A DATA ANALYSIS SOFTWARE TOOL FOR THE VISUAL SIMULATION ENVIRONMENT

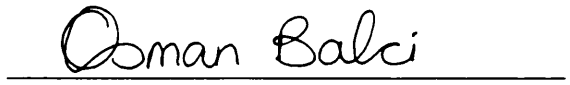
by

Ali Tuglu

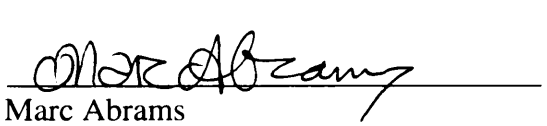
Project Report submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

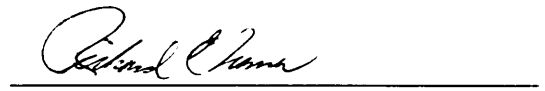
APPROVED:



Osman Balci, Chairman
Department of Computer Science



Marc Abrams
Department of Computer Science



Richard E. Nance
Department of Computer Science

June, 1995
Blacksburg, Virginia

A DATA ANALYSIS SOFTWARE TOOL FOR THE VISUAL SIMULATION ENVIRONMENT

by

Ali Tuglu

Osman Balci, Chairman

Computer Science

(ABSTRACT)

The objective of the research described herein is to develop a prototype data analysis software tool integrated within the Visual Simulation Environment (VSE). The VSE is an integrated set of software tools that provide computer-aided assistance throughout the development life cycle of visual discrete-event simulation models. Simulation input and output data analyses are commonly needed in simulation studies. A software tool performing such data analysis is required within the VSE to provide automated support for input data modeling and output data analysis phases of the model development life cycle. The VSE DataAnalyzer provides general statistics, histograms, confidence intervals, and randomness tests for the data sets. It can also create C modules for generating random variates based on a collected set of data. Furthermore, the VSE DataAnalyzer possesses the basic file management, editing, printing, and formatting functionalities as well as a complete help feature. It has been used in a senior-level Simulation and Modeling course, and the feedback from the students has been positive. New functionalities can easily be added to the VSE DataAnalyzer due to its object-oriented software structure. The VSE DataAnalyzer is yet another software tool created to provide more comprehensive automated support throughout the visual simulation model development.

ACKNOWLEDGEMENTS

I wish to thank Dr. Osman Balci for his endless support and guidance. Acknowledgements are also extended to Dr. Marc Abrams and Dr. Richard E. Nance for serving as committee members for this project.

I would like to thank Anders Bertelrud, Chuck Esterbrook and Michael Talbert for sparing their time to help me.

For their non-technical support, I would like to thank my mother, my family, and all of my Turkish friends, here in the U.S. and back in Turkey, and most importantly my father whom I always miss.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
INTRODUCTION

1.1 Statement of the Problem

The usage of simulation for solving problems is becoming more and more prominent as the problems become increasingly complex. Discrete event simulation is a technique for solving a large variety of problems in many disciplines including computer science, industrial engineering, operations research, management science, and systems engineering [Balci 1986].

The need for automated support in applying simulation for solving large-scale complex problems is undeniable. Such automated support is provided by the Visual Simulation Environment (VSE) which has been in development as part of the Simulation Model Development Environment (SMDE) research project [Balci and Nance 1992 ; Derrick and Balci 1992, 1995]. VSE provides integrated and computer-aided support for visual simulation model development throughout the entire model development life cycle [Derrick and Balci 1995]. VSE contains the following toolset:

- VSE Model Editor
- VSE Visual Simulator
- VSE Learning Support System
- VSE Evaluator

The list above does not contain a tool to provide automated support for the data analysis part of the model development. Therefore a software tool performing both general purpose and VSE-specific data analysis is needed to use in input data modeling and output data analysis.

1.2 Research Objectives

The objectives of the research described herein are to develop a software prototype which:

- produces general statistics for a given data set,
- enables the creation of histograms to visually examine the frequency distribution characteristics of a given data set,
- generates confidence intervals for a given simulation input or output data set,
- enables the application of a randomness test to a given data set,
- enables the creation of a module in C programming language for generating random variates based on a collected set of data,
- conforms to the NEXTSTEP user interface guidelines,
- is truly object-oriented with encapsulation, message passing, and inheritance,
- is simple to use and intuitive to the user,
- provides a comprehensive and detailed on-line assistance for the user, and
- is integrated with the VSE Simulator software tool so as to access the output data of a simulation experiment.

1.3 Report Overview

Chapter 2 gives the definitions of general statistics that are frequently used in simulation studies. Chapter 3 explains the design and implementation issues of the VSE DataAnalyzer (VSEDA) in depth. Object diagrams, implemented classes, and internal logic are given in this chapter to help the future researchers improve the software. Chapter 4 contains a very detailed user's guide explaining all the menus and features of VSEDA. All of the information in this chapter can also be found in the on-line help facility of VSEDA. Chapter 5 presents the conclusions and recommendations for future research.

Chapter 2
DATA ANALYSIS IN SIMULATION STUDIES

Data analysis is an important and natural part of simulation studies. Many types of data analysis are conducted in a typical simulation study. This chapter describes those types of data analysis provided by the VSE DataAnalyzer (VSEDA) software tool; namely, general statistics, histograms, confidence intervals, randomness test, and random variate generation based on collected data.

2.1 General Statistics

Several statistics can be used to get the basic information about the distribution of input and output data. Sample mean, sample variance, median, maximum value, and minimum value are the most useful statistics for this purpose [Walpole and Myers 1978].

2.1.1 Sample mean

If $x_1,x_2,...,x_n$ represents a random sample of size n, then the sample mean is defined by the statistic

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

2.1.2 Sample variance

If $x_1,x_2,...,x_n$ represents a random sample of size n, then the sample variance is defined by the statistic

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})$$

2.1.3 Median

If $x_1, x_2, \ldots, x_n$ represents a random sample of size n, arranged in increasing order of magnitude, then the median is defined by the statistic

$$median = x_{(n+1)/2} \qquad \text{if } n \text{ is odd}$$

$$median = \frac{1}{2}\left(x_{(n/2)} + x_{(n/2)+1}\right) \qquad \text{if } n \text{ is even}$$

2.1.4 Maximum value

Maximum value is the sample value which is equal to or greater than any other sample value in the data set. It is useful to determine the range of the data set.

2.1.4 Minimum value

Minimum value is the sample value which is equal to or less than any other sample value in the data set. It is useful to determine the range of the data set.

2.2 Histograms

A histogram is a graphical method used for displaying the frequency distribution of the data. The entire range of input data is divided into intervals, and a bar showing the number of the data values in each interval is drawn. The bar pertaining to the interval with the highest frequency is adjusted to fit to the maximum available length, and the rest of the bars are adjusted in length accordingly. Interval size and data range can be changed to get more precise graphics for particular purposes. A sample histogram, created by VSEDA, is displayed in Figure 2.1, where each bar has a different color.

2.3 Confidence Intervals

Confidence intervals (interval estimates) are preferred over sample means (point estimates) in presenting simulation output data. An interval estimate provides more

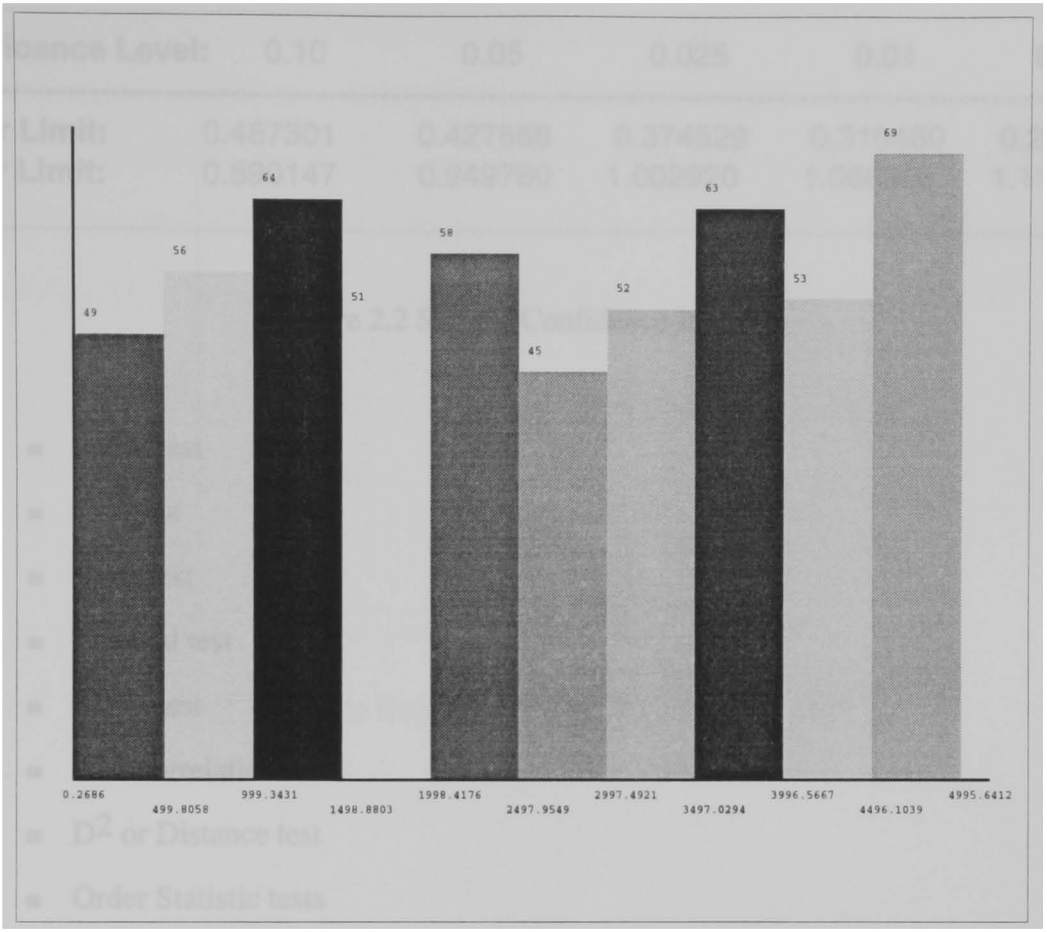


Figure 2.1 A Sample Histogram Created by VSEDA

information than a point estimate. A confidence interval, [a,b], is constructed with a confidence level 1-α, where α is called significance level [Schefler 1988]. It is interpreted that "we are 100*(1-α)% confident that the true mean value of the random variable of interest is contained within the interval [a,b]." Confidence intervals for five sample significance levels are shown in Figure 2.2.

2.4 Randomness Test

Many statistical techniques exist for testing the randomness of a given set of observations [Balci 1988]:

- Frequency test

Significance Level:	0.10	0.05	0.025	0.01	0.005
Lower Limit:	0.487301	0.427688	0.374529	0.310460	0.265290
Upper Limit:	0.890147	0.949760	1.002920	1.066988	1.112158

Figure 2.2 Sample Confidence Intervals

- Serial test
- Gap test
- Runs test
- Spectral test
- Poker test
- Autocorrelation tests
- D^2 or Distance test
- Order Statistic tests
- Yule's test

Here we describe the subjective and graphical technique [Nance 1995] used by VSEDA.

Given a set of observations, $x_1, x_2, x_3, \ldots, x_n$, a graph is constructed by plotting the values of $((x_1, x_{1+lag}), (x_2, x_{2+lag}), \ldots\ldots, (x_{n-lag}, x_n))$. The lag represents the distance between observations for measuring the dependency and is specified by the user. The graph is examined subjectively to determine if patterns can be identified indicating correlations. Figure 2.3 shows a random distribution of 500 values. On the other hand, patterns are easily observed in Figure 2.4, which shows a non-random distribution of 500 values. For both randomness tests, lag value of 1 is used.

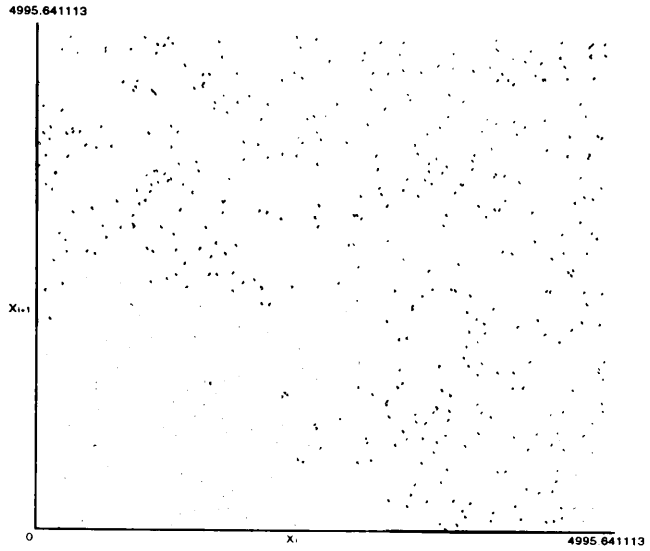


Figure 2.3 A Sample Randomness Test for a Random Data Set

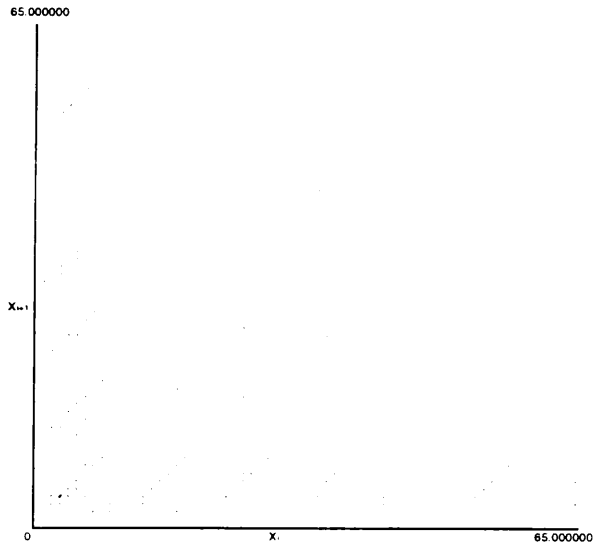


Figure 2.4 A Sample Randomness Test for a Non-random Data Set.

2.5 Random Variate Generation Using Empirical Data

Simulation input data modeling is commonly done by fitting a probability distribution to collected data by using a software product such as UniFit II [Law and Vincent 1994]. The probability distribution identified is used to generate random variates in modeling the system's input process. In those cases where no known probability

Data

```
1 2 1 1 2 2 28 10 15 4 4 13 23 2 15 12
8 20 1 11 2 3 5 6 1 2 8 2 2 14
```

Cumulative Probability Distribution

1.000000	0.033333
2.000000	0.066667
1.000000	0.133333
2.000000	0.200000
28.000000	0.233333
10.000000	0.266667
15.000000	0.300000
4.000000	0.366667
13.000000	0.400000
23.000000	0.433333
2.000000	0.466667
15.000000	0.500000
12.000000	0.533333
8.000000	0.566667
20.000000	0.600000
1.000000	0.633333
11.000000	0.666667
2.000000	0.700000
3.000000	0.733333
5.000000	0.766667
6.000000	0.800000
1.000000	0.833333
2.000000	0.866667
8.000000	0.900000
2.000000	0.966667
14.000000	1.000000

Figure 2.5 A Sample Cumulative Probability Distribution

distribution fits the collected data and the sample size is sufficiently large (>=40), empirical method of random variate generation is used.

VSEDA enables the creation of a module in C programming language to generate random variates based on collected data. The module is produced in two steps. In the first step, cumulative probability distribution, as shown in Figure 2.5, constructed by using the collected data. In the second step, the cumulative probability distribution is used to create a table lookup procedure in C programming language as the random variate generator. The random variate generator for the data of Figure 2.5 is shown in Figure 2.6.

```
double my_function(int randomStream)
{
   const double array[26][2]= {
    {0.000000,0.000000} , {1.000000,0.033333} , {2.000000,0.066667} ,
    {1.000000,0.133333} , {2.000000,0.200000} , {28.000000,0.233333} ,
    {10.000000,0.266667} , {15.000000,0.300000} , {4.000000,0.366667} ,
    {13.000000,0.400000} , {23.000000,0.433333} , {2.000000,0.466667} ,
    {15.000000,0.500000} , {12.000000,0.533333} , {8.000000,0.566667} ,
    {20.000000,0.600000} , {1.000000,0.633333} , {11.000000,0.666667} ,
    {2.000000,0.700000} , {3.000000,0.733333} , {5.000000,0.766667} ,
    {6.000000,0.800000} , {1.000000,0.833333} , {2.000000,0.866667} ,
    {8.000000,0.900000} , {2.000000,0.966667} , {14.000000,1.000000}
   }

   double next = nextRandomNumber(randomStream);
   int i=0;

   while((array[i][0] < next) && (i++<25))
   {
   }
   if (i)
       return (array[i-1][1] + ((array[i][1] -array[i-1][1]) *
        ((next-array[i-1][0]) /
              (array[i][0] - array[i-1][0]))));
   else return (0.0);
}
```

Figure 2.6 A Sample C module to Generate Random Variates by Using the Data in Figure 2.5

Chapter 3
DESIGN AND IMPLEMENTATION

This chapter presents detailed information about the design and implementation of the VSE DataAnalyzer (VSEDA). Components of the object-oriented NEXTSTEP development environment are first introduced. NEXTSTEP interface design principles are then presented. VSEDA development methodology and implementation strategy are described. The software structure and the classes are explained using object diagrams.

3.1 NEXTSTEP Components

The NEXTSTEP development environment is composed of the following layers [Nghiem 1993]:

- Mach
- the Window Server
- the Application Kit and Objective-C
- the Workspace Manager
- Project Builder and Interface Builder.

3.1.1 Mach

Mach, designed by researchers at Carnegie Mellon University (CMU), is the operating system of all NeXT computers. Mach is based on a simple communication-oriented kernel, and is designed to support distributed and parallel computation while still providing UNIX 4.3BSD compatibility [NeXT 1995]. The NeXT Mach operating system is a port of CMU Release 2.0, with additional features both from NeXT and from later versions of CMU Mach.

Mach consists of the following components [NeXT 1995]:

- A small, extensible system kernel that provides scheduling, virtual memory, and interprocess communications; the kernel exports a small number of abstractions to the user through an integrated interface.
- Operating system support environments that provide distributed file access, transparent network interprocess communication, remote execution facilities, and UNIX 4.3BSD emulation. Many traditional operating system functions can be implemented by user programs or servers outside the kernel.

3.1.2 The Window Server

The NEXTSTEP software development environment implements the client-server concept. The client is the program the user develops and the server is the Window Server which is a process working in the background. All the applications communicate with the Window Server to perform their tasks.

The Window Server has a Display PostScript interpreter, results in a WYSIWYG (what-you-see-is-what-you-get) environment. Unlike other environments, NEXTSTEP uses PostScript for displaying in addition to printing.

The responsibilities of the Window Server can be summarized as follows [Nghiem 1993]:

- Keeping track of processes and applications that own them to be able to manage message passing between processes.
- Performing all the drawing to the appropriate peripherals.

3.1.3 The Application Kit

The Application Kit provides objects for constructing interfaces and a framework that insulate the developers from handling events at the hardware level. As a result,

interface building and event handling tasks are facilitated in the NEXTSTEP environment [Nghiem 1993].

These two basic tasks of the Application Kit are given as follows [NeXT 1992a]:

- The Application Kit provides classes—most notably Window and View—that make drawing on the screen exquisitely succinct. Much of the work that's involved in drawing—communicating with hardware devices and screen buffers, clearing areas of the screen before drawing, coordinating overlapping drawing areas—is handled for the programmer.
- The Application Kit handles the events by using the Responder and the Application classes. The Responder class, from which many of the Kit's classes inherit, defines a mechanism by which the user's actions are passed to the objects in the application that can best respond to them. The Application class, which inherits from Responder, establishes the low-level connections that makes this system possible. It provides methods that inform the application of watershed events, such as when the user makes the application active and inactive, and when the user logs out or turns off the computer.

3.1.4 Objective-C

Objective-C is an object-oriented programming language that is a superset of the C programming language. The original version was developed by the Stepstone Corporation. NeXT has produced a version of Objective-C that is modeled closely after Stepstone's version. There are small differences between the Stepstone implementation and the NeXT implementation [Pinson and Wiener 1991]. The entire Application Kit is implemented by using Objective-C.

Objective-C is derived from the new ANSI C standard. Objective-C adds message passing and class definition syntax to ANSI C. It provides support for encapsulation, inheritance, and polymorphism. The NeXT version of Objective-C is a compiled version

of the language that runs only under the NEXTSTEP operating system. All classes in Objective-C inherits from a root class called Object. Multiple inheritance and operator overloading, which are usually available with some other object-oriented languages, are not supported.

3.1.5 The Workspace Manager

The Workspace Manager lets the user navigate the file system and manipulate the files and directories therein. It is automatically started after login. By clicking on a document of any type, the application incorporating with the document is activated automatically as a result of the communication between the application and the Workspace Manager [Nghiem 1993].

Workspace Manager's Inspector panel gives additional information about a selected item, for example, file ownership and size, applications capable of opening the file, and file permissions [NeXT 1995].

3.1.6 Project Builder

The main task that Project Builder performs is the coordination of the development task. It manages the components of the application, and gives the developer access to the other development tools [Nghiem 1993]. The project is the organization unit for the Project Builder. Following files are maintained under a project:

- class files
- header files
- interface files
- image files
- help files
- libraries.

3.1.7 Interface Builder

Interface Builder is used to develop the interfaces, whereas Objective-C and the Application Kit are used to implement the application. Interfaces can be graphically designed and tested by Interface Builder. It also allows the developer to create and inspect objects [Garfinkel and Mahoney 1993]. Interface Builder works together with Project Builder so as to provide a framework that can be improved by the developer by adding application-specific code. Interface Builder is more than a prototyping tool, it is actually an integral part of NEXTSTEP software development environment.

3.2 Development Methodology

Evolutionary prototyping, as illustrated in Figure 3.1, has been used in the development process. This methodology involves the development of a series of prototypes that gradually evolve to the final product. In evolutionary prototyping, product development is accomplished as a part of learning and growth. It implies that one does not know the complete requirements at the beginning, and the experimentation with an operational system is needed to learn it [Balci 1993].

One of the most important differences between the waterfall model and evolutionary prototyping is that integration is an internal part of evolutionary prototyping; whereas in the waterfall model, integration is performed right after each part of the application is implemented. Moreover, requirement specifications may never be written in this approach. As a result of the absence of a requirement specification document, judgment of whether the final product is produced or not is a subjective decision [Balci 1993].

3.3 NEXTSTEP Interface Design Principles

The NEXTSTEP user interface is designed by taking care of several important human-computer interaction issues. Four of them are emphasized specifically in the

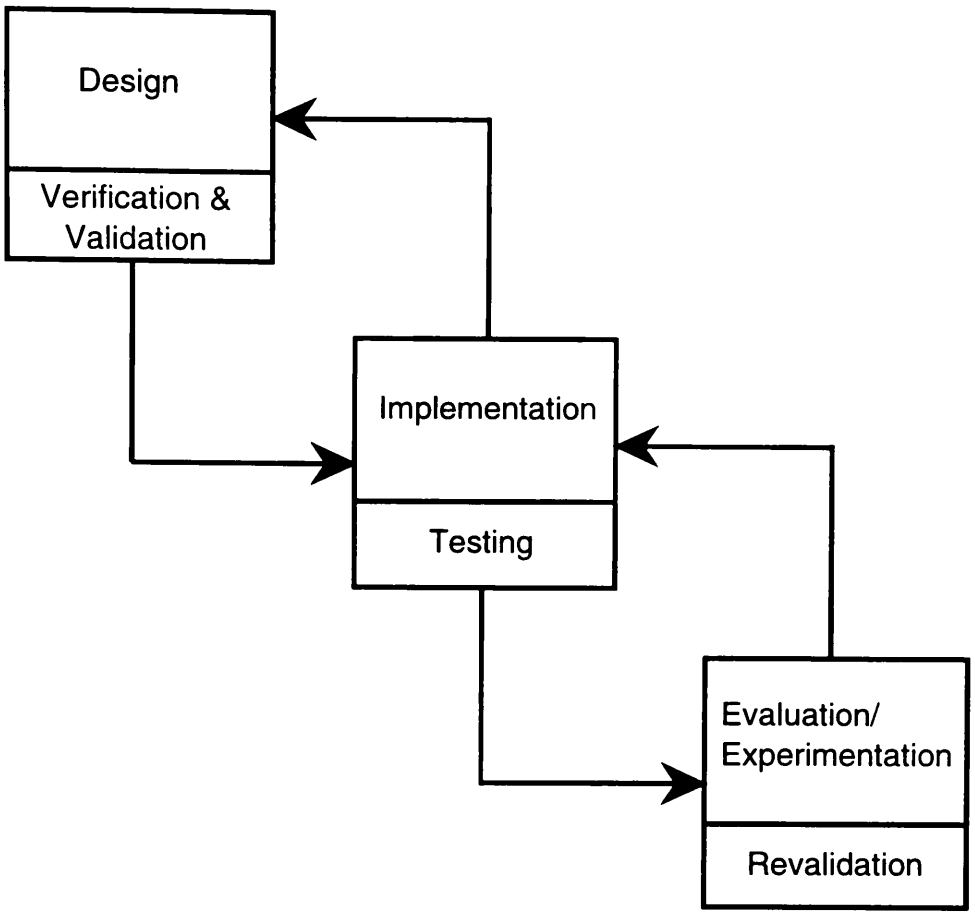


Figure 3.1 Evolutionary Prototyping

NEXTSTEP User Interface Guidelines [NeXT 1992b]:

- The interface should be consistent across all applications.
- The user is in charge of the workspace and its windows.
- The interface should feel natural to the user.
- The mouse (not the keyboard) is the primary instrument for user input.

Consistency makes applications easy to learn, and also increases the likelihood of its acceptance and use. Users can predict the possible results of their interactions, and thus feel confident in using different kind of applications.

The workspace and the tools for working in it belong to the user, not to any one application. The user has the freedom to choose which application they work in, and to rearrange windows in the workspace as they wish.

It feels natural to the user to use graphical user interfaces. The screen becomes a visual metaphor for the real world, and the objects it displays can be manipulated in ways that reflect the ways familiar objects in the real world are manipulated.

Graphical objects are used to represent all aspects of the user interface, and are operated mainly by mouse, not keyboard. The keyboard is principally used for entering text.

3.4 Conceptual Design

3.4.1 User Interface Design

One of the technical challenges in creating VSEDA has been to make the user interface as intuitive as possible. This has been managed by classifying the actions into two categories and placing them in different locations. The first kind of actions are the ones that manipulate the physical data, and the second is the ones that interpret the data.

First kind of actions are the actions for file management, available in the main menu. Depending on the available targets, they apply to the appropriate windows. The following is the kind of actions being classified in this category:

- Opening an existing data file
- Creating a new data file
- Saving data files, and statistics
- Ignoring changes on data files
- Closing data files and statistics
- Editing data files
- Manipulating windows opened by the application
- Printing the contents of the windows

- Applying some services provided by other applications to the open windows

Second kind of actions directly apply to the contents of the windows. These kind of actions are placed in pop-up menus on the windows they apply. If there is only one action applicable to the window, then a button is used.

3.4.2 General Issues in Class Design

The main purpose when designing classes is to create the necessary number of classes. For all NEXTSTEP applications, an application delegate class is created to manage the generally applicable functionality and the menus. Another class may be created to manage the data files, and services applying to them. The statistics on data files can be interpreted as services on that data. Each service involves a few similar and several different characteristics. Creating a class for each service is also a good idea. However, these services can be grouped among themselves, and this lets us use abstract base classes for each different group, and subclass these base classes to create classes managing each service. Although it can be done in Objective-C, conceptually, abstract base classes are not instantiated, and only serve as a base for creating subclasses.

3.5 Implementation Strategy

Several NEXTSTEP components and applications are used in the implementation of VSEDA. The interface Builder, Project Builder, and the Application Kit are the NEXTSTEP components which are used extensively throughout the development. The Edit application is also used to create and modify the classes, the header files, and the help files. Figure 3.2 illustrates the development process.

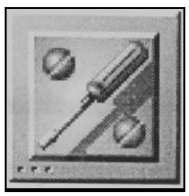
The first step in developing an application in NEXTSTEP environment is to create a new project in the Project Builder. In the next step, interface files are created by using the Interface Builder. Interface Builder provides several tools to be able to use the Application Kit objects very easily. The attributes of the graphical images can be changed

and tested in the Interface Builder. One of the most important features of the Interface Builder is that classes can be created by using the entire Application Kit class hierarchy. The skeleton for the classes and the header files are created in the Interface Builder if this approach is taken. On the other hand, the developer can also create the classes and the header files before creating the interface files, and link them in the Interface Builder. In general, creating the interface files first also helps with the class design issues such as what kind of methods to implement and whether the delegation is needed or not, and this approach is pursued throughout the development of VSEDA.

The class and header files that are created in the Interface Builder do not actually contain the code for the methods. Therefore classes are implemented by using the Edit application. Both Interface Builder and the Project Builder allows the developer to access these files through their browsers.

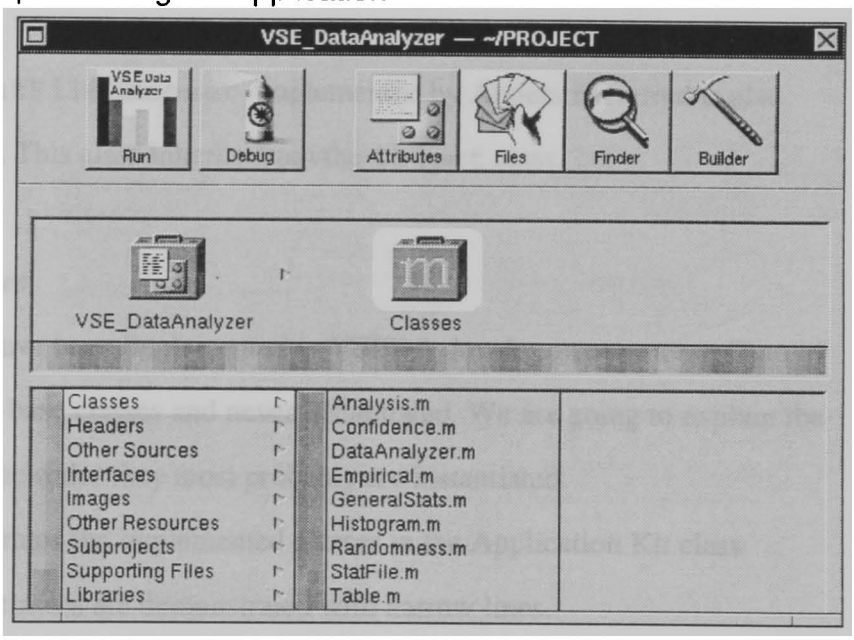
Project Builder can also add a help directory to the projects. This adds a "Help" menu option in the "Info" menu. The help directory created by the Project Builder includes the default help items consisting NEXTSTEP-specific information. This includes general information about the usage of menus, mouse, keyboard, etc. and an index for the entire help directory. Normally, the developer is expected to modify this directory to be used by the application. Edit application is the right tool to modify the help directory, because the entire help directory is based on the concept of links between the help files, and Edit application is capable of putting links between files. The files created can be either in Rich Text Format (RTF), or RTF with Directories (RTFD) format. Files with ASCII format can be modified in Edit by preserving their format. RTF is the standard format used by the Edit application, allowing the use of different fonts, styles, sizes, and color in the text. RTFD is a file package consisting of RTF text as well as images.

1.  Creating a new project

2.  Designing the interface

3.  Designing the classes

4.  Implementing the application



5. 

Debugging the application

Figure 3.2 NEXTSTEP Software Development Process

Moreover, Project Builder allows the compilation of the applications for debugging. Another application, usually a general purpose debugger, is activated for this purpose.

Using the Project Builder an application software can be compiled to execute under different architectures; namely, Motorola, Intel, Sun SPARC, and Hewlett-Packard PA-RISC.

3.6 Implementation of Classes

3.6.1 Frequently Used NEXTSTEP Classes

Several Application Kit classes are extensively used throughout the coding process. These classes are: `Window`, `Text`, `ScrollView`, `View`, `TextField`, `Form`, `Button`, `Matrix`, `Scroller`, `Responder`, and `List`. Detailed information about these classes can be obtained from NEXTSTEP General Reference Volume 1.

Another class, `StatFileChooser`, implemented by Anders Bertelrud is also used in the development. This class inherits from the `Object` class.

3.6.2 Implemented Classes

Nine new classes have been implemented in VSEDA development process. Two of these classes are abstract base classes and never instantiated. We are going to explain the implemented classes in the order they most probably are instantiated.

Figure 3.3 demonstrates the implemented classes in the Application Kit class hierarchy. Implemented classes are demonstrated with narrow lines.

3.6.2.1 DataAnalyzer class

This class has been designed to manage the menus, and to behave as a delegate for the application. Several menu options related to different services and files are first passed to the object instantiated from `DataAnalyzer` class, and then appropriate

messages are sent to the related objects. This class inherits from the `Object` class.

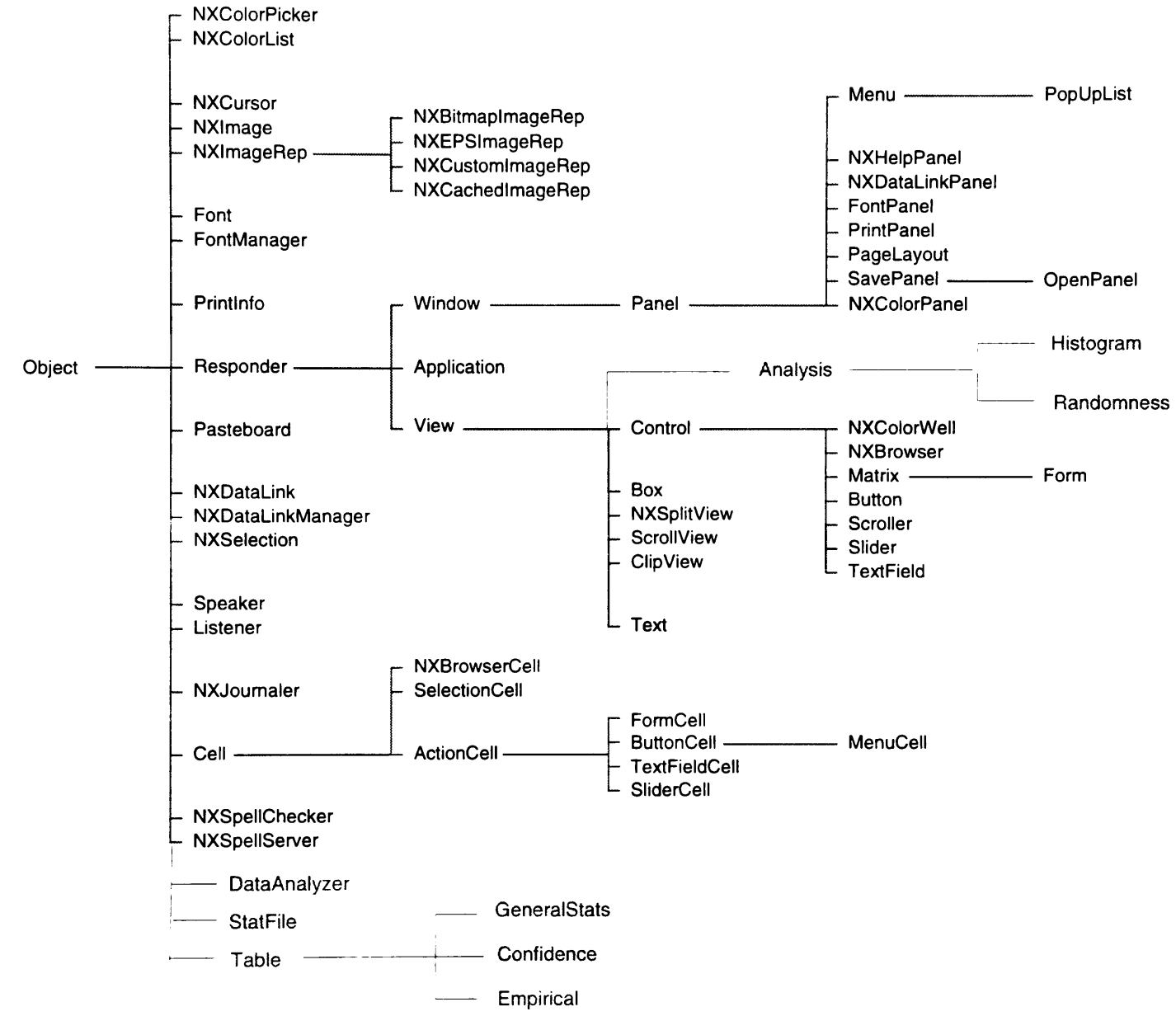


Figure 3.3 Application Kit Hierarchy with Implemented Classes

Delegation is a methodology to handle the events. Delegation extends the functionality of a class without subclassing. It can be thought as a form of message

forwarding. When certain events happen or after these events have occurred, it is usually expected that additional processing is needed. Delegation feature allows the designer of an application to add his/her own customized behavior for the desired events [Nghiem 1993].

An object of this class is instantiated when the interface file for the application is loaded. Interface file also defines the newly created object as the delegate of the `Application` object.

3.6.2.2 StatFile class

Every time a data file is opened, an object of this class is created to handle the services applicable and to manage the editing in the `Text` object. The interface file of this object has a text window together with a pop-up menu consisting of the services for the data displayed in the `Text` object. Selection of any of these services results in the creation of a new object to handle that specific service. The `StatFile` objects also behave as the delegates of their `Window` and `Text` objects. This class inherits from the `Object` class.

3.6.2.3 Table class

This is an abstract base class that is designed to provide basic functionality for its subclasses which are used to present statistics in tabular format. This class inherits from the `Object` class. The most important method in this class is the `prepare::::` method which is used to read the content of the text window in the `StatFile` object, interpret it by checking according to the data file syntax, and give error messages for the strings which can not be interpreted. If the data is error-free, then the numeric values are placed in an array to be used by subclasses. A static array is used instead of a linked list to store the numeric values because of large data files with high number of samples. The default maximum size of the array is 50000 numeric values with double precision. The

array is a protected variable which can only be accessed by the subclasses of the `Table` class.

3.6.2.4 GeneralStats class

This class inherits from `Table` class which is an abstract base class. It is designed to provide general statistics about the data passed to the object which is instantiated from this class. The data is passed to the object of this class as a stream, and it is interpreted by the `prepare:::` method of the `Table` superclass. The `calculate::` method of this class is used for calculating all the statistics and displaying them.

3.6.2.5 Confidence class

This class is designed to provide confidence intervals for the mean value of the data that is passed to the object instantiated from this class. This class inherits from `Table` superclass. The data is passed to the object of this class as a stream, and it is interpreted by the `prepare:::` method of the superclass. Calculations and displaying the results are performed by the `calculate::` method.

3.6.2.6 Empirical class

This class is designed to create the cumulative probability distribution for the empirical data and a C module generating random variates by using the empirical data. The C module can be compiled by the VSE Simulator to be able to use the empirical data in the case that input data does not fit to any kind of known distribution. The data is passed to the object instantiated from this class as a stream. This class inherits from `Table` superclass. `Empirical` class overrides the `print:` method which is implemented in `Table` superclass. Two important methods are implemented in this class: `calculate::` and `create_C_Module:`. The first method is used to create the cumulative probability distribution table, and the latter is for creating the C module.

3.6.2.7 Analysis class

This is an abstract base class providing basic functionality for its subclasses. Because of the fact that its subclasses deal with graphics, `Analysis` class inherits from `View` class. Due to the same reason that is mentioned in the design of the `Table` class, static memory allocation, allowing to store maximum 50000 numeric values with double precision, is used to store the values to be used by subclasses. The `prepare:::` method is used to translate the content of the `Text` in the `StatFile` object to numeric form, and to store the values in the array. In the case that the data stream can not be translated to the numeric form, appropriate error messages are displayed to help the user make the corrections.

3.6.2.8 Histogram class

This class is designed to provide the basic functionality for drawing, printing, saving histograms for the purpose of demonstrating the frequency distribution in data files. It inherits from `Analysis` superclass. `drawself::` method of `View` class is overridden to be able to draw in the `CustomView` by using display-postscript. All drawings are performed by `drawself::` method. `drawself::` message should not be sent directly, instead `display` message of `View` class can be used to activate this method.

3.6.2.9 Randomness class

This class is designed to perform randomness test on the data. Randomness test is a visual and subjective method for displaying the correlation among the sample values. `Randomness` class inherits from `Analysis` super class. All drawings are performed in the `drawself::` method as it is implemented in the `Histogram` class.

3.7 VSE DataAnalyzer Software Structure

3.7.1 Object Diagrams

Object Diagrams are used to provide a formal graphic notation for modeling objects, classes, and their relationships to one another. They can be used for abstract modeling as well as for designing actual programs. Object diagrams are concise, easy to understand, and work well in practice. There are two types of object diagrams; class diagrams and instance diagrams [Rumbaugh *et al.* 1991].

A class diagram can be defined as a schema, pattern, or template for describing many possible instances of data. A class diagram graphically describes object classes.

An instance diagram is used to describe how a particular set of objects relate to each other. An instance diagram graphically describes object instances. Instance diagrams are useful for documenting test cases and discussing examples. They are generally used to show examples to help clarify a complex class diagram.

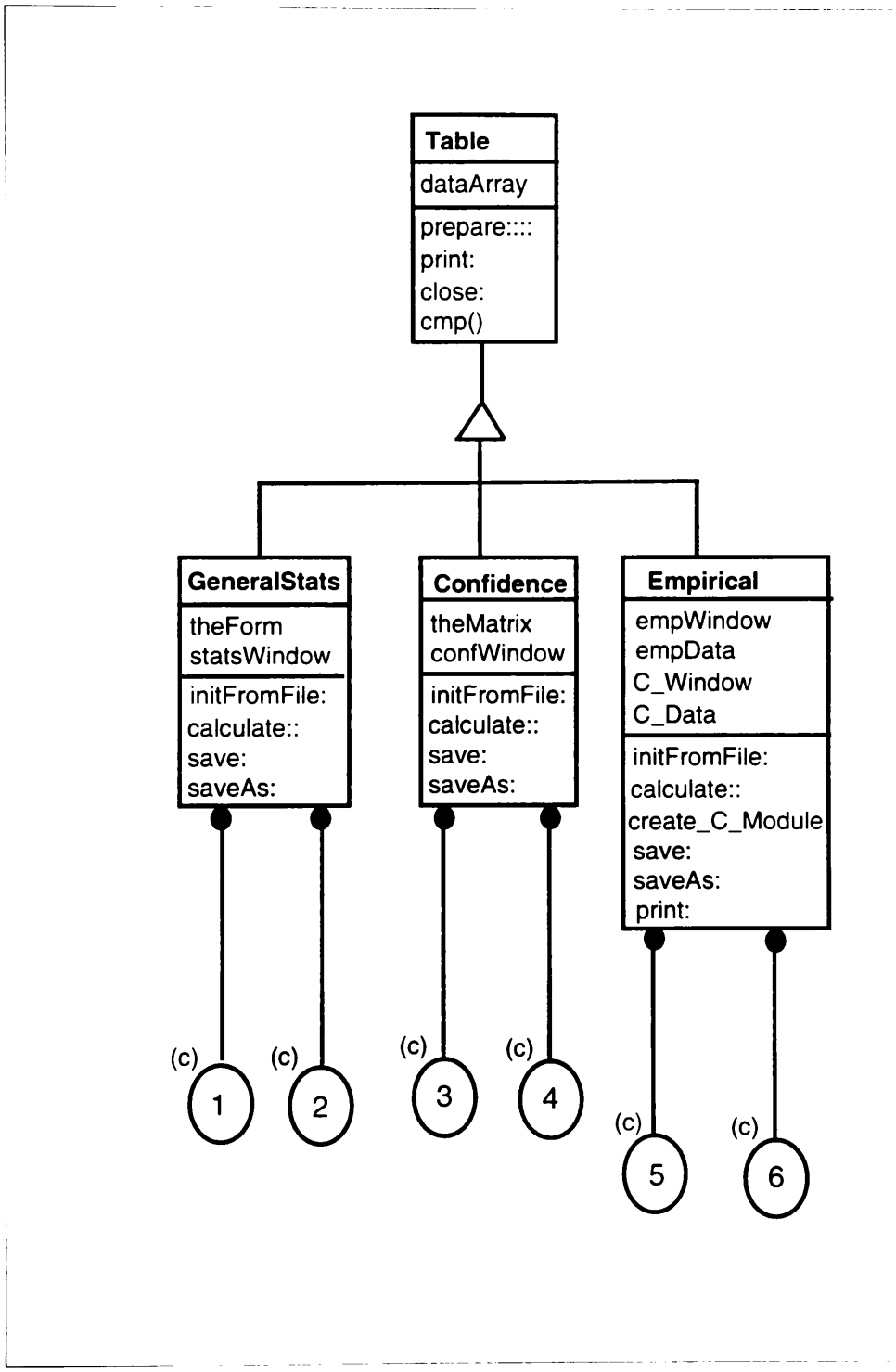
The number of possible objects to be instantiated in VSEDA is unknown, so that using instance diagrams is not a good approach. Class diagram is better for this purpose because they describe the general case in modeling the system.

Figure 3.4 is the class diagram for VSEDA demonstrating the implemented classes and the relationships between the objects of these classes. In these diagrams, we did not show the Application Kit classes due to the fact that they are so many in number.

In class diagrams each box demonstrates a class structure. The upper section in the box is the class name, middle section is the instance variables, and the lower section is the methods implemented in the class. Inheritance relationship is demonstrated with a triangle where its upper corner pointing to the super class.

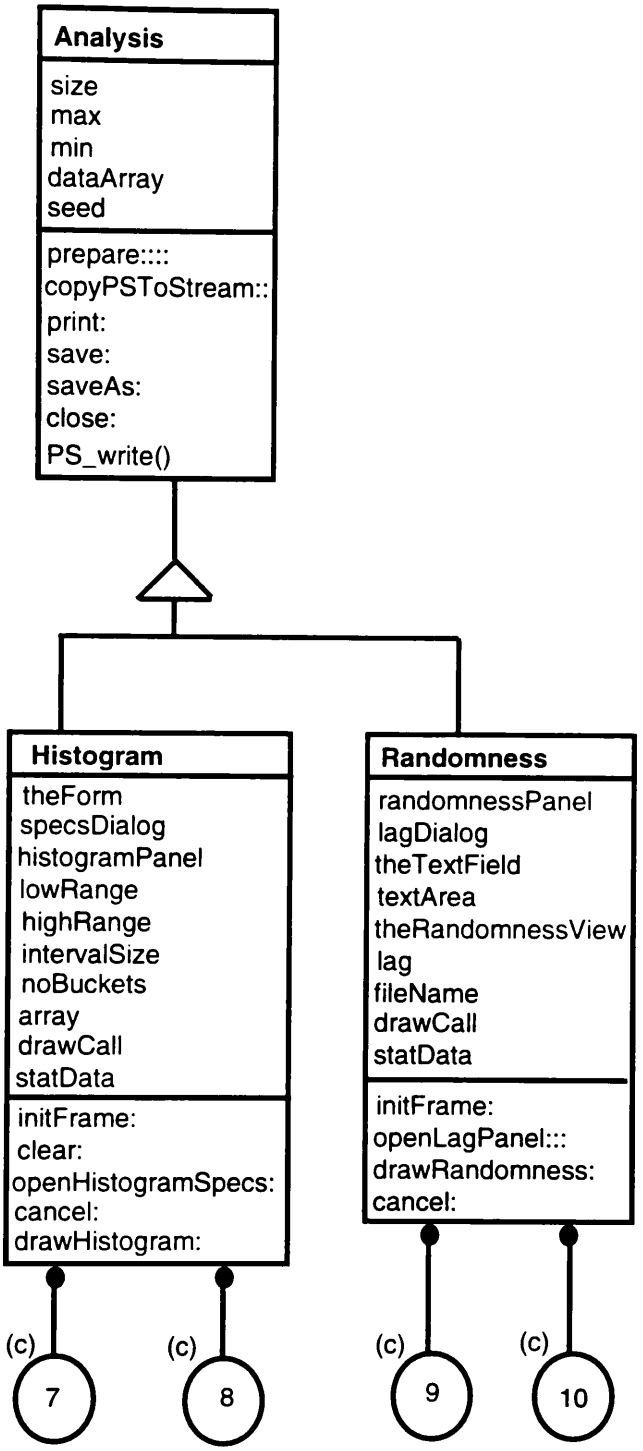
Lines connecting two classes are called associations. Associations are used to establish relationships among objects and classes. An association describes a group of links which are physical or conceptual connection between object instances. Associations

can also be given a name describing the relation, as seen in Figure 3.4.



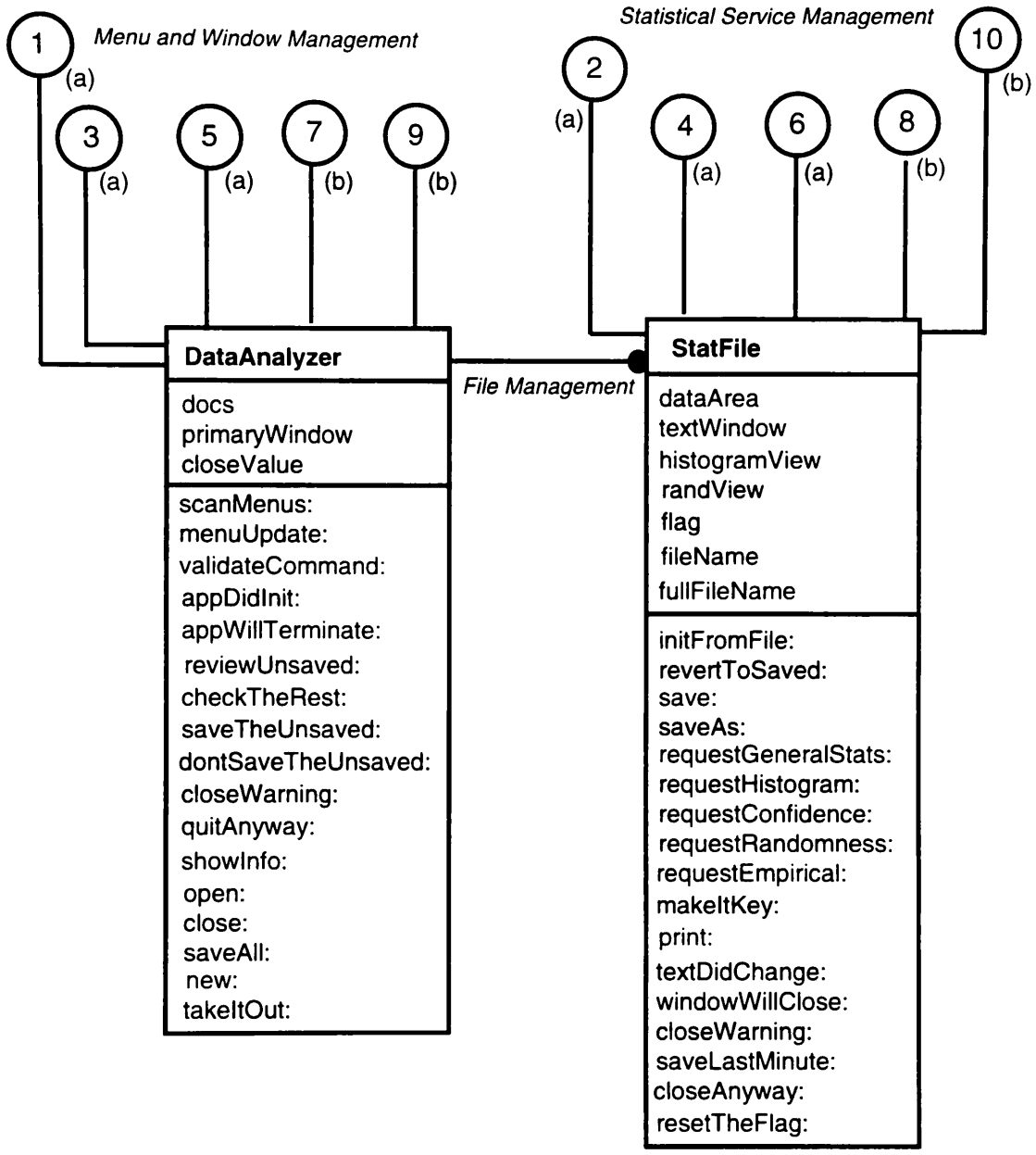
(a)

Figure 3.4 Object Diagram for the VSE DataAnalyzer



(b)

Figure 3.4 Object Diagram of the VSE DataAnalyzer (continued)



(c)

Figure 3.4 Object Diagram of the VSE DataAnalyzer (continued)

Another component of object diagrams is multiplicity which specifies how many instances of one class may relate to a single instance of an associated class. There are three different symbols used for multiplicity feature. A solid ball is used for "many multiplicity" meaning zero or more. A hollow ball is used for "optional multiplicity" meaning zero or more. A straight line without multiplicity symbols indicates a one-to-one association [Rumbaugh *et al.* 1991].

3.7.2 VSE DataAnalyzer Internal Logic

When the user clicks on the VSEDA application, the interface file "VSE_DataAnalyzer.nib" is loaded. This action automatically creates an object called *NXApp* which is an instance of the "Application" class. This interface file also brings up the VSEDA menu, and creates an instance of `DataAnalyzer` class which is also called *DataAnalyzer*. This newly created object is assigned as the delegate of the *NXApp* object. By using the automatic update features of Application Kit classes, menu cells are dimmed automatically by *DataAnalyzer* depending upon if that menu item is responded to by the target, which it sends its message to, at that particular time. `menuUpdate:`, `validateCommand:`, `scanMenus:`, and `appDidInit:` methods are used for this purpose. The basic responsibility of *DataAnalyzer* object is to send appropriate messages to the right target for processing. *DataAnalyzer* uses a `List` object to keep track of open data files, which is very handy especially in `reviewUnsaved:` and `saveAll:` methods of *DataAnalyzer*.

Whenever an existing data file is opened or a new data file is created, a new object of `StatFile` class is instantiated. All the services applying to the data files are managed from this object. It manages the services available in the "File" menu of the VSEDA menu, as well as the ones on the text window. The methods used for handling the "File" menu items are `revertToSaved:`, `print:`, `save:`, `saveAs:`, `saveLastMinute:`, and `closeAnyway:`. The services available on the text window

are presented in a pop-up menu. Each menu item sends a message to the StatFile instance requesting the specific services mentioned in the menu cells. The methods designed to handle these requests are requestGeneralStats:, requestHistogram:, requestConfidence:, requestRandomness:, and requestEmpirical:.

Each method of StatFile instance handling a request creates a new object to handle the related services for the statistics to be presented. The names of the methods imply the names of the classes from which the objects are created. To handle the visual statistics, an object belonging to a class which inherits from Analysis class is created. Histogram and Randomness classes are subclasses of Analysis base class. To handle the statistics to be presented in tabular format, an object belonging to a class which inherits from Table class is created. GeneralStats, Confidence, and Empirical classes are subclasses of Table base class.

Classes providing services for visual statistics have similar methods handling the calculations and the drawing. drawself:: method is used for drawing in the CustomView object. They also have panels getting information about details of drawing. Histogram class uses openHistogramSpecs::: method to handle the Histogram Specifications panel, and prepare:::: and drawHistogram: methods to perform the calculations. Randomness class uses openLagPanel::: method to manage the lag panel, and prepare:::: and drawRandomness: methods to perform the calculations. The drawself:: method is activated at the end of drawHistogram: and drawRandomness: methods by sending the display message to the object itself.

Chapter 4
USER'S GUIDE

The VSE DataAnalyzer (VSEDA) provides two different sets of features: management of files and data analysis results, and administration of data analysis features on data. These features are placed in different menus in order for the user to easily locate them. The first kind of features are located in VSEDA main menu, shown in Figure 4.1, and its components, while the second kind of features are placed in the pop-up menu, shown in Figure 4.2, on the text window. Both menus are consistent with the NEXTSTEP graphical user interface guidelines. The two different sets of features are explained below. The information contained herein can also be found in VSEDA Help Panel.

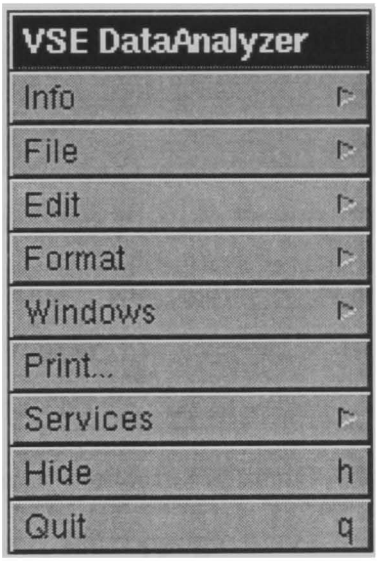


Figure 4.1 The VSE DataAnalyzer Main Menu

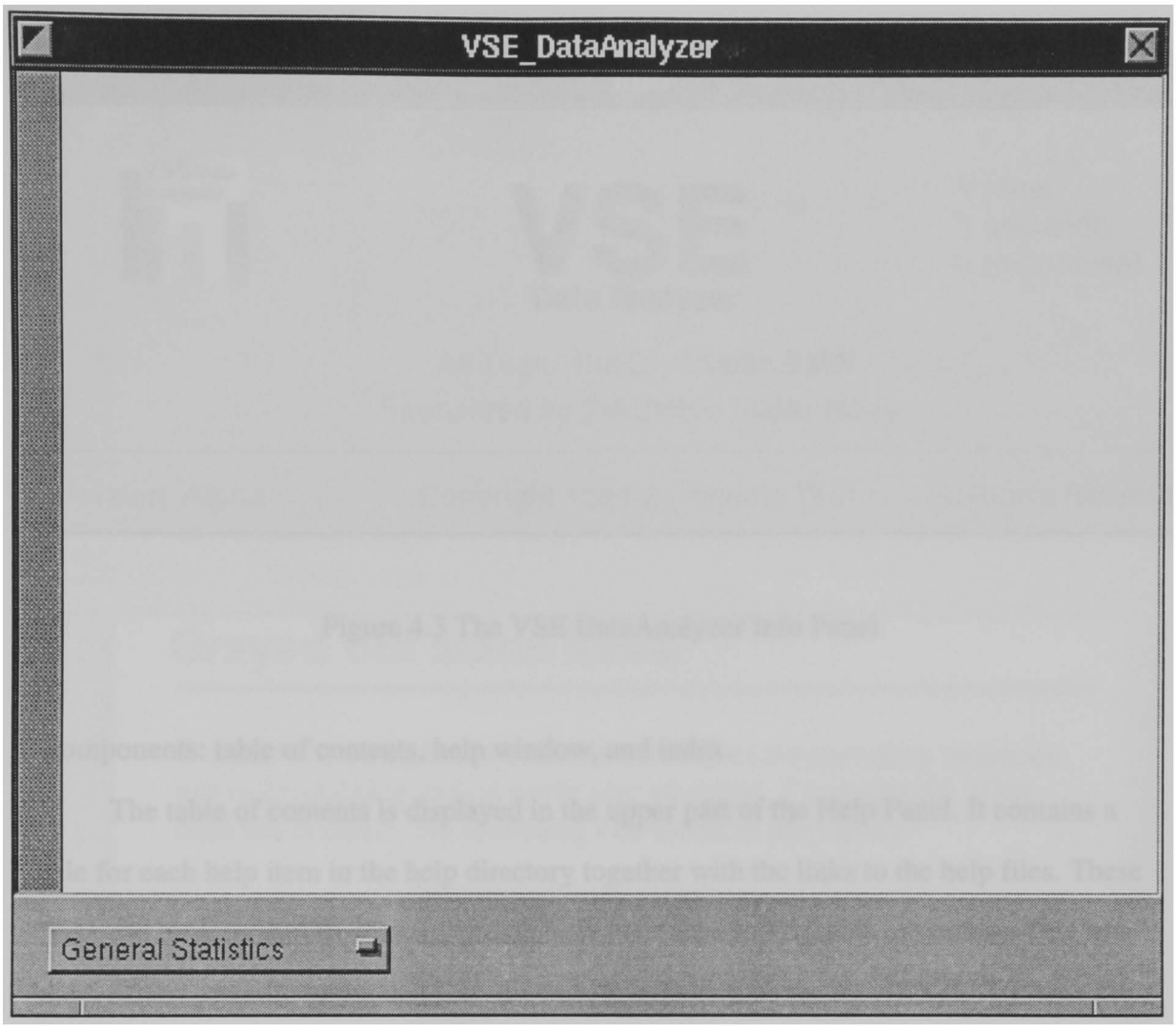


Figure 4.2 Pop-up Menu for Data Analysis Features Placed on the Text Window

4.1 Info Menu

4.1.1 Info Panel

Info Panel brings up a panel, shown in Figure 4.3 , displaying the information about the developers, development date, and copyright issues.

4.1.2 Help Panel

Most of the user-oriented information is placed in the help panel which can be accessed from the "Info" menu. Help panel, shown in Figure 4.4, consists of three

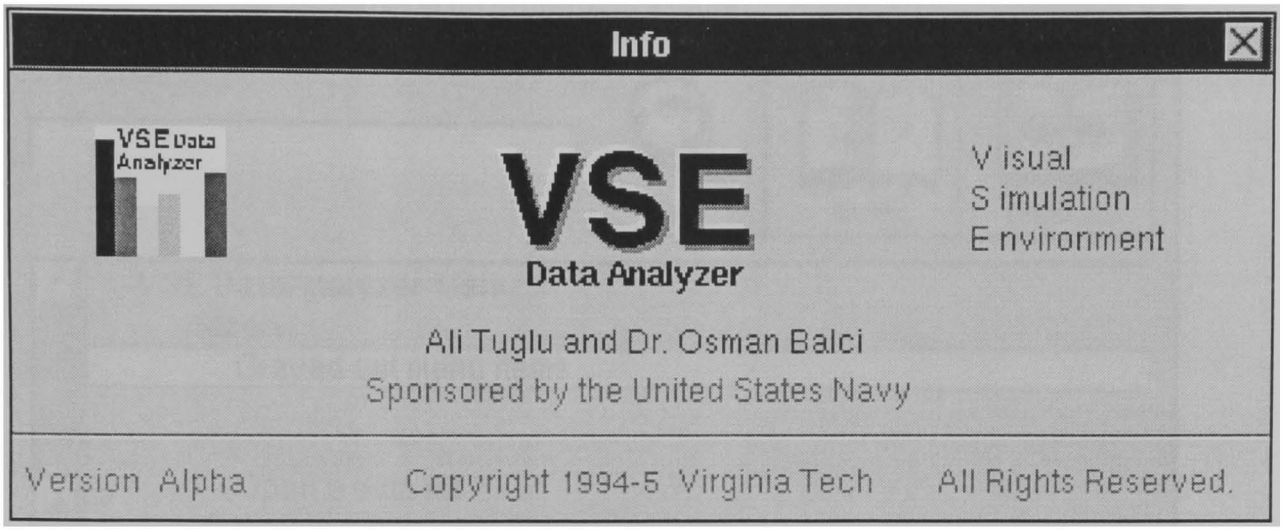


Figure 4.3 The VSE DataAnalyzer Info Panel

components: table of contents, help window, and index.

The table of contents is displayed in the upper part of the Help Panel. It contains a title for each help item in the help directory together with the links to the help files. These links can be activated by clicking on the base-level topics. The table of contents first lists the VSEDA-specific topics and then the help information about the NEXTSTEP-specific topics is provided by the NEXTSTEP software environment. Because most users are already familiar with the NEXTSTEP-specific topics, this information is placed after the VSE DataAnalyzer-specific information in the VSEDA Help Panel.

The help window displays the items selected in the table of contents. The information shown in the help window can also have links to the other files.

The index displays the help items in alphabetical order. Each item in the index has a link to a help file containing related information. These links can be activated by clicking on the index topics.

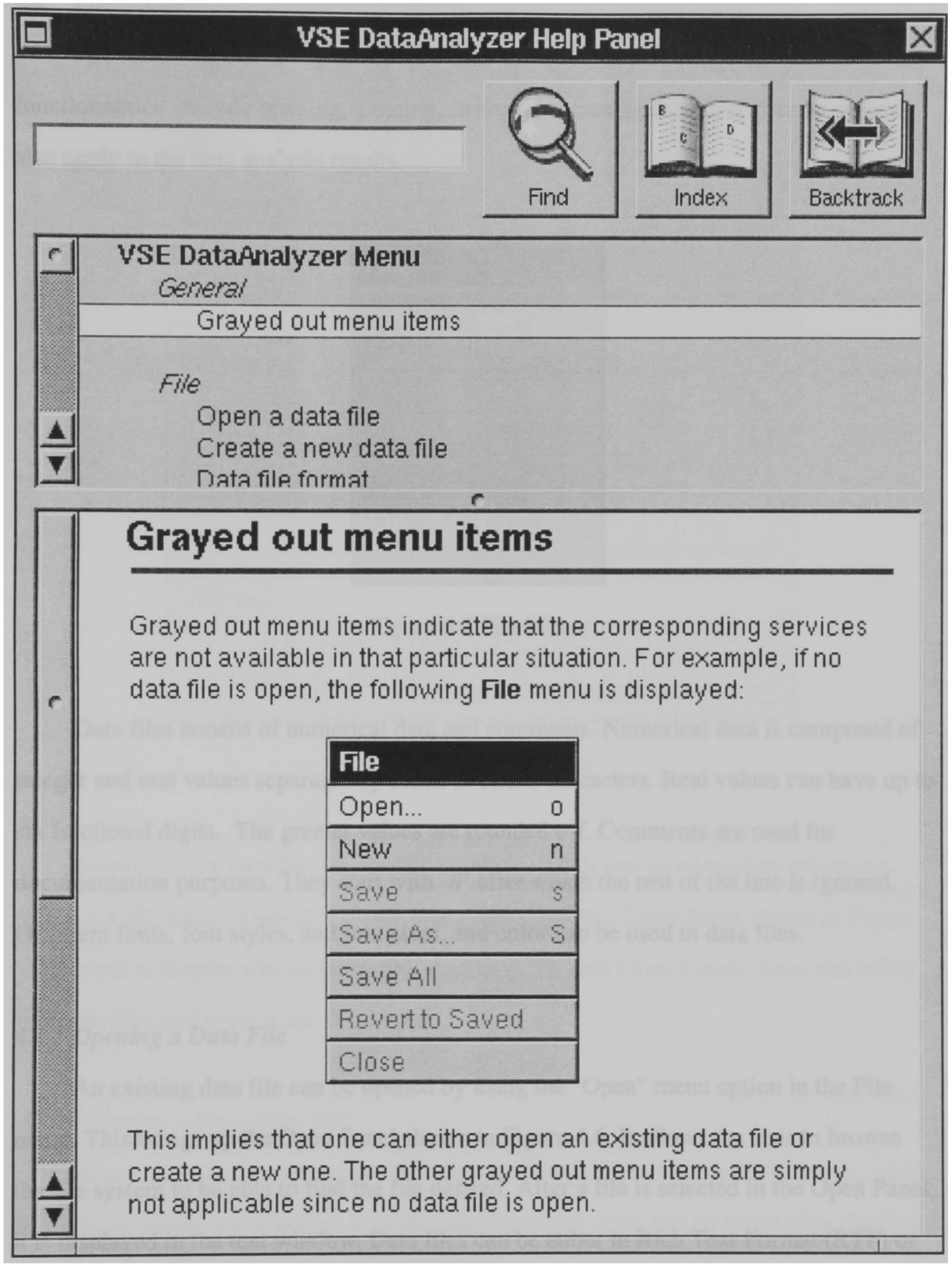


Figure 4.4 Help Panel

4.2 File Menu

File menu, shown in Figure 4.5, provides the services to manage the data files. Its functionalities include opening, creating, saving, and closing data files. Saving options also apply to the data analysis results.

Figure 4.5 File Menu

Data files consist of numerical data and comments. Numerical data is composed of integer and real values separated by blank or return characters. Real values can have up to six fractional digits. The greater values are rounded off. Comments are used for documentation purposes. They start with '//' after which the rest of the line is ignored. Different fonts, font styles, and font sizes, and color can be used in data files.

4.2.1 Opening a Data File

An existing data file can be opened by using the "Open" menu option in the File menu. This brings up the Open Panel shown in Figure 4.6. It allows the user to browse the file system to be able to find the file desired. After a file is selected in the Open Panel, it is displayed in the text window. Data files can be either in Rich Text Format (RTF) or ASCII format.

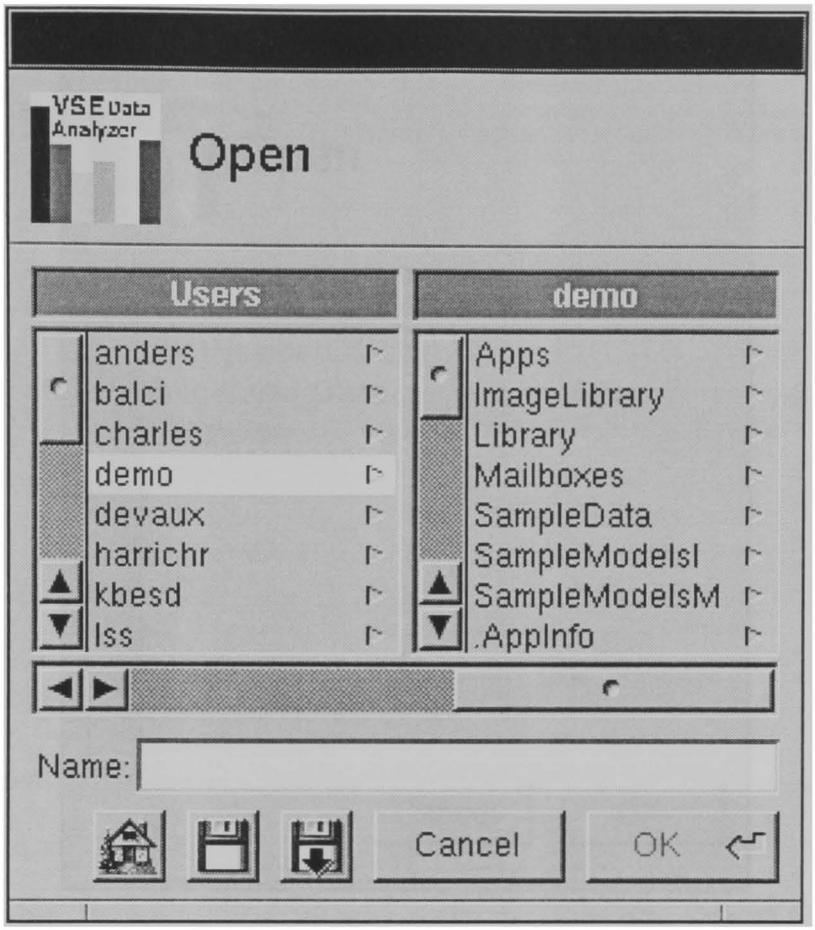


Figure 4.6 Open Panel

Moreover, the data files created in the VSE models can be opened by this menu. When a VSE model is selected on the Open panel, another Open Panel shown in Figure 4.7 is used to display simulation output data files. In both Open Panels, users can select multiple files and open at the same time.

4.2.2 Creating a New Data File

A new data file can be created by using the "New" option in the "File" menu, which displays an empty text window with the title "untitled". Data can be typed after clicking on the window. The format of the newly created data file is assumed to be RTF.

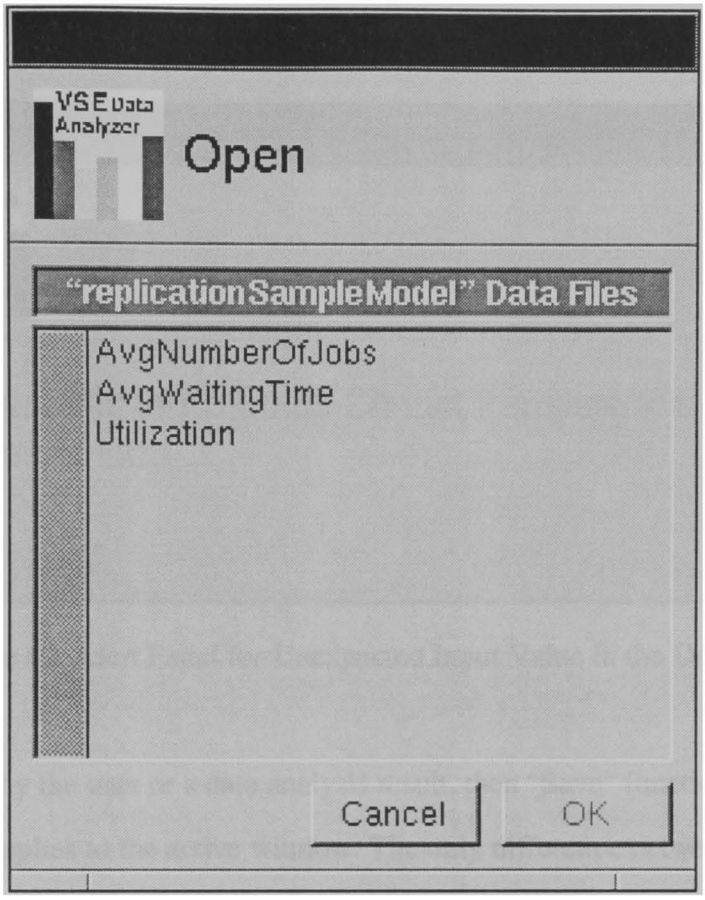


Figure 4.7 Open Panel for Simulation Output Data Files

When anything other than comments or numerical data are encountered in a data file, the panel shown in Figure 4.8 is displayed. The non-numeric input is highlighted for correction.

4.2.3 Saving the Files and the Data Analysis Results

Three different menu items from "File" menu can be used for saving the data files: Save, Save As, and Save All. The first two can also be used to save the data analysis results.

"Save" applies to the active window whose title bar is black. If it is a window consisting of a data file which already exists, then it is automatically saved. If it is a new

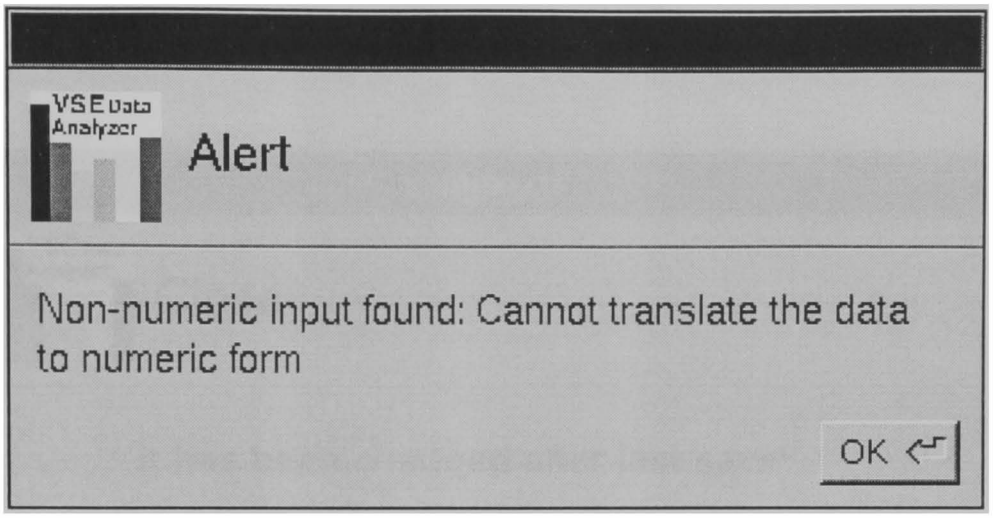


Figure 4.8 Alert Panel for Unexpected Input Value in the Data File

data file created by the user or a data analysis result, then "Save" functions as "Save As". "Save As" also applies to the active window. The only difference between "Save" and "Save As" is that "Save As" brings up the Save Panel for the user to enter the file name.

"Save All" applies to data files only. All the data files open are saved by their current names. If the data file is "untitled", then Open Panel is brought up.

Data files are saved in their original formats. Data analysis results are saved depending upon their presentation format. Tables are saved in the ASCII format, and the graphics are saved in the PostScript format.

4.2.4 Ignoring the Changes

"Revert to Saved" menu item in the "File" menu brings up the last saved version of the active data file. It does not apply to saved and newly created data files.

4.2.5 Closing the Windows

"Close" menu item applies to the active window whether it is a data file or statistics.

If it is a data file, and has been edited after last save, then a warning message shown in Figure 4.9 appears about saving the window before closing.

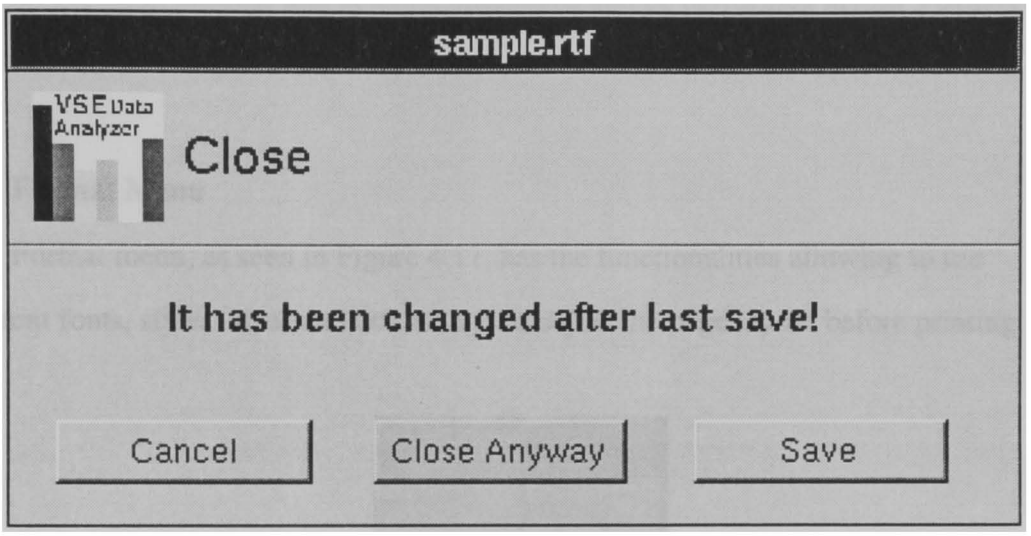


Figure 4.9 Confirmation Panel to Close the Unsaved and Edited Files

4.3 Edit Menu

Editing consists of cutting, copying, pasting, and deleting operations. Each operation is initiated by first selecting the text to be manipulated. "Select All" menu item is used to select the entire data file content. Figure 4.10 demonstrates the "Edit" menu.

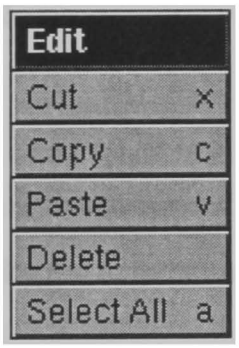


Figure 4.10 Edit Menu

If selected text is cut, it is removed from the data file. If selected text is copied, it stays as is in the data file. Paste is used to insert cut or copied text in the position of the I-beam in the data file. If selected text is deleted, it is removed from the data file. Cutting, copying, pasting, and deleting operations can be performed across VSEDA files and other files.

4.4 Format Menu

Format menu, as seen in Figure 4.11, has the functionalities allowing to use different fonts, styles, sizes and colors, and to adjust the page layout before printing.

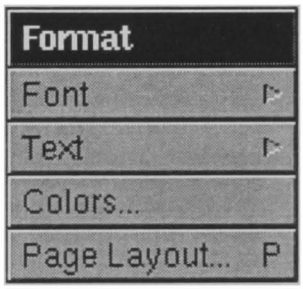


Figure 4.11 Format Menu

4.4.1 Fonts

All fonts available in the system can be used with different sizes and styles by using the Font menu shown in Figure 4.12.

4.4.2 Text Options

Text menu is used to align the text with options shown in the Figure 4.13. All the features of this menu are very well-known to most users.

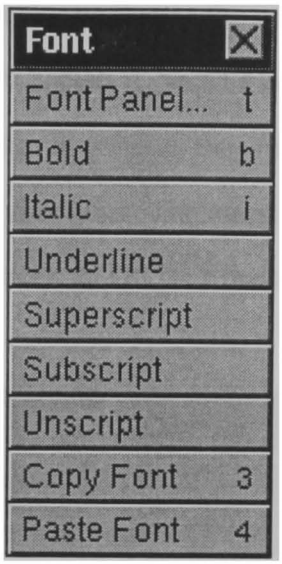


Figure 4.12 Font Menu

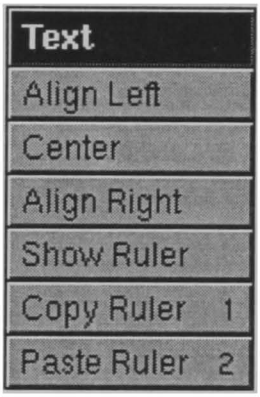


Figure 4.13 Text Menu

4.4.3 Colors

Color can be used in the data files. The "Colors" menu item in the Format menu will bring up the Color Panel shown in Figure 4.14.

Color Panel consists of several components: Magnifying Glass, Color Well, Color Wheel, Grayscale, Color Spectrum, and Color Lists.

Selected color appears in the color well. To change the color of selected text, the

color should be dragged from the color well onto the text. Color wheel, grayscale, spectrum, predefined color lists (PANTONE, NeXT Colors etc.), and magnifying glass can be used to pick the desired color. In general, the window with scroller is used to select the color, except for magnifying glass. The magnifying glass can be used to enlarge a selected area anywhere on the screen and to pick the color in its focus.

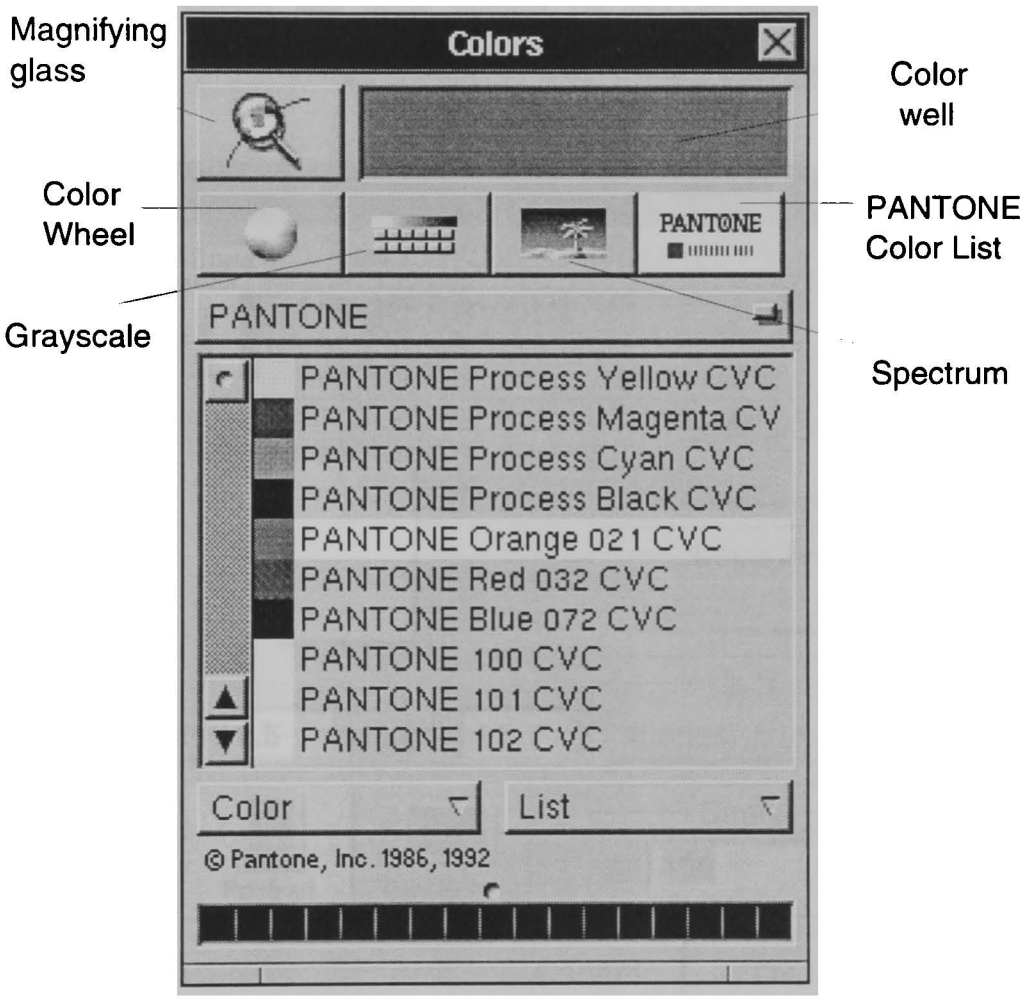


Figure 4.14 Colors Panel

Color can be associated with all text windows, but when it is saved only the data files can preserve the color in themselves. This is because all the statistics displayed in the text windows are saved in ASCII format. If the data file is in ASCII format, same

thing happens and the color is ignored when saved.

4.4.4 Page Layout

Page layout can be changed as desired by using the Page Layout Panel shown in Figure 4.15.

Most of the items on this panel are known to the users. However, we may need to explain about the "Layout" and "Scale" options. The user can reduce or enlarge printed pages by changing the percentage in the Scale field. Scaling does not change the actual

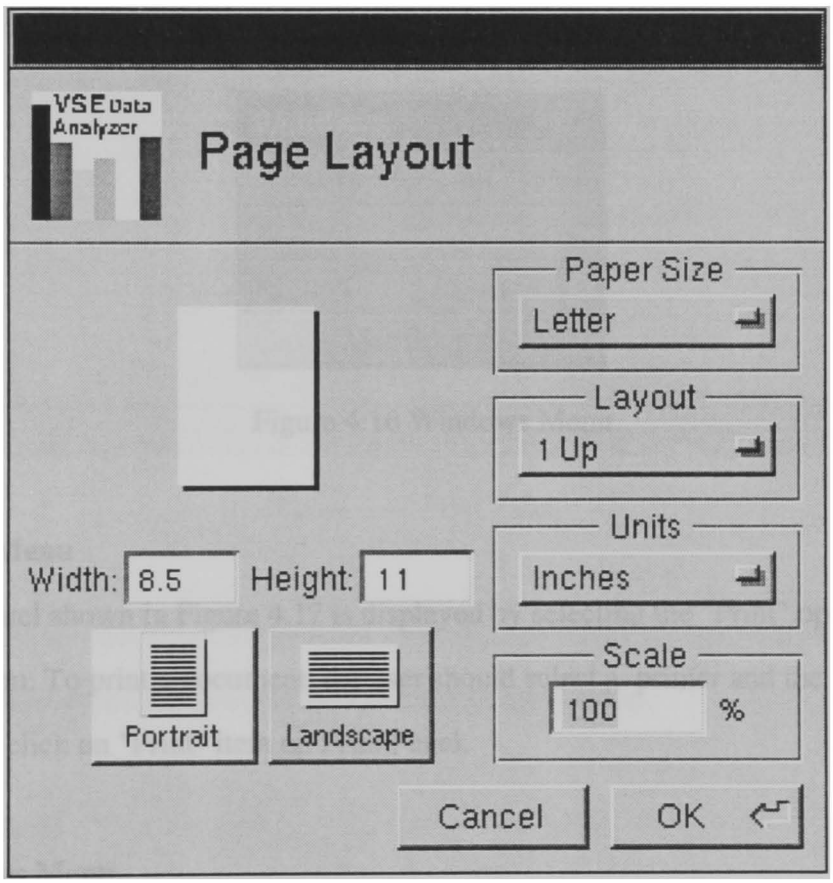


Figure 4.15 Page Layout Panel

contents of a file. Layout is used to choose the number of pages to be printed on each sheet of paper. Normally this is 1 page on each sheet of paper (1 Up), and can be increased up to 16 pages on each sheet (16 Up).

For printing a histogram or the plot of the randomness test, the image is automatically scaled to fit on an 8.5" x 11" paper. If printing is desired using a different scale, the image should first be saved as a PostScript file which then is manipulated by using a program such as Preview.

4.5 Windows Menu

"Windows" menu option brings the menu, shown in Figure 4.16, providing standard NEXTSTEP functionality for window operations. Availability of options in this menu changes depending upon the number of windows open.

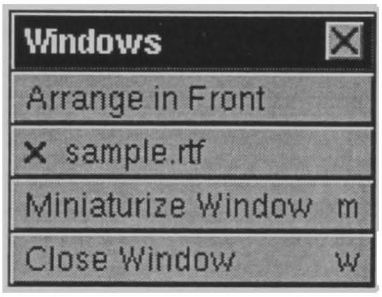


Figure 4.16 Windows Menu

4.6 Print Menu

The panel shown in Figure 4.17 is displayed by selecting the "Print" option from the VSEDA menu. To print a document, the user should select a printer and the proper options, and click on "Print" item on Print Panel.

4.7 Services Menu

Depending on the availability, several applications can be incorporated with your application. In most cases, highlighting a part of text can activate several services to be used. Services available are determined by the operating system. The list of applications

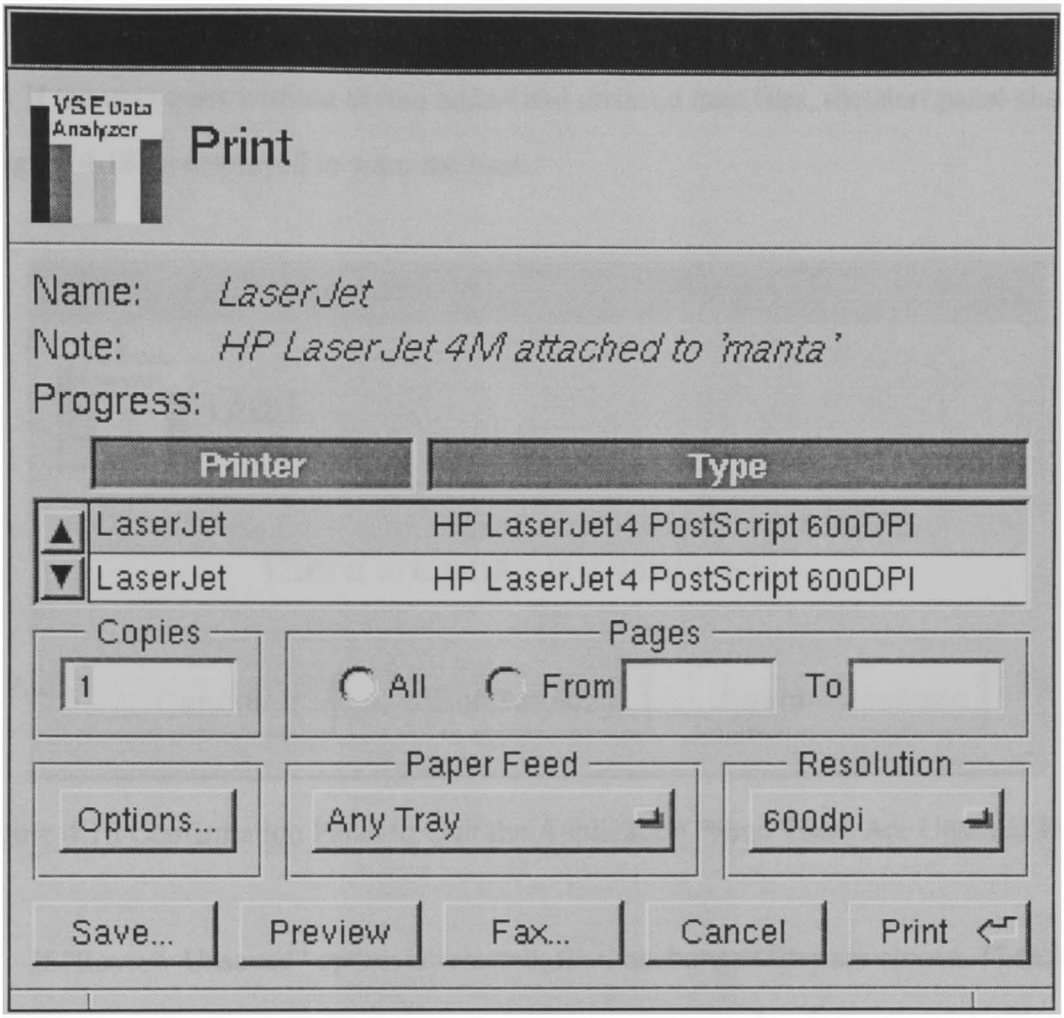


Figure 4.17 Print Panel

are determined by checking out the Apps directories to which the user has access. This list can also be customized through the Preferences application.

4.8 Hide Menu

This menu item hides the application from the view. When an application is hidden, no unsaved work is lost, nor is it saved. By hiding applications, several applications can be run at the same time and be accessed easily.

4.9 Quit Menu

If the user quits without saving edited and unsaved data files, the alert panel shown in Figure 4.18 is displayed to warn the user.

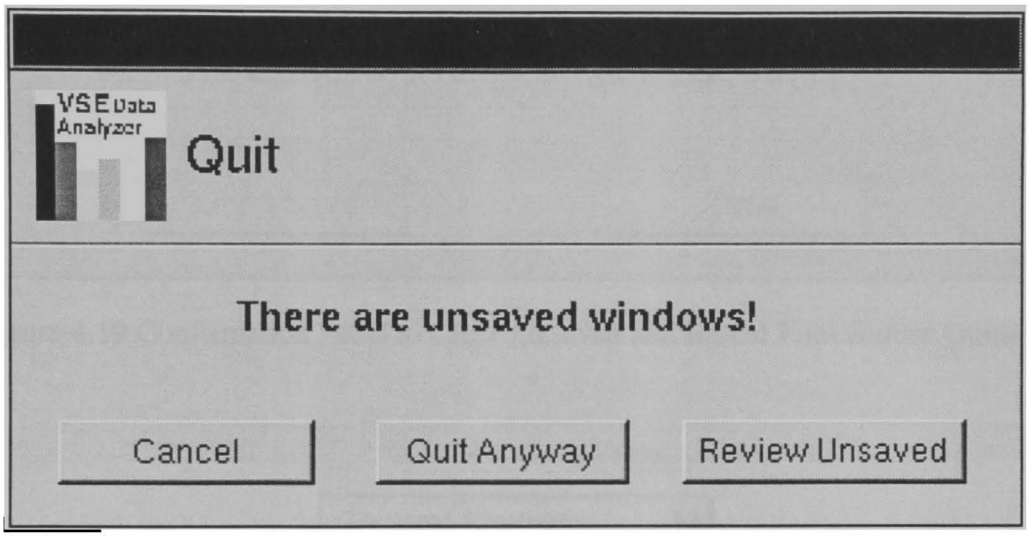


Figure 4.18 Confirmation Panel to Quit the Application When There Are Unsaved Files

If "Review Unsaved" option is selected, first, unchanged files are closed. Then, the user is prompted with the panel in Figure 4.19 for every edited and unsaved data file.

If the data file is newly created or unsaved and the user wants to save it, then the Save Panel is displayed. After reviewing all the unsaved data files, the application is quitted.

4.10 Data Analysis Features Menu

VSEDA analyzes the data files to collect several statistics, and to create information to be used by the other components of Visual Simulation Environment. Five data analysis features are available from the menu shown in Figure 4.20, which is located on the text window as shown in Figure 4.2:

- General Statistics

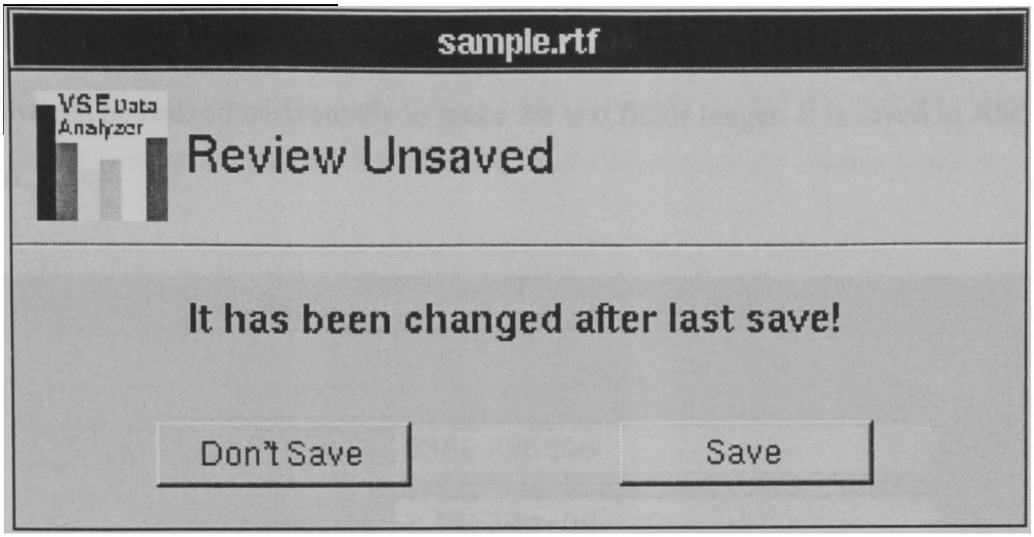


Figure 4.19 Confirmation Panel to Close Unsaved and Edited Files Before Quitting

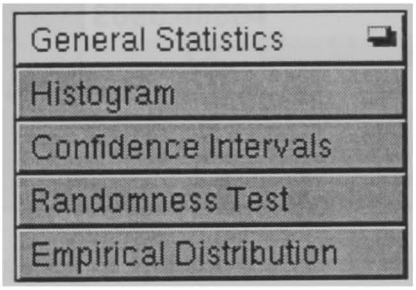


Figure 4.20 Pop-up Menu for the Data Analysis Features

- Histogram
- Confidence Intervals
- Randomness Test
- Random Variate Generation Using Empirical Data

4.10.1 General Statistics

General Statistics panel shown in Figure 4.21 gives the most common statistics about the data file: sample mean, sample variance, minimum, maximum, median, and number of observations.

The General Statistics table can be printed out as it looks. The General Statistics window can be resized horizontally to make the text fields longer. It is saved in ASCII format.

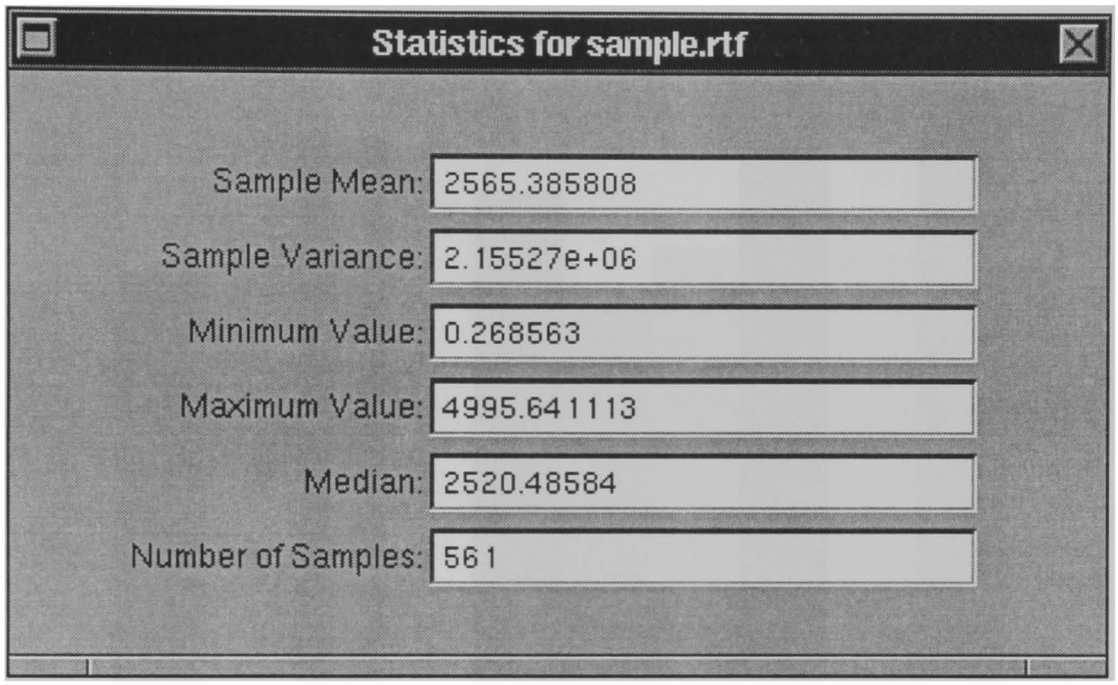


Figure 4.21 General Statistics Panel

4.10.2 Histograms

A histogram is used to show the frequency distribution of the sample values in the data files. To construct a histogram, three pieces of information are needed: lower range, upper range, and interval size. A sample histogram is illustrated in Figure 4.22.

All the values are expected to be entered by the user. Although, by default certain values are provided for the users, users are encouraged to enter more meaningful values depending on the general statistics. Maximum of 30 intervals can be specified.

Histograms can be printed by using the "Print" option on the main menu. It is scaled automatically to fit on an 8.5"x11" page. If the size of the graph needs to be changed, another application such as Preview should be used.

Histograms can be saved by using "Save" and "Save As" options on the menu. They both bring up the Save Panel. Histograms are saved as postscript files, and the user does not need to give an extension for the file name since it is given by VSEDA anyway.

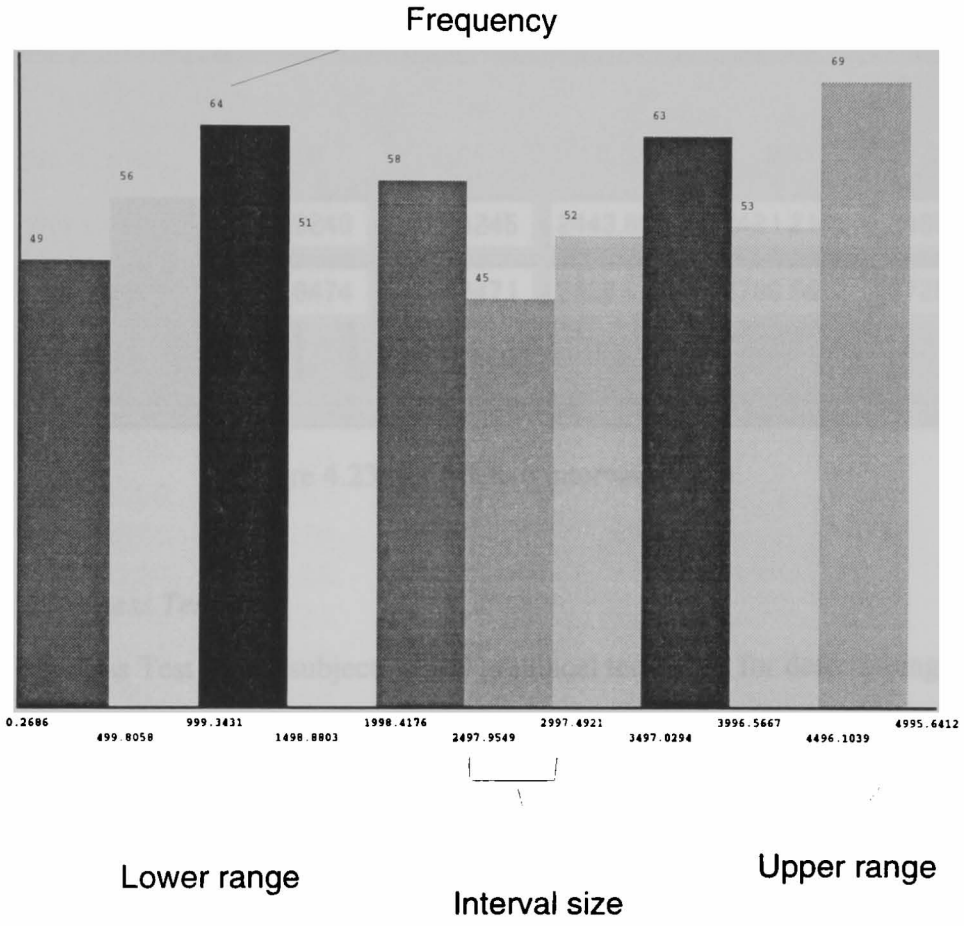


Figure 4.22 A Sample Histogram

4.10.3 Confidence Intervals

Confidence intervals can be constructed for different parameters of interest. In VSEDA, it is calculated for the mean value. Different confidence levels show how much we are confident that the estimated value of that specific parameter will be between those upper and lower bounds.

Confidence Intervals table shown in Figure 4.23 can be printed out as it looks. The

window can be resized horizontally to make the text fields longer. Confidence Intervals are saved in ASCII format.

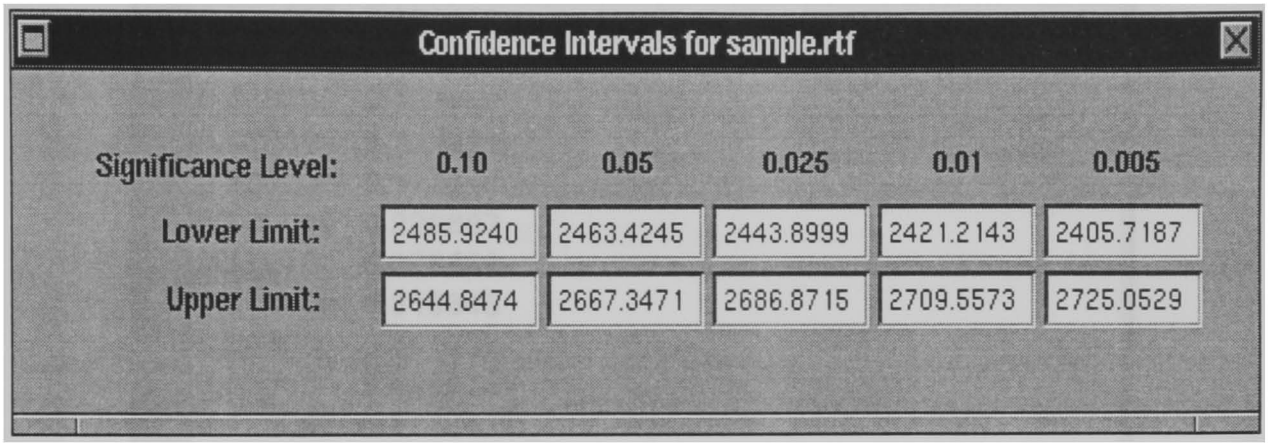


Figure 4.23 Confidence Intervals Table

4.10.4 Randomness Test

Randomness Test uses a subjective and graphical technique for determining the randomness of the data. Interpretation of the test is explained in section 2.4.

Randomness test can be printed by using the "Print" option on the main menu. It is scaled automatically to fit on an 8.5"x11" page. If the size of the graph needs to be changed, another application such as Preview should be used.

4.10.5 Random Variate Generation Using Empirical Data

In the case that the distribution of sample values in the data file does not fit to any kind of known distribution, we need to create random variates using empirical data to be able to drive the simulation model.

In VSEDA, this is achieved through two steps. In the first step, cumulative probability distribution shown in Figure 4.24 is created by using the data file. Depending on the randomness of the values, this table may greatly differ in size. In the second step,

C module that can produce random variates according to this distribution is created by clicking on "Create C Module" button on the empirical distribution window. This module

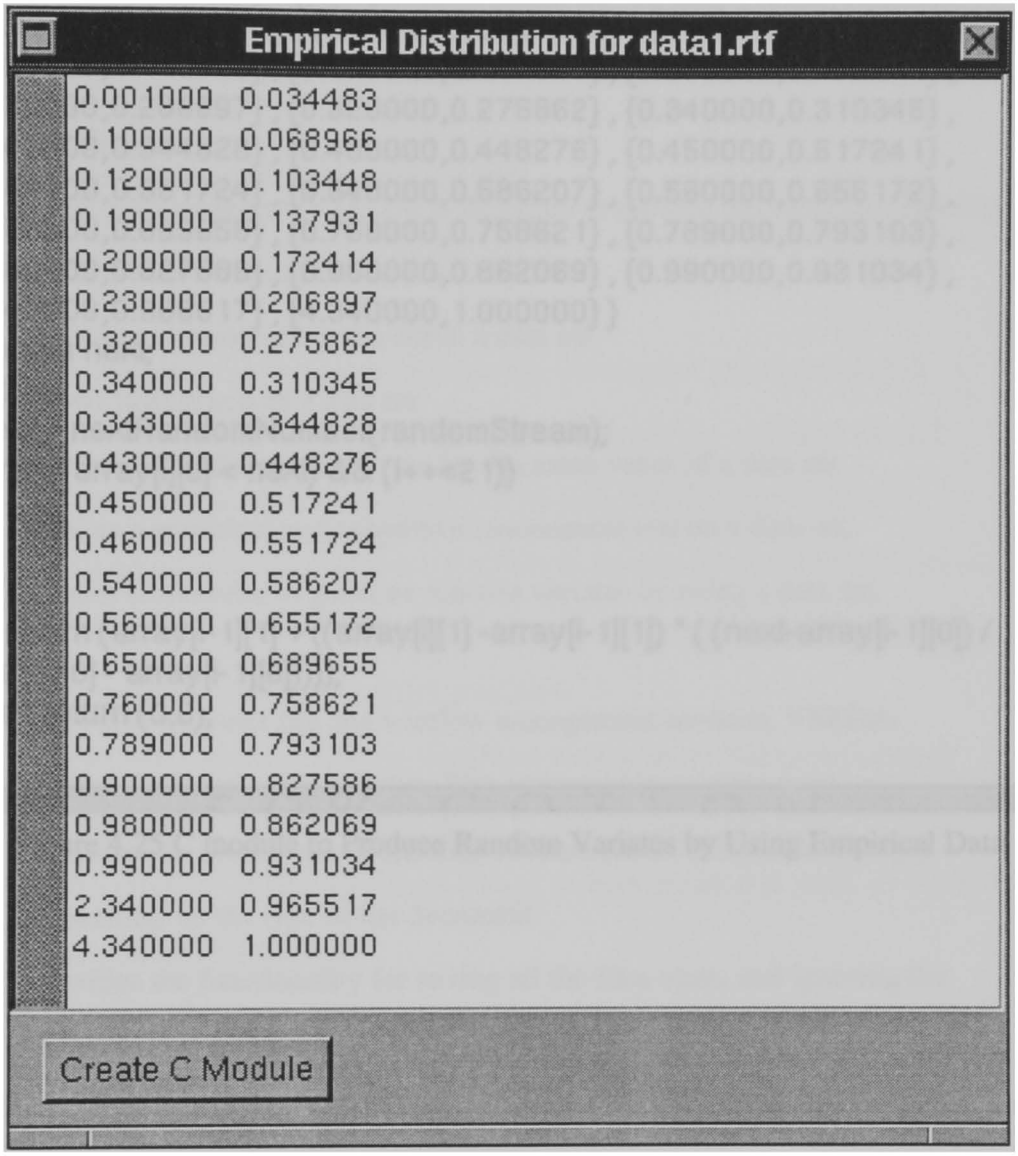


Figure 4.24 Cumulative Probability Distribution Table

can be brought into the VSE model as a function. The user is expected to give a name for the C module which is initially highlighted. If it is not changed, then the default name "my_function" will be used. Figure 4.25 illustrates the C module created by using the cumulative distribution table shown in Figure 4.24.

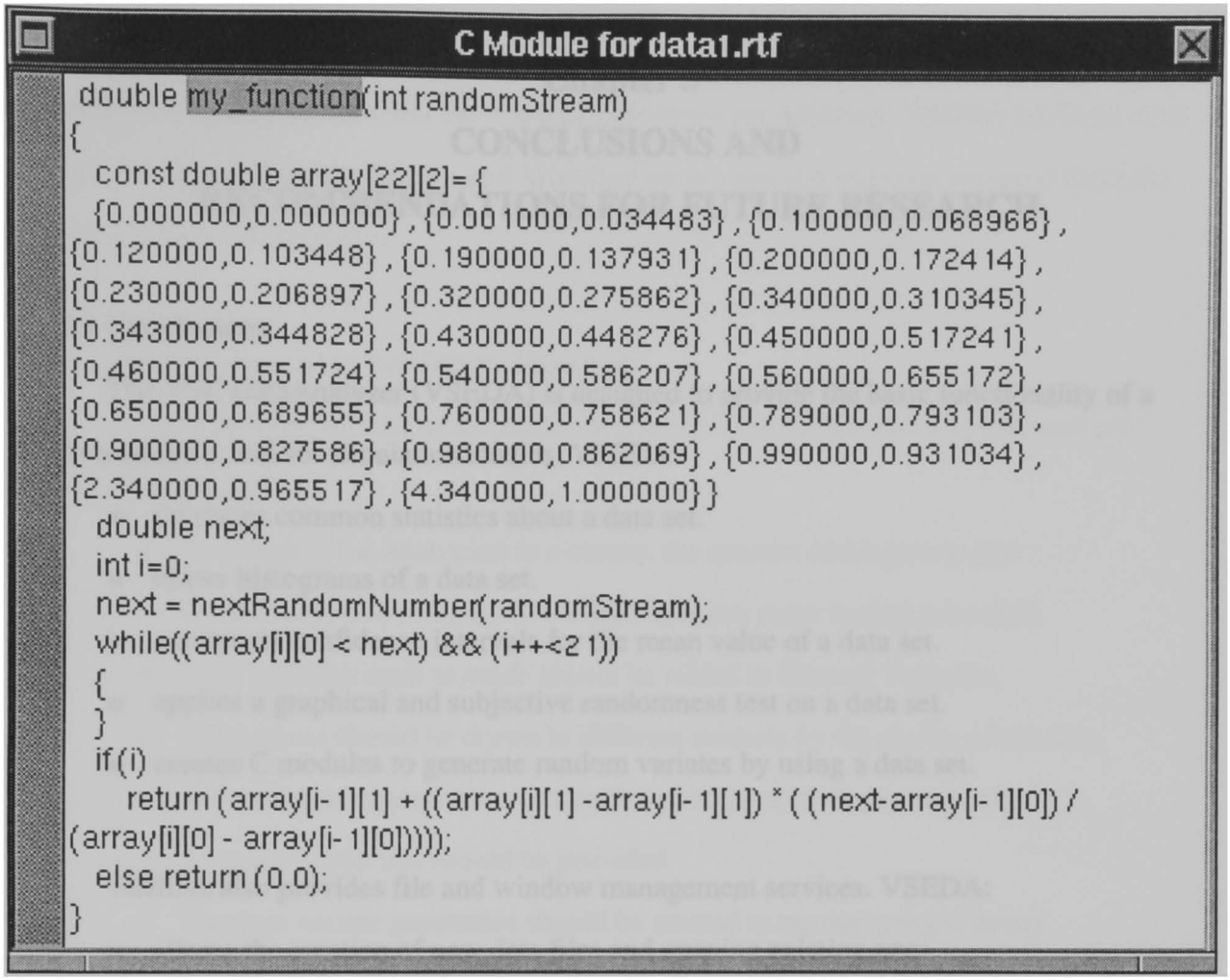


Figure 4.25 C module to Produce Random Variates by Using Empirical Data

Chapter 5
CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

5.1 Conclusions

The VSE DataAnalyzer (VSEDA) is designed to provide the basic functionality of a data analysis tool for simulation studies. VSEDA:

- produces common statistics about a data set.
- draws histograms of a data set.
- constructs confidence intervals for the mean value of a data set.
- applies a graphical and subjective randomness test on a data set.
- creates C modules to generate random variates by using a data set.

VSEDA also provides file and window management services. VSEDA:

- allows the creation of new data files and opening existing ones.
- can save the documents in different formats (RTF, ASCII, PostScript) depending on the type of the document.
- provides the functionality for saving all the files open, and ignoring the changes after last save.
- requires the confirmation of the user in the case of quitting the application or closing the data file.
- provides a complete help feature presented with a state-of-the-art interface.
- has standard editing, printing, formatting functionalities of other NEXTSTEP applications.
- can activate other applications as services.

When we make an overall comparison of research objectives and the features described above, we find that the research objectives are achieved. VSEDA has been used in an undergraduate Simulation and Modeling course and the response received from the students has been positive.

5.2 Recommendations for Future Research

The following is a list of recommendations in no particular order to enhance and enrich VSEDA:

- Although it has been used in a course, the amount of usage was not satisfactory to thoroughly test VSEDA, hence more testing is needed.
- New statistics such as *mode* should be added in General Statistics.
- Histograms should be drawn in different formats by the choice of the user.
- Several other randomness tests such as frequency test, gap test and autocorrelation test should be provided.
- Random variate generators should be created in several programming languages.
- Distribution fitting algorithms should be incorporated in order to use in input data modeling process.
- VSEDA does not permit the user to change the content of the graphics drawn as a result of a visual data analysis feature. This should be done in the future in order to provide a more flexible approach.
- User should make the decision about displaying either the header file or the whole file.
- Relative frequency should be displayed on the histograms.
- In the construction of confidence intervals, a warning panel should be displayed if the number of samples is less than 30.
- Data Analysis features should be applied on the combination of several files.

- When adding completely new features, an abstract base class should be implemented to enable the reuse of the code.

References

Balci, O. (1986), "Requirements for Model Development Environments," *Computers & Operations Research 13*, 1 (Jan.-Feb.), 53-67.

Balci, O. (1988), *CS4214 Simulation and Modeling Lecture Notes,* Kinko's Copy Center, Blacksburg, VA.

Balci, O. (1993), *CS5705 Software Engineering Lecture Notes,* Virginia Tech Copy Center, Blacksburg, VA.

Balci, O. and R. E. Nance (1992), "The Simulation Model Development Environment: An Overview," In *Proceedings of the 1992 Winter Simulation Conference (Arlington, VA, Dec. 13-16).* IEEE, Piscataway, NJ, pp. 726-736.

Derrick, E. J. and O. Balci (1992), "DOMINO: A Multifaceted Conceptual Framework for Visual Simulation Modeling," *Technical Report 92-43*, Department of Computer Science, VPI&SU, Blacksburg, VA, Aug.

Derrick, E. J. and O. Balci (1995), "A Visual Simulation Support Environment Based on the DOMINO Conceptual Framework," *The Journal of Systems and Software*, to appear.

Garfinkel, S.L. and M.K. Mahoney (1993), *NeXTSTEP Programming Step One: Object-Oriented Applications,* Springler-Verlag, New York, NY.

Law, A.M. and S. Vincent (1994), *UniFit II User's Guide*, Averill M. Law and Associates, Tucson, AZ.

Nance, R.E. (1995), Personal Communication.

NeXT Computer, Inc. (1992a), *General Reference Vol. 1,* Addison-Wesley Publishing Company, New York, NY.

NeXT Computer, Inc. (1992b), *NeXTSTEP User Interface Guidelines,* Addison-Wesley Publishing Company, New York, NY.

NeXT Computer, Inc. (1995), *NeXTSTEP Online Documentation,* NeXT Computer, Inc., Redwood City, CA.

Nghiem, A.D. (1993), *NeXTSTEP Programming Concepts and Applications,* Prentice-Hall, New Jersey, NJ.

Pinson, L.J. and R.S. Wiener (1991), *Objective-C*, Addison-Wesley Publishing Company, New York, NY.

Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991), *Object-oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ.

Schefler, W.C. (1988), *Statistics, Concepts and Applications*, The Benjamin/Cummings Publishing Company, Menlo Park, CA.

Walpole, R.E. and R.H. Myers (1978), *Probability and Statistics for Engineers and Scientists*, MacMillan Publishing Company, New York, NY.

Vitae

ALI TUGLU

EDUCATION:

M.S., Computer Science, June 1995
Virginia Polytechnic Institute and State University (VPI&SU)
Blacksburg, Virginia

B.S., Computer Engineering, July 1991
Istanbul Technical University
Istanbul, Turkey

EXPERIENCE:

Graduate Teaching Assistant, January 1993 - May 1995
Department of Computer Science, VPI&SU
Blacksburg, Virginia

Software Engineer, June 1994 - June 1995
Visual Simulation Environment Project
Department of Computer Science, VPI&SU
Blacksburg, Virginia

EMPLOYMENT:

Project Engineer, June 1995 -
Automated Analysis Corporation (AAC)
423 S.W. Washington St.
Peoria, Illinois 61602