INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A FRAMEWORK FOR COMPUTER-ASSISTED SIMULATION EXPERIMENT DESIGN AND ANALYSIS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Yu-Hui Tao, B.S., M.S.

* * * * *

The Ohio State University

1995

Dissertation Committee:

Dr. Barry L. Nelson
Dr. Gordon M. Clark
Dr. Philip J. Smith
Dr. Jason C. Hsu

 Approved by

Adviser
Graduate Program in Industrial and Systems Engineering
To My Parents and My Wife, Chu-Chen Rosa
ACKNOWLEDGEMENTS

I would like to extend my deepest appreciation to Dr. Barry Nelson for his patience and guidance throughout my graduate program at OSU. His thoughtfulness and logics never cease to inspire me. I am very proud to have been advised by the greatest teacher I have ever known. I am especially grateful for his continuous encouragement to finish my degree after a full-time job and two babies. Thanks also go to Dr. Gordon Clark, Dr. Philip Smith, and Dr. Jason Hsu for their valuable contribution and feedback. I would also like to thank Cedric Sze for giving me the opportunity to work and to learn in ISE computer labs and for showing me the great personal quality of having a positive working altitude.

Most of all, I like to acknowledge my wife, Chu-Chen Rosa, who has stood behind me all these years, of her unlimited patience and moral support. Without her understanding and care for our two babies, Jonathan and Michael, during my frequent absence at home, none of these would have been possible.
VITA

May 16, 1964 ............................................... Born - Taiwan, R.O.C.

Sep., 1982 - June, 1986 ....... B.S., Tunghai University, Taichung, Taiwan, R.O.C.

Sep., 1986 - June, 1987 ............... Teaching Assistant, Tunghai University, Taichung, Taiwan, R.O.C.

Mar., 1988 - Mar., 1989 ................ Graduate Research Associate, National Regulatory Research Institute, The Ohio State University, Columbus, Ohio

Sep., 1987 - June, 1989 ............... M.S., Department of Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio

Sep., 1989 - Mar., 1990 ................ Graduate Teaching Associate, Department of Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio

Mar., 1990 - Dec., 1993 ................ Graduate Administrative Associate, Department of Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio

Jan., 1994 - Present ....................... Senior Software Engineer, ComputerPeople Consulting Services, Columbus, Ohio

Publications

Research Publications

Fields of Study

Major Field: Industrial and Systems Engineering

Studies in:
- Computer Simulation: Prof. Gordon M. Clark
- Cognitive Engineering: Prof. Philip J. Smith
- Statistics: Prof. Jason C. Hsu
# Table of Contents

DEDICATION ................................................................. ii  
ACKNOWLEDGEMENTS ................................................... iii 
VITA ................................................................. iv  
LIST OF FIGURES .............................................................. viii  

## CHAPTER  

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Introduction and Motivation</td>
</tr>
<tr>
<td>II</td>
<td>A Conceptual Framework for Simulation Experiment Design and Analysis</td>
</tr>
<tr>
<td></td>
<td>2.1 Objectives and Literature Review</td>
</tr>
<tr>
<td></td>
<td>2.2 SEDA Components</td>
</tr>
<tr>
<td></td>
<td>2.3 A Framework for SEDA</td>
</tr>
<tr>
<td></td>
<td>2.3.1 Dynamic Model of the SEDA Task</td>
</tr>
<tr>
<td></td>
<td>2.3.2 Static Model of SEDA</td>
</tr>
<tr>
<td></td>
<td>2.4 Justification</td>
</tr>
<tr>
<td></td>
<td>2.4.1 Supporting Arguments</td>
</tr>
<tr>
<td></td>
<td>2.4.2 Related SEDA Applications</td>
</tr>
<tr>
<td>III</td>
<td>Issues in Computer-Assisted Simulation Experiment Design and Analysis</td>
</tr>
<tr>
<td></td>
<td>3.1 Introduction</td>
</tr>
<tr>
<td></td>
<td>3.2 Program Structure of the SEDA-CS</td>
</tr>
<tr>
<td></td>
<td>3.3 Knowledge Acquisition Methods</td>
</tr>
<tr>
<td></td>
<td>3.4 System and User Interface Designs</td>
</tr>
<tr>
<td></td>
<td>3.5 Knowledge Representation and Reasoning Methods</td>
</tr>
<tr>
<td></td>
<td>3.6 Connection between CS and Simulation Models</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7 Time Management</td>
<td>42</td>
</tr>
<tr>
<td>3.8 Data Management</td>
<td>45</td>
</tr>
<tr>
<td>3.9 Procedure Management</td>
<td>48</td>
</tr>
<tr>
<td>3.10 Random-Number Assignment</td>
<td>50</td>
</tr>
<tr>
<td>IV A Formative User Evaluation for the Prototype SEDA-CS</td>
<td>54</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>54</td>
</tr>
<tr>
<td>4.2 Descriptions of the SEDA-CS</td>
<td>54</td>
</tr>
<tr>
<td>4.2.1 System-Requirement Level</td>
<td>55</td>
</tr>
<tr>
<td>4.2.2 System-Design Level</td>
<td>57</td>
</tr>
<tr>
<td>4.2.3 System-Implementation Level</td>
<td>62</td>
</tr>
<tr>
<td>4.3 Setup for Evaluating the SEDA-CS</td>
<td>69</td>
</tr>
<tr>
<td>4.3.1 Goals and Expectations for the User Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>4.3.2 Methodology</td>
<td>72</td>
</tr>
<tr>
<td>4.3.3 Settings for the Formative User Evaluation</td>
<td>73</td>
</tr>
<tr>
<td>4.4 Evaluation Results</td>
<td>75</td>
</tr>
<tr>
<td>4.4.1 Results and Interpretation</td>
<td>76</td>
</tr>
<tr>
<td>4.4.2 SEDA Constructs</td>
<td>77</td>
</tr>
<tr>
<td>4.4.3 User Behavior</td>
<td>82</td>
</tr>
<tr>
<td>4.4.4 Conclusions</td>
<td>86</td>
</tr>
<tr>
<td>V Technical Issues, Contributions, and Future Work</td>
<td>87</td>
</tr>
<tr>
<td>5.1 Peripheral Issues</td>
<td>87</td>
</tr>
<tr>
<td>5.1.1 Connection between the SEDA-CS and Simulation Programs</td>
<td>88</td>
</tr>
<tr>
<td>5.1.2 Primitive Procedures</td>
<td>91</td>
</tr>
<tr>
<td>5.1.3 Incremental Data Generation</td>
<td>94</td>
</tr>
<tr>
<td>5.1.4 Graphical User Interface</td>
<td>95</td>
</tr>
<tr>
<td>5.1.5 Memory Management</td>
<td>99</td>
</tr>
<tr>
<td>5.2 Contributions</td>
<td>100</td>
</tr>
<tr>
<td>5.3 Future Work</td>
<td>101</td>
</tr>
<tr>
<td>5.3.1 A Complete Computer-Assisted Simulation Working Environment</td>
<td>101</td>
</tr>
<tr>
<td>5.3.2 Statistical Issues in Designing and Implementing an SEDA-CS</td>
<td>102</td>
</tr>
<tr>
<td>5.3.3 A Comparison Experiment for the Prototype SEDA-CS</td>
<td>102</td>
</tr>
</tbody>
</table>

BIBLIOGRAPHY                                           105
# List of Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Classification of Simulation Problems</td>
</tr>
<tr>
<td>2</td>
<td>Decomposition of Simulation Tasks</td>
</tr>
<tr>
<td>3</td>
<td>Hierarchy of Simulation Procedures</td>
</tr>
<tr>
<td>4</td>
<td>A Summary of the SEDA Framework</td>
</tr>
<tr>
<td>5</td>
<td>Hybrid Representation and Reasoning Methods</td>
</tr>
<tr>
<td>6</td>
<td>Connection Design</td>
</tr>
<tr>
<td>7</td>
<td>SEDA-CS Structure</td>
</tr>
<tr>
<td>8</td>
<td>SEDA-CS Main Screen</td>
</tr>
<tr>
<td>9</td>
<td>SEDA-CS Main Screen with a Pop-Up Window</td>
</tr>
</tbody>
</table>
CHAPTER I

Introduction and Motivation

Computer simulation is a useful tool for supporting decision making, especially when there is no analytical method available. However, computer simulation is a complex task for many users. Simulation experiment design and analysis (SEDA) is as critical as simulation modeling to obtaining useful results. Our concern is how simulation users can be assisted with SEDA. This paper presents a conceptual framework for SEDA. In order to be concrete we focus on the problem of comparing mean or expected performance of queueing-network systems.

A typical simulation project includes the following stages [33]: Formulation, programming, verification, validation, experiment design and analysis, and recommendations. Accordingly, a traditional simulation environment requires four classes of knowledge [22]: Knowledge about the problem domain, knowledge about simulation modeling, knowledge about the programming language, and knowledge about statistics.

Shannon [29] estimated that one needs to spend "at least 720 hours of formal classroom instruction plus another 1440 hours of outside study (more than 1 man-year of effort)" to acquire the basic simulation tools. This does not include the extra effort required to gain real-world, practical experience in order to become proficient.
Therefore, it is not easy for a novice to perform a thorough simulation study.

A survey of large U.S. firms by Thomas and DaCosta [35] indicated that only 28 percent of individuals in operations research departments have backgrounds in statistics or mathematics. Another survey by Nelson et al. [21] showed that computer simulation is the second most used OR tool in production management, after regression analysis. A recent survey by Dyer et al. [7] ranked computer user skills, probability and statistics, and simulation techniques as the number 1, 2, and 3 technical skills, respectively, for a recent graduate of a master's program, and the number 1, 2, and 4 technical skills, respectively, for a seasoned professional in MS/OR.

The implication of the three survey results is that the lack of understanding of how to correctly design the simulation experiments, and analyze and interpret the simulation output, may be a problem in the simulation user population. Moreover, based on our experience, users with basic statistical background nevertheless may not know how to appropriately apply statistics in simulation. As a result, even though the population of simulation users who have statistics background may increase, the problem still exists. Therefore, a computer system (CS) that can aid the users in simulation statistical analysis is critically important and useful to the success of a user population without qualified statisticians or adequate knowledge of the SEDA problem solving.

Chapter 2 presents a conceptual framework for computer-assisted SEDA. Important issues in designing such a CS are raised in Chapter 3. In Chapter 4, a description of a prototype SEDA-CS is summarized followed by a user evaluation. Chapter 5 dis-
cusses some technical issues encountered in implementing this prototype SEDA-CS, and describes potential future work for providing a computer-assisted environment to simulation users.
CHAPTER II

A Conceptual Framework for Simulation Experiment Design and Analysis

2.1 Objectives and Literature Review

Our main objective is to help simulation users with Simulation Experiment Design and Analysis (SEDA) through a Computer System (CS). There is already some software developed to assist simulation users in the simulation problem-solving environment, typical examples are simulation program generators, intelligent front ends (IFE), integrated simulation systems, and simulation analysis systems.

IFEs ([12], [5]) and program generators ([18], [32], [8], [13], [23]) are useful tools for interactive simulation modeling environment with existing simulation software, while integrated simulation systems ([28], [19]) provide AI-based simulation working environments with consistent specification and internal representation in both simulation modeling and analysis. Most of the simulation analysis tools ([34], [15], [17], [3]) help simulation users either via automating the process or focusing on a specific field, such as manufacturing.

Only the following three papers are considered SEDA problem-solving applications:
1. Mellichamp and Park ([20]) organized statistical procedures by simulation problems, and developed "Statistical Expert System for Simulation Analysis (SESSA)" to assisted simulation users in selecting the appropriate procedures for each sub-problem.

2. Taylor and Hurrion ([33]) justified the use of the expert system framework for simulation experimentation, and developed "Warwick Expert Simulation (WES)" to assist simulation users in SEDA problem-solving with embedded problem-solving expertise.

3. Ramachandran, Kimbler, and Naadimuthu ([25]) discussed a framework for an expert post-processor for simulation output analysis (EPSONA), with the emphasis on interactive model validation, to be used in their "Intelligent Simulation Generator (ISG)".

In general, the literature supports the idea that a CS can help simulation users in all aspects of simulation problem solving. The last three papers, in particular, demonstrate the feasibility of such an interactive computer-assisted SEDA application. However, the fundamental characteristics of the SEDA task could not be clearly or completely identified in their papers, and thus we believe that there is room for substantial gains by incorporating these fundamental characteristics; we demonstrate this via a prototype SEDA-CS in Chapter 3 and 4.

We agree with Freeman [9] in "The Nature of Design:"

It is essential that one has a conceptual understanding of a complex ac-
tivity in order to master its intricacies. Without such a framework, one has only isolated facts and techniques whose interrelationships may be obscured. Without an understanding of broad classes of phenomena, one is condemned to understand each new instance by itself.

Therefore, we need a fundamental SEDA framework for understanding the complex activity of SEDA, and for any interested SEDA-CS developer to use as a basis. This framework should encompass a computer-assisted working environment and the practical SEDA problem-solving process.

This chapter presents a conceptual framework for the complex activities in SEDA, and illustrates the framework with a brief example. The definition of six SEDA components forms the basis of the framework. The framework itself includes a dynamic and a static model. The dynamic model is a generic description of the sequential nature of SEDA. The static model, on the other hand, is a very specific description of our chosen problem domain, comparison of expected performance of queueing networks. We close by arguing that the proposed SEDA framework is a good one based on its properties, and a comparison of related CSs.

To sum up, the organization of this chapter is to first define the basic terms in SEDA in Section 2.2, followed by a framework consisting of the dynamic and static models of the SEDA task in Section 2.3. Then Section 2.4 argues that this is a good framework as illustrated by comparing it to related CSs.
2.2 SEDA Components

Our world of SEDA consists of six *components* which form the basis of the SEDA framework. They are:

1. **SYSTEM**: a black box with one or more parameters, which takes prescribed input and produces corresponding output.

   A system in our world is a collective term, such as the M/M/1 system, which represents a class of similar system instances with different parameter values. The term system as it is commonly used refers to one instance of a class of systems in our SEDA world.

2. **PARAMETER**: a collection of constants that define an instance of a system.

   The instances of one system are distinguished by a common set of parameters, but with different values.

3. **RESOURCE**: a constrained quantity that is necessary to solve a problem.

   The resources considered in our SEDA world are (real) time, the computer system, and the user.

   We believe that real time is the most important resource in SEDA and other resources, such as CPU time (power), can be expressed in terms of it. The computer system and the user contribute information, including knowledge and decisions, to this SEDA problem-solving environment.

4. **DESIGN**: the design in our SEDA world consists of the number of replications, the stopping time for each replication, the random number assignment, and the data aggregation.
Data aggregation is any reduction of the raw output data which may be needed if we are not be able to efficiently keep and utilize all the raw data, or any transformation of the data into a more useful form. Within our world, data aggregation includes batch size and data deletion, for batching and weighting the data, respectively. Other types of aggregation could be included.

5. DATA: all of the simulation output.

The data are characterized by a multivariate joint distribution that is typically unknown to us.

6. ANALYSIS: deriving statements about systems. The term “statement” will be formally defined in Section 2.3.1.

The definitions above for the components in our SEDA world establish common ground for further discussion. Although they may not be exhaustive, these components represent our view of the SEDA world. These components are strong organizing principles for SEDA. Their definitions are precise, but not at a working level. Using these organizing principles, the framework of SEDA is presented next.

2.3 A Framework for SEDA

We now present a framework for the SEDA task consisting of a dynamic and a static model. The dynamic model in Section 2.3.1 defines fundamental SEDA concepts, and describes how SEDA proceeds dynamically through the primitives consisting of the concepts and the components in our SEDA world. The static model in Section 2.3.2 presents the structure of simulation problems, SEDA building blocks, and statistical procedures in a hierarchical fashion.
2.3.1 Dynamic Model of the SEDA Task

By the dynamic model of the SEDA task we mean a description of the interplay between the SEDA primitives in an interactive environment. The "primitives" include the SEDA components (defined in Section 2.2), and the concepts, introduced next. In brief, components are generic building blocks of SEDA while concepts define SEDA dynamics. The relationship between primitives is illustrated by an example at the end.

Fundamental SEDA Concepts

The SEDA concepts are:

1. SYSTEM INSTANCE: a system with a set of fixed values of the system-dependent parameters.

The difference between a system and a system instance is that several system instances could be derived from the same system with different values of the parameters.

2. STATEMENT: any declaration about the system instances.

The sources of statements are prior knowledge and experimental analysis. Prior knowledge includes knowledge of the problem domain and of similar classes of systems, while experimental analysis draws intermediate and final conclusions based on data.

3. SCOPE: the subset of data upon which a statement is based.

The scope of a statement therefore implies the applicable system instances.

4. PROCEDURE: a function of data and statements that produces a new
statement.

Parametric and non-parametric statistical procedures are two major categories of procedures.

5. **EXPERIMENT**: executing a system instance according to a design to produce data.

The terminology used for these concepts may not be standard, but their meanings (definitions) are unambiguous in our SEDA world. As opposed to the SEDA components defined in Section 2.2, the definitions of these concepts are at a working level that describes the SEDA task.

Concepts and components are both SEDA primitives. The reason we separate them is because a component is more concrete and traditionally known in SEDA, while a concept is more ambiguous, confusing, and is often ignored.

The main reason why these fundamental concepts are valuable is because through defining them the components of SEDA—the system, the resource, the design, the parameter, the data, and the analysis—can be connected and reasoned in a dynamic, sequential, problem-solving process, as described below.

**Dynamics of SEDA**

Because the SEDA problem solving is basically an iterative process, we first describe the generic cycle. Then, based on this generic cycle, the sequential nature of the SEDA task is presented.

One SEDA-cycle describes the possible activities and interactions between the
primitives. In brief: Resources + Pre-Analysis + Experiment Design + Statements + Data + Post-Analysis $\implies$ New statements + Scope.

Stated differently, the SEDA-cycle proceeds as follows: under the real time constraint, a pre-analysis is performed for deriving new statements and an experiment design that generates the data. Then, a post-analysis is performed using both the data and available statements for producing new statements and their scope. The computer system and the user provide the necessary knowledge and decisions during this SEDA-cycle within their capabilities.

Although a generic SEDA-cycle attempts to describe all the possible interactions between these primitives, all the activities will not necessarily happen in every SEDA-cycle. Nevertheless, one or more statements are produced in every SEDA-cycle.

Statements are produced sequentially during the iterative SEDA-cycles, so their scope may reach backward several cycles. In other words, any statement may depend on previous statements. For example

$$\text{statement}_5 = \text{proc}_5(\text{data}_5, \text{proc}_4(\text{data}_4...)).$$

Notice that the system instances and the subset of data are embedded within this sequential representation of the SEDA task.

Theoretically, if there is no resource limit, then the true values of the system performance measures of interest can be obtained. In reality this sequential SEDA process has to stop before the available real time runs out. Accordingly, the goal of SEDA problem solving, in practice, is to produce a simulation result within a desired error level under the real time constraint.
Notice that our definition of "sequential" is broader than the classical statistical definition. The major difference is that we allow backtracking and interaction, which means that the analysis path is not fixed but flexible.

A final comment is that these fundamental concepts and components are well defined and precise elements of the dynamic SEDA task. However, in real SEDA problems, distinct concepts or components may not be easily identified and separated. For example, experiment design and analysis are tightly bound together and thus it is not easy to clearly identify which is which or to match any design with any analysis. Also, the procedures that are chosen very often affect the experiment design and thus they do not necessary follow the process exactly. An expert understands better than a novice about this tight coupling, which is the ability to look ahead. Yet this generic SEDA cycle also demonstrates the strength of our SEDA primitives which represent the complex SEDA process in a simple but well-defined manner.

An Example

The sequential nature of SEDA can be illustrated by a few segments of a simplified example, which is based on a protocol script from a real simulation problem solved by an SEDA expert.

A user wants an SEDA expert to help find the smallest average waiting time among three queueing system instances with mean interarrival times and mean service times (1.05, .9), (1.0, .8), and (.9 , .7), respectively. Since the final report will be due within eight hours, the user can only spend four hours of simulation study before writing it.
System: the queueing system.

System Instances: the queueing system with three different sets of values for the system parameters, interarrival time and mean service time.

Parameters: (interarrival time, mean service time) = (1.05, .9), (1.0, .8), and (.9, .7) for the three system instances, respectively.

Resource: four hours of real time and the user and the SEDA expert, where the user may supply the knowledge about the system and major SEDA decisions, and the SEDA expert may supply the knowledge of SEDA and queueing systems.

"I (the SEDA expert) am going to make replications and look at the bias for one of the system instances. I have no other reasons for choosing this approach but to get a better look at the bias.... Look at the traffic intensity and find out which is the most congested system. I will use system instance 1 since it has the highest traffic intensity."

Analysis: the quote above is part of the pre-analysis.

Resource: the SEDA expert who is contributing his knowledge of SEDA and queueing systems.

System Instance: the queueing system with the first set of parameters.

"Let me make a quick run of 2 replications of 2,500 observations for system instance 1.... It did not take long (about 3 seconds) to make 2 replications."
Design: the number of replications (2) and the stopping time for each replication (2,500 waiting times).

Data: the simulation output as planned in the above design.

Experiment: Executing system instance 1 with the design of 2 replications each with 2,500 waiting times.

Statement: 3 seconds for 2 replications of 2,500 observations for system instance 1.

Scope: system instance 1 with the data in the statement.

"Look at the data. Some bias at the beginning.... It seems to climb quickly. I am taking a little gamble to do 10 replications each with 200 observations."

Procedure: visual inspection of the trend of data.

Statement: "Some bias at the beginning." It has the same scope as the previous statement.

Analysis: inspecting the trend of the data and producing a statement about a new design.

Design: the new number of replications (20) and stopping time (200 observations).

The following statements with their implicit scope illustrate the sequential nature of the SEDA expert's problem-solving process:

Statement 4 "The three queueing system instances are logically similar."
Statement 20 "I am sure that the appropriate initial bias deletion point is 500 for system instance 1."

Statement 21 "I spent too much time in determining the initial bias deletion for system instance 1 and I have very limited time available now."

Statement 22 "I will use this deletion point for all three queueing system instances."

Statement 23 "The current data, which looks exponential and has lag-1 correlation 0.51, fail the normal and the independence assumptions for system instance 1."

One major element that distinguishes an expert from an intermediate user is that the expert is able to manage the resource limit, real time, in simulation problem solving, but the intermediate user usually is not. Also, the expert reasons by combining statements to obtain a desirable new statement. Those experts who well understand the sequential nature of simulation problem solving know how to achieve the goal effectively and efficiently with the available resources.

2.3.2 Static Model of SEDA

The static model is a description of the SEDA structure for solving a particular class of problems, in our case the problem of comparing expected performance across queueing-network models. Our static structure is represented as a three-layer model.

To help simulation users, we need a static model that is not only based on a solid set of primitives, but is also easily understood and learned by simulation users. The static model need not be unique, but it should be able to be explained by the SEDA
primitives, and be able to explain the actual design of an SEDA-CS.

Accordingly, the difference between the dynamic model and the static model, in addition to the dynamic verses static view, is that the dynamic model forms a high-level abstraction for the nature of SEDA that will not change over time, while the static three-layer model is a lower-level representation of the SEDA task that may change as the technologies or methodologies evolve.

Because this static model is a more detailed description, we present it within our research focus, comparison problems. Our static model of the SEDA task consists of a classification of simulation problems, a decomposition of simulation tasks, and a hierarchy of simulation procedures (a three-layer breakdown of SEDA is presented in Figures 1-3).

Layer one - classification of simulation problems: Figure 1 implies that any given simulation problem on the top node can be classified into a desired solution on the bottom nodes. Within the classification scheme, a comparison with known alternatives problem can be divided into three subproblems: basis of comparison, design, and analysis.

Before the basis of comparison, we first classify a simulation problem into either a comparison problem with known alternatives, or a comparison problem without a known alternative. In our view, all simulation problems are comparison problems. The main difference is to what a system instance is compared. If a system instance is only evaluated to compare to the true value of its performance measure of interest, which is unknown but fixed, then this is a comparison without a known alternative.
Figure 1: Classification of Simulation Problems
On the other hand, when a system instance is compared to other known alternatives, then this is a comparison with known alternatives.

**Layer two - decomposition of simulation tasks:** Since the representation of the task attempts to consider the subtasks and issues involved at different stages of the simulation problem solving, and since design and analysis are two iterative processes, a lower level-representation of the subtasks for design and analysis can be represented as in Figure 2.

Notice that in Figure 2 the comparison problem with a known alternative is further divided into two independent modules: the initial-bias recognition problem and the core comparison problem. Under these two subproblems are, then, the subtasks of design and analysis. One important point is that the subtasks under design and analysis can be derived from the primitives as defined earlier. This is what we think is important in designing a CS: A fundamental conceptual understanding of a complex activity for explaining the empirical model based on some task analysis.

**Layer three - hierarchy of simulation procedures:** The third layer is the procedure tree in Figure 3. Figure 3 first classifies a simulation problem into either an output-analysis problem or an initial-bias recognition problem. The decomposition matches Figure 3 until the “means-comparison problem.” From here on, a statistical procedure can be determined by branching down to the bottom levels of this hierarchy.

The procedures in Figure 3 are just one set of basic procedures used for comparison with known alternatives. There are many other procedures used by other experts, which are expert-dependent, and are not shown here. Accordingly, although this
Figure 2: Decomposition of Simulation Tasks
Figure 3: Hierarchy of Simulation Procedures
layer is necessary for actually solving a problem, the procedures are not exhaustive. Thus, it should be emphasized that the methodologies are evolving and they do not encompass all the details an expert considers in the problem-solving process.

The advantage of this layer is that the procedures are clustered based on the classification of simulation problems in Figure 1. For example, both output analysis and initial bias recognition nodes branch down to the means procedures as in Figure 1. Therefore, our static model can encompass a large range of statistical procedures and analysis.

Figures 1–3 are not complete. One reason is that we only focus on a limited problem, that is, the means-comparison problem with known alternatives. Another reason is that we provide only one possible scheme to represent the fundamentals of simulation problems. However, this scheme is not unique. Moreover, even within this scheme our colleagues may fill in or replace some of the details as the methodology and research advance. In other words, we are more concerned with the fundamental structure and the sketch of SEDA rather than the details within the structure. Consequently, the SEDA primitives should embrace any new methodology and not be limited to the procedures provided in the third layer of the SEDA model.

2.4 Justification

A prototype SEDA-CS has been developed based on our framework, and a formative evaluation has been performed to demonstrate the feasibility and appropriateness of this SEDA framework; this is presented in Chapter 3. We briefly argue that the proposed SEDA framework is a good framework in Section 2.4.1. We examine and
evaluate existing SEDA applications in terms of this framework in Section 2.4.2.

2.4.1 Supporting Arguments

We believe that an SEDA framework should be self-contained, simple, specific only when necessary, and comprehensive (See Figure 4 for a summary of the SEDA framework).

1. Self-contained: Our SEDA framework has in total eleven primitives that are briefly but precisely defined. These primitives help to derive and develop both the dynamic model and the static model. Since our framework is developed based on our SEDA world, it is self-contained.

2. Simple: Our SEDA framework is simple since it contains only the static and dynamic model of the SEDA task. Each of them has only one central theme: sequential natural and three-layer structure, respectively.

3. Specific Only When Necessary: Our SEDA framework involves only fundamental elements without any implementation detail. Moreover, all these fundamental elements are required to describe the SEDA framework. Although there are some specific details in our SEDA framework, they are necessary to completely describe the static model in the bottom level of the hierarchy of simulation statistical procedures.

We believe that the dynamic model will not be affected by technologies and methodologies over time, and only the bottom level of the static model might be.

4. Comprehensive: Our framework is comprehensive in that it is compatible with any statistical procedure. It covers everything from a complete, formal one-step data analysis to sequential data analysis. Therefore, as long as distinct alternatives are
Figure 4: A Summary of the SEDA Framework
simulated, this framework works in other situations such as ANOVA, metamodeling, etc.

For example, suppose that we were interested in knowing the relationship between the throughput rate and two factors, the queueing discipline (A) and buffer size (B), in a queueing network system and we have 2 hours of time available. A $2 \times 2$ factorial design could be used in our simulation study with two levels in each factor: FIFO ($A_1$) or Shortest Processing Time First (SPTF) ($A_2$); and large buffer ($B_1$) or small buffer ($B_2$). We could run 20 replications each of 8 hours for each of the $2 \times 2$ cells within the 2 hours available. The finding could be that the $A_2B_2$ combination produced the highest throughput rate. In terms of our framework, the following concepts and components can be identified:

**System**: The queueing network system.

**Parameter**: Queueing discipline with FIFO and SPTF, and buffer capacity with large size and small size.

**System Instance**: Four system instances corresponding to the $2 \times 2$ design.

**Resource**: 2 hours.

**Design**: 20 replications each with 8 hours of simulated time.

**Experiment**: Executing the above design for 20 replications each with 8 hours of simulated time.

**Data**: Simulation data from the experiment above.

**Procedure**: ANOVA.

**Analysis**: Examining the ANOVA table and stating the finding.
Statement: Interaction exists between queueing discipline and buffer size.

Scope: All 4 system instances each with 20 replications of 8 hours of simulated time.

A good representation not only needs to have the characteristics defined above, but also to be arranged in a smooth sequence for easy comprehension. To this end, the dynamic and static models are decomposed in a hierarchical fashion. Then, the bottom-up approach is used to introduce the dynamic model starting from the SEDA concepts, and then the generic SEDA-cycle and sequential statements of iterative SEDA-cycles. On the other hand, a top-down approach is used to describe the three-layer structure from the top layer to the bottom layer.

2.4.2 Related SEDA Applications

We examined and evaluated three related applications discussed in Section 2.1 in this section.

1. WES ([33]):

A major shortcoming of WES is that it ignores the resource constraint, real time, in the SEDA problem solving. A common result of lacking any time management within a CS is that there is a much higher chance that the SEDA may not be accomplished satisfactorily within the available time.

One example would be that the WES proposes a "perfect" experiment and carries out the experiment by first generating the desired amount of data without interruption for an analysis. The experiment may be perfect, but it is at most
based on the available information up to the stage it was proposed. If the WES
does not know the data-generation speed and monitor the progress of data gen-
eration, then this experiment might exhaust all or most of the available time
for possible further data analysis or collection.

Even though the user might be aware of the time constraint during the data
generation, WES does not provide any information about the progress and
remaining time of the current data-generation for the user to make an immediate
decision. In other words, WES has no way to effectively adjust to the real time
constraint during the SEDA problem solving without any time management.

2. SESSA ([20]):

In addition to the same shortcomings mentioned above, another major short-
coming of SESSA is that it ignores the sequential nature of the SEDA task in the
SEDA problem solving; it only provides a collection of statistical procedures,
which is similar to the third layer of our SEDA model in Figure 2.

One problem of lacking such an embedded sequential SEDA process within
SESSA is that the user actually has to do most of the SEDA reasoning which
may be the most difficult part of the SEDA problem solving. Another problem
with providing only a collection of statistical procedures is that methodologies
are changing rapidly over time. Therefore, both the system and the user may
not be able to catch up with the changes since there is no embedded fundamental
structure for organizing the problems and the statistical procedures. As a result,
SESSA not only fails to provide appropriate help to the simulation users, but
also frustrates the users with so many statistical procedures.

3. EPSONA ([25]):

EPSONA is better than WES and SESSA in expressing the simulation output analysis at a conceptual level. At best, EPSONA is a representation like the three-layer static model in our SEDA framework. Nevertheless, EPSONA is simplified and vague in presenting the conceptual structure compared to our static model. Therefore, this comparison implies that the definitions of our eleven fundamental SEDA constructs matter since it is not clear what the dynamics of simulation output analysis are in EPSONA.

Also, like WES and SESSA, EPSONA ignores the real time constraint.

To sum up, our framework demonstrated that there are some potential problems in their SEDA problem-solving approach. Therefore, the result implies that our SEDA framework encompasses some essential SEDA primitive constructs which are missing in other CSs.
CHAPTER III

Issues in Computer-Assisted Simulation Experiment Design and Analysis

3.1 Introduction

We have designed and developed a prototype computer system (CS) for simulation experiment design and analysis (SEDA) to demonstrate the feasibility and appropriateness of our proposed conceptual framework for SEDA. This chapter addresses the important issues that occurred during the design of our prototype SEDA-CS.

Pressman [24] defines a computer-based system as “A set or arrangement of elements that are organized to accomplish some methods, procedures or control by processing information.” The elements of a computer-based system include software, hardware, people, database, documentation, and procedures. In our setting, the three major SEDA roles—the problem domain (SEDA), the people (user and SEDA expert), the computer systems (hardware and software), and their relationships—are the major elements and concerns.

Although the important issues we addressed may not be exhaustive, they represent our concern for an appropriate SEDA-CS. For each issue, we give a brief definition and a short reason for why it is an issue. Then we describe either how we resolved the issue or how we think the solution should look.
This chapter is organized according to a list of issues, including the program structure of the SEDA-CS (Section 3.2), knowledge acquisition methods (Section 3.3), system and user interface designs (Section 3.4), knowledge representation and reasoning mechanisms (Section 3.5), connections between the CS and simulation models (Section 3.6), time management (Section 3.7), data management (Section 3.8), procedure management (Section 3.9), and random number management (Section 3.10).

3.2 Program Structure of the SEDA-CS

A program structure consists of two major components: the knowledge and the control. A conventional program structure puts all of the functions, knowledge, and step-by-step sequences into an algorithm, while the artificial intelligence/expert system (AI/ES) structure often separates the knowledge from the control.

A program structure is fundamental because the design and development of a CS is usually based on this pre-specified structure. Moreover, the maintenance and lifecycle may be profoundly affected. Therefore, it is important to make an appropriate decision about the program structure at the early stage of the design and development.

We used the AI/ES structure in our prototype SEDA-CS for the following reasons:

1. Literature.

Some literature already justifies the AI/ES application as appropriate for SEDA [1, 16, 26, 30].

2. Guidelines.
There are many guidelines proposed for judging the appropriateness of ES applications SEDA [1, 16, 26, 30]. Based on Silverman's concrete guideline ([30]), SEDA meets four criteria, namely relevance, feasibility, optimality, and success to be an appropriate ES application.

3. SEDA framework.

Based on the conceptual framework for SEDA, we know that the expertise about procedures are changing rapidly over time. As a result, the knowledge needs to be updated frequently. In order to easily manage and encompass such a large set of rapidly changing expertise and map them into the simulation problem structure, the main components of the program structure, the knowledge and control, must be separated in an SEDA-CS application.

However, we do not limit the SEDA-CS to an ES application only, but strive for a useful system which adopts useful features and functionalities from other paradigms of CSs. For instance, one thing we did not do that might be useful is to integrate good features of a database and a decision-support subsystem into this SEDA-CS. The main reason is that an SEDA-CS often deals with huge data sets and provides up-to-date information for the user to make decisions during the SEDA problem-solving process.

3.3 Knowledge Acquisition Methods

Knowledge acquisition (KA) is an activity for obtaining domain knowledge required by a CS application from a human expert. KA activities are important because the
results are used not only to compose the domain-knowledge base, but also throughout the whole development of the CS application, including the design and analysis processes. Accordingly, the quality of the SEDA-CS depends heavily on the quality of the KA work.

We applied three strategies in the KA activities:

1. Combining different KA techniques for different types of knowledge.

Several KA methods were used throughout our research: concurrent verbal protocol analysis, structured interview, and concept sorting. Since each KA technique has its own advantages and disadvantages [31], no single technique can do the KA job well for a complex problem domain.


We used the case-oriented approach in our KA activities as much as possible to make the KA more effective.

Case-oriented approach helps the expert to focus on a small problem scope, and stimulates the problem-solving nature in articulating the expertise. This strategy helped us reduce some common problems in accessing the expert's knowledge [10].

3. Examining several representations for the expertise.

To help us smoothly conduct the KA activities, we represented the expertise in different formats, such as a rule-based representation, a frame-based representation, flow-charts, and a hierarchy of the problem space. This helped us to
raise questions and forced us to think hard about the issues revealed by these representations.

Representing the same knowledge or information using different representations demonstrates different perspectives of the domain expertise. This strategy not only allowed us to exercise the expertise for a deeper understanding of the problem domain, but also helped us dynamically design the appropriate KA process to fill the potential gaps.

In practice, verbal-protocol analysis has become one of the most useful KA methods. An example of verbal protocol sessions is that it can serve as data for a retrospective protocol analysis and observational studies in addition to the concurrent verbal protocol analysis [4]. In our study, verbal protocols helped us in exploring the problem domain and verifying assumptions during the KA activities. We only collected a few sessions of verbal protocols since both collecting verbal-protocol data and performing verbal-protocol analyses are time-consuming.

3.4 System and User Interface Designs

System and user interface designs are the results of integrating various components of a computer-based system. In our case, they are the people, including the users and domain experts, the problem domain (SEDA), and the computer systems, including the SEDA-CS, simulation models, and the computer hardware. While the system design represents the internal behavior of a CS, the user interface is the external functionality of the CS with which the user can interact. If the system and the user
interface designs can successfully cover important aspects for these different roles in the SEDA problem-solving process, then they provide a firm foundation toward a useful SEDA-CS for the users.

Four major tasks were performed to help the system and user interface designs:

1. The scenario.

To organize our thoughts before the design, we first briefly described the scenario in which the SEDA problem-solving occurs:

The human expert will first ask the user a few questions in order to understand some background information about the problem, the objective, the model, the simulation software, the computer, and other available resources.

However, the human expert can not obtain all the necessary information from the user for determining the right statistical procedures to solve the problem. So the expert may execute some pilot runs of the simulation model to know the simulation execution speed, validate some assumptions, obtain some rough statistics, and obtain some preliminary information.

While exercising the simulation model to collect the information, the human expert may obtain other related information through interacting with the user. By collecting the information from the user and the simulation model, the human expert understands the model and
the problem better and thus proposes a "best-guess" experiment design. After calculating the statistics and interpreting the results, the human expert and the user may need to continue the above process until satisfactory results are obtained.

Describing the scenario helped us to establish a baseline before designing an SEDA-CS. In addition, a side benefit is that the process of describing the scenario helped us to merge our thoughts for the design stage.

2. A user study.

In order to better design the CS, we would like to know the users. We have determined the user population, their background and capabilities, their errors and mistakes. We present some results about the process errors of simulation users as an example.

The process error of a simulation novice in the SEDA problem-solving process can be illustrated by the missing concepts from our SEDA framework:

(a) Time management: The novice does not know how to manage the resource, real time, effectively in the problem-solving process.

(b) Sequential nature of the SEDA task: The novice

i. either ignores data aggregation or aggregates too much (into just summary statistics) in the Design stage.

ii. seldom performs data transformations in the Analysis stage.
iii. is not skillful in generating statements from all possible re-
sults of analyses in the Analysis stage.

iv. is unable to correctly combine previous statements into a new
desirable statement in the Analysis stage.

v. seldom indicates or understand the scope of statements.

We conjecture that the novice seldom performs pilot studies to un-
derstand the systems better before performing the production stud-
ies because the sequential nature of SEDA task is missing in their
problem-solving model. In addition to process errors, there is a prob-
lem of misconception about the subjects, concepts, terminology, etc.,
between the user and the expert during their communications.

3. The cooperative style between the CS and the users.

The major categories of cooperative styles are fully-automated systems, function-
allocations systems, share-problem solving systems, and critique systems.

In our design of the SEDA-CS, the human expert is replaced by the computer
system in the scenario. However, we did not try to replicate the human expert
since a computer would not be as smart and flexible as the human expert. Thus,
we combined the function-allocation and shared-problem solving.

The users and the CS have to share problem solving since they have different
knowledge to accomplish the problem. Also, the function allocation strengthens
the usefulness of a CS by letting the user and the computer contribute what
they are good at: A computer system is highly capable of calculating, displaying results, and presenting alternatives with explanations, while a human user is highly capable of relating the results to the problem domain and making decisions based on other external environment consideration.

Notice that the reason why a man-machine cooperative system design is more appropriate than a completely automated system design is because the value of the user brings something to an SEDA problem-solving task: they enhance the SEDA problem solving with domain knowledge that the CS does not have, and they have the capability to override the automated analysis with the data they see in the graphical analysis.

4. The functionalities of the system at three levels.

A functionality is the feature of a computer system, which provides some function or achieves some task. Our approach is as follows: Identify a useful functionality, correctly implement the identified functionality, and make the functionality be usable and easily understood by the user.

One example is the time management functionalities provided in our SEDA-CS:

(a) A “Time Mgt” button is always available to the SEDA-CS user, including display clock time, display due time, display remaining time, and change due time.

(b) A “Time” button, which reports the current status of time and scope of sequential statistical results, is available only in some contents of the SEDA
session.

(c) The same time and scope report as in (b) is automatically popped up between important SEDA transitions, such as changing the subtask or system instance.

(d) Time-related messages are included in the scripts of the SEDA dialogue area when needed, such as a warning message about the shortage of available time to execute the current design.

As suggested by the protocol analysis, the CS and the user will cooperatively solve the simulation problem by providing their different specialties. One important issue about this cooperative problem-solving is that the user should be able to interrupt the CS, change the decisions, and acquire more information at any stage. This means that the CS might lead the SEDA task, but the user could truly participate in the SEDA task.

Our ultimate principle in designing a user interface for our SEDA-CS is a simple, rather than a simplified, user interface.

3.5 Knowledge Representation and Reasoning Methods

Knowledge representation is a formal way of encoding human knowledge, while reasoning methods are an internal mechanism for manipulating and processing information and data with the knowledge at the computational level.

Since the knowledge and the control are the main components of a CS program structure, the way that the knowledge is represented and that the control proceeds
greatly affects the performance of a SEDA-CS. Once the AI/ES structure is justified to be appropriate for the SEDA problem domain, the choice of the knowledge representation and reasoning methods becomes critical because they are the working components of the SEDA-CS.

Based on the cognitive task analyses performed during the KA activities, we decided to use both AI representation and non-AI representation for SEDA knowledge. Two AI representations, frame-based and rule-based representations, were used. The non-AI representation used was the algorithm for some step-by-step SEDA knowledge.

Although it becomes complicated to work within the hybrid representation scheme, the hybrid representation provides a rich and flexible way to execute the reasoning process. Figure 5 presents four representations (frames, rules, algorithms, and control mechanism), three reasoning methods (frame-base, rule-based and general, and hybrid reasoning), and their interactions, which are briefly explained in the four representation categories as follows:

1. Frame-based reasoning.

One built-in frame-based reasoning mechanism is hierarchical inheritance, which propagates the slots from a parent frame to its children frames. Also, through message passing a frame can invoke another frame and together they form a network of frames as the reasoning progresses. Another built-in mechanism is the demon which can execute the attached procedure of a slot whenever the slot is invoked or the slot value is changed, which is more effective and efficient than executing the same procedure via external reasoning.
Figure 5: Hybrid Representation and Reasoning Methods
2. Rule-based reasoning.

Rule-based reasoning is basically the chaining methods within the control mechanism. As mentioned above, frames provide a good way to index rules so that we can use either forward chaining, backward chaining, or both within a cluster of rules.

3. Hybrid reasoning.

Hybrid reasoning is that rules can retrieve information from frames and fill-in some related slot values. Also, rules may perform procedure calls to the algorithms.

4. Control mechanism.

The control mechanism contains the chaining metarules, overall control metarules, and strategic metarules. It can invoke rules, frames, or procedures. Strategic and overall control metarules are stored in this control mechanism to effectively and efficiently help the reasoning process.

Although not explicitly explained here, the knowledge representation in Figure 5 is based on the following criteria [6]: expressive adequacy, reasoning efficacy, primitives, meta-representation, incompleteness, and real-world knowledge. The primary criterion for reasoning methods is to “make appropriate and efficient use of the items in a knowledge base to achieve some purpose.... [2]”

Since no single representation can simultaneously meet all the criteria above, especially for a complex problem domain such as SEDA, we performed cognitive task
analysis as in the KA activities to carefully combine and customize several knowledge representations into one for the SEDA task. Then, the reasoning methods were developed based on this hybrid representation.

3.6 Connection between CS and Simulation Models

The connection between the CS and simulation models is a setup which establishes the communication channel between the CS and simulation models for the SEDA information, including input and output files for simulation models.

In an SEDA-CS environment, the connection between the CS and simulation models becomes important because they both are computer programs which are less flexible than humans and require explicit common specifications for information to be communicated. Moreover, human users who are also the information suppliers add more complications to this issue.

Fundamentally, our approach for the connection design is as in Figure 6. Both the CS and simulation models are computer programs, they have their own I/O specifications. On the other hand, the human user is more flexible without any formal specification. A common set of specifications is required to establish common ground between the CS, the simulation models, and the human user. Moreover, the human user should be able to access the specifications of the CS and simulation models since they are flexible in this interactive environment. Because computer programs are not as flexible as the human user, the context under discussion needs to be very specific in order to be efficient. Therefore, there is a resolver whose job is to retrieve the correct set of common specifications for the current problem domain and simulation
In our implementation of the prototype SEDA-CS, we assume that the simulation models are provided by the users. Therefore, the SEDA-CS has no direct communication channel with the simulation programs. The SEDA-CS can only change the input data files and get the output data generated by the simulation programs, which is considered as an indirect communication channel. On the other hand, the simulation user can either communicate with the SEDA-CS via its interface subsystem or prepared files.

3.7 Time Management

Time management is the arrangement and adjustment of the SEDA primitives according to time-related data or information during a SEDA problem-solving process.

Since time is the main resource constraint in our conceptual framework of SEDA, time management is crucial to the sequential SEDA problem-solving process. With good embedded time management in the SEDA-CS, the quality and resultant scope of the final results are improved.

Our basic strategy is to focus on the parts of SEDA problem-solving which consume most of the time resource: data generation and calculations, whose pre- and post-process are the experiment design and the progress monitoring. Also, the statistical procedures used in data analysis can greatly affect the time resource. Accordingly, a few necessary fundamental elements of time management are:

1. Sequential experiments. A sequential experiment is one efficient means to plan
Figure 6: Connection Design
each experimental cycle based on the data-generation time and some initial data summary statistics, such as means, variance, standard errors, and correlations. The experimental cycle repeats with the increasing knowledge of the simulation problem domain and the computing environment. Therefore, the overall experiment proceeds sequentially with potential efficiency because of the available information about the behaviors of the simulation model and computing environment.

2. Incremental data generation. Incremental data generation means to only generate necessary extra data in addition to what has been collected. However, few general-purpose simulation software packages provide this fundamental feature and it is impossible to do so without the cooperation of the simulation software.

3. Process Monitoring. Two processes whose progress we must monitor are the data generation and the calculations. Because data generation and calculation are usually the most time-consuming activities in SEDA, the SEDA problem-solving could obtain its best possible results by dynamically controlling these two activities. To do so, the progress of these two processes needs to be monitored for any prompt decision making during the SEDA problem solving.

4. Procedure substitution. During a data analysis, some procedures might be used iteratively to obtain the desired results. With the available time, some alternative procedures which require less time to obtain similar information might be considered. This may or may not be in the original design, but is
practical to comply with the time constraint.

Our implementation of time management incorporated all the above fundamental elements. Most of the implementation details were from the expertise obtained during the KA activities.

The fundamental problem with time management is that simulation practitioners either do not have it in their mental SEDA models or do not know how to incorporate it into the SEDA problem-solving even though they are aware of it. One good example is that simulation users often hear that a pilot study, which is part of the sequential experiments, is needed before performing a production simulation run, but seldom do they learn how to do pilot runs from the literature or in courses. One purpose for this pilot study is time management.

3.8 Data Management

Data management is the arrangement and manipulation of the simulation data for supporting different needs during the sequential SEDA problem-solving process.

Since data is one of the major SEDA components and huge data sets with different aggregations might be needed frequently, good data management can make the storage, access, acquisition, and manipulation of the data easier and more efficient.

Fundamentally, a data management design includes a data representation and a set of operations on the data. Several concerns in data management are the physical storage space, integrity, security, and access.

A representation of a data record specifies the types of information to be include
in a certain way. Accordingly, it affects the size of data, and the completeness or availability of the information. The set of operations specifies the actions that can be used upon the data. Accordingly, it affects the availability of data, data accessing time, and data security.

Operations can be classified into two major categories, the control operations and the aggregation operations. The control operations are those fundamental operations, such as search, read, write, and delete. The aggregation operations are those statistical and mathematical procedures used to aggregate the data for a certain purpose, which usually utilizes the control operations as well.

The approach we took is as follows:

1. Representations. One major difficulty in identifying the data representation is that it is domain dependent. Different problem domains, such as queueing networks and inventory systems, might have different sets of data items to be included, which may or may not overlap. Another related concern is data interpretation: even within the same problem domain, different simulation users might need different sets of data items. Therefore, the design of the data representation needs to be domain dependent, and balanced between diverse user groups.

We have identified some fundamental output data elements within our scope: replication number, observation number, simulation time when an observation is recorded, and observation of interest, such as queue time, queue length, and number of busy servers.
2. Operations. A fundamental operation is a procedure for manipulating the data that is required by at least one task. Once all the fundamental operations are identified to meet all the requirements of the simulation users, either these operations can be improved or new operations can be derived to increase the efficiency of certain tasks.

The control operations are mainly on data within a file, such as read, write, append, modify, delete, and search file I/O. The aggregation operations, in addition to the statistical procedures used in the SEDA-CS, are mainly on in-memory data: dynamically allocate and deallocate exact memory for the data set; trim the raw data set into a desired matrix; rearrange the data matrix; copy a partial or full data set; and convert the data type.

3. Concerns. As mentioned above, the concerns in data management are the physical storage space, integrity, security, and access. To meet the storage efficiency, the raw data generated by the simulation programs could be transformed into an intermediate format. The tradeoff is the storage space verses the precision of the intermediate data. The fundamental intermediate data format includes replication or batch number, and (mean) observation of interest.

To satisfy the data integrity, the data items need to be fundamental to deploy to all the required information. To maintain the security of the data, any operation on the data should be limited to one single task if possible and confirm with the simulation users about any important modification of the original data. Finally, as implied above, data accessibility means that the necessary information to be
included in the representation and ready to be used by data operations.

One important strategy used to increase the efficiency is to keep the frequently used aggregated data in the knowledge base as temporary meta-knowledge items. Accordingly, the data operation time can be eliminated after the first time it was generated and aggregated data can be updated when there is any change in the data set.

3.9 Procedure Management

Procedure management is the management of algorithmic step-by-step instructions that are used throughout the SEDA problem-solving.

Procedure management is important because any procedure that will be requested during the SEDA problem-solving process needs to be available and ready to use. Also, some procedures are evolving rapidly over time and thus the procedures contained in an SEDA-CS may change frequently. Accordingly, good procedure management can make the development and maintenance of the SEDA procedures more effective and efficient.

Procedures are all the algorithmic sequence of actions in the SEDA-CS, including the statistical and mathematical procedures, operations on the data, control algorithms, etc. The goal for procedure management is mainly to maintain necessary SEDA procedures up-to-date in the SEDA-CS.

Since some of the procedures are evolving over time rapidly, the main concern is the efficiency for building new or modified procedures with reasonable code efficiency.
Therefore, we considered two fundamental tasks:

1. Identifying new procedures or changes.

We worked closely with the SEDA expert to determine an initial set of procedures used in the SEDA-CS and kept the procedures current with the expert's expertise. The fundamental strategies in identifying procedures are the same as the KA activities in Section 3.3.

2. Building procedures.

Our approach is to provide a fundamental mechanism for composing every procedure based upon a set of primitive procedures that are evolving less rapidly over time. Thus, the focus is to identify those commonly used primitive procedures for building any complex procedures that are used in the SEDA-CS.

When there is a new procedure to be built or a modification in an old procedure, these primitive procedures are retrieved to compose a new procedure or replace old primitive procedures, with some minor effort to polish the connections between these primitive modules. The advantage of these fundamental primitive procedures is that they accumulate and require less effort to maintain over time.

Examples of primitive procedures are sum and sum-of-squares, mean, variance, standard deviation, standard error, correlation, inverse cumulative distribution function, maximum, minimum, median, quantile, sorting, residual, cumulative probability point, cusum, pooled variance, whisker length, and confidence intervals.
One tradeoff with the primitive procedures is the code efficiency. Because primitive procedures are available to compose or substitute complex procedures, there are cases when using several independent modular primitive procedures is much more costly than customizing one big chunk of code for the same task. Nevertheless, even the customized code needs to be modular, standard, and ready to be used by other procedures because this is part of the nature of primitive procedures.

3.10 Random-Number Assignment

Random-number assignment is the allocation of random numbers to different random processes in the simulation program(s) according to the requirements of a simulation experiment design. A random process is a process generated based on a statistical distribution with given random number(s). For instance, job arrival times can be generated from an exponential distribution with a series of random numbers. Random number assignment is important because it interacts with the statistical procedures used, and can promote statistical efficiency. We did not implement any random number assignment in our SEDA-CS. Therefore, we describe the potential problem, the proposed approach, and a sketch of the mechanism as follows:

1. Potential problems.

Traditionally, commercial simulation software provides either random number streams or random number seeds for its users to assign. In a typical simulation comparison problem, there are multiple system instances and each system instance has multiple random processes. Therefore, a simulation user has to
allocate or partition the random numbers for different random processes within different systems.

One potential problem is ineffective allocation of the available random numbers for desired data dependence among the random processes and across the system instances. For generating independent data, the random numbers used for the same processes on different system instances need to be different. For generating dependent data across system instances, this is difficult since synchronization is needed to make the same processes on different system instances use the same random numbers as much as possible. The difficulties are that (a) there might not be parallel random processes across system instances, such as comparing a 2-station tandem queueing network to a 3-station tandem queueing network; and (b) even for parallel random processes across system instances, the quantity of random numbers needed might differ dramatically, such as one system instance has an exponentially distributed job service process which only requires one random number, while the other system instance has an hyperexponential job service process which requires 2 random numbers. Accordingly, to make the synchronization work across system instances and other design factors, such as the number of replications and runlength for each replication, is incredibly difficult.

Since the total number of available random numbers is limited, the user also has to monitor the usage of random numbers and adjust the random number assignment as the sequential experiments progress. Usually, it is difficult for
the user to adjust the random number assignment to meet the assumptions behind their designs because the random numbers are under the control of the simulation software once the initial designs have been specified by the users as input.

2. Proposed approach.

Since the simulation user has no direct control of the random numbers and it is nontrivial to gain control, the fundamental solution is to let the simulation software handle the operations of the random number assignment while the user makes decisions only. This not only keeps the simulation users focusing on their tasks, but also lets each do what they do best. Under this approach, the simulation user only makes the decision about data dependence as before, and lets the simulation program handle the assignment, monitoring, and adjustment.

3. Sketch of one mechanism.

One mechanism to implement the above approach in a simulation software is called non-overlapping random number blocks. The basic idea is to divide the random numbers into many smaller non-overlapping blocks compared with the commonly seen random number streams. Then separate the consecutive random number blocks into bigger random number streams, as before, which are to be used for different processes. The simulation software can monitor the usage of the random number blocks within each stream and make adjustments of the blocks dynamically.
What is valuable with this mechanism is that the simulation software could have different strategies to effectively and dynamically control the assignment of these random number blocks. For example, some of the internal strategies could be:

1. A strategy to re-dimension the random number blocks and streams to meet the decisions of the simulation user.

2. A strategy to select the starting blocks for different random processes in order to be more flexible to handle different situations.

3. A strategy to optimize the synchronization across system instances with desired dependence on the output data with the same random numbers.

4. A strategy to effectively utilize the unused random number blocks of low-load random number processes when a higher load of random process runs out of available random number blocks.

5. A strategy to reuse other random number blocks or streams to reduce the potential problems for a certain experiment design.

This non-overlapping random blocks mechanism is flexible, but requires good strategies to make it work better for different SEDAs. The simulation user now only makes SEDA decisions on random number assignments with available strategies.
CHAPTER IV

A Formative User Evaluation for the Prototype SEDA-CS

4.1 Introduction

We have designed and developed a prototype computer system (CS) for Simulation Experiment Design and Analysis (SEDA). This chapter first describes and summarizes the prototype SEDA-CS and then presents a formative user evaluation. The purpose of this chapter is to demonstrate the feasibility and appropriateness of our conceptual SEDA framework upon which this prototype SEDA-CS is based.

This chapter is organized as follows: Section 4.2 presents the summary of this prototype SEDA-CS. Section 4.3 describes the setup for evaluating this SEDA-CS. Section 4.4 presents and discusses the evaluation results.

4.2 Descriptions of the SEDA-CS

We summarize the prototype SEDA-CS at three levels: the system-requirement level (Section 4.2.1), the system-design level (Section 4.2.2), and the system-implementation level (Section 4.2.3).
4.2.1 System-Requirement Level

The system-requirement level includes the task, the objectives, the intended users, the working environment, the source of expertise, the scope, the taxonomy, and the stage in the evolution:

1. The task. The only task is to solve simulation problems via experiment design and analysis in cooperation with the users. Stated differently, the task is to provide an overall management for the problem-solving process of SEDA.

2. The objectives. Perform qualified (valid) simulation studies, solve simulation problems within available resources (mainly real time), and provide an appropriate environment for learning the expert’s problem-solving approach via a computer-based simulation experiment design and analysis system.

3. The intended users. The simulation user population who has the knowledge of the problem domain, simulation modeling, and simulation programming, but not much of statistics. To be concrete, we assume a background of STAT 425, STAT 426, ISE 502, ISE 513, and ISE 554; or a background of ISE 703 and ISE 704 but without practical experience in applying statistics to SEDA. The users are assumed to provide verified and valid simulation programs for use by this SEDA-CS, and to know their computing environment very well.

4. The working environment. The overall environment is that there is no qualified statistician available and the computing environment is self-contained. The
computer in use is a DECstation 3100 workstation with a 16-inch color monitor, 600 MB disk space, and 24 MB main memory. The software includes the ULTRIX operating system, GNU C++, C/X11-based Motif graphical user interface, and some FORTRAN libraries.

5. The source of expertise. We have an available simulation expert with the knowledge of modeling, programming, statistics, and experience with different classes of problems. Our expert meets some common characteristics of an expert [14]: Effectiveness, efficiency, awareness of limitations, and versatility. Since the availability of the expert is restricted in providing his expertise in the SEDA, we have to further limit our scope for guaranteeing a successful development of a prototype CS: In addition to the scope described below, we limit ourself to use a pre-specified set of statistical procedures in the SEDA-CS problem solving.

6. The scope. This CS focuses on the means-comparison problem with known alternatives for the queueing-network class of systems. From the classification of simulation problems, we know that “comparisons with known alternatives” is one major part of the simulation problems. Under this branch, we focus on the univariate, expected performance, and means problems. With this scope of simulation problems, we further narrow our attention to the queueing-network systems, which is one of the major classes of stochastic systems.

7. The taxonomy. One common way to classify a CS is based on the type of task [27]. The task of our SEDA-CS is a combination of the classification, design,
and monitoring tasks. The classification task occurs at the beginning stage which classifies the given simulation problem, while the design task occurs in the remaining part of the design and analysis stages. Since the CS exercises the simulation program, it also monitors the status of the simulation in order to have a better control of the time and the amount of data in the whole problem-solving process.

8. The stage in the evolution. Waterman [36] provides five stages of evolution of a CS as follows: demonstration prototype, research prototype, field prototype, production model, and commercial system. Our expectation for this prototype CS is between demonstration prototype and research prototype. This CS solves a portion of the problem scope, suggests that the approach is viable and the system development is achievable, and displays credible performance on the problem, but may be fragile due to incomplete testing and revision. In other words, we want to test the conceptual SEDA framework via this prototype SEDA-CS.

4.2.2 System-Design Level

The system level of design includes the scenario, the man-machine cooperative style, the program structure, the knowledge representation and reasoning methods, the connection between the SEDA-CS and simulation programs, the time management, the data management, and the procedure management. Since most of the rationale behind this system design level is discussed in Chapter 3, we only briefly summarize
the "what" rather than the "why" aspects in this chapter.

1. The scenario. We have presented a scenario for typical simulation problem solving between a simulation user and a human SEDA expert in Chapter 3.

The human expert is replaced by a CS in our design. In order to be a useful SEDA-CS, this CS intends to adopt good features of available methodologies and technologies from computer science, cognitive engineering, statistics, and other fields, instead of only duplicating the human expert.

2. The man-machine cooperative style. We combined the function allocation and shared problem solving between the simulation user and this SEDA-CS.

The SEDA-CS and the simulation user cooperatively solve the simulation problem, as in the scenario, with their different knowledge and background. This SEDA-CS is intended to do calculations, display results, and present alternatives with explanations, while the human user is assumed to relate the results to the problem domain and make decisions based on other external environmental considerations.

3. The program structure. We used the AI/ES program structure to take advantage of the separation of the knowledge from the control. Therefore, it is easier to cope with the rapidly evolving statistical procedures and maintain the SEDA-CS up-to-date.

4. The knowledge representation and reasoning methods. According to the results of cognitive task analyses, we decided to use both AI representation and non-
AI representation for SEDA knowledge. Two AI representations, frame-based and rule-based representation, and one non-AI representation, algorithms, were used.

Several reasoning methods which were classified into frame-based, rule-based, hybrid, and control reasoning were used in this hybrid knowledge representation scheme.

5. The connection design between the SEDA-CS and simulation programs. We designed the connection between the SEDA-CS and the simulation programs based on the operating system in use: ULTRIX.

Because of the assumption that simulation programs are provided by the users, we do not impose a requirement for the user to code the UNIX interprocess connection since this is out of the assumed knowledge of the user population. As a result, a useful and powerful tool in establishing the communication channel, the UNIX interprocess communication, was not used. The communication between the SEDA-CS and simulation programs is via a set of common I/O specifications and some primitive UNIX system commands. The common I/O specifications were based on the idea of a resolver: the resolver retrieves an appropriate set of I/O specifications according to the problem domain and simulation software in use. We have determined a set of generic information items to be included in the I/O specification within our scope.
6. Time management. We focused the time management on the parts of SEDA problem solving which consume most of the time resource: data generation and calculations. Accordingly, we included sequential experiments, process monitoring, and procedure substitution in our overall time management for the SEDA task.

The idea of sequential experimentation is based on the SEDA framework. In practice, the real expertise from our SEDA expert has been refined and simplified to a certain scope for the demonstration purpose. Process monitoring is mainly to record the amount of data generated versus the real time consumed, which provides a run-time information for the user to make decisions concerning the time constraint when needed. Procedure substitution is a real-time procedure adjustment for the design to meet the time constraint with allowable precision of the desired information.

7. Data management. Our data management design includes data representation, and a set of operations on the data with concerns about the physical storage space, integrity, security, and access.

We have determined a small set of generic data items to be included in our scope. The simulation raw data is first written onto a disk file because we assume that the simulation programs are provided by the users and the SEDA-CS has no prior requirement about the data representation. We assume that the disk space is adequate for the data generated from one system instance at a time. Then data aggregation is applied to transform the raw data into intermediate data.
representation of smaller and more easily managed size.

Based on our scope, we have determined a set of control operations and aggregation operations on the data. Every operation on the data is limited to one single task if possible and confirms with the simulation user about any important modification of the data.

One important strategy used to increase the efficiency is to keep the frequently used, aggregated data in the knowledge base as temporary meta-knowledge items. Accordingly, the data operation time can be significantly reduced after the first time it was aggregated and the aggregated data can be updated when there is any change in the data set.

8. Procedure management. We have determined an initial set of procedures used within our scope for this SEDA-CS, such as the statistical and mathematical procedures, the operations on the data, and the control algorithms.

Our approach in procedure management is to focus on a set of primitive procedures that are not evolving rapidly over time for composing required procedures in our SEDA-CS. The primitive procedures were either written in C/C++ or borrowed from existing libraries, such as IMSL or some verified and reliable personal FORTRAN routines. We only applied this mechanism of primitive procedures to the statistical and mathematical procedures, and operations on the data.
9. The user interface. We designed a simple window-oriented, multitasking, and point-and-click graphical user interface (GUI) for this SEDA-CS. This GUI is more sophisticated than the traditional command and query interface and simple menu interface. But the implementation tends to be simple and practical, and avoids impractical decorations.

In principle, this user interface intends to provide consistency, offer meaningful feedback, confirm for non-trivial destructive actions, reduce the amount of information to be memorized in between actions, group functions and screen geography, provide context-sensitive help, forgive user mistakes, and seek efficiency in dialogue, motion, and thought [24].

This user interface, as the overall SEDA-CS, is an initial design and implementation. Continuous modifications are needed based on the feedback from the user evaluations.

4.2.3 System-Implementation Level

In this section, we provide a diagram (Figure 7) for describing the implementation of our prototype SEDA-CS, and the overall interactions between the simulation user, simulation programs, and this SEDA-CS. Our main SEDA-CS screen (Figure 8) and the main screen with a pop-up window (Figure 9) are also presented.

Structure of the SEDA-CS

As in Figure 7, our structure for this SEDA-CS includes five modules: GUI, knowledge base, control and reasoning, callback function, and utility and library. Each of these
Figure 7: SEDA-CS Structure
five modules has its own collection of components, and are interconnected for desired support.

The GUI module with different GUI components, such as windows and buttons, presents the information to the user and waits for the response from the user. Because of the AI/ES program structure, the knowledge base, and the control and reasoning are separated modules with their own components and responsibilities. The callback function module handles the user's response and provides different scripts for different purposes, such as feedback, on-line help, and questions for the user. The utility and library module supports all the procedures, such as the statistical procedures, data operations, and convenient functions, for this SEDA-CS.

The object-oriented design provides the modularity on top of these five modules. Therefore, although these modules are separated, they are coherent at a certain level in our implementation. The SEDA-CS structure presented in Figure 7 is a high-level scheme so that each module can be refined and developed as the requirements and design change.

**Overall Interaction between Different Roles**

Figure 7 also depicts the relationships between the three major roles in a typical SEDA problem-solving session: the simulation user, the simulation models, and the SEDA-CS.

Fundamentally, the simulation user and the SEDA-CS can interact with each other, while the simulation models can only take the request from the simulation user
and the SEDA-CS. This is because of the assumption that simulation programs are
provided by the user and the SEDA-CS can only use them as is.

The SEDA-CS and the simulation user have two-way communication because of
the GUI which remains on the screen throughout the problem-solving session. The
GUI displays a question on a dialog window and waits for the response from the
user. The user responds to the questions by either pushing one of the activated
buttons or typing the answer into a text field. Meanwhile, there are several generic
function buttons available all the time, including on-line help, tutor, display, time
management, and quit.

The user response, either an answer or a request, can be handled by the callback
function module which consults other modules for next movement. For an answer, the
callback function module consults the control and reasoning, and the knowledge base
for consequent internal actions and the next question, which might need the support
of the utility and library module. For a request, the callback function module consults
the control and reasoning, and the utility and library modules for consequent internal
actions, which might need the support of the knowledge base module. In either case,
the callback function module displays corresponding scripts on the GUI waiting for
the response from the user.

The above scenario repeats until the end of the problem-solving session. Therefore,
the interaction between the simulation user and the SEDA-CS are important to carry
out the whole SEDA problem solving.

The SEDA-CS has the knowledge and capability to understand the simulation
models and exercise the simulation programs well because Figure 7 meets our design requirements for the connection between the SEDA-CS and the simulation models. The data files of the simulation model comply with the I/O specifications imposed by the SEDA-CS, and the SEDA-CS has a resolver (in utility and library module) to retrieve the appropriate set of I/O specifications regarding the simulation software and the problem domain. In our implementation, we only considered the queueing-network problem domain and general-purpose programming software such as FORTRAN.

Since the simulation programs are provided according to the requirements imposed by the SEDA-CS, the simulation user understands the I/O specifications very well. As a result, the simulation user can directly view and modify the data files as desired. However, any modification should notify the SEDA-CS to avoid inconsistency.

The above interaction between the user, the simulation models, and the SEDA-CS describe how this SEDA-CS environment works based on our requirement and design of the prototype SEDA-CS in Section 4.2.1 and Section 4.2.2.

**SEDA-CS Screens**

Figure 8 is a screen dump of the SEDA-CS main screen remotely displayed on an IBM RS600 AIX workstation. On the left-hand side of the SEDA-CS main screen, there are three areas:

1. Function buttons and time display area. These function buttons (Help, Tutor, Display, Time Mgt., and Quit) are available all of the time during the SEDA-
CS consulting session. The time display area can display the options within the Time Mgt. button: current clock time, remain time, due time, or nothing. In Figure 8, 50 minutes of remaining time was displayed at the time this screen dump was performed.

2. Dialogue area. The upper part of the dialogue area displays the instruction from the SEDA-CS, while the button part highlights those options available to the user. The SEDA-CS was asking the user to determine whether the Q-Q plot matched the referential straight line on the right-hand side of the SEDA-CS main screen when this screen dump was performed. The user could only push button 1, 2, or 3. No What, Why, Time button was available for this part of dialogue.

3. Status area. That status area displays the last action performed by the SEDA-CS. The remaining time was the last action taken before this screen dump.

On the right-hand side of the screen, it is the diagram area. The upper part displays the diagram in question, while the bottom part displays the description of the diagram. The Q-Q (normality) plot for system instance 1 was displayed with the dots above the 45-degree referential line. The supplementary Q-Q (normality) Plot information was shown at the bottom of the Q-Q plot.

Figure 9 is a screen dump of the SEDA-CS main screen at the time that incremental data generation was performed for system instance 1. The pop-up window updated the current status of the time and data every 30 seconds, with an option to cancel the simulation data generation by pushing the Cancel button at the bottom.
Figure 8: SEDA-CS Main Screen
There are other occasions when the pop-up window is used, such as for the Time Mgt. button in the function button and time display area, or for Why, What, and Time buttons in the dialogue area. In this data generation case, the cursor was still an arrow (not a watch) which means the SEDA-CS was available to the user during the data generation (which was a child process spawned from the original SEDA-CS process). Also, at any time, the user can go to UNIX operating system to perform other tasks while SEDA-CS is busy. So the user could work on other tasks, such as writing a report or checking the email, while waiting for the incremental data to be generated.

4.3 Setup for Evaluating the SEDA-CS

We describe the goals and our expectation for this user evaluation in Section 4.3.1, the approach for the evaluation in Section 4.3.2, and a summary of the evaluation setup in Section 4.3.3.

4.3.1 Goals and Expectations for the User Evaluation

We have argued that our SEDA framework is a good one. Therefore, the main goal of this user evaluation is to demonstrate the feasibility and appropriateness of the SEDA framework. In addition, two supplementary goals are:

1. To know more about the users' perceptions and acceptance of the embedded SEDA constructs through the cooperative problem-solving process of the user and this prototype SEDA-CS.
Figure 9: SEDA-CS Main Screen with a Pop-Up Window
Due to limited time and manpower, a feasible way to measure the user's perception of the embedded SEDA constructs is to measure a user's understanding about the solution, recommendation, and results, which have an important relationship to the SEDA framework.

2. To understand more about the users' behaviors in this new SEDA problem-solving environment in order to improving both the SEDA framework and the SEDA-CS.

Fundamentally, the new problem-solving environment has the same roles as before, a simulation user, an SEDA expert, and a computer system, except that the human SEDA expert is integrated into the computer system as the SEDA-CS.

Since our motivation is to provide a framework for computer-assisted SEDA, by studying the user's behavior in this setup we might find important elements that are missing in our SEDA framework. Also, practical design and implementation details might be detected for improving the prototype SEDA-CS in order to push the SEDA-CS closer to the spirit of the SEDA framework.

The expectations are discussed based on the goals described above:

1. Since our main goal is to demonstrate the feasibility of the SEDA framework, we expect to see that the SEDA-CS helps the user to be successful in the SEDA problem solving. Notice that a successful demonstration does not necessary mean that every user will solve the same problem the same way with the same
result. Instead, we expect that the same problem may be solved differently, but reasonably efficiently due to different subjective decisions. Then we can claim that the SEDA framework really works in our evaluation setup.

2. We also expect the user to perceive and learn the embedded SEDA problem solving techniques, and thus the constructs, by examining their understanding of the results. A good result does not necessary imply that a user understands the problem-solving process and the way the result is presented to them. Therefore, we would like to interview the users to see how much do they understand about the result after this cooperative problem solving. By solving only one case, we do not expect that a user can fully understand the result presented by SEDA-CS. But it is a feasible way for us to know their interpretation.

3. This user evaluation is intended to be the first formal testing and user involvement. Therefore, we are only concerned about users' reactions and behaviors at the qualitative level rather than the quantitative level. We expect to see many ways for improving the design and implementation of this SEDA-CS. More importantly, we hope to see ways for improving the SEDA framework, although this is less likely than finding ways for improving the SEDA-CS.

4.3.2 Methodology

Based on our goals and expectations, we need:

1. the user to cooperatively solve a test case with the SEDA-CS,

2. the user to be interviewed for their understanding of the results, and
3. the user to think out of loud during the problem solving for the concurrent verbal protocol data.

Accordingly, the approach is to videotape the problem-solving process of each user (subject), interview the subject after the evaluation, evaluate the result, and perform the verbal protocol analysis. The setup for the user evaluation is summarized in next section.

4.3.3 Settings for the Formative User Evaluation

1. Evaluation Period. Two hours per case per user.

2. Two Cases:

   (a) Three system instances for the classic machine repair problem. This case is a time-independent parameter problem and thus includes the initial-bias subproblem. This is intended to be an easier problem for the one-hour problem-solving session.

   (b) Three system instances for the airline reservation system. This case is a time-dependent parameter problem. This is intended to be a more difficult problem for the one-hour problem-solving session.

3. Subjects. Graduate and undergraduate students with the desired simulation background. The graduates solve the easy case but with a broader range of simulation techniques, while the undergraduates solve the difficult case but with a narrower range of simulation techniques.
There are a total of 6 subjects, of which three are graduates and three are undergraduates. Different levels of simulation background are balanced within each group. Therefore, the total number of subjects might not be many, but they represent different levels of simulation background within our desired simulation user population.

4. Expert. SEDA-CS.


7. The scenario. A subject sits in front of the DECstation with a camera on their back focusing on the screen. Each session has three different evaluation stages: pre-evaluation, core-evaluation, and post-evaluation.

(a) Pre-evaluation stage. A 45-minute training session to introduce the purpose of this evaluation, the test case, the goal of this test case, and the roles of the SEDA-CS and the user. Also, the subject will practice thinking out loud by walking through the test case with the instructor to learn about the user interface.

(b) Core-evaluation stage. The problem solving proceeds up to one hour. The subject is assumed to know the problem domain well, but they may not
know the test cases provided by us. Therefore, we help the subjects with critical questions related to the test cases if necessary. Nevertheless, the subject can make some non-critical but important decisions. Also we help the subjects with some interface questions and problems to prevent the subjects from making fatal decisions and avoid the interface affecting the user's judgment about the overall system design.

Whenever there is over 30 seconds of unusual silence except during the data generation period, the subject will be reminded to continue thinking out loud.

(c) Post-evaluation stage. We interview each subject immediately after each problem-solving session for their understanding of the results, together with their perception and acceptance of the SEDA-CS environment. This interview can take about fifteen minutes or whatever time is remaining from the two-hour evaluation period.

Also, we evaluate and classify the results for measuring the feasibility of the computer-assisted SEDA framework. The verbal protocol analysis is performed to explore the behavior of the user under this new SEDA problem solving environment.

4.4 Evaluation Results

In this section we discuss the evaluation results, organized as results and interpretation (Section 4.4.1), SEDA-CS constructs (Section 4.4.2), and user behavior (Sec-
tion 4.4.3). We conclude this evaluation analysis in Section 4.4.4.

4.4.1 Results and Interpretation

There were six subjects: three graduate and three undergraduate students. Four of six subjects (3 undergraduate and 1 graduate subjects) successfully completed the evaluation, while the first two subjects did not complete the evaluation due to bugs in the SEDA-CS. However, the two who failed to complete the evaluation were close to finishing and their problem-solving processes are rich enough for the analysis. All the subjects performed thinking out loud adequately, although some subjects were silent for up to 5-6 seconds during some important decision-making processes.

The four subjects who completed the evaluation obtained conclusive and significant results, but two of them did not have a valid normality assumption because they did not finish the normality assumption check. They all chose the sample best system instance based on the summarized results at the end of the SEDA-CS session. For the two without a valid normality assumption, one pointed out that he was "not comfortable" with the recommendation and the other one said the recommendation "might be meaningless" due to the invalid normality assumption.

In general, all could interpret the results (statements) and make appropriate recommendations. However, they failed to utilize the data sets (scope of the statements) used to perform procedures to make further recommendations. When asked about their recommendations for future analysis with the scope of the statements, the undergraduate subject examined the data set that failed the normality assumption and said "Since there are 480 batches, I think I can batch them to see whether the nor-
mality will hold or not.... If it still does not, I will need to generate more data...." This is an illustration of what the scope is for: to enhance the recommendation with a qualifier, or to serve as a basis for suggesting possible future work with estimated resource requirements. The graduate subject, on the other hand, examined the data sets and complained about the summarized results not reflecting the normality results he concluded. In this case, the scope helped him to identify the discrepancy between his impression of the results and the actual results stored in SEDA-CS. The main reason for this discrepancy is that this subject was too quick to conclude normality without SEDA-CS’s confirmation.

All four subjects who completed the evaluation interpreted the results appropriately. With a little help, they utilized the scope of the statements to make more constructive recommendations, such as “instead of recommending system instance 1 without feeling comfortable, one option is to spend a little more time to validate the normality assumption, which should be feasible since we have 480 replications.”

4.4.2 SEDA Constructs

We present part of the evaluation results organized by the fundamental SEDA constructs which provide a natural coding category for our analysis.

SYSTEM, SYSTEM INSTANCE and PARAMETER:

All of the subjects seemed to understand the difference between system and system instance (specified via two parameters) when the test case was introduced. The SEDA evaluation sessions supported this observation since “system instance” was used throughout the SEDA-CS session, and some of them even checked back to the
system instance table for the parameter values during the SEDA session. For example, a graduate subject determined the "most congested system instance" for the classic machine repair system case based on the parameter values in the table, instead of the answer provided by the instructor.

DESIGN:

All of the subjects understood the designs (number of replications, simulation stopping time, and data batching and deletion) proposed by the SEDA-CS, although common random numbers was set as the default internally and not mentioned. However, two subjects were initially confused by the simulation time unit and the estimated simulation run time (real clock time). Also, one graduate subject indicated that he would prefer using the "number of observations" instead of the simulation stopping time in obtaining the execution speed so that he would know how much data he should have gotten. This is a strategic design in our prototype SEDA-CS since our SEDA framework includes simulation stopping time, but not the number of observations.

EXPERIMENT

"Experiment" is not explicitly stated in any written format in this SEDA-CS. Nevertheless, all the subjects seemed to know that when they agreed on a proposed design with an estimated real time, then they would get the data via this action of executing the simulation program. There was no evidence of any subject showing any problem with this "experiment" concept.

DATA
Although data is manipulated by the SEDA-CS internally, all the subjects seemed to understand what data they were getting based on the design and estimated real time. Also, every procedure produced statements with scopes, that could be retrieved in the time and scope report (by pushing the “Time” button) or is automatically popped up during important transitions such as switching system instances or sub-tasks. Five subjects read the time and scope report. Although most of them focused on the time report more than the scope report, three of them did carefully read the scope report with data information.

**PROCEDURE**

There are only a limited set of procedures provided for our test cases. Some procedures are only used internally by the SEDA to perform some verifications to protect the users. No subject complained about not having any particular procedure. However, graduate subjects tended to use their own choice rather than the default procedures. This is because graduate subjects had more simulation class training than the undergraduate subjects, and knew how to perform the analysis using the selected procedures. This is part of our design in providing defaults but allowing the users to change them. Also, it is obvious that the subjects were not worried about how complicated the procedures might be, how much effort they had to code the procedures, or how time-consuming those data manipulations were, because this prototype SEDA-CS has a ready-to-use procedure library. Thus, the subjects focused more on what they wanted to use rather than on what is available to use within those choices provided by SEDA-CS.
ANALYSIS

Although undergraduate subjects tended to just follow the route mapped out by the of SEDA-CS, all subjects had their own mental model for doing the analysis. Only one undergraduate subject confirmed the validity of the normality assumption using the alternative test procedures, the other two simply followed the default procedures all the time. On the other hand, all graduate subjects performed the analyses differently from the SEDA-CS during some part of the session. Part of the reason why the two graduate subjects failed to finish the evaluation is because they were confident about what they saw despite what the SEDA-CS warned and allowed. Thus, our fragile prototype SEDA-CS failed for them due to bugs in these non-critical test paths.

RESOURCE

There are three main resource constraints: real time, subject, and SEDA-CS. Real time is the main constraint that this prototype SEDA-CS is designed to address. This is accomplished by

1. A “Time Mgt” function button which allowed the users to display either current clock time, current SEDA due time, remaining SEDA time, or to change the due time at any point of the SEDA session.

2. A “Time” button for the users to review the time and scope report.

3. A time-scope report that is automatically popped up between important transitions, such as switching system instances within the same task, changing sub-tasks, or changing subproblem (initial-bias recognition and core comparison
problem).

4. A monitor window that is automatically popped up during a data generation period to provide current data generation status, including time and amount of data, with an option to cancel the simulation data generation.

5. Various time information provided during the SEDA sessions in the interactive dialogue area to make decisions.

Except for the first graduate subject who used this SEDA-CS, all the subjects used one or more combinations of these facilities to manage the real time. Although the time-management facilities were used by the subjects to manage the time, not a lot of words were spoken about how these facilities helped them. One undergraduate subject said, "It is nice.... I can expect what I am getting every 30 seconds (during simulation data generation)...." That the first graduate subject who did not do a good job in managing the time was partially due to the fact that the introduction and tutorial before the SEDA evaluation session was not conducted well by the instructor; it was improved for the other five subjects who did much better comparatively.

The subject and the SEDA-CS as resources worked well together in terms of making decisions (by the subject) based on the information provided (by SEDA-CS). However, there were some misunderstandings during the interactions between the subject and SEDA-CS, related to the system design and some user-interface issues. This will be discussed in Section 4.4.3.

STATEMENT
The statements explicitly presented to the SEDA-CS users are those results of statistical procedures. The other statements are only stored and used by SEDA-CS internally. All the subjects reviewed the statements during the SEDA sessions. However, the sequential nature of statements, which is an important concept in our SEDA framework, were not clear to most of the subjects due to the way the statements were presented to the users. Also, 2 subjects were confused about some of the statements. For example, one subject said, “Why are they (system instances) still biased? I gave the initial-bias deletion points....” The subject’s mental model was different from the SEDA-CS, despite the fact that the SEDA-CS did provide a “Why” button to explain the situations in which the subjects had trouble.

**SCOPE**

Scope is a very important SEDA construct that the simulation users often ignore. The subjects had no problem connecting a statement to the subset of data upon which it was based when performing a statistical procedure. Although the SEDA-CS utilized the sequential statements and their scope in providing information to the subjects, most of the subjects did not know how to take advantage of the scope in SEDA problem solving. This can be inferred from the way the subjects interpreted the final results and made the recommendations in Section 4.4.1.

### 4.4.3 User Behavior

This section presents and discusses the remaining evaluation results in terms of user comprehension of the data analysis.

**SEDA CONSTRUCTS**
We expected to see some improvements for our conceptual SEDA framework as well as the design and implementation of our prototype SEDA-CS. However, from these six evaluation sessions, there is no indication of any SEDA construct that is not covered in our SEDA framework as defined in Chapter 2. Section 4.4.2 illustrated that the subjects had no problems in accepting these SEDA constructs explicitly or implicitly. Although our SEDA constructs are self-contained in our SEDA world, it does not mean that there are no unidentified SEDA construct since we only had 6 sessions and no intermediate or expert users were involved in this evaluation.

**USER LEVEL AND SYSTEM DESIGN**

The graduate subjects are at a different user level from the undergraduate subjects. They tended to explore the SEDA-CS more than the undergraduate subjects even though our prototype SEDA-CS was not flexible enough to allow such freedom. As a result, the average performance of the graduate subjects was not as good as the undergraduate subjects in the current cooperative problem-solving mode. Since this prototype SEDA-CS was designed for the novice level, the results support the conclusion that if the system design is targeted correctly for its audience, then it will help the simulation users. On the other hand, our undergraduate subjects, to some degree, behaved like the graduate subjects in exploring the SEDA-CS, except with less confidence. So, if the system design is expanded toward the intermediate level, then we should see better and more consistent average performance for those graduate subjects. Also, even a novice will get better, so the system should accommodate them.
USER INTERFACE

Some problems encountered during the evaluation could be attributed to the inadequate user interface, which hampered the system design. The subjects' behaviors and expectations suggest features that should be incorporated into the user interface design:

1. The ability to see some graphical explanation for some of the data analysis or terminology. For instance, to illustrate when a data set is normal and non-normal from a histogram plot by using two diagrams is easier and better than to explain them in pure text.

2. The ability to undo the actions taken one or more steps before.

3. The ability to answer the question by clicking on the text.

4. The ability to use the mouse instead of text entry.

5. The ability to enter information without first clicking on the box if that box is the only choice the user has.

6. The ability to review previous diagrams.

MULTIPLE LEVEL OF ACTIVE INSTRUCTIONS

Although our prototype SEDA-CS provides basic functionalities for supplementary information, sometimes the subjects did not use them when it would have helped if they did. Also, it is difficult to present instructions to all subjects in a "flat" format, and yet meet the individual differences, such as patience, concentration, connection,
imagination, reasoning, etc. A multiple-level instruction mechanism allows users to select the depth of the instructions they prefer, while, an “active” instruction means a natural and intuitive representation for the users to find and use the supplemental instructions.

In our case, the SEDA-CS already reduced most of the dialogue to one screen-full at a time. However, some subjects understood the words but not the meanings. For example, a subject was trying to determine for herself whether the SEDA-CS statement “Do you want to continue to the next system instance for normality assumption checking?” meant she could move on, when she could have pushed the “Why” button for an explanation. However, even if she did, the flat-format explanation might not satisfy her question since it was intended to explain the reason why this screen of instruction was provided. An ideal, active multiple-level instruction would allow her to click on the part of instruction she did not understand. If the system provides any additional information for this particular instruction, a pop-up menu will show the options for her to choose what information to retrieve.

Another example is that a subject said, “No, I already know it is non-normal. I do not need to see the plot again” when asked, “Do you want to continue to the normality assumption for this system instance?” If the subject answered “Yes” the system would first batch the current data to see if there is any possible improvement, and then propose a design for incremental data generation. Although this detail was in the “Why” button, there was no clue to this subject that he interpreted the question incorrectly. With an active, multiple-level instruction, the question could
be expanded to point out the intention of the next move, including internal actions. Thus, the user could have been correctly led by a mechanism that enhances the capability and power of the instruction. With the "active" multiple-level instruction mechanism, the misunderstandings or bad communication between the subjects and SEDA-CS can be reduced.

4.4.4 Conclusions

Since we only studied 6 subjects, we are interested in a qualitative analysis. We are very satisfied with results of the 3 undergraduate subjects, and reasonably satisfied with the graduate subjects. Overall, we are confident that with appropriate system and interface design implementation, our SEDA framework provides a good basis for helping the simulation user.
CHAPTER V

Technical Issues, Contributions, and Future Work

We have demonstrated the feasibility of our SEDA framework with a prototype SEDA-CS. This chapter discusses the peripheral issues that arose during the design and implementation of this SEDA-CS in Section 5.1, and closes this research with the contributions in Section 5.2 and potential future work in Section 5.3.

5.1 Peripheral Issues

A peripheral issue is an issue that is not critical to demonstrate the feasibility of the SEDA framework in our implementation of the prototype SEDA-CS. Therefore, these peripheral issues are issues that could have better solutions than we could obtain in this prototype SEDA-CS. For each peripheral issue, we discuss how we solved it and how it could be done better.

This section is organized according to the peripheral issues: connection between the SEDA-CS and the simulation programs (Section 5.1.1), primitive procedures (Section 5.1.2), incremental data generation (Section 5.1.3), graphical user interface (Section 5.1.4), and memory management (Section 5.1.5).
5.1.1 Connection between the SEDA-CS and Simulation Programs

Since the simulation experiment in our SEDA-CS is performed sequentially and under a real-time constraint, good communication between different computer programs — the SEDA-CS and simulation programs — is important to achieve efficiency in SEDA problem solving. The current implementation of the connection between the SEDA-CS and simulation programs worked for our purpose, but improvements can be made based on the UNIX operating system in use.

Current Implementation

The assumptions are that users provide the simulation program, users are not required to code things other than the simulation models, and the SEDA-CS must be able to exercise and understand the simulation programs.

The current communication channel between the SEDA-CS and the simulation programs is indirect: the SEDA-CS can only read and modify the I/O files of the simulation programs via the I/O functions of the C/C++ language, and execute the simulation programs via some simple system calls to the UNIX operating system in use.

The C++ standard I/O, file I/O, and string I/O are adequate to manipulate the simulation input and output files. Specifically, fork (spawning a child process), exec (executing a simulation program), signal (UNIX signals), and system (executing a UNIX command) are adequate for the SEDA-CS to exercise the simulation programs.
However, there are improvements that can be achieved by applying the interprocess communication utilities and protocols of the UNIX operating system.

**Potential Improvements**

There are various utilities and protocols that we can use to improve the communication channel from indirect to direct inter-communication between the SEDA-CS and the simulation programs, such as a two-way pipe, a named pipe, sockets, or remote procedure calls (RPC).

In order not to impose coding the communication function on the user, the SEDA-CS has to provide a set of ready-to-use communication protocols for the simulation program to establish the direct communication channel. However, the work required to implement direct communication is substantial, and direct connection is not central to implementing our SEDA framework. But it is worthwhile to list the scheme for a direct communication:

1. Direct communication channel. The most important improvement is obtained by using the communication protocols of UNIX to establish a direct communication channel between the SEDA-CS and the simulation programs. Currently, the indirect communication channel allows the SEDA-CS to read and write simulation input and output files, execute and stop the simulation programs, and receive a signal from UNIX for a notification that a child process is dead. To sum up, it is more like a one-way rather than a two-way communication between
the SEDA-CS and the simulation programs.

A direct communication channel means that both the SEDA-CS and the simulation programs can talk to each other freely through an established communication channel of UNIX utilities or protocols. A simple example, such as a two-way pipe, allows both processes to read and write data to the pipes which connect them.

Therefore, with a direct communication channel the SEDA-CS and simulation programs can truly exchange information as desired.

2. Interprocess communication across the network. Another improvement is attained by using the communication protocols of UNIX, such as sockets, so that the SEDA-CS and the simulation programs can be running on different machines over the network.

Currently, we assume that our computing environment is a self-contained UNIX workstation. Although this self-contained environment is adequate, it is not the best computing solution under current computer technologies, especially for heavy computing needs such as SEDA. A network computing environment not only reduces the required hardware resources on each networked computer — such as main memory and disk space — but also helps to efficiently utilize the computing resources and relieve the time constraint.

An example using several networked computers to save time in generating huge simulation data sets while doing the SEDA analysis can be found in [11].
Using the communication protocols of UNIX essentially changes the current design of the SEDA-CS. Nevertheless, we expect it to greatly improve both the design and the performance of the SEDA-CS.

5.1.2 Primitive Procedures

A primitive procedure is our mechanism for procedure management in the SEDA task. The goal of primitive procedures is to provide a set of ready-to-use, independent, single-task functions for any computer language or software to easily compose desired SEDA-CS procedures. Although the current implementations is adequate in our narrowed SEDA problem scope, improvement can be attained with more procedures for different situations.

Current Implementation

Primitive procedures are an efficient mechanism for easing the SEDA procedure development. Our approach is to start with an initial set of SEDA procedures, decompose the initial SEDA procedures into basic elements, select common basic elements across SEDA procedures as the primitive procedures, and develop the primitive procedures.

The common primitive procedures we found were partially based on the decomposition of those SEDA procedures found in Figure 1. In practice, there are levels of primitive procedures based on our definition of a primitive procedure. The lowest level of primitive procedures are those procedures operating on the raw data or batch means, including read-data-file, sum and-sum-of-squares, sample average, batch average, sample variance, batch variance, standard deviation, standard error, correlation,
maximum, minimum, median, quartiles, and sorting. Other higher-level procedures, based on both the lowest primitive procedures and raw data, are cusum, pooled variance, whisker length, MCA bounds, MCB bounds, and MCC bounds procedures.

Potential Improvements

The approach we used to develop primitive procedures is a rough scheme which was adequate in our narrowed problem scope of SEDA. However, there are software engineering concerns, such as efficiency, ease of use, ease of maintenance, modularity, coherence, and independence, in using primitive procedures, which can be resolved by further refining the primitive-procedure development.

1. Efficiency. One major concern is the efficiency in applying these primitive procedures during the SEDA problem solving. Although each primitive procedure is intended for one task if possible, there are times when applying these one-task primitive procedures consumes much more time than a customized multi-task procedure. For example, sample average procedure and sample variance procedure can be separately called by a composed procedure when needed. Nevertheless, if the data set is large, then it is more efficient to apply one customized procedure which performs the sample sample average and sample variance in one loop instead of two separate loops.

On the other hand, there are further primitive procedures that can be developed to improve the efficiency when needed. For example, when performing procedures like sample average, sample variance and correlation, the sum of
the observations and sum of the square of the observations can be extracted as primitive procedures to be used by sample mean, sample variance, and correlation procedures. Therefore, as opposed to the multi-task procedure above, this refinement can achieve the same efficiency while maintaining the single-task modularity.

One comment on efficiency is that, typically, data generation might consume a larger portion of real time than the statistical calculations. However, with the effort in saving the raw data or aggregated data to be used repeatedly, calculation consumes an increasing portion of the real time in SEDA problem solving. Therefore, any easy and efficient improvement contributes to relieve the real-time constraint in SEDA.

2. Other software engineering concerns. Although there might be certain generic criteria for software engineering concerns, such as modularity, coherence, and independence, the criteria are not necessary universally applicable in every situation. The two solutions to the sample mean and sample variance example above is a good illustration of achieving the same efficiency from different directions for modularity and coherence. Another example is that if a function is performed several times in a procedure, then it can be extracted as a dependent high-level primitive which can only be used by that dedicated procedure. Also, for ease of use and maintenance, we might want to break a procedure into several smaller functions, or combine two procedures into one since a large part of the code is similar.
One final comment is that these concerns could lead to potential improvements for the management of the SEDA-CS software, rather than for the SEDA problem-solving.

All of these modifications are case-oriented and may not follow a generic guideline in software engineering. Therefore, a great improvement can be achieved by refining the primitive procedures via carefully observing and studying the SEDA-CS and its problem-solving process.

5.1.3 Incremental Data Generation

Incremental data generation is an important element in time management of our SEDA-CS to comply with the real-time resource constraint in the SEDA framework. The current implementation of incremental data generation in our SEDA-CS was made possible by modifying the self-coded simulation programs of the two simple test cases. Potential improvements could be made if the simulation software provides such a feature.

Current Implementation

In our implementation of the incremental data generation, we provided a mechanism in the self-coded simulation programs to support the two simple test cases in the user evaluation. The idea of the mechanism is as follows: Assuming that the system states that need to be saved to restart the simulation for incremental data generation are known, the simulation program saves them onto an external disk file when it
terminates normally. These system states can be retrieved by the simulation program when the next simulation run is needed.

The current I/O specification was only based on these two cases because there are too many system states to be considered for general queueing networks. A better solution is to let the simulation software do the job with the cooperation of its users.

Potential Improvements

System states include the internal system states of the simulation program and the external system states specified by a user. A simulation user has no way to access all the internal system states of a simulation program unless the simulation software provides such an interface. On the other hand, the simulation software can easily save its own internal system states and the external system states specified by the users, such as some critical flags.

In addition, it is nontrivial and ad hoc to design and implement such a mechanism in user-provided simulation programs for incremental data generation, especially since it also depends on the experiment design and analysis. Therefore, the simulation software should take the responsibility to ease the work of a simulation user. As a result, the improvement is a generic mechanism for incremental data generation rather than a case-dependent solution like the current implementation in SEDA-CS.

5.1.4 Graphical User Interface

The graphical user interface (GUI) acts as a coordinator between the SEDA-CS and the simulation user. A simulation user perceives the SEDA-CS via this GUI. There-
fore, an adequate GUI is needed to convey the simulation user's acceptance of this SEDA-CS, although GUI is not our research focus. The current implementation of our SEDA-CS uses the Motif GUI software to compose the user interface. Potential improvements can be made as the Motif software evolves.

Current Implementation

Currently, most GUI software provides the user similar functionalities and features for easing the significant effort in coding GUI applications. For our prototype SEDA-CS, using existing GUI software saved us time and effort in coding the GUI, and helped us focus more on coding the system design.

Although there is another standard GUI, Open Windows, competing with Motif in the UNIX world, Motif has become the most popular GUI standard on UNIX platforms. Using Motif in our self-contained DECstation provides a consistent GUI for the simulation users since it is the standard GUI for the ULTRIX operating system.

In our application of Motif, we only used existing Motif primitives, Widgets and Gadgets, including application shells; buttons; text; container widgets, such as frames, forms and rowcolumns; popup dialogs; scroll bars; separators; and drawing areas to compose the SEDA-GUI. Colors and fonts were mostly set up in a resource file which can be modified without re-compiling the programs. Responses to the input from the users were combined within the SEDA system design.

Although Motif primitives were helpful to ease composing our SEDA-GUI, we are limited to what Motif provides and other contributed Motif utilities. Therefore,
potential improvements can be made by either building our own customized widgets or obtaining the increasing amount of publicly contributed Motif libraries.

Potential Implementation

The main advantage of using Motif is the great flexibility of building customized widgets. However, it requires the knowledge of X-window programming and the internal specification of Motif, which is beyond our capability and time constraint on this prototype SEDA-CS.

As Motif evolves, more standard Widgets and publicly contributed primitives can be obtained easily, in addition to building our own Widgets. We discuss the potential improvements under this scheme (based on Motif 1.1.3 in use and our limited Motif related knowledge or information):

1. New Widgets. New primitive Widgets are included as Motif evolves. Therefore, improving the SEDA-GUI can be made easier by using new widgets. For example, one important Widget which has been seen in publicly contributed sites is the icon box. Currently, our SEDA-CS uses DECwindow’s icon box which is not flexible. A Motif icon box can be customized to fit into our SEDA-GUI as the door into the ULTRIX operating system.

2. Enhancement of old Widgets. Existing Motif Widgets can be made more useful if more features and options are supported. For example, multiple-color fonts in text fields. So far, Motif text Widget only provide one-color font. Although it does provide a highlighting feature, multiple-color fonts will provide a conve-
nient tool for displaying sophisticated text. For example, in one conversation
text Widget, we might want to use different colors to distinguish whether the
text originates from the SEDA-CS or the user.

Also, there is no easy way to compose a Hypertext-style text Widget now. That
is, a text Widget in which a user can click on a word for querying its explanation.

3. Variations of Widgets. In addition to enhancing the features of existing Wid-
ggets, variations of existing Widgets can provide another rich usage in SEDA-
GUI. For example, scrolled text field with button-like areas. So far, text widget
and buttons are separated primitives. A variation of a text widget is to provide
the possibility of casting a button-like area in a text widget so that the buttons
can be scrolled up and down with the text.

Since we only designed and implemented a narrowed scope of SEDA problem
solving, the desired Motif primitives were limited to what we needed. We expect
that more Motif primitives will be needed as the scope expands. Therefore, potential
improvements from Motif evolution are especially important to a practical SEDA-
GUI.

One final comment is that the above discussion should be applicable to generic GUI
software other than Motif since all GUI software supports similar basic functionalities
and features as Motif.
5.1.5 Memory Management

Memory management is a critical issue in designing large-scale software. It is especially true within the SEDA-CS since simulation data are read in and manipulated by various statistical procedures. Also data and aggregated statistics are stored as knowledge during the SEDA problem solving session. Currently, there is no memory management in our SEDA-CS. Accordingly, improvements can be expected if there is some.

Current Implementation

We did not build any memory management into our SEDA-CS. And C/C++ does not provide the garbage-collection features of other objected-oriented software. Therefore, the usage of existing memory may not be efficient. For a large-scale program with a limited problem scope, the SEDA-CS worked fine under some restrictions in the I/O and the design of the statistical procedures. For example, only intermediate data were stored and read into memory for use if possible. Also, some statistical procedures were designed to avoid reading large data into memory by using CPU time to read the same data several times.

Potential Improvements

Two sources of potential improvements are:

1. Build a memory management module to efficiently utilize the available memory.

   Typically, this requires knowing the available memory addresses and checking,
verifying, and reorganizing memory at run-time. It is feasible based on today's
technologies, but the tradeoff is the CPU time. For example, garbage-collection
in other object-oriented software provides a memory management feature but
presents such a tradeoff in CPU time.

2. Further design and develop more variations of existing statistical procedures.
By providing variations of statistical procedures, the SEDA-CS might choose
the one with the least memory requirement, the one with fastest CPU time, or
in between to increase the performance while using the limited resources.

5.2 Contributions

Our contributions are summarized as follows:

1. First comprehensive SEDA framework.

2. With eleven constructs, including two novel constructs (statement and scope),
we organized and defined an SEDA world with a dynamic and a static model,
which is self-contained, simple, specific only when necessary, and comprehensive.

3. We implemented a prototype SEDA-CS to demonstrate the feasibility and ap­
propriateness of such a framework. The most critical element, the real-time
constraint, is illustrated to show how the working environment and SEDA prob­
lem solving could be accomplished; this was missing in previous software in the
literature.
4. We enumerated important SEDA-CS design issues and presented initial research results.

5.3 Future Work

The emphasis of this research is on the statistical issues in the computer-assisted simulation working environment. Accordingly, the future work can either be to broaden the environment to assist simulation users with problem solving from problem formulation to SEDA to recommendations (Section 5.3.1), or to go deeper into design and implementation issues in applying our SEDA framework to an SEDA-CS for all levels of simulation users (Section 5.3.2). Also, the formative user evaluation can be expanded into a comparison experiment to analyze and demonstrate the potential benefits of our SEDA-CS versus other software tools (Section 5.3.3).

5.3.1 A Complete Computer-Assisted Simulation Working Environment

SEDA is one phase of a typical simulation project including formulation, programming, verification, validation, SEDA, and recommendations. To really create a proper and consistent working environment for simulation users, this SEDA framework needs to be integrated into the overall environment. The future research issue is how our conceptual SEDA framework can fit into various existing or experimental computer-aided simulation environments.
5.3.2 Statistical Issues in Designing and Implementing an SEDA-CS

With our experience in the design and implementation of a prototype SEDA-CS, we investigated and discussed some important issues in this dissertation. However, it is more practical and beneficial to provide more research results for a potential commercial SEDA-CS which is based upon our conceptual framework. Examples of future work include:

1. A SEDA tutorial system design,

2. A SEDA system design that is flexible enough to cover all levels of simulation users,

3. The use of multi-media technologies in all future SEDA work, and

4. More integrated-engineering effort for designing and implementing an experimental SEDA-CS that can be used in solving arbitrary simulation problems.

5.3.3 A Comparison Experiment for the Prototype SEDA-CS

Our evaluation of the prototype SEDA-CS was intended to be the initial formative user evaluation. After incorporating the improvements we learned from this evaluation, we need a follow-up comparison experiment to confirm the potential benefits for an SEDA-CS based on our SEDA framework.

We summarize the following experiment setup and possible measurements for future work:
1. **Experiment Setup.** The subjects solve the same test case under the following setups:

   (a) Our prototype SEDA-CS, which provides the subjects with an integrated SEDA problem-solving environment,

   (b) A WES-like CS which provides an integrated SEDA problem-solving environment without the resource management,

   (c) A SESSA-like CS which provides a library of SEDA procedures for each subproblem, and

   (d) The User's own familiar software tools.

2. **Experiment Objective.** The objective is to compare setup (a) versus each of the other three setups to demonstrate the potential benefits of setup (a), and to learn more about users' needs for further improvements.

3. **Measurements**

   (a) **Overall Measurement.** Does the subject solve the given test case with a justifiable solution?

   (b) **Hypothesis Test.** Each pair of comparisons has a hypothesis to be tested in this experiment:

      i. (a) versus (b): With the real-time constraint incorporated in our SEDA-CS, we expect setup (a) to better manage the real time constraint than setup (b).
ii. (a) versus (c): In addition to the above, with embedded SEDA problem-solving expertise in our SEDA-CS, we expect to see more thorough SEDA problem-solving in setup (a) than in setup (c).

iii. (a) versus (d): In addition to the above, with embedded data management and procedure management in the SEDA-CS, we expect to see the user in setup (a) focusing more on the SEDA problem solving, rather than worrying about the data management and availability of the procedures as the subjects in setup (d).

With these hypotheses in mind, we can easily generate questions for the coding categories for data analysis.

(c) Exploration of user behaviors. While the hypothesis test looks more on the positive side of setup (a) over other setups, this exploration looks for potential user needs that are not met, and for advantages from the other three setups. This helps to understand user behaviors and to enhance the prototype SEDA-CS and the SEDA framework.
BIBLIOGRAPHY


