A LEARNING SUPPORT SYSTEM
FOR THE
VISUAL SIMULATION ENVIRONMENT

by
Rajen Harrichunder

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

Master of Information Systems
in
Computer Science

APPROVED:

Osman Balci
Osman Balci, Chairman
Department of Computer Science

Edward A. Fox
Department of Computer Science

Richard E. Nance
Department of Computer Science

September, 1994
Blacksburg, Virginia
A LEARNING SUPPORT SYSTEM
FOR THE VISUAL SIMULATION ENVIRONMENT

by
Rajen Harrichunder
Osman Balci, Chairman
Computer Science

(ABSTRACT)

The objective of this research project is to develop a computer assisted education system to enable users to effectively learn the Visual Simulation Environment (VSE), simulation and modeling. The vast amount and the relatively high complexity of the knowledge contained in the VSE, simulation, and modeling, dictate a need to provide an effective mechanism for assimilating the knowledge. This report describes the Learning Support System (LSS), a tool developed to present documents to users in an effective manner, thereby facilitating access to the knowledge contained within the documents. Text, graphics and animations are used within documents to support the learning process. The hierarchical organization of the information structure, the use of hypertext links, and search capabilities facilitate the efficient transfer of relevant information. The LSS thus incorporates features that contribute to an improved learning process. Usability tests across a wider spectrum of users would enable the LSS to be modified to meet the needs of the diverse user community. The LSS will be used in the Spring ’95 Simulation classes to aid in the teaching of simulation and modeling. Feedback from users will shape the future enhancements to LSS. Future technology changes are expected to provide advantages for building on the LSS research project described herein.
ACKNOWLEDGEMENTS

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

I wish to thank Chuck Esterbrook and Anders Bertelrud for the times, unknown to them, when I was steered on the correct path by way of their simple answers.

Finally, my deepest feelings go out to the following individuals who shape my life for the better:

My mother, sister and late father;
My friends: Sarith, Sudesh, Vikash, Patrick, Rishen, Yogen and Suntrasen;
And our President Nelson Mandela.

You are all embodied as HOPE.
# TABLE OF CONTENTS

Abstract ii
Acknowledgements iii
List of Figures vi
List of Tables viii

Chapter 1: INTRODUCTION 1

1.1 The SMDE Research Project 1
1.1.1 The SMDE Architecture 2
1.1.2 Minimal SMDE Tools 2
1.2 Problem Statement 4

Chapter 2: LITERATURE SURVEY 6

2.1 Instructional Systems 6
2.2 Help Systems 7
2.2.1 On-Line Help for the NEXT Environment 9
2.3 Context of the LSS in the VSE 9
2.4 Examples of Tutorial Systems 11
2.4.1 The Microsoft Windows Tutorial 11
2.4.2 The Macintosh Basics Tutorial 12

Chapter 3: DESIGN AND IMPLEMENTATION 15

3.1 Overview of NeXTSTEP 15
3.1.1 The NeXT User Interface Design Principles 17
3.2 Conceptual Design 18
3.3 Translation of the Conceptual Model to the NEXTSTEP Environment 20
3.4 Development Strategy 22
3.5 Implementation Strategy 24
3.5.1 NeXTSTEP Program Development Cycle 24
3.5.2 Interface Builder 26
3.5.3 Project Builder 26
3.6 Why not Help Builder and NXHelpPanel? 28
3.7 The Application Kit and Implemented Subclasses 29
3.7.1 BrowserControl Class 29
3.7.2 LessonControl Class 32
3.7.3 SearchControl Class 33
3.8 Implementation of Hypertext Links and Graphics Incorporation 34
3.9 The LifeCycleDiagram Class 36
3.10 The Print Feature 37
3.11 The Icon Design 38
Chapter 4: USER'S GUIDE

4.1 Developer Mode
   4.1.1 The Developer Mode Menu
   4.1.2 Creating a Document
   4.1.3 Linking Documents into the Browser

4.2 User Mode
   4.2.1 Selecting a Document
   4.2.2 Using Hypertext Links
   4.2.3 Activating an Animation
   4.2.4 Using a Graphic that Contains Links
   4.2.5 Searching
   4.2.6 Printing

Chapter 5: CONCLUSIONS AND RECOMMENDATIONS
FOR FUTURE RESEARCH

5.1 Conclusions
5.2 Recommendations for Future Research

Bibliography

Appendix A: USING THE EDIT AND WATCHME APPLICATIONS

A.1 Creating a Document with Edit
   A.1.1 Creating a Hypertext Link
   A.1.2 Inserting a Marker

A.2 Creating an Animation with WatchMe

Appendix B: THE LSS SOFTWARE STRUCTURE


### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Three Major User Dimensions that Help Systems Should Address</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Typical NEXTSTEP Help Window</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Microsoft Windows Tutorial</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Microsoft Windows Tutorial</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Macintosh Basics Tutorial</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Macintosh Basics Tutorial</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>NEXTSTEP System</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Conceptual Design</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Translation of the Conceptual Model to the NEXTSTEP Environment</td>
<td>21</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Evolutionary Prototyping</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>NEXTSTEP Program Development Cycle</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>AppKit Class Hierarchy with LSS Subclasses</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>&quot;RTF&quot; File Created By Edit Application</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>LSS Application Icon</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Developer Mode Menu</td>
<td>40</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Browser in Development Mode</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Browser Entry Link Panel</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Open Panel for Selecting File</td>
<td>42</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Typical Session In LSS</td>
<td>44</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>Tape Image For Animation</td>
<td>45</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Simulation Life Cycle Diagram</td>
<td>46</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Main LSS Menu with Find Selected</td>
<td>47</td>
</tr>
<tr>
<td>Figure 4.9</td>
<td>The &quot;Find - Current Document&quot; Panel</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.10</td>
<td>The &quot;Find - All Documents&quot; Panel</td>
<td>49</td>
</tr>
<tr>
<td>Figure 4.11</td>
<td>The Print Panel</td>
<td>50</td>
</tr>
<tr>
<td>Figure A.1</td>
<td>Associatiag Link Information to a Link Icon (anchor)</td>
<td>58</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>Figure A.2</td>
<td>Attaching a Marker to the Linked Document</td>
<td>59</td>
</tr>
<tr>
<td>Figure A.3</td>
<td>WatchMe Recorder Panel</td>
<td>60</td>
</tr>
<tr>
<td>Figure B.1</td>
<td>The LSS Software Structure</td>
<td>62</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Description of Implemented Classes for the LSS Application</td>
<td>31</td>
</tr>
<tr>
<td>B.1</td>
<td>Interface and Class Files in the LSS Project</td>
<td>61</td>
</tr>
<tr>
<td>B.2</td>
<td>Description of User Mode Menu Target Methods</td>
<td>66</td>
</tr>
<tr>
<td>B.3</td>
<td>Description of Developer Mode Menu Target Methods</td>
<td>67</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The unprecedented growth of interactive computing applications has led to a heightened interest in the ease of use of computers. This has placed a demand on applications to provide effective assistance, through a minimum amount of training and knowledge, to enable the user to perform the required task [Bergman and Keen-Moore 1985]. To meet this demand it has become common practice to incorporate an extensive on-line assistance system. On-line help context sensitive to tasks and objects, on-line documentation, conversational error and warning messages and tutorial systems all form the on-line assistance provided to the user.

The Learning Support System (LSS) is designed as a tutorial and on-line hypertext document referral system for the Visual Simulation Environment (VSE) and visual simulation. It supplements the context sensitive object/task oriented help provided as attachments to interface objects. The graphics and demonstrations incorporated into the body of text that form a document appeal to novice users, yet it does not hinder the advanced user. The hierarchical organization of the information effectively transfers a vast amount of knowledge in a piecemeal method.

1.1 The SMDE Research Project

The ever increasing complexity of visual simulation model development is undeniable. A simulation programming language supports only the programming process. There is a need for automated support throughout the entire visual simulation model development life cycle. The Visual Simulation Environment (VSE) fulfils this need through a set of integrated tools that provide computer aided assistance in the
development and execution of a visual simulation model. [Derrick and Balci 1992]

The VSE has been in development as part of the Simulation Model Development Environment (SMDE) research project. Vast experience was accrued in the SMDE project to first create the Visual Simulation Support Environment (VSSE), and presently the Visual Simulation Environment (VSE). The VSE currently uses the NEXTSTEP operating environment running on NeXT workstations and Intel based personal computers.

1.1.1 The SMDE Architecture

The SMDE architecture consists of four layers: The hardware and operating system reside at layer 0. Layer 1, the Kernel Simulation Model Development Environment (KSMDE), integrates the SMDE tools into the environment. KSMDE provides databases, communication and run-time support functions, and a kernel interface. Layer 2, the Minimal Simulation Model Development Environment (MSMDE), provides a comprehensive toolset considered minimal for the development and execution of a model. Layer 3, the Simulation Model Development Environments, is the highest layer of the environment. It incorporates tools that support specific applications and are of special interest within a particular project.[Derrick and Balci 1992]

1.1.2 Minimal SMDE Tools

The tools in the MSMDE are comprehensive in that the toolset is supportive of all model development phases, processes, and credibility assessment stages. The toolset is minimal in that it is basic and general. It is basic in the sense that the set of tools enables modelers to work within the bounds of the minimal SMDE without significant inconvenience. Generality is claimed in the sense that the toolset is generically applicable
to various simulation modeling tasks. A brief description of the tools of the MSMDE follows [Derrick and Balci 1992]:

The Project Manager administers the project database, records the progress of the project, triggers messages, and responds to queries concerning project status.

The Premodels Manager enables the user to locate and reuse components of previous simulation studies, and facilitates learning from past experience.

The Model Generator assists the modeler in model specification, documentation, and model qualification.

The Model Analyzer diagnoses the model specification created by the model generator and assists in the verification of the communicative model.

The Model Translator translates the model specification into executable code after the quality of the specification is assured by the Model Analyzer.

The Model Verifier assists in verification of the programmed model by creating a cross-reference map, supplying dynamic analysis tools such as traces, breaks, and system snapshots, and incorporating diagnostic measures within the source program.

The Assistance Manager provides effective and efficient transfer of assistance information to the user. It administers the assistance database, provides tutorials, gives definitions of technical terms, and explains the usage of tool usage.

The various tools and extensiveness of features of the VSE amplifies a need for an effective assistance system that is responsive to user’s errors and assistance requests. The Assistance Manager of the SMDE has been replaced in the VSE through embedded help features and the Learning Support System (LSS). Error and warning messages are provided through alert panels to guide the user in his actions. On-line help is incorporated for each tool and it is context sensitive as well as being consistent with the help system provided by the NEXTSTEP environment. The LSS supplements this help facility by providing a means to organize and present more information. The LSS includes tutorials and on-line documentation relevant to visual simulation and the VSE. The research
described herein is concerned with the design and implementation of the LSS.

1.2 Problem Statement

The VSE is a large software environment containing many tools and features. The vastness of knowledge contained in simulation and modeling, and on the use of the VSE requires that the VSE should have a good assistance system to aid users. On-line help systems including context-sensitive help, prompting, warning and error message systems as well as a learning support system are required to aid the user. This project is concerned with the development of a learning support system to provide users a means of viewing documents relevant to learn the VSE, visual simulation, and modeling.

The learning support system must provide a means of structurally presenting the vastness of knowledge required to explain the VSE and simulation. The list of topics include:

1. Distributed Simulation
2. Experimental Design
3. Input Data Modeling
4. Interactive Visual Simulation Modeling
5. Model Development
6. Model Validation, Verification, and Testing
7. Parallel Simulation
8. Random Number Generator
9. Random Variate Generator
10. Statistical Analysis of Simulation Output Data
11. Visualization/Animation

The documents must be linked so that within the context of the user's knowledge, the user has the control to request further information. Due to the complexity and vastness of
the subject matter, the LSS must provide a well human-engineered interface to communicate information to users. Its features should be minimum and easy to understand to enable users to quickly learn the task at hand.

The VSE is a general purpose environment and hence users with varying backgrounds and expertise would use the system [Frankel and Balci 1989]. Help features are embedded in the tools to provide context sensitive help to users. This assistance is minimal and aids the advanced user. The LSS must address the needs of the novice users without being too much of a hindrance to the advanced user who may require specific knowledge. The information in LSS should contain sufficient detail to aid the novice user. It should be provided in a piecemeal method to facilitate effective learning to the novice user, as well as through this organization, meet the need of the experienced user who can easily reference the specific information that is required.

An LSS would be ineffective if too much effort is required by the author to incorporate documents into an organized structure and if it is complex and time consuming to create documents on a computerized environment. Hence the LSS must be able to accept documents with graphics and text, and provide mechanisms to incorporate the documents into the organized information structure.
Chapter 2
LITERATURE SURVEY

The literature survey was conducted to determine the context of LSS within the VSE environment. Since the primary function of LSS is to provide aid to the user, research was done on Computer Assisted Instruction (CAI) and Help systems to determine the context and ideas for the design of LSS.

2.1 Instructional Systems

Instructional software products, from an historical perspective, were designed to teach topics such as mathematics, science or language. These software products are referred to by a variety of names including [Eberts and Brock 1988]:

- CAI  --  Computer assisted instruction
- CBE  --  Computer based education
- CAL  --  Computer assisted learning
- IAC  --  Instructional application of computers
- CBI  --  Computer based instruction

In order for instruction to be effective it is necessary that the following four phases be present [Alessi and Trollip 1985]:

1. Presenting information
2. Guiding the student
3. Practicing
4. Assessing student learning

These phases are based on research in classroom learning. To teach something new, the instructor must first present information. This may take the form of verbal or pictorial
information which may include a presentation of rules and principles, and pictures.
Guiding the student is the interactive part of the instruction. Having observed the presentation, the student would be required to do something, with guidance from the instructor. The student may answer questions about factual information, may apply rules and principles in problem solving activities, or practice procedural skills. The instructor observes the student and corrects any errors. The instructional process is not complete when the student can do something once. The third phase, practice, is student centered. Here practice with the newly acquired knowledge is required to build fluency and speed under varying conditions. This enables better retention of knowledge. Learning should be assessed through tests to ascertain the success of the instruction. [Alessi and Trollip 1985]

Tutorials, drills, simulations, games and tests are the five major types of computer based instruction programs. Tutorials are programs that engage in the first two phases of instruction. They take the role of the instructor or book by presenting the information and guiding the learner in initial acquisition. Drills and games typically engage in the third phase enabling the student to practice for fluency and retention. Tests represent the last phase. Simulations may be used to provide any combination of the four phases of instruction in the same lesson. It is rare for a single lesson to provide all four phases of instruction. Hence instructional programs must be used in conjunction with other programs or media to provide complete and successful instruction. [Hannafin and Peck 1988; Alessi and Trollip 1985]

2.2 Help Systems
A help system is one or more programs designed to provide user assistance embedded in a larger program or computer system [Kearsley 1988]. Help systems consists of two fundamental aspects; the interface and the content. The interface includes how the help information is displayed, how the user accesses the help features, and how
the system can be extended. The content is the information that is provided to assist the user. Both of these aspects, the interface and the content are equally important for the success of the help system. Help features contribute to the overall usability of software by reducing the time it takes to complete an interaction and the numbers of errors made.

![Three Major User Dimensions that Help Systems Should Address](image)

**Figure 2.1 Three Major User Dimensions that Help Systems Should Address [Kearsley 1988]**

It is difficult to develop a help system that works flawlessly for all users. Users differ in terms of computer background or task experience, their assumptions and expectations about the way a program works, and personality traits such as reading speed, attention span, learning style or patience. Fig. 2.1 illustrates the three major experience dimensions that users may differ on: task, computer, and program. User experience is widely variant with these experience dimensions. Users would include the task expert who is thoroughly familiar with the job but has never used a computer or the program before; the computer whiz who is an expert computer user but does not know the task and has never used this specific program; and the "technician" who knows how to use the program but knows little about the task or computers. Each of these types of users would need slightly different types of help. [Kearsley 1988]
2.2.1 On-Line Help for the NeXT Environment

NeXTSTEP has a comprehensive capability for providing on-line help. This takes the form of a book format with a table of contents and index. It supports the mechanism for processing hypertext links that are created in the Edit application. These links allow for documents to be related as the help designer sees beneficial to providing help within the context of the user's needs and the content information. The on-line help is accessible through a menu option under the "Info" menu item. This would provide the Help panel, shown in Figure 2.2, which the user can use to browse through the information using the contents, index and hypertext links as navigation methods to delve the information. The capability for context-sensitive help is provided by attaching a particular help file to an object. When the "Control-Alternate" keys are clicked on an object then the context sensitive help file would be loaded. Extensive support is provided to enable the building of the on-line help. The Edit application can be used to create the hypertext links. The Help Builder is used to attach help files to interface objects. [Ngheim 1993]

2.3 Context of the LSS in the VSE

The primary purpose of the LSS is to present information in a structured and meaningful manner. The information includes topics on simulation, modeling, and on the use of the VSE environment. Hence the LSS is a tutorial system with the added capability of having the information organized in a "library" to convey the vast amount of knowledge contained within the topics of simulation, modeling, and the VSE.

It is a current trend for software developers to provide a tutorial system, together with their software, to aid the user on the use of that software. The Microsoft Windows Tutorial and Macintosh Basics are typical tutorial systems provided to enhance the aid
given to users on Microsoft Windows and the Macintosh respectively. Typically these tutorial systems are aimed specifically to aid the novice user to get started as quickly as possible by providing some basic knowledge. The LSS possesses the same objective of aiding novice users but adds the additional capability of providing support to experienced users who may be lacking experience in a specific area of simulation or the VSE tool use.

Figure 2.2 Typical NEXTSTEP Help Window
The hierarchical organization of the information guides the novice user further into the knowledge structure as learning progresses whilst it allows the experienced user to directly search for a topic or browse through the hierarchy to get the relevant information.

2.4 Examples of Tutorial Systems

2.4.1 The Microsoft Windows Tutorial

The Microsoft Windows tutorial is aimed at first time users to the Windows environment. Figures 2.3 and 2.4 illustrate a typical session with the Microsoft Windows tutorial.

The tutorial exercises control over the sequence of selections that the user can choose. An information structure, like a table of contents, is not present. Instead a few selections are allowed which takes the user further into the tutorial as each selection is made. The tutorial is task oriented. It allows the user to test his ability by trying the same action that was shown previously thus giving feedback on the learning activity. However due to the basic nature of the information, it is appropriate for first time users to the computing environment. Since the NEXTSTEP environment is based on the UNIX operating system, an LSS user can run the VSE application at the same time as the LSS application. Hence actions demonstrated on the LSS can be implemented immediately on the VSE application. This forms a means of current feedback as in the Windows tutorial. An LSS user can choose to read the textual information or the animation which explains the same information.
This Tutorial has two lessons.

- If you want to learn how to use the mouse, or if you need to brush up on your mouse skills, type M to begin the Mouse lesson.
- If you are already a skilled mouse user, type W to begin the Windows Basics lesson.

Or, if you want to run the Tutorial at another time:

- Press the ESC key to exit the Tutorial.

Figure 2.3 Microsoft Windows Tutorial

2.4.2 The Macintosh Basics Tutorial

Macintosh Basics is provided on the Macintosh computers to allow first time users to get familiar with the look and feel of the operating environment [See Figures 2.5 and 2.6]. The instruction is conversational and basic. It serves the purposes of "one time use" in that it is like an introduction and would not be beneficial to experienced users of computers, even if they are new users to the Macintosh machine. The tutorials contain graphic drawings of the concepts that are being relayed. Objects like the keyboard and the mouse are displayed. This is in line with the target audience: first time users of
The Windows Basics lesson teaches you the skills you need to begin using Windows.

- Click instructions to see how to move around in the lesson.

  **Instructions**

- Click Continue to begin the lesson.

  **Continue**

Figure 2.4 Microsoft Windows Tutorial

computers. The tutorial session does contain a list of topics and thus alleviates the direct instructional control as seen in the Windows tutorial. The tutorial is task oriented allowing the user to learn a particular task as well as knowledge oriented by providing information on the features of the Macintosh interface system.
Creating a Document

You can use your Macintosh to create different kinds of documents: letters, memos, drawings, or spreadsheets, for example. In this section, you’ll learn how to create a memo.

Click the right arrow to move forward.

Figure 2.5 Macintosh Basics Tutorial

Main Topics

☐ 1. Getting Around in Macintosh Basics
☐ 2. The Desktop
☐ 3. Creating a Document
☐ 4. Working With More Than One Program
☐ 5. Filing Your Work
☐ 6. More About the Desktop
☐ 7. What’s Next?

◆ Repeat Mouse Skills section
◆ Quit Macintosh Basics

Now you’re back at the Main Topics menu.
A filled (■) box indicates you’ve completed a section.
A grayed (■) box indicates you’ve started a section.
Choose the section you want by clicking its title. When you want to stop working, click Quit Macintosh Basics.

Figure 2.6 Macintosh Basics Tutorial
Chapter 3
DESIGN AND IMPLEMENTATION

This chapter presents an overview of the development of the LSS application. Since the design and implementation of this software and the NEXTSTEP programming environment are interrelated, this chapter also introduces NEXTSTEP concepts and tools relevant in the design and implementation of the LSS. First, an overview of the NEXTSTEP environment is presented. Second, a conceptual design of the LSS is presented. Third, evolutionary prototyping as a development strategy for the LSS is explained. Fourth, the implementation strategy describes the building of the LSS software within the NEXTSTEP programming environment. The subsequent sections then discuss specific implementation details including: reasons for not using the Help Builder tool and the NXHelpPanel class, the Application Kit and the implemented subclasses for the LSS, the implementation of the hypertext, graphics and print features and the creation of the application icon.

3.1 Overview of NEXTSTEP

There are four levels of software between a NEXT application program and the hardware that it runs on. These levels as illustrated in Figure 3.1 are [NEXT 1990]:

- the NEXT Interface Builder
- Objective-C software "kits"
- the NEXT Window Server and specialized C libraries
- the Mach operating system
Figure 3.1 NEXTSTEP System [NEXT 1990]

Interface Builder is a powerful tool that lets the developer graphically design the application’s user interface. It further facilitates the establishment of connections between user interface objects and the code written.

The NEXT library of classes are written in Objective-C. Objective-C is an extension to C that adds object-oriented concepts to the language. The software kits define a number of classes, or object templates, that can be used by the developer in the applications. These kits include [NEXT 1990]:

- an Application Kit that every application uses to implement the NEXT window based user interface
- the Sound Kit for adding sounds to the application
- a Music Kit for music composition, synthesis, and performance

The NEXT Window Server is a low-level background process used by the Application Kit to manage windows and to send user events back to an application. The
Display PostScript interpreter is used for all drawing of the text and graphics on the screen or printed page.

Mach is the multitasking UNIX operating system developed at Carnegie Mellon University. It acts as an interface between the upper levels of the software and the hardware platform. [NEXT 1990]

3.1.1 The NEXT User Interface Design Principles

Four important user interface principles are [NEXT 1990]:

- the interface should be consistent across all applications
- the user is in charge of the workspace and its windows
- the interface should feel natural to the user
- the mouse, rather than the keyboard, is the primary instrument for user interaction with the interface.

When all applications have the same basic user interface, every application benefits. The consistency makes each application easier to learn, thus increasing the likelihood of its acceptance and use. Although different applications are designed to accomplish different tasks, they all share, to one degree or another, a set of common operations like selecting, editing, scrolling, setting options, making choices from a menu, managing windows and so on. Reliable conventions thus allow users to carry knowledge of how one application works on to another. [NEXT 1990]

With the user in charge of the event flow, s/he is free to choose which application and which window to work in, and to arrange windows in the workspace to suit his/her own needs and tastes. When working with an application the user should be afforded the widest possible freedom of action. It is inappropriate for an application to arbitrarily restrict what the user can do. If an action makes sense it should be allowed. [NEXT 1990]

The great advantage of a graphical user interface is that it can feel natural to the
user. The user interface can be said to be "intuitive" if it behaves as users expect it. The similarity of graphical to real objects is fundamental. User interface objects should be easily recognizable by the users. [NEXT 1990]

All graphical objects are manipulated mainly by the mouse. The keyboard should be principally used for entering text. Keyboard alternatives to certain mouse actions should be provided. These can be efficient shortcuts for experienced users. [NEXT 1990]

3.2 Conceptual Design

Figure 3.2 shows the architectural layout of the LSS design. The design was created through an investigation of the problem statement (Chapter 1).

It was decided to adopt a hierarchical representation to communicate the organization of the information. Hierarchical structures are flexible and facilitate a natural tendency for users to identify with [Conklin 1987]. Information can be arranged hierarchically: by process or objects, in a book format, or as a sequence that best expresses the information. Each node in the hierarchical information space represents the content information. This is shown as the "is-a" link in Figure 3.2. The nodes are related to each other by organizational links. Organizational links connect a parent node with its children and thus form a tree structure [Conklin 1987].

The document contains text, graphics, and animations to communicate the required information effectively to the user. Within the context of this information the user has the capability to request further information via referential links. A referential link connects points or regions in the text [Conklin 1987]. Referential links for the Document in the conceptual model (Figure 3.2) have two ends: an origination and a destination. The origination and destination of the link are point references, also known as anchors [Haan, et al. 1992], which correspond to character positions in documents.
Figure 3.2 Conceptual Design
The Graphic serves a similar purpose as the Document, i.e., conveying information related to the node in the hierarchy. The Graphic however contains anchors that refer to graphic objects. A referential link connects the anchor in the graphic to an anchor in a Document. Thus a flow diagram for example could contain links to Documents that are associated with respective blocks in the diagram.

3.3 Translation of the Conceptual Model to the NEXTSTEP Environment

Once the conceptual design for the LSS was determined, the NEXTSTEP environment was investigated to determine objects, features and applications that directly supported the conceptual model. Figure 3.3 depicts the translation of the conceptual model into the context of the NEXTSTEP environment.

It was decided to use the WatchMe application to create animations. Images, or graphics, could be created by any application that supported the ".eps" and ".tiff" file formats.

The Edit application which is part of the NEXTSTEP environment, is used to create the document. The animation and images are easily incorporated into the document by "drag and drop". The document is stored as a file package with the ".rtfd" extension. In the file package the imported files are stored together with a Rich Text File (RTF), "TXT.rtf". "TXT.rtf" defines the formatted text information of the document. An imported image or animation is treated as a character in the document. It can be identified in the "TXT.rtf" file by the control word "\NeXTGrphic". The Edit application provides hypertext link capability for documents. A link source can be identified by the "\NeXTHelpLink" control word. An optional link destination to a character position in a document can be identified "\NeXTHelpMarker" control word.
Figure 3.3 Translation of Conceptual Model to the NEXTSTEP Environment

(adapted from [Nghiem 1993])
The Draw application could be used to create a graphic that contains anchors linked to anchors in other documents. The file structure and linking capabilities of the Draw application has to be investigated. Presently Interface Builder is used to create the Graphic. Buttons embedded in the Graphic serve as link icons.

The browser, a standard feature of the NEXTSTEP environment for navigating through hierarchical structures, provides the interface functionality for describing the hierarchical information structure in the conceptual model. A cell in the browser is associated with a node in the hierarchical information structure. Interaction with the browser provides an easy mechanism for traversing and selecting nodes in the hierarchy.

The Text and ScrollView classes provide extensive functionality for text manipulation and graphics incorporation. The flexibility of the Text class include: dynamically incorporating graphic cells as the text stream is read, search capabilities for strings within the text content, support for the RTF format, and file storage mechanisms that support streams and file packages. The Document View is easily implemented using the Text and ScrollView objects.

A Graphic View to draw ".eps" and ".tiff" is easily implemented. However for images that contain graphic objects as link sources the application specific file format has to be known, like the ".draw" file format for the Draw application. Alternatively, Graphic Views can be created with Interface Builder if the need for such views is rare.

Link icons can be implemented with button cells within the Document and Graphic Views to accomplish the functionality of hypertext/hypermedia link sources. The ButtonCell class on the NEXTSTEP environment provides the basic functionality for receiving and responding to mouse events.

3.4 Development Strategy
The development strategy followed was evolutionary prototyping. This methodology, as depicted in Figure 3.4, involves the development of a series of
prototypes that gradually evolve into the final product [Balci 1993]. This method is well suited to this research project for the following reasons:

- The complexity of the project could not be accurately gauged. Hence considering the time constraints certain major design decisions would have to be researched to determine its feasibility of incorporation into the framework of the purpose that the LSS is to serve.

- The LSS software developer is new to the NEXTSTEP environment. The capabilities of the system could only be ascertained with evolutionary experimentation on key issues.

- In order to determine that the correct product is being built for the required purposes, frequent communication was required between the software developer and the procurer. This was necessary to discuss current design decisions and to bring to the fore the difficulties of implementing some of these design strategies so that alternative ideas could be sought.

![Diagram](image)

Figure 3.4 Evolutionary Prototyping

23
3.5 Implementation Strategy

NEXTSTEP provides a rich assortment of interface objects to integrate into an application. The Interface Builder provides a mechanism to create interface files easily. The essential objects like buttons, text views, panels, etc. can be integrated by "drag and drop" using Interface Builder. Their attributes and connections to other objects can also be set where required. This reduces a more significant amount of coding and testing than would otherwise be required. Together with the Interface Builder a user friendly Project Builder makes it easy to manage a large application. Classes, header files, images, interface files can all be efficiently integrated into the application.

This environment provides a non intimidating atmosphere for the inexperienced NEXTSTEP developer. The developer experimented on the general structure and use of the interface objects, and the tools provided. The knowledge gained was then applied to the specific task at hand. Each module in the final application was first experimented with in standalone programs and then incrementally incorporated into the application. These formed the subprojects in the final application’s project directory in Project Builder.

3.5.1 NEXTSTEP Program Development Cycle

The NEXTSTEP program development cycle is shown in Figure 3.5 [Nghiem 1993]. This program development is used within the context of the broader development strategy, like evolutionary prototyping in the case of LSS.

A project is physically represented by a directory under control of Project Builder. The project directory together with source files common to all applications is automatically created when a new project is started on Project Builder. Other source files are added as the application develops and Project Builder manages these files for
1. Create a new project

2. Design the classes

3. Design the interface

4. Implement the application

5. Debug the application

Figure 3.5 NEXTSTEP Program Development Cycle
compiling, debugging, running and installing. Before any proper coding of the application can commence the application must be designed with the object-oriented approach. This involves identifying objects that the application is to model and functions those objects are to perform. These would form the class definitions which include the methods required to achieve functionality. Interface objects and other required objects that can be utilized from the library are identified. Most of the interfaces are then designed and tested using Interface Builder. The application is implemented through the use of Project Builder and the editor. The methods are coded and functionality of each module is tested. Other file sources are added, subprojects declared or added, icons imported, and the application is progressively built and tested. Testing the application is applied throughout the development of the software.

3.5.2 Interface Builder

Interface Builder is NEXTSTEP’s main development tool for writing applications. It lets the developer graphically design the windows that the application will use, together with all of the associated menus, buttons, sliders, and other objects. The objects once created can have their attributes set, and connected to each other and to other objects that they are a part of. All of these specifications are saved so that it can be used by the application when it runs. Interface Builder works together with Project Builder to provide a framework of source code to which a developer can add application-specific code. Interface Builder is more than a prototyper; it is an integral part of the NEXTSTEP programming environment. [Garfinkel and Mahoney 1993]

3.5.3 Project Builder

Project Builder is the hub of application development in NEXTSTEP. It manages
the components of the application and gives the developer access to the other
development tools needed to create and modify the components. The project is the unit
of organization for the Project Builder. The files viewed in the Project Builder panel are
organized into categories. The files though are stored under a project directory. These
categories include [NEXT 1992a]:

- **Classes** files containing code for custom classes used by an application
- **Headers** files containing declarations of methods and functions used by an application
- **Other Sources** files containing code, other than class code. These may be
  ".c" (standard C), ".m" (Objective-C), and ".psw" (PostScript) files.
- **Interfaces** Nib files that define the interfaces used in an application
- **Images** files containing images used by the application, including TIFF and EPS files
- **Other Resources** files, such as sound files, for other resources used by the application
- **Subprojects** directories for the subprojects used by the application
- **Supporting Files** files not directly used by the application but should be kept
  with the application
- **Libraries** Libraries referenced by an application.

There are three subprojects for the LSS application. This can be seen in Project
Builder as: Browser.subproj; Lesson.subproj; and LifeCycle.subproj. These subprojects
were built and experimented with individually to test the functionality and modularize the
application from a macro perspective. The files contained within each subproject were
further modularized through classes and functionality.
3.6 Why Not HelpBuilder and NXHelpPanel?

Initially it was envisaged that the Help Builder and NXHelpPanel class would be used to form the framework of LSS for document integration and displaying. However it was found that this method was too restrictive and certain major requirements of LSS were not being met by the help facilities provided by NEXTSTEP. These included:

- The NXHelpPanel class was inflexible to adjustment to interface functionality and look. All of the data in this class is private and hence buttons and other such interface objects could not be overridden by the LSS developer.

- The HelpPanel displays images. For the special case of the tape image however, it is required that the associated animation should play back when the tape image is selected. The Help Panel lacks in this functionality and neither can it be incorporated because the data for the NXHelpPanel class is private.

- The Help Builder would have to be used to integrate the documents into the system.

- All document files created would have to be part of the project. Integration of these files would pose extra manipulation on the part of the author of the documents since it would require a compilation each time a document is created.

- The author would require knowledge of Interface Builder and Project Builder in order for him to successfully integrate documents into the application.

- The HelpPanel follows a book format. There might be confusion between the hierarchy describing the information structure for a browser and the hierarchy inherent in a table of contents. The browser is used to reflect the hierarchically organized information that the user can access. However an hierarchically organized structure appearing on the table of contents for the Help Panel might not be reflected in the browser. Since the user traverses the browser for information, s/he might not be aware that further information is available through the structure in the Help Panel.

Due to the serious limitations imposed by the HelpPanel and inelegance of integrating documents, the help facilities provided by NEXT could not be used in the design of LSS. However to maintain standard look and feel the hypertext linking was performed in a similar manner that is provided by the on-line help facility.
3.7 The Application Kit and Implemented Subclasses

NEXTSTEP includes a library of more than fifty classes, which is collectively called the Application Kit (AppKit). Most interface objects are created using the AppKit. The AppKit classes provide support for implementing a user interface and responding to user input. Examples of these classes include Button, View, Window and Menu. NEXTSTEP also includes another collection of classes that provide more general functionality such as memory and storage allocation. These classes are called the Common Classes. The List and Storage classes were used in the LSS to manage list structures of objects and structured data respectively.[Ng him 1993]

Figure 3.6 shows the inheritance and hierarchical structure of the AppKit classes [NEXT 1992b]. The classes in underlined italics in Figure 3.6, e.g., LSSPrintPanel, denote the classes that were subclassed to provide additional functionality and to override some existing features so that it served the purposes needed for LSS. Table 3.1 gives a brief description for each of the classes implemented in the LSS application. Subsequent sections discuss the functionality of the main classes: BrowserControl, LessonControl and SearchControl; in more detail.

3.7.1 BrowserControl Class

In order to represent the hierarchical structure of the information, the browser is used. Hence BrowserControl fulfills the requirements of the Browser class, which include:

- Setting up the tree structure from the file "tree.dat". This involves reading each node information from the file and instantiating the Node class. Each node has the following information:
  - title  - a title to be displayed for the cell of the browser
  - filelink - the file pathname for the file that contains the content
  - owner  - the parent of the node
  - leaf  - flag denoting whether node is leaf or not.
A list representing the tree organized structure (hierarchy) of the documents would be set up, using the id's of the nodes.

Figure 3.6 AppKit Class Hierarchy with LSS Subclasses
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppControl</td>
<td>It is the main control object of the LSS application. It sets either the Developer or User mode by instantiating DevModeControl or NormalModeControl</td>
</tr>
<tr>
<td>BrowserControl</td>
<td>Sets up the browser to enable the user to browse through the hierarchical information structure and to enable him/her to select a particular node.</td>
</tr>
<tr>
<td>DevBrowserControl</td>
<td>Sets up the browser to enable the administrator to maintain the hierarchical information structure. Nodes and link information to the respective documents can be added, deleted, or modified.</td>
</tr>
<tr>
<td>DevModeControl</td>
<td>Control object for the Development mode. It sets up the menu for Development mode and responds to selections from the menu.</td>
</tr>
<tr>
<td>DocumentLinkCell</td>
<td>Instantiated when the control word &quot;NeXTHelpLink&quot; is encountered in the stream being read into the Text object, referred to as the LSS Document View. The DocumentLinkCell instance draws itself as a diamond shaped link button. It functions like a button cell. It can be queried for the link file information.</td>
</tr>
<tr>
<td>LessonController</td>
<td>Manages the Document View for viewing the contents of the document associated with a node.</td>
</tr>
<tr>
<td>LifeCycleDiagram</td>
<td>Manages the Graphic View for the life-cycle diagram. Responds to user selections to buttons on the diagram by loading the respective linked document into the Document View.</td>
</tr>
<tr>
<td>LSSDiagram</td>
<td>Instantiated when the control word &quot;NeXTGraphic&quot; is encountered in the &quot;Selection&quot; stream passed to a new Text instance during printing of a part of the open document. The LSSDiagram object draws the corresponding graphic without any button functionality.</td>
</tr>
<tr>
<td>LSSLink</td>
<td>During printing of a &quot;Selection&quot; of the open document the stream for the &quot;Selection&quot; is passed to a new instance of Text. The Text object will instantiate LSSLink for every &quot;NeXTHelpLink&quot; control word encountered in the stream. The LSSLink instance would draw only the image of the link button i.e. no functionality of a button is associated with the image.</td>
</tr>
<tr>
<td>LSSPrintPanel</td>
<td>Accepts user print options and selections. Incorporates radio buttons for &quot;Selection&quot; and &quot;All&quot; enable the user to select the printing of a part of the document or the full document.</td>
</tr>
<tr>
<td>Node</td>
<td>Contains instance variables that describe the node in the hierarchical information structure. It is instantiated for each node read from the file &quot;tree.dat&quot;.</td>
</tr>
<tr>
<td>NodeNXBrowserCell</td>
<td>Adds an instance variable, the id of a Node instance, to the NXBrowser class. This associates the cell in the browser with a particular node in the hierarchical information structure.</td>
</tr>
</tbody>
</table>

Table 3.1 Description of Implemented Classes for the LSS Application
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NormalModeControl</td>
<td>Control object for the <em>User</em> mode. It sets up the main menu and responds to the selections made. It instantiates BrowserControl and implements the print capability.</td>
</tr>
<tr>
<td>SearchControl</td>
<td>Manages the search for a string in all the documents that make up the hierarchical information structure.</td>
</tr>
<tr>
<td>TapeButtonCell</td>
<td>Instantiated when the control word &quot;\NeXTGraphic&quot; is encountered in the stream being read into the Text object which is the <em>Document View</em>. The TapeButtonCell instance draws the graphic that the control word describes. It sets the tape image used for animations to functions as a button cell.</td>
</tr>
<tr>
<td>TempCell</td>
<td>For a string search in all documents in the hierarchical information structure, only the textual information is required. When the stream for the document being searched is read into a Text object, the TempCell class is instantiated when the control word &quot;\NeXTGraphic&quot; or &quot;\NeXTHelpLink&quot; is encountered. The TempCell instance extracts the string associated with the control word from the stream, thereby enabling only textual information to be read into the Text object.</td>
</tr>
</tbody>
</table>

Table 3.1 Description of Implemented Classes for the LSS Application (continued)

- Using the information contained in the tree structure, BrowserControl loads the required column in the browser when it is queried to do so.

- Sets the browser's cell structure to be of class NodeNXBrowserCell. This is required so that each cell in the browser can be set up to contain the node id for the node in the information structure. Hence each browser cell can be queried for the id of the associated node.

- Responds to the mouse actions of the browser. If a double click message is received BrowserControl would message LessonControl to load the required document. The document file name is obtained by querying the selected cell for the node and then extracting the linked file information.

3.7.2 *LessonControl Class*

This class provides the control for the document panel. These include:

- responding to the double click request from the browser to load the associated document,

- setting up graphics capabilities viz. adding cells for hypertext links, tapes for animations and other graphic pictures,
• responding to actions of the graphic cells viz. activating an animation and loading a linked document file,

• and performing the local search for a string in the open document when a search request is received.

3.7.3 SearchControl Class

SearchControl was set up to manage the search through all the associated documents in the tree structure and to co-ordinate the search in all the documents with the local search. The local search occurs in LessonControl as it controls the activities of the open documents. The sequence of tasks that SearchControl is responsible for include:

• Checks if a local search was done previously. Uses this string as a default and searches in all the documents.

• Searches for the string input by the user.

• The search mechanism is facilitated by creating a Text object and loading it repeatedly with each file in turn for each node in the tree. Since the files also contain graphic elements the TempCell class is used to create a temporary dummy cell that takes in the descriptive graphics information but does not draw itself.

• The Text object is queried each time it is loaded to check if it contains the required string. If the string is found then a list which contains all documents found thus far, is updated.

• A single column browser is used to provide to the user the linear list of documents found.

• A selection made in this browser would message LessonControl to load the associated document and perform a local search so that the user is taken to the first occurrence of the string in the document.
3.8 Implementation of Hypertext Links and Graphics Incorporation

The central object for displaying the information in the LSS is the Text object. The Text class inherits from View and hence it is possible to incorporate graphics as cells in this view. The Edit application identifies the graphics cells in its files through control words. A part of the file structure is shown in Figure 3.7. When the file stream is read into the Text object in LSS, it uses the `registerDirectiveForClass:` method to instantiate a class which is a subclass of cell.

For the hypertext links the class DocumentLinkCell is instantiated when the control word "\NeXTHelpLink" is encountered in the stream being read. The DocumentLinkCell object would then read the subsequent control words to determine the linked file name (\linkfilename), and an associated marker name (\linkmarkername). The system image named NXLinkButton is used to provide the graphic for the cell. The cell is subclassed from ButtonCell to provide functionality of a button to the help link.

To process the graphics images the Text object looks for the control word "\NeXTGraphic". It then instantiates the TapeButtonCell class. The TapeButtonCell object then reads the image file name and the width and height of the image that is to appear in the Text view. The image file is part of the file package for the document being read. Hence depending on whether it is an ".eps" or ".tiff" format the image for the cell is drawn. The tape image however requires that the cell function like a button allowing the action of a click on it to run the WatchMe application with the designated animation file. For all other image cells there is no added functionality besides drawing itself. To differentiate the functionality of the tape image from any other image the image file name is checked for the ".watch" extension. For example if the string after the control word "\NeXTGraphic" is "tape.watch" then the image is read from "tape.watch.tiff" and the application WatchMe is run with the file "tape.watch". These files would be part of the
LSS has been developed to support you in the learning process towards acquiring knowledge on the VSE and Simulation. To view a demonstration on the use of LSS click on the tape image:

Figure 3.7 "RTF" File Created By Edit Application
file package for the document.

A hypertext link associates one file with another. A marker further associates a hypertext link to a position in the associated file. Markers are not seen by the user. They are identified as the "\NeXTHelpMarker" control word. When a link button is activated the linked file is not immediately loaded into the text object. It is first parsed for the "\NeXTHelpMarker" control word to form a list of markers. If there is a marker associated with the link button its position is obtained from the list. The Text object is then loaded with the stream and moved into view at the position of the marker. The first word that appears after the marker is highlighted. If there is no marker then the linked document is loaded with the beginning of the document brought into view.

3.9 The LifeCycleDiagram Class

In the conceptual model a Graphic class to manage the incorporation of full pictures with hypertext links embedded within the structure was considered. A separate tool would be required to construct such an image with hypertext links. The Draw application does support hypertext linking but through object links. Object links is implemented through an additional file that contains the link information. This link information is managed by the LinkManager class. Due to the effort required to decipher the Draw file format, incorporate code to process object links and the rarity of occurrence of the Graphic class feature, it was decided that Interface Builder would suffice and be much faster in building the graphic image with links. This method loses the flexibility of having just one graphic class definition into which a file definition of the graphic with links, could be read. This leads to programmatic incorporation of this feature for each graphic with embedded hypertext links.

The LifeCycleDiagram class is one of the cases which requires that a diagram contain links to other files for different selections within the diagram. In Interface Builder
the life cycle image file is included as the image of a button. The button is the size of the image and it is set to be disabled as there is no use for any functionality other than displaying of the life cycle image. For all the areas that require the hypertext links invisible buttons are inserted. The functionality for each button is then set to load the required file. This results in a life cycle diagram that allows the user to click on the text in order to load a document that would contain further information.

In order to incorporate the link for the images like the life cycle diagram into one of the browser selections the "classname.class" format is used for the link information of the cell. This differentiates it from other cells which have links to files with ".rtf" and ".rtfd" extensions. Hence for the life cycle diagram the "LifeCycleDiagram.class" is set as the link for the cell that loads the diagram. When the cell is selected the LifeCycleDiagram class is instantiated.

3.10 The Print Feature

Using the evolutionary methodology, it was realized that the print feature required an additional support for printing of selected text. Hence the LSSPrintPanel class was subclassed from the AppKit’s PrintPanel class to provide the additional functionality and control. The subview for "Range" selection was added. Since the print panel is run modally it is presented first after a print request. According to the user selection, the modal loop is broken and the required operation for either "Selection" or "All" is carried out with the respective text object. The print feature incorporates a "resizable to page fit" of the information contained in the window. This means that if the window is resized horizontally beyond the range of the page width, then the size of the text that is printed is scaled before printing so that it fits within a page length. This is fairly easy to accomplish as all printing is done in PostScript.
3.11 The Icon Design

The icon was designed using the Icon Builder tool. The image was then incorporated into ProjectBuilder and was part of the application when compiled and run. The idea for the use of the graduation cap as the image for the LSS application was obtained from the previous SMDE Assistant Manager. In the SMDE Assistance Manager the image of the graduation cap was a "wire" image. The image constructed for the LSS icon has better definition than the icon used in the SMDE Assistant Manager and the word "LSS" appears repeatedly as part of the image. Attempts at other icon images were tried, like a chalkboard and books. However, the image developed seems to be the best as it is easily recognizable. Associated with "learning" as in student, graduation, university and professor this image seems to be the most apt. The icon is shown in Figure 3.8.

Figure 3.8 LSS Application Icon
Chapter 4
USER'S GUIDE

The LSS has two modes of operation; the developer mode and the user mode. In the developer mode, features are provided for the creation and maintenance of the documents. Users would interact in the user mode to query for required documents during the learning process. When the application is launched the default mode is the user mode.

4.1 Developer Mode

The developer mode is activated by selecting "Info -> Developer" from the main menu. The user would then be queried for a password. If the correct password is entered the developer mode is enabled otherwise the application remains in the user mode of operation.

Creating a document and linking documents in the browser are the two main processes required when integrating documents into the LSS.

4.1.1 The Developer Mode Menu

The main menu items are "Lesson" and "Tape" as shown in Figure 4.1. Selection of the "Tape" menu option would result in the WatchMe application being loaded so that a tape can be created. The selection of the "Lesson" menu option allows for the maintenance of LSS documents. The "Lesson->New" option results in the template.rtf file being loaded with the Edit application. The "Lesson->Open" selection would present the OpenPanel set to the default directory of "/lss/Database". The required file to be
opened can then be selected. The file is opened using the Edit application. The "Lesson->Save Links" option provides the means to save the links in the browser when they are changed. The "Lesson->Update Links" serves the purpose of updating the links in the browser. Facilities for adding, deleting, and modifying links are provided.

<table>
<thead>
<tr>
<th>LSS DEV</th>
<th>Lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>New</td>
</tr>
<tr>
<td>Lesson</td>
<td>Open</td>
</tr>
<tr>
<td>Tape</td>
<td>Update Links</td>
</tr>
<tr>
<td>Windows</td>
<td>Save Links</td>
</tr>
<tr>
<td>Services</td>
<td></td>
</tr>
<tr>
<td>Hide</td>
<td>h</td>
</tr>
<tr>
<td>Quit</td>
<td>q</td>
</tr>
</tbody>
</table>

Figure 4.1 Developer Mode Menu

4.1.2 Creating a Document

In order to open a new file the "Lesson->New" menu option is chosen. The file "template.rtf" would be opened. The content for the document can then be entered. Appendix A describes using Edit to create documents that contain images, animations and hypertext links. The document when complete should be saved as the desired filename for that document. The browser list should then be updated to integrate the document into the browser. This is described in the next section.

4.1.3 Linking Documents into the Browser

The entries in the browser in the Developer mode have their leaf elements as "NULL" as shown in Figure 4.2. This is intended so that an entry can be made into the
next column by selecting the "NULL" entry and performing one of the link update operations. Using the close window button the user can exit the application. He would be prompted to save the list of entries of the browser if it has been altered.

![Figure 4.2 Browser in Development Mode](image)

To update the links the "Lesson->Link" menu item is chosen. The "Browser Entry Link" panel, depicted in Figure 4.3, would be presented. The title and the linked filename attributes for selected browser entry appear on the panel. Hence the panel provides a means of viewing the file links as the hierarchy is browsed.

![Figure 4.3 Browser Entry Link Panel](image)
To modify the title and file link for the currently selected cell the "Title" and "File" fields can be changed in the Browser Entry Link panel. The "File" field can be set by selecting an existing file from the directory. The "Set..." button, next to the "File" field, is used for this purpose. The "Select Link File" panel, shown in Figure 4.4, is opened when the "Set..." button is selected. A file can then be selected which replaces the data in the "File" field. Once the "Title" and "File" fields have been changed, the "Set" button should be selected to reflect the change in the entry. The browser would then be updated with the changed data for the selected entry.

To add a new entry into the browser the point of insertion must be selected in the browser prior to performing the "Add" operation. The "Title" and "File" fields in the Browser Entry Link panel can then be changed to correspond to the new entry’s data. Thereafter select the "Add" button which inserts the entry, with the values specified in the Browser Entry Link panel, at the position of the selected entry in the browser.
To delete an entry and its branch elements, the entry must first be selected in the browser and then the "Delete" button selected. An alert panel would be presented to ensure that the "Delete" operation is the desired action.

Once the links have been updated, save the list by selecting "Lesson -> Save Links" from the menu.

For graphic classes the "File" field which links the browser entry to the document should be set to "classname.class". For example, to set the life cycle diagram as the required document to be opened then "LifeCycleDiagram.class" should be entered into the "File" field.

4.2 User Mode

The application is in user mode by default when it is launched. However to switch to the user mode from developer mode the "Info -> User" option must be selected. Figure 4.5 shows the menu, browser, and document viewing panel for a typical session in LSS.

4.2.1 Selecting a Document

The browser shown in Figure 4.5 provides the mechanism to browse through the hierarchical structure of the document organization. Entries with a "†" symbol denote that further entries are available at the next level. To go to the next level a single left mouse click would load the next level of entries for the selection that was clicked. A double click on the entry in the browser would load the associated document file.

4.2.2 Using Hypertext Links

The graphic "🔗" indicates a link button in the document being viewed. To view the
Virginia Tech Visual Simulation Environment (VTVSE) Learning Support System (LSS) is a software system that provides computer-assisted learning for the VTVSE and visual simulation modeling. You can use the browser shown above to browse through the hierarchical information structure by clicking on a line item which contains a triangular icon showing that the item is further decomposed. You can double click on a line item to open the hypertext document associated with that line item. The documents contain hypertext links represented by the diamond shaped blue icons as shown below. Clicking on a hypertext link icon results in the display of a related hypertext document. The documents may also contain a video tape icon as shown below. Clicking on a tape icon results in a demonstration recorded on the computer. Click on the video tape icon below to see a demonstration on the use of the LSS:

Introduction to LSS

Figure 4.5 A Typical Session In LSS
document associated with the link button, the button should be clicked. The linked
document would then be loaded into the document viewing panel. The title of the panel
however is left unchanged so that the context of the original selection is maintained.

4.2.3 Activating an Animation

If the document being viewed contains tape images, similar to the one shown in
Figure 4.6, it indicates that there are animations that can be viewed. A click on the tape
image would run the corresponding animation. Once the animation is completed, click on
the LSS browser or document viewing panel to return to LSS. To stop an animation, click
anywhere on the screen. Thereafter select the LSS application.

Figure 4.6 Tape Image For Animation

4.2.4 Using a Graphic that Contains Links

There are diagrams or images that contain links to other documents within them.
These are normally displayed alongside the document viewing panel. Selections can be
made from the graphic so that it loads the relevant linked document into the document
viewing panel. The LSS thus far contains only one graphic that contains embedded links.
This is the "Simulation Life Cycle Diagram". Each of the flows and nodes that are
labelled on this diagram can be clicked on, selecting the respective text to load further
documentation with respect to that selection. The diagram would remain in view and it
can be resized, and the vertical and horizontal scroll bars used to view it. Only another selection from the browser or closing of the graphic viewing window would remove the view. The window with the life cycle diagram is shown in Figure 4.7.

Figure 4.7 Simulation Life Cycle Diagram
4.2.5 Searching

Searching for a string within the current open document and searching in all documents in the browser are provided. From the main menu in Figure 4.8, "Find" should be clicked and thereafter "Current" or "All" to activate a search within the current document or a search through all the associated documents in the browser respectively.

<table>
<thead>
<tr>
<th>LSS</th>
<th>Find</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>Current f</td>
</tr>
<tr>
<td>Find</td>
<td>All</td>
</tr>
<tr>
<td>Print</td>
<td>p</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>Services</td>
<td></td>
</tr>
<tr>
<td>Hide</td>
<td>h</td>
</tr>
<tr>
<td>Quit</td>
<td>q</td>
</tr>
</tbody>
</table>

Figure 4.8 Main LSS Menu with Find Selected

For the "Current" selection a search panel as shown in Figure 4.9 appears. A string should be typed in and "Enter" pressed or the mouse clicked on the "Next" button. Each click on the button would find the next occurrence of the string. The search mechanism works on a wrap around mechanism. The string if found would be highlighted in the open document. If not found, no highlight of any text is shown on the open document, and the string in the find panel is grayed out to allow another string to be easily entered.
If "Find -> All" is selected then the panel in Figure 4.10 is presented. This panel has two main components; the field to input the search string and a scrollable list that contains all the entries that contain the search string. Clicking on an entry in the list would update the document view panel with the required selection. The "Find - Current Document" panel in Figure 4.9 would also be activated, and a local search for the string in the selected document could commence. If the "Find -> All" is selected with the "Find - Current Document" already open then "Find - All Documents" would do an automatic search in all documents with the string in the "Find - Current Document" used as the default search string. The user could specify a different string thereafter if desired. If the string is not found in any of the documents then the list would contain no entries.

4.2.6 Printing

The printing includes the features of printing the full open document or a selection of it. The print panel shown in Figure 4.11 has the same functionality as the general print panel used in all applications for NEXTSTEP. Added to it is the "Range" feature to enable the printing of a selection or the full document. Windows that have been resized to be larger than the standard size will result in the printout being scaled in order to fit the width of the page.

To print a part of the document, the document has to be marked before printing can
commence. Marking is achieved by holding down the mouse button and dragging until the desired area has been selected. The selected area would be highlighted in gray.

Thereafter select "Print" from the LSS menu. The print panel would then be presented. Under the "Range" option in the print panel the "Selection" button should be clicked. The "Print" button from the print panel can then be selected to print.

To print the full document, select "Print" from the LSS menu, and then the "Print" button from the print panel.
Figure 4.11 The Print Panel
Chapter 5

CONCLUSIONS AND
RECOMMENDATIONS FOR FUTURE RESEARCH

5.1 Conclusions

The LSS was designed to present to the user a vast amount of knowledge contained in the VSE and in simulation and modeling, in an organized and effective manner. To achieve this goal the LSS incorporated the following features:

- A hierarchical representation of the information structure through the use of the browser.

- The use of graphics and text within documents.

- Animations contained within documents to demonstrate an interaction with the software. The sound and screen images for the session are played back to the LSS user.

- Hypertext links within documents facilitate the request of further relevant information.

- The LSS has capabilities to incorporate diagrams with embedded hypertext links. Hence the diagram as well as the document can be viewed.

To maintain the overall effectiveness of LSS, the creation and incorporation of documents was important. To ensure simplicity, the following strategy was pursued:

- The NeXTSTEP provided Edit application is used to create documents.

- The WatchMe application is used to create animations.

- The Developer mode is provided in LSS to link the documents into the information structure.
The NeXTSTEP operating environment supported the efficient creation of documents, and provided features that led towards effectiveness of LSS. These included:

- An extensive set of effective and well designed tools for developing the application.
- The object-oriented programming environment provided for modularization of the program structure and inheritance facilitated the easy incorporation of newer features into objects through subclassing.
- The NeXTSTEP rich library of classes allowed for the features of the LSS to be quickly built and prototyped.

5.2 Recommendations for Future Research

Computer technology is constantly changing, hence the LSS application should be updated to reflect the current trends so that it meets user expectations. The following immediate recommendations are suggested:

- The animation is run through the WatchMe application. It is possible to purchase the classes and incorporate them into LSS. This would provide a greater flexibility into allowing users to stop an animation for example.
- Investigation should be carried out into doing the animation in a scaled form in a separate window instead of having the full screen occupied.
- Feedback from students using the LSS should determine the future changes that are required.
- Feedback from the document creators should be taken into account to incorporate further capabilities to the LSS. Support for more graphic file formats is an example.
- One common graphic class, as envisioned in the conceptual design, can be created in the future if the need arises for more graphic files to have embedded links. The NeXTSTEP Draw application can be used to create and store the graphic file.
Bibliography


NEXT Computer, Inc. (1990), *NeXTstep Concepts*, Next Computer, Redwood City, CA.


Appendix A

USING THE EDIT AND WATCHME APPLICATIONS

This appendix provides a brief introduction to the Edit and WatchMe applications used to create documents and recorded sessions for the LSS respectively.

A.1 Creating a Document with Edit

The Edit application is a NEXTSTEP supplied tool for creating formatted documents. Edit can be started up from the icon in the dock or by opening an RTF file. Text formatting such as font, margin setting and so on are easily implemented using the menu options in Edit. The general wordprocessing functions of the Edit application are fairly easy to grasp and [NEXT 1992a] can be consulted. The incorporation of graphics and providing hypertext links will be explained in this section.

The Edit application allows for graphics incorporation into a text document by dragging and dropping the image into the document. The image will be placed at the location of the cursor in the document. Hence the cursor location should be set before the image is imported. When the image is inserted, Edit creates the document as a file package with the "rtfd" extension. An image can also be inserted through the "copy/cut and paste" method.

It would be better to open the file "template.rtf" to create a new document than using the default "untitled.rtf" file that the Edit application opens as a new file. The document can then be saved under a different name. This alleviates the possible duplication and confusion of first creating an "rtf" file and then importing images into it and then saving it as an "rtfd" file.

Hypertext link facilities are created in documents using the Edit application in
"Developer" mode. To set Edit in Developer mode select "Info->Preferences->User Options" and then select the "Developer" button. A link connects an anchor in the source document to an anchor in the destination document. A link is created by associating the anchor in the source document to the destination document's file name. The destination of the link would be the destination document. If the anchor in the source document is selected then the beginning of the destination document would be brought into view. An optional anchor in the destination document, also known as a marker, can be associated with the anchor in the source document. This link would then connect the anchor in the source document to the anchor (marker) in the destination document i.e. the position corresponding to the anchor in the destination document would be brought into view when the anchor in the source document is selected.

A.1.1 Creating a Hypertext Link

To create a hypertext link the cursor must be placed in the position that the link icon [Conklin 1987] is to be inserted. In the case of the document in Figure A.1, it was positioned at the start of the line that "Boundary Analysis" appears. The "Format->Help->Insert Link" menu option is then used to insert the link. Once the blue colored diamond shaped link is inserted a Link Inspector panel appears as shown in Figure A.1. In this panel the required link file should be entered by either selecting "Set..." or entering the file information through the keyboard. It is best to use "Set..." to avoid any mistakes. An optional marker name can be entered if required. Select the "OK" button to complete the operation. For the example shown in Figure A.1 the link at the text location "Boundary Analysis" for the document "listOfTerms.rtfd" was associated with "glossary.rtfd" and the marker "m4". To update a link perform a "Command -- Click" (hold Command key and click left mouse button) on the link icon. This will bring up the inspector panel. The fields describing the link information can then be changed.
A.1.2 Inserting a Marker

To insert a marker the cursor must be placed at the location that the marker is desired. Select the "Format->Help->Show Markers" to make the markers visible. Then select "Format->Help->Insert Marker". In Figure A.2 the linked file for the hypertext link that was created, "glossary.rtfd", was opened. The cursor was placed at the location of "Boundary Analysis". The "Format -> Help -> Insert Marker" menu item is then chosen to insert a marker. A Marker Inspector panel shown in Figure A.2 is presented. The name of the marker is entered as "m4" in the marker name field of the panel. This associates the position in "glossary.rtfd" with the hypertext link created in "listOfTerms.rtfd". Hence when the hypertext link is selected in "listOfTerms.rtfd" the file "glossary.rtfd" would be opened and the position "m4" would be brought into view.

A.2 Creating an animation with WatchMe

Animations are created and played back using the WatchMe application. An interaction with the computer is captured via the screen images that appear during the recorded interaction. The tape that is created can be edited for its sound and images.

When the WatchMe application is run the recorder panel in Figure A.3 is displayed. To record a tape the following sequence of general steps should be followed:

1. Click the record button in the Record window.
2. Go through the session explaining the sequence of events in the interaction.
3. Complete the recording by clicking on the Stop button in the Recorder Window.
Figure A.1 Associating Link Information to a Link Icon (anchor)
Glossary

◆ **Assertion Checking**: A technique for examining what is happening against what the modeler assumes is happening so as to guard model execution against potential errors.

◆ **Audit**: A technique for investigating how adequately the simulation study is conducted with respect to established practices, standards, and guidelines and for establishing traceability within the simulation study.

◆ **Black-Box Testing**: A technique for assessing the accuracy of model input-output transformation. It is applied by feeding inputs (test data) to the model and evaluating the corresponding outputs. The concern is how accurately the model transforms a given set of input data into a set of output data.

◆ **Bottom-Up Testing**: A technique, used in conjunction with bottom-up model development strategy, in which testing starts with the submodels at the base level (i.e., the ones that are not decomposed further) and culminates with the submodels at the highest level. As each submodel is completed, it is thoroughly tested. When submodels belonging to the same parent have been developed and tested, the submodels are integrated and integration testing is performed in a bottom-up manner until the whole model has been integrated.

◆ **Boundary Analysis**: A technique for assessing the boundaries of input equivalence classes which are function the same way.

**Calibration**: An iterative process in which a predefined fixed value for a parameter is tried until the model response is within acceptable limits.

◆ **Cause-Effect Graphing**: A technique conducted on the model representation—"Is there any system component which is not modeled and by examining if they are accurately reflected in the model specification. Then, the semantics are expressed in a cause-effect graph. A decision table is created by tracing back through the graph to determine combinations of causes which result in each effect. The decision table is then converted into test cases with which the model is tested.

**Communicative Model**: A model representation which can be communicated to other humans and can be judged or compared against the system and the study objectives by more than one human [Nance 1981].

**Communicative Model VV&T**: Confirming the adequacy of the communicative model to provide an acceptable level of agreement for the domain of intended application.

**Conceptual Model**: The model which is formulated in the mind of the modeler [Nance 1981].

**Consistency Checking**: A technique which deals with substantiating that: (a) the model representation does not contain contradictions, (b) the cosmetic style with which language elements (e.g., naming conventions, use of upper, lower, and mixed case, etc.) are applied is used consistently, and (c) the data elements are manipulated uniformly (e.g., data assignment to variables, data use within computations, data passing among submodels, data representation and use during model input and output).

**Data Flow Analysis**: A technique for assessing model accuracy with respect to the use of model variables. A data flow graph is constructed to aid in the data flow analysis. The nodes of the graph represent statements and corresponding variables. The edges represent control flow.
Select "Info -> Preferences" from the WatchMe menu and then ensure that the "Show recorder window" button is unchecked. This will prevent the recorder window from being displayed when the tape is played back. The recorded tape can then be stored away by selecting the "Tape -> Save" menu item. It is advisable to set the background color different from the default setting before recording a session. This would help alleviate confusion between a normal session and a played back recording. The WatchMe on-line help should be consulted for additional facilities that are available to edit the recorded tape.

The tape can easily be incorporated into the document created under the Edit application. The tape icon can be dragged from the workspace and dropped into the document view. The tape would be inserted at the location of the cursor in the document.
Appendix B
THE LSS SOFTWARE STRUCTURE

This appendix describes the LSS software structure. Table B.1 depicts the interface and class files found in the LSS project.

<table>
<thead>
<tr>
<th>Project/Subproject</th>
<th>Interface Files (.nib)</th>
<th>Classes (.m and .h files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSS main project</td>
<td>infoPanel, LSS, menuDEV, menuNOR, password selection</td>
<td>AppControl, DevModeControl, LSSDiagram, LSSLink, LSSPrintPanel, NormalModeControl</td>
</tr>
<tr>
<td>Browser.subproj</td>
<td>theBrowser, updateLink</td>
<td>BrowserControl, DevBrowserControl, Node, NodeNXBrowser</td>
</tr>
<tr>
<td>Lesson.subproj</td>
<td>find, lesson, search, searchTextPanel</td>
<td>DocumentLinkCell, LessonControl, SearchControl, TapeButtonCell, TempCell</td>
</tr>
<tr>
<td>LifeCycle.subproj</td>
<td>lifecycle</td>
<td>LifeCycleDiagram</td>
</tr>
</tbody>
</table>

Table B.1 Interface and Class Files in the LSS Project

The software structure diagram shown in Figure B.1 is organized in the form of the "owner" concept: the object that creates an object is regarded as the "owner" of the created object. Owners relay messages to the objects they create and the created objects send messages to other objects via their owner. This approach isolates and modularizes the communication between objects, thus allowing for an easier understanding and maintenance of the program. The messages, known as methods in
Figure B.1 The LSS Software Structure
Figure B.1 The LSS Software Structure (continued)
Figure B.1 The LSS Software Structure (continued)
Figure B.1 The LSS Software Structure (continued)

NEXTSTEP/Objective-C terminology, that are communicated between objects are shown through the arrow notation. Methods and functions local to a class are not included in the message diagram.
The Application instance sends `init` to initialize an instance of AppControl, as shown in Figure B.1, page 62. This sets up an initialization of the NormalModeControl instance with `init`. The `setDevMode:` and `setNormalMode:` messages set the development or normal mode of operation. The objects for the mode that is being changed is freed and the objects for the new mode are initializes by these methods.

The NormalModeControl instance would load the interface file "menuNOR.nib". This initializes the menu for the normal mode. The description for the target methods to the selections in the menu for the User mode are included in Table B.2.

<table>
<thead>
<tr>
<th>User Mode Menu Target Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>devModeRequest:</code></td>
<td>Loads the interface file &quot;password.nib&quot; to request the password from the user. The target method for the password panel, <code>checkPassword:</code>, verifies the password, frees NormalModeControl and initializes DevModeControl.</td>
</tr>
<tr>
<td><code>processPrint:</code></td>
<td>Initializes the print panel with the range accessory view by sending to LSSPrintPanel the <code>initPrintPanel:</code> message. The print preference for &quot;All&quot; or &quot;Selection&quot; is obtained through the <code>getSelection:</code> method to the print panel. This occurs after a print option has been selected. Local methods <code>printLesson:</code> is used to print either the full document or a part of it, and <code>setPrintScale</code> is used to scale the document if the view is larger than the width of a page. The <code>setPrintScale</code> method uses the global PrintInfo object to set the scaling factor. To print out a selection the <code>printLesson:</code> method initializes a new Text object with the same dimensions as the Text object with the full view. This new object is left in the background. The selected area for the full document is copied into the stream. The new Text object is loaded with this stream. It instantiates the LSSLink or the LSSDiagram class if the control word &quot;NeXTHelpLink&quot; or &quot;NeXTGraphic&quot; respectively, are encountered as the stream is read. The LSSLink object is responsible for creating the image of the blue link button. The LSSDiagram object uses the id of the image, that was written into the stream by the TapeControlCell object of the DocumentView, to draw itself.</td>
</tr>
<tr>
<td><code>processFindCurrent:</code></td>
<td>Queries the BrowserControl instance for the id of LessonControl through the <code>getOpenLesson</code> method. The LessonControl object is then sent the message <code>findInOpenDoc:</code>.</td>
</tr>
<tr>
<td><code>processFindAll:</code></td>
<td>Sends to the LessonControl object, after obtaining the id from BrowserControl, the method <code>findForAllDoc</code>.</td>
</tr>
</tbody>
</table>

Table B.2 Description of User Mode Menu Target Methods
If the AppControl object sends an init: message to DevModeControl then the interface definition file "menuDEV.nib" is loaded and DevBrowserControl is initialized. The Developer mode menu target methods and their descriptions are given in Table B.3.

<table>
<thead>
<tr>
<th>Developer Mode Menu Target Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalModeRequest:</td>
<td>Frees NormalModeControl and initializes DevModeControl.</td>
</tr>
<tr>
<td>processNew:</td>
<td>Opens the file &quot;template.rtfd&quot; under the Edit application</td>
</tr>
<tr>
<td>processOpen:</td>
<td>Presents the Open Panel and opens the user requested file.</td>
</tr>
<tr>
<td>processLinks:</td>
<td>Sends to DevBrowserControl the instance method doUpdateLinks to update the link information in the browser.</td>
</tr>
<tr>
<td>processSave:</td>
<td>Saves the list that contains all the link data to file through the writeListToFile method toDevBrowserControl.</td>
</tr>
</tbody>
</table>

Table B.3 Description of Developer Mode Menu Target Methods

In normal mode the init: method to BrowserControl, as shown in Figure B.1, page 63, results in an init: message to List. A local method readListFromFile initializes the list from the file "tree.dat". The nib file "theBrowser.nib" is loaded. This is the browser panel interface object. The awakeFromNib method which is run just after the nib file is loaded, sets the cell class of the browser to NodeNXBrowserCell. The first column of the browser is loaded through the delegate method browser:fill:Matrix:inColumn:. The LessonControl is then initialized with the initial file associated with the first element in the List instance. This completes the initialization sequence for BrowserControl.

The readListFromFile method opens a typed stream to retrieve the title, file link, node owner, and the leaf status information. For each set of these variables the Node class is instantiated, set with the init::: message, and its id is added to the list. Whenever a
column in the browser needs to be loaded the delegate method

`browser:fill:Matrix:inColumn: ` is invoked. The browser is queried for the selected cell. The cell which is of type `NodeNXBrowserCell` is then queried its Node `id` instance variable through the `node` method. The List is then searched for the nodes that have the selected cell’s node `id` as their parent. These are loaded as the next column cells. The leaf status of the cell is set through `setLeaf:`. `setStringValue:` initializes the title of the cell. The values of leaf status and cell title are obtained by querying the respective Node instance.

`processDoubleClick:` responds to the double click on a cell in the browser. The associated node of the selected cell is queried. The `fileLink` method to the node returns the link to the file. If the file link has the ".class" extension then the specific graphic class is instantiated with the name of the class since the file link would appear as "classname.class". For ".rtf" and ".rtfd" files the LessonControl is sent the message `readFromRTFFile:` to load the file associated with the cell.

`getOpenLesson` returns the id of LessonControl. The `windowWillClose:` method responds to the close button on the browser panel. It frees the BrowserControl and LessonControl instance. A selection of "Lesson" in the menu would instantiate these classes.

In Figure B.1, page 64, the designated initializer method for LessonController is `init:withFile:`. It loads the file "lesson.nib" for the text panel. The Text object is sent the class method `registerDirective:forClass:` method to set the instantiation of the DocumentLinkClass and the TapeButtonClass when the control words "NeXTHelpLink" and "NeXTHGraphic" are encountered when the text stream is read into the text object.

The method `readFromRTFFile:` is invoked to load the file into the text object. This method reads the file into a stream. The text is then loaded into the text object with the `readRichText:` method. The selection is set to the beginning of the document and the text object is scrolled to the selection with `scrollSelToVisible:`. A function
parseForLinkMarkers() is used to retrieve all the markers contained in the stream into a Storage object. Storage is used to create a list of structured data. The name and character position are retrieved for each marker encountered in the stream. The marker is identified by the "NeXTHelpMarker" keyword in the stream.

The search mechanism results in a local search through the findInOpenDoc: method and a search in all documents through the findForAllDoc method.

**findInOpenDoc:** loads "find.nib" to allow the user to enter a search string in the find panel. findNext: is the target method for the "Next" button on the find panel. It reads the input string in the find panel and then searches the text for the string with the findText:ignoreCase:backwards:wrap: message. findForAllDoc initializes the SearchControl object. The list of the nodes is obtained by sending the list message to the BrowserControl object. If there is a local search panel open then that string is used for the initial search in all the documents. When the SearchControl object is sent the init message, it loads "search.nib" and allocates a List object to maintain the nodes that have associated files which have the string that is being searched for. The **doSearchFrom:forString:** method to SearchControl accomplishes the search in all the documents. This method uses the local method searchForString:. searchForString: loads "searchTextPanel.nib", initializing a Text object. For each of the nodes in the List, the associated file is loaded into the Text object and searched for the string. The nodes that contain the string are added to a list of found documents. The browser in the Search Panel is updated with the list of found documents. The TempCell class is instantiated when the text object encounters the NeXTGraphic and NeXTHelpLink control words in the stream. The TempCell object acts as a dummy graphics cell, "drawing" an empty cell, as it is only the text that is required. **doSelection:** is the target message for the click of a selection in the single column browser found in the search panel. This method loads the corresponding document sending to LessonControl the instance message

**readFromRTFFile:**. If the "find in current document panel" is not instantiated it sends
**findInOpenDoc:** to LessonControl to perform a local search in the newly opened document, so that the first occurrence of the string is brought into view.

The cells that are formed by the Text object, tapes and hypertext links, use the **readRichText:forView:** method to read the additional information that comes after the control words NeXTGraphic and NeXTHelpLink. When one of the cells is selected the LessonControl object requires to know which file to load or action to perform. To identify the selected cell the **trackMouseDown:inRect:ofView:** method had to be overridden. In this method the method **setCellSelection:** is sent to LessonControl so that if the cell is selected LessonControl can query it for further information. The methods **getTapeFileName:** to TapeButtonCell, and **getDocFileName:** and **linkMarkerName:** to Document link cell provide LessonControl to either run the required tape, or load the linked document depending on the cell selected.

The **init** message received by DevBrowserControl, as shown in Figure B.1, page 65, sends an **init:** message to the superclass BrowserControl, reads the list from the file with the local **readListFromFile** method, and loads the "Browser.nib" file through the superclass method **loadNibFile:**. The **readListFromFile** method invokes the superclass’s **readListFromFile** method. This list is then adjusted to reflect the "NULL" entries seen in the browser. This is necessary so that further entries can be added down the hierarchy. The process involves locating the leaf nodes in the list and then attaching a "NULL" node to them which then becomes the leaf node. Each time a node is added to the list the function **insertionAdjust()** is used to correctly insert the node at a particular index of the list and adjust the variable in the nodes that make up the list. The function **getNullInsertion** index determines the insertion index for the "NULL" node.

The **doUpdateLinks** method loads "node.nib". **setCurrentInputValues** then sets the title and file link field to reflect the currently selected cell in the browser. The **doMatrixInput:** is the target for buttons in the update links panel. Depending on the button selected the methods **processDelete**, **processAdd**, and **processSet** would be
invoked. **processDelete** uses the local **doDeletion** method to delete the branch from for the selected index and then adjust the list to reflect the change. **processAdd** would add the new element to the list and adjust the list. **processSet** changes the attributes of the selected node to the strings entered in the panel. The functions **deletionAdjust()**, **insertionAdjust()**, **getNULLInsertionIndex()**, and **removeFromList()** manipulate the list for addition and deletion of nodes.