Courseware Development System:
The Bernoulli Process

by

Jason A. Beardsley

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1991

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 1991

Certified by . . . .
Edward J. Moriarty
Thesis Supervisor

Certified by . . . .
Professor Alvin W. Drake
Thesis Supervisor

Accepted by . . . .
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

Courseware Development System: The Bernoulli Process

by

Jason A. Beardsley

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 1991, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

Computers can be productive tools in education, allowing self-guided learning and interactive, repeatable lessons. Computer-based instruction aids can be especially helpful in the fields of mathematics, science, and engineering, due to the computer's ability to perform simulations, analyze data, and provide dynamic, graphical representation of information. Unfortunately, there is a scarcity of software powerful and simple enough to allow teachers, not programmers, to develop their own educational programs. The Courseware Development System (CDS) project was formed with the goal of producing a computational environment in which such tools are easily created by educators.

This thesis describes the design and implementation of a CDS application which introduces a topic in the area of applied probability, the Bernoulli process. The application is tailored towards a specific class, *Probabilistic Systems Analysis* (6.041), and will be used for lecture demonstration in future terms. The presentation is also general enough to be useful outside of 6.041 lecture. This application represented the first real deployment of CDS apart from basic testing; thus, another objective of this thesis was to confirm the worth of CDS as a development platform. In addition, this thesis project involved extending the CDS environment to support Unix socket-based interprocess communication.

Thesis Supervisor: Edward J. Moriarty
Title: Project Manager

Thesis Supervisor: Alvin W. Drake
Title: Professor of Systems Science and Engineering

Courseware Development System: The Bernoulli Process

by

Jason A. Beardsley

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 1991, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

Computers can be productive tools in education, allowing self-guided learning and interactive, repeatable lessons. Computer-based instruction aids can be especially helpful in the fields of mathematics, science, and engineering, due to the computer's ability to perform simulations, analyze data, and provide dynamic, graphical representation of information. Unfortunately, there is a scarcity of software powerful and simple enough to allow teachers, not programmers, to develop their own educational programs. The Courseware Development System (CDS) project was formed with the goal of producing a computational environment in which such tools are easily created by educators.

This thesis describes the design and implementation of a CDS application which introduces a topic in the area of applied probability, the Bernoulli process. The application is tailored towards a specific class, *Probabilistic Systems Analysis* (6.041), and will be used for lecture demonstration in future terms. The presentation is also general enough to be useful outside of 6.041 lecture. This application represented the first real deployment of CDS apart from basic testing; thus, another objective of this thesis was to confirm the worth of CDS as a development platform. In addition, this thesis project involved extending the CDS environment to support Unix socket-based interprocess communication.

Thesis Supervisor: Edward J. Moriarty
Title: Project Manager

Thesis Supervisor: Alvin W. Drake
Title: Professor of Systems Science and Engineering

Acknowledgments

I would like to first thank my parents, for their patience and support during these long four years — I could not ever have survived MIT without you, Mom and Dad.

To the “management” of CDS, Jon Haass and Ed Moriarty, I extend my sincere gratitude for allowing me to be a part of this project; it has been a worthwhile and enjoyable experience. Thanks also to the members of the LAS and CDS projects, including: Joe Wang, Marc Schaub, Sourabh Niyogi, John Piscitello, Jeff Turcotte, and Adam Feder.

Thanks to Project Athena, DEC, IBM, and UROP for providing the computing environment and resources which made these research efforts possible. Thanks also to Donald Knuth and Leslie Lamport, creators of TeX and LaTeX, respectively, for making this thesis much easier to write.

Last, but not least, I wish to thank my thesis advisor, Professor Alvin Drake, whose insight and guidance was invaluable during these last few months.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is CDS?

The Courseware Development System (CDS) is an X Windows based authoring environment; CDS is intended to be used by non-programmers, especially teachers, to prepare educational materials which can be used in lecture, homework assignments, and individual student exploration. CDS applications support interactive modelling and graphics, as well as text-based presentations. CDS is the result of an ongoing, cooperative effort by the EECS and Mathematics departments. [1]

CDS is the successor of LAS, the Lecture Authoring System, also based on X Windows, and intended for use by educators. The reasoning and design goals behind LAS are detailed in [Miller 90]; they are essentially the same as those for CDS. In January 1990, the members of Project LAS realized that the current system did not meet the needs it was supposed to address, and decided to redesign and create an entirely new platform. The specific problems with LAS that led to this decision, and the solutions employed by CDS are:

[1] More comprehensive descriptions of CDS and its history can be found in [Moriarty 91] and [CDS Programmer].

- The editing and displaying of LAS lecture files took place in separate programs; the author could not edit an object and immediately see the results. Often, the creation of a complicated application required much additional editing, in Emacs — in order to use LAS effectively, one had to learn to "program" in the language interpreted by LAS.

  CDS solves these problems by having a single programming environment, providing WYSIWYG [2] layout capability, as well as the ability to test and execute a presentation during the editing process. This same executable is used when running a CDS application.

- The LAS modelling environment did not provide a great deal of flexibility, and could be extended only through complicated C programming. New display objects were similarly difficult to create.

  The CDS modelling environment is based on Scheme, and is capable of controlling any aspect of the application. Scheme was chosen mainly because it is relatively simple to learn, and because every EECS undergraduate knows how to program in Scheme after taking 6.001.

- The implementation of LAS was in C, and originally based on Xlib, a low-level X Windows programming library. Porting the code to the X Toolkit and Motif introduced user interface problems, and gave LAS an inconsistent look-and-feel.

  CDS is based on the Motif user interface, which has become a widely-accepted standard; CDS should look familiar to anyone who has used a Macintosh or Microsoft Windows.

- LAS system code was becoming more and more difficult to understand or modify, due to the different coding styles of the many programmers,

[2] "What you see is what you get," meaning that changes made during editing are immediately noticeable.

and the deficiencies of C.

CDS is written in C++, in an object-oriented fashion, and is therefore much more modular in design. In addition, the C++ language enforces better coding, and is generally easier to understand.

1.2 The Bernoulli Process Application

This thesis builds upon earlier work done for the Lecture Authoring System, as documented in [Zeitler 89] and [LaPlante 88]. My application is intended to supersede this older presentation, which has been used during past terms in 6.041 lecture for demonstrating properties of the Bernoulli process. Other CDS application development efforts that took place this year include those described in [Piscitello 91], [Turcotte 91], and [Nunez 91].

The LAS application was limited in scope and functionality, mostly due to the shortcomings of LAS; these deficiencies prevented the application from being used by a wide audience — only someone familiar with its operation could safely run it. The problems which relate specifically to the Bernoulli application were:

- The system call facility in LAS did not allow for the dynamic alteration of the command to be executed. The Bernoulli application relies on an external program, which displays a simulation of the Bernoulli process (a Bernoulli "bus line"); thus, any execution of this program from inside LAS was statically bound, during the creation of the lecture, and could not reflect any current state (for example, the probability of a success).

  Figure 1-1 shows a screen from the LAS Bernoulli application, illustrating this problem. The user may only elect to start the "bus line" simulation with one of the given four probabilities, and cannot change that parameter.

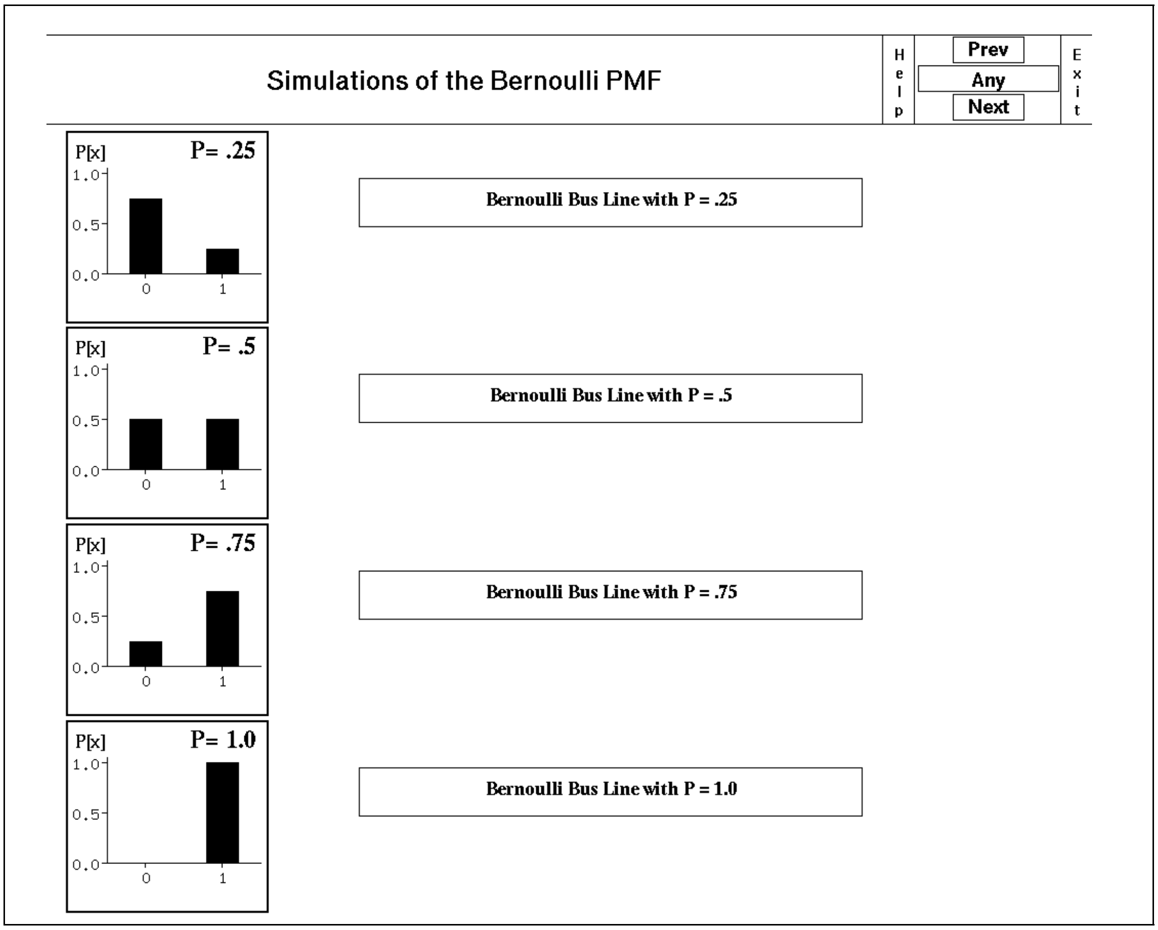

Figure 1-1: Starting Bernoulli Simulations in LAS

- LAS offered no method of interprocess communication (IPC), and therefore could not notify external programs of any change in state. For example, in Figure 1-1, once a simulation was started, its probability of success could not be altered.

- The LAS presentation had few safeguards, and could easily be "broken" by someone unfamiliar with it — for example, the user could inadvertently start multiple Bernoulli simulations, and be unable to stop all of them.

CDS is more powerful, flexible, and simply does a better job at presenting the material. Its socket-based IPC facility allows for the dynamic control of external programs, and the Scheme interpreter gives the application developer a method of preventing undesired or harmful activity, in the form of state variables and conditional execution.

Until recently, there existed no "real" applications that ran on CDS, only short tests and demonstrations. In effect, this application was needed to verify that CDS worked as designed, was sufficiently useful, and provided the necessary building blocks for complex presentations. In addition, as LAS was being phased out of use, it was important to port over existing LAS applications into CDS; the Bernoulli application was already a part of teaching 6.041, and thus there existed an immediate need to have this particular application converted.

The final, and perhaps most important objective of this thesis project is to gain experience in developing an involved CDS application. This knowledge is invaluable to the project, because it provides the project with some feedback concerning the real usability of CDS. Future application developers will benefit directly from this thesis, both because CDS will be an improved environment, and because a set of development guidelines will be defined.

1.3 Document Organization

Chapter 2 describes the CDS development environment, and includes guidelines for application programming in CDS.

Chapter 3 discusses some background theory and methods of analysis relating to the Bernoulli process.

Chapter 4 presents the design and implementation of the Bernoulli application, including a comparison between it and its LAS predecessor. Extensions to the CDS development environment that were made to support this application are also included.

Chapter 5 explains what needs to be done in the future, both for the application and CDS, and what has been learned during the development process.

Appendix A is a short tutorial describing the creation of a simple CDS application. The sample presentation, entitled *SineWave*, plots a sine curve and allows the varying of frequency and amplitude. The CDS source listing of the application is also included.

Appendix B contains the full source code listings (Scheme and CDS) of the Bernoulli application.

Appendix C contains CDS system code (C++) that implements interprocess communication, one of the most powerful features of CDS. This code has been incorporated into the CDS executable, as an extension to the CDS modelling environment, and was written during the development of the Bernoulli application.

Chapter 2

Application Programming in CDS

This chapter briefly characterizes the internal structure of CDS, and describes the user interface objects used to create an application. It is meant as an introduction to CDS and its capabilities, from a developer's point of view. Lastly, guidelines for application development are presented. These guidelines are the result of my experiences in developing one of the first CDS applications.

2.1 System Architecture

CDS consists of three basic modules, each of which has a specific and well-defined purpose. Figure 2-1 depicts the relationships among these components, which are described below.

The Navigator: responsible for loading an application, organizing it into pages, and providing an outline which is used to travel through the presentation. [1]

[1] The design and implementation of the Navigator is the topic of [Wang 91].

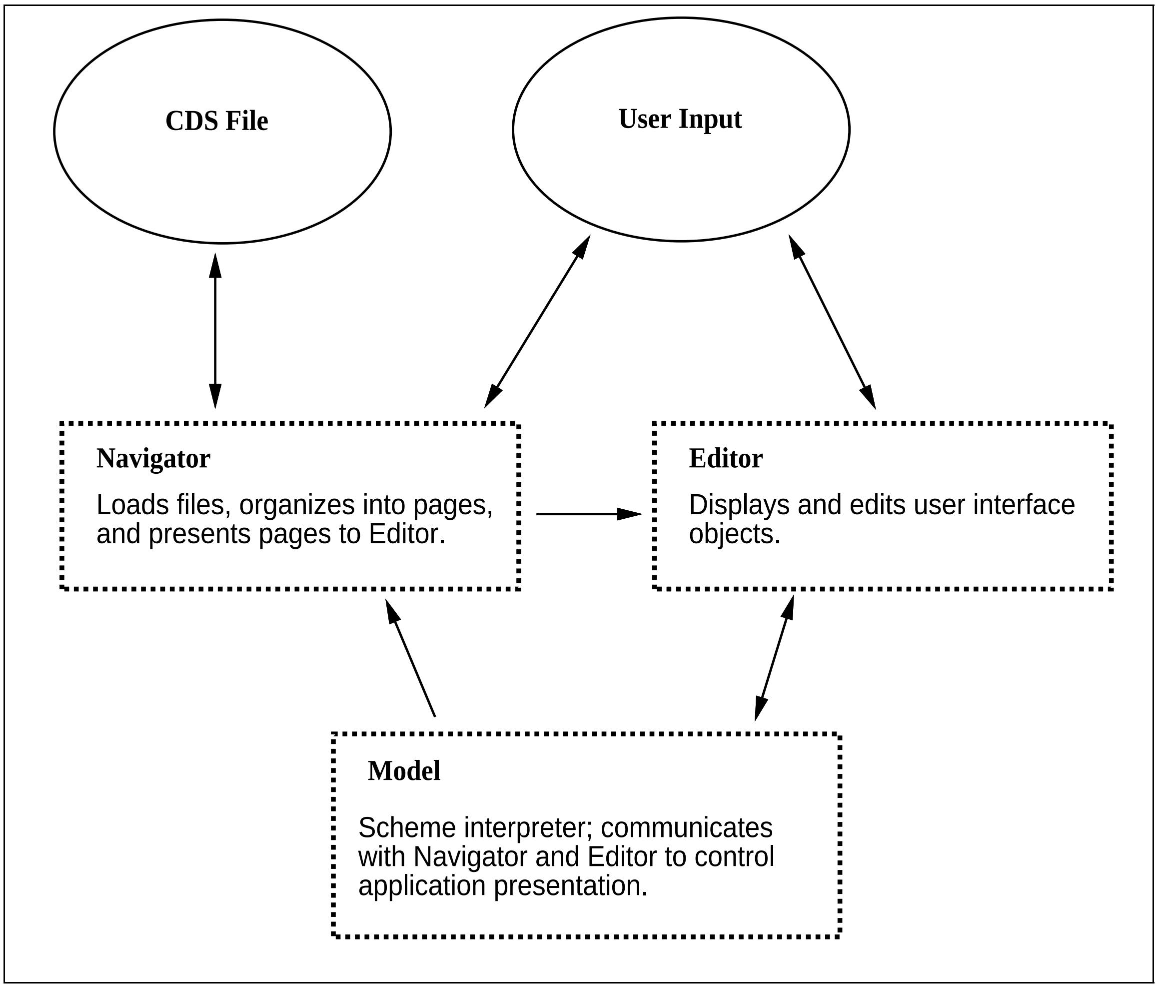

Figure 2-1: CDS Internal Architecture

The Editor: performs interactive, graphical editing of an application, with WYSIWYG functionality. In addition, it controls the display of the user interface objects associated with the current page, and handles most user input.

The Model: provides a mathematical modelling environment. It also allows user interface objects to interact with the user and other objects, and is capable of executing and communicating with external programs. [2]

[2] This facility was added during this thesis project; see Chapter 4 for details.

2.2 User Interface Objects

Each CDS application is organized into pages, which contain the user interface objects and instructions that define the execution of the application. User interface objects are made up of a displayable component (an X Window widget [3]), and Scheme code which defines its interactions with the Model, known as the object's *actions*. An object's action code is sent to the Model for evaluation whenever a certain Model or user-initiated event takes place on that object.

There are eight different classes of CDS user interface objects, six of which are shown in Figure 2-2. [4] Following is a description of each object and its associated actions.

Page: Although not explicitly presented as a user interface object, the Page does have Model actions, which are executed during Page Construction, Entry, and Exit.

TextTool: An object inherited from LAS, the TextTool is the preferred method for displaying formatted text; it provides automatic font resizing, word wrap, centering, boldface, italics, super- and sub-scripts, and a complete mathematical symbol library. The TextTool is documented thoroughly in [Giang 89].

Label: A dual purpose object, the Label can either contain simple text (in a single font, without complex formatting options) or a picture which is in X Windows pixmap format.

Scale: The Scale allows the selection of a variable's value, from a given range. Also referred to as sliders, Scales can be horizontal or vertical. A Scale's

[3] A widget is the basic X Toolkit structure that handles screen output. Widgets are covered in in more detail in [Young 90].

[4] This figure is a page taken from an application designed to be an introduction to CDS, and is located in **`/mit/cds/demo/help.cds`**.

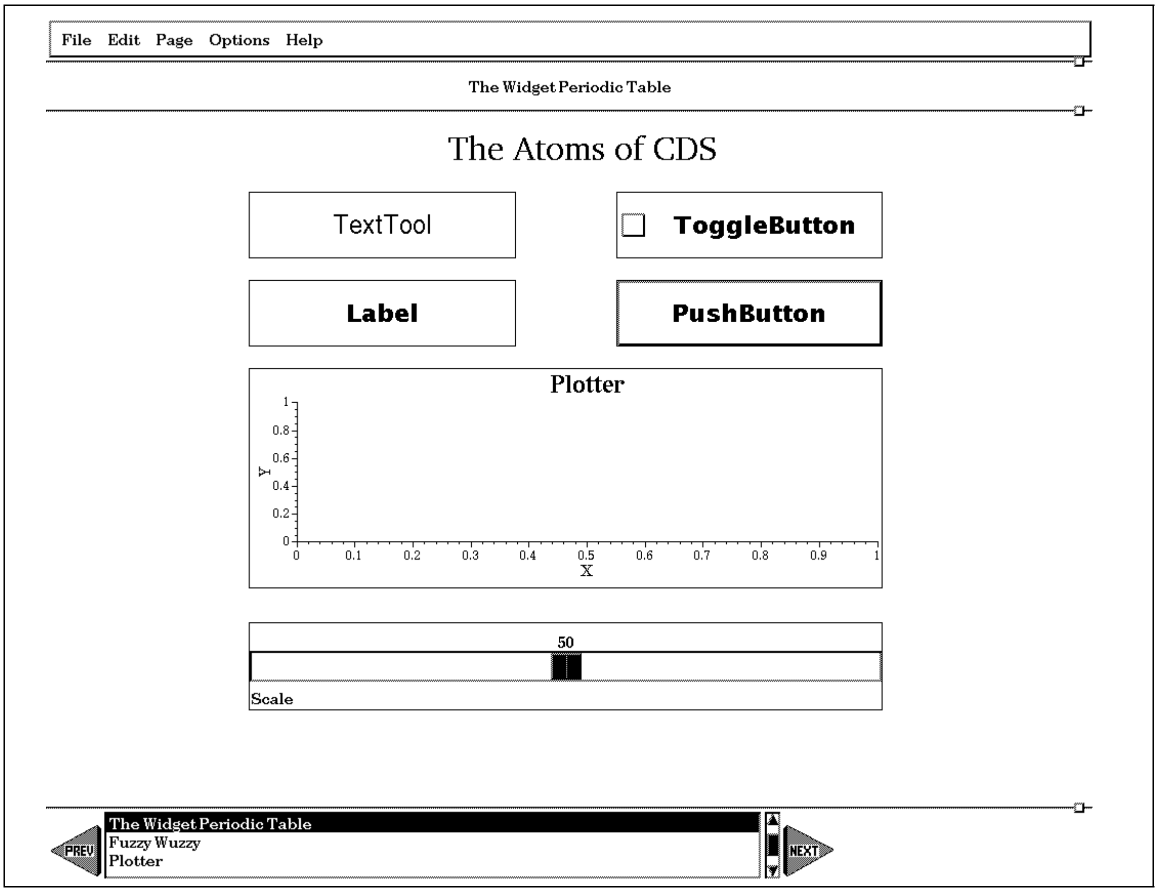

Figure 2-2: CDS User Interface Objects: Appearance

Drag action is send to the Model whenever the Scale's pointer is moved.

PushButton: A PushButton is a simple and convenient method of obtaining user input, in the form of a mouse click; its title is a Label object. The **Activate** action of a PushButton is evaluated when the left mouse button is clicked and released on its label. **Arm** is executed when the mouse button is clicked; **Disarm** when the mouse button is released.

Plotter: The Plotter's purpose is simply to plot data; it can currently display two types of graphs, a two-dimensional X-Y plot, or a bar chart. (Other plot types, such as scatter plot, contour plot, and 3-dimensional figures are planned.) Its features include automatic axis marking, legend display, zooming in and out of a plot, and logarithmic or linear axis stepping.

TextInput: When an application needs more complex input from the user than the Scale or buttons can provide, a TextInput object may be used.

ToggleButton: The ToggleButton is similar to the PushButton, except that it contains state: analogous to a light switch, it is either on, or off.

2.3 Modelling Environment

By using only the user interface objects, an application developer cannot accomplish much except for page layout, and perhaps creating a presentation consisting of "viewgraph" screens containing text objects and bitmaps. It is the Model which gives developers the power and flexibility to create an interactive, dynamic presentation. Essentially, the Model is the glue which holds an application together.

The CDS Model is built on a Scheme interpreter, which currently supports a substantial subset of Scheme, as well as a number of CDS-specific procedures; this combination is known as CmScheme. [5] CmScheme contains a full complement of mathematical functions, standard Scheme language constructs, interface object manipulation routines, subprocess calling and communication, and presentation control mechanisms. [CDS User] contains a reference manual documenting all of the built-in CmScheme primitive functions. [6]

[5] The design and implementation of the CDS Scheme interpreter was done by Sourabh Niyogi 93 during December 1990-January 1991; extensions have since been added by all members of Project CDS.

[6] This information is also available inside of CDS, in the Help system, the creation of which was largely due the efforts of Marc Schaub 93.

2.4 Development Guidelines

As one of the first people to actually use CDS as a development platform, part of my responsibility was to arrive at a set of suggestions and protocols that will make it easier for future developers to create useful applications. The following recommendations cover a broad area, from general procedures to follow, layout hints, to coding practices and style guidelines.

Appendix A takes a more in-depth look at the development process, demonstrating the steps taken to create a simple presentation, and showing these principles in use.

- As with any type of presentation, it is wise to plan ahead when designing a CDS application. An important first step is page layout: decide what each page should accomplish, what objects need to be on the page, and where they are located.

- Keep a consistent "look" throughout an application. Use a small set of compatible fonts, and present a uniform interface to the reader. Pages that do parallel tasks should resemble each other. These steps prevent the reader from being confused by an incoherent application.

- On the other hand, do not restrict the application so much that it becomes boring to look at. One of CDS' best features is its ability to present information in an attractive, inviting format — take advantage of this.

- When writing Scheme code that is executed inside an object's actions, try to minimize the amount of code that actually resides inside the object. Instead, define a procedure external [7] to the object, and have the actions refer to this. This does incur a small penalty in execution time, but

[7]The code can be placed either in a separate Scheme file, or in a Page Constructor action.

allows for the reuse of code, and makes the code easier to edit, because it is in one central location.

- When naming items in the application — graphical objects, Scheme procedures, or variables — use a unique, descriptive name. By keeping names unique, problems with multiple references are avoided; descriptive names help to keep track of an item's type, as well as aiding others in modifying the application. For instance, all Scale objects might be named **`scale_variable`**, where **`variable`** is the Scheme variable the Scale modifies.

Chapter 3

The Bernoulli Process

The Bernoulli process is a simple and well-understood probabilistic process; it is one of the five basic types of stochastic processes. [1] One of its most attractive features is that it can be completely described analytically. In addition, its assumptions and conditions fit many real-world situations, and many interesting and useful experiments can be modelled as a Bernoulli process. One general example is that of capacity planning for handling events which occur randomly.

Computer simulations of the process and automated analytical tools can be of great pedagogical value; these help students to develop an understanding of the workings of this type of model, its limitations, and the applications of Bernoulli processes in solving a large range of problems.

3.1 Basic Definitions

The remainder of this chapter assumes that the reader is familiar with the basic concepts in the applied probability. In particular, the reader should

[1] The other four being: finite sampling, uniform, Poisson, and Weiner, as given in [Baclawski 89, page 30].

be familiar with the following concepts: random variables, probability mass functions (PMF), expected value, and variance. All of this, and much more, can be found in [Drake 67], *Fundamentals of Applied Probability Theory*, by Alvin W. Drake. [2]

An individual Bernoulli trial results in an experimental value of discrete random variable x, which can be described by the PMF

$$p_x(x_0) = \begin{cases} 1-P & \text{if } x_0 = 0 \\ P & \text{if } x_0 = 1 \\ 0 & \text{otherwise} \end{cases}$$

As defined above, random variable x is termed a Bernoulli random variable. Its expectation $E(x)$ and variance σ_x^2 are as follows:

$$E(x) = P \qquad\qquad \sigma_x^2 = P(1-P)$$

The result of a Bernoulli trial is often referred to as a success (or arrival) when the experimental value of x is one, and a failure (or non-arrival) when the experimental value of x is zero.

"A Bernoulli process is a series of independent Bernoulli trials, each with the same probability of success". A simple example of a Bernoulli Process is the process of repeatedly tossing a coin, where each toss has a probability P of landing heads up, denoting success.

The independence of each Bernoulli trial simplifies the mathematical analysis of Bernoulli processes, due to the following properties of random variables:

$$\begin{aligned} E(x+y) &= E(x) + E(y) && \text{for all } x, y \\ E(xy) &= E(x)E(y) && \text{for independent } x, y \end{aligned}$$

[2] In fact, a great deal of this chapter based on Chapter 4 of this very book; any quoted text in this chapter is from [Drake 67], unless otherwise noted.

3.2 Typical Experiments Defined on a Bernoulli Process

There are two practical questions one can ask, or experiments that can be performed, on a Bernoulli process: can we derive a distribution that describes the number successes in a given number of Bernoulli trials, and, can we derive a distribution that describes the number of trials before a certain number of successes occur. The answers to these problems can be easily found, using the binomial and Pascal probability mass functions, respectively.

The Binomial PMF

Define discrete random variable k as the number of successes in n Bernoulli trials; k is therefore the sum of n independent Bernoulli random variables, and is described by the binomial PMF: (named so because of the binomial expansion)

$$p_k(k_0) = \binom{n}{k_0} P^{k_0}(1-P)^{n-k_0} \qquad k_0 = 0, 1, 2, \ldots, n$$

Due to the previously encountered properties of the expected value and variance functions as applied to a sum/product of linearly independent random variables, $E(k)$ and σ_k^2 are then equal to

$$E(k) = nE(x) = nP \qquad \sigma_k^2 = n\sigma_x^2 = nP(1-P)$$

The Geometric and Pascal PMF

By referring to the successes in a Bernoulli process as arrivals, it is convenient to define discrete random variable l_1 as the number of Bernoulli trials up to and including the first arrival; l_1 is known as the first-order interarrival time, and is described by the geometric PMF:

$$p_{l_1}(l) = P(1-P)^{l-1} \qquad l = 1, 2, \ldots$$

The successive terms of this PMF decrease in a geometric progression, giving it its name. The expected value and variance of random variable l_1 are derived in [Drake 67, page 127], and are:

$$E(l_1) = \frac{1}{P} \qquad\qquad \sigma^2_{l_1} = \frac{1-P}{P^2}$$

"The Pascal PMF describes the higher order interarrival times for a Bernoulli process." This implies that the geometric PMF is simply a special instance of a Pascal PMF.

Random variable l_r represents the number of trials up to and including the r^{th} success, or the r^{th}-order interarrival time; l_r is the sum of r independent experimental values of l_1, and is characterized by the Pascal PMF:

$$p_{l_r}(l) = \begin{pmatrix} l-1 \\ r-1 \end{pmatrix} P^r (1-P)^{l-r} \qquad l = r, r+1, \ldots; r = 1, 2, \ldots$$

The expected value and variance of l_r are easily derived from those of l_1, and are:

$$E(l_r) = \frac{r}{P} \qquad\qquad \sigma^2_{l_r} = \frac{r(1-P)}{P^2}$$

The terminology associated with the Pascal and geometric PMF point out that the Bernoulli process is an example of a renewal process: a random process in which arrivals occur at points in time, and whose first-order interarrival times are independent random variables described by the same probability density function.

3.3 Applications of the Bernoulli Process

As stated in this chapter's introduction, one general application of the Bernoulli process is in capacity planning. For example, consider the example of scheduling city buses. The arrival of passengers at a bus stop can be modelled as a Bernoulli process; it is then possible to determine when it is necessary to send a bus to pick up the passengers after a certain number are likely to be waiting.

Another example of this type of application is in manufacturing: the arrival of packages at a storage area can be modelled as a Bernoulli process. We can then find out when a truck is needed to pick up the packages, so that there is less than a 1 percent chance of overflowing the storage facility.

These two examples are essentially the same problem — and are solvable through analysis of their mathematical model, a Bernoulli process; the calculations involved in solving these problems are beyond the scope of this document, however.

Chapter 4

Design and Implementation of the Bernoulli Application

4.1 Design Goals

Both the CDS and LAS Bernoulli applications were designed with the same general goals; however, in order to demonstrate the enhanced features of CDS, the CDS application needed to show definite advancements in usability and function, to better satisfy these requirements, which are:

- Present the underlying concepts of a Bernoulli process in an understandable, interesting manner.
- Permit students to view simulations of a typical Bernoulli process, and to interactively alter its parameters.
- Show how the various probability mass functions change as the parameters are varied (such as probability of success, order).
- Display the relationships between the different types of PMFs.

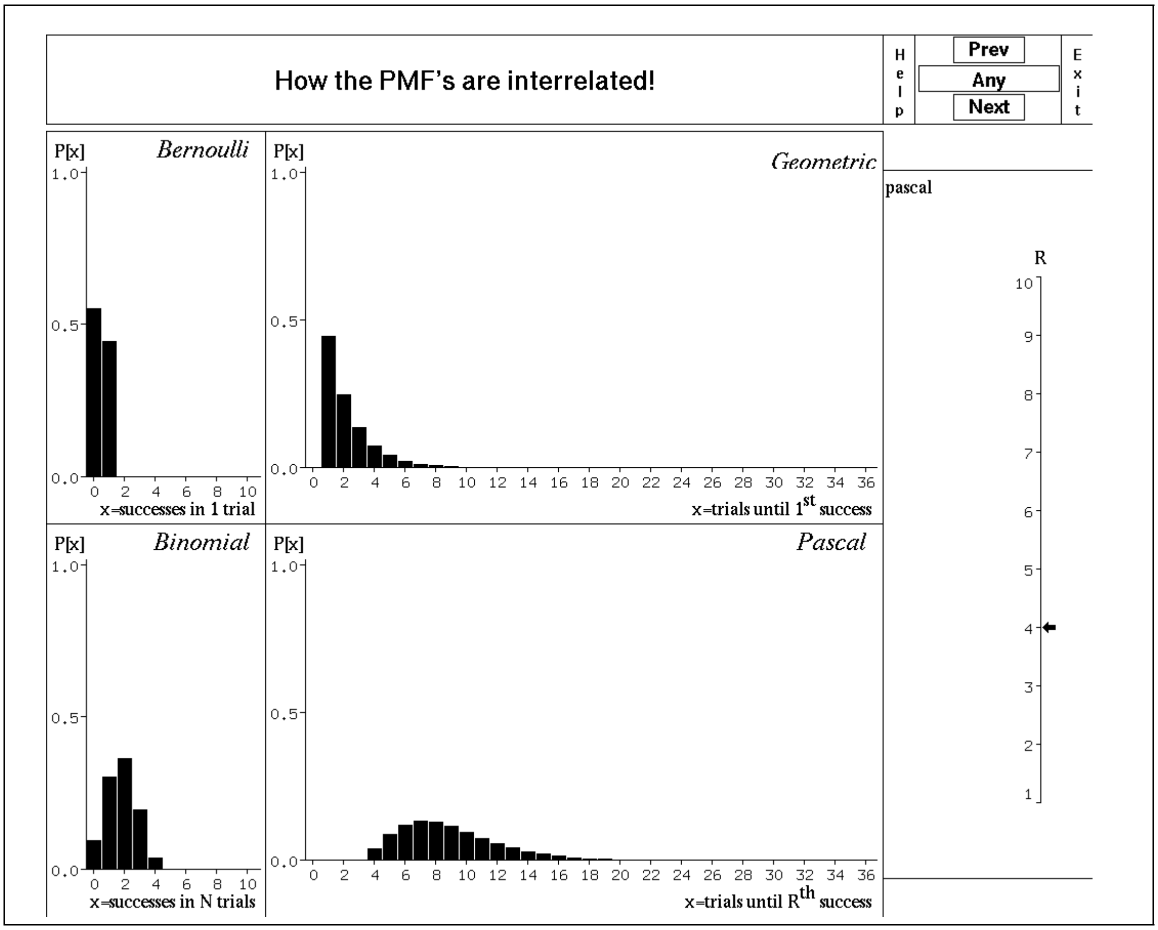

Figure 4-1: LAS: How The Bernoulli PMFs Are Related

- Relate to the 6.041 lecture on the Bernoulli process in a cohesive, clarifying way.

In retrospect, the LAS application was an innovative and well-designed lecture aid, and proved to be an excellent prototype. Figure 4-1, which depicts the connections between four different PMFs related to a Bernoulli process, shows an example of the effectiveness of the LAS presentation. However, some inherent limitations of LAS made it impractical for student use; these problems were summarized in Chapter 1. The primary object of this effort was to improve upon the LAS application in the following ways.

- Simplify and stabilize the user interface — thereby increasing the robustness of the application, and allowing it to be used by a wider audience (The LAS application's major use was as a visual aid during lecture; the goal for CDS is to allow the students to be able to use it as well.)
- Allow the initial parameters of the simulation (an external program) to be selected by the user, instead of being statically bound at the time of application creation.
- Allow the simulation parameters to be dynamically altered, during execution, through the simple manipulation of user interface objects.
- Present additional information, such as the mathematical equations that relate to the Bernoulli process.

The final consideration that factored into the design of the CDS Bernoulli application was performance — it was of obvious importance that these additional features not result in a significant loss of efficiency, or the application would fail to show that CDS was indeed superior to LAS.

4.2 Implementation Steps

The implementation of the application was divided into two main activities: developing the actual CDS presentation, and extending the CDS environment. The application expanded functionally, as CDS itself was enhanced to support extra features. The editing capabilities of CDS allowed this type of evolution to occur without the danger of losing previous work, in most instances.

Developing the actual application took part in several steps; first, an application was created that emulated the previous LAS presentation. [1] This

[1] The process involved in creating a CDS application is described in Appendix A.

was relatively uncomplicated, as CDS possessed all the requisite tools (i.e. CDS was already equivalent to LAS, regarding the needs of this application). Specifically, CDS provided the following necessary user interface objects: a plotter capable of displaying bar charts; a slider used to manipulate variables; a text output object; an interface to the Unix system call, used to start external programs; and, a mathematical modelling environment. Once the Scheme functions that computed the PMF distributions were written,[2] this application was equal in function to the LAS lecture.

However, the LAS version was much faster in calculating the PMFs as parameters were dynamically varied using sliders. Due to the interpretive nature of the Scheme evaluator, the repeated calls to distribution-computing functions were costly — CDS was much slower in updating the PMF plots. The solution to this problem was to write C++ functions which computed the distributions, and then link these with the Model. As a result of the model's well-designed internal structure, this proved to be almost trivial.

This case illustrates the parallel development that transpired during this project: just as the application was being created, the CDS environment was also undergoing change. At times, this change was benevolent, and directed at supporting this specific application. However, there were occasions when modifications to CDS were problematic, to the point of causing the Bernoulli application to stop working; these had to be dealt with accordingly.

The next step of development dealt with adding function to the existing screens of the application. One such feature, of paramount importance, was the ability of a slider to affect the parameters of the Bernoulli process simulation (an external program which displays a Bernoulli "bus line"), in addition to updating any displayed PMF. In order to accomplish this task, CDS needed

[2] I'd like to thank Jeff Turcotte 91 for writing these Scheme functions, as well as the C++ versions that were needed later.

some method of interprocess communication (IPC) — CDS had to be extended once more. This specific extension, socket-based IPC, was added by the author, and is detailed in the next section. [3]

This is perhaps the most significant perceived advantage of CDS: LAS could only start a "bus line" with static, pre-defined parameters (for example, Figure 1-1); CDS is able to update variables while the program is running, making the application much more user-friendly, as well as conveying a better understanding of how the Bernoulli process works. Figure 4-3 shows a typical use of this capability, showing an ongoing simulation of a geometric PMF. [4]

The final steps of implementation involved the creation of new screens, adding to the overall functionality of the application. The first of these screens, shown later in Figure 4-2, is one in which every aspect of the "bus line" can be controlled: speed, probability, experiment type, and arrival icon. This screen allows students to compare and contrast the various experiments defined on a Bernoulli process, and lets them directly observe how changing parameters affects the experiment's results.

Another screen new to the CDS application shows the equations which describe the four PMF types: Bernoulli, geometric, binomial, and Pascal. This screen follows the CDS equivalent of Figure 4-1 (displaying four PMF plots), and is designed to further emphasize the relationships between the PMFs, by exhibiting the similarities in their mathematical properties.

[3] In addition to changes to CDS, the actual simulation program had to be modified to accept the dynamic alteration of its state; this work was done by Edward Moriarty. The simulation program was originally written by Patricia Zeitler 89; see [Zeitler 89] for details.

[4] Note: screens referenced in this section appear later, in Section 4.4, the application description.

4.3 Enhancements Made to the CDS Modelling Environment

The major addition to CDS during this thesis project is the socket-based interprocess communication facility; five Scheme primitive functions were added to the CDS modelling environment allowing two-way connections with external programs. The C++ code that implements these functions, as well as a summary of how to use them, appears in Appendix C. Although these functions were added to support this application's needs, they are general enough for IPC with any external program which conforms to the Unix standard input/output file usage; and, whereas using simple pipes only allows one-way communication (either read or write), sockets give CDS the capability of reading from and writing to the same program.

Other Scheme additions that were designed for use in this specific application are the four PMF primitives; these are documented in the CDS help system as well as [CDS User].

4.4 Application Description

The final Bernoulli application consists of seven screens, with three "action" screens that are the main focus of the presentation. These seven screens are:

1. The application's title page: *The Bernoulli Process is our friend.* (This screen is not pictured.)

2. An introductory page which defines the terms **Bernoulli random variable**, and **Bernoulli process**. See Chapter 3 for these definitions. (This screen is not pictured.)

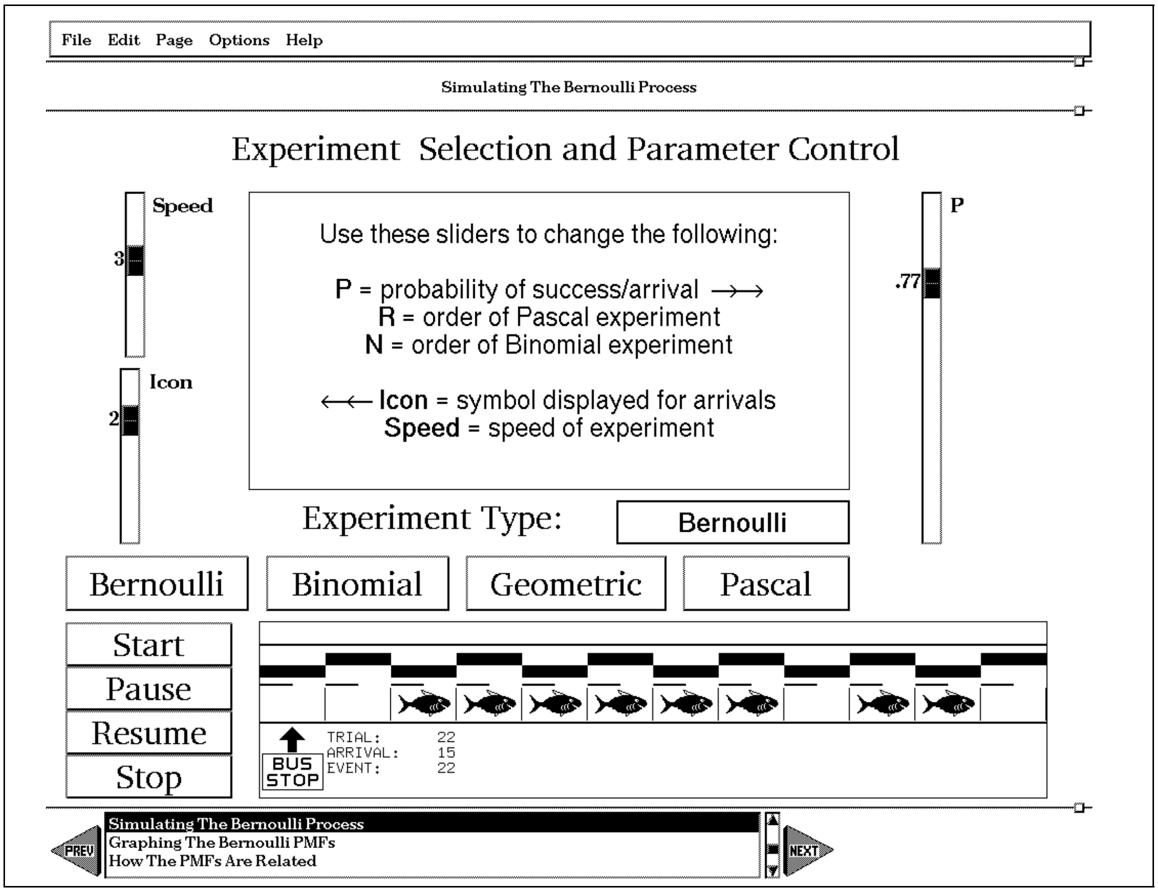

Figure 4-2: CDS: Controlling the Bernoulli Simulation

3. The simulation screen, contained in Figure 4-2, entitled *Simulations of the Bernoulli Process*. This screen allows the user to dynamically alter any parameter of the simulation, and immediately see the results.

4. The application's most-utilized screen, shown in Figure 4-3, which displays one of four PMF plots (depending on the user's selection), allows the parameters of the PMF to be varied (probability and order), and also can display the simulation process. In the LAS lecture, this screen was divided into five separate screens: one for each PMF, and a simulation starting screen, which was severely limited (Figure 1-1).

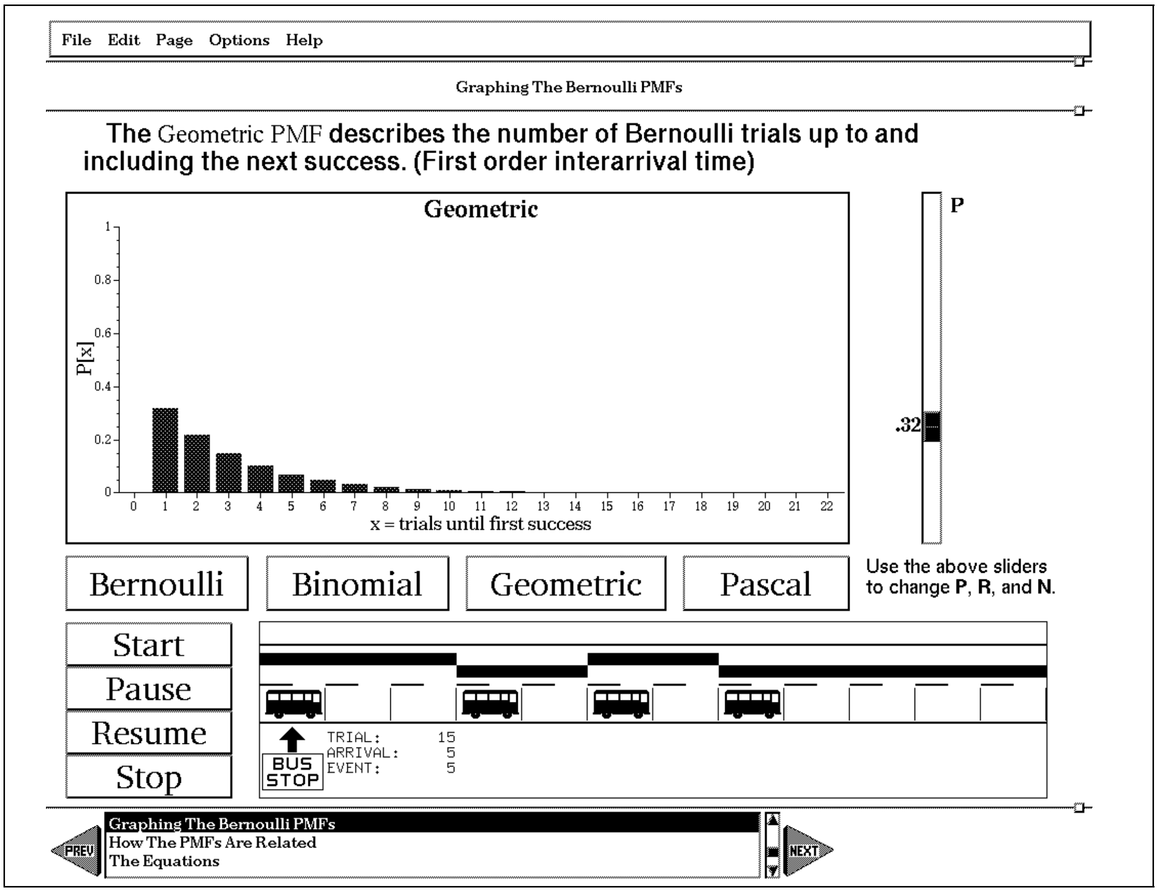

Figure 4-3: Bernoulli: Graphing the Bernoulli PMFs

5. A page which shows the inter-relationships of the four PMFs; the LAS version is shown in Figure 4-1, its CDS counterpart in Figure 4-4.

6. The equations page; for each PMF, the expected value and variance is displayed (using the mathematical formatting capabilities of the TextTool object). These formulas are exactly the same as those found in Chapter 3. (This screen is not pictured.)

7. Finally, an acknowledgement page, giving credit to the members of Project CDS and others who have made this application, and the entire CDS development platform, a reality. Figure 4-5 contains this screen.

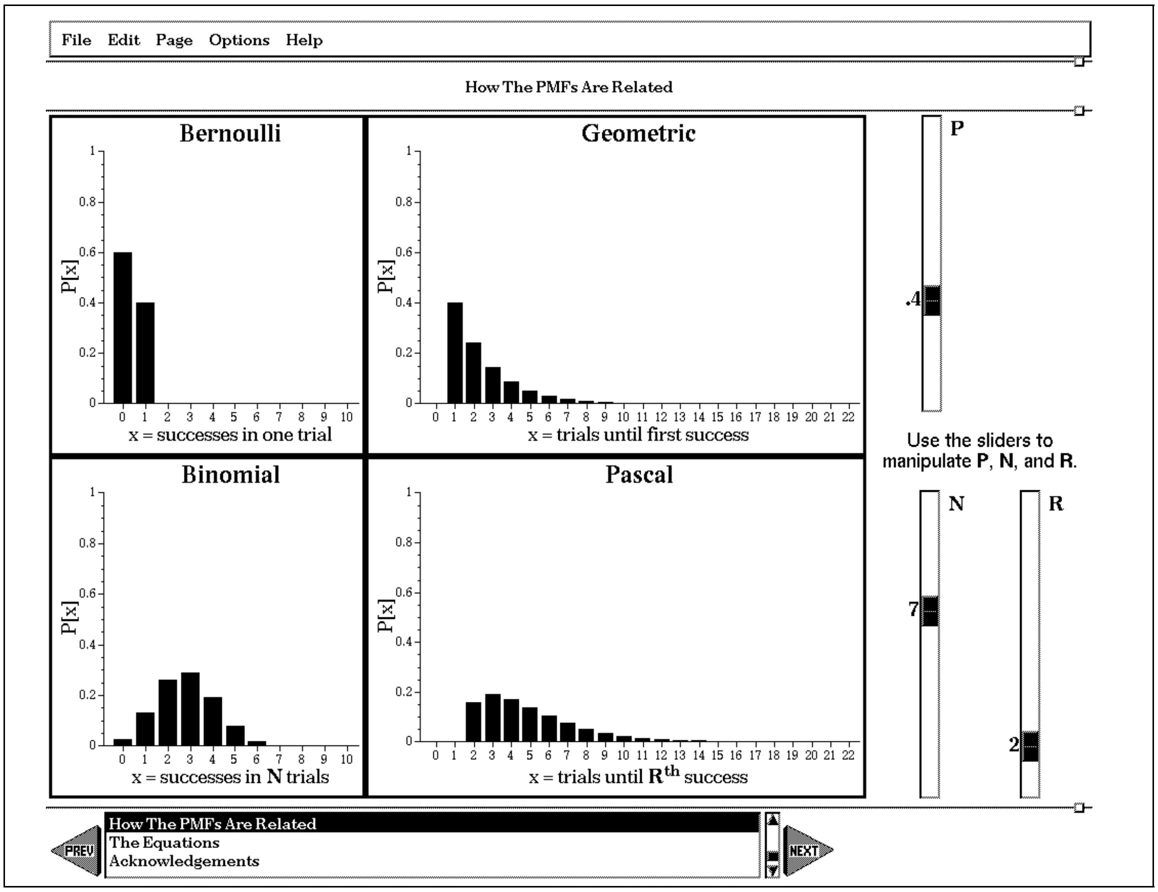

Figure 4-4: Bernoulli: How the PMFs are Interrelated

4.5 Summary of Results

The evaluation of the CDS Bernoulli application, as compared to its LAS predecessor, falls into two categories: those aspects related to using CDS versus LAS, and the intrinsic differences in the applications.

There is little inherent distinction between the two applications, because they both were designed to satisfy the same criteria, as listed at the start of this chapter, and because the CDS application was created using the LAS lecture as a working prototype. The CDS application does however have two extra screens which are not present in the LAS lecture: the simulation control

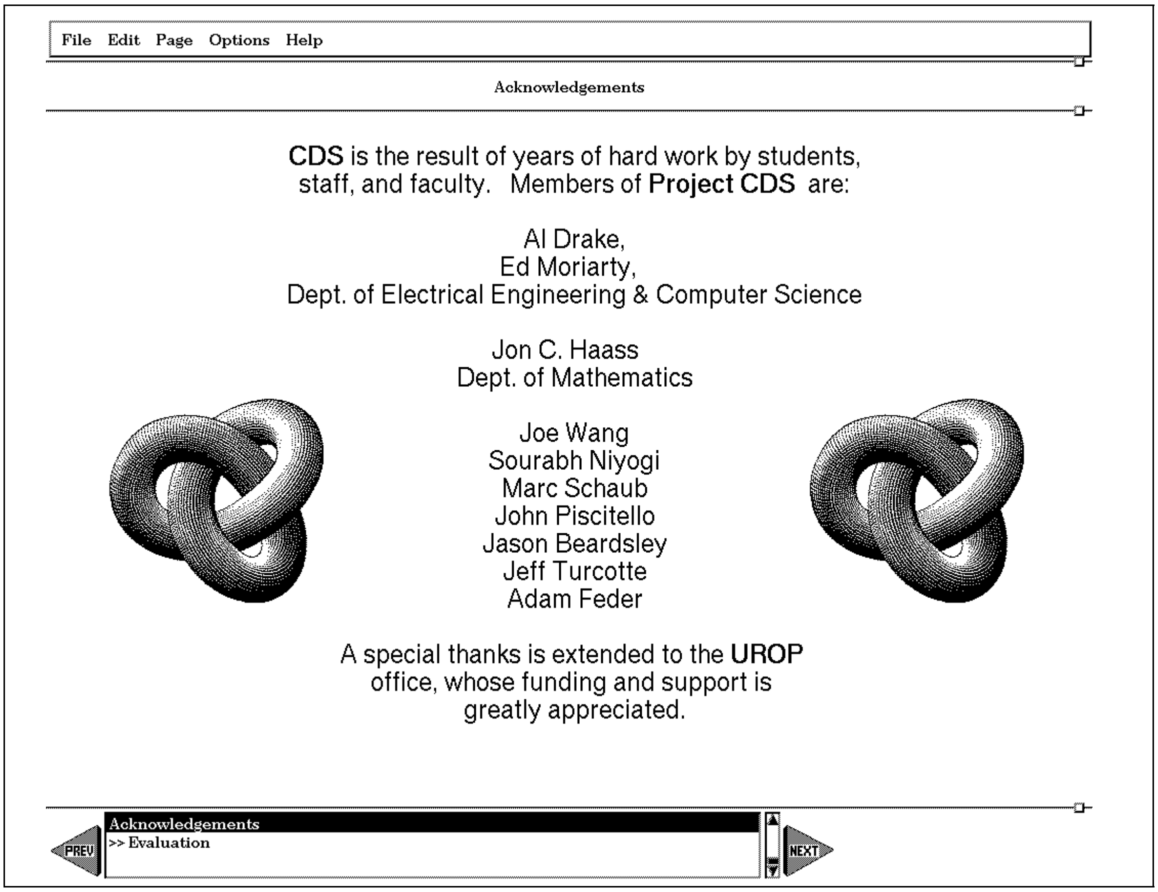

Figure 4-5: Bernoulli: Credits Screen

page, and the equation page. The equations could have been easily added to the LAS version, since LAS also has the capability of producing formatted text output. The simulation screen, though, relies on CDS' interprocess communication, and would therefore not be easily added to the LAS presentation, if at all.

Functionally, the CDS application is far superior — due in great part to the IPC Scheme primitives written during the development of the application. Performance-wise, LAS still holds a small but noticeable advantage; the extra power and flexibility of CDS are not completely free. Another issue where CDS outshines LAS is in modifiability; changing the layout of the LAS application is a complicated, time consuming process, where in CDS the layout can be changed immediately, even when running the application. This in itself is quite valuable, because it allows the application to be customized for use in other classes, or for a homework assignment, without much effort.

The CDS application is also usable by a wider audience, including students; LAS was fragile, and prone to crashing when used incorrectly. Instead of being limited to in-class demonstrations, the CDS presentation can be used outside of class by students at their own pace.

Overall, the problems with LAS that limited the application's usefulness and use have been eliminated in CDS, and the resultant improvement in the Bernoulli application can be seen as proof that CDS is indeed a better development environment. Whether or not CDS truly meets its own goals is a question that will be discussed in the next chapter.

Chapter 5

Conclusions and Discussion

5.1 Future Development of the Bernoulli Application

The Bernoulli application, as it now stands, is a finished, functional presentation. However, this does not mean it cannot stand some improvement — the applications developed subsequent to the completion of Bernoulli [1] show that CDS is capable of a higher level of user-friendliness, as well as being able to present a great deal more information. At the time of development, CDS was in also in a state of evolution, and thus not every feature available in the current version of CDS was able to find its way into the Bernoulli application.

For example, the Scheme primitives written by John Piscitello that enable the addition of menu items are not employed by this application, but probably should be used in the future, to add help information to each page. The same idea could be applied to controlling the simulation process, or displaying the PMFs — it is up to future developers to decide this.

Another area where the Bernoulli application may be improved is in its

[1] These applications are the topics of [Piscitello 91], [Turcotte 91], and [Nunez 91]

overall layout, with regards to the needs of 6.041. Since CDS allows for easy editing of a page's appearance, the application could be tailored to meet Professor Drake's specific requirements, without losing any functionality (i.e., he could make it look just the way he wants it to look). Also related to its role in 6.041, the application may be extended further to better fit the Bernoulli process lecture, either through more pages with more examples, or with comparisons to other types of probabilistic processes (for instance, Poisson).

5.2 Future Directions for CDS

CDS is now, at last, a viable development platform, proven in part by the existence and utility of the Bernoulli application. CDS is not, however, a finished product. As one of the first CDS application programmers, I have noted much overall improvement in the development environment, for which credit is due to the entire project team. And, I feel that CDS does not have much further to go before it is capable of meeting its goal of being easy enough to use by educators — essentially, the ability to create the applications is there, but it needs some polish to be used effectively.

My recommendations for future CDS system development, running from general areas needing improvement, outright bugs, to specific features that I could have benefited from, are (in no special order):

- The TextTool object's list of fonts and useful sizes is somewhat short; the X Window system has a large number of fonts that the TextTool does not yet use — these should be added.

- Page titles, and any other displayed text (i.e. menu titles, PushButton labels), should be capable of the same formatting features as the TextTool user interface object. By presenting a consistent, powerful method of displaying text, every application will look better.

- Given that, plain text Labels are somewhat redundant, and can safely be removed — however, pixmap Labels should be kept; in addition, if Labels could automatically scale their pixmaps, applications that use them would look better when resized.

- The performance of the Model is the main processing bottleneck in most complex applications; it needs to be speeded up. [2]

- The Model's Scheme interpreter is currently far from being "true Scheme", as defined by the 6.001 textbook *Structure and Interpretation of Computer Programs*, by Gerald J. Sussman et al. If CDS is advertised as having a Scheme interpreter, then this interpreter must implement a fairly complete subset of what people think Scheme is, or they will become frustrated with the entire system. This is a top priority.

- The Editor currently does an excessive amount of widget creation, using up more and more memory as an application executes. For instance, when a widget is shown using the **`show-widget`** primitive, it is actually re-created. There must be a better solution to this problem.

- The interprocess communication facility is not completely debugged; occasionally, data is lost when being sent to an external program (specifically, the Bernoulli simulation). All efforts to trace this to CDS have failed, yet the possibility remains that the bug is not in the simulation program.

[2] Efforts to rewrite the Model, using more efficient and modular coding techniques are currently underway, and should see fruition by the end of this summer. Integrating an existing Scheme interpreter is also being researched.

5.3 Conclusion

The primary goal of this thesis project was to create a CDS application dealing with the Bernoulli process; indeed, this was accomplished. In the course of pursuing that goal, a number of other secondary objectives were also encountered, including: extending CDS so the application could function; creating guidelines and suggestions for future application developers (so that they can benefit from my experience, perhaps flattening the CDS learning curve); and evaluating the current status of the system, through the eyes of an application programmer.

In addition to these, other less well-defined issues arose: working productively in a group project, and making the switch from system development to application development — viewing CDS from both a programmer's and user's perspective. The experience I have gained in these areas will be directly applicable for the rest of my life; I view this as the most educational aspect of this thesis.

Appendix A

Developing a Sample CDS Application

This appendix describes the creation of a sample CDS application, *SineWave*, which plots the sine function, and allows the user to change the curve's frequency and amplitude. The application will consist of a single page, containing five objects: a Plotter, two Scales, a PushButton, and a TextTool.

Note: I assume that the reader is somewhat familiar with the Athena environment, especially in use of the mouse, dialog boxes, and menu interfaces. It is also very helpful, although not required, to have read the CDS user documentation, [CDS User]; eventually, this appendix will be assimilated into that document.

A.1 Application Development: Step by Step

The first step in application programming is, naturally, running the CDS executable. This is accomplished by first logging into an Athena workstation [1],

[1] Currently, CDS runs only on VAX machines; efforts to support other hardware platforms are underway. I recommend using a VaxStation 3100, if possible.

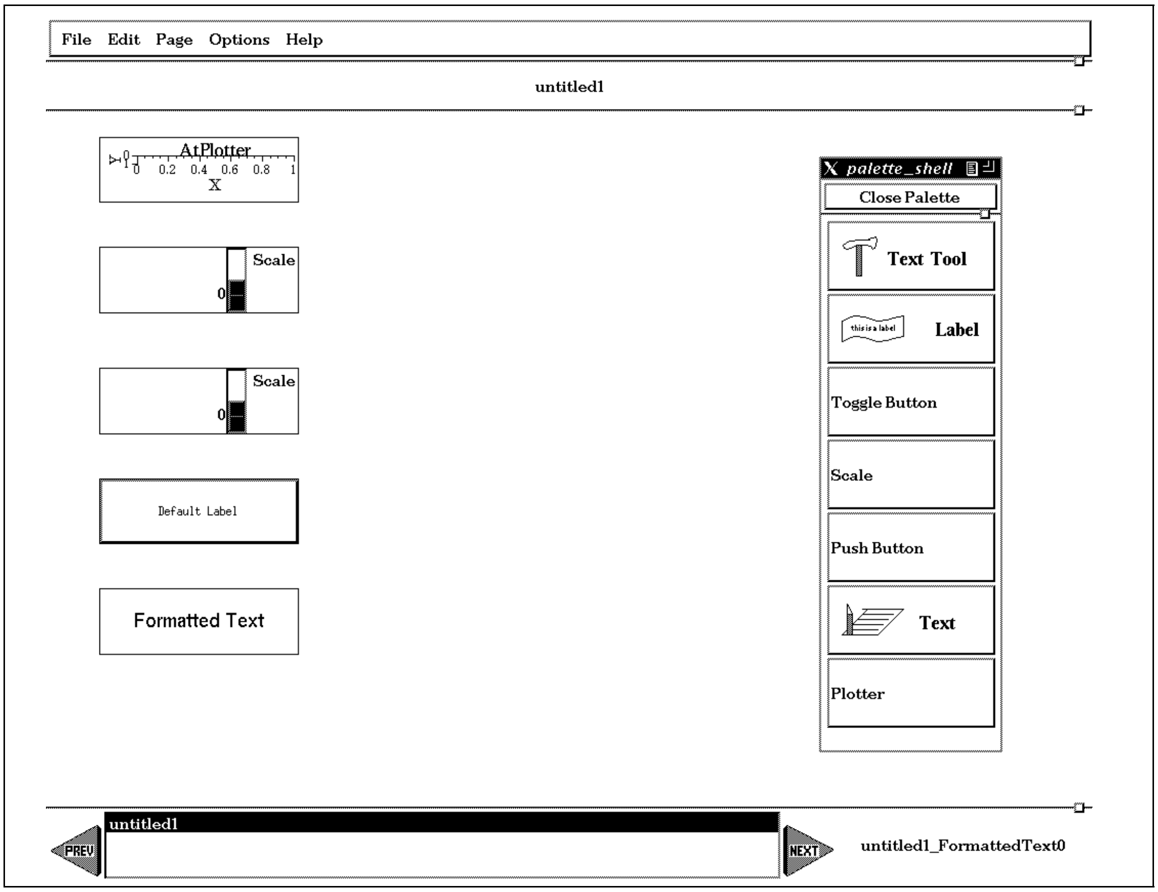

Figure A-1: SineWave Application: After Creating Objects

and following the instructions given in [CDS User] for bringing up the CDS window.

Creating the User Interface Objects

From the main menu, selecting **Edit**, and then **Create**, displays the CDS Palette, which contains visual representations of all selectable graphical objects. By clicking the left mouse button on the pictures in the Palette, objects are created and placed on the current page. Figure A-1 shows the screen after the creation of a Plotter, two Scales, a PushButton, and a TextTool. At this point, no more objects are required, so the Palette may be closed.

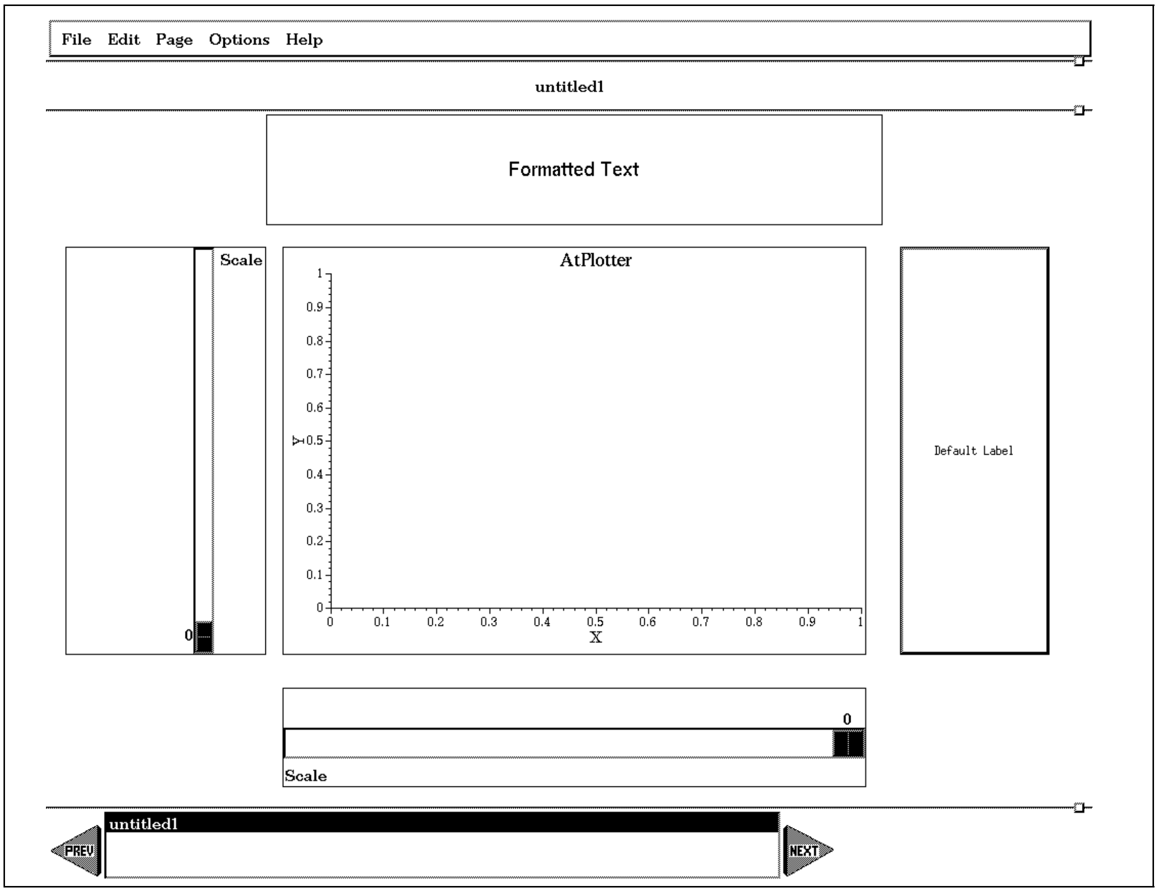

Figure A-2: SineWave Application: After Moving and Resizing

Customizing the Visual Appearance

The next step is in fixing the page's layout — this involves moving, sizing, and editing the objects. Resizing and positioning is accomplished using only the middle button: to move an object, click on the object, and drag it around the page; to resize, click on an object's border, and drag its border. Figure A-2 shows the screen after this step.

To edit an object's resources, or characteristics, double-click the middle button on it; this brings up a dialog box containing all of object's modifiable resources. Figure A-3 shows a typical dialog box, which happens to be associated with the frequency Scale, **`scale_freq`**. Through the dialog box, all

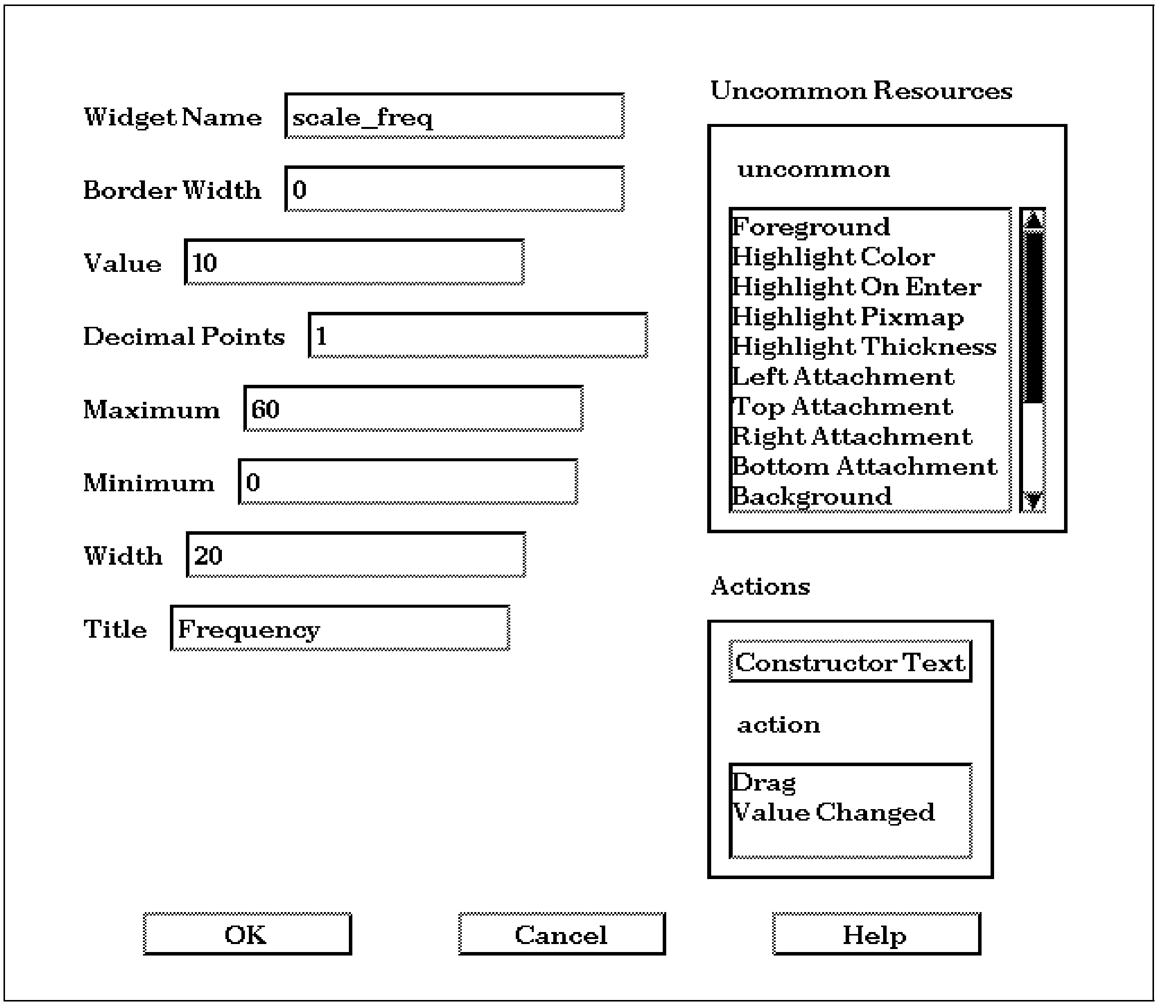

Figure A-3: A Scale Dialog Box

aspects of an object's appearance *and* Scheme actions are changeable.

The last step is to give the page a title; select **Page**, **Change Title**... and enter the page's title into the dialog box. Figure A-4 shows the final application layout, after all editing has been completed.

Programming the Actions

Once the visual appearance of the objects has been determined, their inter-

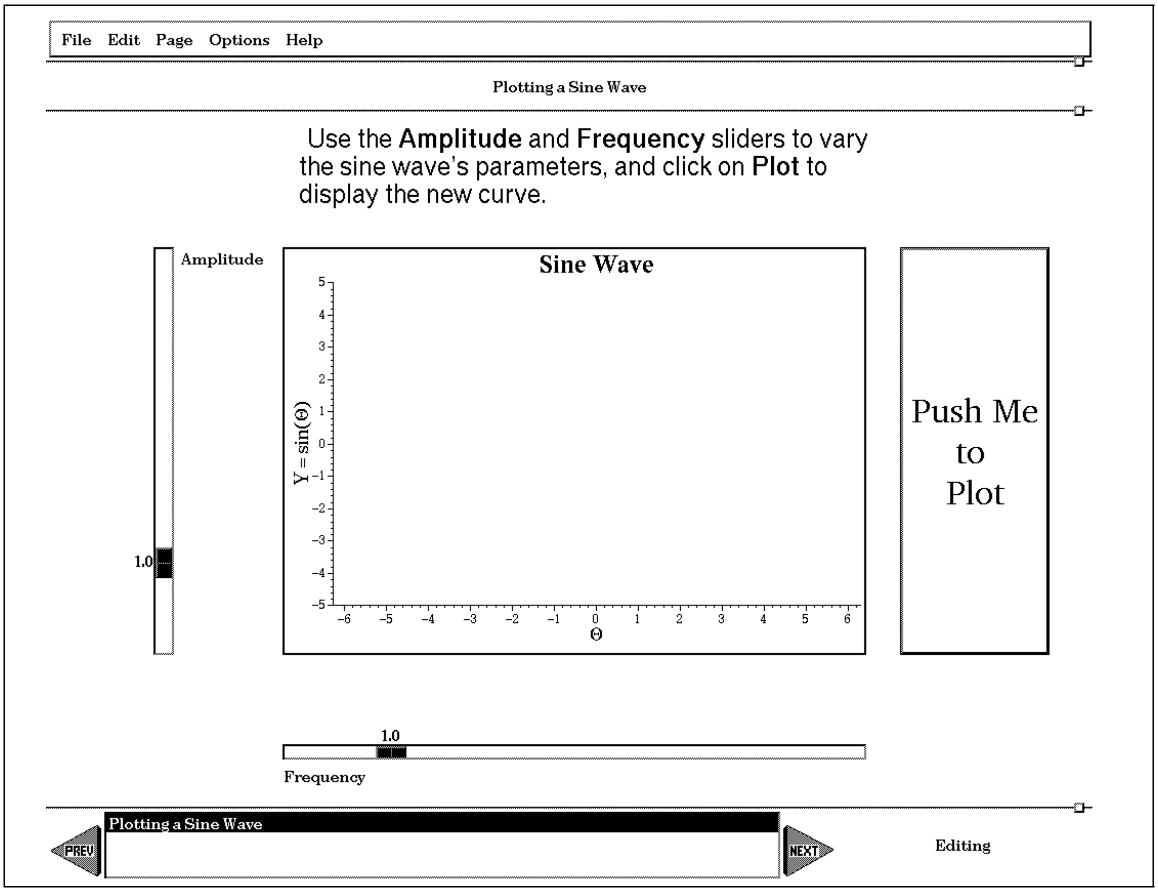

Figure A-4: SineWave Application: After Editing Resources

actions must be considered — each object's function in the application needs to be clarified, and Scheme code can then be written and placed in the objects' actions. In addition, objects which need to be referred to in these actions may need to be "renamed".

1. The Plotter displays the sine function, $Y = a \sin fx$.

2. The two Scales change the frequency f and amplitude a.

3. The PushButton performs the actual plotting, when activated by a mouse click.

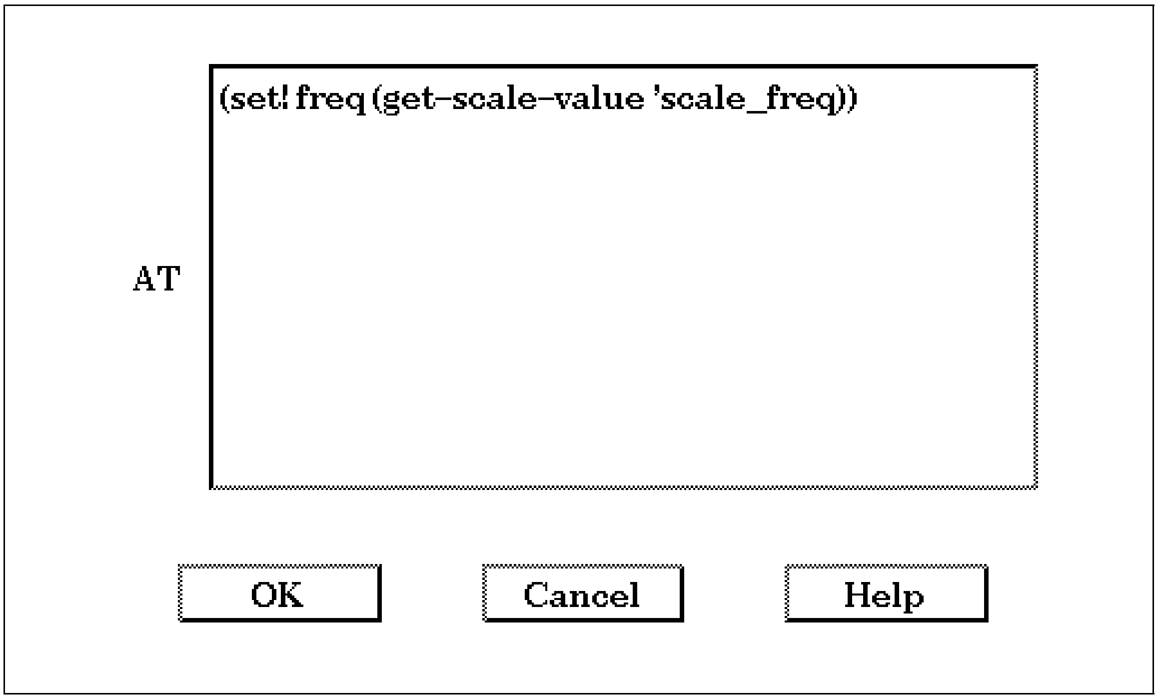

Figure A-5: Drag Action Code for the Frequency Scale

4. The TextTool describes the purpose and function of the page.

Two variables, **`freq`** and **`amp`** are needed to store the Scale parameters; these are defined later in the Page **Constructor** action.

Associating code with an object's action is simple: edit the object, by double-clicking the middle mouse button over it, and select the appropriate action to change. Another dialog box then appears, into which Scheme expressions may be entered. For example, as shown in Figure A-3 (a Scale dialog box), the possible actions of a Scale are **Drag**, and **ValueChanged**. Choosing **Drag** brings up the dialog box in Figure A-5, displaying the actual code inside **`scale_freq`** which sets **`freq`** to its new value.

A Page **Constructor** action is evaluated exactly once, during the creation of its associated Page object. Therefore, it is a good location to place code which should only be executed once — such as defining global variables, or Scheme procedures.

In addition to defining **freq** and **amp**, this application's "state variables", the Page **Constructor** also defines a Scheme convenience routine which is used to return a Scale's value:

```
; scale convenience routine
(define (get-scale-value scale)
 (/ (get-resource scale 'value)
   (^ 10 (get-resource scale 'decimalPoints))))
; initialize state variables
(define freq (get-scale-value 'scale_freq))
(define amp (get-scale-value 'scale_amp))
```

The Drag action for the amplitude Scale, **scale_amp**, is almost identical to the frequency Scale:

```
(set! amp (get-scale-value 'scale_amp))
```

The **Constructor** action of the Plotter, **plotter**, is responsible for initializing a plot variable, [2] using the **add-xyplot** primitive.

```
(add-xyplot 'plotter 'plot_sinewave)
```

The PushButton **button** contains the commands necessary to produce a plot in the Plotter:

```
; define the function to plot
(define (f x)
 (* amp (sin (* freq x))))
; plot it
(xyplot 'plot_sinewave f)
```

[2] Actually, a plot or barchart "variable" is just another type of user interface object, one that is created dynamically, and associated with a given Plotter object.

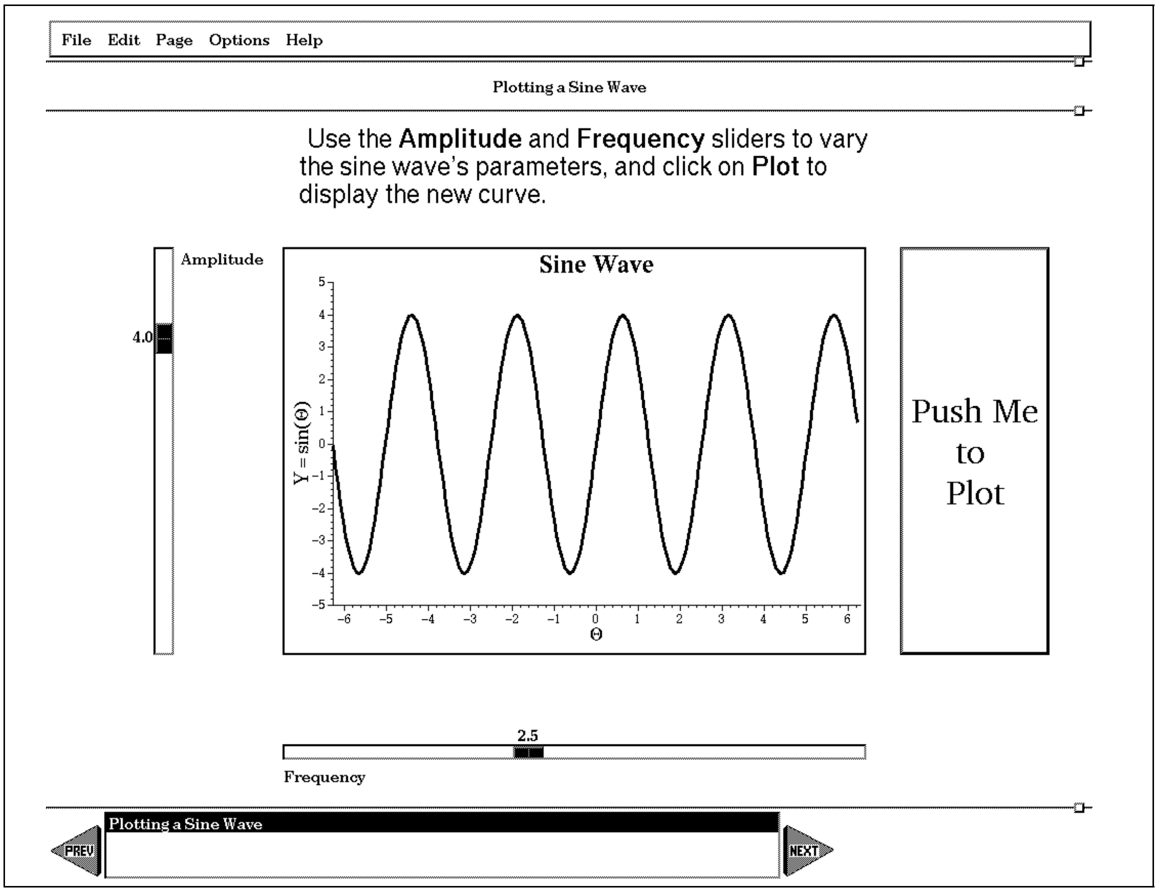

Figure A-6: The SineWave Application In Action

Finally, the Page **Entry** action is modified to activate **`button`**, so that whenever this page is displayed, the sine wave plot will be automatically updated:

```
(activate-widget 'button 'Activate)
```

With this, the SineWave application is complete, and ready to use. Figure A-6 shows a typical usage of the presentation.

A.2 The SineWave CDS File

Following is the code produced by CDS for the SineWave application. It is included here only for those interested in finding out exactly how CDS code is structured; ideally, it should never be necessary for an application developer to actually modify this "program" with an external editor. This file is **/mit/cds/demo/sinewave.cds**.

```
\CDS_Version "b47"

\begin_outline
"Plotting a Sine Wave"
\end_outline

\begin_node{untitled1}
\title{Plotting a Sine Wave}
\page_constructor_text "(define (get-scale-value scale)
  (/ (get-resource scale 'value)
    (^ 10 (get-resource scale 'decimalPoints))))
(define freq (get-scale-value 'scale_freq))
(define amp (get-scale-value 'scale_amp))"
\page_entry_text "(activate-widget 'button 'Activate)"
\begin_screen

\begin_figure {FormattedText}
\widget_name "untitled1_FormattedText0"
\mapping "True"
borderWidth "0";
label "Use the @b(Amplitude) and @b(Frequency)
sliders to vary the sine wave's parameters, and
click on @b(Plot) to display the new curve.";
fontSize "big";
textMode "fill";
leftPosition "240";
topPosition "0";
rightPosition "800";
bottomPosition "192";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_freq"
\mapping "True"
\action "Drag" "(set! freq (get-scale-value 'scale_freq))"
borderWidth "0";
```

```
value "10";
decimalPoints "1";
maximum "60";
orientation "HORIZONTAL";
processingDirection "MAX_ON_RIGHT";
titleString "Frequency";
leftPosition "224";
topPosition "848";
rightPosition "784";
bottomPosition "976";
\end_figure

\begin_figure {XmPushButton}
\widget_name "button"
\mapping "True"
\action "Activate" "(define (f x)
  (* amp (sin (* freq x))))
(xyplot 'plot_sinewave f)"
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Push Me
to
Plot";
leftPosition "816";
topPosition "192";
rightPosition "960";
bottomPosition "784";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotter"
\mapping "True"
\constructor_text "(add-xyplot 'plotter 'plot_sinewave)"
borderWidth "2";
title "@bigger(@b(Sine Wave))";
xaxis.label "@bigger(@b(@Theta))";
yaxis.label "@bigger[Y = sin(@Theta)]";
xaxis.min "-6.28";
yaxis.min "-5";
xaxis.max "6.28";
yaxis.max "5";
*lineWidth "3";
leftPosition "224";
topPosition "192";
rightPosition "784";
bottomPosition "784";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_amp"
\mapping "True"
\action "Drag" "(set! amp (get-scale-value 'scale_amp))"
```

```
borderWidth "0";
value "10";
decimalPoints "1";
maximum "50";
titleString "Amplitude";
leftPosition "48";
topPosition "192";
rightPosition "208";
bottomPosition "784";
\end_figure
\end_screen
\end_node
```

Appendix B

Application Code

This appendix contains the listings of the two files which comprise the Bernoulli application: **bernoulli.scm**, which contains the CmScheme procedures that are loaded in by the main application; **bernoulli.cds**, which is the actual application code. The latter file is included here primarily as a reference for future developers (showing the overall methodology and organization of the application), as well as for comparison purposes (showing the differences between my and other developers' styles).

B.1 bernoulli.scm

```
;;
;; Define global and state variables
;;

(define space (quote  ))      ;; space character

(define bernoulli-length 2)    ;; length of Bernoulli PMF
(define binomial-length 10)    ;; Binomial PMF
(define geometric-length 23)   ;; Geometric PMF
(define pascal-length 23)     ;; Pascal PMF

(define pmf-type 'bernoulli)   ;; type of PMF being plotted
(define bus-started 'nil)     ;; indicates if bus program is running
(define bus-paused 'nil)      ;; indicates if bus program is paused
```

```
(define prob 0.5)           ;; current value of probability slider
(define N 3)              ;; current order of Binomial experiment
(define R 3)              ;; current order of Pascal experiment

(define exporder 1)        ;; current order of bus program experiment
(define exptype 2)         ;; simulation type (Pascal = 1, Binomial = 2)
(define icon 0)           ;; icon displayed for arrivals (0-3)
(define speed 2)          ;; speed of bus line (1-5)

;;
;; General-use Scheme procedures
;;

;; returns a Bernoulli PMF

(define (bernoulli P)
 (list (- 1 P) P))

;; returns the current value of a slider

(define (get-slider-value slider)
 (/ (get-resource slider 'value)
   (^ 10 (get-resource slider 'decimalPoints))))

;;
;; Procedures that handle the Bernoulli bus simulation program
;;

;; starts the Bernoulli bus simulation, with the current values of
;; experiment type, speed, icon, probability, and order

(define (start-bus)
 (define cmd (quote ren -xy 210 612 -trials -arrivals -events ))
 (set! cmd (concat cmd '-speed space speed space))
 (set! cmd (concat cmd '-icon space icon space))
 (set! cmd (concat cmd '-exporder space exporder space))
 (set! cmd (concat cmd '-exptype space exptype space))
 (set! cmd (concat cmd '-prob space prob))
 (socket-open 'bus cmd)
 (set! bus-started 't))

;; toggles the simulation's pause state

(define (pause-bus)
 (socket-writeln 'bus 'pause)
 (set! bus-paused (not bus-paused)))

;; updates the simulation with current order, experiment type, and
;; probability

(define (update-bus)
```

```
  (socket-writeln 'bus (concat 'exporder exporder))
  (socket-writeln 'bus (concat 'exptype exptype))
  (socket-writeln 'bus (concat 'prob prob)))

;; halts the simulation program

(define (stop-bus)
  (socket-writeln 'bus 'quit)
  (socket-close 'bus)
  (set! bus-paused 'nil)
  (set! bus-started 'nil))

;;
;; Procedures that update a PMF plot (barchart)
;;

;; update Bernoulli barchart

(define (update-ber ch)
  (barchart ch (bernoulli prob)))

;; update Binomial barchart

(define (update-bin ch)
  (barchart ch (binomial prob N binomial-length)))

;; update Geometric barchart

(define (update-geom ch)
  (barchart ch (geometric prob geometric-length)))

;; update Pascal barchart

(define (update-pas ch)
  (barchart ch (pascal prob R pascal-length)))

;;
;; Specialized procedures for updating a specific screen
;;

;; update screen 1, which displays a single PMF and simulation

(define (update-screen1)
  (cond ((eq? pmf-type 'bernoulli) (update-ber 'chartBer1))
        ((eq? pmf-type 'geometric) (update-geom 'chartGeom1))
        ((eq? pmf-type 'binomial)  (update-bin 'chartBin1))
        ((eq? pmf-type 'pascal)    (update-pas 'chartPas1)))
  (if bus-started (update-bus) nil))
```

B.2 bernoulli.cds

```
\CDS_Version "b47"

\begin_outline
"Introduction"
"Definition: Bernoulli Process"
"Simulating The Bernoulli Process"
"Graphing The Bernoulli PMFs"
"How The PMFs Are Related"
"The Equations"
"Acknowledgements"
\end_outline

\begin_node{untitled0}
\title{Introduction}
\page_constructor_text "(eval-file 'bernoulli.scm)"
\begin_screen

\begin_figure {XmLabel}
\widget_name "label_title"
\mapping "True"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "The Bernoulli Process is our friend.";
leftPosition "32";
topPosition "464";
rightPosition "960";
bottomPosition "544";
\end_figure
\end_screen
\end_node

\begin_node{The Bernoulli Process}
\title{Definition: Bernoulli Process}
\begin_screen

\begin_figure {FormattedText}
\widget_name "text1"
\mapping "True"
borderWidth "2";
label "@c(A @b(Bernoulli Random Variable) is used to model)
@c(experiments that result in a value of zero or one.)
@.
@c(@i(Examples:))
@c([success / failure])
@c([heads / tails])
@c([arrival / no arrival])
@.
@c(The @b(Bernoulli Process) is a sequence of)
```

```
@c(independent Bernoulli trials.)
@.
@c(The following screens contain simulations)
@c(of Bernoulli Processes.)";
fontSize "big";
xFlag "left";
leftPosition "0";
topPosition "0";
rightPosition "992";
bottomPosition "992";
\end_figure
\end_screen
\end_node

\begin_node{The Bernoulli PMF}
\title{Graphing The Bernoulli PMFs}
\page_entry_text "(set! N (get-slider-value 'scale_N))
(set! R (get-slider-value 'scale_R))
(set! prob (get-slider-value 'scale_P))
(activate-widget (concat pmf-type '_button) 'Activate)
(hide-widget 'text_warning)
(hide-widget 'label_homer)"
\begin_screen

\begin_figure {FormattedText}
\widget_name "pmf_text"
\mapping "True"
borderWidth "0";
label "@b[A single @i(Bernoulli trial) generates an
experimental value of discrete random variable @i(x),
as described by the following PMF:]";
fontSize "big";
textMode "fill";
leftPosition "32";
topPosition "0";
rightPosition "864";
bottomPosition "96";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotBer"
\mapping "True"
\constructor_text "(add-barchart 'plotBer 'chartBer1 bernoulli-length)
(set-barchart-resource 'chartBer1 'xMin 0.0)
(set-barchart-resource 'chartBer1 'xMax 1.0)
(set-barchart-resource 'chartBer1 'density 0.4)
(set-barchart-resource 'chartBer1 'shading 'true)
(set-barchart-resource 'chartBer1 'pattern 'AtGRAY5)"
borderWidth "2";
title "@bigger(@b(Bernoulli))";
autoScale "False";
```

```
xaxis.label "@bigger(x = successes in one trial)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.99";
xaxis.max "1.99";
xaxis.autoNumber "false";
leftPosition "16";
topPosition "112";
rightPosition "768";
bottomPosition "624";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotBin"
\mapping "True"
\constructor_text "(add-barchart 'plotBin 'chartBin1 binomial-length)
(set-barchart-resource 'chartBin1 'xMin 0.0)
(set-barchart-resource 'chartBin1 'xMax (- binomial-length 1))
(set-barchart-resource 'chartBin1 'shading 'true)
(set-barchart-resource 'chartBin1 'pattern 'AtGRAY5)"
borderWidth "2";
title "@bigger(@b(Binomial))";
autoScale "False";
xaxis.label "@bigger(x = successes in @b(N) trials)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.9";
xaxis.max "10.5";
xaxis.autoNumber "false";
leftPosition "16";
topPosition "112";
rightPosition "768";
bottomPosition "624";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotGeom"
\mapping "True"
\constructor_text "(add-barchart 'plotGeom 'chartGeom1 geometric-length)
(set-barchart-resource 'chartGeom1 'xMin 0.0)
(set-barchart-resource 'chartGeom1 'xMax (- geometric-length 1))
(set-barchart-resource 'chartGeom1 'shading 'true)
(set-barchart-resource 'chartGeom1 'pattern 'AtGRAY5)"
borderWidth "2";
title "@bigger(@b(Geometric))";
autoScale "False";
xaxis.label "@bigger(x = trials until first success)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.5";
xaxis.max "22.5";
xaxis.autoNumber "false";
leftPosition "16";
topPosition "112";
```

```
rightPosition "768";
bottomPosition "624";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotPas"
\mapping "True"
\constructor_text "(add-barchart 'plotPas 'chartPas1 pascal-length)
(set-barchart-resource 'chartPas1 'xMin 0.0)
(set-barchart-resource 'chartPas1 'xMax (- pascal-length 1))
(set-barchart-resource 'chartPas1 'shading 'true)
(set-barchart-resource 'chartPas1 'pattern 'AtGRAY5)"
borderWidth "2";
title "@bigger(@b(Pascal))";
autoScale "False";
xaxis.label "@bigger(x = trials until @b(R@+(th)) success)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.5";
xaxis.max "22.5";
xaxis.autoNumber "false";
leftPosition "16";
topPosition "112";
rightPosition "768";
bottomPosition "624";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_P"
\mapping "True"
\action "Drag" "(set! prob (get-slider-value 'scale_P))
(update-screen1)        "
\action "Value Changed" "(activate-widget 'scale_P 'Drag)"
borderWidth "0";
value "50";
decimalPoints "2";
titleString "P";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "340";
leftPosition "784";
topPosition "112";
rightPosition "880";
bottomPosition "624";
\end_figure

\begin_figure {FormattedText}
\widget_name "FormattedText0"
\mapping "True"
borderWidth "0";
label "Use the above sliders to change @b(P), @b(R), and @b(N).";
fontSize "big";
xFlag "left";
```

```
textMode "fill";
leftPosition "784";
topPosition "624";
rightPosition "976";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "stop_button"
\mapping "True"
\action "Activate" "(hide-widget 'text_warning)
(hide-widget 'label_homer)
(if (not bus-started) nil (stop-bus))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Stop";
leftPosition "16";
topPosition "928";
rightPosition "176";
bottomPosition "992";
\end_figure

\begin_figure {XmPushButton}
\widget_name "pause_button"
\mapping "True"
\action "Activate" "(if (not bus-started) nil
  (if bus-paused nil (pause-bus)))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Pause";
leftPosition "16";
topPosition "800";
rightPosition "176";
bottomPosition "864";
\end_figure

\begin_figure {XmPushButton}
\widget_name "start_button"
\mapping "True"
\action "Activate" "(if bus-started nil (show-widget 'label_homer))
(if bus-started nil (show-widget 'text_warning))
(process-events)
(if bus-started nil (start-bus))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Start";
leftPosition "16";
topPosition "736";
rightPosition "176";
bottomPosition "800";
\end_figure
```

```
\begin_figure {XmPushButton}
\widget_name "resume_button"
\mapping "True"
\action "Activate" "(if (not bus-started) nil
  (if (not bus-paused) nil (pause-bus)))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Resume";
leftPosition "16";
topPosition "864";
rightPosition "176";
bottomPosition "928";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_N"
\mapping "False"
\action "Drag" "(set! N (get-slider-value 'scale_N))
(set! exporder N)
(update-screen1)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "N";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
leftPosition "880";
topPosition "112";
rightPosition "976";
bottomPosition "624";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_R"
\mapping "False"
\action "Drag" "(set! R (get-slider-value 'scale_R))
(set! exporder R)
(update-screen1)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "R";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
leftPosition "880";
topPosition "112";
rightPosition "976";
bottomPosition "624";
\end_figure
```

```
\begin_figure {XmPushButton}
\widget_name "pascal_button"
\mapping "True"
\action "Activate" "(set! pmf-type ’pascal)
(show-widget ’plotPas)
(hide-widget ’scale_N)
(show-widget ’scale_R)
(set-resource ’pmf_text ’label
 (quote @b[The @i(Pascal PMF) describes the number of Bernoulli trials
up to and including the @i(R@+[@v1(th)]) success. (@i(R@+{@v1(th)}) order
interarrival time)]))
(set! exptype 1)
(activate-widget ’scale_R ’Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Pascal";
leftPosition "608";
topPosition "640";
rightPosition "768";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "geometric_button"
\mapping "True"
\action "Activate" "(set! pmf-type ’geometric)
(show-widget ’plotGeom)
(hide-widget ’scale_N)
(hide-widget ’scale_R)
(set-resource ’pmf_text ’label
 (quote @b[The @i(Geometric PMF) describes the number of Bernoulli
trials up to and including the next success.  (First order interarrival
time)]))
(set! exporder 1)
(set! exptype 1)
(activate-widget ’scale_P ’Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Geometric";
leftPosition "400";
topPosition "640";
rightPosition "592";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "binomial_button"
\mapping "True"
\action "Activate" "(set! pmf-type ’binomial)
(show-widget ’plotBin)
(show-widget ’scale_N)
```

```
(hide-widget 'scale_R)
(set-resource 'pmf_text 'label
 (quote @b[The @i(Binomial PMF) describes the sum of a series of @i(N)
independent Bernoulli trials, each with the same probability of
success.]))
(set! exptype 2)
(activate-widget 'scale_N 'Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Binomial";
leftPosition "208";
topPosition "640";
rightPosition "384";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "bernoulli_button"
\mapping "True"
\action "Activate" "(set! pmf-type 'bernoulli)
(show-widget 'plotBer)
(hide-widget 'scale_N)
(hide-widget 'scale_R)
(set-resource 'pmf_text 'label
 (quote @b[A single @i(Bernoulli trial) generates an experimental value
of discrete random variable @i(x), as described by the following PMF:]))
(set! exporder 1)
(set! exptype 2)
(activate-widget 'scale_P 'Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Bernoulli";
leftPosition "16";
topPosition "640";
rightPosition "192";
bottomPosition "720";
\end_figure

\begin_figure {FormattedText}
\widget_name "text_info"
\mapping "True"
borderWidth "0";
label "@b[@c[
@uparrow @#@#@# Use these buttons to select the experiment type.@.
@(@v2(@i(``Ack. Ooop.  Thhffpt!''))))@.
@leftarrow @#@#@# Use these buttons to control the experiment's progress.
]]";
fontSize "big";
xFlag "left";
yFlag "bottom";
textMode "fill";
```

```
leftPosition "208";
topPosition "736";
rightPosition "640";
bottomPosition "992";
\end_figure

\begin_figure {FormattedText}
\widget_name "text_warning"
\mapping "False"
borderWidth "0";
label "@c(``@i(Douughh!!)@.
@b(Please wait for the Bernoulli Simulation to appear.)'')";
fontSize "big";
xFlag "left";
textMode "fill";
leftPosition "400";
topPosition "736";
rightPosition "768";
bottomPosition "992";
\end_figure

\begin_figure {XmLabel}
\widget_name "label_homer"
\mapping "False"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/homer";
labelType "PIXMAP";
leftPosition "208";
topPosition "736";
rightPosition "400";
bottomPosition "992";
\end_figure

\begin_figure {XmLabel}
\widget_name "The Bernoulli PMF_XmLabel0"
\mapping "True"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/bill3";
labelType "PIXMAP";
leftPosition "640";
topPosition "736";
rightPosition "784";
bottomPosition "992";
\end_figure
\end_screen
\end_node

\begin_node{untitled1}
\title{How The PMFs Are Related}
\page_entry_text "(set! prob (get-slider-value 'scale_P))
(set! N (get-slider-value 'scale_R))
```

```
(set! R (get-slider-value 'scale_R))
(update-ber 'chartBer2)
(update-bin 'chartBin2)
(update-geom 'chartGeom2)
(update-pas 'chartPas2)"
\begin_screen

\begin_figure {AtPlotter}
\widget_name "plotBer"
\mapping "True"
\constructor_text "(add-barchart 'plotBer 'chartBer2 bernoulli-length)
(set-barchart-resource 'chartBer2 'xMin 0.0)
(set-barchart-resource 'chartBer2 'xMax 1.0)"
borderWidth "3";
title "@bigger(@b(Bernoulli))";
xaxis.label "@bigger(x = successes in one trial)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.9";
xaxis.max "10.5";
xaxis.autoNumber "false";
leftPosition "0";
topPosition "0";
rightPosition "304";
bottomPosition "496";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_P"
\mapping "True"
\action "Drag" "(set! prob (get-slider-value 'scale_P))
(update-ber 'chartBer2)
(update-bin 'chartBin2)
(update-geom 'chartGeom2)
(update-pas 'chartPas2)"
\action "Value Changed" "(activate-widget 'scale_P 'Drag)"
borderWidth "0";
value "5";
decimalPoints "1";
maximum "10";
titleString "P";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "250";
leftPosition "784";
topPosition "0";
rightPosition "880";
bottomPosition "432";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_R"
\mapping "True"
```

```
\action "Drag" "(set! R (get-slider-value 'scale_R))
(update-pas 'chartPas2)"
\action "Value Changed" "(activate-widget 'scale_R 'Drag)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "R";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "250";
leftPosition "880";
topPosition "544";
rightPosition "976";
bottomPosition "992";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_N"
\mapping "True"
\action "Drag" "(set! N (get-slider-value 'scale_N))
(update-bin 'chartBin2)"
\action "Value Changed" "(activate-widget 'scale_N 'Drag)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "N";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "250";
leftPosition "784";
topPosition "544";
rightPosition "880";
bottomPosition "992";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotPas"
\mapping "True"
\constructor_text "(add-barchart 'plotPas 'chartPas2 pascal-length)
(set-barchart-resource 'chartPas2 'xMin 0.0)
(set-barchart-resource 'chartPas2 'xMax (- pascal-length 1))"
borderWidth "3";
title "@bigger(@b(Pascal))";
xaxis.label "@bigger(x = trials until @b(R@+(th)) success)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.9";
xaxis.max "22.5";
xaxis.autoNumber "false";
leftPosition "304";
topPosition "496";
rightPosition "784";
```

```
bottomPosition "992";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotBin"
\mapping "True"
\constructor_text "(add-barchart 'plotBin 'chartBin2 binomial-length)
(set-barchart-resource 'chartBin2 'xMin 0.0)
(set-barchart-resource 'chartBin2 'xMax (- binomial-length 1))"
borderWidth "3";
title "@bigger(@b(Binomial))";
xaxis.label "@bigger(x = successes in @b(N) trials)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.9";
xaxis.max "10.5";
xaxis.autoNumber "false";
leftPosition "0";
topPosition "496";
rightPosition "304";
bottomPosition "992";
\end_figure

\begin_figure {AtPlotter}
\widget_name "plotGeom"
\mapping "True"
\constructor_text "(add-barchart 'plotGeom 'chartGeom2 geometric-length)
(set-barchart-resource 'chartGeom2 'xMin 0.0)
(set-barchart-resource 'chartGeom2 'xMax (- geometric-length 1))"
borderWidth "3";
title "@bigger(@b(Geometric))";
xaxis.label "@bigger(x = trials until first success)";
yaxis.label "@bigger(P[x])";
xaxis.min "-0.9";
xaxis.max "22.5";
xaxis.autoNumber "false";
leftPosition "304";
topPosition "0";
rightPosition "784";
bottomPosition "496";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled1_FormattedText0"
\mapping "True"
borderWidth "0";
label "Use the sliders to manipulate @b(P), @b(N), and @b(R).";
fontSize "big";
textMode "fill";
leftPosition "800";
topPosition "432";
rightPosition "992";
```

```
bottomPosition "544";
\end_figure
\end_screen
\end_node

\begin_node{untitled1}
\title{Acknowledgements}
\begin_screen

\begin_figure {FormattedText}
\widget_name "credits"
\mapping "True"
borderWidth "0";
label "@c[
@b(CDS) is the result of years of hard work by students,
staff, and faculty.  Members of @b(Project CDS) are:
@b()
Al Drake,
Ed Moriarty,
Dept. of Electrical Engineering & Computer Science
@b()
Jon C. Haass
Dept. of Mathematics
@b()
Joe Wang
Sourabh Niyogi
Marc Schaub
John Piscitello
Jason Beardsley
Jeff Turcotte
Adam Feder
@b()
A special thanks is extended to the @b(UROP)
office, whose funding and support is
greatly appreciated.]";
fontSize "big";
xFlag "left";
yFlag "top";
leftPosition "144";
topPosition "0";
rightPosition "864";
bottomPosition "992";
\end_figure

\begin_figure {XmLabel}
\widget_name "untitled1_XmLabel0"
\mapping "True"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/escherknot";
labelType "PIXMAP";
leftPosition "48";
```

```
topPosition "368";
rightPosition "272";
bottomPosition "752";
\end_figure

\begin_figure {XmLabel}
\widget_name "untitled1_XmLabel0"
\mapping "True"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/escherknot";
labelType "PIXMAP";
leftPosition "720";
topPosition "368";
rightPosition "944";
bottomPosition "752";
\end_figure
\end_screen
\end_node

\begin_node{untitled0}
\title{Simulating The Bernoulli Process}
\page_entry_text "(activate-widget (concat pmf-type '_button) 'Activate)
(hide-widget 'text_warning)
(hide-widget 'label_guide)"
\begin_screen

\begin_figure {XmPushButton}
\widget_name "bernoulli_button"
\mapping "True"
\action "Activate" "(set! pmf-type 'bernoulli)
(hide-widget 'scale_N)
(hide-widget 'scale_R)
(set-resource 'label_type 'labelString 'Bernoulli)
(set! exporder 1)
(set! exptype 2)
(activate-widget 'scale_P 'Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Bernoulli";
leftPosition "16";
topPosition "640";
rightPosition "192";
bottomPosition "720";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_Speed"
\mapping "True"
\action "Drag" "(set! speed (get-slider-value 'scale_Speed))
(if bus-started (socket-writeln 'bus (concat 'speed speed)) nil)"
\action "Value Changed" "(activate-widget 'scale_Speed 'Drag)"
```

```
borderWidth "0";
value "2";
maximum "5";
minimum "1";
titleString "Speed";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "108";
leftPosition "32";
topPosition "112";
rightPosition "160";
bottomPosition "352";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_N"
\mapping "False"
\action "Drag" "(set! exporder (get-slider-value 'scale_N))
(set! N exporder)
(if bus-started (update-bus) nil)"
\action "Value Changed" "(activate-widget 'scale_N 'Drag)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "N";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "110";
leftPosition "880";
topPosition "112";
rightPosition "976";
bottomPosition "624";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_P"
\mapping "True"
\action "Drag" "(set! prob (get-slider-value 'scale_P))
(if bus-started (update-bus) nil)"
\action "Value Changed" "(activate-widget 'scale_P 'Drag)"
borderWidth "0";
value "50";
decimalPoints "2";
titleString "P";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "110";
leftPosition "784";
topPosition "112";
rightPosition "880";
bottomPosition "624";
\end_figure
```

```
\begin_figure {XmScale}
\widget_name "scale_R"
\mapping "False"
\action "Drag" "(set! exporder (get-slider-value ’scale_R))
(set! R exporder)
(if bus-started (update-bus) nil)"
\action "Value Changed" "(activate-widget ’scale_R ’Drag)"
borderWidth "0";
value "3";
maximum "10";
minimum "1";
titleString "R";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "110";
leftPosition "880";
topPosition "112";
rightPosition "976";
bottomPosition "624";
\end_figure

\begin_figure {XmPushButton}
\widget_name "binomial_button"
\mapping "True"
\action "Activate" "(set! pmf-type ’binomial)
(show-widget ’scale_N)
(hide-widget ’scale_R)
(set-resource ’label_type ’labelString ’Binomial)
(set! exptype 2)
(activate-widget ’scale_N ’Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Binomial";
leftPosition "208";
topPosition "640";
rightPosition "384";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "geometric_button"
\mapping "True"
\action "Activate" "(set! pmf-type ’geometric)
(hide-widget ’scale_N)
(hide-widget ’scale_R)
(set-resource ’label_type ’labelString ’Geometric)
(set! exporder 1)
(set! exptype 1)
(activate-widget ’scale_P ’Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Geometric";
```

```
leftPosition "400";
topPosition "640";
rightPosition "592";
bottomPosition "720";
\end_figure

\begin_figure {XmPushButton}
\widget_name "pascal_button"
\mapping "True"
\action "Activate" "(set! pmf-type 'pascal)
(hide-widget 'scale_N)
(show-widget 'scale_R)
(set-resource 'label_type 'labelString 'Pascal)
(set! exptype 1)
(activate-widget 'scale_R 'Drag)"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Pascal";
leftPosition "608";
topPosition "640";
rightPosition "768";
bottomPosition "720";
\end_figure

\begin_figure {XmScale}
\widget_name "scale_Icon"
\mapping "True"
\action "Drag" "(set! icon (get-slider-value 'scale_Icon))
(if bus-started (socket-writeln 'bus (concat 'icon icon)) nil)"
\action "Value Changed" "(activate-widget 'scale_Icon 'Drag)"
borderWidth "0";
maximum "3";
titleString "Icon   ";
fontList "-*-new*schoolbook-bold-r-*-*-18-*-*-*-*-*-*-*";
scaleHeight "108";
leftPosition "32";
topPosition "368";
rightPosition "160";
bottomPosition "624";
\end_figure

\begin_figure {XmLabel}
\widget_name "experiment"
\mapping "True"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Experiment  Selection and Parameter Control";
leftPosition "32";
topPosition "0";
rightPosition "960";
bottomPosition "96";
```

```
\end_figure

\begin_figure {XmPushButton}
\widget_name "untitled0_XmPushButton0"
\mapping "True"
\action "Activate" "(if (not bus-started) nil
  (if bus-paused nil (pause-bus)))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Pause";
leftPosition "16";
topPosition "800";
rightPosition "176";
bottomPosition "864";
\end_figure

\begin_figure {XmPushButton}
\widget_name "untitled0_XmPushButton0"
\mapping "True"
\action "Activate" "(if (not bus-started) nil
  (if (not bus-paused) nil (pause-bus)))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Resume";
leftPosition "16";
topPosition "864";
rightPosition "176";
bottomPosition "928";
\end_figure

\begin_figure {XmPushButton}
\widget_name "untitled0_XmPushButton0"
\mapping "True"
\action "Activate" "(hide-widget 'label_guide)
(hide-widget 'text_warning)
(if (not bus-started) nil (stop-bus))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Stop";
leftPosition "16";
topPosition "928";
rightPosition "176";
bottomPosition "992";
\end_figure

\begin_figure {XmPushButton}
\widget_name "untitled0_XmPushButton0"
\mapping "True"
\action "Activate" "(if bus-started nil (show-widget 'label_guide))
(if bus-started nil (show-widget 'text_warning))
(process-events)
```

```
(if bus-started nil (start-bus))"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Start";
leftPosition "16";
topPosition "736";
rightPosition "176";
bottomPosition "800";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled0_FormattedText0"
\mapping "True"
label "@c[
Use these sliders to change the following:
@b()
@b(P) = probability of success/arrival @#@#@#@rightarrow @rightarrow
@b(R) = order of Pascal experiment
@b(N) = order of Binomial experiment
@b()
@leftarrow @leftarrow @#@#@#@b(Icon) = symbol displayed for arrivals
@b(Speed) = speed of experiment
]";
fontSize "big";
xFlag "left";
yFlag "top";
leftPosition "192";
topPosition "112";
rightPosition "768";
bottomPosition "544";
\end_figure

\begin_figure {XmLabel}
\widget_name "label_type"
\mapping "True"
borderWidth "2";
fontList "*helvetica-*-r-*-*-24-*";
labelString "Geometric";
leftPosition "544";
topPosition "560";
rightPosition "768";
bottomPosition "624";
\end_figure

\begin_figure {XmLabel}
\widget_name "untitled0_XmLabel0"
\mapping "True"
borderWidth "0";
fontList "*bitstream-charter-medium-r-*-33-*-p-*";
labelString "Experiment Type:";
leftPosition "192";
```

```
topPosition "544";
rightPosition "544";
bottomPosition "624";
\end_figure

\begin_figure {XmLabel}
\widget_name "label_guide"
\mapping "False"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/guide2";
labelType "PIXMAP";
leftPosition "208";
topPosition "736";
rightPosition "400";
bottomPosition "992";
\end_figure

\begin_figure {FormattedText}
\widget_name "text_warning"
\mapping "False"
borderWidth "0";
label "@c(``@i(Don't Panic!!)@.
@b(Please wait for the Bernoulli Simulation to appear.)'')";
fontSize "big";
xFlag "left";
textMode "fill";
leftPosition "400";
topPosition "736";
rightPosition "768";
bottomPosition "992";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled0_FormattedText0"
\mapping "True"
borderWidth "0";
label "@c[@b[
@uparrow @#@#@# Use these buttons to
select the experiment type.@.
@(@v2(@i(``All my circuits are operating perfectly.'')))@.
@leftarrow @#@#@# Use these buttons to
control the experiment's progress.
]]";
fontSize "big";
xFlag "left";
yFlag "bottom";
textMode "fill";
leftPosition "208";
topPosition "736";
rightPosition "640";
bottomPosition "992";
```

```
\end_figure

\begin_figure {XmLabel}
\widget_name "untitled0_XmLabel0"
\mapping "True"
borderWidth "0";
labelPixmap "/mit/cds/beta/pixmaps/hal9000";
labelType "PIXMAP";
leftPosition "640";
topPosition "736";
rightPosition "784";
bottomPosition "992";
\end_figure
\end_screen
\end_node

\begin_node{untitled2}
\title{The Equations}
\begin_screen

\begin_figure {FormattedText}
\widget_name "untitled2_FormattedText0"
\mapping "True"
borderWidth "3";
label "@c[
@b(Bernoulli)
@b()
E(@i[x]) @i[= P]
@b()
@i[@g(s)@-(x)@+(@v1(2)) = P(1 - P)]
]";
fontFamily "times";
fontSize "big";
yFlag "top";
leftPosition "0";
topPosition "0";
rightPosition "496";
bottomPosition "496";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled2_FormattedText0"
\mapping "True"
borderWidth "3";
label "@c[
@b(Binomial)
@b()
E(@i[k]) @i[= n]E(@i[x]) @i[= nP]
@b()
@i[@g(s)@-(k)@+(@v1(2)) = n@g(s)@-(x)@+(@v1(2)) = nP(1 - P)]
]";
```

```
fontFamily "times";
fontSize "big";
yFlag "top";
leftPosition "0";
topPosition "496";
rightPosition "496";
bottomPosition "992";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled2_FormattedText0"
\mapping "True"
borderWidth "3";
label "@c[
@b(Geometric)
@b()
E(@i[l@-[@v1[1]]]) @i[= @Over(Num[1] Denom[P])]
@b()
@i[@g(s)@-(l@-(1))@+(@v1(2)) = @Over(Num[1 - P] Denom[P@+(2)])]
]";
fontFamily "times";
fontSize "big";
yFlag "top";
leftPosition "496";
topPosition "0";
rightPosition "992";
bottomPosition "496";
\end_figure

\begin_figure {FormattedText}
\widget_name "untitled2_FormattedText0"
\mapping "True"
borderWidth "3";
label "@c[
@b(Pascal)
@b()
E(@i[l@-[@v1[R]]]) @i[= @Over(Num[R] Denom[P])]
@b()
@i[@g(s)@-(l@-(R))@+(@v1(2)) = @Over(Num[R(1 - P)] Denom[P@+(2)])]
]";
fontFamily "times";
fontSize "big";
yFlag "top";
leftPosition "496";
topPosition "496";
rightPosition "992";
bottomPosition "992";
\end_figure
\end_screen
\end_node
```

Appendix C

System Code

This appendix contains the file **sjab.cc**, which implements the interprocess communication (IPC) facility provided by CDS, as an extension to the CDS modelling environment. This code, when linked into the Model, defines the following five Scheme primitives:

- **(socket-open 'name 'command)**: starts the execution of **command**, and links its standard input and output streams to a pair of file descriptors which are then stored inside the Model. The **name** is used as an identifier in future socket-related calls. [1]

- **(socket-read 'name)**: reads from the external program's standard output until no characters are pending, end of file is reached, or an error occurs, and then returns any string it received.

- **(socket-write 'name 'text)**: writes **text** to the external program's standard input.

- **(socket-writeln 'name 'text)**: same as above, except appends a newline character. (This additional function was necessary because the

[1] The implementation of this function is based on code written for the **popen()** Unix system call, as given in [Kernighan 78].

Model currently has no way of including non-printing control characters in a string; once it can, this primitive will be unnecessary.)

- **(socket-close 'name)**: closes the connection, and waits for the external program to finish execution.

The code is written in C++, but is highly dependent on the Unix operating system for its method of implementing IPC. Thus, it may resemble C more than C++.

sjab.cc

```
/* sjab.cc : CmScheme primitives written by Jason Beardsley */

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <CDS.h>
#include <CmCDS.h>
#include <CmScheme.h>
#include <CmWidget.h>
#include <CmMenu.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>

// define this, because socket.h doesn't (why?)
extern "C" {
int socketpair(int d, int type, int protocol, int sv[2]);
}

char* handle_socket_open(char** arg, CmScheme* s)
    // arg[0] = socket name
    // arg[1] = command passed to sh
{
  // sock1[1] is to write to child, sock2[1] is to read from child
  int sock1[2], sock2[2], child;

  // open socket pairs
  if (socketpair(AF_UNIX, SOCK_STREAM, 0, sock1) < 0) {
    s->Error("Unable to open sockets.");
    return dup(NIL_STR);
  }
```

```
  if (socketpair(AF_UNIX, SOCK_STREAM, 0, sock2) < 0) {
    s->Error("Unable to open sockets.");
    return dup(NIL_STR);
  }
  // fork
  if ((child = fork()) == -1) {
    s->Error("Unable to start child process.");
    return dup(NIL_STR);
  }
  // this is the parent process (i.e. CDS)
  else if (child > 0) {
    // close child sockets
    close(sock1[0]);
    close(sock2[0]);
    // save real sockets' numbers (write, read) and process ID
    // in CmScheme special environment
    char* evalstr = new char[128];
    sprintf(evalstr, "%d %d %d", sock1[1], sock2[1], child);
    s->GetSpecialEnvironment()->Define(arg[0],evalstr);
    delete evalstr;
    // open write socket (as file) and set the buffer length to zero.
    // this makes the line unbuffered, in theory
    FILE *outsock = fdopen(sock1[1], "w");
    setbuf(outsock, (char*)NULL);
    return dup(OK_STR);
  }
  // this is the child process
  else {
    // close parent sockets
    close(sock1[1]);
    close(sock2[1]);
    dup2(sock1[0],0);   // make sock1[0] the child's stdin
    dup2(sock2[0],1);   // make sock2[0] the child's stdout
    execl("/bin/sh", "sh", "-c", arg[1], 0);
    _exit(1);           // this should never be reached
  }
}

char* handle_socket_write(char** arg, CmScheme* s)
    // arg[0] = name
    // arg[1] = string to write
    // returns: ok, or nil (error)
{
  int sockets[2], child;
  char* pstr = s->GetSpecialEnvironment()->Value(arg[0]);
  if ((pstr == NULL) || (pstr == NIL_STR)) {
    s->Error("Unknown socket identifier: ", arg[0]);
    return dup(NIL_STR);
  }
  sscanf(pstr, "%d %d %d", sockets, sockets+1, &child); // get values
  // remember: write socket, read socket
```

```
  if (write(sockets[0], arg[1], strlen(arg[1])) < 0) {
    return dup(NIL_STR);
  }
  else {
    return dup(OK_STR);
  }
}

char* handle_socket_writeln(char** arg, CmScheme* s)
    // arg[0] = name
    // arg[1] = string to write (appends a newline)
    // returns: ok, or nil (error)
{
  int sockets[2], child;
  char* pstr = s->GetSpecialEnvironment()->Value(arg[0]);
  if ((pstr == NULL) || (pstr == NIL_STR)) {
    s->Error("Unknown socket identifier: ", arg[0]);
    return dup(NIL_STR);
  }
  sscanf(pstr, "%d %d %d", sockets, sockets+1, &child); // get values
  // remember: write socket, read socket
  CmString outstr(arg[1]);
  outstr += "\n";
  if (write(sockets[0], outstr.Chars(), strlen(arg[1])+1) < 0) {
    return dup(NIL_STR);
  }
  else {
    return dup(OK_STR);
  }
}

char* handle_socket_read(char** arg, CmScheme* s)
    // arg[0] = name
    // returns a string
{
  int sockets[2], child, nfound;
  char *buf = new char[1024]; // 1k buffer
  CmString rtnstr;
  char* pstr = s->GetSpecialEnvironment()->Value(arg[0]);
  if ((pstr == NULL) || (pstr == NIL_STR)) {
    s->Error("Unknown socket identifier: ", arg[0]);
    return dup(NIL_STR);
  }
  sscanf(pstr, "%d %d %d", sockets, sockets+1, &child); // get values
  // remember: write socket, read socket

  // poll socket for any available data
  struct timeval tv = {0, 0}; // seconds, useconds
  nfound = select(1, &sockets[1], 0, 0, &tv);
  if (nfound == -1 || nfound == 0) { // error or time expired
    return dup(NIL_STR);
```

```
  }
  else {
    // EOF is still a possibility
    if (read(sockets[1], buf, sizeof(buf)) == 0) { // end-of-file
      return dup("EOF");
    }
    else {
      // read in data (into CmString rtnstr) until EOF
      while (read(sockets[1], buf, sizeof(buf)) != 0) {
        rtnstr += buf;
      }
      // return a char*
      return dup(rtnstr.Chars());
    }
  }
}

char* handle_socket_close(char** arg, CmScheme* s)
    // arg[0] = name
    // returns ok, or nil (error)
{
  int sockets[2], child, status;
  char* pstr = s->GetSpecialEnvironment()->Value(arg[0]);
  if ((pstr == NULL) || (pstr == NIL_STR)) {
    s->Error("Unknown socket identifier: ", arg[0]);
    return dup(NIL_STR);
  }
  sscanf(pstr, "%d %d %d", sockets, sockets+1, &child); // get values
  close(sockets[0]); // close sockets up
  close(sockets[1]);
  // hope that the process ends...
  wait(&status);
  // clear socket name
  s->GetSpecialEnvironment()->Set(arg[0],NIL_STR);
  return dup(OK_STR);
}

void sjab(CmScheme* s)
{
  s->AddMethod("socket-open", handle_socket_open, 2);
  s->AddMethod("socket-write", handle_socket_write, 2);
  s->AddMethod("socket-writeln", handle_socket_writeln, 2);
  s->AddMethod("socket-read", handle_socket_read, 1);
  s->AddMethod("socket-close", handle_socket_close, 1);
}
```

Bibliography

[Baclawski 89] Baclawski, Kenneth, Gian-Carlo Rota, and Sara Billey. *An Introduction to the Theory of Probability*. Second Preliminary Edition, MIT, 1989.

[CDS Programmer] *Courseware Development System: Programmers Guide*. Project CDS, MIT, 1991.

[CDS User] *Courseware Development System: Users Manual*. Project CDS, MIT, 1991.

[Drake 67] Drake, Alvin W. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967.

[Giang 89] Giang, Linh Kim. *The Lecture Authoring System: Text Pidget and Text Output Utility*. Bachelor's thesis, May 1989.

[Kernighan 78] Kernighan, Brian W. and Dennis M. Ritchie. *UNIX Programming – Second Edition*. Bell Laboratories, New Jersey, November 12, 1978.

[LAS 89] *Lecture Authoring System: Program Documentation*. Project LAS, MIT, 1989.

[Lamport 86] Lamport, Leslie. LATEX*: A Documentation Preparation System*. Addison-Wesley, Reading, Massachusetts, 1986.

[LaPlante 88] LaPlante, Ryan A. *The Lecture Authoring System and Lecture Variables*. Bachelor's thesis, May 1988.

[Miller 90] Miller, Eric L. *Lecture Authoring System: Editor and Interpreter*. Bachelor's thesis, May 1990.

[Moriarty 89] Moriarty, Edward J. “Lecture Authoring System”, *Windows On Athena*, Volume 1, Chapter 9. Project Athena, MIT, August 1989.

[Moriarty 91] Moriarty, Edward J. “Courseware Development System”, *Windows On Athena*, Volume 2. Project Athena, MIT, 1991.

[Nunez 91] Nunez, Robert J. *Courseware Development System: A Newton's Method Application*. Bachelor's thesis, May 1991.

[Piscitello 91] Piscitello, John P. *Courseware Development System: Adding Independent Random Variables and Convolution*. Bachelor's thesis, May 1991.

[Stroustrup 90] Stroustrup, Bjarne and Margaret A. Ellis. *The Annotated C++ Referenece Manual*. Addison-Wesley, New York, 1990.

[Turcotte 91] Turcotte, Jeffrey R. *The Courseware Development System (CDS): A Queuing Theory Application*. Bachelor's thesis, May 1991.

[Wang 91] Wang, Joseph. *Information Organization in the Courseware Development System*. Bachelor's thesis, May 1991.

[Young 90] Young, Douglas A. *The* X *Window System: Programming and Applications with* Xt*, OSF/Motif Edition*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[Zeitler 89] Zeitler, Patricia A. *Lecture Authoring System: Graphics for Simulations of Renewal Processes*. Bachelor's thesis, May 1989.