HYPERMENU:

INTEGRATING FUNCTION ACCESS AND WORK OBJECT REPRESENTATION IN GRAPHICS APPLICATIONS

by

Albert Chan

B.S.E.E., University of Texas at Austin, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in the School
of
Engineering Science

APPROVAL

Name:	Albert Chan
Degree:	Master of Applied Science
Title of thesis:	HYPERMENU: INTEGRATING FUNCTION ACCESS AND WORK OBJECT REPRESENTATION IN GRAPHICS APPLICATIONS
Examining Committee:	Dr. Paul Ho Associate Professor, Engineering Science, SFU Chairman
	Dr. Tom Calvert Professor, Engineering Science, SFU Supervisor
	Dr. John Dill Professor, Engineering Science, SFU Senior Supervisor
	Dr. John Jones Associate Professor, Engineering Science, SFU Examiner
Date Approved:	November 15, 1996

ABSTRACT

Traditionally, interactive computer programs consist of a set of functions to be applied to an underlying "work object". The user interfaces of these programs thus consist of mechanisms for accessing functions and some kind of visual representation of the underlying work object. Indeed, many user interfaces function like "menu applications". Unfortunately, traditional menu applications quickly become limited and difficult to use as the number of functions and complexity of the work object increases.

To facilitate interaction and preserve the user's orientation, this thesis proposes a new menu mechanism, called **hypermenu**, designed for hierarchical work objects. It uses a hierarchy and a context management scheme. Each component in the hierarchy is a *combined representation* of function access mechanism and a corresponding component in the work object representation. The context management scheme uses visualization techniques to organize the on-screen layout and appearance of all the components in the hierarchy. Ideally, our hypermenu approach turns an application into its own menu, and lets the user see details in context and access application functions via direct manipulation. We have implemented two hypermenu applications for evaluation, one for a computer-aided-design tool based on Group Technology, the other for the user interface of a telecommunication network testing system.

To my church and family

ACKNOWLEDGMENTS

I am grateful to my Lord and personal saviour Jesus Christ for providing excellent faculties and facilities at Simon Fraser University.

I would especially like to thank my supervisor Dr. John C. Dill for providing me the opportunity to work on this challenging and interesting project, for his many valuable advice and suggestions, and tremendous help in revising this thesis.

I would like to thank Frank Henigman for his kind help, and Mark Leonard for allowing me to include material related to their telecommunication network test system in this thesis. Also, many thanks to the graduate secretary, Brigitte Rabold, for her kind help.

I am very grateful to my committee members, Dr. Tom Calvert , Dr. John Jones for their assistance and help.

CONTENTS

LIST OF FIGURES

VARIABLES

S	Scale factor of a node.
L_f	Final size of a node.
L_i	Initial size of a node.
X_{req}	Total amount of space requested by a cluster in the X direction.
x_i	Normal length of the i^{th} interval.
s_i	the scale factor of the i^{th} interval.
L	The length of an intermediate node.
k	the k^{th} step during zooming.
n	the total number of steps during zooming.

CHAPTER 1

INTRODUCTION

Traditionally, interactive computer programs consist of a set of functions to be applied to an underlying "work object". User interfaces of these programs thus consist of mechanisms for accessing functions and some kind of visual representation of the underlying work object. For example, modern word processors provide menus for the user to apply application functions to their document representations. A portion of these programs contains hierarchically organized work objects or hierarchical systems.

As technology advances, hierarchical systems, from automated teller machines to complex computer aided design packages, have become a part of our lives. Consider for example a microprocessor consisting of cache, arithmetic logic unit, and a floating point unit. The arithmetic logic unit is made up of flip flops; flip flops in turn are made up of logic gates, and each logic gate is also made up of transistors, resistors, and capacitors. Hierarchical organization facilitates management of complex systems and is widely used nowadays. Equally significant, the user interface of hierarchical systems directly affects the efficiency of user interactions.

Figure 1.1 illustrates the user interface of a modern hierarchical system taken from a telecommunication network testing system. The user interface comprises over twenty dialog boxes, where each dialog box represents a functional part of the system. In this example, the user interface is essentially a menu mechanism providing access to different parts of the underlying system. Indeed, user interfaces of many hierarchical systems function like "menu applications".

Though the complexity level of hierarchical systems has increased rapidly, screen space has not. As a result, it is important to develop user interfaces meeting the following criteria:

1. provides a comprehensive representation of the work object and preserves the user's orientation.

2. facilitates interactions and navigation within the application.

In this thesis, we focus on developing a new menu mechanism that can be used effectively in hierarchical systems.

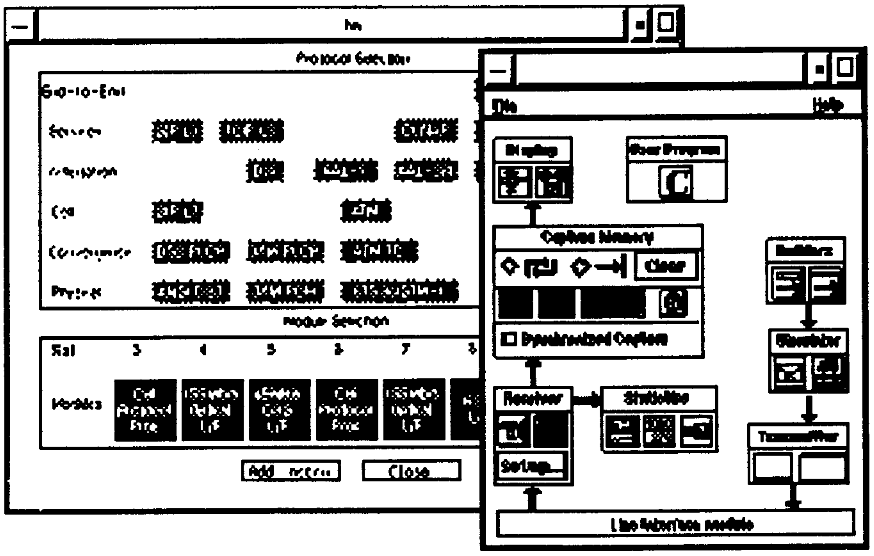


Figure 1.1: Typical user interface for hierarchical systems.

Problem Statement: Often, the primary job of a hierarchical system is to allow management, configuration and visualization of a work object. Traditional user interfaces require users to remember each configuration step and the purpose of every window as well as its position. Such a mental load violates the well-known human memory capacity limitation of no more than seven plus or minus two "chunks" [Miller 56] and therefore handicaps navigation. For example, random placement of windows indicates no intuitive relationship among the component objects they represent and becomes disorienting to the users. Overlapping windows and dialog boxes induce significant user interface overhead and hinder user interaction. Furthermore, it is difficult for the user to form a full picture of the work object by mentally integrating dialog boxes and windows.

On the other hand, a menu system can at best present a subset of information in detail to the user at any given time. Very often, individual menu items are organized into a one-dimensional hierarchical menu system [Callahan 88], i.e. single linear list, because of insufficient screen space. Organized menu hierarchies have been proven to improve user performance [Miller 81] [Barnard 77]. Figure 1.2 shows a menu mechanism widely used

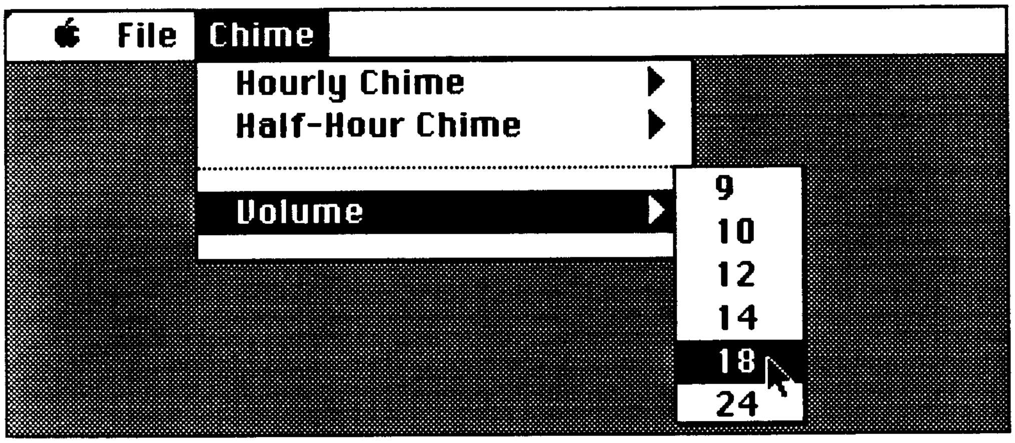


Figure 1.2: Traditional menu systems such as nested pull-down menus are popular.

by today's applications and operating systems. Unfortunately, as the number of menu items grows, it will be much more difficult for the user to locate and access individual menu items even with hierarchical menu systems. Instead of aiding the user, complex menu systems may prevent the user from accessing functions quickly and accurately [Kiger 84]. Most traditional menu systems suffer from the following weaknesses:

1. It is separate from the work object. This requires the user to track the menu system separately and thus increases the cognitive load of the user. One must access functions through a separate menu mechanism that links the functions and the work object representations.

2. Very often, selecting a menu item brings up a separate dialog box that provides the user with no immediate hints on its relationship with the menu item or the application.

3. The menu system gives the user no overview of its organization. Providing the global context of the menu system is important when there is a large number of menu items.

4. A separate menu system consumes screen space, processor cycles to display, and is very likely to obscure the work object and thus become disruptive [Kurtenbach 93].

5. It is impossible for the user to simultaneously look at and compare two menu items from different substructures of the menu hierarchy.

6. Traditional menu systems are limited to represent only functions, not other objects in the system.

In modern systems, menu mechanisms have become even more vital to the speed and effectiveness with which users navigate and access system functions. Many menu mechanism variations have been developed to facilitate function access. Improving menu mechanisms continues to be a focus for many researchers, and new strategies continue to be investigated [Landauer 87] [Hopkins 91].

Based on previous research in the IGI project [Schaffer 93, Dill 94, Bartram 95], this thesis develops a new menu mechanism, called a **hypermenu,** which utilizes the continuous zoom [Dill 94], to facilitate accessing functions and work objects in large hierarchical systems.

Our hypermenu approach integrates the menu mechanism with the work object representation, and uses visualization techniques to let the user see details in context. In addition, the hypermenu approach allows navigation and function access via direct manipulation. This new mechanism has been developed for general hierarchical systems and may be applied to a range of application domains.

Thesis Organization: This thesis is organized as follows: Chapter 2 reviews current technologies on menu systems and hierarchical information visualizations. Chapter 3 describes general hypermenu concepts. Chapter 4 describes two illustrative applications to which our hypermenu approach was applied: a Group Technology design aid, and a telecommunication network test system. Chapter 5 analyzes our hypermenu approach and describes its strengths and weaknesses, and Chapter 6 summarizes our hypermenu approach and discuss further research directions.

CHAPTER 2

LITERATURE REVIEW

The main reason for the success of menus is allowing users to work with recognition memory, where visual images (text or iconic menu items) are associated with already-familiar words and meanings [Foley 90]. Generally speaking, approaches to menus can be categorized into four classes: (1) explicit menu; (2) on-demand menu; (3) see-through menu; and (4) embedded menu mechanisms (see Figure 2.1). This taxonomy emphasizes the embedded menu mechanism category, which is closely related to our new mechanism. This chapter first describes and reviews these menu systems. Then, we review some recent research on hierarchical visualization and navigation to provide the necessary background for our approach. Lastly, we briefly introduce our new mechanism which solves some problems in previous methods and may be applied to a variety of applications.

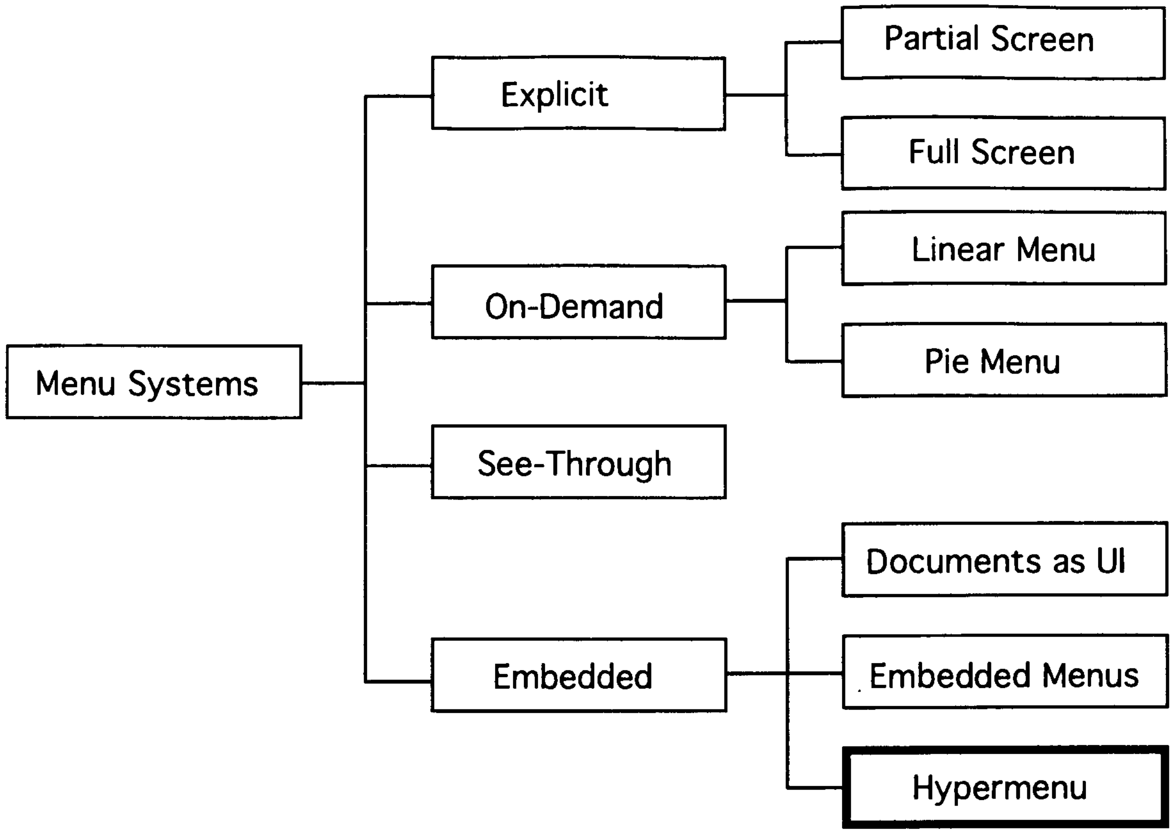


Figure 2.1: A taxonomy of menu systems.

2.1 Explicit Menu Systems

The inherent limitations of human memory combined with the current information explosion increasingly force users to rely on their abilities to navigate to an item rather than memorizing its exact location [Barnard 77]. For this reason, menu systems have been developed to facilitate locating individual items among a very large set of functions. Today, most menu systems display available functions as input choices and are separate and distinct from the work object representation. Explicit menu systems [Koved 86] occupy fixed locations and use fixed amount of screen space on the display. They have the advantage of being visible to the user at all times. Explicit menu systems usually supply an explicitly enumerated list of items from which the user selects. In the following subsections, we describe two variations: partial screen approach, and full screen approach.

2.1.1 Partial Screen Approach

This approach is widely used today, with variations including toolbars, palettes in graphics applications, and tear-off menus. They usually accommodate frequently accessed functions. Many explicit menus are only one level deep and may accommodate a limited number of items to reduce effort spent on tracking. To accommodate a large number of menu items, menu designers have employed hierarchical schemes. Autocad [Autodesk 89] is such an example. Nevertheless, modern menu design attempts to reduce the number of levels in the hierarchy to facilitate tracking and navigation.

Figure 2.2 shows a screen shot of Autocad. Residing on one side of the screen, the

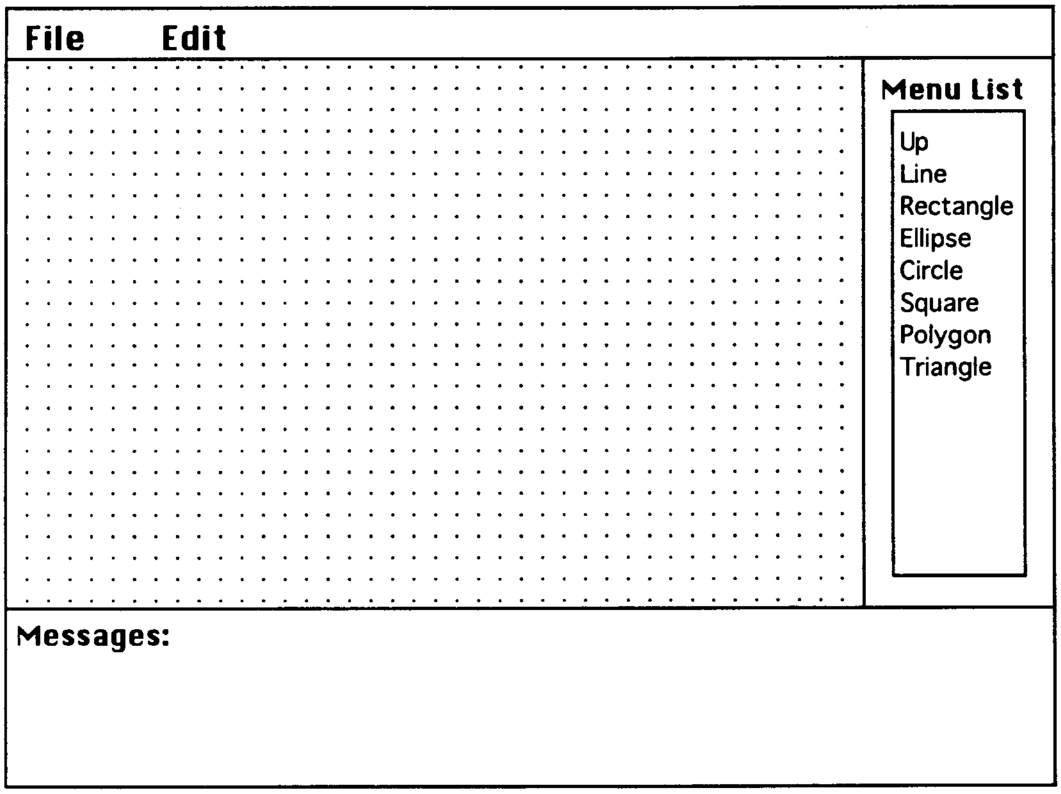


Figure 2.2: An Autocad application. The work object is displayed on the top-left display area, and the menu system is located on the top-right display area [Autodesk 91].

menu can accommodate only a few items at a time. With this approach, the work object and the menu system each have their own non-overlapping display areas. A menu item is represented by a text string in the menu area. Because of the limited menu display area, menu items belonging to the same level in the menu hierarchy are organized into groups. The user accesses functions by traversing the menu hierarchy. Selecting the top-most menu item brings the user back to the top level of the menu hierarchy. Selecting other menu items either executes the corresponding function or displays menu items one level down in the hierarchy. Finally, the user may switch between menu groups in the same level.

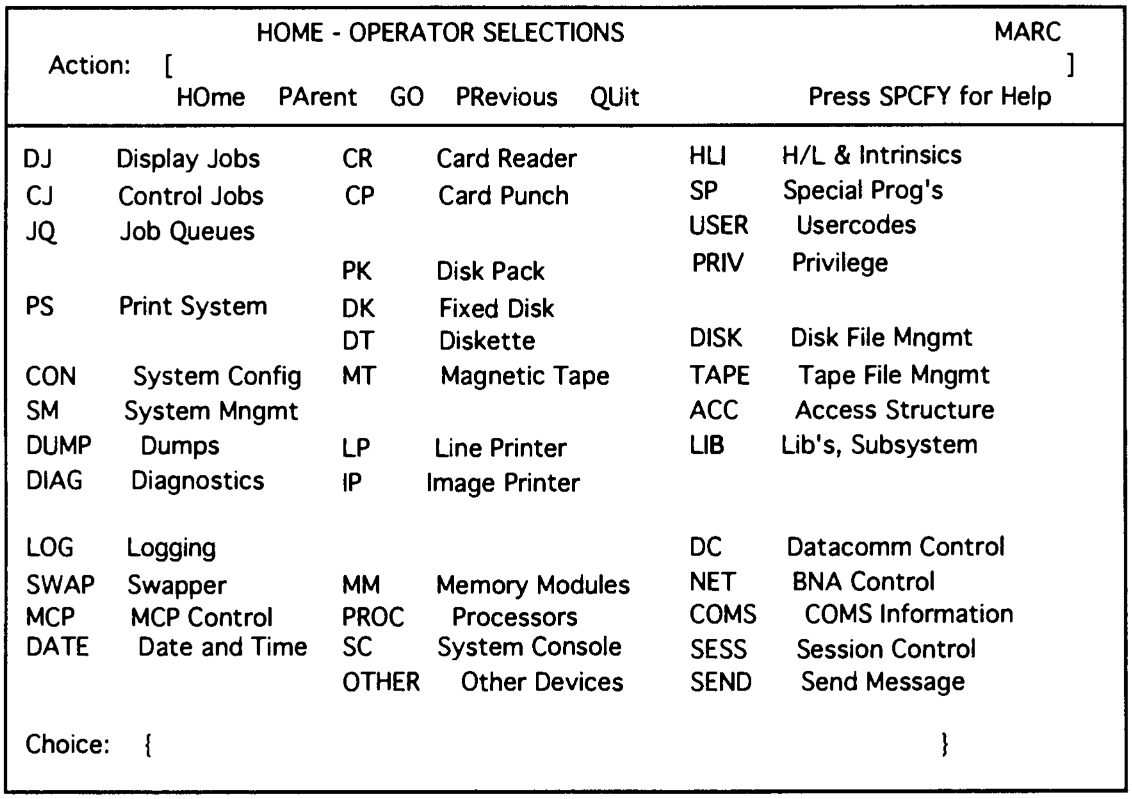
The major advantage of the partial screen approach is allowing relatively quick access to menu items. In addition, the hierarchical version allows a virtually unlimited number of menu items. Also, the user always has a working set of functions on screen. However, the main disadvantage of this approach is the permanent consumption of screen space. For the hierarchical approach found in Autocad, there are two more disadvantages. First, it can be disorienting to the user because there is no indication of current path. Second, it can be distractive and confusing to the user because of the additional memory load required to track and navigate a menu system with many hierarchical levels. Users tend to lose track of the target they seek. As one moves through deeper levels of a hierarchy, the ability and time taken to make a correct selection depends on remembering the previous selection [Snowberry 83].

Short-term memory capacity for young adults averages seven plus or minus two digits [Miller 56]. If each choice in a hierarchy requires a unit of short-term memory, six levels of hierarchy is pushing a user's short-term memory capacity to the limits [Snowberry 83]. Our new menu mechanism, in contrast, takes context into consideration to reduce the user's tracking effort and attempts to increase the number of levels in the hierarchy without inducing disorientation.

2.1.2 Full Screen Approach

With this approach, the menu system takes up the entire screen. An example is the Menu-Assisted Resource Control System (MARC) designed to help both new and experienced users access any of the features provided by an operating system [Tullis 85]. In designing MARC, researchers attempted to answer two questions: (1) how to determine the user's view of the relationships among the large number of functions in an operating system; and (2) how to reflect those relationships in a menu hierarchy.

MARC is a very large text-based hierarchical menu system in which available system commands are sorted into logically related groups, and then inserted into the menu hierarchy. The main menu of MARC is shown in Figure 2.3. MARC is a two-level deep multi-column menu system. All functions available from the operating system can be



```
                HOME - OPERATOR SELECTIONS                                 MARC
  Action:  [                                                                   ]
            HOme   PArent   GO   PRevious   QUit           Press SPCFY for Help

DJ     Display Jobs      CR     Card Reader        HLI    H/L & Intrinsics
CJ     Control Jobs      CP     Card Punch         SP     Special Prog's
JQ     Job Queues                                  USER   Usercodes
                         PK     Disk Pack          PRIV   Privilege
PS     Print System      DK     Fixed Disk
                         DT     Diskette           DISK   Disk File Mngmt
CON    System Config     MT     Magnetic Tape      TAPE   Tape File Mngmt
SM     System Mngmt                                ACC    Access Structure
DUMP   Dumps             LP     Line Printer       LIB    Lib's, Subsystem
DIAG   Diagnostics       IP     Image Printer

LOG    Logging                                     DC     Datacomm Control
SWAP   Swapper           MM     Memory Modules     NET    BNA Control
MCP    MCP Control       PROC   Processors         COMS   COMS Information
DATE   Date and Time     SC     System Console     SESS   Session Control
                         OTHER  Other Devices      SEND   Send Message

Choice:  {                                                        }
```

Figure 2.3: The main menu for MARC [Tullis 85].

accomplished by viewing only two menus. To navigate within MARC, the user types the index of a selectable item. Then, a new screen of menus replaces the previous menu.

Other full screen applications such as rn, a UNIX news reading program, require the user to switch back and forth between the work object and a full screen menu system.

A full screen menu system has three main advantages over command-line driven interface. First, it improves user performance by allowing the user to type an index rather than the entire command. Second, it helps the user to make fewer mistakes by providing available options on screen. Third, it reduces users' memory load by constantly reminding the user what is available to him or her. However, it also shares some disadvantages with the partial screen approach. First, it can be disorienting to the user because there is no indication of current path. Second, a deep menu system is difficult to navigate because of limitations on human memory [Snowberry 83]. Third, the full screen approach becomes very disruptive by forcing the user to switch back and forth between the work object and menu system.

2.2 On-Demand Menu Systems

Unfortunately, explicit menu systems require their own display area and must contend with the work object for screen space. To minimize screen space usage, on-demand menu systems such as pull-down menus from Apple Macintosh, based primarily on pioneering work at Xerox in the mid 1970's [Foley 90], have been developed, and much research effort has been spent on menu organization, format and physical layout [Perlman 84] [Callahan 88] [Landauer 85] [Snowberry 83] [McDonald 83] [Barnard 77]. In the following subsections, we will describe two different menu layouts: (1) rectilinear layout, which is widely used, and (2) circular layout, which is still being investigated.

In general, on-demand menu systems appear temporarily when activated by the user. They have been designed to occupy a minimum amount of screen space when not activated.

On-demand rectilinear menus have many advantages: (a) they conserve screen space, (b) they reduce user memory load, (c) they are strictly organized; thus predictable to users, and (d) they may easily be applied to most applications. However, they also have drawbacks. First, on-demand menus obscure the application object and may become disruptive [Kurtenbach 93]. Second, it takes processor time to display the menu and hence reduces performance [Kurtenbach 93]. Third, the effectiveness of the menu format depends highly on the intended user [Barnard 77]. Finally, seek time increases proportionally to the distance of the target from the initial cursor location [Callahan 88] and list length also has a linear effect on the time a user takes to find an item [Perlman 85]. Seek time is the time it takes to reach the target after initiating a search for it.

In presenting a list of choices to the user, most computer systems use a rectilinear format because of available hardware and software limitations. With a rectilinear format, menu items are listed horizontally or vertically, sometimes with a keyboard equivalent for each item. Hierarchical schemes have also been employed, but most menu systems are essentially one dimensional, i.e. simple linear lists of items. The essential idea of hierarchical scheme is allowing the user to bring up a sub menu list from an existing menu item. The look-aside or cascade menu, (Figure 1.2), is an example of such a scheme. Most on-demand menu systems are activated from mouse actions in two formats: pull-down or pop-up. As mentioned before, the menu system itself appears to be separated from the work object representation and activated by users. To navigate within the menu system, the user only needs to activate the menu system and select the desired item. In case of a hierarchical or cascading menu system, the user may recursively select menu items that eventually lead to the desired item.

A variant of the rectilinear approach is the so called "pie menu" (Figure 2.4) whose goal is to allow quicker menu selection and to minimize mistakes made during menu selection [Hopkins 91]. Items in the pie menu are placed at equal radial distances around the circumference of a circle. Users typically accesses pie menus via the traditional pop-up method. However, the starting cursor position is at the center of the pie menu instead of being at the menu title or the first item in the traditional pull-down menus. Since items are placed at equal radial distances from the center of the menu, the user only needs to move the cursor by the same amount in different directions to select. Pie menus support hierarchical schemes with menu item selection bringing up another pie menu centered at that menu item. Thus, the user navigates the menu system by recursively selecting and activating desired items.

Pie menus offer advantages such as decreased selection distance and increased target size; therefore, it keeps the seek time fairly constant. However, pie menus also have some disadvantages in addition to those shared by rectilinear menus. Pie menus consume more screen space and become polynomially larger than rectilinear menus with increased item size and number of items [Callahan 88].

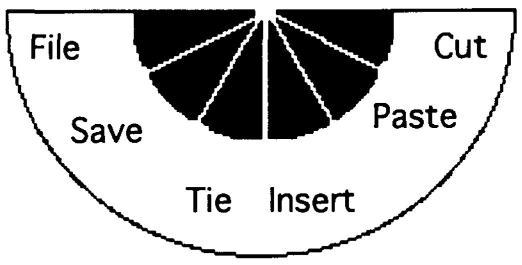


Figure 2.4: An example of pie menus [Callahan 88].

2.3 See-Through Menu System

A more recent innovation uses semi-transparent, often icon-like, menus that appear as though on a transparent sheet of glass, between an application and a traditional cursor [Bier 93]. This method can provide context-dependent feedback and the ability to view details and context simultaneously. Bier's Toolglass and Magic Lenses [Bier 93] is an example of a See-Through interface. It is intended to be used with graphics-oriented applications such as image editing. Their approach makes use of semi-transparent interactive tools, called Toolglass widgets, that sit between the application and a cursor. These widgets can provide a customized view of the underlying application object using viewing filters called Magic Lenses. Each lens is a screen region with an embedded function or command such as "magnify" and "apply color". The user positions a Toolglass sheet over desired objects and then points and clicks through the widgets and lenses. These tools create spatial modes that can replace temporal modes in user interface systems [Bier 93].

Figure 2.5 shows a set of simple widgets called click-through buttons, which can be used to change the color of objects below them. The user first positions the button on top of the object to be colored and then clicks "through" the button.

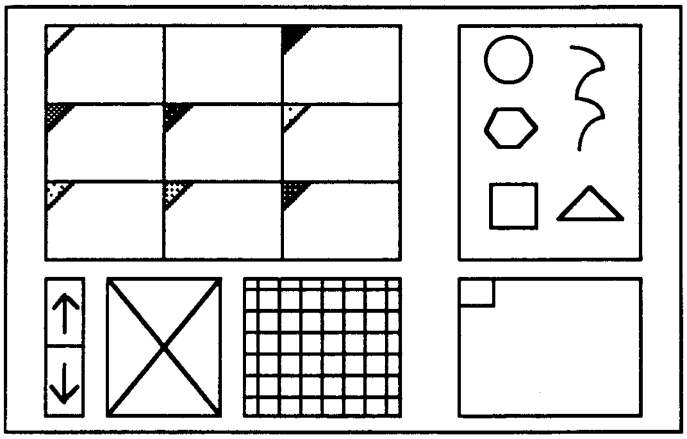


Figure 2.5: A Toolglass sheet of widgets. Clockwise from upper left: color palette, shape palette, clipboard, grid, delete button, and buttons that navigate to additional widgets [Bier 93].

There are three approaches to support navigation within the See-Through menu system. The first approach is to put all widgets on a single sheet that can be navigated by scrolling. The second approach is to employ a hierarchical scheme that allows a master toolglass sheet to generate other sheets. The third approach is to allow a single toolglass sheet to display different sets of widgets at different times. For example, the user can click on a special widget to navigate to another set of widgets.

See-Through interface brings four major advantages. First, it consumes little screen space because it can be dragged off the screen any time. Second, using a spatial mode to access functions or commands avoids confusion brought by temporal modes. Third, providing instant feedback on button functions further minimizes memorization of commands. Finally, all alternative representations of the underlying application are limited within lenses when viewed through them; thus, the See-Through interface preserves the work context with minimal disruption.

Unfortunately, See-Through Menu and the work object representation are still separate entities in the user's mind and requires extra effort on tracking and manipulation.

2.4 Embedded Menu Systems

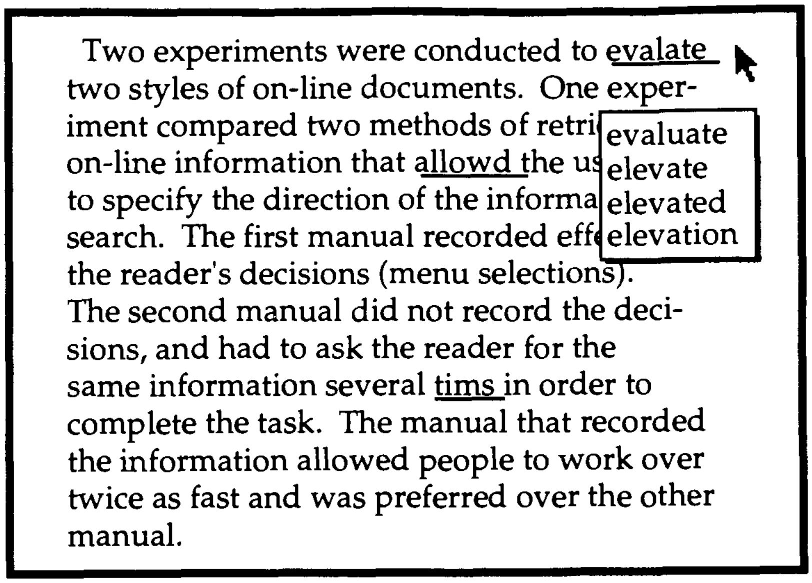
In early 1980's, Koved coined the term "embedded menus" and applied it to text-based applications in [Koved 86]. Other researchers later extended the concept and applied to other application domains. Despite the differences in these embedded menu systems, they share some common goals. First, designers of embedded menu systems attempt to embed the menu system into the application or document object. Doing so reduces the need for a separate menu system. Without a separate menu system, distraction can be reduced. Second, designers of embedded menu systems attempt to use the application or document object itself to provide global context for the menu system. Doing so reduces the chance of disorienting the user during navigation. Third, embedded menu systems allow users to access application functions via direct manipulation. In the following subsections, we describe two different embedded menu applications: *Embedded Menus*, designed for text editors and database retrieval systems, and *Documents as User Interfaces*, designed for text and graphics editors.

2.4.1 Embedded Menus

[Koved 86] describes Embedded Menus, targeted at text-based applications such as text editors, database and on-line manual systems. Embedded Menus allow menu items to be embedded within the information displayed on the screen. This information thus provides context for the menu items. In embedded menu, highlighted or underlined words or phrases within the text become the menu items and are selectable using any pointing device.

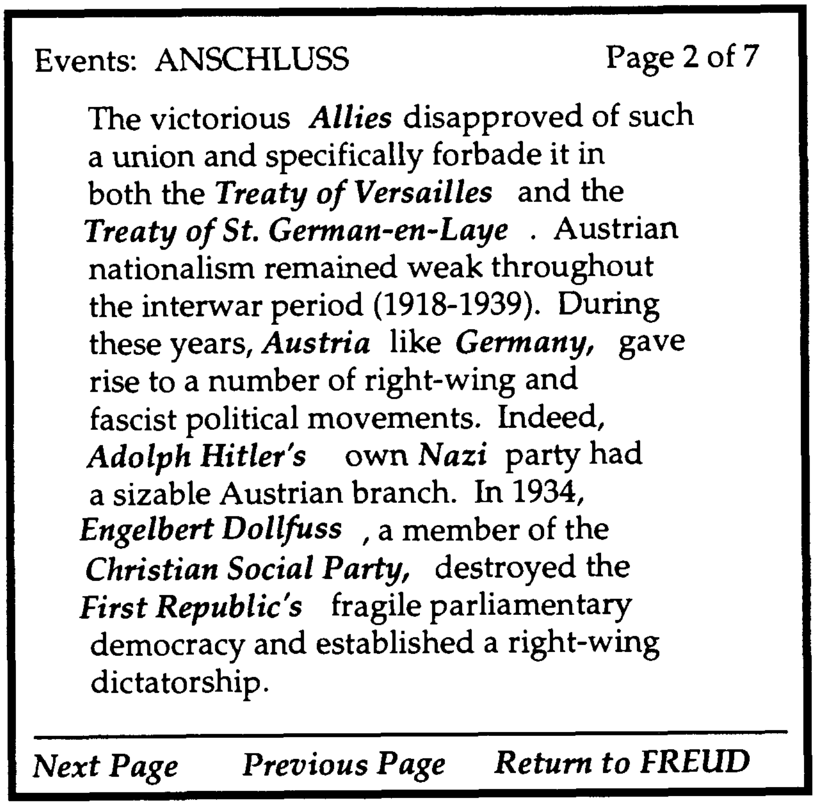
Figure 2.6(a) shows a screen shot of a spell checker used as one of the examples in [Koved 86]. Instead of extracting and displaying the list of incorrect spellings in an explicit menu, the spell checker underlines them directly in the document. With a pointing device, the user can select them individually which brings up a closely positioned explicit menu suggesting correct spellings. The user can then select the correct spelling from the explicit menu.

Embedded Menus have been applied to history databases (Figure 2.6(b)) and on-line manual systems [Koved 86]. Selecting a menu item retrieves the corresponding article. Today's hypertext links [Nielsen 90], where menu items were embedded in the on-line documents but distinguished in appearance by being bolded or italicized, closely resemble Embedded Menus.



Two experiments were conducted to evalate two styles of on-line documents. One experiment compared two methods of retri on-line information that allowd the us to specify the direction of the informa search. The first manual recorded eff the reader's decisions (menu selections). The second manual did not record the decisions, and had to ask the reader for the same information several tims in order to complete the task. The manual that recorded the information allowed people to work over twice as fast and was preferred over the other manual.

(a)



Events: ANSCHLUSS Page 2 of 7

The victorious ***Allies*** disapproved of such a union and specifically forbade it in both the ***Treaty of Versailles*** and the ***Treaty of St. German-en-Laye*** . Austrian nationalism remained weak throughout the interwar period (1918-1939). During these years, ***Austria*** like ***Germany,*** gave rise to a number of right-wing and fascist political movements. Indeed, ***Adolph Hitler's*** own ***Nazi*** party had a sizable Austrian branch. In 1934, ***Engelbert Dollfuss*** , a member of the ***Christian Social Party,*** destroyed the ***First Republic's*** fragile parliamentary democracy and established a right-wing dictatorship.

(b)

Figure 2.6: Examples of Embedded Menus. (a) Misspelled words are highlighted in the Embedded Menus. (b) Example from Interactive Encyclopedia System. Bold italic words are buttons of Embedded Menus [Koved 86].

There are three advantages to the Embedded Menus method. First, it allows the user to navigate and access application functions via direct manipulation. Second, it provides context for each menu item with the surrounding document. Third, it conserves screen space by using a combination of pop-up menus and highlighted items. However, when applied to on-line document systems as shown in Figure 2.6(b), Embedded Menus provide no clue on orientation. For example, there is no indication of user location in the system, how one arrived at the current page, nor relationships between previous pages and the current page. As a result, it becomes disorienting to the users.

2.4.2 Documents As Menus

In [Bier 90], the authors describe a framework that embeds menu systems into the work object (mainly documents) representation. Each document in turn provides a context for its embedded menu items. Later, [Bier 91] described their approach, called EmbeddedButtons in more detail. When designing EmbeddedButtons, the authors extended the embedded menus concept to support both text and graphics editors. In addition, the authors attempted to provide user interface layout tools along with active documents. An active document contains both information and embedded buttons which the user selects to execute the corresponding functions.

EmbeddedButtons is an architecture that allows arbitrary document objects to become buttons. Documents can be linked to an application to serve as control panels or menu palettes. Furthermore, EmbeddedButtons also support pop-up menus, multi-state buttons and radio buttons.

When applied to graphic applications, EmbeddedButtons allows the user to build a sheet of embedded buttons with a graphics editor and use the sheet as a menu palette. Figure 2.7 shows a palette of embedded buttons for a graphics editor. A text or graphics

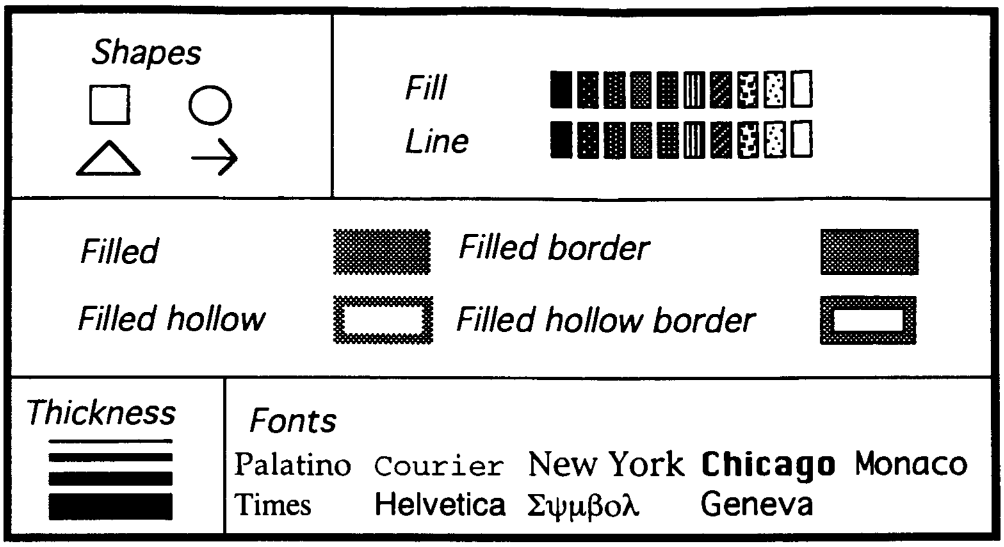


Figure 2.7: EmbeddedButtons example: A menu palette that is itself a picture and is created with the same graphics editor [Bier 91].

editor is said to be in active mode when all mouse motion and mouse button events are delivered to EmbeddedButtons; otherwise, all such events are treated as normal editor operations. When a button is in active mode, it is enclosed by a rectangular box.

EmbeddedButtons has three major advantages. First, it reduces the chance of disorientation by providing context for menu items using the application or document object. Second, it allows direct manipulation on the underlying application through interacting with the embedded menus and buttons. Third, embedding buttons and menus may be used to conserve screen space. However, when applied to graphics editors, a palette of embedded buttons is still separate from the work object representation and inherits the disadvantages of explicit menu systems.

2.5 Hierarchical Information Visualization

Since we can consider a hierarchical menu system itself to be an information system, it is relevant to review some recent research on visualizing hierarchical information systems.

[Card 91] described some observations on human information processing. Humans attempt to simplify voluminous information through aggregation, abstraction and selective omission, which facilitates pattern recognition. For this reason, human beings have used hierarchical schemes in structuring complex systems to facilitate information absorption.

[Furnas 86] described a fisheye technique to display hierarchical structures. It was designed to provide a balance of local detail and global context. This technique can show an area of interest, called the focus, in great detail while still showing remote regions in successively less detail. Early prototypes were designed to be used on text-based applications. The major drawback to this method is the fact that components are either present or absent. It is impossible to vary size and level of detail.

In 1993, [Schaffer 93] described the Variable Zoom which evolved from Furnas' fisheye technique. The Variable Zoom was designed to display large hierarchical structures. Zooming is the fundamental interaction technique. The Variable Zoom supports multiple foci and preserves the global context at any given time. However, zooming induces size changes in a single step, and destroys visual continuity.

Shortly after, [Dill 94] described the Continuous Zoom which is an extension to the Variable Zoom. The Continuous Zoom offers fine control over the size of an object during zooming, and preserves visual continuity. Unfortunately, the user also has to track the object size constantly during zooming. Consequently, zooming requires a certain amount of cognitive effort from the user.

Later, Graphical Fisheye Views [Sarkar 94] is another extension of Furnas' work to visualize graph structures. The size and position of an object varies based on its distance from the focus. It is an improvement on Furnas' original fisheye technique, but does not support hierarchical abstraction.

More recently, [Lamping 95] described a new focus+context (fisheye) scheme for visualizing and manipulating large hierarchies by laying out an hierarchy uniformly on a hyperbolic surface and then mapping it onto a circular display region. The center of the circular region becomes the focus. Because in a hyperbolic surface, parallel lines diverge from one another, there is exponentially more space with increasing circumference of the circular display region. With this scheme, hierarchies that tend to expand exponentially with depth can be laid out easily on the display. Unfortunately, this method also suffers from two drawbacks. First, it supports only one focus. Second, a circular display format also makes it difficult to use screen space efficiently.

In 1991, Perspective Wall [Mackinlay 91] was developed at Xerox PARC corporation to visualize linear information. It divides a 2D layout into three sections and then folds the two side sections away from the viewer, leaving the center section for detail and the rest for context. The user may move any item to the central section for a detailed view. However, the Perspective Wall supports only one focus at a time, which becomes a limitation during investigation of multiple areas of interest. The shape of Perspective Wall also makes it difficult to use screen space efficiently. Finally, it is difficult to display hierarchical structures with Perspective Wall.

At the same place and approximately the same time, the Cone Tree [Robertson 91] was also developed to display hierarchical information in three dimensions with coloring and perspective distortion (Figure 2.8). The hierarchy can be placed like an inverted tree. Each component in the hierarchy is rendered transparent so that it does not obscure other components. However, Cone Trees are not as effective in displaying balanced hierarchical structures as unbalanced hierarchical structures. Moreover, rendering complex three-dimensional structures in real time may be time-consuming.

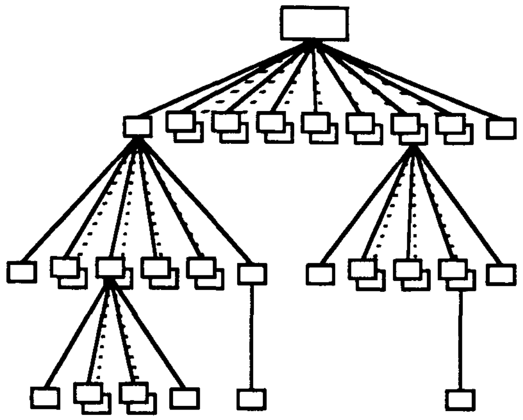


Figure 2.8: Cone Tree [Robertson 91].

To display hierarchical file structures, [Johnson 91] developed a new method called a tree-map. This method works by representing each node in the hierarchical file structure as a rectangle. Nodes representing directories are divided into 'sub rectangles', one for each directory member. At successive levels in the hierarchy, the direction of the subdivision alternates between horizontal and vertical. The lack of support for emphasizing areas of interest becomes the major drawback of this approach.

Similarly, Pad++ [Bederson 94] was designed to facilitate visualization and navigation within large hierarchical information spaces. Zooming is a fundamental interaction technique for Pad++. In addition to size difference, Pad++ also employs semantic zooming and animation. Semantic zooming means displaying a different abstract representation of the same object at different sizes. Instead of simply displaying a scaled down version of an object, a simplified abstraction is displayed. In addition, animation is used to preserve visual continuity during zooming. To create a work context, Pad++ rates information in a way to make the most highly rated information the largest and most obvious, while placing less important information nearby and smaller. The viewer double clicks on an object of interest to see further details. Viewers can recursively zoom in and zoom out on any Pad++ work objects. However, the global context may not be entirely available to the viewer at all times.

2.6 Hypermenu: A New Mechanism

The approaches described in previous sections have met with varying degrees of success, but suffer drawbacks as noted. We seek a new menu mechanism to facilitate navigation and function access within hierarchical systems.

This thesis develops a new menu mechanism, which we called the **hypermenu** method, to optimize interactions between user and application. It addresses some of the problems with existing methods by providing support in the following areas:

- hierarchical abstraction and management of the work object;
- tightly integrates menu mechanism with work object and accurately reflects logical relationships of all components;
- direct manipulation of system components rather than using a separate menu mechanism;
- preserves global context and user orientation at all times;
- displays size-dependent representation;
- multiple foci visualization for the multiple areas of interest.

In the next chapter, we describe our hypermenu method in more detail. We then evaluate the achievements and weaknesses of the hypermenu method.

CHAPTER 3

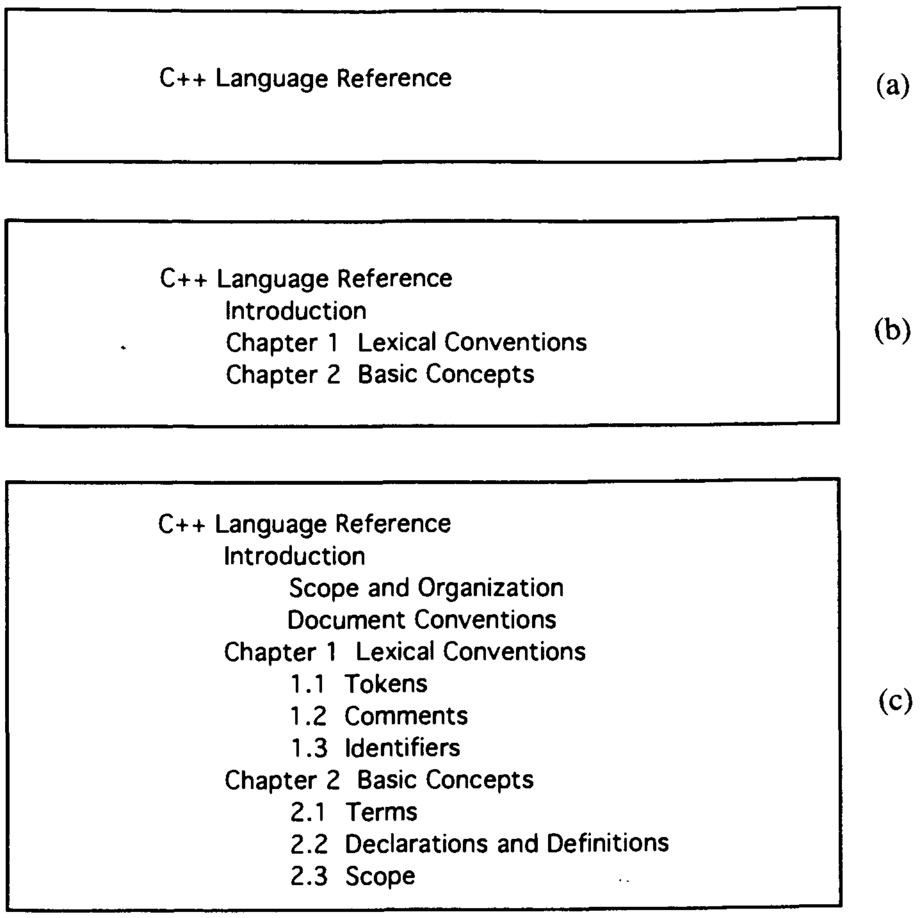
THE HYPERMENU CONCEPT

In this chapter, we first discuss the concepts underlying our hypermenu interface method. Following this we describe navigation and function access issues, and hypermenu construction.

3.1 Hypermenu Concepts

3.1.1 Hierarchical Organization

Hierarchical organization is a common practice in managing complex systems. For example, books and manuals are organized hierarchically using titles, chapters and sections as shown in Figure 3.1. In the figure, entities belonging to the same abstract level are indented by the same amount. Hierarchical abstractions allow omission of lower level details that are not of interest, and greatly simplify the representation of the entire object.



```
C++ Language Reference                                    (a)
```

```
C++ Language Reference
      Introduction
      Chapter 1  Lexical Conventions                      (b)
      Chapter 2  Basic Concepts
```

```
C++ Language Reference
      Introduction
            Scope and Organization
            Document Conventions
      Chapter 1  Lexical Conventions
            1.1  Tokens
            1.2  Comments                                 (c)
            1.3  Identifiers
      Chapter 2  Basic Concepts
            2.1  Terms
            2.2  Declarations and Definitions
            2.3  Scope
```

Figure 3.1: Hierarchical organization and representation for an on-line reference manual, showing three different levels of abstractions. (a) Highest abstract level: Book title. (b) Chapter level. (c) Section level.

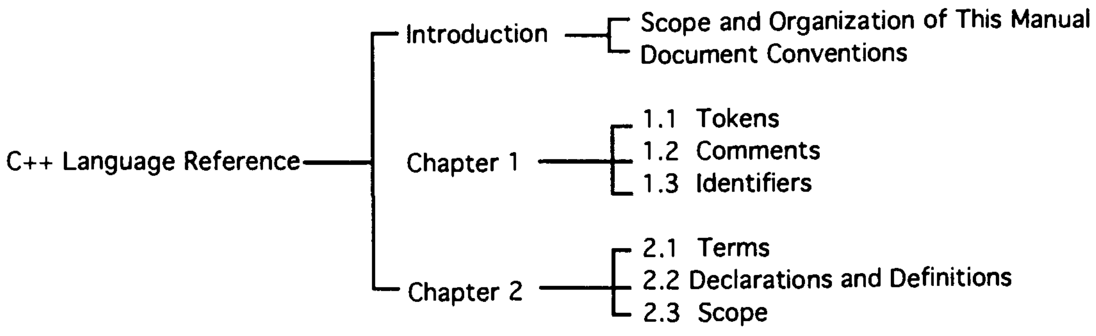


Figure 3.2: Tree structure for an on-line reference manual.

Often, we use tree structures to express hierarchical relationships. Figure 3.2 is the tree structure of the on-line manual shown in Figure 3.1. The ability to organize a system into a hierarchical structure is essential to constructing a hypermenu.

3.1.2 Definitions And Traditional Menu Systems

Generally, an application program consists of some work object representations and a set of functions. The work object representation is an on-screen representation of an underlying work object; e.g., a text document in a text editor. Availability of the application functions is governed by a set of rules in the application program, called work object constraints. For example, a work object constraint in a word processor suppresses "spell checking" from the application menu when a graphics object is selected. Application functions are accessed from a function access mechanism; e.g., menus are a currently popular function access mechanism. In this thesis, application space refers to the screen space occupied by an application program.

As mentioned in previous chapters, traditional user interfaces provide separate work object representations and function access mechanisms. Modern word processors, for example, provide menus and dialog boxes with which users invoke application functions. Unfortunately, switching between a work object representation and a large menu space requires extra effort from the user. Moreover, menus usually display only a subset of all available functions due to menu space shortage. Very often, the work object representation is displayed in a separate window apart from the menus, and large work object representations are often broken up to be displayed in multiple windows. Furthermore, applications such as the front-end software of an automatic teller machine often have large menu spaces. Here, the work object representation contains text descriptions at the top of each menu screen. The menu space contains all the banking options that are menu items themselves.

Similarly, the purposes of many applications resemble those of a menu system. For example, a control application allows access to functions belonging to different parts of an underlying system. Many of these applications are indeed "menu applications".

Unfortunately, because of the inherent shortcomings of traditional user interfaces and the increases of functions in modern systems, we need a more advanced menu mechanism to facilitate function access and, at the same time, allow users to remain focused on their tasks. This is our motivation for the present work to develop an improved menu mechanism which we refer to as a "hypermenu". This approach has evolved in part from the continuous zoom, which was developed as part of the Intelligent Graphic Interface [Bartram et al 95]. In the rest of this thesis, we refer to applications using the hypermenu concept as hypermenu applications.

3.1.3 Hypermenu Overview

The hypermenu method is a general approach for building applications with hierarchical work object representations so that we can combine them with function access mechanisms into hypermenu hierarchies. Our hypermenu approach thus integrates the menu mechanism with the work object representation, reducing the interaction overhead caused by switching between the two. Consequently, the user interacts directly with the integrated menu mechanism to access application functions and the underlying system. The hypermenu hierarchy follows the structure of the work object representation.

The hypermenu approach provides the following features to facilitate navigation and function access. First, it allows control over the amount of detail on screen. The user may eliminate distracting and irrelevant details from the screen by interactively navigating up and down the hypermenu hierarchy. Second, the hypermenu approach allows magnification of areas of interest and demagnification of irrelevant areas. Further, instead of a mere

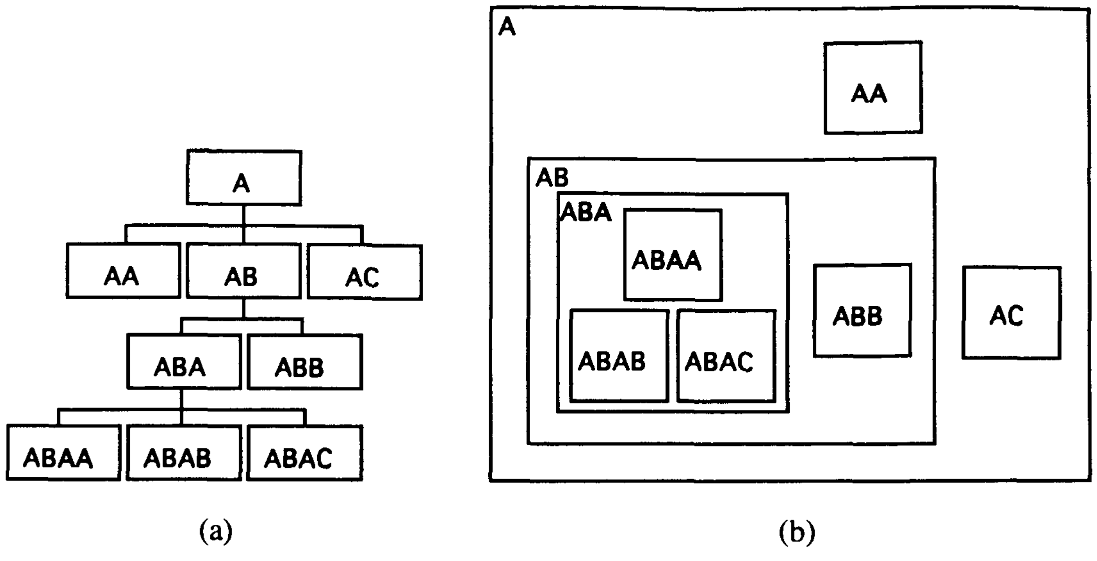


Figure 3.3: Example of a hypermenu layout. (a) Tree structure of a hierarchy. (b) Hypermenu rendering of the same hierarchy, which is implied by geometric enclosure.

scaling up or down version, it can provide different representations depending on the needs of the application and the work object constraints. This is equivalent to providing semantic zooming [Bederson 94] or semantic scaling [ZiZi 94]. Since the hypermenu approach allows multiple areas to be viewed simultaneously, the user may examine and compare multiple areas of interest, a facility not possible with traditional menu mechanisms. Third and perhaps most importantly, it allows the user to access application functions by interacting directly with the work object representation. In other words, the application is the menu. Finally, the hypermenu approach preserves the global context of the work object representation, which, in turn, provides context for the integrated menu mechanism, thereby allowing function access via direct manipulation.

To provide the above features, our hypermenu approach uses a hierarchy and a context management scheme. Each hypermenu component in the hierarchy is a *combined representation* of function access mechanism and a corresponding component in the work object representation. The context management scheme organizes the on-screen layout and appearance of every hypermenu component. The hypermenu hierarchy is reflected in geometric enclosure (Figure 3.3). Lastly, work object constraints are integrated into the context management scheme to reflect the work context which includes the global context of the work object representation and the application state. In the next subsections, we discuss properties of hypermenu components and the hypermenu's context management scheme.

3.1.4 Hypermenu Components

A hypermenu hierarchy consists of leaves, clusters, and a root. A leaf contains no child component; a cluster contains one or more child components. The top-most cluster is called the root. Normally, clusters provide higher-level abstractions or hierarchical abstractions for their child components. Each hypermenu component has open, close and zero or more zoom states.

In general, opening a cluster makes visible its child components, thereby providing increased detail (See Figure 3.4). When a cluster is opened, all its child components are initially closed. Closing a cluster also closes all its descendant components and therefore hides lower level details. A leaf is always closed.

Often, details away from focus areas in the work object representation may need to be suppressed in order to display the focus areas in greater detail, and to avoid distracting the

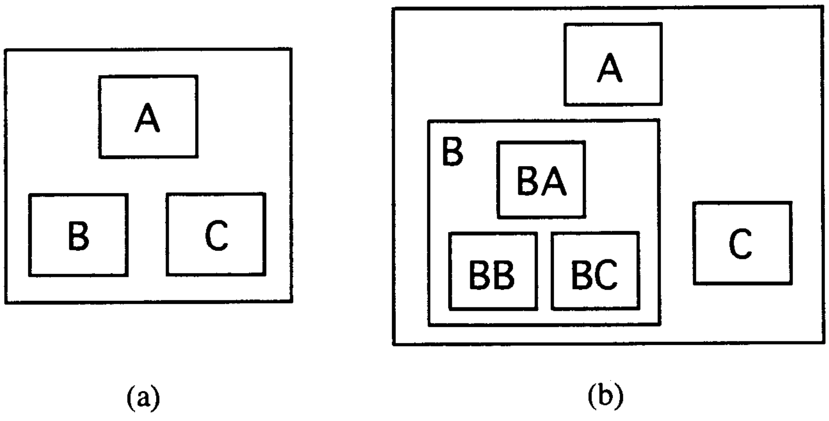


Figure 3.4: Example of opening a hypermenu component. (a) Initially, all hypermenu components are closed. (b) Opening hypermenu component B makes visible its child components .

viewer. In a hypermenu application, the user may interactively open and close hypermenu components to obtain details on demand and to hide details not of current interest. Opening and closing allows the user to see multiple components in different levels of detail simultaneously, thereby providing a degree of control over the amount of context on screen.

Hierarchical abstractions summarize lower-level detail and contribute to the overall work context. Closed clusters are represented by hierarchical abstractions which are dynamically updated to reflect proper component status. Therefore, a hypermenu component is allowed to have multiple hierarchical abstractions depending on the state of the underlying object. For example, a battery may have two hierarchical abstractions with the first representing a fully charged battery and the second representing a dead battery. Hierarchical abstractions are usually symbolic representations of the underlying system components.

Very often, information on the screen needs to be controlled and filtered [Noik 94]. Filtering means eliminating unnecessary data from the screen. Typically, only a subset of data about an object is needed at any given time. To achieve filtering, each closed hypermenu component has multiple zoom states to accommodate multiple representations with different sizes. Switching from one zoom state to another is achieved by zooming. Zooming involves changing the sizes and representations of hypermenu components. Instead of supplying a mere scaled version, supporting multiple representations allows displaying a more appropriate and informative representation during zooming. Furthermore, object attributes to be displayed and the screen space required for each zoom state can be pre-determined, hence facilitating efficient use of screen space. For example, the first zoom state may be an output-only functional representation displaying the most up-to-date status of the underlying object, and the second zoom state may be a functional representation allowing both input and output.

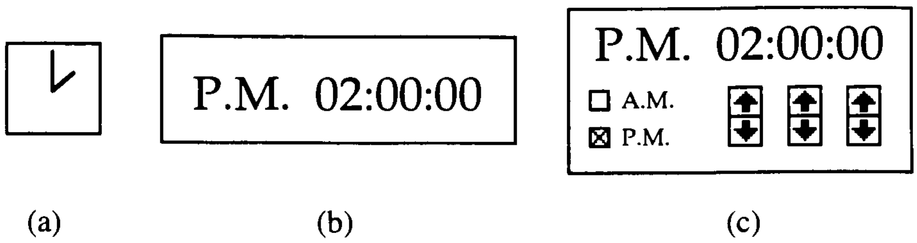


Figure 3.5: Example of zooming and size-dependent representation. (a) A clock symbol. (b) A functional representation of the clock. (c) A functional representation of the clock with time adjustment mechanism.

Zoom states are also called size-dependent representations. They provide combined representations of function access mechanisms and the underlying system component. Therefore, each size-dependent representation maintains a set of user interface elements that serves as a function-access mechanism capable of displaying appropriate component status and accepting user input. Conceptually, size-dependent representations let the user feel he is interacting with the underlying system component directly. Figure 3.5 shows an example of zooming in on a hypermenu component representing a clock. Initially, a clock-like icon, which is a static symbol, represents the clock in its closed state (Figure 3.5(a)). When zoomed in, the hypermenu displays a functional representation of the clock showing the up-to-date time (Figure 3.5(b)). When zoomed in further, the time adjustment mechanism is also displayed, allowing the user to adjust the time interactively (Figure 3.5(c)). To provide all the above features, the hypermenu approach relies heavily on its context management scheme, described next.

3.1.5 The Context Management Scheme Of Our Hypermenu Approach

Context proved to be vital to user orientation when navigating and accessing functions in complex systems [Schaffer 93]. The goal of the context management scheme is to maintain an intuitive work context. It is responsible for rendering the visible portion of the hypermenu hierarchy onto the screen. It consists of the discrete zoom algorithm and domain-specific logic to support proper hierarchical abstractions, size-dependent representations and restructuring of the hypermenu hierarchy according to the work object constraints.

3.1.5.1 Discrete Zoom Algorithm

This is a variation of the global version of the continuous zoom algorithm [Dill 94]. In both algorithms, the magnification or scale factor changes the size of a hypermenu component without affecting its open/close state (Figure 3.6). Both zoom algorithms control only the sizes and positions of hypermenu components according to their open, close and zoom states. In general, the scale factor is controlled by mapping functions in these algorithms. The mapping function maps a magnification factor to a corresponding size and is controlled by the user.

As its name implies, the continuous zoom algorithm provides a feeling of continuous change during zooming. To achieve this, it uses a smooth continuous mapping function in controlling the size of a rectangular object. More precisely, the rectangular bounding box of each hierarchical component is controlled by a monotonically increasing function. For example:

$$\text{New size} = f(\text{Magnification factor}) \times \text{Initial size}$$

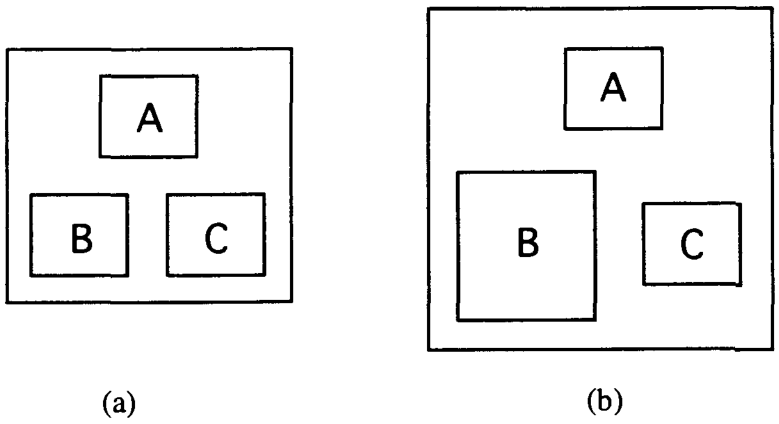


Figure 3.6: Magnification example. (a) The hypermenu layout before magnification. (b) Magnifying hypermenu component B enlarges its size without changing the open/close state.

where f is a continuous function [Dill 94]. An advantage of the continuous zoom algorithm is allowing continuous control over the size of each hierarchical component. This can, however, result in increased cognitive effort to magnify or demagnify a hypermenu component to a specific size because the user must continuously monitor the size of components being zoomed.

The discrete zoom algorithm, on the other hand, is designed to support hypermenu components with multiple fixed-size representations as discussed in the previous subsection. To facilitate achieving specific fixed size goals, the discrete zoom also supports a finite number of sizes instead of supporting a continuous size change. This is equivalent to using a step function as the mapping function. In the context of the discrete zoom, a zoom state is simply a fixed size allocated from the available screen space for a hypermenu component. The major drawback to sudden changes in size is the fact that it induces visual discontinuity. To alleviate this problem, size change is achieved in a number of steps. The major advantage of the discrete zoom is minimizing the interaction needed to switch from one size to another, regardless of how much they differ. It also requires significantly less cognitive effort from the user during zooming.

In overview, the discrete zoom algorithm accepts the initial layout of a hierarchy, which we call the *normal geometry*, and the sizes of all closed and zoom states when program execution starts. The initial layout of a hierarchy describes the size and position of hierarchical components by geometric enclosure. For example, a cluster is described by its location, open size, and closed size. Descriptions for leaves are similar except that sizes of all possible zoom states are included. Each component, also called a node, is assigned a scale factor. The scale factor used in calculating component size is $S = L_f/L_i$, where L_f is the final size and L_i is the initial size of a component. The zoom algorithm combines the normal geometry and the scale factors to produce the *zoomed geometry*, which is displayed. Unlike the continuous zoom algorithm, users do not change the scale factor directly during interaction; rather, scale factors for each node are derived dynamically during opening, closing and zooming according to the pre-specified sizes.

Like the global continuous zoom, the discreet zoom works on X and Y axes separately. The first step is to project all node boundaries into the X and Y axes. The algorithm then works with the one-dimensional intervals between adjacent projections (Figure 3.7). Each interval is either a projection of one or more nodes, or is a gap interval such as X_1, X_3 and X_5. Intervals are assigned the maximum scale factor of all nodes that project into them. This prevents the size of the node from exceeding the sum of the sizes of the intervals that contain it.

Given the scale factors of all the nodes within a cluster, the total amount of space requested by a cluster (in the X direction) is:

$$Xreq = \sum_{i} x_i s_i$$

where x_i is the normal length of the ith interval and s_i is its scale factor. Space in the Y direction is calculated similarly. Intervals are used instead of node widths because the former never overlap.

The intersections of the lengths of the intervals containing the edges of a node in both X and Y directions constitute the total space available to the node. This space is rectangular in

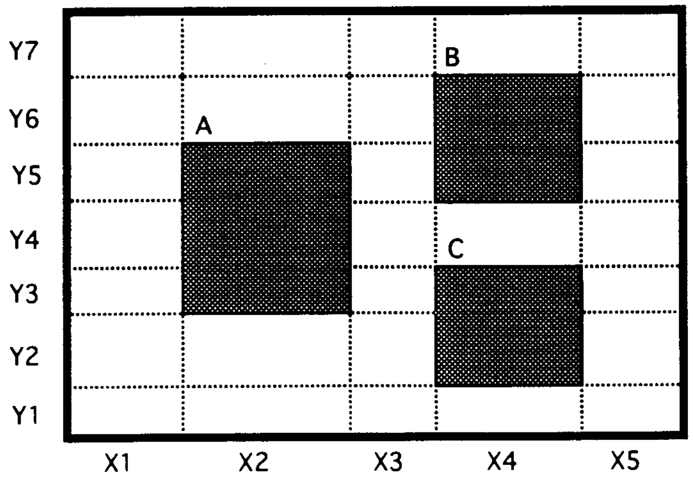


Figure 3.7: Sample three-node hierarchy, initial size.

shape and is called a *zoom hole*. After computing the sizes of the intervals and nodes, the nodes are repositioned according to the location of their center points. Note that as the size of its containing interval changes, a node's center stays at the same relative position in its zoom hole (Figure 3.8).

Calculations for the scale factors and space requests propagate only upward in the hierarchy until the root node is reached. If the total requested size is less than or equal to the available screen space, the discrete zoom algorithm grants the size change requests; otherwise, it denies the size changes requests and signals the hypermenu application to notify its user.

To divide the size changes into a number of steps, the discrete zoom algorithm calculates a number of intermediate sizes. The starting size is equal to the initial size and the ending size is equal to the final size of the node. If a size change involves n steps, the kth intermediate size is:

$$L = (k/n)(L_f - L_i) + L_i$$

where L_f is the final length, L_i is the initial length and L is the intermediate length of an edge. Calculations for L, L_i, and L_f are done independently in both X and Y directions.

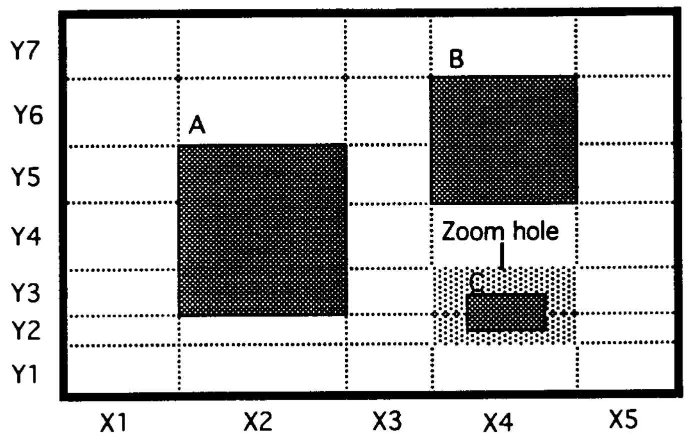


Figure 3.8: Node C zoomed out from Figure 3.7.

3.1.5.2 Outline Animation

The global context is so important to the user during navigation because it provides a sense of orientation in the system. It is equally important to help the user perceive and follow all the changes in the context. Opening, closing, and zooming changes the global context of the work object representation. For instance, zooming in on a component changes its size and shifts all the components from their previous positions. As mentioned before, sudden changes in size and position cause visual discontinuity. In order to let the user follow the changes mentally, we need to minimize visual discontinuity. Ideally, morphing yields a smooth transition one can easily follow. However, it quickly becomes computationally intensive and impractical when applied to complex work object representations. Instead, we use "outline animation", a planar (2D) animation, in our hypermenu approach. Outline animation consists of three steps. First, all the visible hypermenu components are changed into filled rectangles. Each filled rectangle is color-coded to indicate its depth in the hierarchy. Second, it divides the open, close and zoom operations into several steps. At each step, the intermediate size and position of the changing component is calculated and drawn as a color-filled rectangle. The same process is also carried out on all other visible components. Third, when the changing component reaches its final size, the appropriate hierarchical abstractions and size-dependent representations are drawn for all visible components. Outline animation is much less computationally intensive than morphing. The following paragraphs provide illustrations on zooming and opening.

In the case of zooming, the context management scheme first verifies that there is enough screen space available. If there is, the existing representation (Figure 3.9(a)) changes into a filled rectangle. Then, intermediate sizes and positions are repeatedly calculated for the changing component and rendered as rectangles (Figure 3.9(b)). Once the hypermenu component has reached its final size, the rectangle changes into the final

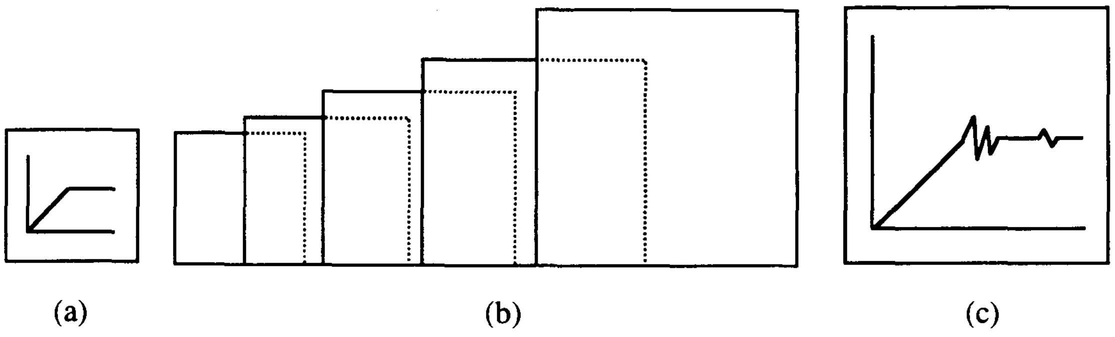


Figure 3.9: Example of zooming in a hypermenu application. (a) Initial hypermenu component. (b) A set of intermediate rectangles belonging to outline animation. (c) Final hypermenu component appearance.

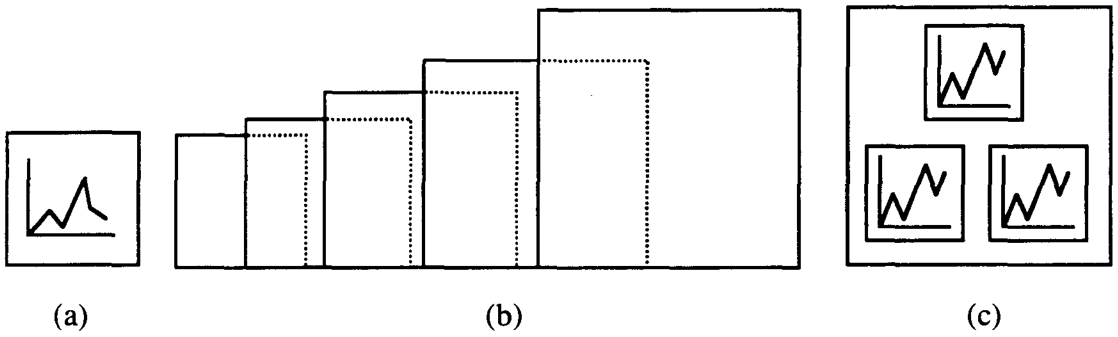


Figure 3.10: Example of opening in a hypermenu application. (a) Initial hypermenu component. (b) A set of intermediate rectangles belonging to outline animation. (c) Final hypermenu component appearance.

representation (Figure 3.9(c)). The entire process (Figure 3.9(a) to Figure 3.9(c)) is referred to as outline animation.

Opening works similarly; the new sizes and positions for all hypermenu components are calculated to verify that there is enough screen space. If there is, the existing hierarchical abstraction (Figure 3.10(a)) changes into a filled rectangle with intermediate rectangles drawn as in zooming (Figure 3.10(b)). Once the hypermenu component has reached its fully opened size, the hierarchical abstractions of its children are displayed (Figure 3.10(c)). Closing is done similarly except that it reverses the open operation.

3.1.5.3 Work-Object-Constraint Controlled Hypermenu Hierarchy

As described earlier, work object constraints are rules that govern the availability of functions and application states. Integrating them into the context management scheme enables a hypermenu application to display proper functional representations and provide necessary functions according to the application states. For instance, a hypermenu application may need to dynamically suppress various components representing unneeded functions to comply with work object constraints. Figure 3.11 provides an abstract example. In this example, when the application is in state 1, hypermenu components ABA, ABB, and ABC are not displayed when AA is opened. When the application is in state 2, only hypermenu component AAA is displayed when AA is opened. Finally, when the application is in state 3, only hypermenu components AAA and AAB are displayed when AA is opened. The work object constraints state that when hypermenu component AA is opened the second time, the application changes from state 1 to state 2, when it is opened the third time, the application changes from state 2 to state 3, and when it is opened the third time, the application changes from state 3 back to state 1. In this way, a hypermenu application can dynamically restructure and re-render its hierarchy using the work object constraints. We provide examples based on real-world applications in the next chapter.

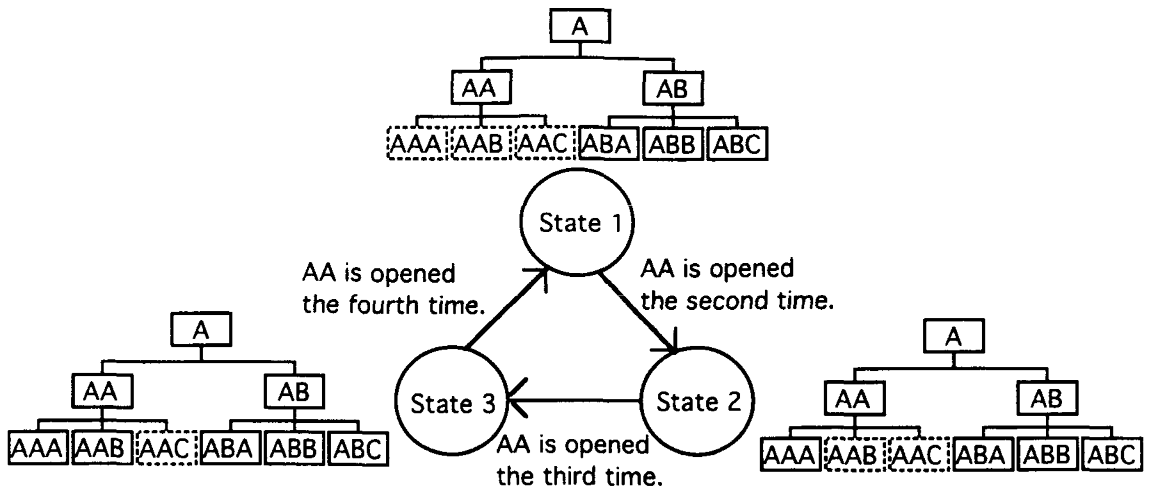

A
AA
AB
AAA
AAB
AAC
ABA
ABB
ABC
State 1
State 2
State 3
AA is opened the fourth time.
AA is opened the second time.
AA is opened the third time.

Figure 3.11: Example of work-object-constraint controlled hypermenu behavior. When the application is in state 1, hypermenu components ABA, ABB, and ABC are not displayed even if component AA is opened. When the application is in state 2, only hypermenu component AAA is displayed when component AA is opened. Finally, when the application is in state 3, only hypermenu components AAA and AAB are displayed when component AA is opened.

3.2 Navigation And Function Access

To navigate within a hypermenu application, the user interactively opens and closes hypermenu components. Support for direct manipulation preserves continuity of the user's mental model for the work object representation without depending on separate navigation mechanisms. Opening and closing hypermenu item corresponds to navigating down and up the hypermenu hierarchy to a lower/higher level abstraction. Outline animation is provided during opening, closing and zooming to preserve the visual continuity of the work context during the transition. Most importantly, navigation depends on available screen space. As long as there is enough screen space, the user can open and zoom in any component. However, if there is not enough screen space, the user must close or zoom out less important components to make room for the component to be opened or zoomed in. Zooming allows control over the type and amount of data to be displayed. Opening allows the user to see lower level details. Closing shrinks a component into a more concise abstraction and fuses it to the work context. The work context, in turn, serves as a vital aid to orient the user during navigation. Moreover, the hypermenu's context management scheme facilitates visual searching which benefits random browsing and serendipitous searching.

To access embedded functions, the user zooms in a hypermenu component to bring up a functional representation of the underlying object. Then, the user interacts directly with the functional representation to access functions in the underlying object. The hypermenu approach brings two major advantages to function access. First, it allows navigation and function access via direct manipulation such as point and click. Second, the hypermenu's context management allows function access in context.

3.3 Construction Of Hypermenu Applications

In addition to conventional design approach, constructing a hypermenu application involves three extra steps:

1. Define an appropriate hierarchical structure for the system, consistent with its functions. If this is not possible, the system may not be suitable for a hypermenu approach.

2. Identify the availability of functions and corresponding hypermenu components for each application state. For example, at each application state, only a subset of the hierarchy should be available to the user.

3. Define open, close and zoom states for individual hypermenu components. Determine appropriate hierarchical abstractions and size-dependent representations according to the work object constraints.

The following chapter presents two hypermenu applications designed using the above guidelines. The first example applies the hypermenu approach to a Group Technology database browser and designer. The second example illustrates the use of our hypermenu approach in designing a user interface for a complex piece of telecommunication network test equipment[1]. In these examples, we illustrate: (1) constructing hypermenu applications, (2) the look and feel of hypermenu applications, and (3) visualization, navigation, and function access enhancements brought by them.

[1] The second example is based on Hewlett Packard's Broadband Series Test System.

CHAPTER 4

HYPERMENU ILLUSTRATIONS

4.1 Introduction

The hypermenu concept is a general user interface approach and may be applied to a range of application domains. In this chapter, we illustrate the potential of this technology by applying it to two different areas. We first illustrate its application to computer-aided-design by introducing a new approach for designing mechanical parts with group technology. We then apply the hypermenu concept to the user interface of a testing controller for telecommunication networks and show that it facilitates user interactions and prevents cluttering resulting from a small display area.

4.2 Hypermenu Approach To Group Technology

4.2.1 Introduction To Group Technology

As manufacturing companies strive to enhance their competitiveness in the global marketplace, they are constantly exploring for new technologies. An approach increasingly useful in helping achieve a higher level of integration between design and manufacturing is that of Group Technology (GT). Group Technology is an approach to design and manufacture in which parts are grouped into families according to their general shape, the material they are made of, the series of steps needed to manufacture them, etc. Parts belonging to the same families exhibit similar characteristics and features.

Coding parts and grouping them into families is the essence of group technology and has many advantages for a manufacturer. First, the manufacturer no longer has to deal with thousands of parts and processes, but can focus on a small number of part families and processes. Second, group technology provides a basis for a company to switch from inefficient job-shop manufacturing methods to modern cellular manufacturing. Third, design engineers can finally stop "reinventing the wheel": when faced with designing a new part, the engineer first assigns a group-technology code to the part, then checks the computer database for the same or similar parts. Often, he ends up using an existing part, preventing needless and costly parts proliferation and freeing up design time.

Finally, Group Technology greatly simplifies new-part process planning. The designer codes, then classifies the new part into the appropriate existing family. Since all the parts in that family are made by the same process, the manufacturing process for the new part already exists: it is merely a matter of fine-tuning it [Snead 89].

Group Technology relies significantly on classification and coding. Classification is the process of grouping together similar things. Coding is a technique of allocating predetermined symbols to describe and communicate the classification. Coding describes a physical object in a notation that is easy for computers to store and retrieve [Snead 89].

In this section, we use MDSI CODE as an example from which we build our hypermenu application. The reason for choosing MDSI CODE is because a detailed description was easily available. CODE is the name of the classification and coding system provided by Manufacturing Data Systems, Incorporated (MDSI). It is an eight-digit hexadecimal-based code used primarily to classify and code mechanical parts. The code structure is shown in Figure 4.1. The first digit of the code is used to identify the major divisions and was set to one by MDSI due to the company's internal structure. The remaining seven digits are division-specific codes used to describe the shape, features and dimensions of a part. Some examples are shown in Figure 4.2.

Once the mechanical parts have been classified and coded, they are stored into a group technology database for retrieval. The objectives of a group technology database include: (1) facilitate serendipitous searching for an existing or similar mechanical part, (2) facilitate defining a new part or part family in the group technology database according to the classification used. Most of the time, geometry properties such as shape are used to identify a mechanical part.

To facilitate serendipitous searching and random browsing, the hypermenu approach offers the following advantages. First, it stores all the mechanical parts in a similar manner to many hierarchical classification and coding systems such as MDSI CODE. Each digit in MDSI CODE corresponds to a level in the hypermenu hierarchy, which is called a digit level. Second, the hypermenu approach allows searching for a mechanical part in the group technology database with direct manipulation as one would with physical storage bins. Third, the hypermenu approach displays the shape of mechanical parts and part

MDSI CODE

MAJOR CATEGORY	SHAPE AND FEATURES	DIMENSIONS
$\boxed{1}$	$\boxed{3}\boxed{1}\boxed{8}\boxed{8}\boxed{4}$	$\boxed{7}\boxed{8}$

Figure 4.1: MDSI code structure.

MDSI

Major Dimension $\boxed{1}\boxed{\ }\boxed{\ }\boxed{\ }\boxed{\ }\boxed{\ }\boxed{\ }\boxed{\ }$

Description	Second	Third	Fourth	Fifth	Sixth	Seventh		Eighth	
	O.D. or section	Center hole	Holes (other then center)	Grooves threads	Misc.	MAX O.D. or section across flats		MAX. overall length	
						>	<=	>	<=
0	Other than	Other than	Other than or none	Other than or none	Other than or none		0.10		1.00
1	Single cylinder	None	Longitudinal other than bolt circle	Groove(S) external	Concentric variations	0.10	0.16	1.00	1.60
2	Multi-concave cylinder	Single thru going hole	Radial round	Groove(s) internal	Protrusions from main shape	0.16	0.27	1.60	2.70
3	Multi-convex cylinder	Single blind hole	Longitudinal and radial round	Groove(s) external & internal	Concentric variations & protrusions from main shape	0.27	0.39	2.70	3.90
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 4.2: MDSI code specification.

families in the database to support visual searching. Allowing visual searching is an advantage of our hypermenu approach because it does not require the user to manually translate a mechanical part into a part code and then have the computer match it in the database. Rather, the user can search the parts database directly based on the geometric attributes illustrated in the drawings. Fourth, the hypermenu approach helps the user visualize characteristics of part families that are not physical entities. Fifth, it clearly indicates hierarchical relationships between parts and part families according to the classification and coding system used. Therefore, we believe that the hypermenu approach is a powerful and appropriate interface to group technology databases.

Our hypermenu approach also offers the following advantages to facilitate defining a new part in a group technology database. First, it helps the user visualizes relationships between existing parts and the new part to be design. Second, the hypermenu approach allows the user to define a new part *in context* with instant feedback on its characteristics. Third, it offers a preview of the to-be-designed item at each digit level. Finally, the hypermenu approach supports the idea of incremental fine-tuning. For example, the user may only fine-tune attributes allowed by each digit level.

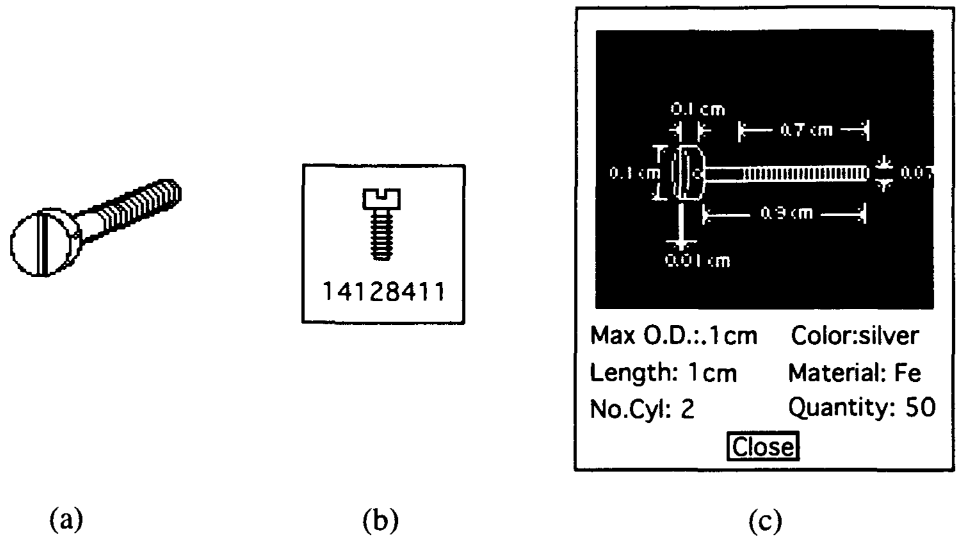
4.2.2 Constructing A GT Database Hypermenu Application

Following the guidelines for constructing hypermenu applications in chapter 3, we first identify the work object constraints for the front-end software of the GT database application. The application should provide a browse mode and a design mode. In browse mode, the hypermenu application displays only *defined* parts and part families in the GT database. Defined or existing parts and part families have their features completely specified and are stored in the GT database. In design mode, the application displays *all possible* parts and part families supported by the classification and coding system. Parts and part families not in the GT database but supported by the classification and coding system are called *undefined* parts and *undefined* part families.

Parts in the GT database become leaves, and part families become clusters. Defined parts represented by leaves are always closed and have only one zoom state. The hierarchical abstraction for the closed state displays the shape and the part code of the corresponding mechanical part. The size-dependent representation for the zoom state displays a dialog box containing the part drawing, and text labels showing all the attributes. Part families represented by clusters need open and closed states only. The hierarchical abstraction for the closed state also displays the shape and part code of the corresponding part family. Note that a part family does not have a complete part encoding; for example, a part family with the second and third digits equal to two and four might have part code 124XXXXX, where "X" is any valid character in the range of the underlying classification and coding system.

Opening a defined part family exposes members of the part family one level below, and zooming in on defined parts provides further details on the object parameters.

We illustrate with an example. Figure 4.3(a) shows a bolt. The hierarchical abstraction of the corresponding hypermenu component shows the shape and part code after classification and coding (Figure 4.3(b)). The size-dependent representation of the zoom-in state contains the drawing and text labels displaying the object attributes (Figure 4.3(c)). Similarly, the hierarchical abstraction corresponding to a defined part family shows the shape and the partial part code (Figure 4.4(a)). Note that defined part families are represented by clusters that do not have zoom states, only open and close states. Opening defined part families shows members of the part family one level below (Figure 4.4(b)).



Max O.D.:.1cm Color:silver
Length: 1cm Material: Fe
No.Cyl: 2 Quantity: 50

Figure 4.3: Hierarchical abstraction and size-dependent representation of a defined mechanical part. (a) Bolt to be represented. (b) Hierarchical abstraction of the bolt. (c) The size-dependent representation of the bolt is a dialog box containing a drawing and description about the bolt.

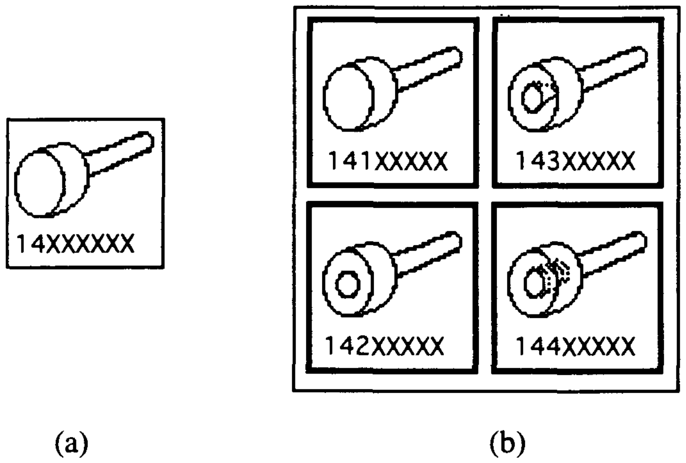
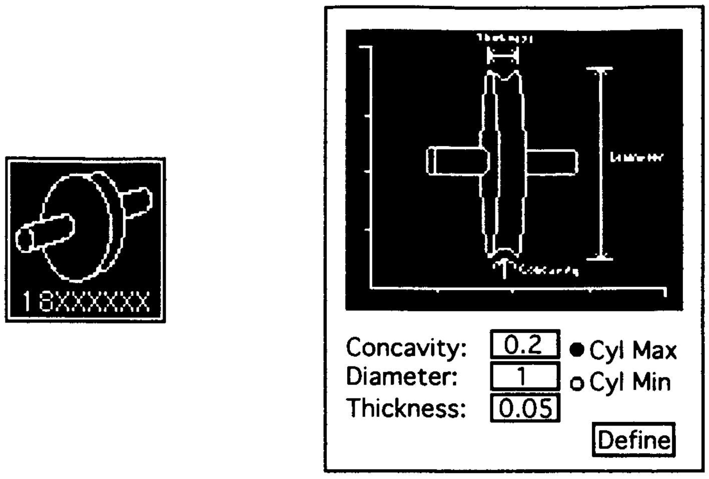


Figure 4.4: Hierarchical abstraction and corresponding part family. (a) Cluster representing hierarchical abstraction of a part family. (b) Cluster of (a) opened to show members of the part family one level below.



Concavity: 0.2 [✓ Cyl Max]
Diameter: 1 Cyl Min
Thickness: 0.05

(a) (b)

Figure 4.5: Hierarchical abstraction and size-dependent representation of an undefined mechanical part. (a) Hierarchical abstraction of an incompletely defined pulley. (b) Size-dependent representation of the pulley contains text boxes requesting pulley-related information.

To distinguish undefined parts and families from defined parts and families, the former are highlighted (Figure 4.5(a)). When an undefined part family is opened, the hypermenu application displays a dialog box for the user to adjust object parameters according to the digit level. For example, the dialog box in Figure 4.5(b) allows the user to fine-tune parameters like pulley width, length and concavity. Undefined parts are represented by leaves. A single part, not a part family, may be zoomed-in. The result is a dialog box similar to Figure 4.5(b).

4.2.3 Behaviors Of The Hypermenu Application

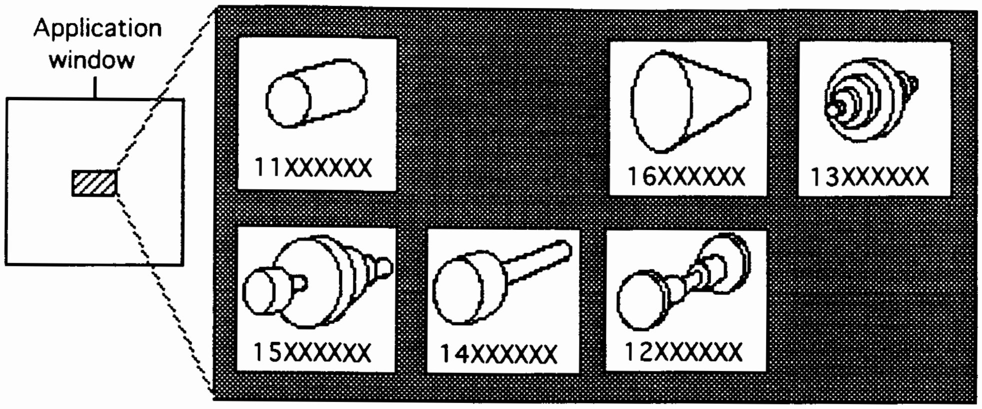
To illustrate our hypermenu approach, we show how to search a GT database for a mechanical part, and how one would design a new part using a hypermenu-based GT system.

4.2.3.1 Search for a mechanical part in browse mode

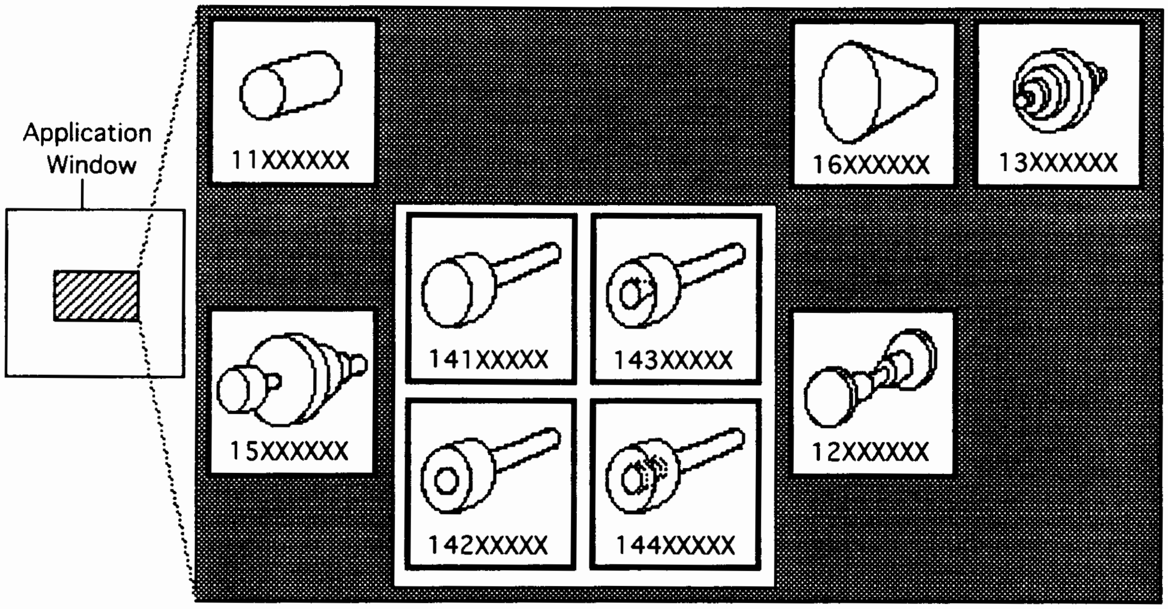
Suppose we would like to search for the bolt shown in Figure 4.3(a). To start searching, we first put the hypermenu application into browse mode to display only defined parts and part families, thus reducing distraction. The first-digit level in the hypermenu hierarchy contains only one component because the first digit in MDSI CODE is always one. The root hypermenu component could display the logo of the classification and coding system when it is closed.

Opening the first-digit-level component exposes closed components belonging to the second digit level (Figure 4.6(a)), and, we say that we are in the second digit-level. At this level, we can search visually by geometric properties or part code. Following the hierarchical abstractions, we navigate through the seven digit levels as shown in Figure 4.6(b) to 4.6(g). In general, we say that we are in $(n+1)^{th}$ digit level after opening a component or part family belonging to n^{th} digit level. Opening part families from the second to the seventh digit level exposes more detailed part families. Opening part families in the eighth level exposes defined mechanical parts. Once we are inside the eighth digit level (Figure 4.6(g)), we can zoom in any mechanical part to see a detailed description of its attributes (Figure 4.6(h)).

As implied by our illustration, screen space required to display the GT database is proportional to the depth of the hypermenu hierarchy and the number of items within each level. Opening and zooming in a hypermenu component enlarges all its ancestors so that they are able to hold both the enlarged component and its siblings. For deep and large GT databases, screen space runs out quickly as one navigates downward in the hierarchies. When the hypermenu application runs out of screen space, the user will not be able to navigate further down the hierarchy or zoom in item of interest any more. To alleviate this

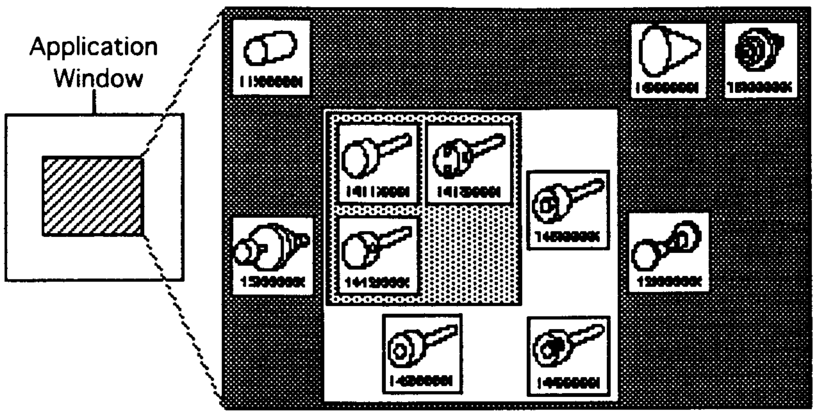


(a)

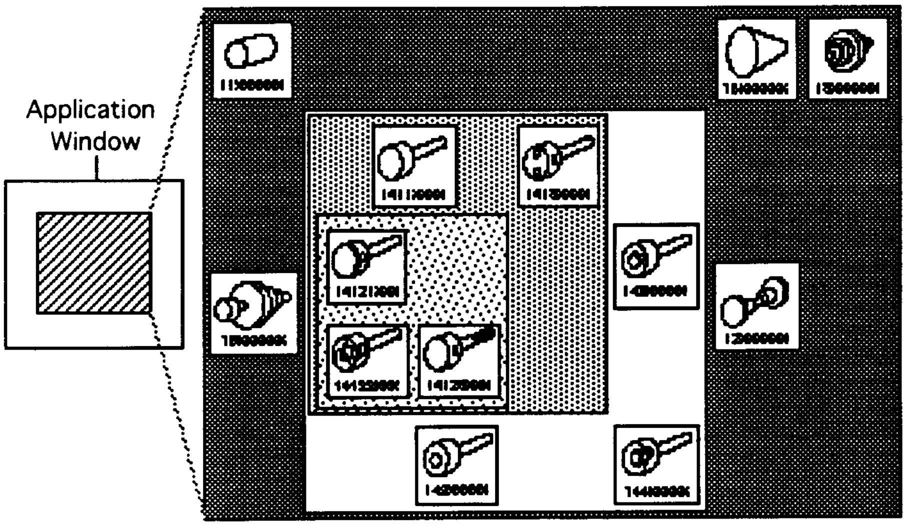


(b)

Figure 4.6: Searching for a defined mechanical part in a hypermenu application. (a) Second digit level. (b) Third digit level. Note that we have opened the 14XXXXXX hypermenu component to show its members (141XXXXX, 142XXXXX, 143XXXXX and 144XXXXX).

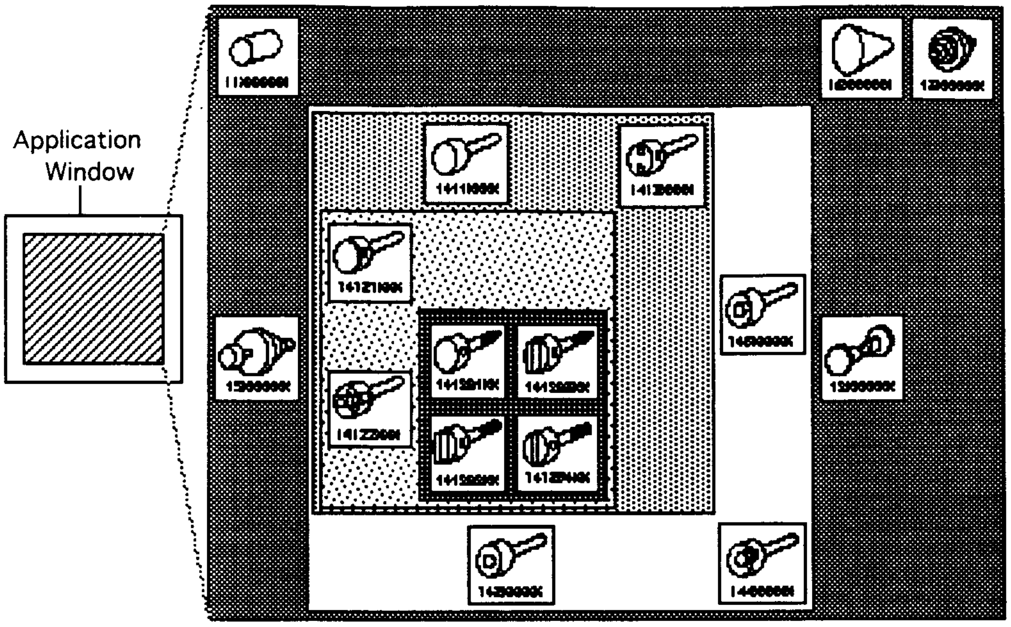


(c)

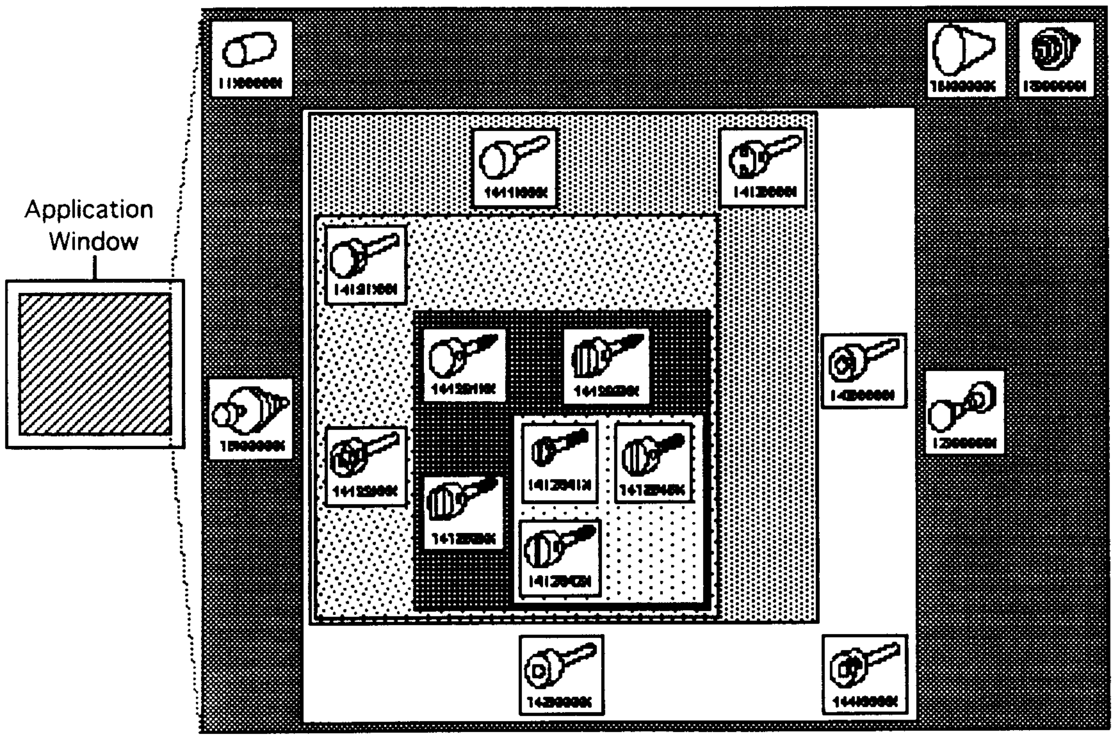


(d)

Figure 4.6 (Continued): Searching for a defined mechanical part in a hypermenu application. (c) Fourth digit level. (d) Fifth digit level.

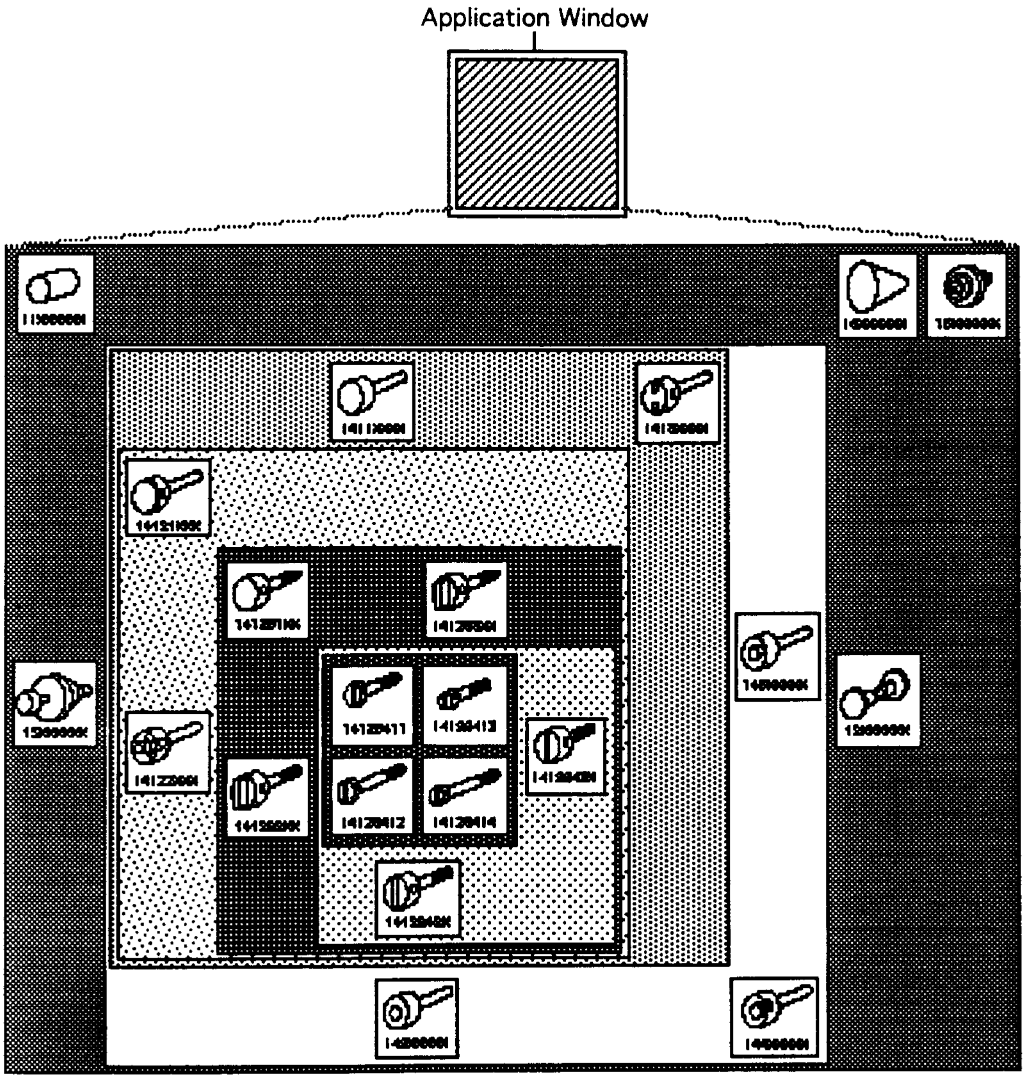


(e)



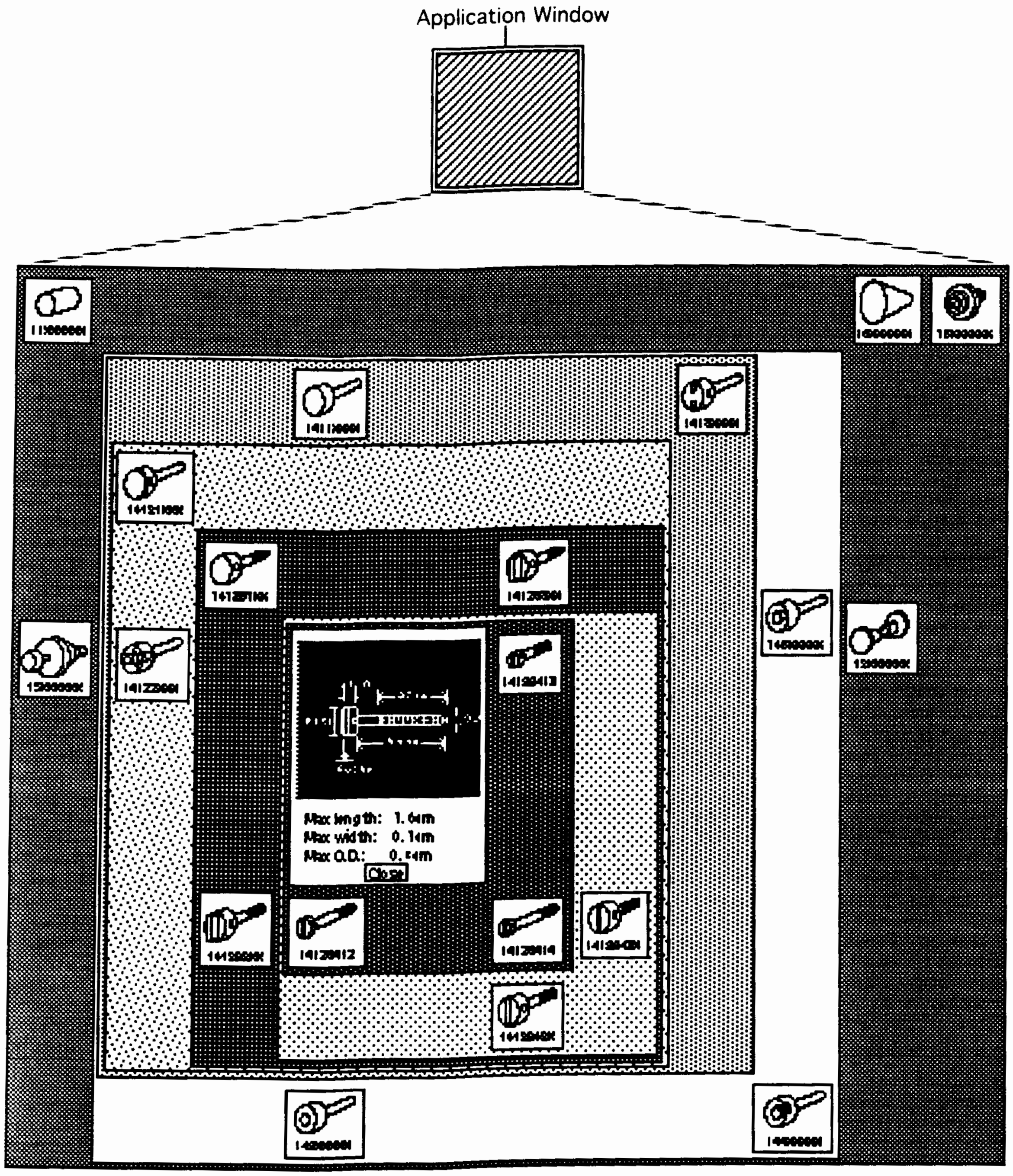
(f)

Figure 4.6 (Continued): Searching for a defined mechanical part in a hypermenu application. (e) Sixth digit level. (f) Seventh digit level.



(g)

Figure 4.6 (Continued): Searching for a defined mechanical part in a hypermenu application. (g) Eighth digit level.



(h)

Figure 4.6 (Continued): Searching for a defined mechanical part in a hypermenu application. (h) Eight digit level.

problem, designers might reduce the size of hierarchical abstractions by using more concise descriptions. In addition, clusters can have multiple zoom states. For example, the hierarchical abstraction might show only the part code. The first zoom-in state shows only the shape and the second zoom-in state shows both the shape and the part code.

In MDSI CODE, each of the last two digits denote ranges rather than an exact value. Therefore, it is possible to have more than one mechanical part with the same part code. For example, a mechanical part with overall length between 1 and 1.6 units has a MDSI code ending in "2". There are two ways to support this in our hypermenu application. First, each mechanical part can be made into hypermenu components belonging to the ninth level in the hierarchy. Second, we can provide mechanisms in the size-dependent representation for the user to cycle through all the mechanical parts at the eight digit level.

4.2.3.2 Defining and fine-tuning a mechanical part in design mode

Defining a part code for a new part is equivalent to selecting and defining a design path in the hierarchical classification and coding system. For demonstration purposes, we limit the range of the second digit from one to eight, and the remaining digits except the first digit from one to four.

Suppose we wish to design the pulley shown in Figure 4.7. First, we enter design mode to display both existing and undefined parts and part families. Hierarchical abstractions for undefined parts and part families are highlighted and become design options in the hypermenu application. It is necessary to display defined part families to let the user choose a similar part for fine-tuning. To start, we navigate to the second digit level (Figure 4.8(a)). In this case, no defined part family resembles the pulley we want to design. Therefore, we define a new part family by choosing available design options in the

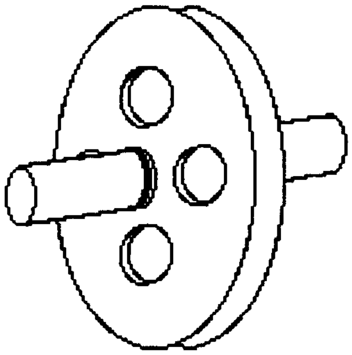
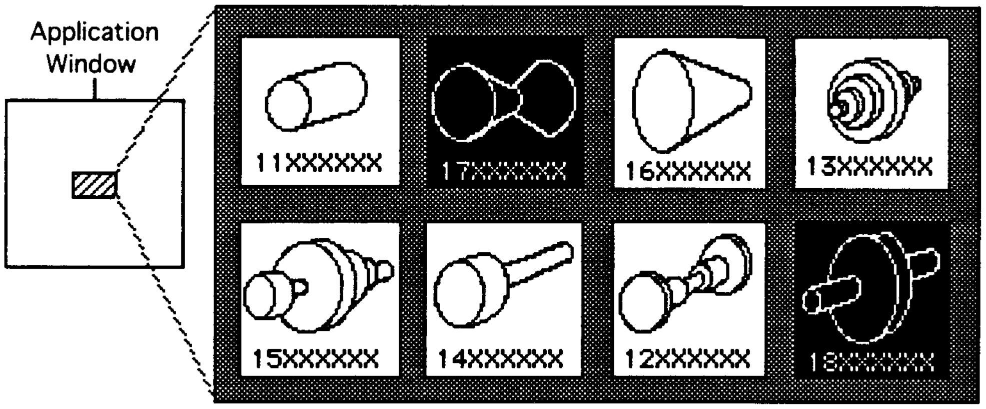
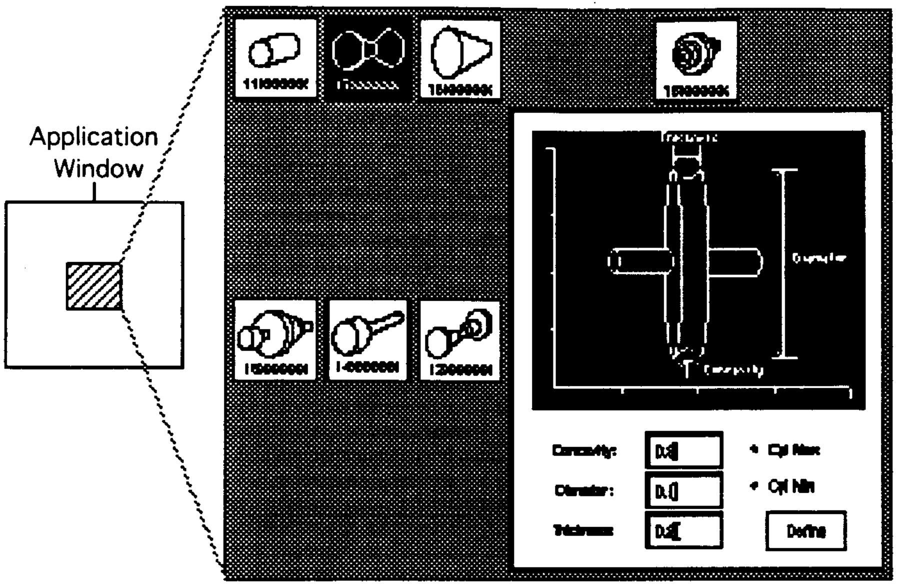


Figure 4.7: Pulley to be designed.

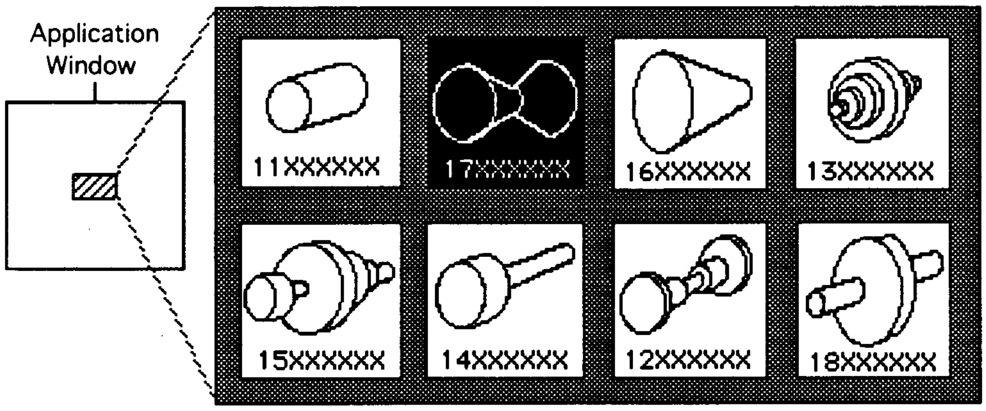


(a)

Figure 4.8: Example of defining a new mechanical part at second digit level. (a) In design mode, representations of undefined mechanical parts are highlighted using reverse video.



(b)



(c)

Figure 4.8 (Continued): (b) Opening part family 18XXXXXX brings up a dialog box for the user to define attributes for the current digit level. (c) The hierarchical abstraction of the part family changes to that of a defined part family after the attributes have been defined.

second digit level. According to the hierarchical abstractions, part family 18XXXXXX resembles the mechanical part we are designing.

Opening the undefined part family brings up a dialog box for adjusting the parameters (Figure 4.8(b)). Once we have filled in the information and click the "Define" button, the hypermenu component closes itself and updates the hierarchical abstraction to reflect the proper attributes and the existence of the new part family (Figure 4.8(c)). Similarly, we define attributes for the next six levels. Once we have defined all digit levels, a new mechanical part is defined, and it can be viewed just like other existing mechanical parts in the GT database.

The hypermenu approach brings the following benefits to GT database systems. The context management scheme supports visual searching at each level during parts retrieval. Neighboring items are available on screen and the viewer may search *in context*. Most importantly, the hypermenu approach provides an easily navigable visual representation of the GT database and provides full access to all the items within. Therefore, the user may interactively control items in the GT database to be displayed, and easily zoom in on areas of interest. Direct manipulation also reduces navigation overhead. Finally, the hypermenu approach allows simultaneous viewing of multiple components which facilitates comparing different parts in the GT database.

During design, the hypermenu approach helps the user to visualize how one selects design paths. At each digit level, visual representations of design options are displayed, therefore allowing the user to rely on recognition memory in choosing a similar part for fine-tuning. Lastly, the user may simultaneously display and reference any other defined parts, which is extremely important during fine-tuning.

4.3 A Hypermenu Interface To An Equipment Tester

4.3.1 Background

To test our approach in a quite different type of application, we developed a hypermenu interface for a complex test controller used to set up, execute and monitor the performance of very high speed (Broadband) telecommunication equipment (HP Broadband Series Test System).

High speed information transfer is a complex multilevel process [Vetter 95]. In simplified form, at the highest level, information is passed from an end-to-end application, such as Local Area Network management software, to a service layer. The service layer attaches additional information depending on the type of service requested, and then passes the information to the adaptation layer for transmission. The adaptation layer breaks down the information and packages it into cells. A convergence layer accepts the cells and maps them to the physical layer which encodes the data into electrical waveforms for transmission. Each layer provides a variety of options to choose from. The combination of all the layers and the communication format of the end-to-end application defines a network transmission protocol.

The test system in question (BSTS) is designed to evaluate the network performance on transmitting data from one node to another using a variety of protocols. BSTS's user interface is responsible for providing access to the underlying equipment. Our goals for improving the quality of BSTS's user interface were: (1) preserve the user's work context and orientation, (2) reflect relationships among system components, (3) allow hierarchical management of the user interface and work context, (4) provide effective screen layout for the work context, especially components of interest, (5) reduce user interface overhead to increase user throughput, (6) permit quick and accurate component access, and (7) indicate configuration sequence.

The original user interface of BSTS provides a traditional window environment primarily made up of dialog boxes and menus. Each device is represented by a collection of dialog boxes containing all available function, and the resulting interface is quite complex. Unfortunately, overlapping dialog boxes significantly increase user interface overhead and tend to disorient the user quickly.

4.3.2 The Hypermenu Approach- An Improved Interface

We begin by outlining the overall task of a typical BSTS user. Then, we briefly describe our hypermenu interface built for the BSTS.

To conduct a test with the BSTS, the user first defines a session. A session refers to the process of setting up the BSTS for a test and running the test. Once a session has been defined, the user builds an instrument. Building an instrument requires the user to specify the protocol and devices to be used for the test. At the present, the user can choose a maximum number of two devices for each instrument. Then, the user specifies device parameters for each device belong to an instrument. Test manager and testers are logical entities added to the BSTS to form a hierarchy (Figure 4.9).

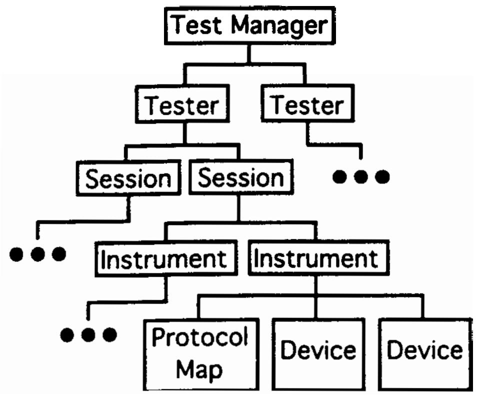
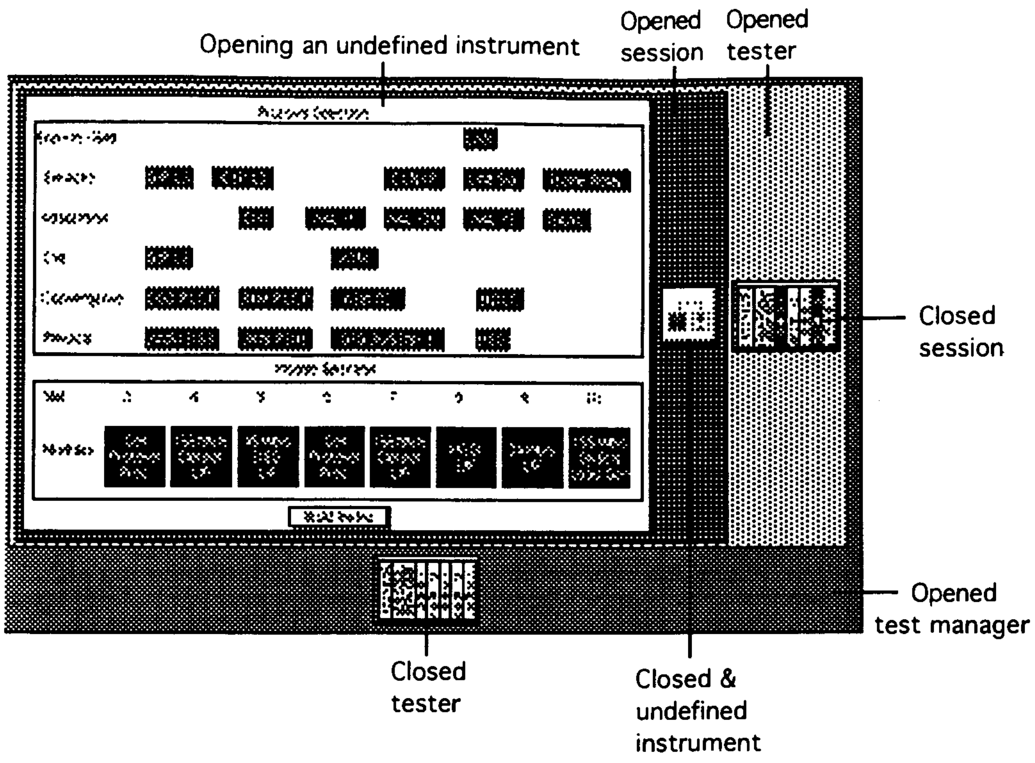


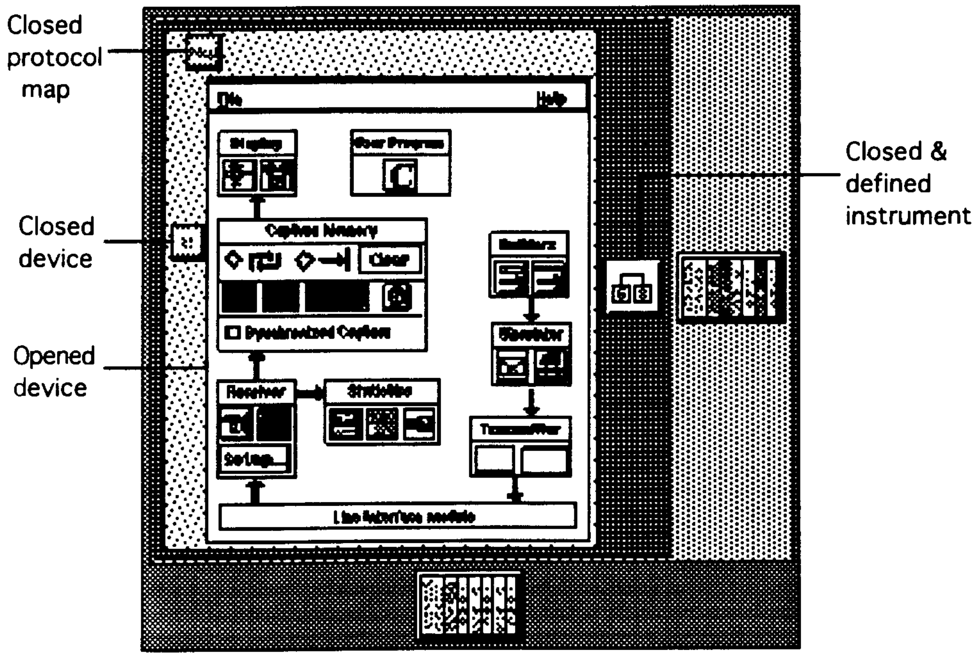
Figure 4.9: Organization of the hypermenu hierarchy for HP's test system.

Naturally, the test manager maps to the root in the hypermenu hierarchy, the testers map to the second-level components and so on. In addition to all the entities mentioned, we include a hypermenu component to represent the selected protocol at the same level as the devices for each defined instrument to indicate which protocol the instrument is set up for. All the clusters have only open/close states, and the leaves (except the protocol representation) have two zoom states. All system entities have static hierarchical abstractions, while each instrument has two different hierarchical abstractions to indicate whether or not it is defined. Devices have two size-dependent representations with which the user can interact.

Tasks the user must frequently undertake include: (1) set up test equipment, (2) examine test results, and (3) compare results from different devices. Defining and configuring an instrument is a very tedious decision-making process. Thus, preserving the configuration sequence is important in order to avoid confusion and disorientation. An advantage of the hypermenu approach is its context management scheme which assists the user in visualizing the entire sequence while focusing on one part of the system. For example, configuring an instrument is easily accomplished by navigating downwards in the hypermenu hierarchy. The hypermenu approach significantly benefits examining test results and comparing devices. Figure 4.10 shows the look-and-feel of our hypermenu application for HP's BSTS. For a detailed description of the hypermenu application for the BSTS, please refer to the appendix.



(a)



(b)

Figure 4.10: Illustrations of our hypermenu application for HP's test system.

4.4 Implementation

The heart of our hypermenu applications is the context management scheme. It consists of the discrete zoom algorithm and domain-specific logic to support proper hierarchical abstractions, size-dependent representations and restructuring of the hypermenu hierarchy according to the work object constraints. For instance, in our GT hypermenu application, children of undefined part families are temporarily removed from their parents and later restored when they have been defined. Proper hierarchical abstractions are rendered dynamically according to the work object constraints.

Both applications have been designed to run under the X/Windows system using Motif. Opened clusters are drawn as a color-filled rectangle. A closed hypermenu component is represented by a Motif push button.

The hypermenu hierarchies are specified in external files which contain the sizes and positions of all hypermenu components. In addition, each file contains closed sizes for all the hypermenu components and size-dependent representations for all the leaves.

Finally, our hypermenu applications currently run on Silicon Graphics and Hewlett Packard workstations and may be ported to any other X/Motif system.

CHAPTER 5

DISCUSSION

In this section, we discuss the hypermenu approach in the following order: (1) evaluate the strengths and weaknesses of our hypermenu approach as illustrated by the two hypermenu applications, (2) compare the use of discrete zoom with continuous zoom as the hypermenu's context management scheme.

5.1 Evaluation Of The Hypermenu Approach

Both our hypermenu applications were evaluated by domain experts. Our GT application was demonstrated to two CAD operators. We asked one of them to act as a user of the GT application to get a direct feeling of the user interface. The other CAD operator watched closely. Both operators are involved in mechanical design and one of them has been using CAD packages for eight years. Their experience is primarily with Autocad, though they have used other commercial systems as well. Similarly, we demonstrated our BSTS application to the BSTS developers; we also supplied them a copy of the software for evaluation and for demonstration to other groups within the company. In the paragraphs below, we first present the evaluations of the GT application, then evaluations of the BSTS application from the domain experts.

According to the CAD operators, the hypermenu user interface gives the following advantages. First, they felt that the hierarchical abstraction showing the shape and the part code of an object family made browsing easier. The reason they gave was that the graphics representations made it more clear which objects were available. In contrast, some of the design software with which they are familiar required parameter specification before any part became visible. Second, they liked the direct manipulation approach to selecting objects in the GT database. They felt that it required less effort than specifying a set of search parameters or criteria as required by other software they have used, although one of them mentioned that he did not mind entering numbers. Third, they believed that preserving the context facilitated navigation. In general, they felt that the context helped to orient the user in a large system by showing the relationship between the current focus and the remaining items. In addition, they said that the context helped them to visualize the GT database. Fourth, one of the operators thought that it was very helpful to preserve the spatial relationships of all the objects on screen. This helped him to get familiar with the context layout more easily and navigate more quickly after he became familiar with the user interface. Fifth, they greatly appreciated the support for multiple foci because it allowed them to see multiple parts and part families simultaneously. For example, they felt that it would be potentially helpful to see the part intended to be used with the one that the operator is designing. One of the CAD operators said: "...seeing the dimensions on the part makes things a little clearer, especially when looking at mating parts."

In addition to the above advantages, the CAD operators also provided us some constructive suggestions on how we could improve the hypermenu user interface. As we discussed in the previous chapter, there is a problem when available screen space is exhausted. They agreed with the approach of using smaller representations to conserve screen space. Therefore, we suggested providing each cluster a zoom state showing only the part code. The user might shrink the part families far above the current level to show only the part codes. The CAD operators' immediate response was they only need the graphic representation at the current and the immediate parent level. They also felt that the rest of the hierarchical abstraction could shrink into textual descriptions without degrading the context. In addition, they told us that they seldom refer to objects far away from the current foci. They would rather have bigger hierarchical abstractions for objects of current interest and neighboring objects. Most importantly, they believe that omitting intermediate levels in the hierarchy would not significantly degrade the context.

The developers of the original user interface for the BSTS gave evaluations analogous to the CAD operators when they reviewed the BSTS application. In short, they too appreciated the capability to see detail in context and the capability to help the user visualize hierarchical structures with the hypermenu user interface. They also agreed that outline animation was effective in letting the user follow changes in the context.

Similarly, the BSTS developer provided us some constructive suggestions on potential improvements to the hypermenu user interface. First, he pointed out that context is needed for navigation. However, when the user is focusing on problem solving, the context is no longer needed. Unneeded context consumes screen space and tends to distract the user. According to the BSTS developer, the user needs to see only the testers and sessions during startup, the selected instrument and subsequent devices during configuration, and the related devices during examination. This suggests the idea of "usage context". A usage context would consist of task-specific context elements only. It could be achieved by displaying only task-related components or omitting unneeded levels and items. To provide this capability in the hypermenu approach, the usage context would need to be encoded into the user interface so that it could display the proper task-dependent context.

Second, the user often needs to put objects of interest immediately side by side regardless of how far apart they are located in hierarchy. In other words, the hypermenu user interface should allow arbitrary repositioning of objects without altering the hierarchy. Repositioning may also permit more efficient use of screen space since the size of a parent depends on the positions of its child components. Third, the BSTS developers suggested that it would be useful to allow hypermenu components to overlap as in the traditional window environment. Combined with repositioning, the users could arbitrarily expose information by controlling overlapping areas. It also takes less screen space to display overlapped objects. The BSTS developers suggested that overlapping is permissible because modern computer users are accustomed to deal with overlapping in traditional windows environment. However, allowing repositioning and overlapping could reduce the benefits of preserving the spatial relationships among on-screen objects and require more work on the user's part which hypermenu has eliminated. Finally, it was reported that a senior engineer and researcher in the company wondered if the dramatic differences between the hypermenu user interface and the traditional window environment might hinder its acceptance by change-resistant users.

To summarize, knowledgeable users in each of the two applications felt the hypermenu approach was a useful improvement to navigational issues in the respective user interfaces. Additionally, the evaluators suggested developing ways to reduce and/or eliminate unneeded context, a recommendation we feel is worth investigating.

5.2 Discrete Zoom verses Continuous Zoom

For the context management scheme of our hypermenu approach, we could have used the global version of continuous zoom. However, we developed the discrete zoom for the following reasons. First, we would like to support hierarchical systems whose components can be more efficiently represented by a fixed number of size-dependent representations. BSTS belongs to this category. Switching among fixed-size representations consumes significantly less computation than real-time scaling. The only drawback is that the user has less control over sizing. The continuous zoom, in contrast, allows the user to zoom a component to any arbitrary size. It is designed for hierarchical systems whose components have a size range instead of a fixed number of sizes. Support for arbitrary sizes requires significantly more computation than switching among fixed-size representations, but it provides more control over sizing. Second, we would like to reduce the cognitive effort during zooming. The discrete zoom takes only one step to zoom from one size-dependent representation to another. Unlike the continuous zoom, the discrete zoom requires no constant tracking on object size during zooming, thus reducing cognitive effort. Finally, we do not want to affect the sizes of other components other than the one being zoomed. Unfortunately, zooming a component in the global version of continuous zoom affects the sizes of all other components in the hierarchy.

CHAPTER 6

SUMMARY

6.1 Summary For Our Hypermenu Approach

Computer applications are increasingly driven by graphical user interfaces; many of these applications function like "menu applications", allowing user to access functions from the underlying work object. As the complexity of hierarchical systems increases, traditional user interface approaches become an **interaction bottleneck**.

Several new technologies have been proposed to improve the user interface, especially the menu mechanism and the work object representation to facilitate interaction and preserve user orientation. Improved menu mechanisms include pie menus, partial-screen and full screen menus, see-through menus, documents as user interface, and embedded menus. Improved screen layout technology includes fisheye, continuous zoom, perspective wall, cone trees, hyperbolic surface, tree-map and Pad++. All of them have met with varying degree of success. After reviewing and evaluating these current technologies, we find that a new technique is needed to facilitate interaction for complex hierarchical systems.

This thesis has described a new approach to integrate the function access mechanism and the work object representation based on an extension to SFU's Intelligent Graphic Interface research. Our new technology, called a **hypermenu**, addresses problems with existing technologies and provides the following advantages:

- It preserves the global context and user orientation at all time;
- It facilitates visualization and hierarchical management of the underlying system by letting the user control the amount of context.
- It allows the user to interact with system components through direct manipulation.
- It provides a transparent menu mechanism with minimum user interface overhead;
- It provides hierarchical abstractions and size-dependent representations;
- It allows simultaneous viewing of multiple areas of interest.

To offer the above advantages, our hypermenu approach uses a hierarchy and a context management scheme. Each hypermenu component in the hierarchy is a *combined representation* of function access mechanism and a corresponding component in the work object representation. The context management scheme organizes the on-screen layout and appearance of hypermenu components. By integrating the function access mechanism and the work object representation, our hypermenu approach minimizes the need for a separate menu mechanism and facilitates interaction. Ideally, the hypermenu approach turns an application into its own menu.

We have implemented two hypermenu applications for evaluation. One was designed for computer-aided-design software based on Group Technology. The other was designed for the user interface of a telecommunication network testing system. Both are written in C++ and run under standard X/Windows and Motif.

Our hypermenu approach reduces user interface overhead by integrating the work object representation with the function access mechanism, so that the user can interact with the underlying system via direct manipulation. In addition, it also provides a full view of the work context to preserve the user's orientation and hence facilitates navigation.

6.2 Further Research Directions

In our future research to improve the hypermenu approach, we would focus on : (1) providing support for dynamic determination of abstractions, (2) allowing simultaneous display of hypermenu components belong to multiple hierarchies, and (3) exploring more efficient zoom algorithms.

In many applications, abstractions for hypermenu components depend on a lot of parameters. Dynamic determination of abstraction based on all these parameters may produce a more meaningful and informative abstraction to the user.

The hypermenu components may be organized into different hierarchies depending on the needs of an application. For example, mechanical parts belong to different hierarchies when they are organized by shape, manufacturing process and other properties. Very often, manufacturers need to look at these hierarchies simultaneously. Therefore, it would be useful to display hypermenu components belonging to multiple hierarchies simultaneously.

As illustrated in our hypermenu applications, screen space required to display the underlying system is proportional to the depth of the hierarchy and the number of items within each level. Navigation is limited by the amount of screen space available. At present, the discrete zoom and the continuous zoom are interval-based algorithms and could induce large zoom holes. The unoccupied portion of a zoom hole wastes screen space. Continual search for more efficient zoom algorithms will be a focus of future researches.

APPENDIX

The Hypermenu Approach To The User Interface Of HP's Test System

This appendix describes our second hypermenu approach to a user interface for the HP telecommunication network testing system[1] in the following order: introduction to HP's test equipment, and the hypermenu application for HP's test equipment.

A.1 Introduction to HP's test equipment

A.1.1 Telecommunication network Concepts

A telecommunication network is used to transmit voice and data. A network is logically represented by nodes and links. Voice and data may enter and leave a node. A node is the only place in a network where voice and data may originate. A link connects two nodes. Voice and data travel through a link from one node to another.

[1] S.F.U. researchers involved in the hypermenu interface include Dr. John Dill, Mr. Frank Henigman and Albert Chan. In the beginning of the project, S.F.U. researchers worked with HP IDACOM personnel to become familiar with the original BSTS user interface and to develop a framework for a hypermenu approach.

Transmitting information is a five step process [Vetter 95]. First, information is sent out from an end-to-end application, such as Local Area Network management software, to a service layer. Second, the service layer passes the information to the adaptation layer for transmission. Third, the adaptation layer breaks down the information and packages it into cells, where a cell consists of a header and an information field, The header field contains information on identification and destination. Fourth, a convergence layer accepts the cells and maps them to the physical layer. Finally, the physical layer encodes the data into electrical waveforms for transmission. Each layer provides a variety of options. For example, the adaptation layer may package cells into ATM or SIP L2 formats. Combination of all the layers and the communication format of the end-to-end application defines a network transmission protocol.

A.1.2 HP Test Equipment Concepts

HP's portable test equipment [HP 93], also called Broadband Series Test System (BSTS), is designed to evaluate the performance of a network in transmitting data from one node to another. When connected to a network, it measures performance indicators such as cell transit delay and transmission errors. The test equipment houses a number of instruments and may be used to test both nodes and links in the network. When testing a node, the user needs only one instrument. When testing a link, the user needs two instruments each connected to a terminating node.

To conduct a test, the user first defines a *session*, which is the process of setting up instruments for a test and running the test. Once a session has been defined, the user next specifies the instrument used to make the performance measure. To build an instrument, the user specifies the protocol and one or two devices to be used for the test. To facilitate building instruments, an instrument builder is provided. The instrument builder lets the

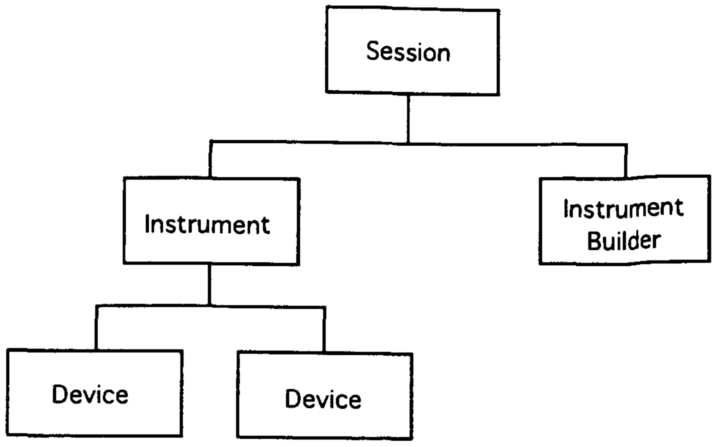


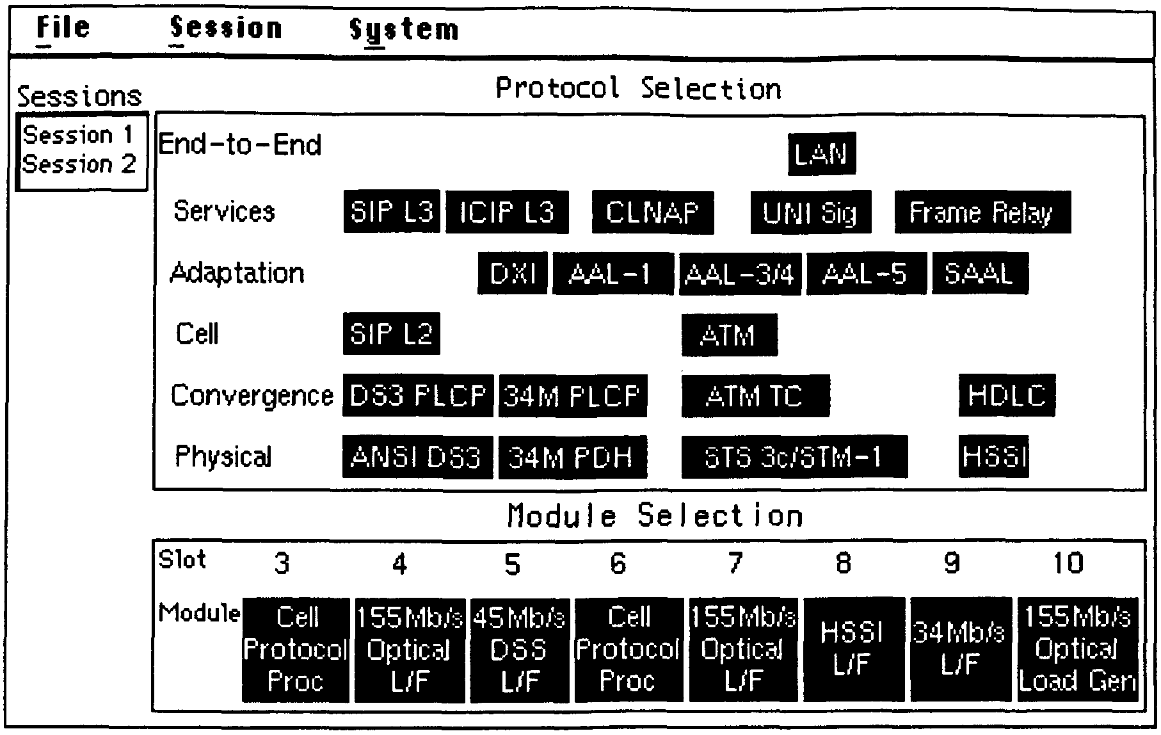
Figure A.1.2.1: Logical entities in HP's BSTS.

user specify the protocol and device(s) to be used. Finally, the user specifies device parameters for each device inside the instrument. Figure A.1.2.1 shows the relationships among all entities. Tasks the user must frequently undertake include: (1) set up the test equipment, (2) examine test results, and (3) compare results from different devices.

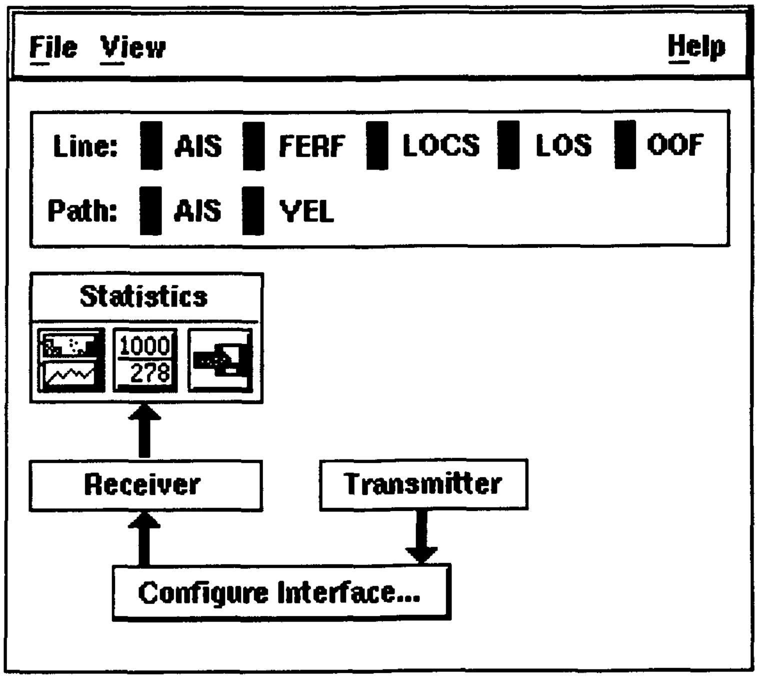
A.1.3 Existing BSTS User Interface

BSTS adopts a graphical user interface that provides the user access to and control of its hardware, which is the work object. In the existing graphical user interface, the work object is represented by dialog boxes corresponding to session, instrument, instrument builder, and devices. Collectively, they become the menu system for BSTS.

HP originally used traditional user interface components such as dialog boxes and pull-down menus for its graphical user interface. Each device is represented by a collection of dialog boxes containing all available functions. Figure A.1.3.1 shows two dialog boxes that represent some of the entities shown in Figure A.1.2.1. Figure A.1.3.1(a) shows both session and instrument builder in the same dialog box. Session names are displayed in the



(a)



(b)

Figure A.1.3.1: Dialog boxes representing some entities of HP's test equipment. (a) Instrument builder dialog box includes session panel on upper-left corner. (b) First-level dialog box for line interface module.

upper left opening of the dialog box. An instrument builder contains panels for both protocol selection and module selection. All the entities inside the module selection panel are devices. Labels for devices are truncated because of limited screen space. For example, the label of a cell protocol processor is "Cell Protocol Proc" and the labels for lines interfaces end with "L/F".

Figure A.1.3.1(b) shows the first level dialog box for a cell protocol processor. Buttons are grouped to represent function sets provided by different elements from the device. Pressing some of the buttons brings up still other dialog boxes. A similar dialog box is used to represent a line interface module.

To conduct a test, the user first gives the session a name in order to define it. Once this is completed, the name of the session is displayed inside the session panel. Then the user specifies an instrument from the instrument builder. To specify an instrument, the user first selects an appropriate protocol path from the protocol selection panel. Then, he or she selects appropriate devices from the module selection panel. Once an instrument is specified, the instrument builder dialog dims out all irrelevant choices. To specify device parameters, the user brings up corresponding dialog boxes by double clicking on the device in the module selection panel.

A.2 Hypermenu Prototype for HP's Test Equipment

In this section, we discuss some major design objectives of HP's user interface and the construction of the hypermenu prototype for HP's test equipment. Then, we describe in detail how our hypermenu interface functions.

A.2.1 Design Objectives Of The User Interface

HP's goals for improving the quality of BSTS's user interface are: (1) preserve the user's work context and orientation, (2) reflect relationships among system components, (3) allow hierarchical management of the user interface and work context, (4) provide effective screen layout for the work context, especially components of interest, (5) reduce user interface overhead to increase user throughput, (6) permit quick and accurate component access, and (7) indicate configuration sequence.

The following outlines our reason for believing the hypermenu approach is appropriate for improving BSTS's user interface. First, the hypermenu's context management scheme preserves the work context at all times, while reflecting the hierarchical relationships of its components for HP's test equipment. Second, with support for hierarchical abstractions and the capability to provide or remove details on demand, the hypermenu approach allows hierarçhical management of the work context. For example, the user may remove unneeded detail by closing a higher-level component. Third, displaying size-dependent representations provides an effective way for the user to examine areas of interest depending on the amount of screen space available. Also, the hypermenu components never overlap one another. Fourth, the hypermenu approach reduces user interface overhead by combining the work object and the user interface. Thus, the user may interact more quickly and accurately with direct manipulation on the work object. For HP's test equipment, the user directly interacts with dialog boxes that corresponds to hypermenu components. Finally, we can use the hypermenu's hierarchical structure to support configuration for HP's test equipment. A level in the hypermenu hierarchy corresponds to a configuration step. Each hypermenu component in a given level represents a valid choice. In this way, the user goes through the configuration by navigating down the hypermenu hierarchy. Again, the hypermenu's context management scheme provides the work context during the entire configuration process.

To further facilitate hierarchical management of BSTS's user interface, HP adds a test manager and a tester in addition to the components shown in Figure A.1.2.1. A test manager maintains two testers and each tester contains two sessions. Both entities are designed solely for hierarchical management. Recall that a session accommodates a maximum of two instruments, so that the user can test a link in a single session.

A.2.2 Constructing A Hypermenu Prototype for HP's Test Equipment

To construct a hypermenu for HP's test equipment, we first need to identify the work object representation. The original work object representation is made up of the entities shown in Figure A.1.2.1. Test manager, testers, sessions, and instruments become cluster hypermenu components, while devices are leaf hypermenu components. In our hypermenu prototype, we slightly modified the work object to take full advantage of the hypermenu's features. First, we added a representation under each instrument to indicate the selected protocol. Therefore, in addition to devices, an instrument also contains a selected protocol map. Second, we replace the concept of an instrument builder with an undefined instrument. Hereafter, an instrument is either defined or undefined. Opening a defined instrument exposes all its devices and the selected protocol map. In contrast, an undefined instrument is always closed and has one zoom state represented by a dialog box that provides similar functions to an instrument builder.

Once we have completely identified the work object and all subsequent components, we insert them into our hypermenu hierarchy. As shown in Figure A.2.2.1, test manager, tester, and session are made into cluster hypermenu components. Figure A.2.2.2 shows

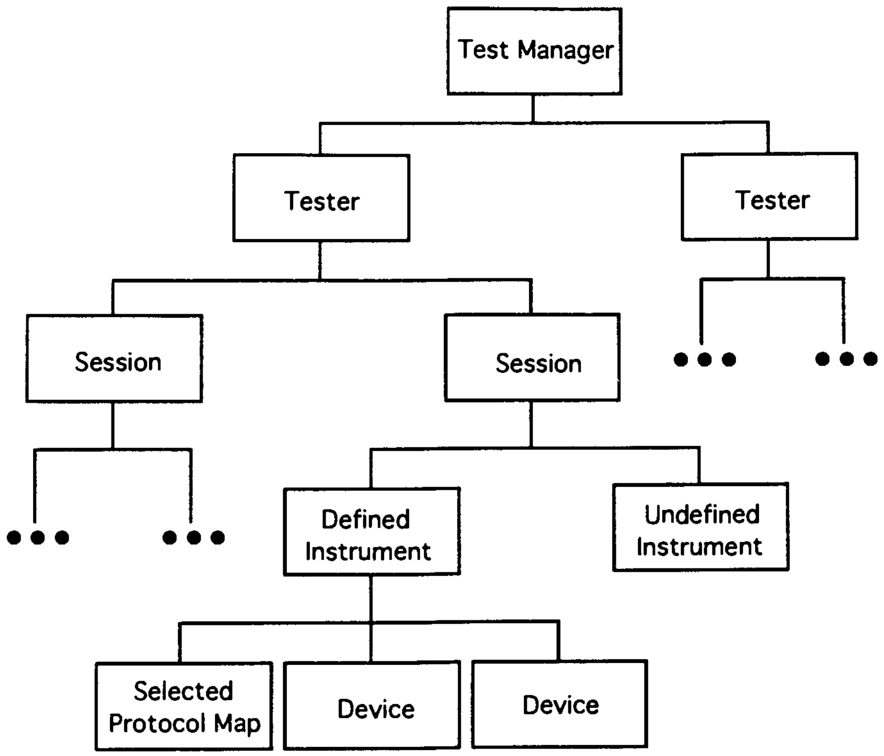


Figure A.2.2.1: The hypermenu hierarchy constructed for HP's test equipment. Note that the notion of "undefined instrument" replaces the previous instrument builder. Thus, instruments are either defined or undefined.

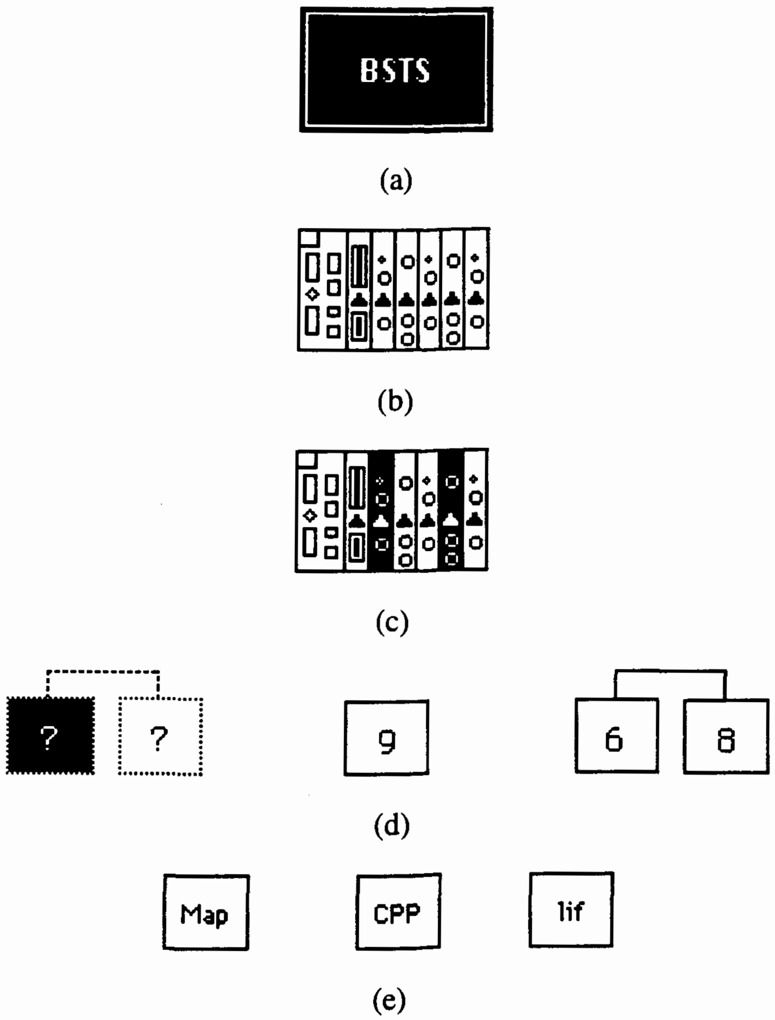
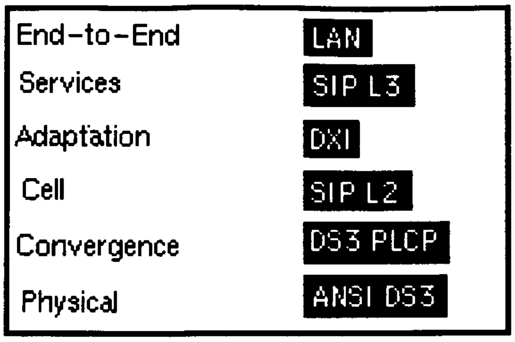


Figure A.2.2.2: Abstraction representations. (a) A test manager, (b) A tester, (c) A session, note highlight, indicating some devices have been selected. (d) Instruments: from left to right: an undefined instrument, an instrument with one device occupying slot nine, an instrument with two devices occupying slot six and eight. (e) From left to right: a selected protocol map, a cell protocol processor, a line interface.

the closed representation for all the hypermenu components. The close abstractions for the test manager, tester and session need to convey only their identification. Thus, they remain unchanged at all times. However, this is not true for an instrument. When an instrument is undefined, it is a leaf hypermenu component; but once an instrument is defined, it becomes a cluster hypermenu component. Since an instrument is composed of either one or two devices, different abstractions are needed to represent the number of devices. In fact, the closed representation of an instrument also needs to report the number of devices and slot numbers they occupied in the instrument. Figure A.2.2.2(d) shows a few sample abstractions for an instrument.

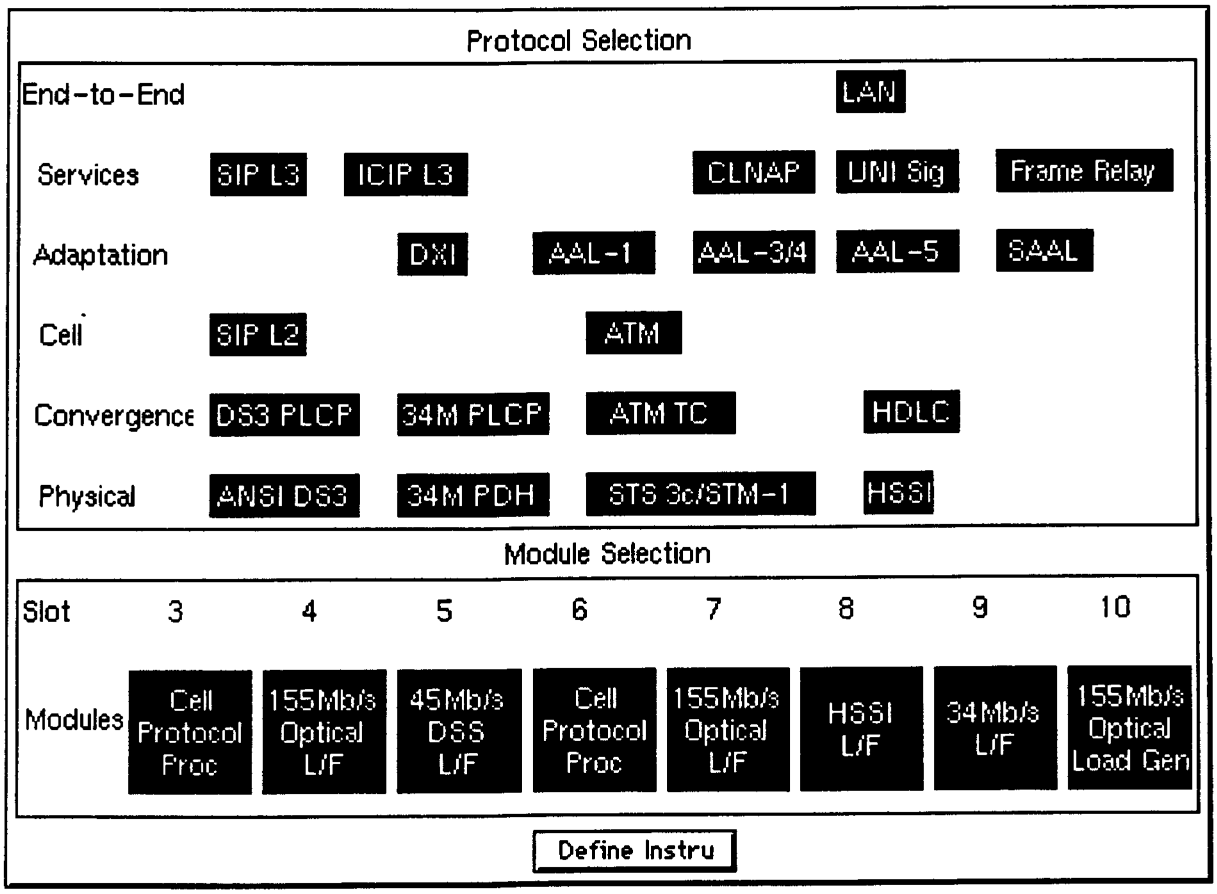
Note that an instrument's closed representation needs to change dynamically to update its status. When a cluster hypermenu component is opened, it is drawn as a filled rectangle. Additionally, some leaf hypermenu components have only one zoom state. Among them are the selected protocol map and the undefined instrument (See Figure A.2.2.3).

Our hypermenu prototype supports size-dependent representations for all the devices. Each device has two zoom states with different sizes and representations. According to HP, a device only needs to be fully opened for configuration. Otherwise, it only needs to display component status for examination. In general, it consumes less screen space to display component status than the entire dialog for configuration. With support for size-dependent representation, the hypermenu approach facilitates optimal use of screen space. Figure A.2.2.4 shows all the size-dependent representations for the cell protocol processor and line interface modules.



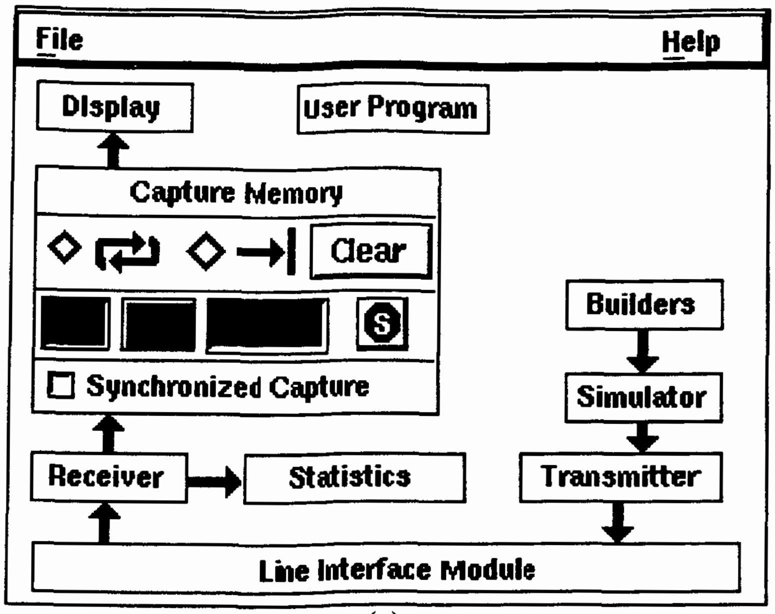
End-to-End	LAN
Services	SIP L3
Adaptation	DXI
Cell	SIP L2
Convergence	DS3 PLCP
Physical	ANSI DS3

(a)

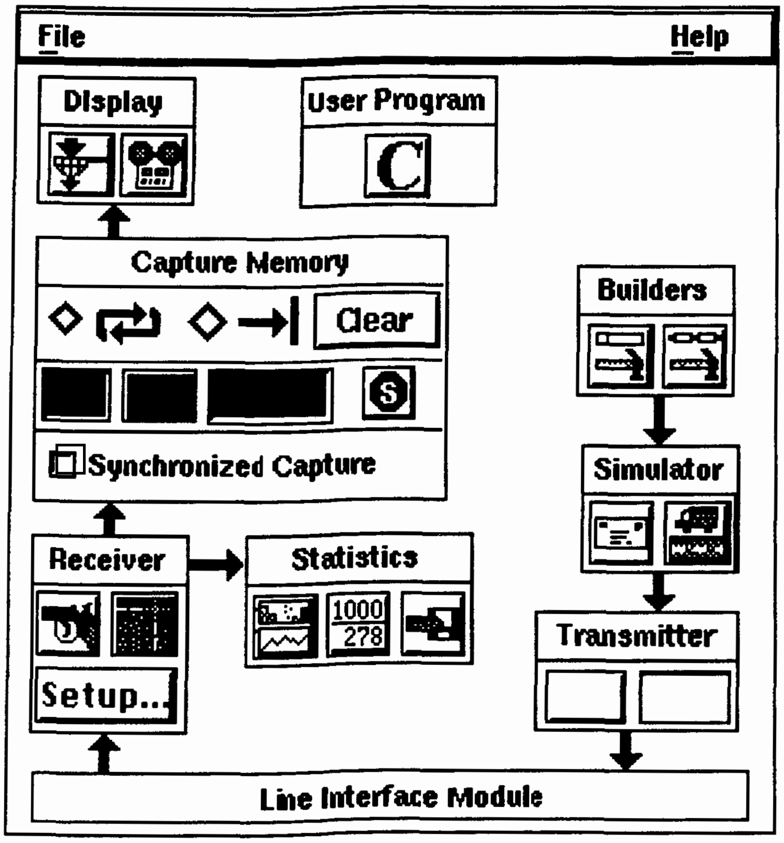


(b)

Figure A.2.2.3: Abstraction representations. (a) Open abstraction of a selected protocol map. (b) Open abstraction of an undefined instrument.

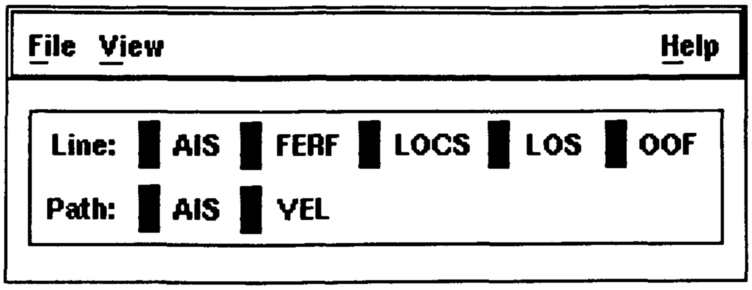


(a)

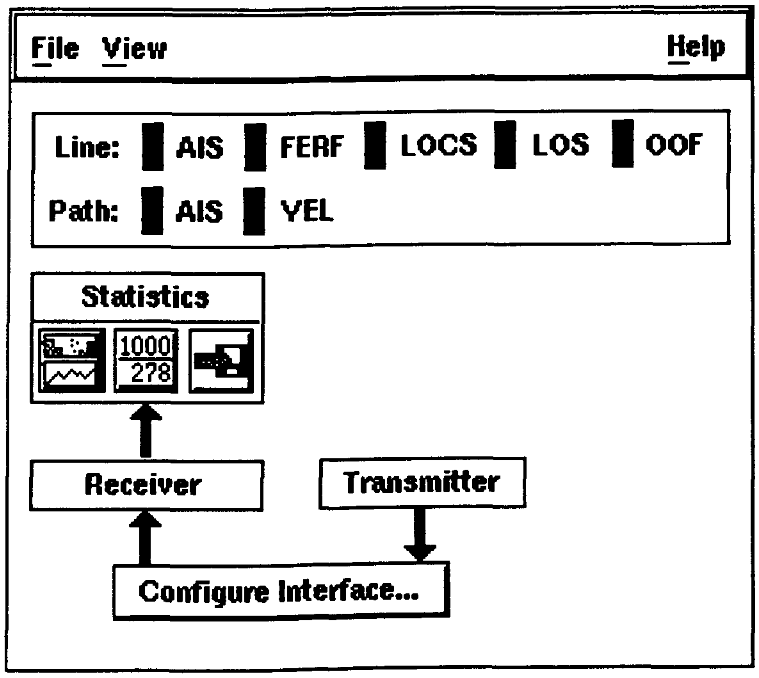


(b)

Figure A.2.2.4: Abstraction representations. (a) Initial open abstraction of a cell protocol processor module.(b) Fully opened abstraction for a cell protocol processor module.



(c)



(d)

Figure A.2.2.4 (continued): (c) Initial open abstraction for a line interface. (d) Fully opened abstraction for a line interface.

A.2.3 Behaviors Of The Hypermenu Interface

To illustrate the hypermenu approach to a BSTS user interface, we show how a user can perform the two most common tasks with the hypermenu interface. First, we will show how a user can build an instrument and then interact with its components. Second, we will show a scenario where the user examines and compares multiple devices simultaneously.

A.2.3.1 Defining And Configuring An Instrument

Defining and configuring an instrument is a very tedious decision-making process. According to HP developers, users tend to be confused during the lengthy process. Thus, preserving the configuration sequence is extremely important to avoid confusion and disorientation. To define an instrument with our hypermenu prototype, the user takes the following steps:

(1) Launch the software. Initially, the hypermenu interface shows the closed test manager, which can be the product logo (See Figure A.2.3.1.1(a)).

(2) Open the test manager. An opened hypermenu component is drawn as a filled rectangle. Each level in the hypermenu hierarchy is assigned a unique color to indicate the depth of a component in the hierarchy. In the context of configuration, opening a hypermenu component immediately exposes all its sub-components which correspond to the next set of choices. Opening the test manager exposes two testers as shown in Figure A.2.3.1.1(b). The hypermenu interface animates all size changes during a basic operation to provide·visual continuity. Throughout configuration, the user selects a choice by opening or zooming it directly.

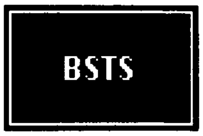
(3) Selecting and opening a tester exposes two sessions within. (See Figure A.2.3.1.1(c)).

(4) Selecting and opening a session exposes two initially undefined instruments (See Figure A.2.3.1.1(d).

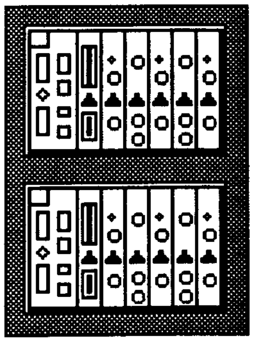
(5) Selecting and zooming in an undefined instrument (See Figure A.2.3.1.1(e)). At this stage, the user selects proper parameters from the protocol and module selection panels. Pressing "Define Instru" defines the instrument. Immediately afterwards, the hypermenu interface zooms out the instrument and displays the proper abstraction. Figure A.2.3.1.1(f) shows an example where the user has selected two devices occupying slot six and eight. Defining an instrument involves creating hypermenu components that contain the selected protocol map and the composing devices.

(6) If further configuration is needed, the user can open an instrument and get into lower levels of details. When a defined instrument is opened, all its components are initially in closed state (See Figure A.2.3.1.1(g)). To configure a device, the user needs to zoom it in fully in order to access all details. Figure A.2.3.1.1(h) shows a fully zoomed-in cell protocol processor module.

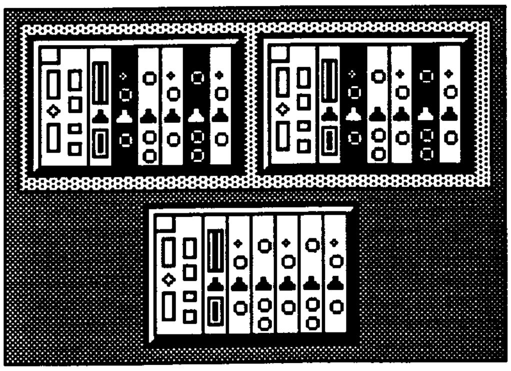
If a mistake is discovered after defining an instrument, the user can delete the instrument and start over again (See Figure A.2.3.1.1(d)).



(a) A closed test manager.

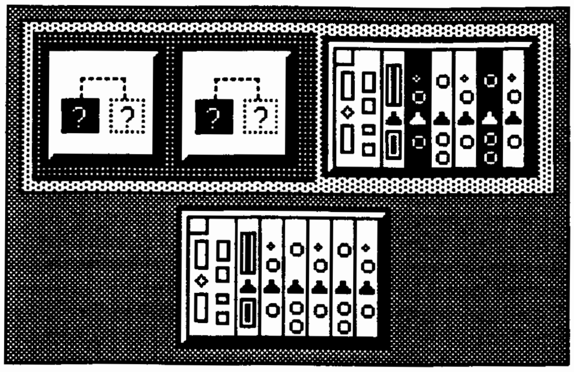


(b) Open a test manager to see two testers.

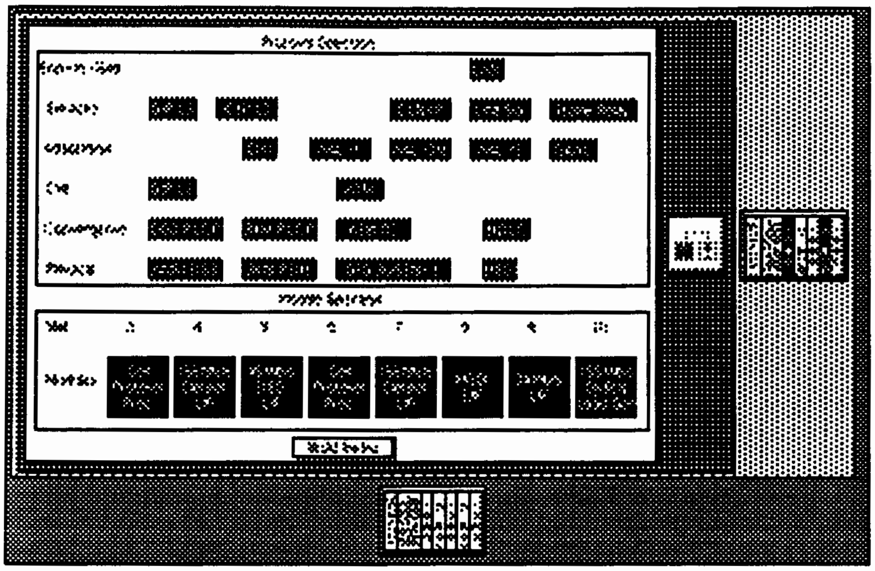


(c) Open a tester to see two sessions.

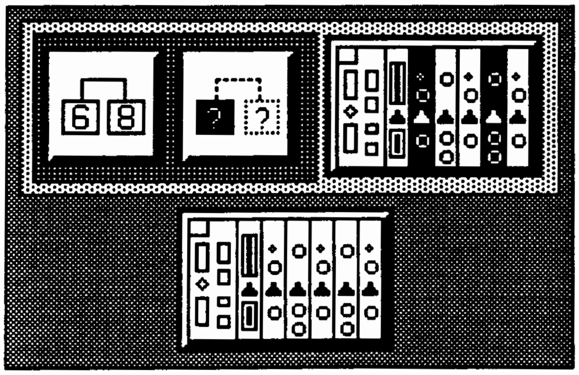
Figure A.2.3.1.1: Configuration using the hypermenu interface.



(d) Open a session to see two undefined instruments.

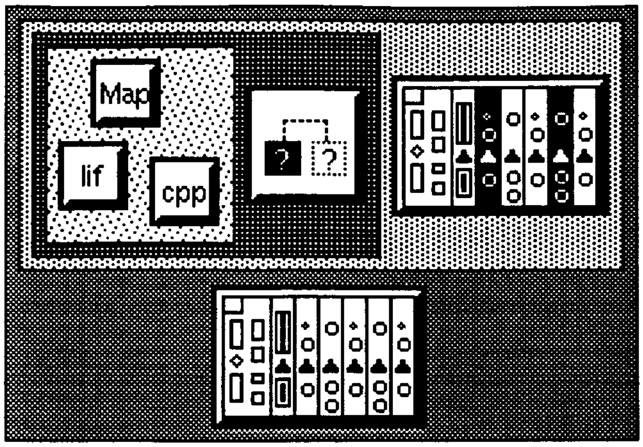


(e) Open undefined instrument to select protocol and modules.

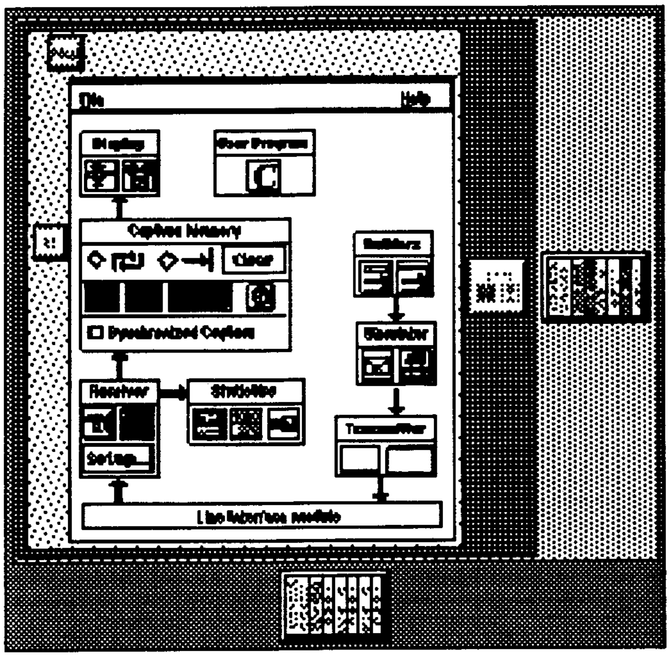


(f) Devices in slots 6 and 8 defined in the leftmost instrument.

Figure A.2.3.1.1: (continued).



(g) Opening a defined instrument shows the selected protocol map and device(s) within.



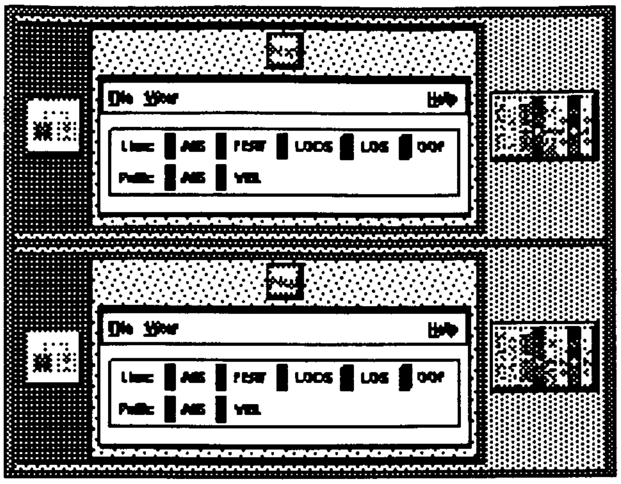
(h) Instrument cpp opened for parameter adjustment.

Figure A.2.3.1.1: (continued).

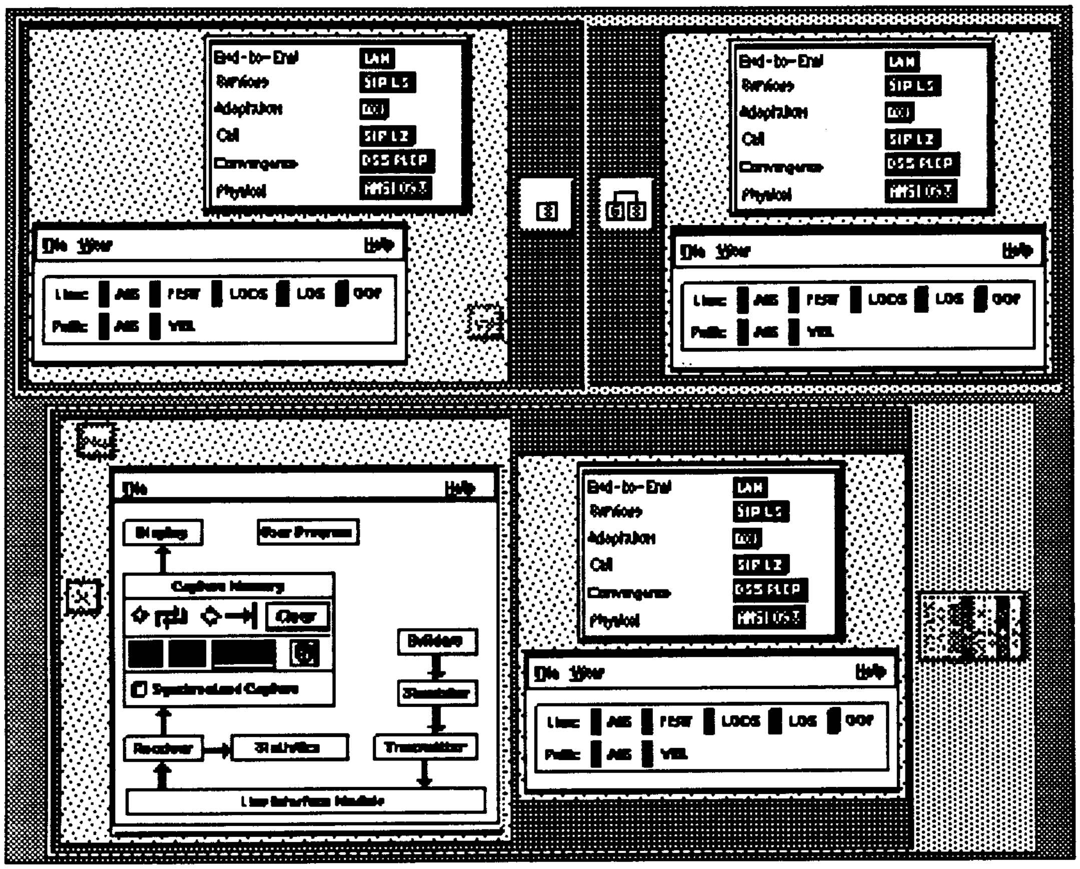
4.2.3.2 Examining And Comparing Hypermenu Components

There are times when the user needs to examine and compare components in different testers or test sessions. Such situations arise when multiple instruments are needed to test different parts of a network. During configuration, the user may want to place components side by side for identical settings. During examination, the user may need to look at multiple components simultaneously. In both situations, displaying all the components of interest is a primary concern of the user.

Since the discrete zoom algorithm supports multiple foci, the hypermenu interface is capable of displaying multiple components in detail as long as there is enough screen space. To examine a device, the user only needs to zoom into it. Figure A.2.3.2.1(a) shows a scenario where two devices have been zoomed in. All other components and filled rectangles constitute the work context, which reflects relationships among components and the entire system. With the work context in full view, the user can identify where a component belongs and the chance of confusion is reduced. Moreover, since hypermenu components do not overlap one another, every component on the display is fully visible to the user. In case of insufficient screen space to display a component in detail, the user can zoom out or close other components to increase available screen space. Alternatively, the user may enlarge the hypermenu's application window as much as the display allows. Figure A.2.3.2.1(b) shows a scenario where four devices and three protocol maps are simultaneously zoomed in. When shrinking its application window, the hypermenu interface automatically closes all hypermenu components to prevent any of them from lying outside the application window.



(a)



(b)

Figure A.2.3.2.1: Examine system components.

REFERENCES

[Autodesk 91] Autocad Reference Manual.

[Barnard 77] Barnard, P.J., Morton, J., Long, J. and Ottley, E.A. Planning Menus For Display: Some Effects of Their Structure on User Performance. International Conference on Displays for Man-Machine Systems, pp.130-133, April 1977.

[Bartram 95] Bartram, L., Henigman, F. and Dill, J. The Intelligent Zoom as Metaphor and Navigation Tool in a MultiScreen Interface for Network Control Systems. IEEE International Conference on Systems, Man and Cybernetics, Oct. 22-25, pp.3122-3127, 1995

[Bederson 94] Bederson, B. and Hollan, J. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. Proceedings of the ACM Symposium on User Interface Software and Technology, pp.45-53, 1994.

[Bier 90] Bier, E.A. and Goodisman A. Documents as user interfaces. Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, pp.249-262, 1990

[Bier 91] Bier, E.A. EmbeddedButtons: Documents as User Interfaces. Proceedings of the ACM Symposium on User Interface Software and Technology, pp.45-53, 1991.

[Bier 93] Bier, E.A., Stone M.C., Pier, K., Buxton, W. and DeRose, T. D. Toolglass and Magic Lenses: The See-Through Interface. COMPUTER GRAPHICS Proceedings, pp.73-80, 1993.

[Callahan 88] Callahan, J., Hopkins, D., Weiser, M. and Shneiderman, B. An empirical comparison of pie vs. linear menus. Proceedings of Computer Human Interaction, pp.95-100, 1988.

[Card 91] Card, S.K., Robertson, G.G. and Mackinlay, J.D. The Information Visualizer, an Information Workspace. Proceedings of Computer Human Interaction, pp.181-188, 1991.

[Dill 94] Dill, J., Bartram, L., Ho, A. and Henigman, F. A Continuously Variable Zoom for Navigating Large Hierarchical Networks. IEEE International Conference on Systems, Man, and Cybernetics, pp.387-390, 1994.

[Foley 90] Foley, J., van Dam, A., Feiner, S. and Hughes, J. Computer Graphics PRINCIPLES AND PRACTICE, second edition, 1990.

[Furnas 86] Furnas, G.W. Generalized Fisheye Views., Proceedings of Computer Human Interaction, pp.16-23, 1986.

[Groover 80] Groover, M.P. Automation, Production Systems, and Computer-Aided Manufacturing, 1980.

[Hopkins 91] Hopkins, D. The Design and Implementation of Pie Menus. Dr. Dobb's Journal, pp.16-26, 1991.

[HP 93] Hewlett Packard 75000 Broadband Series Test System brochure .

[Johnson 91] Johnson, B. and Shneiderman, B. Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. IEEE Conference on Visualization, Oct. 22-25, pp.284-291, 1991.

[Kiger 84] Kiger, J.I. The depth/breadth trade-off in the design of menu-driven user interfaces. International Journal of Man-Machine Studies, 20, pp.201-213, 1984.

[Koved 86] Koved, L. and Shneiderman, B. Embedded Menus: selecting items in context. Communications of the ACM, Vol. 29, No. 4, pp.312-318, April 1986.

[Kurtenbach 93] Kurtenbach, G. and Buxton, W. The limits of Expert Performance Using Hierarchical Marking Menus. Proceedings of Computer Human Interaction, pp.482-487, 1993.

[Lamping 95] Lamping, J., Rao, R. and Pirdli, P. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Proceedings of Computer Human Interaction, pp.401-408, 1995.

[Landauer 85] Landauer, T. Selection from alphabetic and numeric menu trees using a touch screen: Breadth, depth and width. Proceedings of Computer Human Interaction, pp.73-78, 1985.

[Landauer 87] Landauer, T. Relations between Cognitive Psychology and Computer System Design. Interfacing Thought: Cognitive Aspects of Human-Computer Interaction. pp.1-25, 1987.

[Mackinlay 91] Mackinlay, J.D., Robertson, G.G. and Card, S.K. The Perspective Wall: Detail and Context Smoothly Integrated. Proceedings of Computer Human Interaction, pp.173-179, 1991.

[McDonald 83] McDonald, J., Stone, J. and Liebelt, L. Searching for items in menus: The effects of organization and type of target. Proceedings of the 27th Annual Meeting of the Human Factors Society, pp.834-837, 1983.

[Miller 81] Miller, D.P. The depth-breath tradeoff in hierarchical computer menus. Proceedings of the 25th Annual Meeting of the Human Factors Society, pp.296-300, 1981.

[Miller 56] Miller, G.A. The Magic Number seven, Plus or Minus Two, Psychological Review, 63, pp.81-97, 1956.

[Nielsen 90] Nielsen, J. The Art of Navigating HYPERTEXT. Communications of the ACM, Vol. 33, No. 3, pp.296-310, March 1990.

[Noik 94] Noik, E. A Space of Presentation Emphasis Techniques for Visualizing Graphs, Graphics Interface '94, pp.225-233, 1994.

[Perlman 84] Perlman, G. Making the Right Choices with Menus. Proceedings of INTERACT 84, pp.317-321, 1984.

[Robertson 91] Robertson, G.G., Mackinlay, J.D. and Card, S.K. Cone trees: animated 3D visualizations of hierarchical information. Proceedings of Computer Human Interaction, pp.189-194, 1991.

[Sarkar 94] Sarkar, M. and Brown M.H. Graphical Fisheye Views of Graphics. Proceedings of Computer Human Interaction, pp.73-84, 1994.

[Schaffer 93] Schaffer, D., Zuo, Z., Bartram, L., Dill, J., Dubs, S., Greenberg, S. and Roseman, M. Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks. Graphics Interface '93, pp.87-96, 1993.

[Snead 89] Snead, C. S. Group Technology Foundation for Competitive Manufacturing, 1989.

[Snowberry 83] Snowberry, K., Parkinson, S.R. and Sisson, N. Computer display menus. Ergonomics, v.26, pp.699-712, 1983.

[Tullis 85] Tullis, T., Designing a Menu-based Interface to an Operating System, Proceedings of Computer Human Interaction, pp.79-84, 1985.

[Vetter 95] Vetter, Ronald J. and Du, David H.C. Issues and Challenges in ATM Networks. Communications of the ACM. Vol. 38, No. 2, pp.31-38, February 1995.

[ZiZi 94] ZiZi, Mountaz and Beaudouin-Lafon, Michel. Accessing Hyperdocuments through Interactive Dynamic Maps. ECHT '94 Proceedings. pp.126-135, September 1994.