer leaves the system when backlogged ($p_r = 1$). This may be applicable to products which are readily available from other sources. Figure 4 shows that case when p_r is 1, that is there is no backlog.

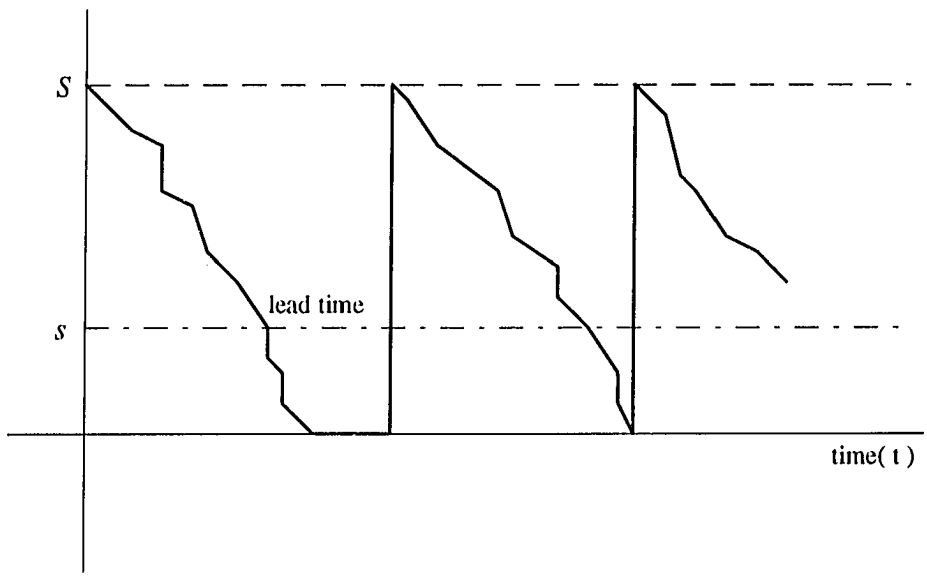

Figure 4. Diagram of the Continuous Review Model With No Backlog (p_r=1).

(b) Customer does not leave the system regardless of length of customer queue ($p_r = 0$). This may occur when there is only one producer, or retailer in the market (i.e., monopoly), and substitution of the same product by another is not possible.

2. Number of units backordered: Customer leaves the system without waiting if total number of units backordered at that moment exceeds the pre-defined limit. The two extreme cases explained in 1(a), and (b) occur if the limit is set to zero, and infinity respectively.

3. Customer leaves the system at a conditional probability for a given expected waiting in the system before his demand is satisfied.

Let A be the event that customer leaves the system. We define a conditional probability function of A for a given expected waiting time t_w, that is:

$$P(A/t_w = t) \begin{cases} 0 & 0 \le t < t_1 \\ \dfrac{t - t_1}{t_2 - t_1} & t_1 \le t < t_2 \\ 1 & t_2 \le t < \infty \end{cases} \qquad (10)$$

where t_1 and t_2 are lower limit and upper limits for expected waiting time respectively. It is assumed that probability of canceling order by the customer is a linear function of expected waiting time between the lower, and upper limits. Figure 5 shows this conditional probability function. As seen in the graph the customer does not cancel the order when t_w is up to t_1 while the customer cancels his order when t_w is greater than upper limit t_2 with the probability of one.

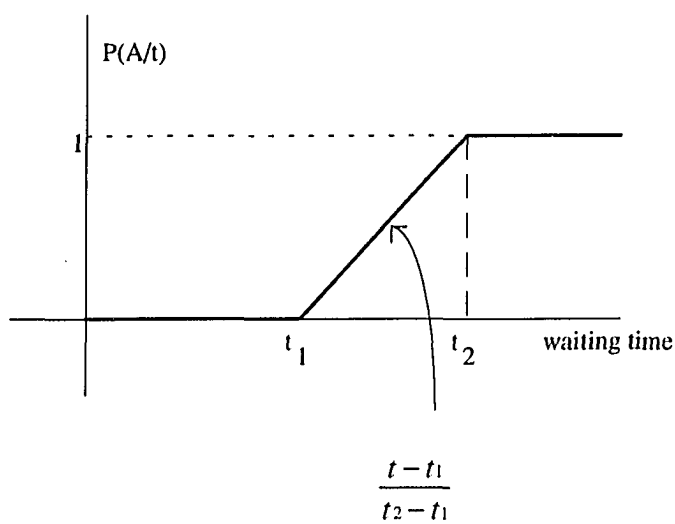

Figure 5. Conditional Probability Function of Customer Reneging for a Given Expected Waiting Time (t_w).

Substitution Mechanism

This mechanism enables unsatisfied demand to be substituted by other products if possible. Two prominent data for substitution are probability matrix, and amount multiplier matrix. Equations 9, and 10 are the general form of these two matrices for a problem with I products respectively. p_{ij} is the probability of substituting ith product with jth product. q_{ij} represents the multiplication factor if ith product and jth product cannot be substituted one to one in terms of quantity. Value of one is used to indicate one to one substitution.

$$\begin{bmatrix} 0 & p_{12} & p_{13} & \dots & p_{1I} \\ p_{21} & 0 & p & \dots & p_{2I} \\ p_{31} & p_{32} & 0 & \dots & p_{3I} \\ \dots & \dots & \dots & \dots & \dots \\ p_{I1} & p_{I2} & p_{I3} & \dots & 0 \end{bmatrix} \qquad (11)$$

$$\begin{bmatrix} 0 & q_{12} & q_{13} & \dots & q_{1I} \\ q_{21} & 0 & q_{23} & \dots & q_{2I} \\ q_{31} & q_{32} & 0 & \dots & q_{3I} \\ \dots & \dots & \dots & \dots & \dots \\ q_{I1} & q_{I2} & q_{I3} & \dots & 0 \end{bmatrix} \qquad (12)$$

The detailed flow chart of the decision mechanism is depicted in Figure 6. It is assumed that more than one item can be candidate for substitution for a particular product. Therefore the sum of probabilities for a given product (i.e., the sum of corresponding row in probability matrix) does not have to be one.

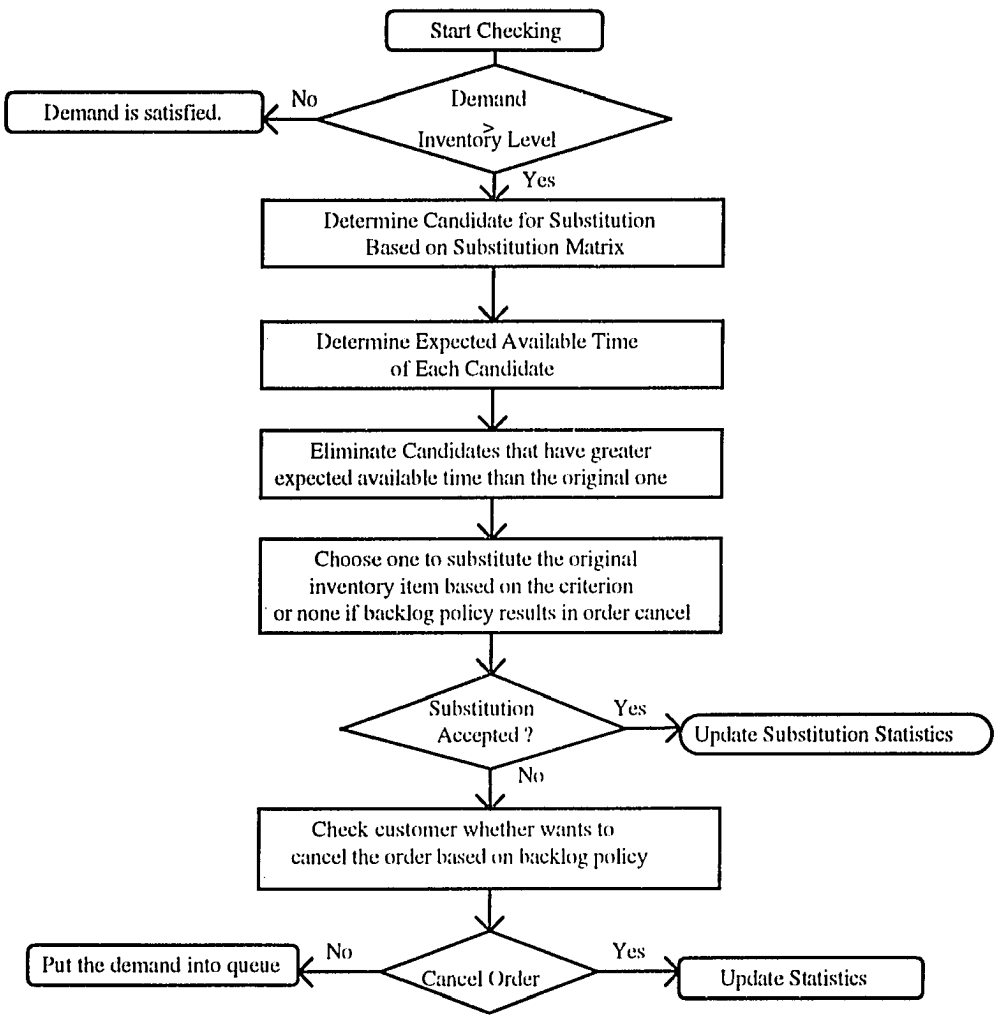

Figure 6. Flow Chart of the Substitution Decision Mechanism.

Selection of the product to substitute is state-dependent meaning that it is based on either on its expected available time at the current state of the simulation, or total price. Any of these two selection criteria can be set by the user.

Performance Measures

The simulation programs generated for various types of inventory problems are designed to provide some system performance measures at the end of each simulation run using the data collection facilities of SLAM II. Along with mean value of each performance measure, standard deviation, maximum, and minimum value, number of observations are also supplied if appropriate. Multiple-product inventory models provide both individual statistics for each product, and overall system performance statistics. The following measures are calculated by the simulation: (a) number of setups, i.e., total number of setups (or, orders) during simulation run length; (b) average holding level per unit of time; (c) average backlog level as time weighted average of number of units backlogged; (d) number of customer demands, and demand rate per unit of time; (e) number of satisfied demands, total amount of satisfied within a simulation run; (f) number of lost demands, and total amount of lost demand within a simulation run; (g) average holding cost per unit of time; (h) average backlog cost per unit of time; (i) average setup cost per unit of time; (j) average lost demand cost per unit of time; (k) average purchase (or, production) cost per unit of time; (l) average review cost per unit of time; (m) average sales revenue per unit of time; (n) average inventory cost per unit of time which is the sum of holding, backlog, setup, lost demand, and review costs; (o) average profit per unit of time; (p) average safety stock level; (q) average cycle length, and (r) substitution statistics, i.e., total number of demands, and amount of units that are used to substitute *i*th product for the place of

*j*th product. These statistics show the significance of the interaction between different product inventory levels caused by the substitution mechanism used in this study.

Some of the performance measures cited here such as average inventory cost, and average profit can be considered as an overall system response variable, and can be optimized to find the optimum inventory policy. However, for most of the measures explained here, there is no known mathematical formulation, or function for actual inventory problems having probabilistic patterns, and decision mechanisms. In this regard RSM can give better insight into the relationships between a certain measure and its corresponding influential factors using regression models. RSM was used in Chapter VII to find optimum policy for continuous review case. Furthermore, the same methodology can be applied to other types of inventory problems with the same principles.

CHAPTER IV

ARCHITECTURE OF THE INTEGRATED SIMULATION MODEL DEVELOPMENT ENVIRONMENT

Computer System Specifications

Operating System

The selection of the operating system in which the computer system of this study would be operating was a crucial decision. The Microsoft Windows operating system was chosen due to its following powerful features (Microsoft Press, 1990):

1. It is easier to integrate other related applications with our program such as statistical software, and more importantly the SLAM II software itself.

2. It has advanced user interface elements standard for all programs such as menus, dialog boxes, buttons, and multiple windows.

3. Dynamic Link Libraries (DLL) allows to link functions during run-time rather than compile time resulting in smaller program code size.

4. The user can work on more than one simulation project by running more than one copy of the developed program simultaneously.

5. Rapid model prototyping is easier by using the multi-tasking and multi-windows facilities.

6. It permits direct access to other programs in the system without terminating working session.

7. Object-oriented programming can take the full advantage of the message-based execution in Windows by transferring system, and user input messages to corresponding objects created in computer program.

Programming Paradigm, and Language

As a new programming paradigm, the object-oriented approach was first proposed in order to overcome the software crisis that took place in early 70s due to the shortcomings of procedural programming approach in developing large-scale software systems. In procedural programming approach, the entire computer program is broken down into logical functions (i.e., subroutines) to cope with the complexity of the system, and to achieve modularity. Implementation of functions usually becomes an immediate issue in software development process. On the contrary the software developer is encouraged to give more attention to the design stage assuring a well-defined architecture required for modular and robust software design when he, or she uses object-oriented programming (Eckel, 1989, p. 15).

As being discussed, object-oriented programming is the most powerful programming paradigm currently available in computer science. Object-oriented features of C++ have indicated significant benefits at the current state of the system. At present Smalltalk and C++ are the two most promising object-oriented languages in computer programming area. The powerful features of the C language were enhanced in C++ by adding the object-oriented extensions by Bjarne Stroustrup in AT&T labs (Jordan, 1990).

In terms of execution speed C++ is faster than Smalltalk which is not as restrictive as C++ in type checking and data binding processes. Dynamic binding decides which object to call during run time while static binding makes decision during

compiling the program. Smalltalk performs dynamic binding which gives greater flexibility to the programmer along with some reduction in execution speed. However, C++ is the primary language for application development in the Microsoft Windows operating system. In this study Borland C++ Version 3.1 compiler was used along with its ObjectWindows class library for user-interface objects, and its container class library to implement some object handling operations (e.g., sorting, queue operations, and so on). The C++ language has been selected due to its following advantages: (a) dynamic memory allocation, (b) extensive class library support, (c) greater execution speed, (c) complete support of OOP concepts such as inheritance, and data abstraction, and (d) portability.

System Architecture

System Components

The integrated development system was built on a system architecture of the modular computer programs, and files to store data, and results. Figure 7 shows the interaction, and data flow between the program modules, and the files. A detailed discussion of each module is as follows:

Project Manager: It is a computer program written in C++ to administer main menu operations by calling other C++ programs when a menu selection is made. It has the following responsibilities: (a) open, and save simulation project files, (b) associate files in a project with the other programs, (c) call program generator, (d) call user-interactive model entry program, (e) call the experimental design program, and (f) close a working session when requested.

Experimental Design: It is a C++ program to form the experimental design matrix based on the input supplied by the user. Program screens given in Appendix F present some idea on the input for experimental design entered by the user during a working session. Prior to executing the SLAM II program, experimental design stage has to be completed so that model execution can be performed based on the factor levels in the design matrix. This program writes experimental design matrix to the file specified in the project file (see Figure 8 for a project file example) after design parameters are completely specified.

Interactive Model Entry Program: It is a C++ program to create the model description file for a given problem based on the input collected from the user during interactive session. It is equipped with program routines to get input and make logical checks on customer arrival patterns, demand patterns, price breaks, backlog policies, etc.

Code Generator: It is a simulation program generator written in C++ for the SLAM II language. It reads, skims the model description file, and finally generates the appropriate SLAM II code based on the rules, and guidelines built in its inference engine.

SLAM II simulation program: It is a program code written in SLAM II by the program generator. It collaborates with the FORTRAN routines during model execution, and reports the simulation output to the corresponding report files (i.e., simulation summary report, and regression input file) specified in the project file.

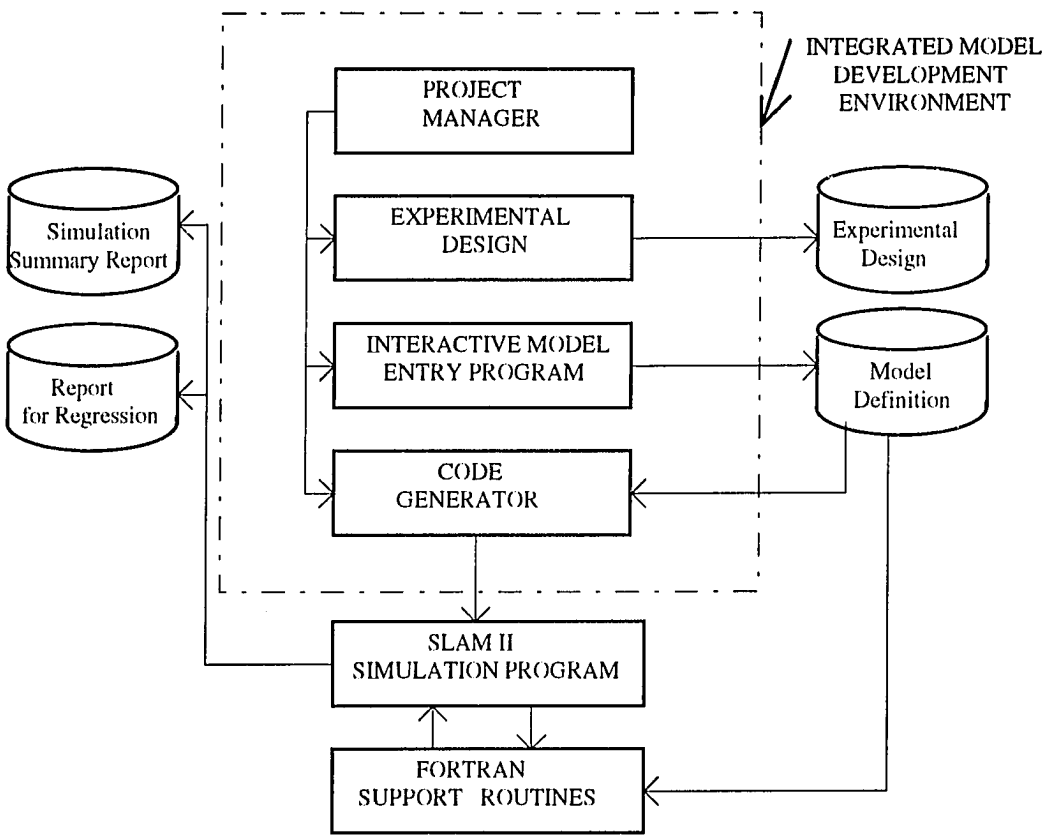

Figure 7. Organization of the Integrated Model Development Environment.

FORTRAN support routines: They are the collection of FORTRAN subprograms to support the generated SLAM II program during model execution. It was essential to design these subprograms as flexible as possible so that they can cooperate with various SLAM II codes without any need for change. SLAM II allows the modeler to write FORTRAN subprograms to be linked with the main SLAM II code when the flexibility of the SLAM II nodes does not satisfy the needs of the modeler. Especially complex mathematical calculations, user-defined functions, and complex discrete-event mechanisms can be modeled in FORTRAN routines. This capability results in greater flexibility for the modeler. The classification of the FORTRAN routines created for this system is shown in Figure 8.

Subroutine INTLC initializes the SLAM II variables prior to each simulation run. The model reading routines are called by this subprogram to read model description blocks. For the first simulation run, this subroutine reads the entire model description file, and transfers the necessary data into corresponding SLAM II variables and arrays. Initialization of the controllable factors (i.e., inventory policy parameters) is carried out by reading the appropriate record from the experimental design file for succeeding runs.

Subroutine OUTPT is called at the end of each run to collect statistics, and write them into eport files. The event subroutines are invoked by using the EVENT node in SLAM II code to update performance measures when a change occurs. The user-defined functions are mainly designed for random variable generation, price calculations, expected waiting time, and other complex calculations. Some standard subprograms supplied by the SLAM II environment for discrete-event simulation are also called when required.

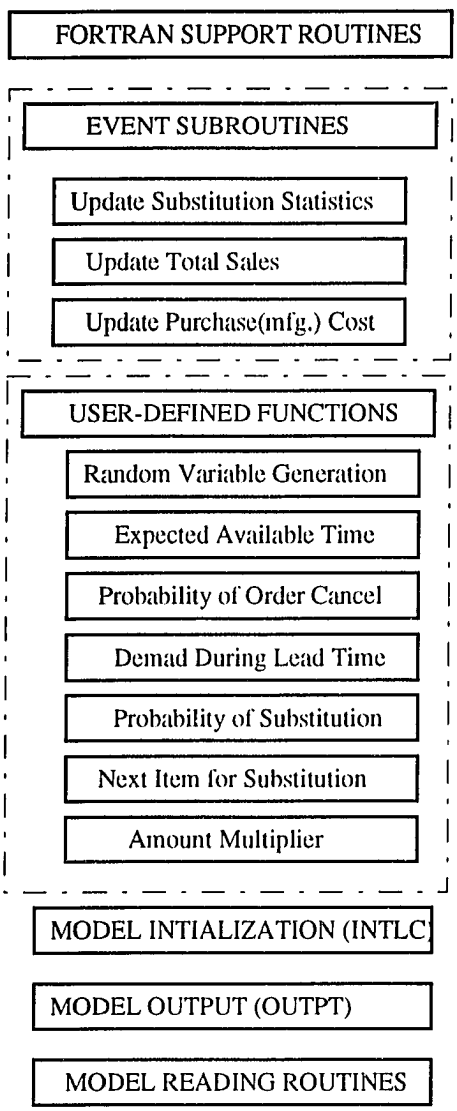

Figure 8. Organization of the FORTRAN Subroutines.

Class Hierarchy

As mentioned earlier, the integrated simulation system is implemented using object-oriented programming approach. Classes form the skeleton of an object-oriented computer program. In simple terms, a class is a functional unit containing functions and data to be processed by those functions aimed for a particular purpose. There is a similarity between classes and subroutines used in procedural programming languages such as FORTRAN and Pascal in terms of their functionality in simplifying the complexity of the program. This is to break down the entire program into more manageable submodules. However, classes have better tools such as inheritance and encapsulation to deal with program complexity .

In this section, all classes used in this study are discussed in detail with respect to their position in the class hierarchy. We suggest the reader to review terminology used in object-oriented programming introduced in Appendix E when needed. Classes are discussed in six main groups based on their functionality in the system.

Classes for the SLAM II Nodes and Control Statements

A SLAM II program code consists of nodes, and control statements subject to some syntax and structural rules. Nodes represent queues, servers, decision points, and entity manipulation functions in a simulation program. On the other hand, control statements are used to initialize variables, and to send declarations to the SLAM II preprocessor.

The taxonomy of SLAM II is completely represented in object-oriented way using a well-defined class hierarchy for the nodes and control statements. Figure 9 shows the class hierarchy of these classes. These classes are solely designed to be

used at program generation stage by creating instances of them. At the top of the hierarchy, the class *SlamStructure*, an abstract class, describes common data and pure virtual functions for the classes related to the SLAM node and the control statements, for instance a unique code to identify one node from another. Two new abstract classes derived from *SlamStructure* are: (a) *SlamNodes*, and (b) *SlamStatements*. Both classes serve as an umbrella to the other classes by declaring common data and member functions. The syntax, and rules of the nodes and control statements of SLAM II are encapsulated in the classes derived from these two base classes (i.e., *SlamNodes*, and *SlamStatements*).

Notice that all the classes related to the SLAM nodes appearing at the bottom of the class hierarchy take the class *Sortable* as base class (e.g., *AwaitNode*). By making these node classes sortable, they can be stored in *PriorityQueue* which is another class designed to sort and hold objects based on a pre-defined priority rule. These rules are implemented within the corresponding classes based on the general guidelines for the SLAM nodes, and application specific rules. As order in which nodes appear in a SLAM II program is important, during program generation process the defined nodes in the priority queue can be extracted in order, and then written to the user-defined SLAM II file. On the other hand, the SLAM II control statement classes (e.g., *Gen*, *Array*, *Limits*) in the hierarchy are not made sortable since where a control statement should appear is almost fixed in SLAM II.

Code Generation Classes

These classes form the architecture of the simulation program generator. *GeneratorBase* is an abstract base class to all the derived classes in this hierarchy (see Figure 10). Notice that the class *Sortable* is again a base class to some classes which

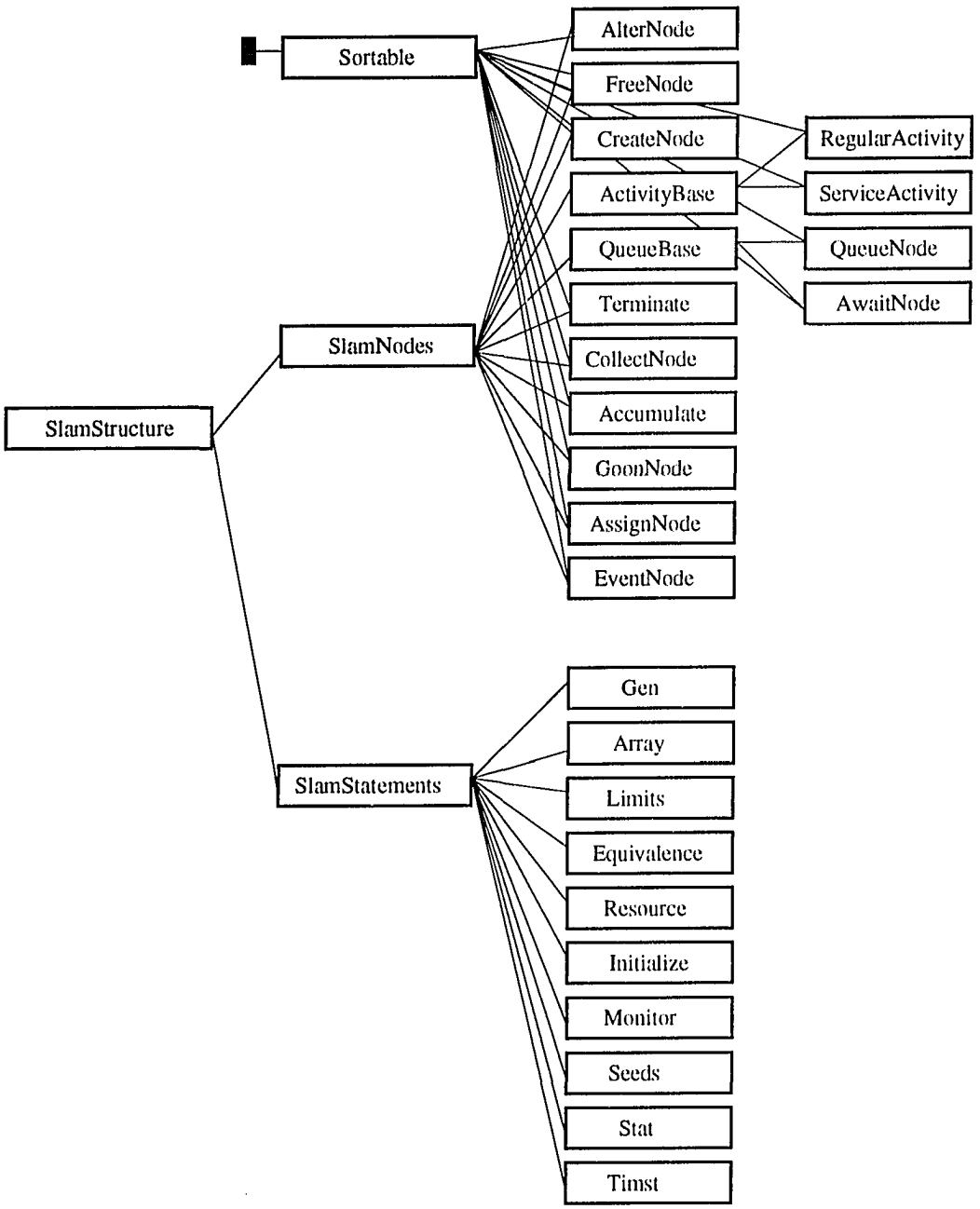

Figure 9. Hierarchy of the SLAM II Classes.

are responsible for deciding appropriate SLAM II control statements, and nodes to carry out a particular task. For example, the class *CancelOrder* is responsible for generating the nodes required to cancel a customer order. In the design process of the generator, first a generic SLAM II program for inventory problems was developed, and, then this generic program was broken down into some tasks such as creation of customer arrivals, and defining variables, and arrays. At the stage of combining these code blocks to form a complete SLAM II code, their order in the final SLAM II code is crucial to generating the correct code. Therefore, objects of these classes are designed to be sortable so that they can be sorted by using priority rules. The following is the brief discussion of each class in this group.

1. LoadModel: It reads model description file, and assigns data to dynamic objects.

2. RunModel: It generates the simulation program in SLAM II using other objects in this hierarchy, *CreateCustomers*, *DefineArrays*, and others. It skims the information in description file, and puts into variables so that other classes can use them during code generation.

3. DefineArrays: It is responsible for selection of variables, attributes, time-persistent variables, and arrays to be used in SLAM II program, and generation of the ARRAY, EQUIVALENCE, TIMST, and STATS statements.

4. Define Resources: Inventory on hand is modeled as resource in the SLAM II program. This class specifies the parameters of resource blocks used in SLAM II.

5. CreateCustomers: It is responsible for specifying nodes for customer arrivals.

6. DoPeriodicReview: It creates a control entity to review inventory level periodically if periodic review mechanism is specified for any item.

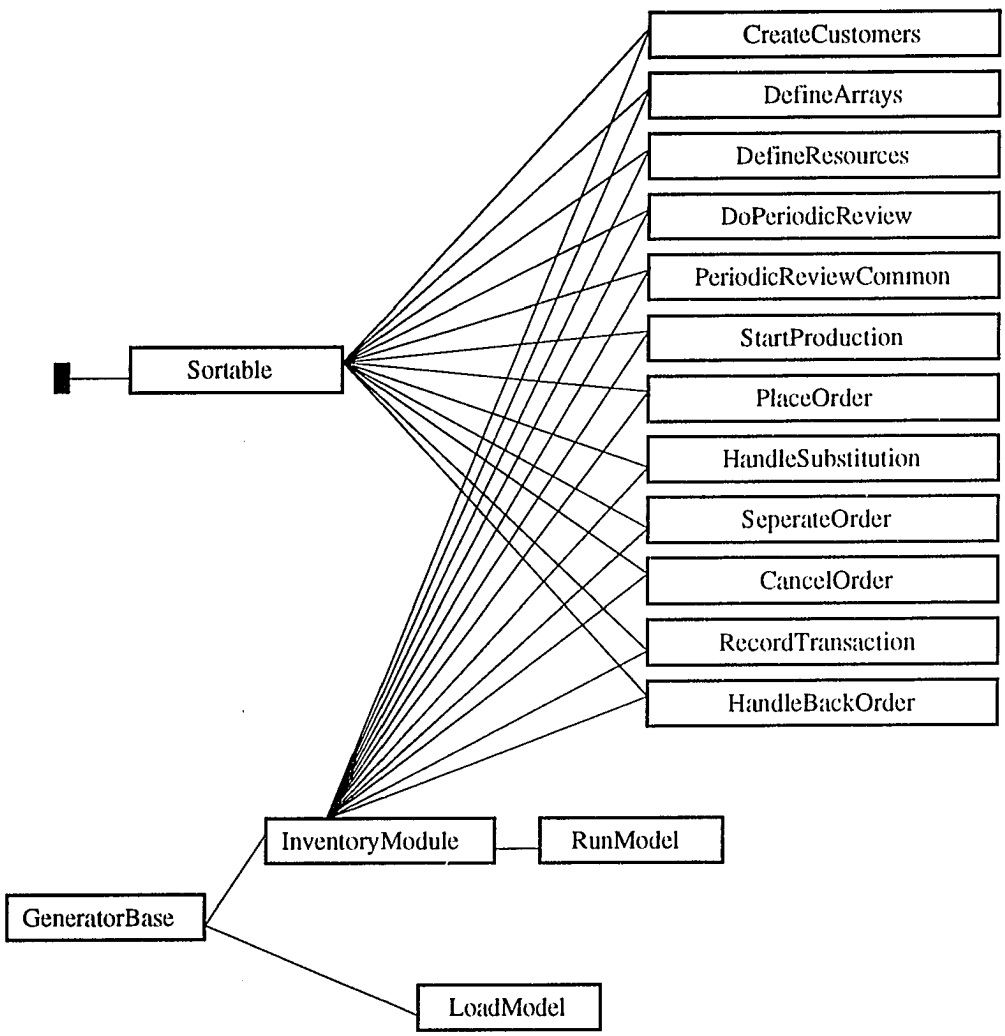

Figure 10. Hierarchy of the Classes for Code Generation.

7. PeriodicReviewCommon: It is responsible for specifying nodes common to all periodic review mechanisms.

8. StartProduction: It is responsible for specifying nodes to start production when inventory level is less than or equal to the reorder point, and then to continue until inventory level is equal to the desired level (S).

Model Description Classes

A model description file is used to store the information about an inventory problem. This file consists of some blocks referring to information pieces in the entire problem such as arrival pattern, price breaks, and inventory control mechanism. Figure 11 shows the hierarchy of the classes designed to handle operations related to model description blocks. Each class in this hierarchy is derived from an abstract base class called *ModelFileBlocks* which defines the common data and functions. These classes are capable of reading from a model description file, and writing modified model description data to the same file based on the pre-defined file structure. The following is the list of data described in individual classes:

1. ModelOutline: modeler name, model name, date, number of items, and types of control mechanism selected.

2. ReportOptions: selected options for simulation reports.

3. SubstitutionData: probabilities of substitution between items, corresponding amount multipliers if any.

4. Distributions: selected distributions for customer arrival, demand, lead time, and processing time.

5. PriceBreaks: definition of breaks for selling and purchasing price for a particular item.

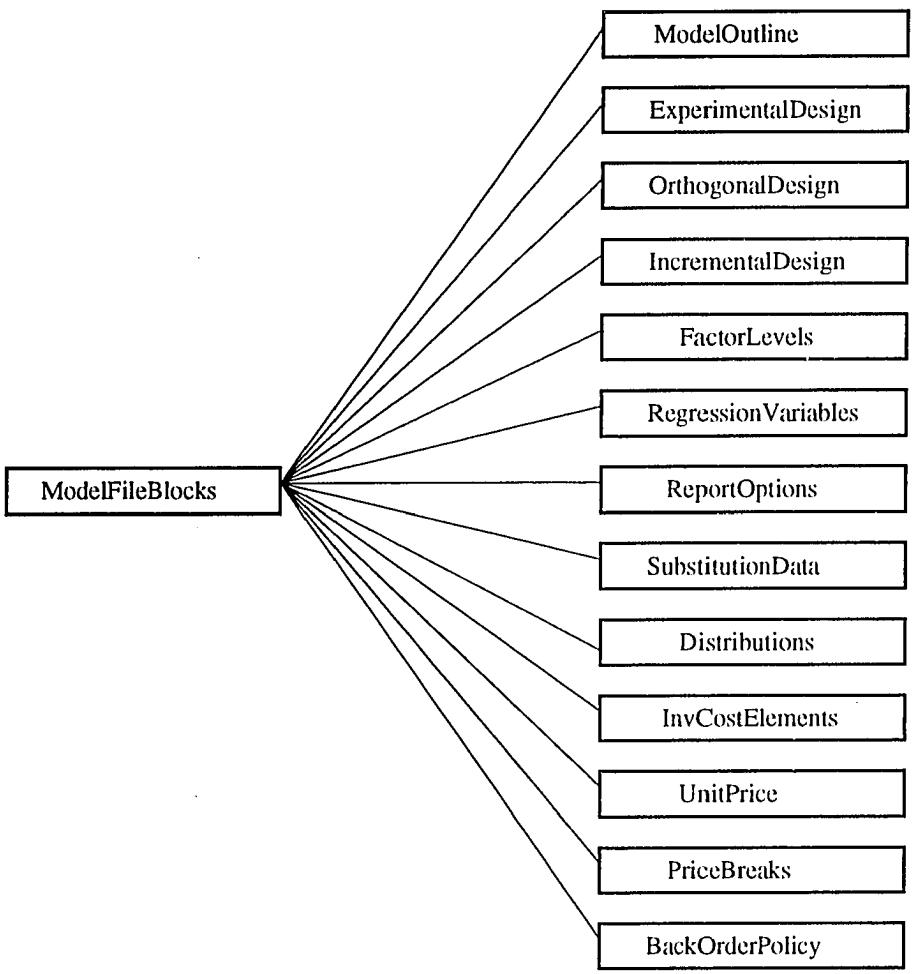

Figure 11. Hierarchy of the Model Definition Classes.

6. BackOrderPolicy: definition of backlog policy and its parameters.

7. InvCostElements: unit costs for setup, holding, backlog, lost sale, and reviewing.

User-Interface Classes

User-Interface classes are all derived from the ObjectWindows Library (OWL) classes supplied by Borland C++ to write Microsoft Windows Applications. These classes are mainly responsible for the following tasks: (a) management of the program windows and user dialogs, (b) handling interactive data entry routines and transferring data into objects in the program, and (c) calling appropriate main menu operation objects. The description of some important classes in the hierarchy (see Figure 12 for individual classes) is as follows:

1. Program initialization classes: They are responsible for performing initialization operations of a Windows program at the beginning such as registering the program into the operating system (i.e., Windows). Classes *MainProgram* and *GeneratorApp* belong to this group.

2. User dialog classes: All classes in this group are derived from the class *TDialog*. They mainly provide the necessary user interface for interactive data entry at model description stage. For example, the class *PriceBreaks* administers user interaction to define price breaks for an inventory item. Some other classes in this group are *PriceDlg*, *InvItemDataDlg*, and *DefineProblem*.

3. Experimental Design Classes: They are constructed to create experimental design matrix for controllable factors (i.e., reorder point, time period, production rate) for a given inventory model. There are three classes in this group :

(a) SelectDesign: choosing experimental design type.

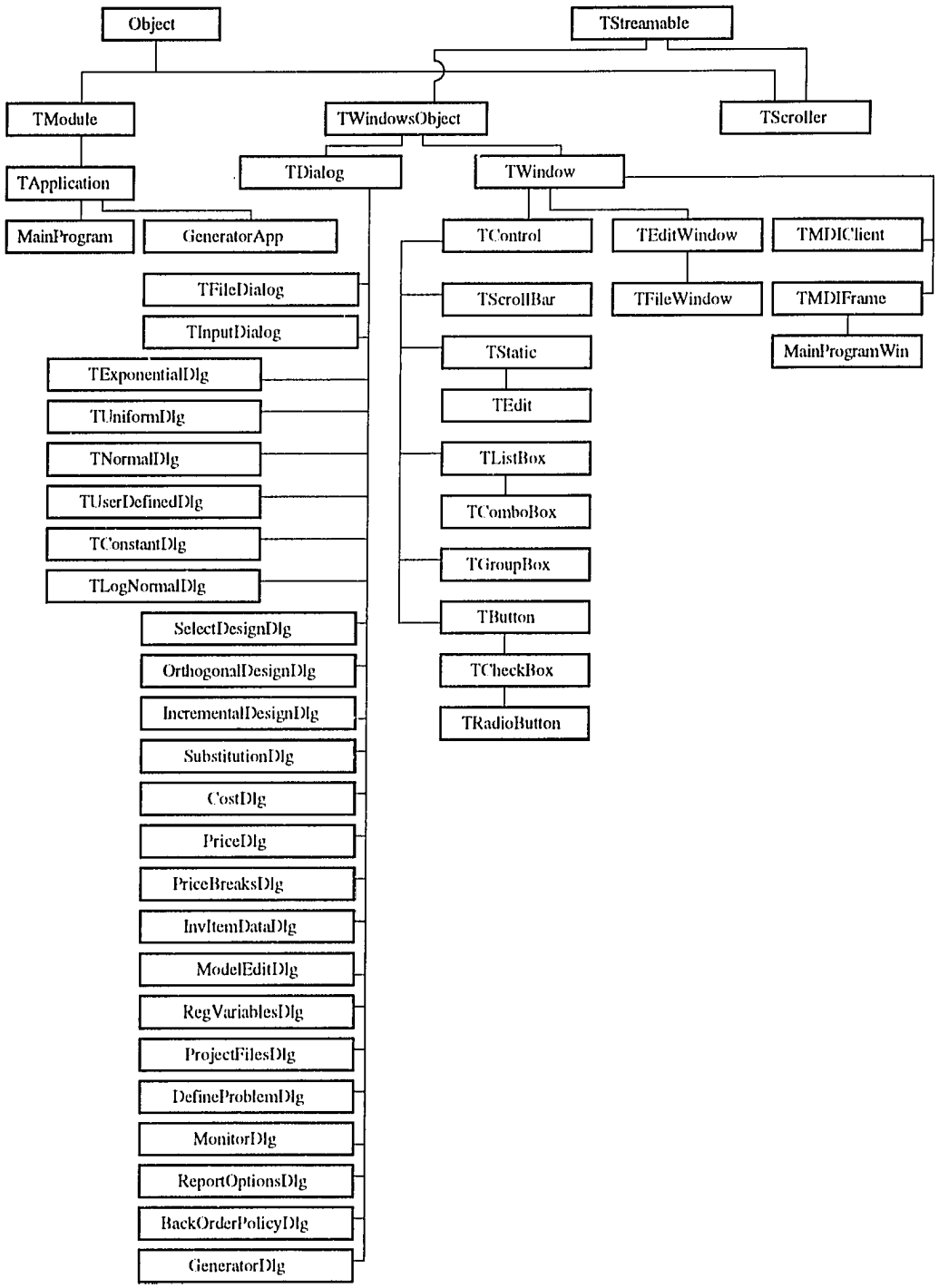

Figure 12. Class Hierarchy of the User-Interface Classes.

(b) OrthogonalDesign: It is used to design full or fractional factorial experiment.

(c) IncrementalDesign: It is used to construct simple incremental design.

4. Interface Control Classes: They are created to control the interface objects in a user dialog box such as list boxes, buttons, and check boxes.

File Organization

A well-defined, flexible, and consistent file handling system is crucial to performing model retrieval, storage and review operations efficiently. In order to satisfy these requirements the developed computer system stores data and results associated with a simulation project in appropriate files with pre-defined structure. At this stage, C++'s flexible and object-oriented file handling system provided excellent tools to perform file operation routines efficiently. In C++ data flow (a) from the console to RAM, (b) from the console to external files, and (c) data flow in the reverse direction are all handled through its unified approach called streams.

The current system has five types of files of which structures are subject to pre-defined rules. These files are: (1) model description file, (2) SLAM II code file, (3) experimental design file, (4) regression input file, and (5) simulation summary report file. Besides these files a project file is used to hold all the file names related to a simulation project to identify them during a session. The name of a project file must have always .PRJ as its extension. A project file example is shown in Figure 13.

The project file can be created either through the program editor, or through the dialog box shown in Appendix F. The content of a project file can be changed any time to associate different files with a project.

```
[PROJECTFILE]
$Files
    ModelFile       = 'INV.MOD',
    SlamFile        = 'INV.DAT',
    DesignFile      = 'DESIGN.DAT',
    RegFile         = 'REG.OUT',
    ReportFile      = 'INV.OUT' $
```

Figure 13. A Project File Example.

All user files in a project are ASCII files. Therefore, any number of these files can be reviewed without quitting the working session through program editor featured with multiple document interface (MDI). MDI is an advanced interface capability that enables the user to edit more than one file during a working session.

Files used in code generation, and model execution are discussed in two main groups in terms of type of data that they contain.

1. Input files: They contain data for the program generator, and experimental design information.

2. Output files: These files are produced by the C++ program (e.g., generated SLAM II code), and by the generated SLAM II program to present results after simulation.

Input Files

Model Description File. It contains information about the inventory problem in pre-defined blocks collected from the user in interactive manner. The order of data blocks, and their structure are designed to be self-explanatory so that the user can understand its content without any difficulty. Its content is used by both the program generator and the SLAM II program.

The general structure of model description file is illustrated in Figure 14. Each italic expression in the figure represents a data block containing specific piece of information about the problem. In Appendix A, an example model description file is provided. Also, Appendix B provides the full description of all variables that appear in the model description file. The model description file is subject to the following basic syntax rules.

1. Fields with italic letters are user input to a particular variable.

2. The comma sign follows each input field to separate it from others.

3. The dollar sign must appear at the beginning and at the end of each description block to show beginning and ending of a block. A description block name follows the first dollar sign in the beginning of a block.

4. Any number of spaces can be inserted between fields to increase readability.

5. All variable names which are typed in normal style must be followed by the equal sign before being initialized.

As a matter of fact the user does not have to know all these rules as long as he or she uses the interactive model entry mode at model description stage. However, after becoming familiar with the structure and syntax, the user can make direct changes in the description file through the program editor.

Experimental Design File. It consists of rows of input data of controllable factors, i.e. inventory policy parameters such as S, and s, for corresponding simulation runs. It is created by the experimental design module based on factor levels, and design type, i.e., orthogonal, or incremental.

[problem]
PROBLEM OUTLINE
[experimental_design]
DESIGN OPTIONS
ORTHOGONAL DESIGN or INCREMENTAL DESIGN
FACTOR LEVELS
REGRESSION VARIABLES
[substitution]
SUBSTITUTION_MATRIX
[options]
REPORT_OPTIONS
MONITOR_OPTIONS

[products]

[$ITEM\_NAME_1$]
[distributions]
CUSTOMER ARRIVALS
DEMAND SIZE
LEAD TIME
ORDER PROCESSING TIME
[cost]
INVENTORY COST ELEMENTS
[pricebreaks]
PURCHASING PRICE
PURCHASING PRICE BREAKS (if any)
SELLING PRICE
SELLING PRICE BREAKS (if any)
[backorder]
BACKORDER POLICY
[END]
[$ITEM\_NAME_2$]
...
define other inventory items(if any) as above.
... ...
... ...
[END]
[EndOfModel]

Figure 14. An Overview of the Model Description File.

Output Files

SLAM II Code File. It is a SLAM II simulation program file generated by the program generator based on the information given in the model description file.

Regression Input File. This file stores the values of user-selected system variables (i.e., controllable factors, and corresponding performance measures) for each run in columns. Later, this file can be used in any of the statistical packages (e.g., SAS) for statistical analysis (i.e., regression analysis, analysis of variance) without any format change. This feature permits integration of the current system with a statistical package to automate optimization process in the future.

Simulation Summary Report File. The simulation output is stored in this file. Model input data (i.e., distributions, costs, price breaks, etc.) are printed in the beginning of the file. Performance measures (e.g., total cost, average holding level, etc.) for individual items in the system are summarized in order. At the end, overall system performance measures (e.g., overall total cost, average profit, etc.) are reported.

Code Generation Process

The developed system has a domain-dependent program generator. At present the generator is designed to generate simulation code for inventory systems. It is possible to add more application domains to the existing system using the classes defined for the present system.

Program generators are intelligent front ends to existing simulation languages (O'Keefe, 1986). Program generation is an intelligent work which requires to develop reliable, and correct simulation code for a given problem based on the pre-defined

rules and facts. A system designer should follow the following logical steps to design a generator:

1. Define problem domain: It is to outline the boundaries of the problem domain in clear terms. In general the more flexibility the program generator has, the more complex it becomes.

2. Familiarize with domain: This step is devoted to understand the problem domain thoroughly in terms of input-output relations between system variables. The purpose of this step can be accomplished by familiarizing with variants of the problem domain from different degree of difficulty. Coding each category of problems helps in the requirements of the program generator. The designer of generator attempts to draw rules, and facts regarding the structure of the simulation code. At this stage, the developer should be able to construct a family of well-structured program codes.

3. Transfer of knowledge: Next step is to transfer the knowledge of the expert as rules, and facts to the program generator's inference engine. Inference engine is an intelligent program which is capable of deciding what SLAM II nodes, and control statements should be used for a given problem. This program usually contains many if-then statements to process model description file to make its own decisions in generating simulation program. In this respect the correctness of the rules in the inference engine is vital.

For the present generator, the same steps were carried out. A family of SLAM II programs for various inventory problems were created to see how the program structure was changing, and what types of simulation event handling blocks were needed. Later, the entire program code was broken down into submodules of SLAM II program blocks. The rules related to selection of the SLAM II nodes, statements, and specifying their arguments are encapsulated within the code generation classes dis-

cussed earlier. For example, the class *CreateCustomers* is responsible for deciding the necessary nodes, and their specifications to create customer entities in the system. Figure 15 shows the generalized SLAM II network diagram used as modeling rule in this class. In the figure, note that depending upon number of items, the number of the CREATE nodes required changes along with arguments (i.e., ARRIVAL1, ARRIVAL2). Also, the number of resource blocks is determined by the number of inventory items. All the code generation classes are capable of sorting the required SLAM II nodes for processing.

Another important decision at code generation stage is to select the required variables, arrays, and attributes associated with entities that are used throughout the program. The class *DefineArrays* performs this operation in the beginning of code generation for a given problem.

Object-oriented approach is implemented throughout the code generation process. Figure 16 illustrates the two main stages in code generation. The preparation stage prior to code production consists of the collection of data from the user, creation of model description file, and then skimming the information in that file to detect factors that affect the selection of the SLAM II nodes. For example, types of inventory control mechanisms, backlog policies, and whether it is a single or multi-item system are some of the factors that the generator takes into consideration.

Next step is the code production stage in which the program generator creates objects of the code generation classes described earlier. For instance, it is necessary to create the object of the class *StartProduction*, if the inventory problem is a production model. Next, the created objects of code generation classes make decision on the selection of the nodes internally. Later, based on the pre-defined priority rules, a priority number is attached to each object of code generation classes before being inserted to

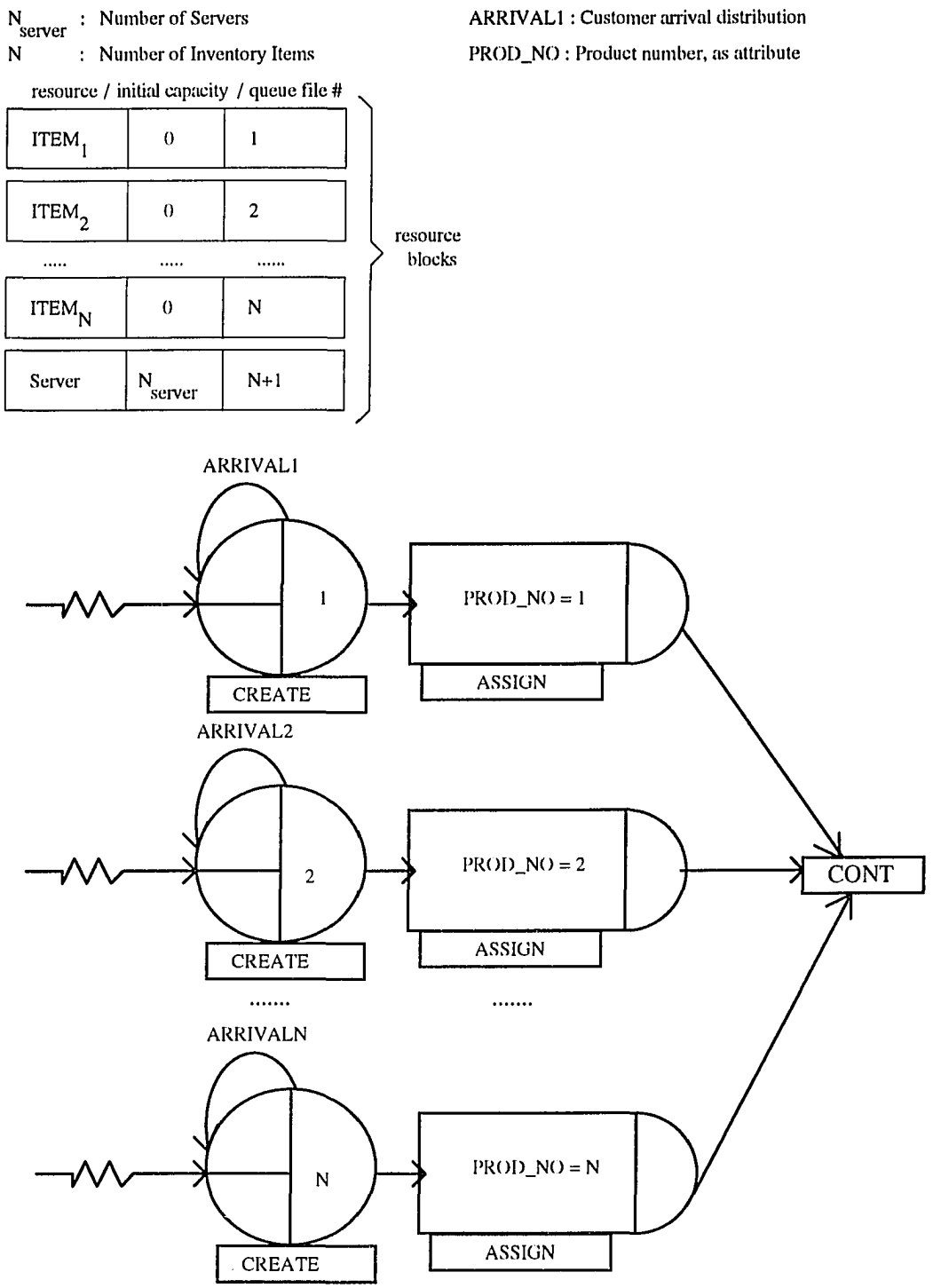

Figure 15. Generic SLAM II Network Diagram of Customer Arrivals and Resource Blocks.

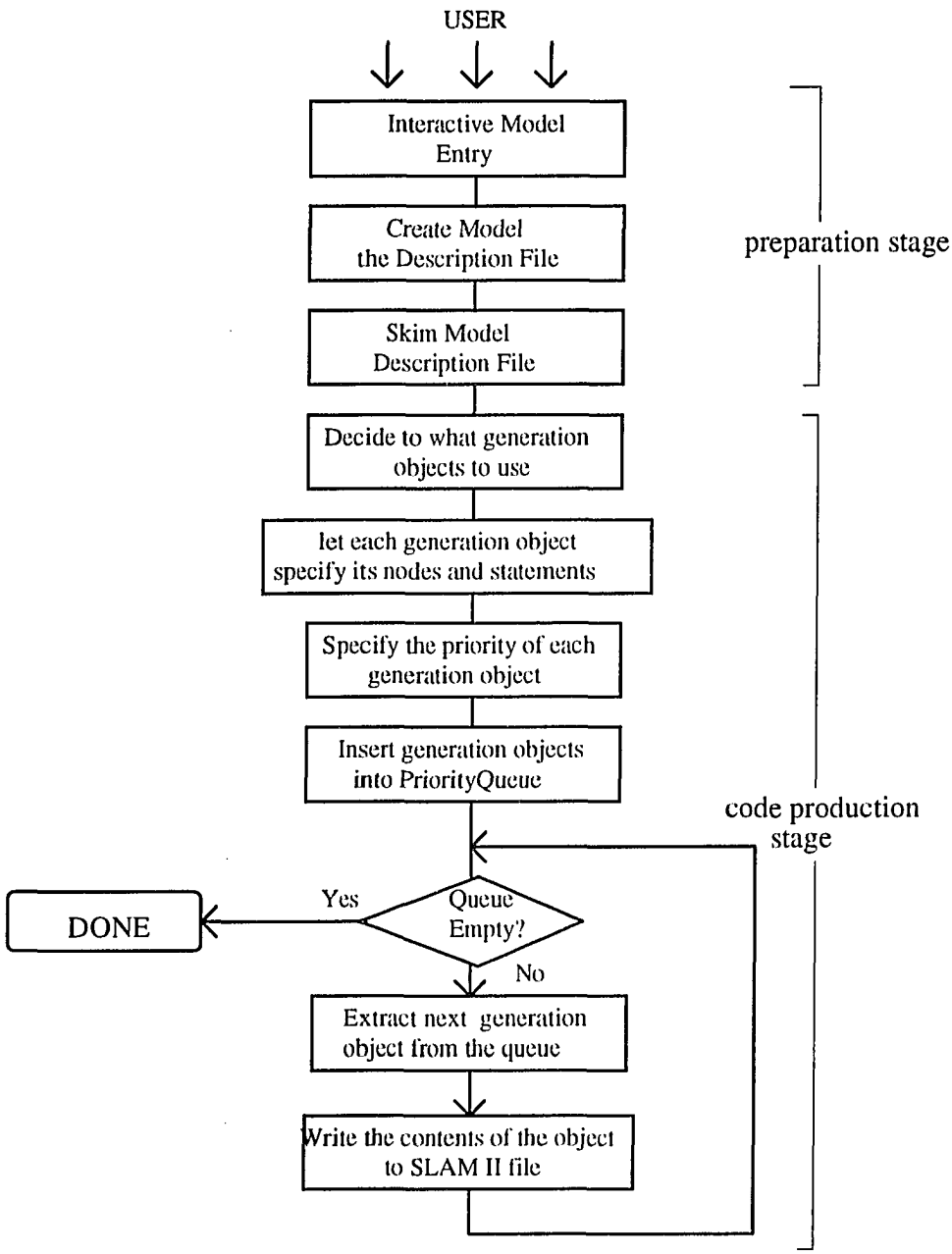

Figure 16. Flow Chart of the Code Generation Process.

the *BlocksQueue* which is an instance of the class *PriorityQueue*. The logic behind this is to make sure that the correct order of the objects (i.e., blocks) of code generation classes as it appears in the simulation program code is achieved. At the last step, the contents of the objects (i.e. SLAM II nodes) are written to the user-defined SLAM II file by extracting them from the *BlocksQueue* in order.

CHAPTER V

AN IMPLEMENTATION - CASE STUDY

Definition of the Example Inventory Model

An inventory model having three products was built to demonstrate some modeling features of the integrated model development environment of this study. All products were subject to different inventory control mechanisms. From now on the products in the model will be referred as Product 1, Product 2, and Product 3. The model description file of this example is provided in Appendix A. It is recommended to review this file for more detailed data on the example problem along with the detailed description of the fields provided in Appendix B. The following is the summary of some important model data:

1. Control mechanisms: Product 1, and Product 2 are reviewed continuously, and periodically respectively to decide to place a new order. Product 3 is also reviewed continuously with one difference that when a new setup takes place, replenishment is made continuously at a constant rate (i.e., production, or supply rate) rather than a bulk at once.

2. Factor levels: In this example single simulation was run for inventory policies obtained using the simple EOQ model with some modifications for Product 1, Product 2 (see Table 1). Production rate was additional parameter for Product 3 in the model. A production rate (65 items/day) higher than the demand rate (i.e., 50 items/day) was selected to prevent the possibility of infinite queue of demands (customers).

3. Statistical distributions: Statistical distributions were attributed to demand arrivals, demand size, and lead time. Table 2 summarizes the distribution types and their parameters used in the example with respect to each product.

4. Unit costs: Holding, backlog, setup, demand cancellation, and review costs associated with each product in the model can be seen in the model description file provided in Appendix A.

Table 1

Inventory Policy Parameters for the Example Inventory Problem

Product No	*S* (reorder level) [units]	*s* (reorder point) [units]	*t* (review period) [days]	*r* (rate) [units per day]
1	290	132	N/A	N/A
2	1224	250	9	N/A
3	300	200	N/A	65

5. Substitution: Substitution between Product 1 and Product 3 was allowed on one to one basis. On the other hand Product 2 cannot be substituted by any product in case of backorder.

6. Price breaks: Product 1 has price breaks for both purchase (or, manufacturing) and selling prices, while Product 2, and Product 3 have a single unit price instead (see the model description file for the related data).

7. Backlog polices: Product 2 has customer renege probability of 0.5 for back-order cases. If 50 units are waiting in the queue (i.e., backordered), next customer for Product 1 reneges the system. Customer renege probability for Product 3 depends on the expected waiting time of the customer. Lower, and upper limits are supplied by the modeler for expected waiting time as parameters to determine the probability of customer renege. Particularly for this example, if expected waiting time for a customer arriving to the system is less than or equal to the lower limit, 2 days, the customer will accept to be backordered (i.e., no demand cancellation). Expected times longer than the upper limit, 7 days, cause the customer to leave the system (i.e., cancel the order). A conditional probability depending on expected waiting time was defined for expected waiting times between two and 7 days (see Equation 10).

Table 2

Parameters of Statistical Distributions Used in the Example Model

	Product No		
Random Variable	1	2	3
Customer Arrivals	Exponential $\lambda = 0.5$	Constant = 0.2	Exponential $\lambda = 0.5$
Demand Size	Discrete $\mu = 22$	Normal $\mu = 25$, $\sigma = 5.0$	Normal $\mu = 15$, $\sigma = 2.5$
Lead Time	Constant 3.0	Constant 3.0	Uniform $a = 2.0$, $b = 3.0$

Code Generation in SLAM II and Execution of the Model

After entering the model data using the interactive session of the data collection program, corresponding simulation program code in SLAM II can be created through the program generator. In Appendix C the list of the SLAM II program is provided for the example.

A single simulation of the example problem was run for 3500 days to get fairly good estimates of the performance measures mentioned in Chapter III resulting in less error variance, and tighter confidence intervals. This way, randomness of some system parameters such as lead time, and customer arrivals can be absorbed in longer run to get steady-state estimates of the system output variables, average holding level, average backorder level, average cycle length, and etc.

Discussion of the Simulation Results

Generated SLAM II program provides some statistics in well-organized reports at the end of simulation for pre-defined system performance measures mentioned in Chapter III. SLAM II outputs are discussed in the following sections in terms of their content:

1. Standard output for individual products: This report is produced by the FORTRAN OTPUT subroutine in addition to the SLAM II standard output to provide separate output for each product in the model (see Figure 17). In this output, statistics on the average holding level, average backlog level, inventory costs, average profit, and the other performance measures are provided. Looking at the report, we observed that within the simulation run length, i.e., 3500 days, 7070 customers arrived for Product 1, and 307 of them were actually customers for the other products that were

substituted by Product 1 due to the shortage. 6928 customer orders were met while 31 customers (i.e., 3% of the demand) canceled their order due to shortage on hand inventory. Backorder policy was the influential factor that determined the number of the order cancellations. In the report, inventory cost elements are classified into setup cost, holding cost, backlog cost, order cancellation cost, and review cost. The last line in the report provides the average profit per day from the product, which is average sales per day minus average inventory and purchasing (or, manufacturing) cost. The average profit was $161.83 per day for Product 1.

2. Overall system performance measures: In this report results for overall holding cost, backlog cost, setup cost, average profit, and other related cost terms are provided as aggregate total of costs given in individual product reports (see Figure 18 for the example report). As an important performance variable, average profit per day for the entire system was $367.8.

3. Substitution statistics: In this report the number of customer orders substituted and its total amount in units is provided for each pair of products. For instance, looking at the report for the example in Figure 19, 307 customers, 4650 units of Product 1 were substituted by Product 3 due to shortage in inventory of Product 1 on hand. On the other hand substitution figures for the other cells are zero as the substitution probabilities are defined to be zero for these cells in the substitution matrix. Significance, and magnitude of customer flow between products can be evaluated analyzing the substitution statistics. Ability to substitute one product by another may decrease the number of customer order cancellation drastically if reorder levels, and reorder points permit the significant interaction. In other words, if the two products have very high reorder points, substitution may not occur at all as the shortage probabilities of both products appear to be very low.

4. SLAM II standard outputs: SLAM II provides standard outputs for queues, resources, time-persistent variables, and so on. Simulation output is organized in the following groups.

(a) Statistics based on observation: Average safety stock level, average time between customer reneges, and average cycle length for each product are calculated based on observations acquired during simulation. Looking at Figure 20 it can be seen that there is no customer renege observed for Product 3 due to its high safety stock level, i.e., 63.7 units. On the other hand, Product 2 has no safety stock on the average resulting in the highest number of order cancellations, i.e., 3840 customers. Cycle length is defined to be the time interval between two consecutive production starts, or manufacturer orders. Product 3 had the largest average cycle length as being consistent with 147 setups occurred simulation run. Since the production rate of Product 3 was chosen above the demand rate in order to absorb the fluctuations in probabilistic demand, the manufacturer was able to satisfy the demand on regular basis instead of using bulk orders.

(b) Statistics for time-persistent variables: Time-weighted average of backorder level was determined for each product by recording changes over time. Product 1 had the highest average backlog level as being consistent with the pervious results for customer renege, and safety stock. Average backlog level for Product 3, 0.36 units was the lowest among others with a maximum value of 173 units.

(c) File statistics: In SLAM II program customer entities were subject to wait in corresponding queues when they are backordered. Looking at queue file statistics (see Figure 22) we can get information on average customer length, maximum length, and average waiting time. There were 5 customers waiting for Product 2 on the average, and the maximum customer length was 29 customers.

PRODUCT NO : 1

DEMAND
(in number of customers)

Satisfied	:	6928
Lost(cancel)	:	31
Substituted	:	418
(-) Used for substitution	:	307
Total	:	7070

(in units of products)

Satisfied	:	151050
Lost(cancel)	:	760
Substituted	:	9370
(-) Used for substitution:		4650
Total	:	156530

HOLDING/BACKLOG/SETUPS

Holding Level [units/day]	:	158.71
Backlog Level [units/day]	:	1.19
Number of Setups	:	506
Number of Reviews	:	6928

COSTS/PROFIT

(+) Sales [$/day]	:	718.04
Setup Cost[$/day]	:	21.69
Holding Cost[$/day]	:	15.87
Backlog Cost[$/day]	:	0.24
Cancel Cost[$/day]	:	0.54
Review Cost[$/day]	:	0.02
Inventory Cost[$/day]	:	38.37
Purchase Cost [$/day]	:	517.86
(-) Total Cost [$/day]	:	556.22
Profit [$/day]	:	161.83

Figure 17. Simulation Report for Individual Products (Example Given for Product 1).

Overall System Performance Measures

COSTS/PROFIT		
(+) Sales [$/day]	:	1410.11
Setup Cost[$/day]	:	43.53
Holding Cost[$/day]	:	38.67
Backlog Cost[$/day]	:	9.84
Cancel Cost[$/day]	:	11.49
Review Cost[$/day]	:	0.59
Inventory Cost[$/day]	:	104.11
Purchase Cost [$/day]	:	938.20
(-) Total Cost [$/day]	:	1042.31
Profit [$/day]	:	367.80

Figure 18. Simulation Report for Aggregated Product Measures.

Substitution Statistics

Substituted by	number of customers(amount in units)		
	1	2	3
1	0(0)	0(0)	418(9370)
2	0(0)	0(0)	0(0)
3	307(4650)	0(0)	0(0)

Figure 19. Simulation Report for Substitution Statistics.

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
SAFETY STOCK1	0.1825E+02	0.2766E+02	0.1516E+01	0.0000E+00	0.1220E+03	506
SAFETY STOCK2	0.0000E+00	0.0000E+00	0.9999E+04	0.0000E+00	0.0000E+00	196
SAFETY STOCK3	0.6374E+02	0.4512E+02	0.7079E+00	0.0000E+00	0.1630E+03	147
TIME BET. CANCEL1	0.1082E+03	0.1759E+03	0.1626E+01	0.3174E-02	0.8429E+03	31
TIME BET. CANCEL2	0.9097E+00	0.2293E+01	0.2521E+01	0.2000E+00	0.2120E+02	3840
TIME BET. CANCEL3			NO VALUES RECORDED			
CYCLE LENGTH1	0.6915E+01	0.1721E+01	0.2489E+00	0.3605E+01	0.1214E+02	506
CYCLE LENGTH2	0.1783E+02	0.1211E+01	0.6795E-01	0.9000E+01	0.1800E+02	196
CYCLE LENGTH3	0.2360E+02	0.1039E+02	0.4404E+00	0.2594E+01	0.7859E+02	147

Figure 20. SLAM II Output for Safety Stock, Customer Renege, and Cycle Length.

STATISTICS FOR TIME-PERSISTENT VARIABLES

	MEAN VALUE	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	TIME INTERVAL	CURRENT VALUE
BACKLOG LEVEL1	0.1188E+01	0.5727E+01	0.0000E+00	0.7700E+02	0.3500E+04	0.0000E+00
BACKLOG LEVEL2	0.1196E+03	0.1697E+03	0.0000E+00	0.7080E+03	0.3500E+04	0.0000E+00
BACKLOG LEVEL3	0.3619E+00	0.4152E+01	0.0000E+00	0.1730E+03	0.3500E+04	0.0000E+00

Figure 21. SLAM II Output for Time-Weighted Average of Backlog Level.

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	AWAIT	0.0824	0.3383	5	0	0.0401
2	AWAIT	5.0044	6.9767	29	0	1.2647
3	AWAIT	0.0396	0.3503	11	0	0.0116

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	ITEM1	282	158.7029	0	412
2	ITEM2	492	338.2148	0	1308
3	ITEM3	221	147.1078	0	360

Figure 22. SLAM II Output for Queue, and Resource Statistics.

CHAPTER VI

APPLICATION OF RESPONSE SURFACE METHODOLOGY AT OPTIMIZATION STAGE

Description of the Problem

It is a difficult task to build exact analytical models under few assumptions for complex inventory models that embrace probabilistic demand, customer arrivals, lead times, and state-dependent decision mechanisms. Simulation can represent these types of inventory systems in greater detail due to its unrestrictive approach which requires fewer assumptions in modeling a problem. However, due to the existence of unknown mathematical relationships between system variables, intense computational effort is required for optimization in simulation. In Chapter II some of the well-known optimization methods for simulation were investigated. Here in this chapter, the applicability of RSM to inventory problems is demonstrated by finding the optimum inventory policy of the probabilistic continuous review inventory problem based on the simulation output. A deterministic approximation of the original problem was also developed in order to compare and evaluate the performance of RSM, and be able to validate the results.

The proposed inventory problem has the following characteristics:

1. Time between demand arrivals is exponential.
2. Demand size for a customer is normally distributed.
3. Lead time is normally distributed.
4. Inventory level is reviewed after each transaction to decide a new order, i.e.,

(S,s) model with continuous review.

5. Customers renege the system at certain probability p_r when backordered.

6. Holding, backlog, setup costs are constant.

7. Purchase (manufacturing) cost, and selling price are assumed to be constant.

8. Response variable to be maximized is average profit per day.

The following specific data are supplied for the example problem:

1. Customer arrivals: exponential, $\lambda=10$ customers per day.

2. Demand size: normal distribution, $\mu=20$, $\sigma=2.5$.

3. Lead time: normal distribution, $\mu=3.0$, $\sigma=0.5$.

4. Costs: holding cost = \$0.3 per unit per day, backlog cost = \$0.4 per unit per day, setup cost = \$1500 per setup, purchase cost = \$11 per unit, and selling price = \$14 per unit.

5. Renege probability $p_r=0.6$.

Optimization Using RSM

Statistical, and mathematical principles of RSM were well presented previously in Chapter II. Generalized steps of the optimization procedure for inventory problem are summarized in Figure 23.

The distance of the starting point from the true optimum affects the number of runs needed to reach the optimum in great deal. In general it can be claimed that the closer the starting point to the true optimum, the fewer the number of runs. In this regard the simple EOQ model with no-shortage permitted can provide a good starting point for the search process. The equivalent (S,s) policy of an EOQ solution can be calculated from the following formulas:

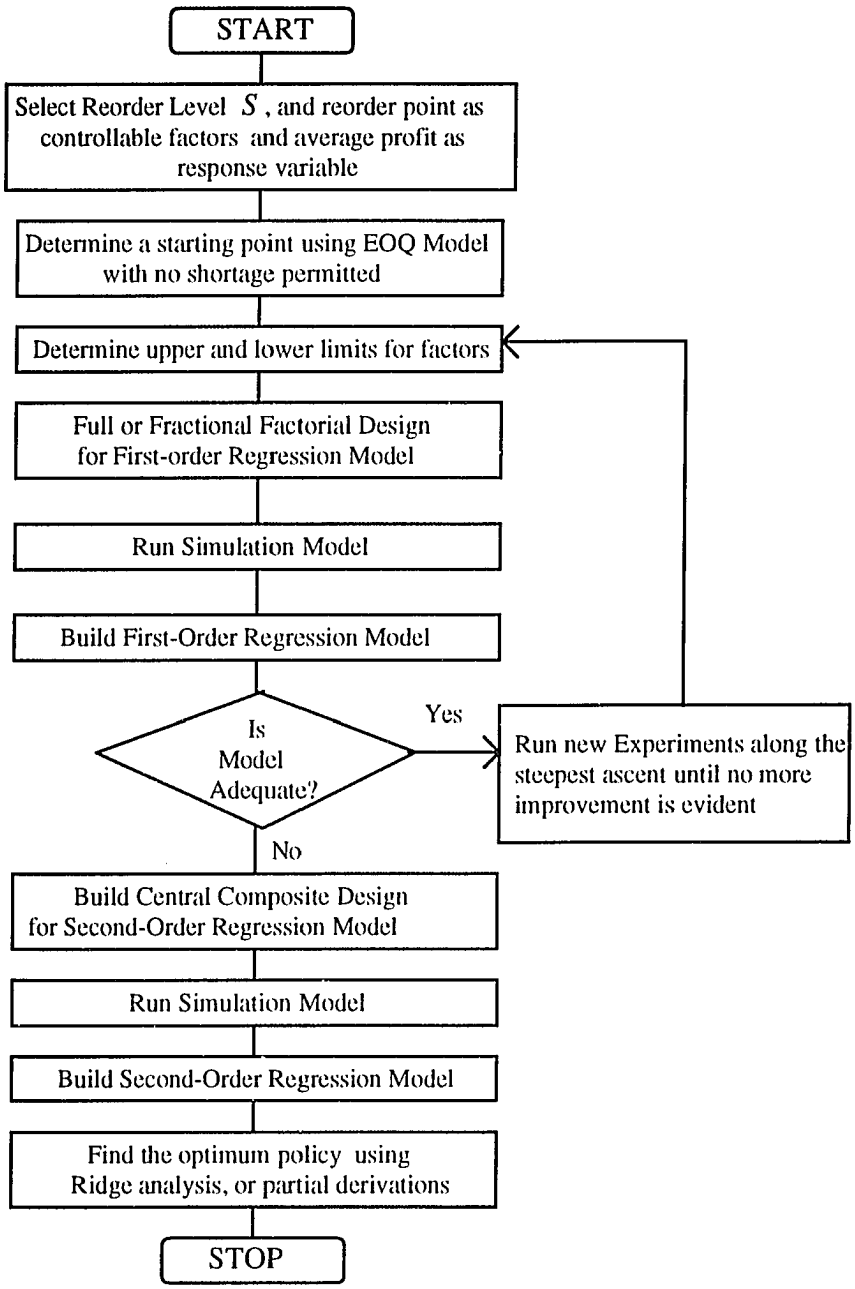

Figure 23. Flow Diagram of Response Surface Methodology Along With the Steepest-Ascent for Optimization of the Inventory Model.

$$S = \sqrt{\frac{2aK}{h}} = \sqrt{\frac{2*200*1500}{0.3}} = 1414 \text{ units}$$

$$s = La = 3*200 = 600 \text{ units}$$

A two-level orthogonal design with starting point (1414,600) being the center of the design was constructed to make runs for the first-order model, and to estimate the first steepest-ascent, that is

$$\begin{vmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{vmatrix} \Rightarrow \begin{vmatrix} 1364 & 575 \\ 1364 & 625 \\ 1464 & 575 \\ 1464 & 625 \end{vmatrix}$$

Based on the model description, corresponding simulation model in SLAM II was generated automatically by the program generator. Initial four simulation runs for the first experimental design were performed, and the results are summarized in Table 3.

Table 3

Results of Simulation Runs for First-Order Regression Model in Iteration I

Run No	S	s	Average Profit
1	1364	575	176.88
2	1364	625	168.95
3	1464	575	176.89
4	1464	625	168.72

Let x_1, x_2 be the coded variables for X_1, and X_2 which represent S, and s respectively. Based on data in Table 3 the following first-order regression model can be constructed using SAS statistical software (SAS Institute, 1985):

$$y = 172.86 - 0.055x_1 - 4.025x_2$$

where y is average profit per day. The F-value was 2250.7, and corresponding p-value (i.e., significance probability) was 0.015 which was less than 0.05 indicating that linear relationship was significant. Since the coefficient of the term x_1 in the model is negative, and the problem is a maximization, a negative increment for X_2 must be selected to increase y. Let ΔX_2 be -20 units from the center of the design (i.e., 600), and then using the following equation increment in the coded variable can be also calculated:

$$x_i = \frac{X_i - \overline{X_i}}{s_{xi}} \qquad (13)$$

where s_{xi} is the scale factor of i th variable X_i which is the selected equal distance from the center of design to the lower, and upper levels. By placing the values in Equation 13, we get

$$x_2 = \frac{-20}{25} = -0.8 \text{units}$$

Using Equations 6, and 7 in Chapter II, we find the Lagrange multiplier to be

$$\lambda = \frac{-4.025}{-2 * 0.8} = 2.515,$$

and then the corresponding increment in x_1

$$x_1 = \frac{-0.055}{2*2.515} = -0.01$$

Next ΔX_1 is obtained to be -0.5 units by multiplying x_1 by s_{x1}, i.e. 50.

A series of simulation starting at the center of the design were run until no more increase in y was evident. The simulation run length was determined to be 3500 days for each run so that steady-state results could be achieved with less random error variance in the response variable. Common pseudo-random numbers for each run were also used to reduce variance in error. The results are summarized in Table 4.

Table 4

Coordinates Along the Path of Steepest Ascent (Uncoded Variables), and the Response Variable (Average Profit Per Day), for Iteration I

Run #	Increment	X_1	X_2	y
1	base	1414.0	600	176.03
2	base + Δ	1413.5	580	177.83
3	base + 2Δ	1413.0	560	177.98
4	base + 3Δ	1412.5	540	178.71
5	base + 4Δ	1412.0	520	179.41
6	base + 5Δ	1411.5	500	180.34
7	base + 6Δ	1411.0	480	180.71
8	base + 7Δ	1410.5	460	180.89
9	base + 8Δ	1410.0	440	181.93
10	base + 9Δ	1409.5	420	182.38
11	base + 10Δ	1409.0	400	182.16

Table 4--Continued

Run #	Increment	X_1	X_2	y
12	base + 11Δ	1408.5	380	182.43
13	base + 12Δ	1408.0	360	182.81
14	base + 13Δ	1407.5	340	183.30
15	base + 14Δ	1407.0	320	183.35
16	base + 15Δ	1406.5	300	183.44
17	base + 16Δ	1406.0	280	183.99
18	base + 17Δ	1405.5	260	184.45
19	base + 18Δ	1405.0	240	184.47
20	base + 19Δ	1404.5	220	183.66
21	base + 20Δ	1404.0	200	183.24

The two consecutive decreases in y after 19th run (1405, 240) suggested that a new path is required to increase y. Another two-level orthogonal design with (1405,240) being the center of the design was constructed within a smaller search region to estimate the next search path. Table 5 shows the results of simulation runs made at this step.

Table 5

Results of Simulation Runs for First-Order Regression Model in Iteration II

Run #	S	s	Average Profit
1	1385	260	182.95
2	1385	220	182.97
3	1425	260	182.86
4	1425	220	182.87

Based on the observations in Table 5, first-order regression model in coded variables for Iteration II was

$$y = 182.91 - 0.0475x_1 + 0.0075x_2,$$

and F-value, and p-value of the H_0 hypothesis that there is a linear associationship between the dependent, and the independent variables were 185.000, 0.05 respectively. The new path was determined in the same way as in Iteration I by selecting ΔX_1 to be -5 units, and then $\Delta X_2 = 0.8$. Further runs are made in order to increase average profit along the new path. As seen in Table 6 , increase in y stops in 4th run.

Table 6

Coordinates Along Path of Steepest Ascent (Uncoded Variables), and the Response Variable (Average Profit Per Day), for Iteration II

Run #	Increment	X_1	X_2	y
0	base	1405.0	240.0	184.47
1	base + Δ	1400.0	239.2	184.56
2	base + 2Δ	1395.0	238.4	184.90
3	base + 3Δ	1390.0	237.6	185.12
4	base + 4Δ	1385.0	236.8	183.55

However when another first-order model was built about (1390,237.6), $p-$value for linear associationship was 0.505 indicating the need for a second-order design due to inadequacy of first-order design. Therefore, a central composite design having nine design points which are equidistant from the center of the design, i.e., (1390,238) was built to achieve rotatability necessary for less error variance, and

fewer runs (see Figure 24 for the plot of the points). The design matrix used was as follows:

$$\begin{vmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 0 & 0 \\ 1.414 & 0 \\ -1.414 & 0 \\ 0 & 1.414 \\ 0 & -1.414 \end{vmatrix} \Rightarrow \begin{vmatrix} 1340 & 208 \\ 1340 & 268 \\ 1440 & 208 \\ 1440 & 268 \\ 1390 & 238 \\ 1460 & 238 \\ 1319 & 238 \\ 1390 & 280 \\ 1390 & 195 \end{vmatrix}$$

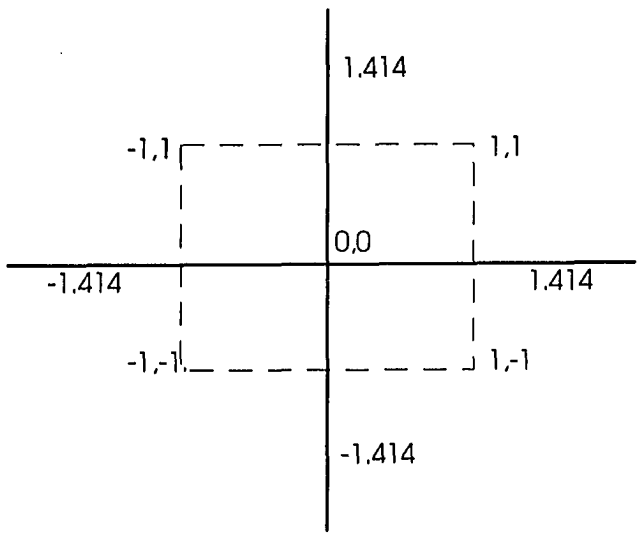

Figure 24. Central Composite Design for Two-Factor Experiment With the Distance From the Center $\alpha = 1.414$.

Four replications are made at the center of the design to improve the quality of the parameter estimation due its highly influential location in the design. Based on the twelve simulation runs made, the following second-order model in terms of the un-coded variables was reached:

$$y = 15.605 + 0.239X_1 + 0.0216X_2 - 0.86.10^{-4}X_1^2 - 0.45.10^{-4}X_2^2 \qquad (12).$$

Looking at Table 7, it appears that all terms in the model, i.e., intercept, first-order, and second-order terms, except the interaction term X_1X_2 were significant.

Table 7

Second-Order Regression Model Parameter Estimates and t - Test Results

Parameter	Degrees of Freedom	Estimate	Standard Error	t -value	p-value
Intercept	1	15.6050	2.0310	7.683	0.0003
X_1	1	0.2390	0.0056	86.898	0.0000
X_2	1	0.0216	0.0031	6.958	0.0004
X_1X_2	1	$5.9.10^{-15}$	$0.2\text{x}10^{-5}$	$2.9\text{x}10^{-9}$	1.0000
X_1^2	1	-0.8610^{-4}	$0.974\text{x}10^{-6}$	-88.205	0.0000
X_2^2	1	$-0.45\text{x}10^{-4}$	$0.267\text{x}10^{-5}$	-16.602	0.0000

By obtaining partial derivations of Equation 12 with respect to X_1,and X_2, and equating them to zero, we can find the following optimum solution:

$$\frac{\partial y}{\partial X_1} = \frac{0.239}{2*0.86*10^{-4}} = 1389.5$$

$$\frac{\partial y}{\partial X_2} = \frac{0.0216}{2*0.45*10^{-4}} = 240.0.$$

In order to confirm the mathematical solution with the canonical ridge analysis results, the RSREG procedure of SAS was called using the 12 observations of the second-order model. As expected, the optimum solution results were very close the ones obtained through partial derivations of the second-order model. Namely, the ridge analysis solution was 1391.1, 242.5 for S and s respectively. Average profit at this point was predicted to be $184.48 per day.

Deterministic Model Approximation

A deterministic model approximation of the present probabilistic inventory problem was introduced in order to validate simulation, and optimization results. Basic approach in approximation was to use the mean values of the distribution functions attributed to customer arrivals, demand size, and lead time by ignoring the randomness in demand size, customer arrivals. Two additional approximations are also made with respect to depletion rate separately for positive inventory level, and negative inventory level periods. It is approximated that depletion rate can be calculated as follows:

$$a = \frac{1}{t_a} d$$

where t_a is mean time between customer arrivals, d is mean demand size per customer. Subsequently, depletion rate during negative inventory level period is approximated to be ap_r, where p_r is customer renege probability when customer is

backordered. The determistic model was broken down into three different inventory submodels with respect to relationships among S, s and Q as formulas for order quantity, cycle time, time-weighted average holding, and backlog cost differ from one case to another. For all submodels average profit is defined as

Sales – Purchasing cost – Holding cost – Backlog cost – Setup cost [$ per unit of time]

Note that all cost formulations are also dollar per unit of time in all cases.

Submodel I. $(s \geq 0, Q > S)$

Figure 25 depicts the change inventory level over time for this submodel. The following formulas are obtained to find average profit per unit of time.

(a) order quantity:

$$Q = S + (L - \frac{s}{a})a(1 - p_r),$$

(b) cycle length in unit of time:

$$t_c = \frac{(S - s)}{a} + L,$$

(c) holding cost per unit of time:

$$\frac{S^2 h}{2at_c},$$

(d) backlog cost per unit of time:

$$\frac{(L - \frac{s}{a})^2 a(1 - p_r)p}{2t_c}$$

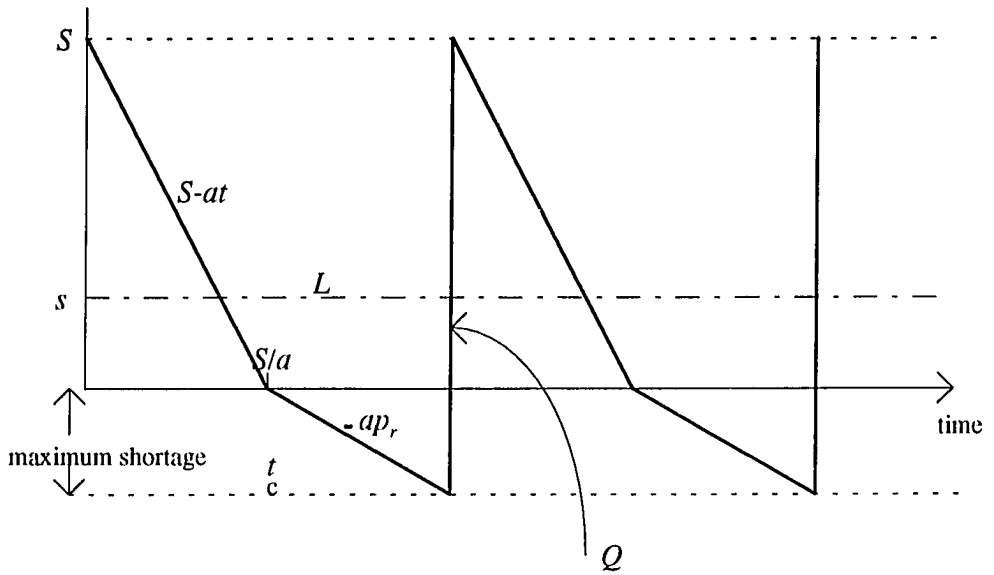

Figure 25. Diagram of the Submodel I.

Submodel II ($s > 0, Q < S$):

Based on Figure 26 the following formulas are valid for this case:

(a) cycle length: It is the same as case I.

(b) order quantity: $Q = S - s + La$

(c) holding cost

$$\frac{2S - Q}{2t_c}$$

(d) backlog cost $= 0$

Submodel III ($s < 0$):

Figure 27 shows the diagram of this case where reorder point is negative. This case may be applicable to situations where a new order is placed only after a certain amount of demand is accumulated in the system. The formulas for this case are as follows:

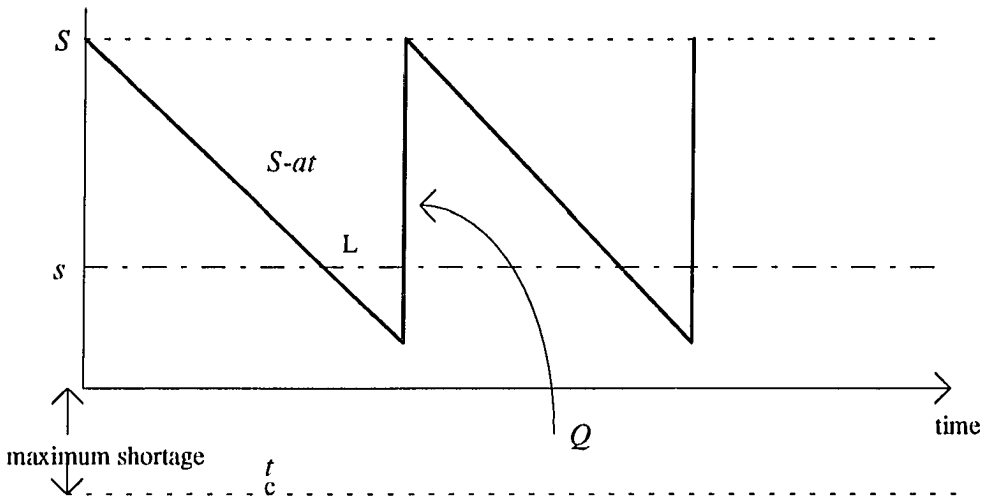

Figure 26. Diagram of Submodel II.

(a) cycle time:

$$t_c = \frac{S}{a} + L - \frac{s}{a(1-p_r)} \quad (p_r > 0),$$

(b) order quantity:

$$Q = S - s + La(1-p_r),$$

(c) holding cost:

$$\frac{S^2 h}{2at_c},$$

(d) backlog cost:

$$\frac{(-s+La(1-p_r)}{2t_c}\left|L-\frac{s}{a(1-p_r)}\right|p \quad (p_r>0).$$

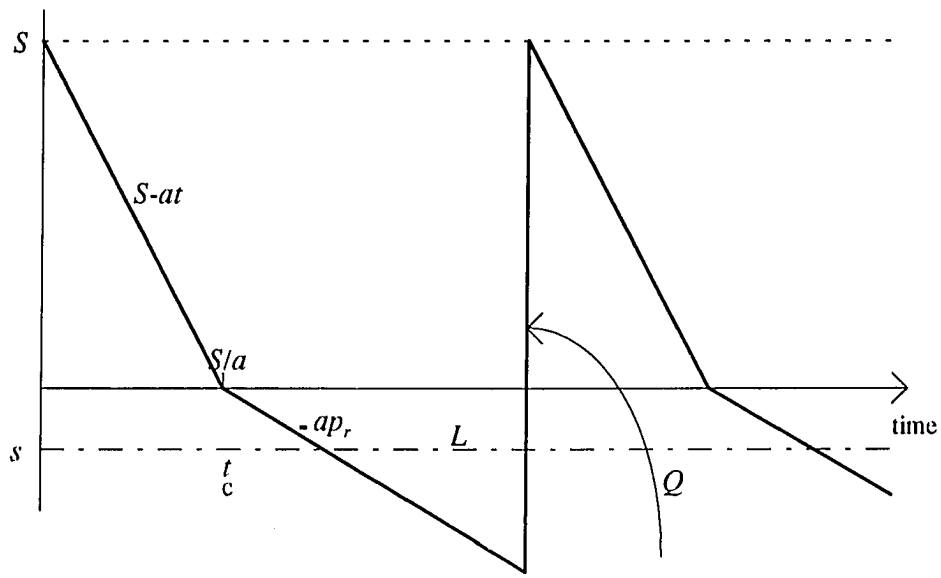

Figure 27. Diagram of Submodel III.

Validation of the Results Using the Deterministic Model

In order to validate the optimization results obtained through RSM, the deterministic model was tested under the same inventory policies. For all different inventory policies tested, very close results to those of the simulation model were obtained, which confirmed the optimization results strongly.

Using the deterministic formulas developed for the three cases, the theoritical response surface was plotted within reasonably large interval of S (400-2000), and s (-200,1500). As seen in Figure 28 the surface is convex indicating that there is no

local optimum in the search region. In the contour plot of the surface shown in Figure 29, a steady increase in the average profit toward the optimum point can be observed easliy. The optimum solution maximizing the average profit ontained using the determistic model (i.e., S= 1383, s = 242) was very close to the optimum solution obtained from RSM (i.e., S=1389, s =240).

As a conclusion, the proposed determistic model confirmed the success of RSM in locating the optimum solution with great accuracy.

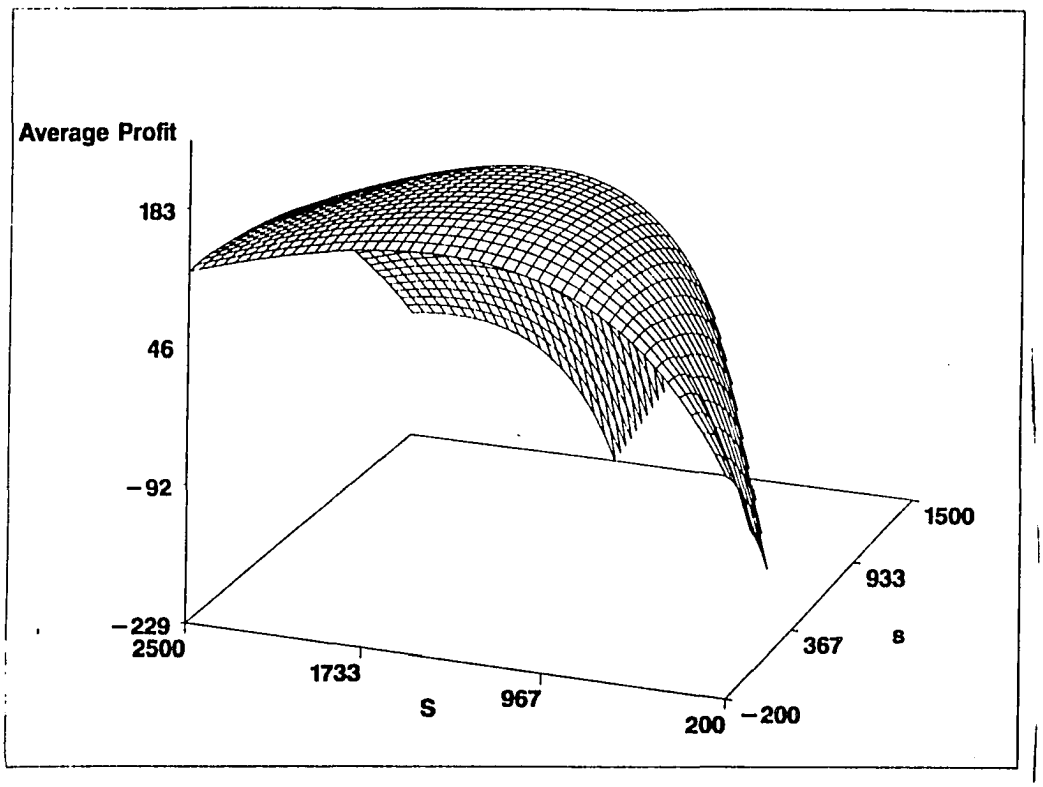

Figure 28. Response Surface of Average Profit as a Function of Reorder Level, and Reorder Point Based on the Deterministic Model Data.

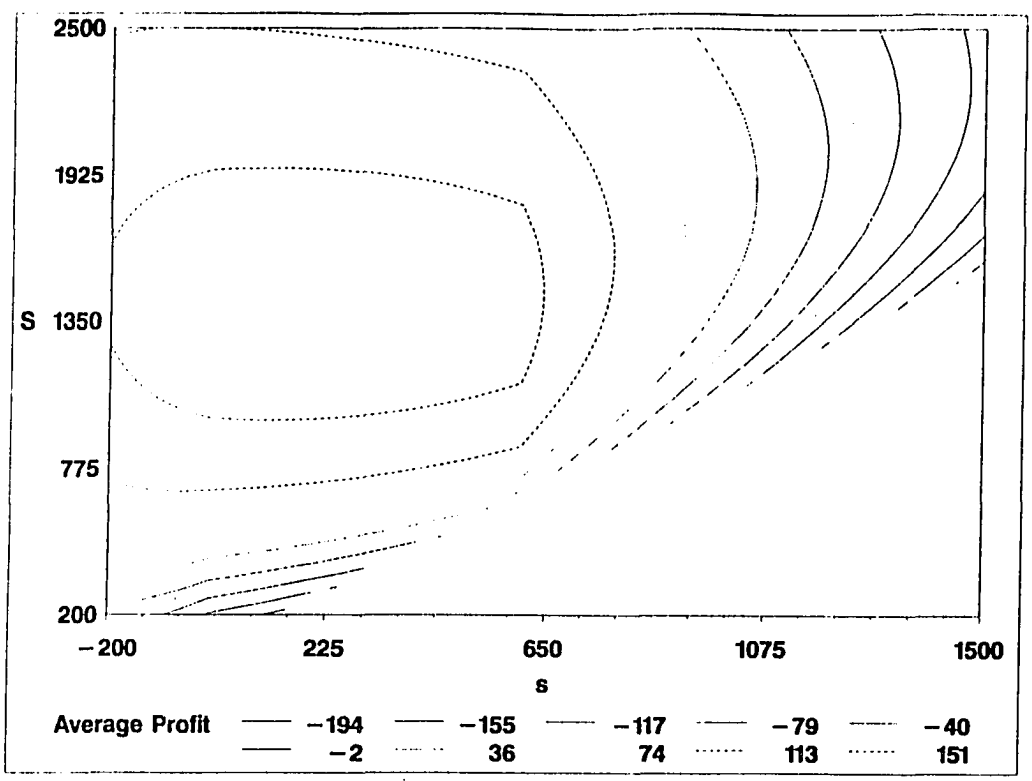

Figure 29. Contour Plot of Average Profit as a Function of Reorder Level, and Reorder Point Based on the Deterministic Model Data.

CHAPTER VII

FUTURE RESEARCH SUGGESTIONS

The present system can be considered as first step implementations of a greater system which can improve the simulation modeling in more diverse domains, and more automated way in the future. The potential extensions to the present study can be done in two aspects:

1. Improvements in simulation model development: At present, only inventory systems can be modeled using the current computer program. However, we have already designed object-oriented classes which can be used for more application domains. Therefore, addition of new applications domains such as flexible manufacturing systems, and job-shop production can be carried out a little more effort due to the ability to reuse and extend the previously defined classes in object-oriented programming.

2. Improvements in optimization process: As optimization in simulation using RSM requires considerable amount of time, and statistical expertise, the first priority should be given to the automation of this method. In this regard, integration of the present system with a statistical software is required so that statistical calculations such as model building, and performing hypothesis tests can be carried out internally during the optimization process without the user involvement.

Second improvement can be for multiple-item inventory problems which have significant demand flow from one product to another through substitution. In this case finding individual optimum policies for products would be obsolete as this approach ignores the significance of the interaction between policy parameters. Group screening

methods can be an alternative to cut down the number of runs by grouping the parameters which have similar effects. However, one should note that if there are too many products considered, then even group screening methods may be inefficient easily due to the unmanageable number of factors.

CHAPTER VIII

CONCLUSIONS

In this study, an expert computer system has been developed to automate some aspects of simulation modeling, which were model building, automatic simulation program generation in SLAM II, and experimental design. Subsequently, response surface methodology was employed to optimize the simulation output, i.e. average profit of the probabilistic continuous review inventory problem. Optimization results were validated through the proposed deterministic approximation (relaxation) of the original probabilistic model. In conclusion, the following results have been reached in the two following aspects of this study:

1. Simulation model development using the expert system.

(a) Potential benefits of the present system to the user can be expressed in terms of time, reliability, and correctness of simulation models. It is expected that the present system can boost the efficiency of both experienced, and inexperienced simulation modelers. People with very little simulation background can benefit from the system without requiring the knowledge of a simulation language. Advanced user-interface, multi-tasking features of Windows operating system had great contribution to creating rapid prototypes. A number of design alternatives for a inventory system can be tested within a short time by changing model description file, and generating the corresponding SLAM II code automatically.

(b) Advantages of object-oriented programming in the development of the software were tremendous. Most prominent of them were data abstraction through encapsulation, and class prototypes, ability to reuse and extent earlier classes. One

potential advantage of using OOP for future extensions to the present system is that addition of new application domains will require less effort as compared to the procedural programming by taking advantage of modularity, and extendibility of the present system. Object-oriented programming was able to represent the taxonomy of SLAM II in modular, and natural way through the classes organized in a top-down hierarchy. Future application domains will be able to use those classes without any change.

2. Optimization using RSM:

(a) RSM showed a great performance as optimization tool. The results of RSM for the example inventory problem were very close to those obtained from the proposed deterministic model. This can be attributed to the consistency of RSM for a given optimization problem due to its well-defined statistical, and mathematical foundations. RSM provides a unified approach that can be applied to any type of optimization problem in simulation studies. However, one should note that its performance heavily depends on the location of starting point, characteristics of the response surface (i.e., existence of local optima), and the size of error variance. In case of local optima, RSM should be performed with different starting points so that the chance of missing the global optimum can be decreased.

(b) Use of the deterministic models related to a problem domain can help in locating a good starting point which usually results in fewer runs. In this study the simple EOQ model was very successful in determining a good starting point. Another important use of the deterministic model was validation of the optimization results.

Appendix A

Model Description File of the Example

The following is the list of the model description file for the example discussed in Chapter VI.

```
[problem]
 $Problem      Model='MULTI-PRODUCT',
               Modeler = 'RIZVAN',
               Date    = '7/18/92',
               OrderSep     ='Y',
               nOfProducts  = 3,
               Product(1)   ='ITEM1', InvType(1) = 1,
               Product(2)   ='ITEM2', InvType(2) = 2,
               Product(3)   ='ITEM3', InvType(3) = 3$

[experimental_design]

 $Design       DesignType = 1,
               nOfReplications=1,
               nOfCells=1,
               SimulationLength=3500$

 $Orthogonal   Option = 1,
               nOfSSLevels  = 1,
               nOfsLevels   = 1,
               nOftLevels   = 1,
               nOfrateLevels = 1 $

 $FactorLevels SSValues(1,1) = 290,
               sValues(1,1)  = 132,
               SSValues(1,2) = 1224,
               sValues(1,2)  = 250,
               tValues(1,2)  = 9,
               SSValues(1,3) = 300,
               sValues(1,3)  = 200,
               rateValues(1,3) = 65$
 $RegVariables
               nOfVariables(1)  = 4,
               RegPrintVar(1,1) = 'SCL(1)',        Index(1,1)=1,
               RegPrintVar(1,2) = 'RPT(1)',        Index(1,2)=2,
               RegPrintVar(1,3) = 'N_SETUP',       Index(1,3)=3,
               RegPrintVar(1,4) = 'AVE_COST',      Index(1,4)=4,
               nOfVariables(2) = 2,
```

```
                RegPrintVar(2,1)= 'SCL(2)',          Index(2,1)=1,
                                    (Continued)

                RegPrintVar(2,2)= 'AVE_COST',        Index(2,2)=2,
                nOfVariables(3) = 0 $


[substitution]
 $SubstitutionMatrix
                IsThereSubstitution = 'Y',
                pro(1,2)=0.0,  pro(1,3)=1.0,
                pro(2,1)=0.0,  pro(2,3)=0.0,
                pro(3,1)=1.0,  pro(3,2)=0.0,
                qcf(1,2)=1.0,  qcf(1,3)=1.0,
                qcf(2,1)=1.0,  qcf(2,3)=1.0,
                qcf(3,1)=1.0,  qcf(3,2)=1.0,
                SelectionRule = 1 $

[options]
 $Options       PrintData      ='Y',
                PrintReg       ='Y',
                PrintFull      ='Y',
                PrintSlam      ='Y'$

 $Monitor       UseMonitor     ='Y',
                MonitorOp      ='TRACE',
                FromTime = 0.0,ToTime=15.0,
                variable(1)='INV_POS',
                variable(2)='NNQ(1)',
                variable(3)='AMOUNT'$


[products]

[ITEM1]
[distributions]

 $Arrivals      Distribution='exponential',    Parameter=0.5$
 $Demand        Distribution='user defined',
                   nOfValues = 3,
                   x(1)=10,px(1)=0.2,
                   x(2)=20,px(2)=0.4,
                   x(3)=30,px(3)=0.4$
```

```
 $Leadtime    Distribution='constant',      Parameter=3.0$
 $Processing  Distribution='constant',      Parameter=0.0$

[cost]
 $COST        SetupCost   =150,
              HoldingCost =0.1,
              BacklogCost =0.2,
              LostSaleCost=2.5,
              ReviewCost  =0.01$

[pricebreaks]
 $Purchase    nOfPurchaseBreaks = 3$
 $PurchaseBreaks
              From(1)=0,    To(1)=600,    Price(1)=12.0,
              From(2)=601,  To(2)=1000,   Price(2)=10.0,
              From(3)=1001,To(3)=3000,    Price(3)=7.0 $
 $Sales       nOfSalesBreaks = 3$
 $SalesBreaks
              From(1)=0,    To(1)=10,     Price(1)=19.0,
              From(2)=11,   To(2)=25,     Price(2)=17.0,
              From(3)=25,   To(3)=1000,   Price(3)=16.0$
[backorder]
 $BackOrder   Case=2,       RejectPoint=50$
[END]

[ITEM2]
[distributions]
 $Arrivals    Distribution='constant',      Parameter=0.2$
 $Demand      Distribution='normal',        Parameter=25.0,5.0$
 $LeadTime    Distribution='constant',      Parameter=2.0$
 $Processing  Distribution='constant',      Parameter=0.0$

[cost]
 $Cost        SetupCost    =300,
              HoldingCost  =0.05,
              BacklogCost  =0.08,
              LostSaleCost =0.4,
              ReviewCost   =3.6$

[priceBreaks]
 $Purchase    nOfPurchaseBreaks=0,          UnitPrice = 3.0$
 $Sales       nOfSalesBreaks=0,             UnitPrice = 5.2$
```

```
[backorder]
 $BackOrder  Case=1,        Probability=0.5$
[END]

[ITEM3]
[distributions]
 $Arrivals     Distribution='exponential',   Parameter=0.3$
 $Demand       Distribution='normal',        Parameter=15.0,2.5$
 $LeadTime     Distribution='uniform'        Parameter=2.0,3.0$
 $Processing   Distribution='constant',      Parameter=0.0$

[cost]
 $Cost         SetupCost  =120,
               HoldingCost =0.04,
               BacklogCost =0.09,
               LostSaleCost=0.9,
               ReviewCost  =0.05$

[priceBreaks]
 $Purchase     nOfPurchaseBreaks=0,        UnitPrice = 2.5$
 $Sales        nOfSalesBreaks =0,          UnitPrice = 3.6$
[backorder]
 $BackOrder  Case=3, time1=2.0, time2=7.0$
[END]
[EndOfModel]
```

Appendix B

Description of the Variables Used in Model Description File

Block Problem

Model : simulation project name.

Modeler : modeler name.

Date : project date, MM/DD/YY.

nOfProducts : number of products in inventory system.

Product(i) : name of *i*th product.

InvType(i) : type of inventory control of *i*th product, the following codes are used:

(1) continuous review.

(2) periodic review.

(3) production problem with continuous review.

Block Design

DesignType : type of experimental design used, the following codes are used:

(1) orthogonal design.

(2) central composite design.

(3) incremental design.

nOfReplications : number of replications (n) made in experimental design.

nOfCells : number of rows in design matrix.

SimulationLength : length of the simulation run in unit of time.

Block Orthogonal

Option : experimental design option, the following codes are used:

(1) contruct a common experimental design for all products.

(2) seperate design for each product.

nOfSSLevels: number of levels for (S).

nOfsLevels: number of levels for (s).

nOftLevels: number of levels for (t).

nOfrateLevels : number of levels for production rate.

Block Incremental

nOfIncrements : number of increments (iterations) used.

SSStart(i) : starting point of (S) for ith product.

SSIncrement(i) : increment in (S) for ith product.

sStart(i): starting point of (s) for ith product.

sIncrement(i) : increment in (s) for ith product.

tStart(i) : starting point of (t) for ith product.

tIncrement(i) : increment i (t) for ith product.

rateStart(i): starting point of production rate for ith product.

rateIncrement(i): increment in production rate for ith product.

Block FactorLevels

SSValues(i): S values for ith product.

sValues(i): s values for ith product.

tValues(i): t values for ith product.

rateValues(i): production rate values for ith product.

Block RegVariables

nOfVariables(i) : number of variables that will be used in regression for product$_i$.

RegPrintvar(i,j) : name of jth regression variable ith for product.

VarIndexes(i,j) : index number of jth regression variable for ith product.

Block SubstitutionMatrix

UseSubstitution: "Y" : there is substituion, "N" : no substitution.

pro(i,j): the probability of substituting ith product by jth product.

qcf(i,j): amount multiplier between ith product and jth product.

SelectionRule: selection criterion to select product to substitute.

(1) expected waiting time,

(2) total price.

Block Distribution for standard distributions

distribution: name of distribution.

parameter(i) : *i*th parameter of the selected distribution.

Block Distribution for user-defined empirical distributions

x(i): x value

p(x): probability of x, $p(x)$.

Block Cost

SetupCost : setup cost, $ per setup.

HoldingCost: holding cost, $ per item per unit of time.

BacklogCost: backlog cost, $ per item per unit of time.

LostSaleCost: cancellation cost, $ per item.

ReviewCost: cost of reviewing inventory level, $ per review.

Block PurchasePrice/SellingPrice

nOfBreaks: number of price breaks.

UnitPrice: price per unit.

Block PurchaseBreaks/SellingBreaks

From(i): lower limit of *i*th break in units.

To(i): upper limit of *i*th break in units.

Price(i): unit price within *i*th break.

Block Backlog

Case: backlog policy type. Customer may cancel his order when it is backlogged depending upon:

(1) a constant probability.

(2) number of units backlogged.

(3) expected waiting time.

probability: probability of cancelling order.

RejectPoint: reject point for number of units backlogged.

Time1: lower limit of expected waiting time.

Time2: upper limit of expected waiting time.

Appendix C

SLAM II Code of the Example Generated by
the Program Generator

The following is the list of SLAM II code generated by the program generator for the example given in Chapter VI.

```
GEN,RIZVAN,INVENTORYMODEL,1/12/92,1,NO,YES,YES/YES,YES,YES/F,13
2;
    LIMITS,4,9,100;
    ARRAY(1,3)/2,1,3;Backorder cases (CASE)
    ARRAY(2,3)/0,0.8,0;Probability of cancellation (P)
    ARRAY(3,3)/100,0,0;Threashold value(units backordered)for balking(CUT)
    ARRAY(4,3)/1,2,3;Type of inventory control (INV_TYPE)
    ARRAY(5,3);Current Inventory Level(INV_POS)
    ARRAY(6,3);number of customer orders (N_ORDERS)
    ARRAY(7,3);number of setups (N_SETUPS)
    ARRAY(8,3);number of lost sales (N_LOST)
    ARRAY(9,3);number of customer orders satisfied (N_SATISFIED)
    ARRAY(10,3);amount of lost sales[units] (TOTAL_LOST)
    ARRAY(11,3);maximum inventory level (SCL)
    ARRAY(12,3);reorder point (RPT)
    ARRAY(13,3);review period (PERIOD)
    ARRAY(14,3);production rate (RATE)
    ARRAY(15,3);whether or not place new order (CAN_ORDER)
    ARRAY(16,3);time that the last order is placed
    ARRAY(17,3);Earliest time that product is available
    ARRAY(18,3);Order quantity (ORDER_QT)
    ARRAY(19,3);Number of reviews (N_REVIEW)
    ARRAY(20,3);Last time of customer balking
    ARRAY(21,3);Satisfied Quantity (SATISFIED_QT)
    ARRAY(22,3)/3.464967e-22,3.464967e-22,3.464967e-22;Current Backlog Level
    EQUIVALENCE/XX(1),INDEX;
    EQUIVALENCE/ATRIB(1),ARRIVAL;
    EQUIVALENCE/ATRIB(2),AMOUNT;
    EQUIVALENCE/ATRIB(3),PROD_NO;
    EQUIVALENCE/ATRIB(4),EXTWAIT;
    EQUIVALENCE/ATRIB(5),SUB_AMOUNT;
    EQUIVALENCE/ATRIB(6),CRITERIA;
    EQUIVALENCE/ATRIB(7),SUB_PRNO;
    EQUIVALENCE/ATRIB(8),REMAIN;
    EQUIVALENCE/USERF(6),NEXT;
    EQUIVALENCE/USERF(7),PRO;
    EQUIVALENCE/USERF(8),COEF;
```

```
EQUIVALENCE/USERF(9),GETPRO;
EQUIVALENCE/USERF(10),GETTIME;
EQUIVALENCE/ARRAY(1,PROD_NO),CASE;
EQUIVALENCE/ARRAY(2,PROD_NO),P;
EQUIVALENCE/ARRAY(3,PROD_NO),CUT;
EQUIVALENCE/ARRAY(1,SUB_PRNO),SUB_CASE;
EQUIVALENCE/ARRAY(2,SUB_PRNO),SUB_P;
EQUIVALENCE/ARRAY(3,SUB_PRNO),SUB_CUT;
EQUIVALENCE/ARRAY(4,PROD_NO),INV_TYPE;
EQUIVALENCE/ARRAY(4,SUB_PRNO),SUB_INVTYPE;
EQUIVALENCE/ARRAY(5,PROD_NO),INV_POS;
EQUIVALENCE/ARRAY(5,SUB_PRNO),SUB_INV_POS;
EQUIVALENCE/ARRAY(6,PROD_NO),N_ORDERS;
EQUIVALENCE/ARRAY(7,PROD_NO),N_SETUP;
EQUIVALENCE/ARRAY(8,PROD_NO),N_LOST;
EQUIVALENCE/ARRAY(9,PROD_NO),N_SATISFIED;
EQUIVALENCE/ARRAY(10,PROD_NO),TOTAL_LOST;
EQUIVALENCE/ARRAY(11,PROD_NO),SCL;
EQUIVALENCE/ARRAY(12,PROD_NO),RPT;
EQUIVALENCE/ARRAY(13,PROD_NO),PERIOD;
EQUIVALENCE/ARRAY(14,PROD_NO),RATE;
EQUIVALENCE/ARRAY(15,PROD_NO),CAN_ORDER;
EQUIVALENCE/ARRAY(16,PROD_NO),LAST_ORDER_TIME;
EQUIVALENCE/ARRAY(17,SUB_PRNO),SUB_AVAILABLE_TIME;
EQUIVALENCE/ARRAY(17,PROD_NO),AVAILABLE_TIME;
EQUIVALENCE/ARRAY(18,PROD_NO),ORDER_QT;
EQUIVALENCE/ARRAY(21,PROD_NO),SATISFIED_QT;
EQUIVALENCE/ARRAY(19,PROD_NO),NREVIEW;
EQUIVALENCE/USERF(2),DEMAND_SIZE;
EQUIVALENCE/USERF(3),LEAD_TIME;
EQUIVALENCE/USERF(5),QUANTITY;
EQUIVALENCE/USERF(4),PROCESSING;
EQUIVALENCE/XX(2),PRONOTEMP;
EQUIVALENCE/XX(3),AMOUNTTEMP;
EQUIVALENCE/ARRAY(22,PROD_NO),BACKLOG;
EQUIVALENCE/ARRAY(22,1),BACKLOG1;
EQUIVALENCE/ARRAY(22,2),BACKLOG2;
EQUIVALENCE/ARRAY(22,3),BACKLOG3;
EQUIVALENCE/USERF(101),ARRIVAL1;
EQUIVALENCE/USERF(102),ARRIVAL2;
EQUIVALENCE/USERF(103),ARRIVAL3;
EQUIVALENCE/ARRAY(13,2),PERIOD2;
```

```
;    Time-persistent variables
     TIMST,BACKLOG1,BACKLOG LEVEL1
     TIMST,BACKLOG2,BACKLOG LEVEL2
     TIMST,BACKLOG3,BACKLOG LEVEL3

;    Statistics based on observations
     STAT,1,SAFETY STOCK1
     STAT,2,SAFETY STOCK2
     STAT,3,SAFETY STOCK3
     STAT,4,TIME BET. CANCEL1
     STAT,5,TIME BET. CANCEL2
     STAT,6,TIME BET. CANCEL3
     STAT,7,CYCLE LENGTH1
     STAT,8,CYCLE LENGTH2
     STAT,9,CYCLE LENGTH3

;    Beginning of SLAM II Network
     NETWORK;

;       ResourceBlocks
        RESOURCE/1,ITEM1(100),1;
        RESOURCE/2,ITEM2(100),2;
        RESOURCE/3,ITEM3(100),3;
        RESOURCE/4,SERVER(3),4;

        CREATE,ARRIVAL1,0,1,,;Create Customers for Product 1
        ASSIGN,PROD_NO=1;
        ACT,,,CONT;
        CREATE,ARRIVAL2,0,2,,;Create Customers for Product 2
        ASSIGN,PROD_NO=2;
        ACT,,,CONT;
        CREATE,ARRIVAL3,0,3,,;Create Customers for Product 3
        ASSIGN,PROD_NO=3;
        ACT,,,CONT;

CONT    ASSIGN,AMOUNT=DEMAND_SIZE;
        ASSIGN,N_ORDERS = N_ORDERS + 1;
        EVENT(1);   Update demand statistics
        ASSIGN,II=PROD_NO;
        GOON,1;
        ACT,,NNRSC(II).EQ.0,OTHR;
        ACT,,NNRSC(II).GT.0,KEEP;
```

```
;       Demand will be satisfied
KEEP    ASSIGN,N_SATISFIED=N_SATISFIED+1;
        ASSIGN,SATISFIED_QT=SATISFIED_QT+AMOUNT;
        EVENT(3);       Update statistics on revenues
        GOON,2;
        ACT,,,UPDT;
        ACT,,,SEPR;

QUE1    ASSIGN,BACKLOG=BACKLOG+AMOUNT;
        AWAIT(PROD_NO = 1,3),PROD_NO/AMOUNT;
        ASSIGN,BACKLOG=BACKLOG-AMOUNT;
        AWAIT(4),SERVER/1;
        ACT,PROCESSING,,;
        FREE,SERVER/1;
        TERM;

UPDT    ASSIGN,INV_POS = INV_POS - AMOUNT;
        GOON,1;
        ACT,,INV_TYPE.EQ.1.OR.INV_TYPE.EQ.3,CHOD;
        ACT,,,;
        TERM;

CHOD    ASSIGN,NREVIEW=NREVIEW+1;
        GOON,1;
        ACT,,INV_POS.LE.RPT.AND.CAN_ORDER.EQ.1,PTOD;
        ACT,,INV_POS.GT.RPT.OR.CAN_ORDER.EQ.0,;
        TERM;

PTOD    GOON,1;
        ACT,,INV_TYPE.EQ.3,PROD;
        ACT,,INV_TYPE.EQ.1,BULK;

;   Split the demand into two orders
SEPR    ASSIGN,II=PROD_NO;
        GOON,1;
        ACT,,NNRSC(II).GT.0.AND.AMOUNT.GT.NNRSC(II),SPOK;
        ACT,,NNRSC(II).LE.0.OR.AMOUNT.LE.NNRSC(II),QUE1;

;   Separation of the order
SPOK    ASSIGN,II = PROD_NO;
        ASSIGN,REMAIN = AMOUNT - NNRSC(II);
        ASSIGN,AMOUNT = NNRSC(II);
```

```
        ACT,,,QUE1;
        ACT,0.0000001,,QUE2;

QUE2    ASSIGN,BACKLOG=BACKLOG+REMAIN;
        AWAIT(PROD_NO=1,3),PROD_NO/AMOUNT;
        ASSIGN,BACKLOG=BACKLOG-REMAIN;
        AWAIT(4),SERVER/1;
        ACT,PROCESSING,,;
        FREE,SERVER/1;
        TERM;

;   Handling Backlog Cases
CHBL    GOON,1;
        ACT,,CASE.EQ.1,CAS1;
        ACT,,CASE.EQ.2,CAS2;
        ACT,,CASE.EQ.3,CAS3;

CAS1    GOON,1;
        ACT,,P,LOSE;
        ACT,,1-P,KEEP;

CAS2    GOON,1;
        ACT,,-1*INV_POS.GE.CUT,LOSE;
        ACT,,-1*INV_POS.LT.CUT,KEEP;

CAS3    ASSIGN,PRONOTEMP = PROD_NO;
        ASSIGN,AMOUNTTEMP = AMOUNT;
        ASSIGN,EXTWAIT = GETTIME;
        ASSIGN,P = GETPRO;
        ACT,,,CAS1;

;   Try to substitute with another product
OTHR    GOON,2;
        ACT,,,PR1;
        ACT,,,PR2;

PR1     ASSIGN,INDEX = 1;
        ACT,,,GENR;

PR2     ASSIGN,INDEX = 2;
        ACT,,,GENR;

GENR    ASSIGN,SUB_PRNO = NEXT;
```

```
        ASSIGN,PRONOTEMP = PROD_NO;
        ASSIGN,AMOUNTTEMP = AMOUNT;
        ASSIGN,AVAILABLE_TIME = GETTIME;
        ASSIGN,PRONOTEMP = SUB_PRNO;
        ASSIGN,SUB_AMOUNT = AMOUNT*COEF;
        ASSIGN,AMOUNTTEMP = SUB_AMOUNT;
        ASSIGN,SUB_AVAILABLE_TIME = GETTIME;
        GOON,1;
        ACT,,AVAILABLE_TIME.GT.SUB_AVAILABLE_TIME,CHK1;
        ACT,,,A1;

;   check whether customer wants to substitute
CHK1  GOON,1;
        ACT,,PRO,A2;
        ACT,,1-PRO,A1;

;Assignment of available time as criterion

A1      ASSIGN,CRITERIA = 10.0E20;
        ACT,,,ACCU;

A2      ASSIGN,CRITERIA = SUB_AVAILABLE_TIME;
        ACT,,,ACCU;

;   Selection of the product for substitution
ACCU  ACCUMULATE,2,2,LOW(7),1;
        ACT,,CRITERIA.EQ.10.0E20,CHBL;No Substitution
        ACT,,CRITERIA.NE.10.0E20,CHAV;Check availability

;Checking the availability of the selected product for substitution
CHAV  ASSIGN,II=SUB_PRNO;
        GOON,1;
        ACT,,NNRSC(II).GE.SUB_AMOUNT,ACSU;Accept Substitution
        ACT,,NNRSC(II).LT.SUB_AMOUNT,CHSB;

; Handle Backlog Situation
CHSB  GOON,1;
        ACT,,SUB_CASE.EQ.1,SCA1;
        ACT,,SUB_CASE.EQ.2,SCA2;
        ACT,,SUB_CASE.EQ.3,SCA3;

SCA1  GOON,1;
        ACT,,SUB_P,CHBL;
```

```
        ACT,,1-SUB_P,ACSU;

SCA2    GOON,1;
        ACT,,-1*INV_POS.GE.SUB_CUT,CHBL;
        ACT,,-1*INV_POS.LT.SUB_CUT,ACSU;

SCA3    ASSIGN,PRONOTEMP = SUB_PRNO;
        ASSIGN,AMOUNTTEMP = SUB_AMOUNT;
        ASSIGN,EXTWAIT = GETTIME;
        ASSIGN,SUB_P = GETPRO;
        ACT,,,SCA1;

;       Accept Substitution
ACSU    EVENT(4);  Update statistcs on substitution
        ASSIGN,PROD_NO=SUB_PRNO;
        ASSIGN,AMOUNT=SUB_AMOUNT;
        ACT,,,KEEP;

;  Customer cancels the order
LOSE    ASSIGN,N_LOST = N_LOST + 1;
        ASSIGN,TOTAL_LOST = TOTAL_LOST + AMOUNT;
        EVENT(6);    Collect stat. on time between cancels
        TERM;

        CREATE,PERIOD2,0,,,;Review Inventory Level of Product : ITEM2
        ASSIGN,PROD_NO=2,
             NREVIEW=NREVIEW+1;
        ACT,,,CPER;

;       Periodic Review for ITEM2
CPER    GOON,1;Decide to a new setup
        ACT,,INV_POS.LE.RPT,ORD;Place a new order
        ACT,,INV_POS.GT.RPT,TERM;Do not place any order
ORD     ASSIGN,ORDER_QT = QUANTITY,
             N_SETUP = N_SETUP + 1,
             LAST_ORDER_TIME = TNOW;
        EVENT(2);Calculate Purchase Cost
        ACT,LEAD_TIME,,;Lead time of new order arrival
        EVENT(8);    Collect stat. on cycle length
        EVENT(5);   Collect stat. on safety stock
        ALTER,ITEM2/ORDER_QT;
        ASSIGN,INV_POS = INV_POS + ORDER_QT;
TERM    TERM;
```

```
;       Start production, or supply
PROD  ASSIGN,N_SETUP=N_SETUP+1;
      ASSIGN,CAN_ORDER = 0;
      ASSIGN,LAST_ORDER_TIME = TNOW;
      ACT,LEAD_TIME,,;
      EVENT(8);  Collect stat. on cycle length
      EVENT(5);  Collect stat. on safety stock
CPRO  GOON,1;
      ACT,,INV_POS.GE.SCL,STOP;
      ACT,1,INV_POS.LT.SCL,;
      ALTER,PROD_NO/RATE;
      ASSIGN,INV_POS=INV_POS+RATE;
      ASSIGN,ORDER_QT = RATE;
      EVENT(2);  Update total purchase cost
      ACT,,,CPRO;
STOP  ASSIGN,CAN_ORDER = 1;
      TERM;

; (Continuous Review ) Place a new order  for ITEM1
BULK  ASSIGN,ORDER_QT = QUANTITY;
      ASSIGN,N_SETUP = N_SETUP + 1;
      ASSIGN,CAN_ORDER = 0;
      ASSIGN,LAST_ORDER_TIME = TNOW;
      EVENT(2);
      ACT,LEAD_TIME,,;
      EVENT(8);   Collect stat. on cycle length
      EVENT(5);   Collect stat. on safety stock
      ALTER,ITEM1/ORDER_QT;
      ASSIGN,INV_POS = INV_POS + ORDER_QT;
      ASSIGN,CAN_ORDER = 1;
      TERM;

   ENDNETWORK;
   INITIALIZE,0,720, YES/1,YES,YES;
   SEEDS,0(1)/YES,0(2)/YES;
   FIN;
```

Appendix D

User's Guide for the Integrated Simulation Environment

Computer requirements: The model development environment runs in the user-friendly Windows operating system. Windows 3.0, or Windows 3.1 must be installed on the PC prior to running the program. A Windows compatible mouse is suggested in order to take the advantage of the mouse-driven operation facility. At least 2M RAM memory is recommended to be able to run the program without any low in memory problem.

Basic features: The program has multiple a document interface (MDI) that allows the user to edit more than one file at once, and switch from one to another instantly. The program is hosted by a main window having top-down menu options. Appropriate dialog boxes are displayed at the user request to collect data interactively. By running multiple copies of the program the user can work on more than one project at the same time. Access to other programs in the system without quitting the generator is enabled through the operating system when needed.

Opening a new, or existing project file: Project in the program indicates a simulation project that consists of model description file, SLAM II code file, experimental design file, regression input file, and simulation summary report file. To start a new project:

1. Choose Project, and New Project options respectively from the menu.

2. A simulation model definition dialog box will be displayed to initiate the interactive data collection process.

3. Press the Define Problem button to display the dialog box to enter data.

4. Type first Modeler Name, Model Name, Date in the appropriate fields if you want to change the default values.

5. Type Item No, Item Name fields for each product in the problem along with checking the desired Inventory Control Type. Then, press Add button.

6. Press Ok to save data, or Cancel not to save changes.

File names associated with a project can be changed by selecting Options, File Locations from the menu that displays the corresponding dialog box (Figure 44). Type appropriate file names in the boxes, and press *Ok* button to validate the changes made.

Working on text files: From File menu select New to start new text file, and select *Open* to open an existing file. A file dialog box will appear to allow you to select a file among the list of files in the current directory. You might change the current directory by simply a double click on a particular directory name. You may cancel this process using the *Cancel* button. If a file is selected, the content of the file will be displayed on the screen. The program editor is fully equipped with cut, paste facility, searching specify text, and so on. Model description file may be edited directly through the program editor. After changes are made, the file can be saved by selecting *File*, and *Save* options in order from the menu. If file is new, you should supply a file name through the file dialog box.

Outlining problem: This step is to specify how many products exist in the system, and their corresponding inventory control mechanism. This step must be done first for new projects as other steps (i.e., buttons) are automatically disabled in the beginning. Pushing *Define Problem* button causes the corresponding dialog box to be displayed on the screen (Figure 33). Model Name, Modeler, Date fields should be typed in the indicated boxes. Product no, product name, and control mechanism are specified for each product. Using *Add* button adds the current product to the list. Press *Ok* to save changes.

Experimental design: There are three user dialog boxes to get data, and form a experimental design matrix for controllable factors, i.e., reorder level, reorder point,

and so on. After pressing *Experimental Design* button from the model definition dialog box, there are three options available to the user as design option in the first dialog box (Figure 34). Incremental design is a design in which the variable value is increased steadily by an increment. This option is for designs required to perform some runs along the steepest ascent. Separate, and Combined design options are for orthogonal design. When Separate Design is selected, individual design matrices are created for each product. On the other hand Combined Design option creates a single design matrix for all products resulting in greater number of runs since number of the factors is increased. Simulation length is also entered in the first design dialog box. To define design parameters, press *Go Experimental Design* button. One of the two different dialog boxes is displayed based on the user selection. These are:

1. Incremental design: In this dialog box (Figure 36) starting point, and increment for each factor, and an overall number of increments are acquired from the user. By pressing *Ok* button, corresponding design matrix is written to the experimental design file.

2. Orthogonal design: Full, or fractional factorial design, and central composite designs are constructed through the dialog box in Figure 35. Choose one of the design options, full, fractional, or central composite design. Supply factor levels for each inventory policy parameter. Finally, press *Ok* button, to create the design matrix.

Regression variables: This facility is to specify the name of the variables for regression analysis, eventually for the optimization of the output. Press *Regression Variables* button to display the dialog box in Figure 37. Select the variables that you want to write the regression analysis input file, and then press *Ok* button to save changes, *Cancel* otherwise.

Substitution matrix: This button is to specify the substitution matrix for a given problem. An initial dialog box will appear to ask whether there is substitution in the model. If you check the question box, then you may enter the substitution matrix using *Define Substitution* button, and display the dialog box in Figure 38. Enter probability of substitution, and amount multiplier between product pairs, and update the list using *Add* button. When you are done, press *Ok* to save changes, *Cancel* otherwise.

Specifying statistical distributions: You can assign statistical distributions to customer arrivals, demand size, and lead time. When you press any of the buttons to specify these parameters, a dialog box showing a list of available statistical distributions will appear. Choose a distribution, and specify the distribution parameters, as in the example dialog box in Figure 40. You may also specify, a user-defined empirical distribution when any of the well-known distribution functions does not fit your data well. Use the dialog box in Figure 41 to enter an empirical distribution.

Entering price breaks: As an option you may enter a single unit price for purchasing price, and selling price, or enter multiple price breaks for any of the items in the model. Use the following steps to enter price breaks:

1. Press purchasing price, or selling price button.

2. A dialog box will appear to ask you whether you have a single price, or price breaks. If you do not have price breaks, do not check *Price breaks* check box, and type your unit price, and press *Ok*.

3. If you check Price breaks, then press *Define Price Breaks* button, and enter price breaks using the dialog box in Figure 42.

Entering backlog policy: Each product in the model may have a different backlog policy. Specify a backlog policy for a product, press *Backlog policy* button. A dialog box will appear (Figure 43), you will have three options as backlog policy:

1. You may define a certain probability for customer renege when customer is backordered.

2. You may set an upper limit for the number of units backordered above which customers leave the system without being their orders are met.

3. You may set a lower, and upper limit for expected waiting time to determine the probability of customer renege.

Generating SLAM II code: Upon completion of data entry, you save your model description using the *Save* button on the *Model Definition* Box. To generate SLAM II code for the problem, select Simulation, and Generate from the menu. This will run the program generator automatically by displaying its window. Press *Generate* button to initiate code generation process. After completion of the code generation, The list of the SLAM II code will be displayed in the bottom window. You may browse the code using the scroller of the window, or loading the SLAM II file onto the program editor after exiting the program generator by pressing *Exit* button.

Appendix E

Terminology Used in
Object-Oriented Programming (OOP)

Function: Building blocks of C++ program within which all program activities occur. It has the same functionality as FORTRAN subroutines.

Class: Its declaration forms a new type that links functions and data. This new type is then used to declare objects of that class. When we say "Apple is a tree.", "tree" defines a class type in that sentence. Classes provide mechanisms for data abstraction, and information hiding. Every class in OOP has three sections describing the accessibility to its members from outside users (see Figure 29).

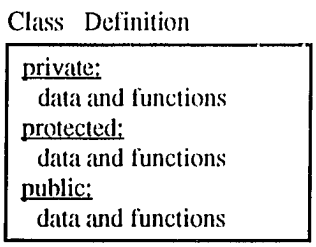

Figure 30. Prototype of a Class.

Private and protected members of a class are accessible to only its members while public members can be accessible to other parts of the program. Protected members are also available to inherited classes. Two types of classes used in OOP as follows:

1. Instance classes which can be instantiated to create usable objects. When all functions in a class are clearly described at least by default behavior, that class becomes an instance class.

2. Abstract classes serve as an umbrella for related classes. As such, it has few if any data members, and some or all of its member functions are pure virtual

functions. Pure virtual functions serve as a placeholder for the functions with the same name in classes derived from that class.

Object: Objects are instances of pre-defined classes. When they are created, they take up space in memory. What an object should do is described by its classes. "Apple is a tree." is just a short way of saying "Apple is an instance of class tree.". In OOP, objects communicate with each other via messages. When we send a message to an object, it performs internally particular operations designed for that message.

Inheritance: Inheritance is the process by which one class (i.e., derived class) can access the properties of another class (i.e., base class). It is an important mechanism in OOP because it supports the concept of classification and reuse of code created in base class. Knowledge and functions in the system become more manageable by using inheritance between related classes and putting them in a class hierarchy. Figure 2 shows two classes derived form a base class. Inheritance protocol is used to specify what members of a base class can be transferred to the derived class.

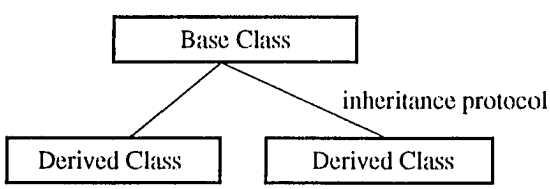

Figure 31. Class Hierarchy and Inheritance.

C++ allows multiple inheritance that a class can take more than one class as base class when it is inherited.

Encapsulation: It is the mechanism which prevents other parts of program from changing the members of a class. Placing data members and functions into private and protected parts of a class prevents any accidental change.

Polymorphism: It is characterized by the phrase "one interface, multiple implementation". In OOP one function name can be used for several related but slightly different purposes. In essence, polymorphism permits one interface for a general class of actions. Virtual functions, function and operator overloading are used to achieve polymorphism.

Virtual functions: A virtual function is defined as a function which can be overridden by its derived class versions. When a base class does not give any definition of a function but prototype of the same function, that function is said to be pure virtual function. In pure virtual function case, all classes derived from the base class must give their own definition of the pure virtual function. Calling the right version during run time is the responsibility of the compiler. Figure 3 is the illustration of the *take off* method as a virtual function in the derived classes of flying objects, i.e., airplane and helicopter.

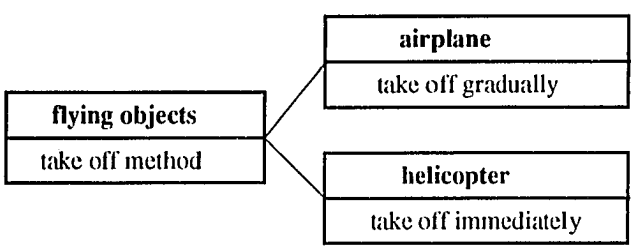

Figure 32. Describing the *take off* Method as a Virtual Function in the *FlyingObjects* Class.

Function and class templates: They are also called *generic types*, to construct a family of related functions or classes which can work with many types of objects. For instance, you may create a generic function to write any type of object to an external file. Generic classes and functions decrease the size of the program code and simplify the program.

Function and operator overloading: Two or more functions can share the same name as long as their argument lists are different. In this situation, functions that share the same name are said to be *overloaded*, and the process is referred to as *function overloading*.

By the same token, standard operators used in computer languages (e.g., =, +) can have different meaning relative to a specific class. Overloading mechanism is another way in OOP to deal with complexity and achieve polymorphism in large software systems.

Appendix F

Program Screens

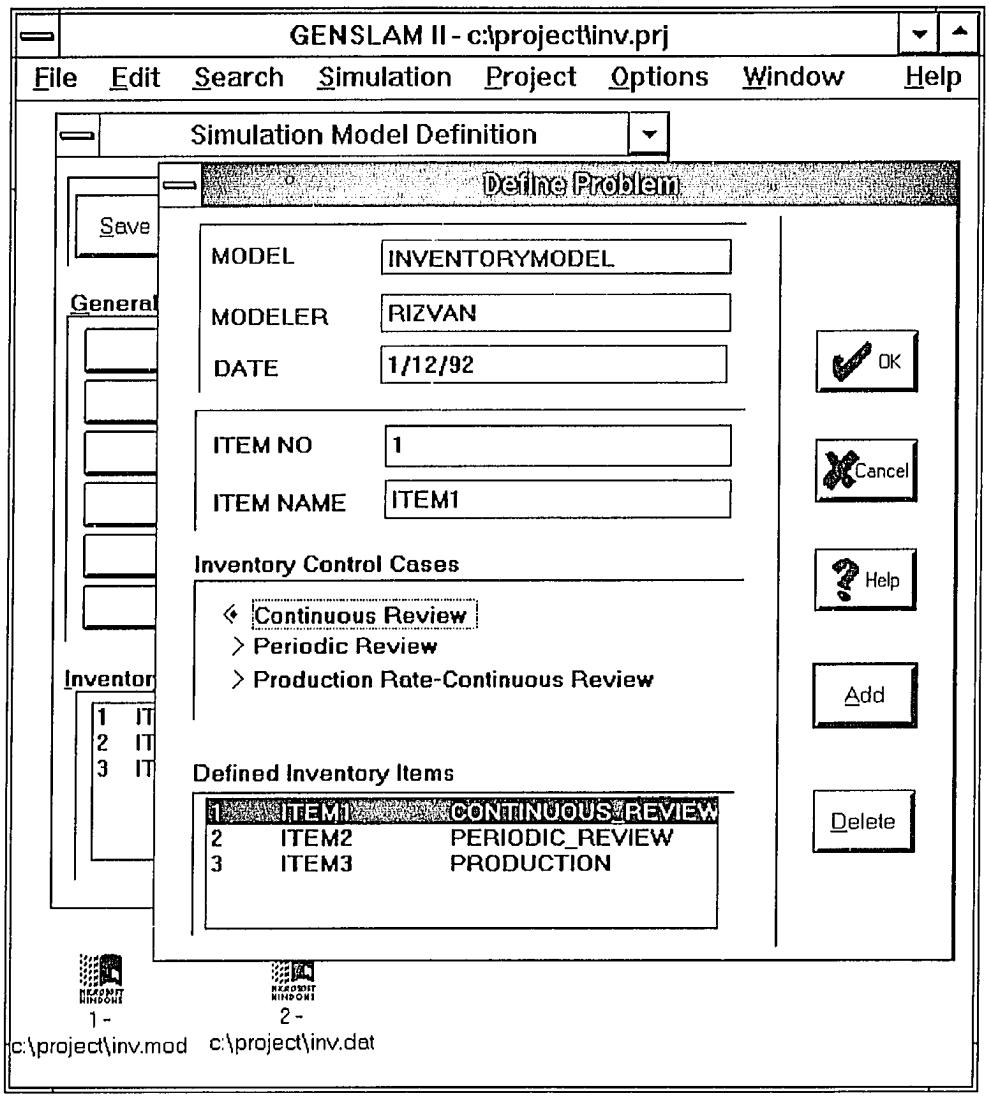

Figure 33. Dialog Box for Defining Control Mechanisms for Products.

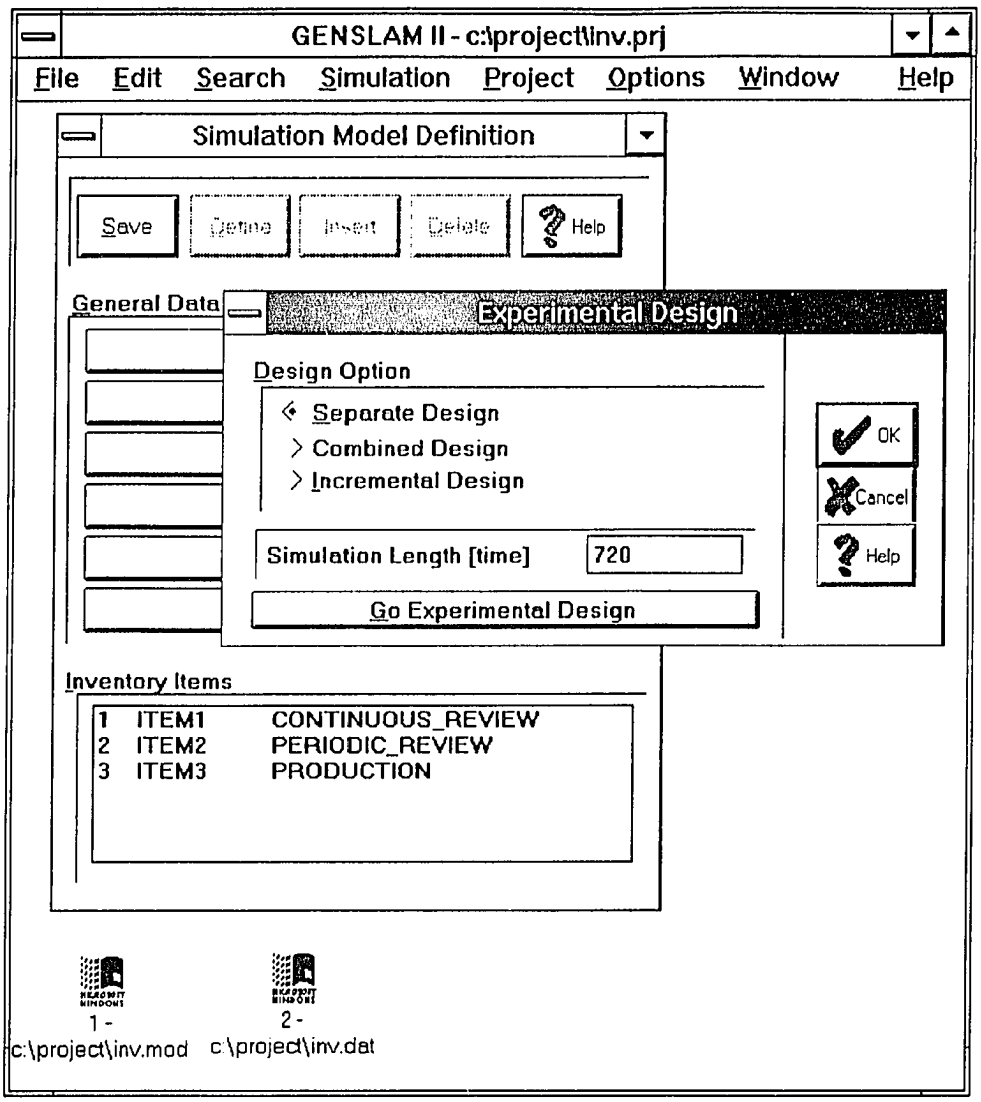

Figure 34. Dialog Box for Selecting Experimental Design Type.

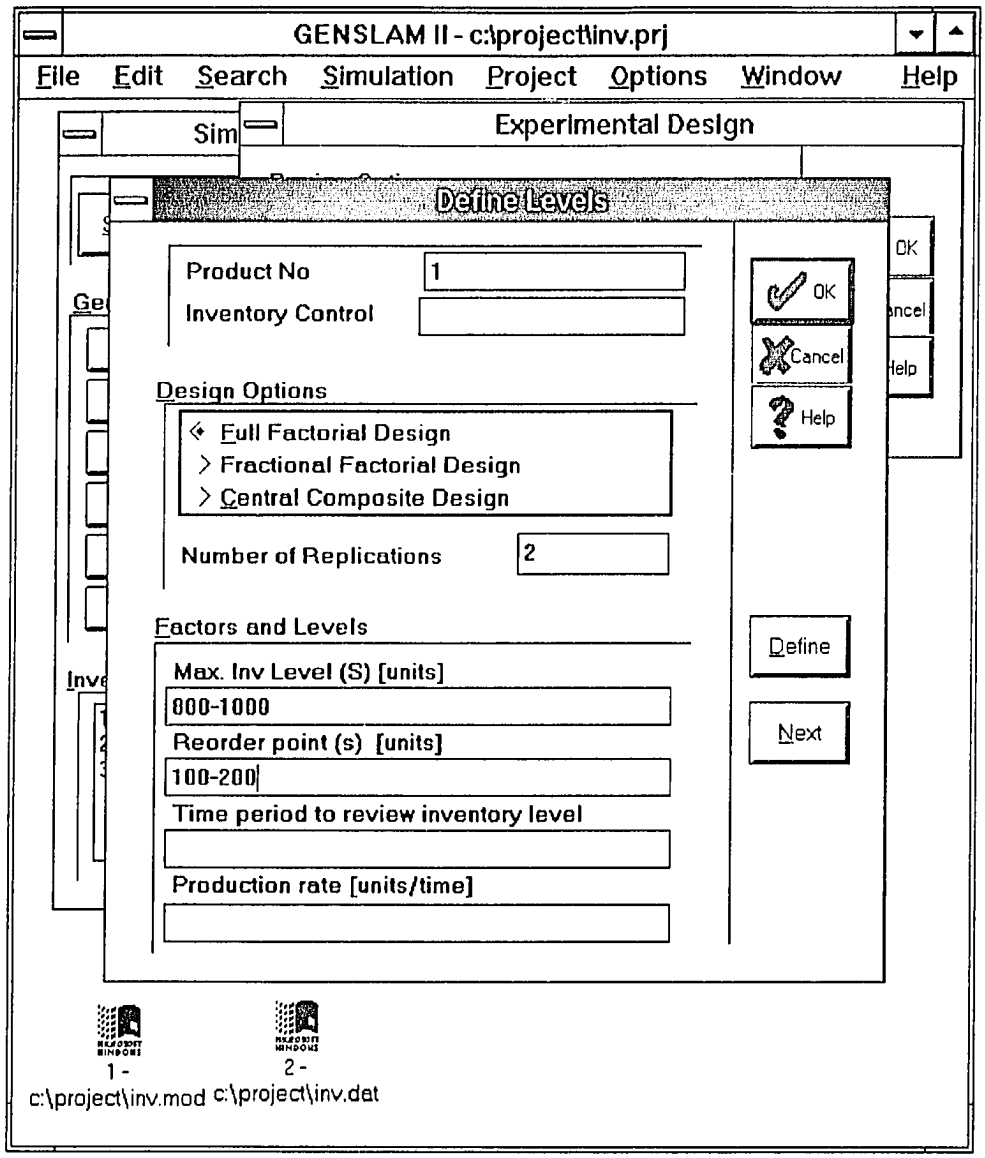

Figure 35. Dialog Box for Defining Parameters of Orthogonal Experimental Design.

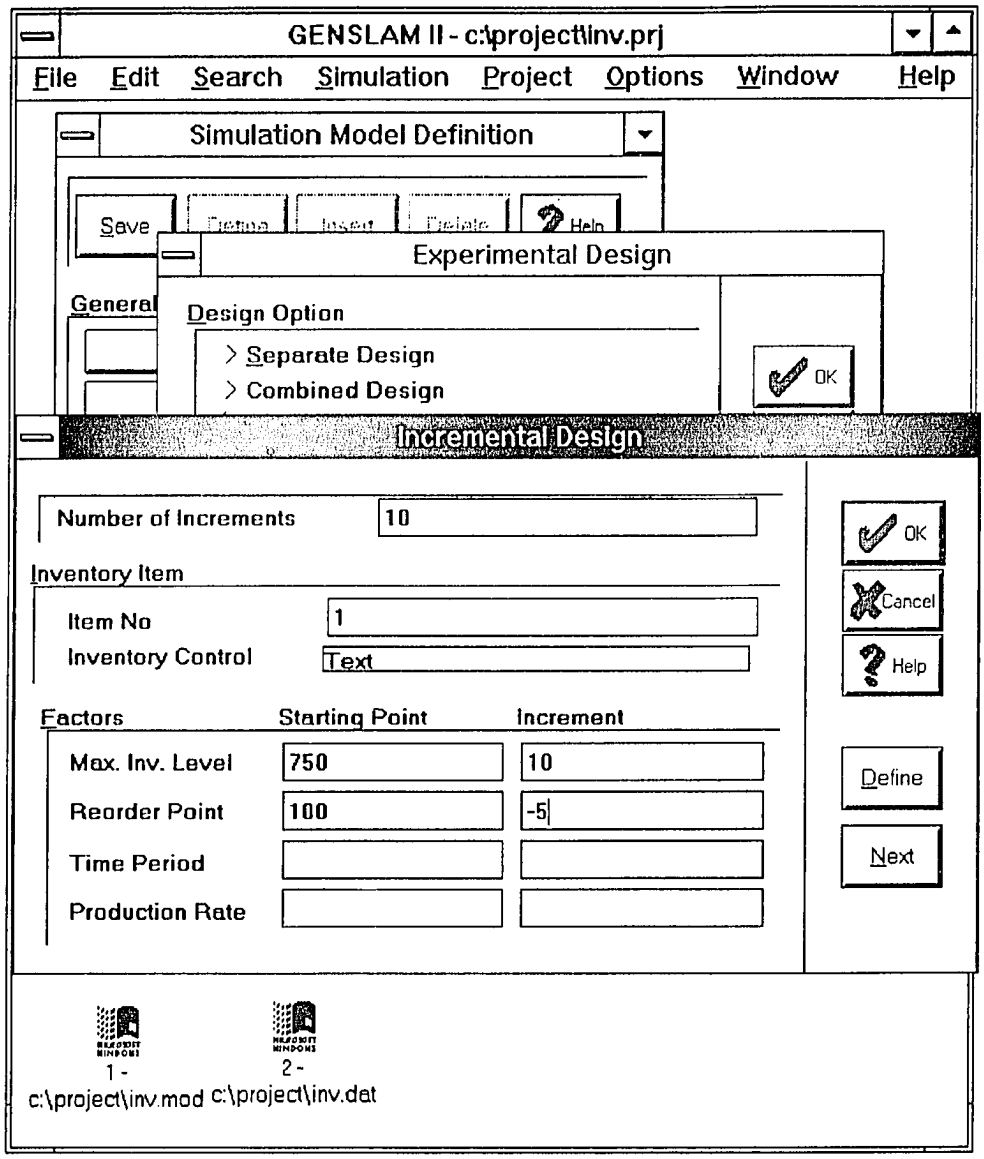

Figure 36. Dialog Box for Defining Increments for Inventory Policy Parameters.

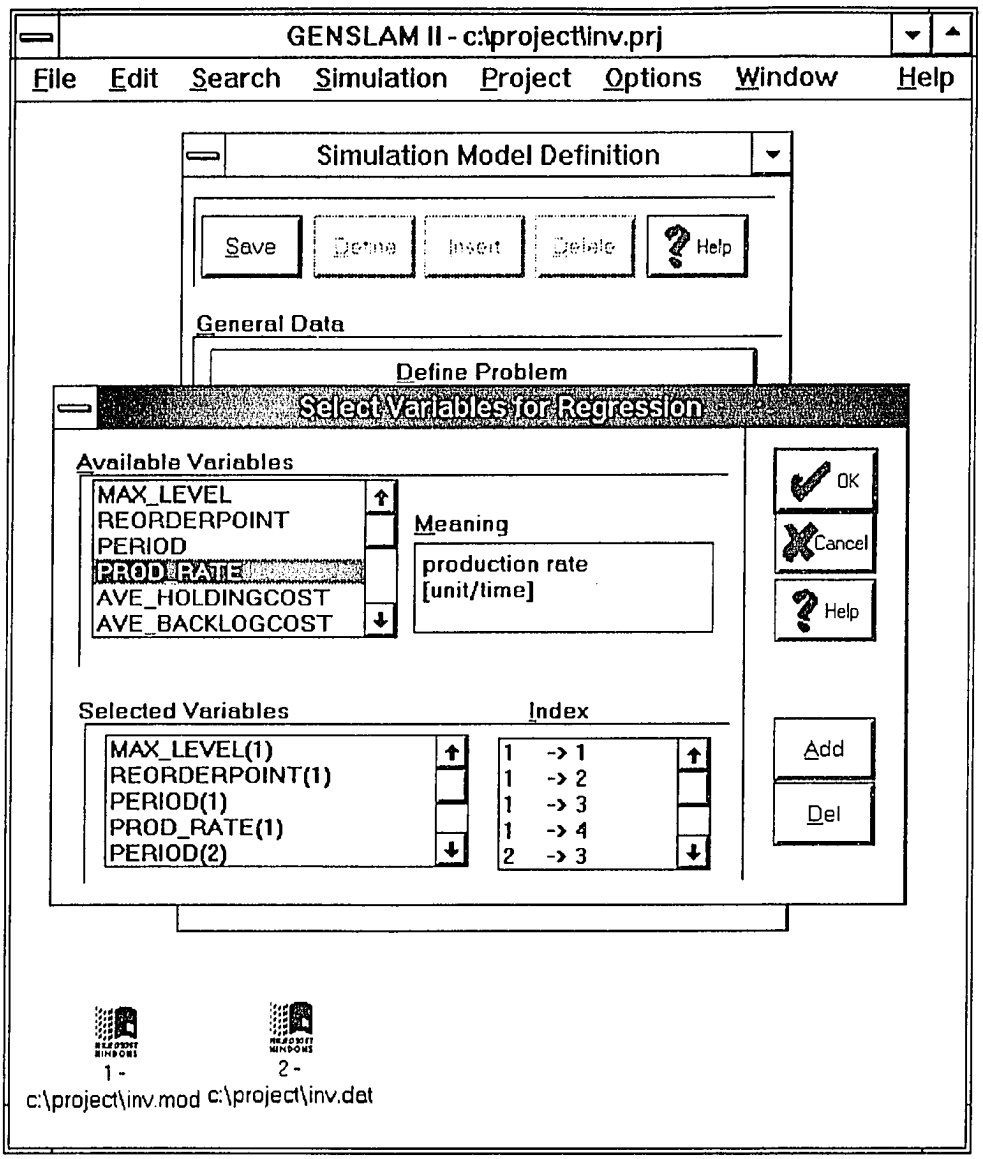

Figure 37. Dialog Box for Selecting System Variables for Regression Analysis.

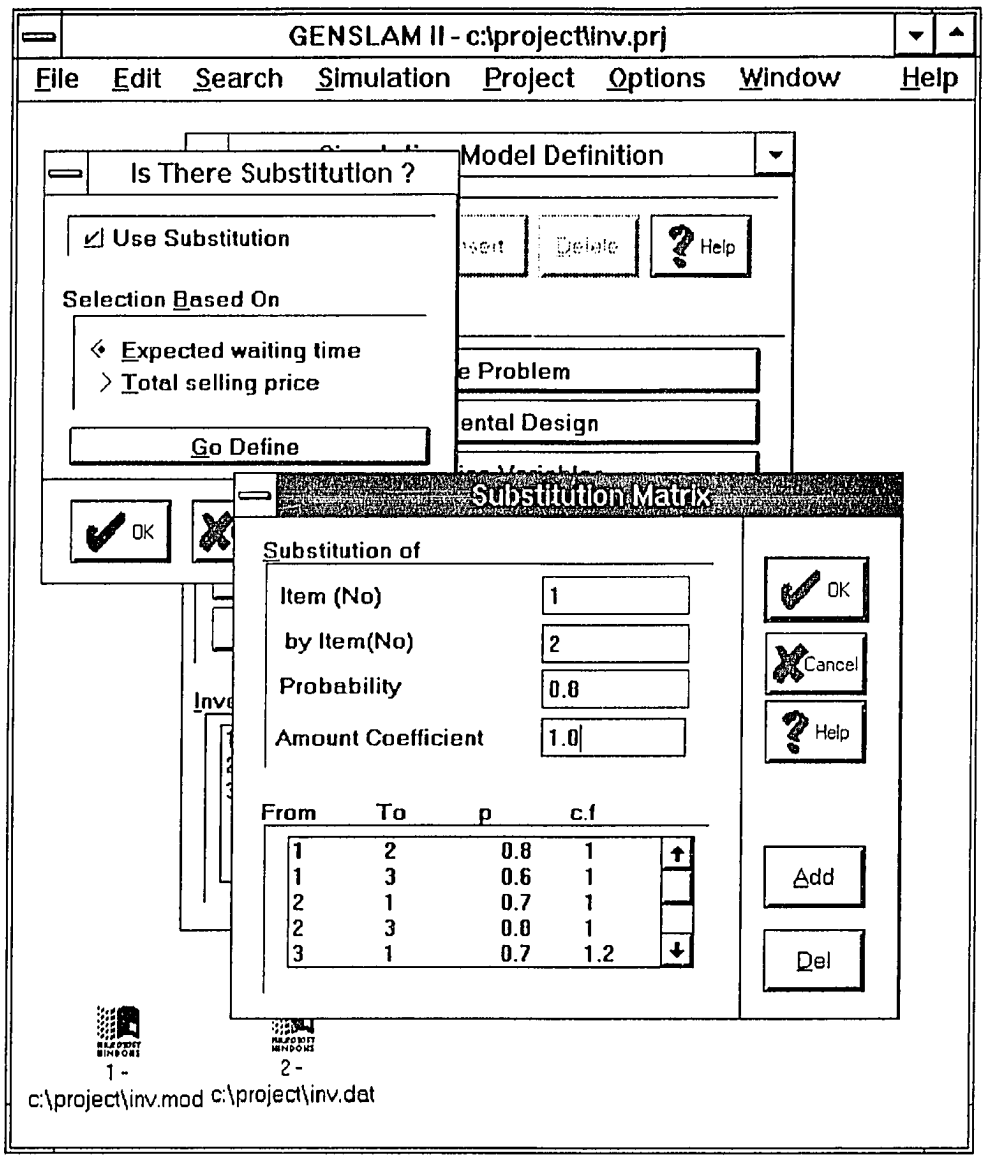

Figure 38. Dialog Box for Substitution Matrix.

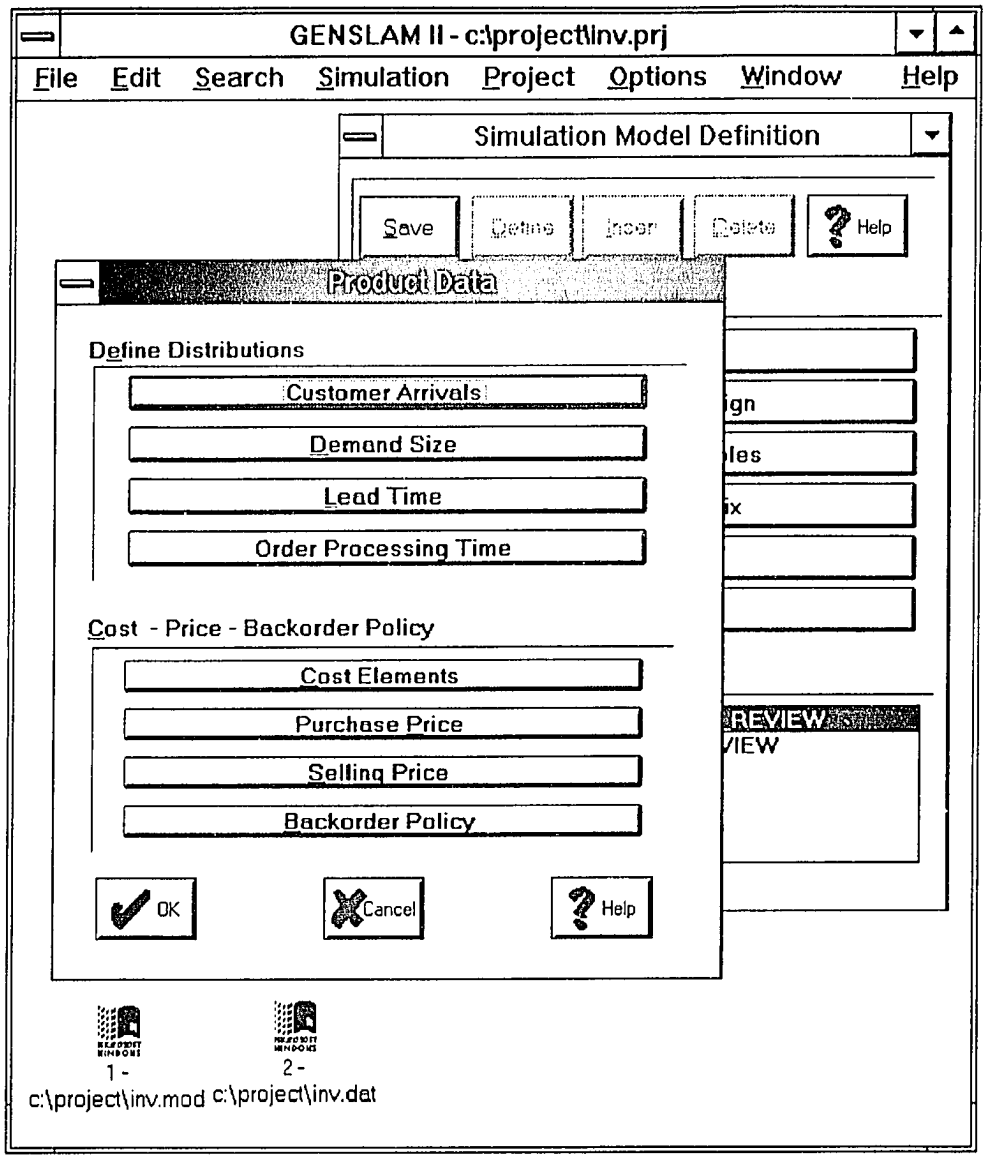

Figure 39. Main Dialog Box for Product Information Entry.

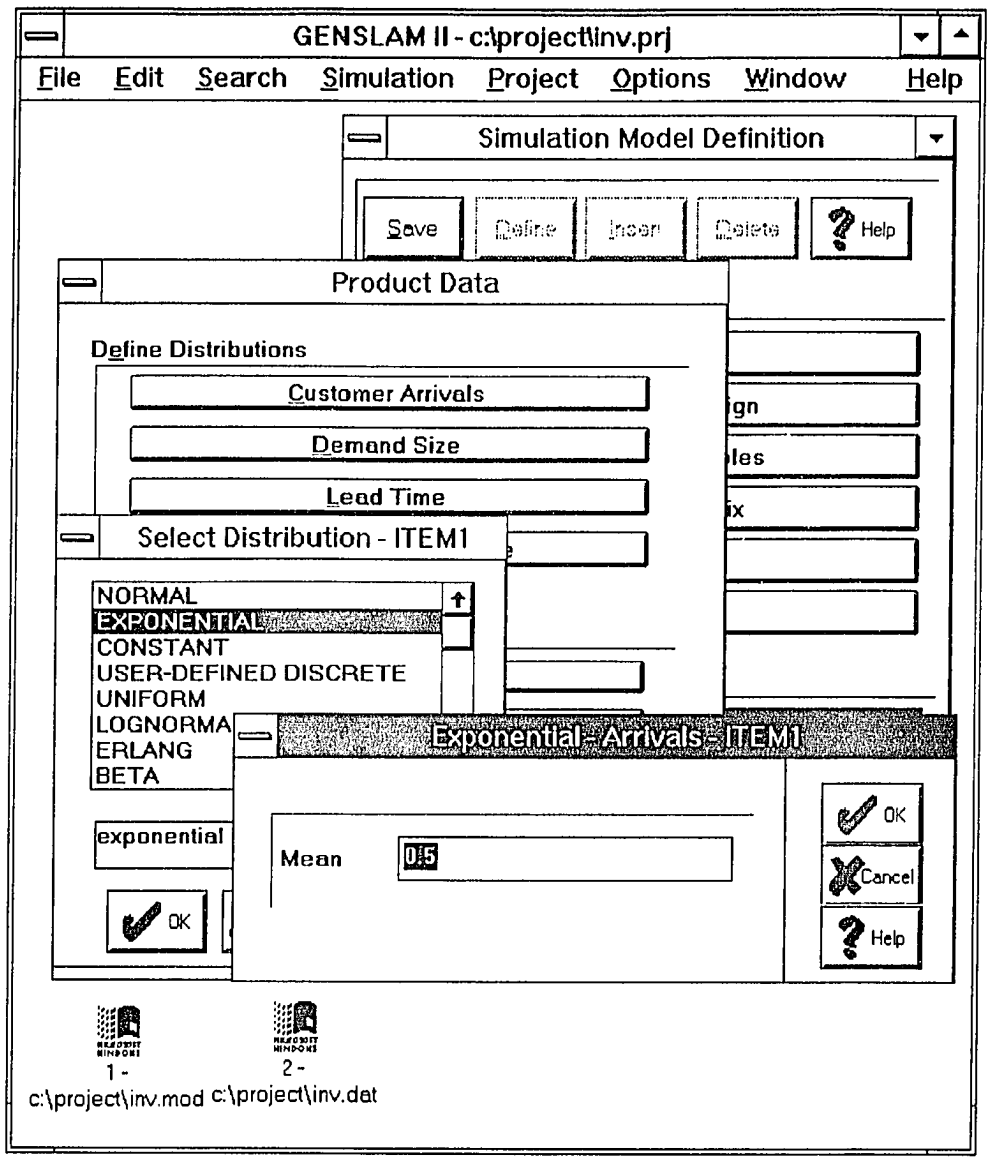

Figure 40. Dialog Box for Distribution Parameters.

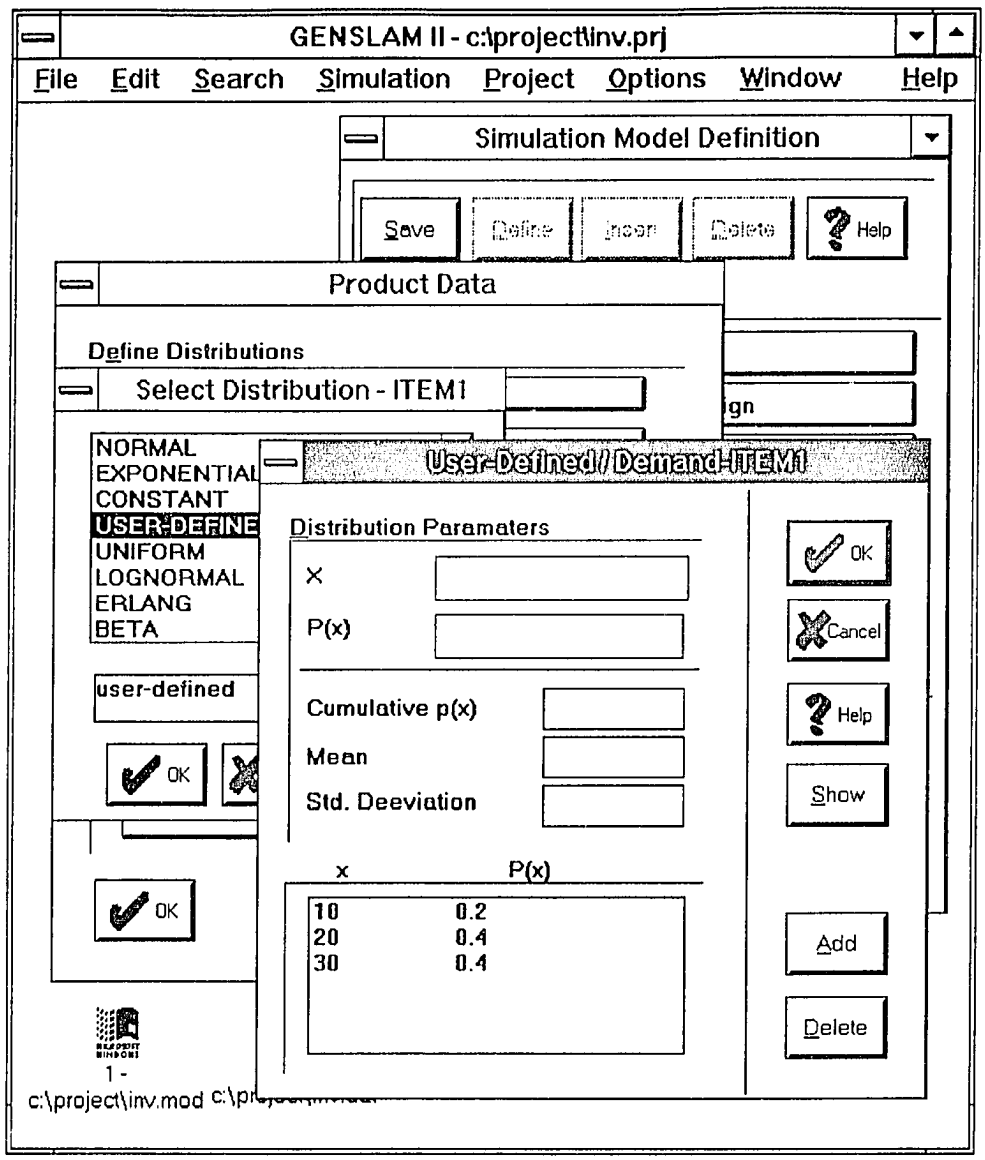

Figure 41. Dialog Box for User-Defined Empirical Distribution.

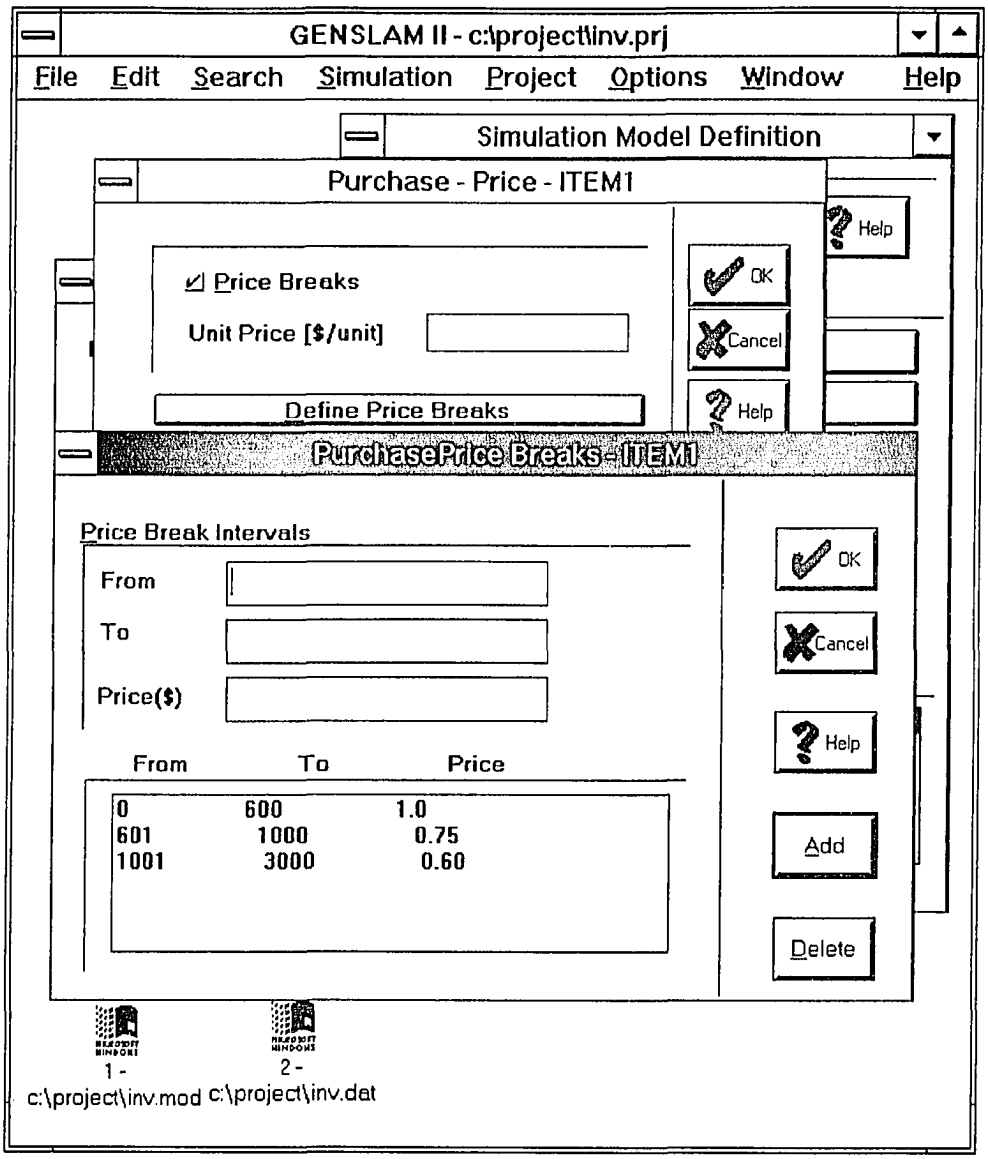

Figure 42. Dialog Box for Price Breaks.

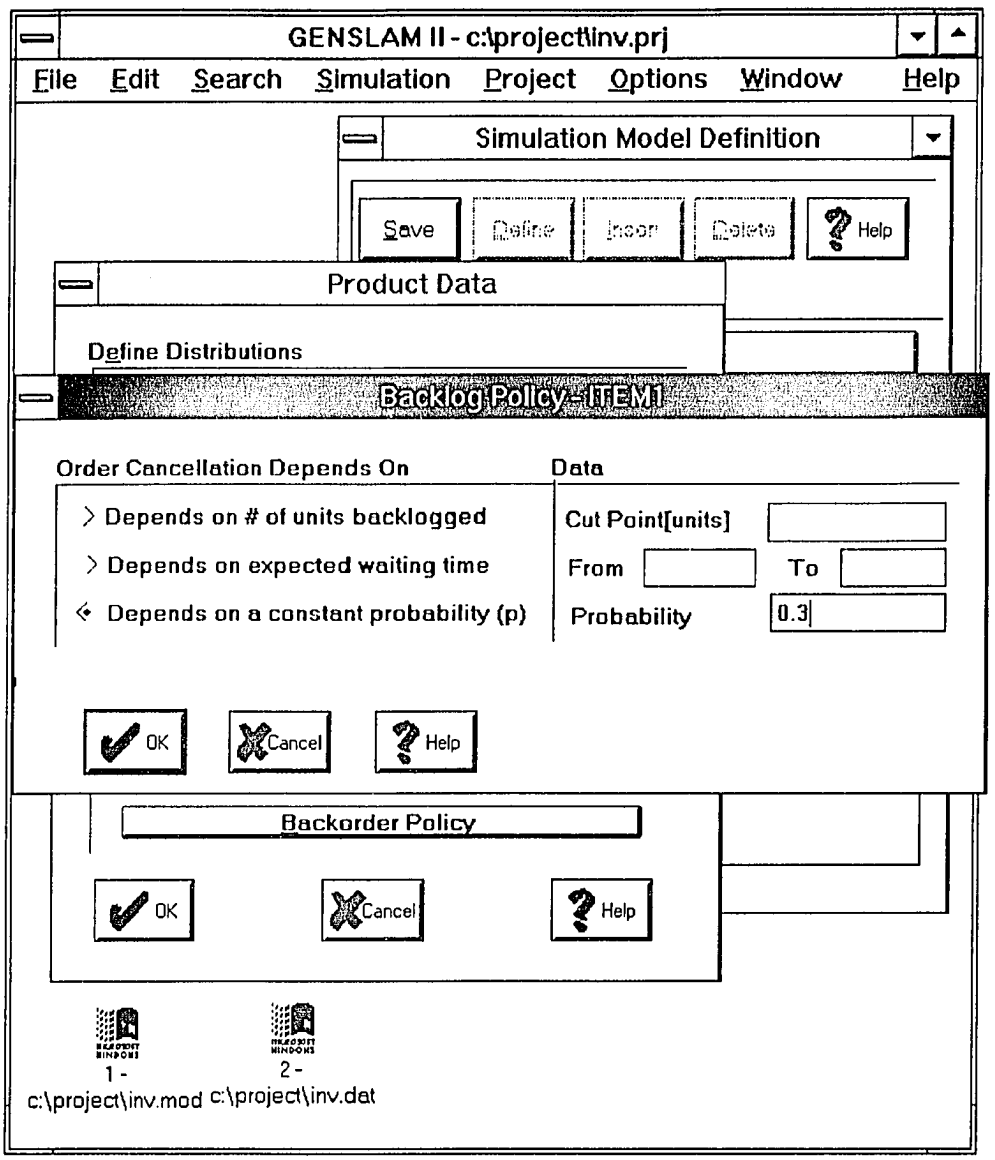

Figure 43. Dialog Box for Backlog Policy.

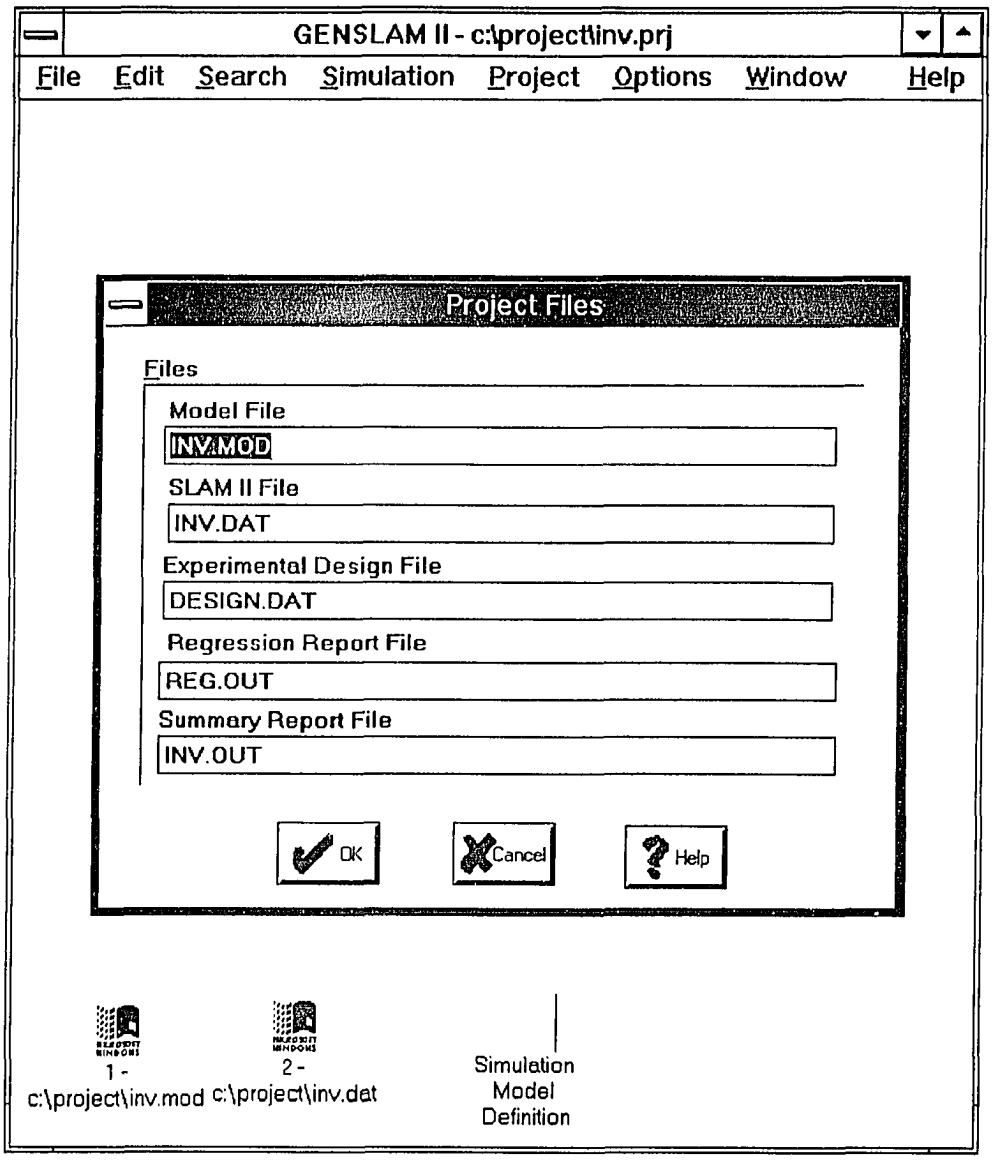

Figure 44. Dialog Box for Project File Names.

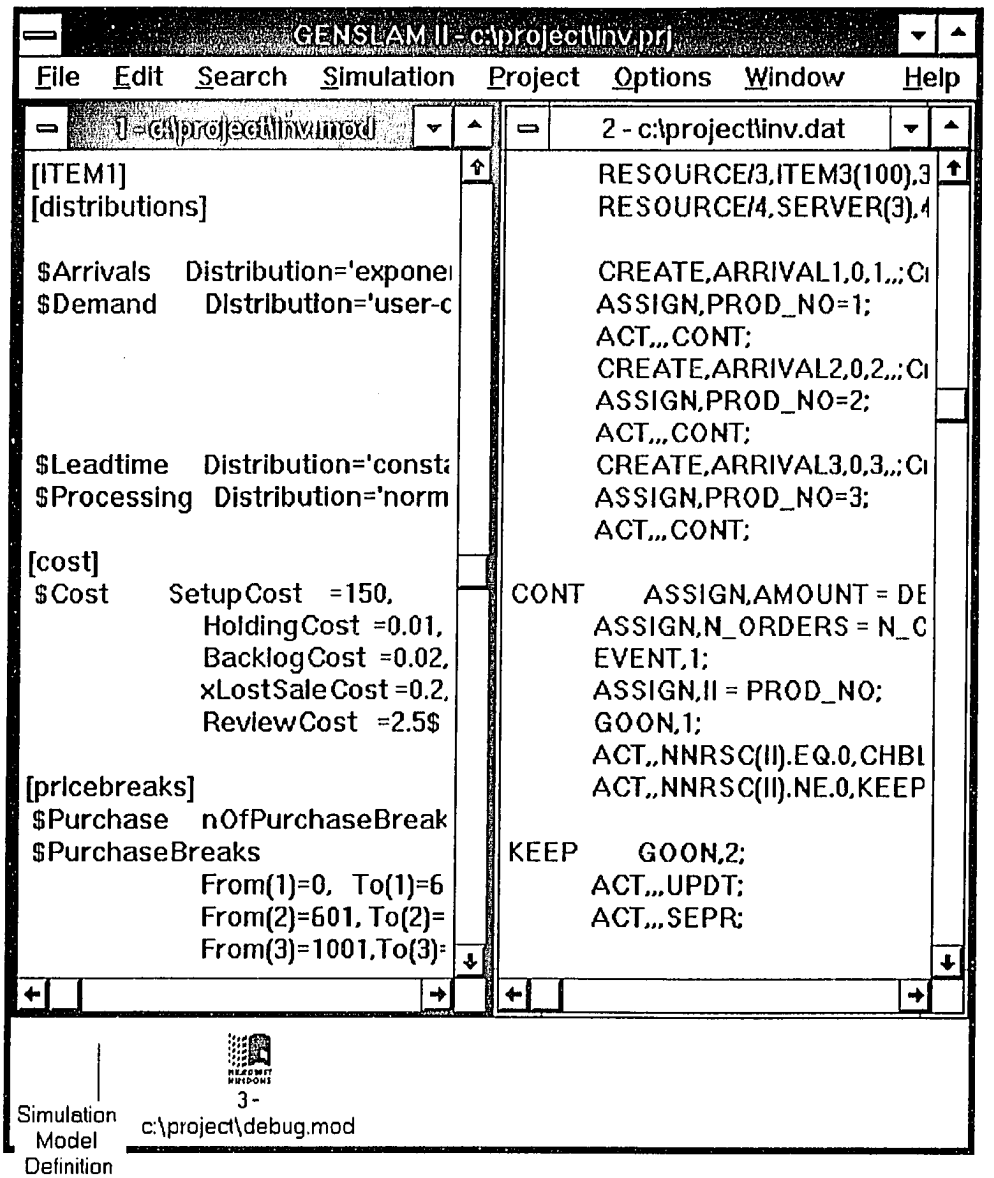

Figure 45. Program Editor.

BIBLIOGRAPHY

Balci, O., & Nance, R. E. (1987). Simulation model development environments: A research prototype. Journal of Operational Research Society, 38(8),753-763.

Biles, W. E., & Ozmen, H. T. (1987). Optimization of simulation responses in a multicomputing environment. In A. Thesen (Ed.), Proceedings of the 1987 Winter Simulation Conference (pp. 402-407). Atlanta, GA: IEEE.

Box, G. E. P., & Draper, N. R. (1987). Empirical model-building and response surfaces. New York: John Wiley & Sons, Inc.

Box, G. E. P., & Wilson, K. B. (1951). On the experimental attainment of optimum conditions. Journal of the Royal Statistical Society, 13, 1-45.

Chaharbaghi, K. (1990). Using simulation to solve design and operational problems. International Journal of Operations & Production Management, 10(9), 89-105.

Co, H. C., & Chen, S. K. (1988). Design of a model generator for simulation in SLAM. Engineering Costs and Production Economics, 14, 188-198.

Cochran, J. K., & Chang, J. (1990). Optimization of multivariate simulation output models using a group screening method. Computers & Industrial Engineering, 18(1), 95-103.

Crookes, J. G. (1987). Generators, generic models and methodology. Journal of Operational Research Society, 38(8), 765-768.

Datta, T. K., & Pal, A. K. (1990). A note on an inventory model with inventory-level-dependent demand rate. Journal of Operational Research Society, 41(10), 971-975.

Douikidis, G. I. (1987). An anthology on the homology of simulation with artificial intelligence. Journal of Operational Research Society, 38(8), 701-712

Eckel, B. (1989). Using C++. Berkeley: McGraw-Hill Inc.

Farrell, W. (1977). Literature review and bibliography of simulation optimization. In H. J. Highland (Ed.), Proceedings of the 1977 Winter Simulation Conference (pp. 117-124). Gaitersburg, MD: IEEE.

Flitman, A. M., & Hurrion, R. D. (1987). Linking discrete-event simulation models with expert systems. Journal of Operational Research Society, 38(8), 723-733.

Guasch, A., & Luker, P. A. (1990). SIMBIOS: Simulation based on icons and objects. In V. Maolisetti (Ed.), Proceedings of the 1990 SCS Multiconference (pp. 61-67). San Diego, CA: Society for Computer Simulation.

Haddock, J. (1987). An expert system framework based on a simulation generator. Simulation, 48(2), 45-53.

Haddock, J. (1988, March). A simulation generator for flexible manufacturing systems design and control. IEE Transactions, pp. 22-30.

Haddock, J., & O'Keefe, R. M. (1990). Using artificial intelligence to facilitate manufacturing systems simulation. Computers & Industrial Engineering, 18(3), 275-283.

Herring, C. (1990). ModSim: A new object-oriented simulation language. In V. Maolisetti (Ed.), Proceedings of the 1990 SCS Multiconference (pp. 55-60). San Diego, CA: Society for Computer Simulation.

Ho, Y. C., & Cao, X. (1983). Perturbation analysis and optimization of queueing networks. Journal of Optimization Theory and Applications, 40(4), 559-582.

Jordan, D. (1990). Implementation benefits of C++ language mechanisms. Communications of the ACM, 33(9), 61-64.

Koulamas, C. P. (1990). Optimal lot-sizing and machining economics. Journal of Operational Research Society, 41(10), 943-952.

Kumar, S., & Arora, S. (1990). Optimal ordering policy for a multi-item, single supplier system with constant demand rates. Journal of Operational Research Society, 41(4), 345-349.

Law, A. M., & Haider, S. W. (1989). Selecting simulation software for manufacturing applications. In E. A. Nachal (Ed.), Proceedings of the 1989 Winter Simulation Conference (pp. 29-32). Washington, DC: IEEE.

Law, A. M., & Kelton, W. D. (1991). Simulation modeling & analysis (2nd ed.). New York: McGraw-Hill Inc.

Mauro, C. A., & Smith, D. E. (1982). The performance of two-stage group screening in factor screening experiments. Technometrics, 24(4), 325-330.

Meketon, M. S. (1987). Optimization in simulation: A survey of recent results. In H. J. Highland (Ed.), Proceedings of the 1987 Winter Simulation Conference (pp. 58-67). Atlanta, GA: IEEE.

Microsoft Press. (1990). Micosoft Windows: Guide to programming. Redmond, WA: Author.

Montgomery, D. C. (1984). Design and analysis of experiments (2nd ed). New York: McGraw-Hill Inc.

Myers, R. H. (1976). Response surface methodology. Blacksburg, VA: Author.

Myers, R. H., Khuri, A. I., & Carter, W. H. (1989). Response surface methodology: 1966-1988. Technometrics, 31(2),137-157.

O'Keefe, R. M. (1986). Simulation and expert systems - a taxonomy and some examples. Simulation, 46(1), 10-16.

O'Keefe, R. M., & Roach, John W. (1987). Artificial intelligence approaches to simulation. Journal of Operational Research Society, 38(8), 713-722.

Park, K. S. (1989). Stochastic (Q,r) inventory model with customer reneging. Computers & Industrial Engineering, 16(4), 545-551.

Paul, R. J. (1991). Recent developments in simulation modeling. Journal of Operational Research Society, 42(3), 217-226.

Paul, R. J., & Chew, S. E. (1987). Simulation modeling using an interactive simulation program generator. Journal of Operational Research Society, 38(8), 735-752.

Raczynski, S. (1990). Graphical description and a program generator for queuing models. Simulation, 55, 147-152.

Sabuncuoglu, I., & Hommertzheim, D. L. (1989). Expert simulation systems-recent developments and applications in flexible manufacturing systems. Computers & Industrial Engineering, 16(4), 575-585.

Safizadeh, M. H. (1990). Optimization in simulation: Current issues and the future outlook. Naval Reserach Logistics, 37, 807-825.

SAS Institute. (1985). SAS User's guide: Statistics. Cary, NC: Author.

Shannon, R. E., Mayer, R., & Adelsberger, H. H. (1985). Expert systems and simulation. Simulation, 44(6), 275-284.

Shearn, D. C. S. (1990). PASSIM-A Pascal discrete event simulation program generator. Simulation, 55, 31-38.

Smith, D. E. (1973a). An empirical investigation of optimum-seeking in the computer simulation stiution. Operations Research, 21(2), 475-497.

Smith, D. E. (1973b). Requirements of an "optimizer" for computer simulations. Naval Research Logistics, 20, 475-497.

Smith, D. E. (1976). Automatic optimum-seeking program for digital simulation. Simulation, 27, 27-31.

Suri, R., & Leung, Y. T. (1989). Single run optimization of discrete event simulations-an emprical study using the M/M/1 queue. IEE Transactions, 21(1), 35-49.

Suri, R., & Zazanis, M. A. (1988). Perturbation analysis gives strongly consistent sensitivity estimates for the M/G/1 queue. Management Science, 34(1), 39-64.

Ulgen, O. M., & Thomasma, T. (1989). Computer simulation modeling in the hands of decision-makers. In E. A. Nachal (Ed.), Proceedings of the 1989 Winter Simulation Conference (pp. 89-94). Washington, DC: IEEE.

Ulgen, O. M., Thomasma, T., & Mao, Y. (1989), Object-oriented toolkits for simulation program generators. In E. A. Nachal (Ed.), Proceedings of the 1989 Winter Simulation Conference (pp. 593-600). Washington, DC: IEEE.

Wilson, J. R. (1987). Future directions in response surface methodology for simulation. In A. Thesen (Ed.), Proceedings of the 1987 Winter Simulation Conference (pp. 378-381). Atlanta, GA: IEEE.

Zeigler, B. P. (1987). Hierarchical, modular discrete-event modelling in an object-oriented environment. Simulation, 49(5), 219-230.