A Menu Interface Development Environment Based on Lean Cuisine

A thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at Massey University

Jianpeng Gu

1995

Abstract

The integrated user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] is a combination of software tools used to support user interface development from initial design, rapid prototyping through to direct implementation. This thesis describes the development of three software tools used in the integrated user interface development environment.

The Lean Cuisine graphical editor (Elc) provides an interactive design environment for graphical specifications of menu-based interfaces and shows that the Lean Cuisine notation described in [Apperley & Spence, 89] has been implemented in a practical computer environment as an interactive interface design tool.

The user interface simulator (Slc) is a very effective and reliable interface simulating and testing tool which supports quick and convenient user interface simulation. Using Slc, a menu interface can be quickly simulated in its design environment, where a menu-based interface can be partially or wholly simulated and invalid menu structures can be dynamically modified, or in its application environment, where evaluators are given a real feel of how this menu-based user interface works.

The user interface generator (Glc) is used to generate basic interface source code files for a user interface from its Lean Cuisine graphical specification file, and a working model of a user interface can be easily and quickly implemented without programming.

The integrated user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] successfully integrates a graphical notation, the visual programming technique, with an existing programming toolkit and offers advantages over other User Interface Programming Toolkits, language-based UIMSs and current Visual Programming Tools. It supports three main phases (design, prototyping and implementation) of the graphical user interface development lifecycle. This approach has not been found in previous user interface development tools and user interface management systems.

Acknowledgments

I would like to thank my supervisor Professor Mark Apperley for his guidance, encouragement and help during my study.

Thanks also extend to all staff and postgraduate students in the Department of Computer Science for their assistance and friendship.

On the personal side, I would especially like to thank my wife, Bei Zhong, and my family for their continued support and encouragement.

Table of Contents

Chapter 1

Introduction

1.1. User Interface Development

Creating a good user interface for a system is a difficult task, and the software to support user interfaces is often large, complex, and difficult to debug and modify [Myers, 88]. Current techniques for the specification and development of user interfaces can be divided into three basic groups: User Interface Programming Toolkits, User Interface Management Systems (UIMSs) or User Interface Development Systems (UIDSs), and Visual Programming Tools. Although many tools have been created to make user interfaces easier to design and implement, their development is still a hard work. It is generally the case that as user interfaces become easier to use for the end user, they become more complex and harder to create for the user interface creator. Direct-manipulation user interfaces are particularly difficult to create because they often provide elaborate graphics, many ways to give the same command, a mode free interface, and rapid semantic feedback [Myers, 89].

1.2. The Limitation of Current Approaches

Many solutions for finding better methods of creating good user interface have been proposed in the areas of User Interface Programming Toolkits, User Interface Management Systems(UIMSs) and Visual Programming Tools. Many tools have been created to make user interfaces easier to design and implement but each method has its own attendant problems.

1.2.1. User Interface Programming Toolkits

User Interface Programming Toolkits such as Motif [Young, 89], Macintosh Toolbox [Apple, 85], OpenLook [Sun, 90A], and XView [Heller, 90], provide a conventional programming interface and these programming toolkits are widely used for user interface construction. With all toolkits, the user interface designer writes programs in a conventional programming environment to create the user interface. The disadvantages of using toolkits are that they provide limited interaction styles, lack of support for easy prototyping, difficulties in ease of use at least as great as the base language, no abstraction mechanisms, no path of communication between the user interface designer and the end user [England, 90], and they cannot be used by non-programmers.

1.2.2. UIMSs

The problems with User Interface Programming Toolkits have led to the creation of UIMSs (User Interface Management Systems). UIMSs attempt to provide solutions to some of the problems of conventional programming toolkits. In most UIMSs, the designer specifies the user interface with a special purpose language and provides abstract user-system dialogue specification mechanisms. Their descriptions are generally interpreted, thus aiding prototyping to some degree. But they still require programming, and with an ordinary text editor it is not easy to create, debug, and modify a textual specification file for a complex user interface, and textual specifications of user interfaces are too complicated to be understand and used. So language-based UIMSs are not suitable for the non-programming interface designer.

1.2.3. Visual Programming Tools

Visual Programming Tools offer advantages over interface toolkits and language-based UIMSs inherent in their direct manipulation approach. No programming language has to be learnt, and immediate feedback is provided to the designer. Visual user interface tools fall into two categories, those built on existing programming toolkits and those aimed at more general user interface construction. An example of former category is OpenWindows Developer's Guide (Devguide) [SUN, 90B], it is a graphical user interface design tool built on XView user interface toolkits and that allows the user interface designer to interactively construct the user interface by drawing the desired interface primitives on the screen. The interface is automatically prototyped. But Devguide cannot offer this drawing function to all interface primitives; for example, Devguide uses the menu editor window to construct menus. Also Devguide cannot organize a complex user interface because it lacks specification of the user interface and cannot support the dynamical creation of hierarchical menus from their specification at run-time. An example of later category is Peridot [Myers, 88] which offers programming by demonstration for component construction and component behaviour specification. The information used to construct an interface is implicit and therefore hidden in the system. Also the interventions of the inference mechanism can become tedious for an experienced user [England, 90].

1.3. A New Menu Interface Development Environment

In order to make user interface development much easier and quicker, a new menu user interface development environment based on the Lean Cuisine [Apperley & Spence, 89] diagrammatic design tool has been built. The main goals for developing this menu user interface development environment are :

- ♦ To provide an interactive menu interface design environment and allow user interface designers who are not programmers to create menu interfaces by using Lean Cuisine graphical notation.
- ♦ To provide an automatic prototype environment and allow user interface designer to simulate the design partially or wholly at any design stage.
- ♦ To provide a tool used to generate source code for a user interface from its Lean Cuisine specification file.
- ♦ To support user interface development from design, simulation to direct implementation based on the Lean Cuisine graphical description.
- ♦ To ensure a true separation of concerns between the application and the user interface.

The new menu user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] aims to fulfil some of the deficiencies outlined in User Interface Programming Toolkits, UIMSs and Visual Programming Tools above, and to support the graphical user interface development lifecycle. The details of how these goals were achieved will be discussed in later chapters.

1.4. Overview of the Thesis

Chapter 2 presents a review of several existing user interface development tools in chronological order, and briefly discusses the limitation of these tools.

Chapter 3 discusses models of software development lifecycle and graphical user interface development lifecycle, main graphical environments for user interface development, notations for dialogue specification, and the integrated interface development environment based on the Lean Cuisine graphical notation.

Chapter 4 briefly reviews menu structures in the Lean Cuisine graphical notation, then describes in detail the basic structure and functions of Elc, the Lean Cuisine graphical editor, and discusses how a user interface designer interactively design the graphical dialogue specification by using the Elc tool. The user interface simulation at the design stage, and the implementation environment of Elc are also discussed.

Chapter 5 briefly reviews some existing interface testing tools and describes in detail Slc, an interface simulating tool used in the Lean Cuisine menu interface development environment to support quick and convenient user interface simulation based on the Lean Cuisine graphical notation.

Chapter 6 describes in detail Glc, a software tool used to generate basic user interface source files for a user interface from its Lean Cuisine graphical specification file.

Chapter 7 gives two detailed examples of software development in the menu interface development environment based on the Lean Cuisine graphical notation, and shows that a menu-based user interface can be rapidly simulated, modified and implemented based on its Lean Cuisine graphical specification.

Chapter 8 reviews the integrated user interface development environment based on the Lean Cuisine graphical notation and provides conclusions.

Chapter 2

Related Research

2.1. Introduction

In recent years, significant advances have been made in the area of user interface development. A number of user interface development tools like user interface builders, user interface management systems (UIMSs) or user interface development systems (UIDSs), and user interface development environments (UIDEs) have been developed and provided solutions to some of the problems of conventional programming user interface toolkits.

Some of these tools are designed to allow an interface designer to easily create, modify and generate an interface to an application through high level specifications [Apollo, 88; Olsen, 86; Olsen, 89; Schmucker, 86; Singh & Green, 89a]. Each of these tools introduces its own notation (a special purpose language) for the higher level user interface specification. The language based notation can be used by the user interface designer to specify the interface's syntax [Apollo, 88; Olsen, 89; Schmucker, 86] or the application's semantics [Olsen, 86; Singh & Green, 89a]. The special purpose language used by these tools may take several different forms: menu networks, state-transition diagram languages, context-free grammars, event languages, declarative languages and object-oriented languages [Myers, 89]. Obviously, these languages and tools can only be used by programmers.

Some of these tools are designed to let an interface designer define the user interface by placing objects on the screen with a mouse [Buxton et al., 83; Henderson, 86; Myers, 88; Myers et al., 93; Sun, 90b]. These tools can be used by nonprogrammers or programmers and user interface can be created without conventional programming. But these systems are more difficult to build and only support the creation of a limited range of interface [Myers, 89].

The following sections present a review of several user interface development tools in chronological order.

2.2. MIKE

Olsen (1986) developed MIKE (the Menu Interaction Kontrol Environment) [Olsen, 86] in Pascal for Pascal programmers. The goals for MIKE's development [Olsen, 92] are :

- Provide a high level UIMS model that would allow many new user interface capabilities to be automatically supported by the UIMS.
- Provide a dialogue specification model that would readily explained to nonprogrammers.
- Automatically generate a working interface from a minimal specification and then provide refinement tools for improving the generated interface.
- Integrate the visual layout specification with the dialogue and semantic specification.

Figure 2.1 shows the system architecture of MIKE. The four major components of MIKE system are the interface editor, the interface profile, the generated interface and the standard run-time code.

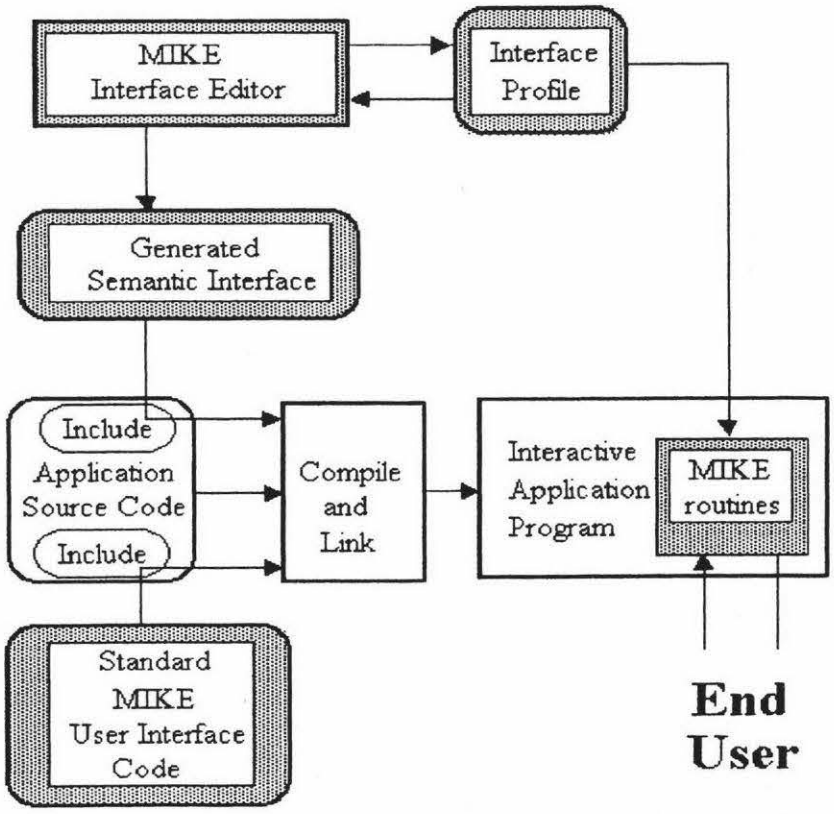


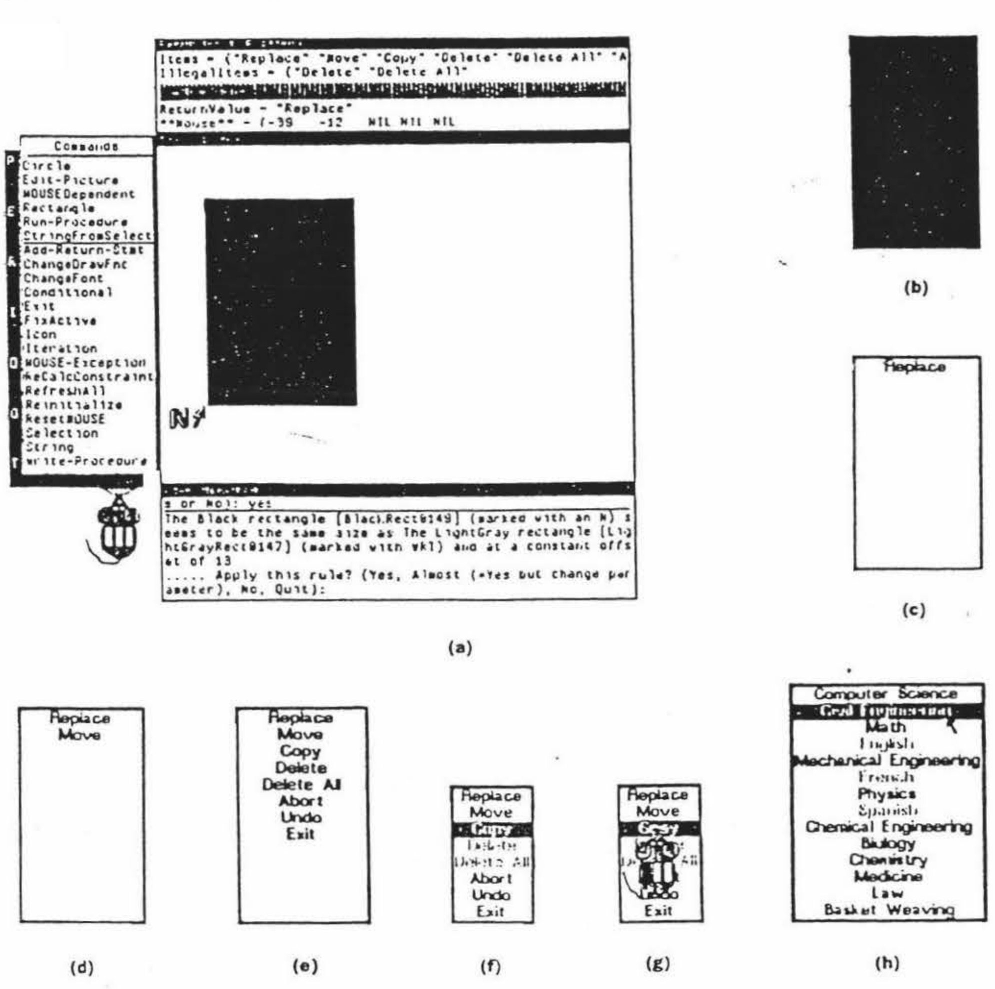
Figure 2.1. The system architecture of MIKE [Olsen, 86].

In this Menu Interface Kontrol Environment, the interface designer works through the graphical interface editor that provides access to the interface profile, which contains the complete definition of the dialogue's semantic and presentation aspects.

MIKE does not have a separate dialogue specification; the dialogue is controlled by semantics and the presentation. MIKE supports the user to extend the user interface by creating new macro commands by example. A major drawback of MIKE was its command line orientation and its limited range of interface techniques. In particular, it could not be used to provide the kinds of user interfaces that one finds on the Apple Macintosh and similar systems. In response to these difficulties, Olsen developed the Mickey system which will be discussed in section 2.5.

2.3. Peridot

Peridot [Myers, 88] stands for Programming by Example for Real-time Interface Design Obviating Typing and is a working prototype that is able to create many types of interaction techniques. Figure 2.2 shows a sequence of snapshots during the creation of a pop-up menu with Peridot .



```
Items = ("Replace" "Move" "Copy" "Delete" "Delete All" "A
IllegalItems = ("Delete" "Delete All")
ReturnValue = "Replace"
**Mouse** = (-39   -12   NIL NIL NIL
```

```
s or No): yes
The Black rectangle [BlackRect0149] (marked with an N) s
eems to be the same size as The LightGray rectangle [Lig
htGrayRect0147] (marked with vk1) and at a constant offs
et of 13
..... Apply this rule? (Yes, Almost (=Yes but change par
ameter), No, Quit):
```

Figure 2.2. A sequence of snapshots during the creation of a pop-up menu with Peridot[Myers, 89].

In particular, Peridot uses an inferencing mechanism for placing the constraints based on what the designer has drawn on the screen. This inferencing works not only for placement of single items but also for conditional and interactive placements. Peridot also allows the interactive mouse behaviours of visual objects to be defined by example.

Peridot is very different from other graphical user interface development systems because it lets the designer create the interaction themselves. It can create most of the interaction in the Macintosh Toolbox and its own interface [Myers, 89]. Peridot chiefly performs the interaction-technique builder function, and does not provide the support for sequencing, also called syntax or dialogue control of the user interface.

2.4. The UofA* UIMS

Singh & Green (1989) developed a high-level UIMS called the UofA* UIMS [Singh & Green, 89a] which uses a high-level descriptions of the semantic commands supported by the application to automatically generate the lexical and syntactic design of graphical user interfaces. Figure 2.3. shows the structure of the UofA* UIMS.

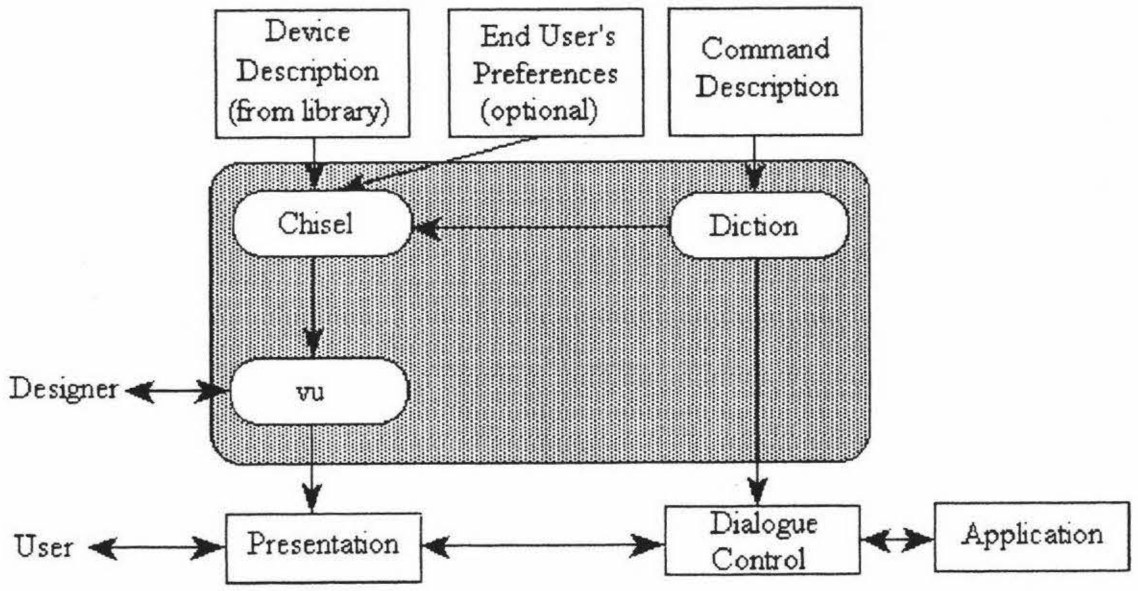


Figure 2.3. The structure of the UofA* UIMS [Singh & Green, 89a].

The main parts of the UofA* UIMS are Diction, Chisel and vu. Diction produces the dialogue control component of the user interface based on a description of the semantic commands and it also produces another output used by Chisel, in conjunction with a device description and optional user's preferences, to produce the presentation component of the interface. Vu is an interface graphical tool that can be used by the designer to refine the presentation component produced by Chisel. Figure 2.4 shows an example of command description used by the UofA* UIMS.

```
/*Declare global arguments. Center of Mass (COFM), length of the limb (LENGTH)
  and mass of the limb (MASS) are subranges of real numbers, whereas bend
  (BEND) and rotation (ROTATE) angles are subranges of integers.
  TORQUE represents a list of torque functions that the user can assign to limbs.
  LIMB is an interaction technique which implements 3-dimensional pick.
  The INFO window is used to display limb attributes.*/
COFM = [0.0 : 1.0]
LENGTH = [1.0 : 5.0]
MASS = [1.0 : 20.0]
BEND = [0 : 360]
ROTATE = [0 : 360]
TORQUE = (TorqueFunc1 TorqueFunc2 TorqueFunc3 TorqueFunc4
  TorqueFunc5 TorqueFunc6)
LIMB = pick3d
INFO = window
/* The commands are declared as follows. Command name is optionally
  followed by the syntax type and the selection type of the command.
  After this come the argument declarations. For each argument its
  type, range or enumerations, if any, and default value, if any, are
  specified. In the following declarations none of the commands specifies
  syntax or selection type.
  The Add_Limb, Remove_Limb, Move_Limb, Show_Attributes, Change_Orientation
  Change_Length, Change_Mass, Change_Cofm, and Change_Torque commands are
  declared as OPEN_ENDED. The rest of the commands follow the default
  selection type which is CLOSE_ENDED. The mass, length, cofm, bend, rotate,
  and torque arguments of the Add_Limb command assume the
  Currently Selected Values (CSVs) of their respective interaction techniques*/
Change_Root (position : LIMB)
ADD_Limb {OPEN_ENDED} (parent_limb : LIMB, mass : MASS {CSV},
  length : LENGTH {CSV}, cofm : COFM {CSV},
  bend : BEND {CSV}, rotate : ROTATE {CSV},
  torque : TORQUE {CSV}),
Remove_Limb {OPEN_ENDED} (limb : LIMB)
Move_Limb {OPEN_ENDED} (limb : LIMB, new_parent : LIMB)
Show_Attributes {OPEN_ENDED} (limb: LIMB)
Change_Orientation {OPEN_ENDED} (limb : LIMB,
  new_rotation : ROTATE, new_bend : BEND)
Change_Length {OPEN_ENDED} (limb : LIMB, new_length : LENGTH)
Change_Mass {OPEN_ENDED} (limb : LIMB, new_mass : MASS)
Change_Cofm {OPEN_ENDED} (limb : LIMB, new_cofm : COFM)
Change_Torque {OPEN_ENDED} (limb : LIMB, new_torque : TORQUE)
/* The Save, Load, and Exit commands do not have any arguments*/
Save( )
Load( )
Exit( )
```

Figure 2.4. Command Description for the Skeleton Editor [Singh & Green, 91].

From the command description shown in Figure 2.4, the interface produced by the UofA* UIMS is shown in Figure 2.5. All the commands in the interface are placed along the right edge of the screen. The interface can be refined in the vu environment.

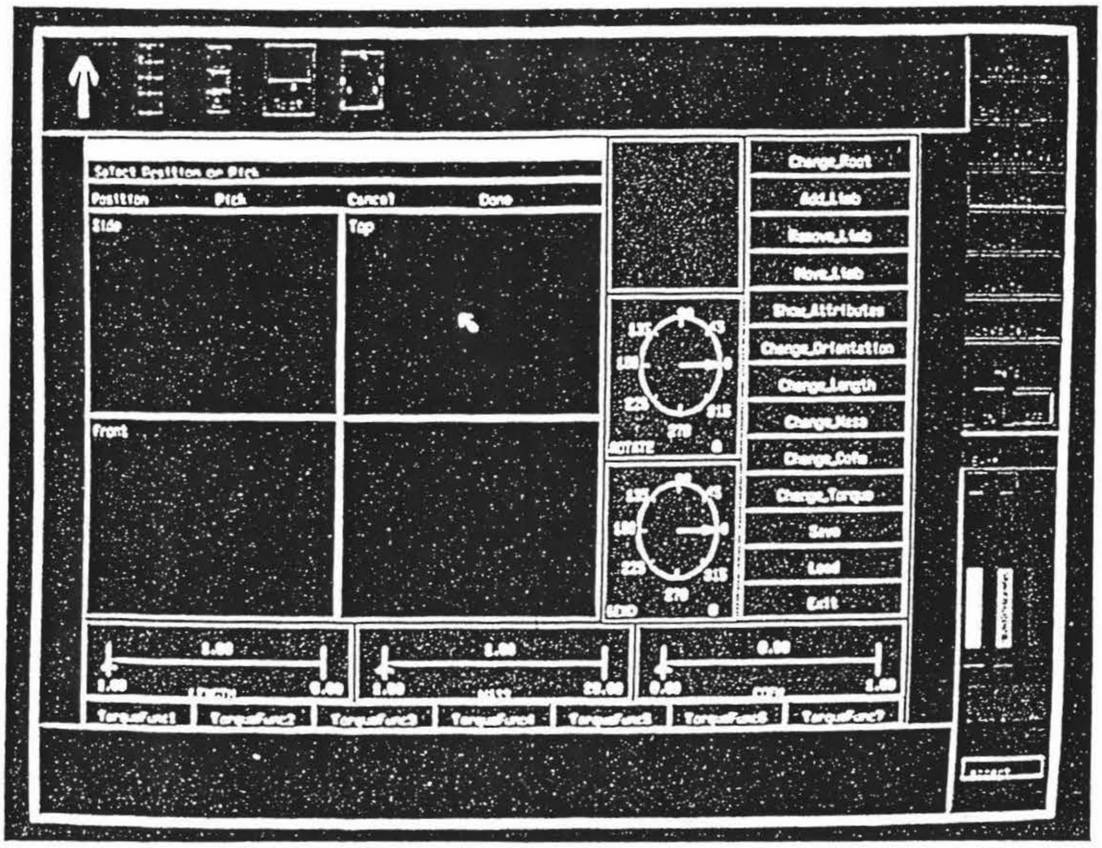


Figure 2.5. The interface generated from the command description (shown in Figure 2.4) can be refined in the vu environment [Singh & Green, 91].

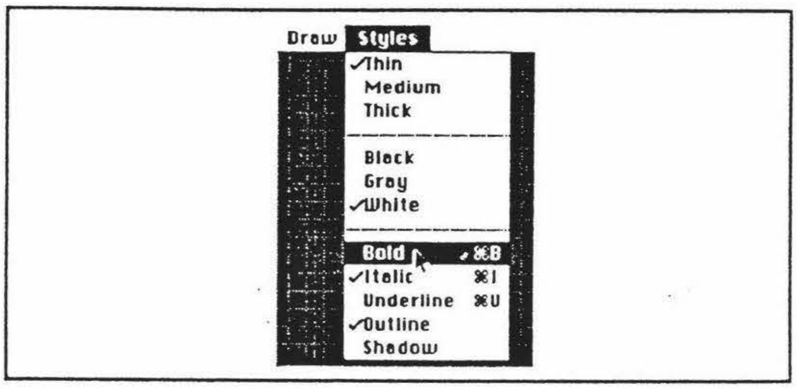
2.5. Mickey

The Mickey [Olsen, 89] system was developed by Olsen (1989) to explore the language UIMS paradigm in the context of direct manipulation interfaces. It was designed both as an attempt to simplify the user interface specification process and to broaden the scope of interfaces that could be modelled in a language-based UIMS [Olsen, 89].

Mickey extracts its user interface specification directly from the Pascal code. The presentation issues are handled by the Macintosh resource editor. Figure 2.6a shows the Pascal declarations of Mickey for a sample Macintosh menu *styles* and the corresponding *styles* menu generated by Mickey from its Pascal declarations is shown in Figure 2.6b.

```
TYPE
  LineWidth = (ThinLine (*Name=Thin *),
          MedLine (*Name=Medium*),
          ThickLine (*Name = Thick *) );
  FillColor = (BlackFill (*Name=Black*),
          GrayFill(*Name=Gray*),
          WhiteFill (*Name=White*) );
VAR
  LineSize (* Menu=Styles *): LineWidth;
  ColorSetting (* Menu=Styles *): FillColor;
VAR
  BoldText (* Menu=Styles Name=Bold Key=B*) : Boolean;
  ItalicText (* Menu=Styles Name=Italic Key=I*) : Boolean;
  UnderLineText (* Menu=Styles Name=Underline Key=U*) : Boolean;
  OutlineText (* Menu=Styles Name=Outline *) : Boolean;
  ShadowText (* Menu=Styles Name=Shadow *) : Boolean;
```

(a)



(b)

Figure 2.6. (a).The Pascal declarations of Mickey for a sample Macintosh menu *styles* and (b). the corresponding *styles* menu generated by Mickey from its Pascal declarations [Olsen, 89].

Obviously, Mickey is suitable for programmers to learn to use because of its programming language orientation. It is targeted specifically for application analysts because it is driven by the semantic definition of the interface. The form of specification of Mickey leads to a very tight binding between a particular user interface style and a particular programming language[Olsen, 89].

2.6. Devguide

Devguide [Sun, 90B] is a graphical user interface design tool that allows the user interface designer to create user interfaces without ever writing a line of code. To create an interface with Devguide, the designer simply drags element glyphs from Devguide's base window onto the workspace where each glyph turns into an interface element. Once the elements of a user interface are in place, the designer can edit each individual interface element by changing settings and values in a property window listing the properties of that element. After creating an interface, the designer can see how its elements look and feel when users use them. In Devguide's test mode, the interface is automatically prototyped. What the designer sees is the interface the application will have. When the interface has been specified the programmer has only to implement the functionality of the application.

Devguide implements the OPEN LOOK look and feel for the application by generating program calls to the XView user interface toolkit. The generated code is well formatted and easily readable and can be edited or directly compiled. Figure 2.7 shows the development procedure of a user interface by using Devguide.

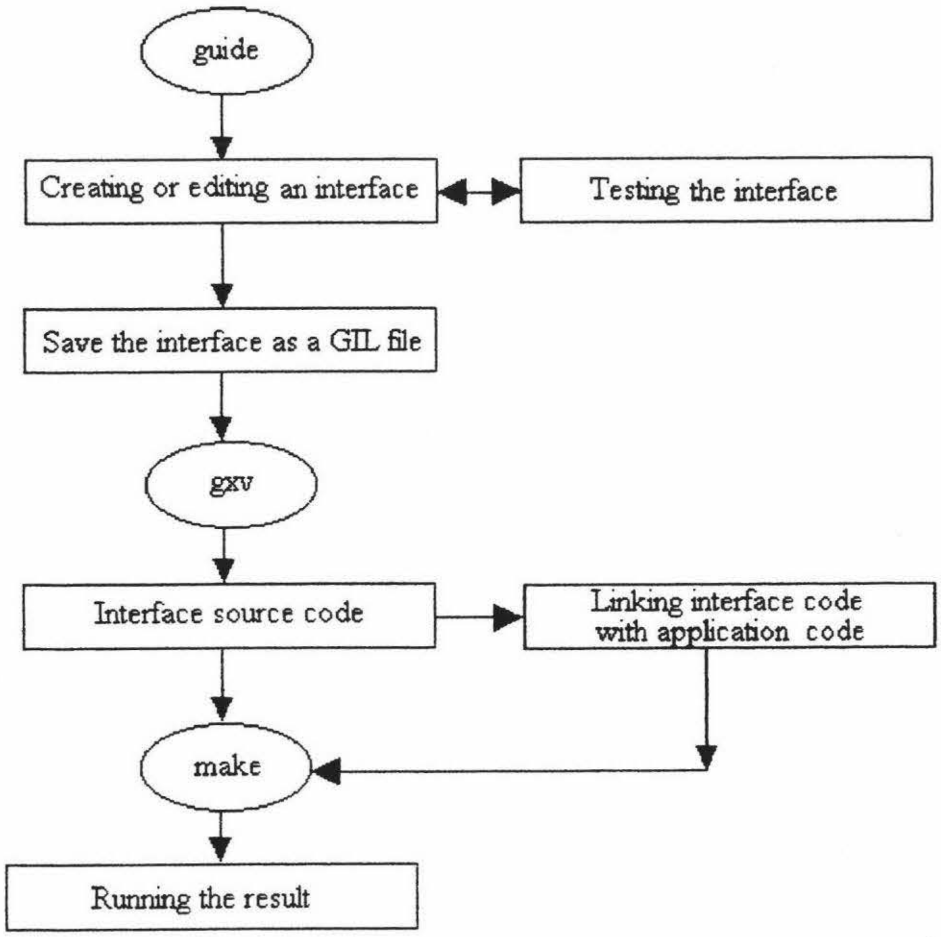

guide
Creating or editing an interface
Testing the interface
Save the interface as a GIL file
gxv
Interface source code
Linking interface code with application code
make
Running the result

Figure 2.7. The development procedure of a user interface by using Devguide

Although Devguide is a commercial user interface designing tool, it lacks a complete specification of the user interface. It does well in creation and layout of simple interface elements, but when used to create or modify complex interface elements, for example, hierarchical menus, it causes confusion because it cannot give a clear and complete specification of menus and sub-menus. Although the hierarchical menus can be created statically by using Devguide, Devguide cannot support the dynamic creation of hierarchical menus from their specification at run-time.

2.7. SCENARIOO

SCENARIOO [Roudaud et al., 90] is a UIMS which allows interface designers to prototype, develop, test and debug user interfaces either in a simulated or true application environment. The process used to build a user interface consists of creating and modifying three internal database; the Widget database, the Dialogue database, and the Simulation database. Each database has its own interactive editor. The global functional architecture for SCENARIOO is shown in Figure 2.8.

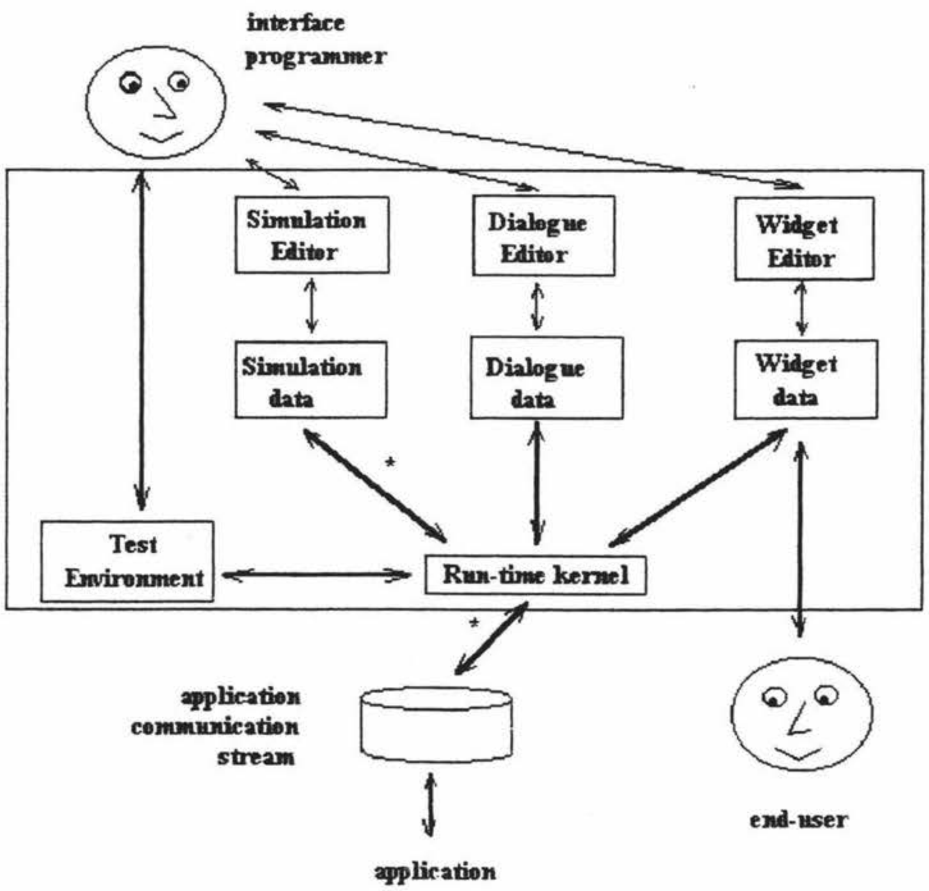


Figure 2.8. The global functional architecture for SCENARIOO [Roudaud et al., 90].

In Figure 2.8, the Widget Editor is used to create and modify widgets; the Dialogue Editor is used to construct event graphs which are used to decompose a complex dialogue into simpler ones and contribute the visual aspects of the dialogue; the Simulation Editor is used to create sets of application simulation data. The run-time kernel is connected either to the simulation database or to the actual application, the simulation database is used during prototyping and early debugging, and the connection to the actual application is made for final debugging and tests. At run time, either the real application or the simulation can be run, but not both together.

2.8. GENIUS

In the GENIUS [Janssen et al., 93] environment (GENerator for user Interfaces Using Software ergonomic rules), user interfaces for database-oriented applications can be automatically generated from extended data models and petri net-based dialogue descriptions. The GENIUS environment is shown in Figure 2.9.

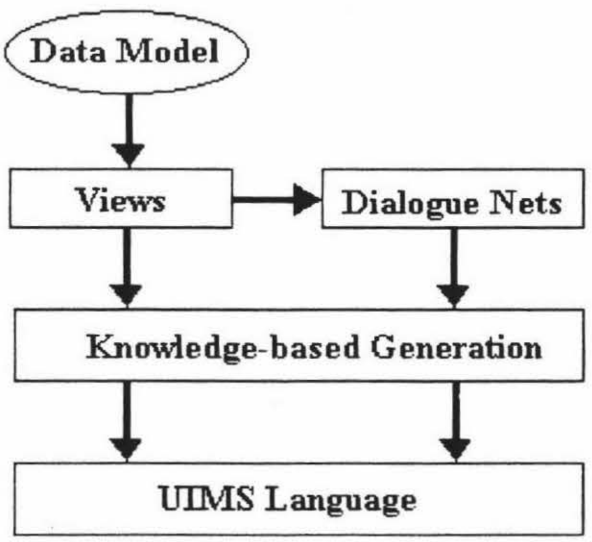


Figure 2.9. The GENIUS environment [Janssen et al., 93].

In the GENIUS approach shown in above, views are defined in the Entity Relationship Diagram Editor (ERDE) based on the Entity Relationship (ER) data model, transitions between views are specified in dialogue nets, a knowledge-based component generates the static user interfaces from the view definitions. As a target system, an existing UIMS is used. The dialogue nets as will are transformed into code for the UIMS. The GENIUS approach offers a much better integration of the user interface development with general software engineering methods. The specification of the dynamics is much easier with the graphical dialogue net representation than with a textual language. In the current system, refinements of the generated dialogue are done by means of the underlying user interface management system, the support for maintaining consistency between view and dialogue definition will be enhanced in the future work.

2.9. UIDE

UIDE [Foley et al., 91; Sukaviriya et al., 93], the User Interface Design Environment which utilizes a model of an application to automatically create an interface, is designed to allow interface designers to easily create, modify, and generate an interface to an application through high-level specification. Through UIDE, a designer creates an interface for an application by first describing objects and operations in the application, and then chooses various interface functional components by linking them to application operations. Once the application model is mapped to interface functional components, the UIDE run-time environment uses the specifications in the model to generate a desired interface. Figure 2.10 shows the run-time architecture of UIDE.

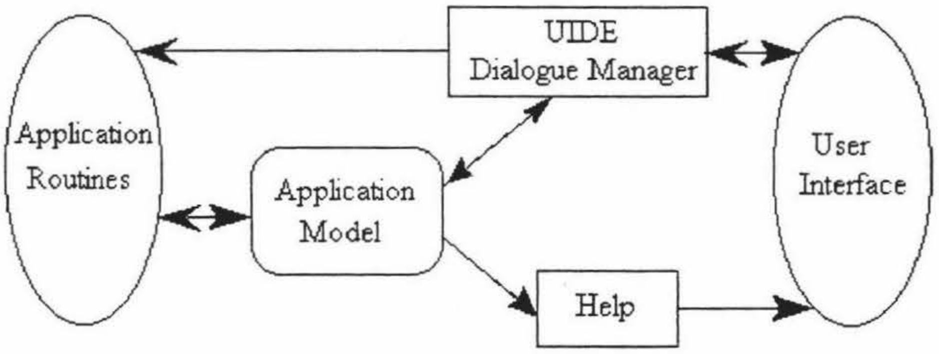


Figure 2.10. The run-time architecture of UIDE [Sukaviriya et al., 93].

Currently, defining the model in the C++ version of UIDE is still done through the standard C++ declaration and instantiation syntax. A visual tool to create an application model has been planned.

2.10. Summary

Many current user interface management systems (UIMSs) are language-based UIMSs. With an ordinary text editor, it is not easy to create, debug, and modify a textual specification file for a complex user interface. Textual specifications are too complicated to be understand and used by non-programmers, so these language-based UIMSs are very hard to use [Myers, 88]. Existing user interface development tools are not suitable for both interface designers and application developers. Graphical layout tools lack specification of the user interface. Very few UIMSs can provide software tools to support user interface development from user interface design, user interface simulation to direct implementation based on graphical description.

Chapter 3

Motivation

3.1. Software Development Lifecycle

Software development lifecycles have their roots in engineering methods and more specifically in the procedures of system engineering [Sage & Palmer, 90], the use of the tools of system engineering to develop, analyze, and implement large-scale system is embedded in the several software development lifecycles in use today. The models of software development lifecycle are still the subject of current research, but a number of different general models of software development lifecycle have been applied to practical software development projects [Boehm, 88; Cattell, 91; Fairley, 85; Hix & Hartson, 93; Laplante, 93; Lee, 93; Martin, 88; Sommerville, 92]. Some of these models are: the waterfall model, the structured approach, the structured evolutionary rapid prototyping approach, the spiral model, and the object-oriented software development lifecycle.

The waterfall model of the lifecycle

The initial waterfall lifecycle model has undergone numerous refinements, so there are many versions of this appearing in the literature [Boehm, 88; Bischofberger & Pomberger, 92; Laplante, 93; Sage & Palmer, 90; Sommerville, 92], and one of these is shown in Figure 3.1.

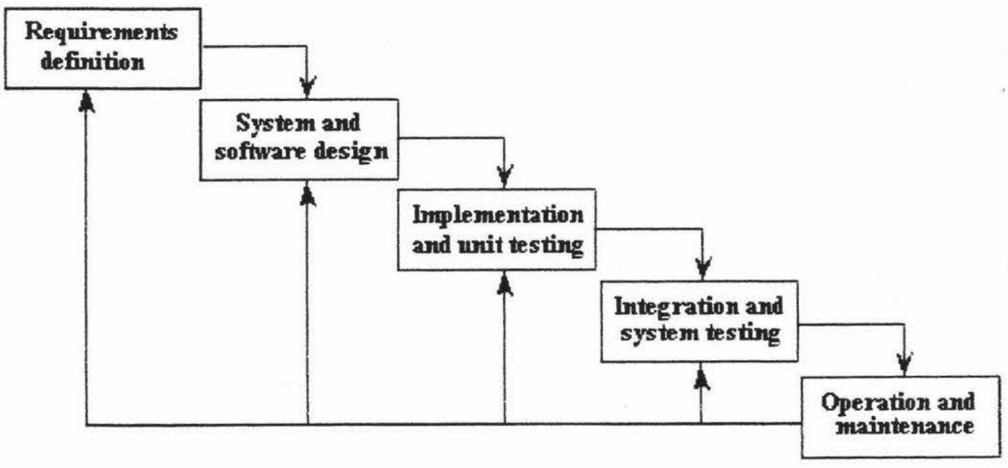

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 3.1. The waterfall model of the lifecycle [Sommerville, 92].

In Figure 3.1, the software process is made up of five stages; they are requirements analysis and definition, system and software design, implementation and unit testing, integration and system testing, and operation and maintenance. In an ideal situation, each of the above stages could be carried out in sequence. Actually, the software process is not a simple linear model, the development stages overlap and feed information to each other; one more stages are often repeated.

The structured approach

The concept of the structured software development lifecycle was introduced by Yourdon [Yourdon, 82]. The structured software development lifecycle model is shown in Figure 3.2.

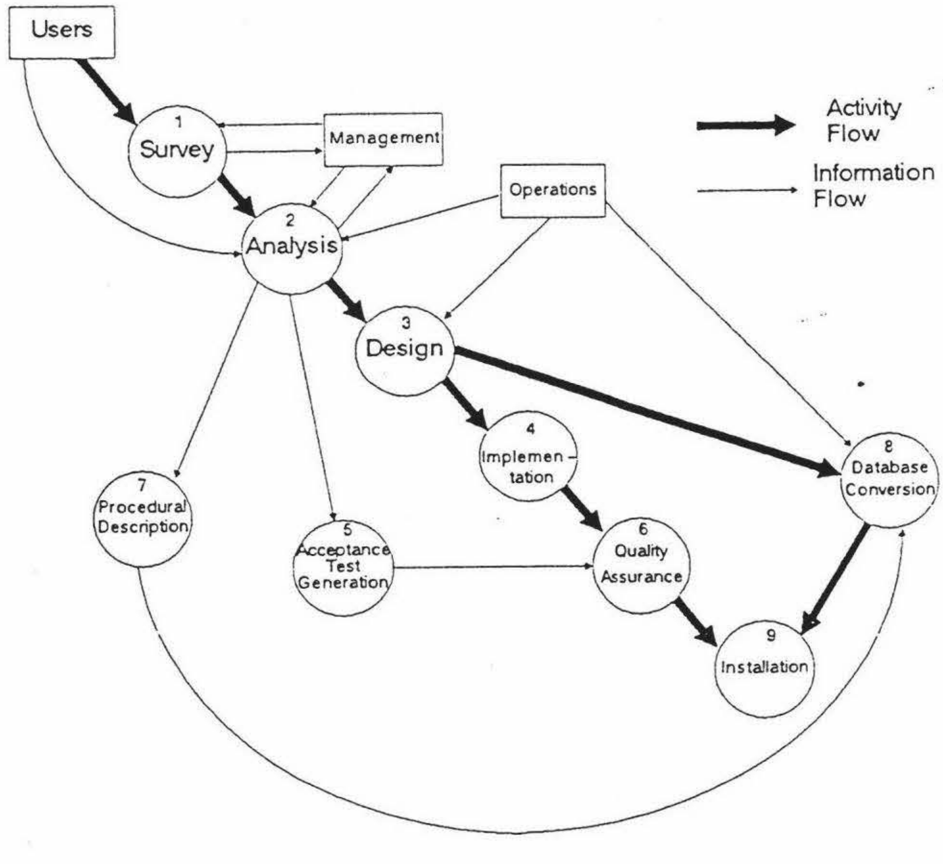


Figure 3.2. The structured software development lifecycle model [Yourdon, 82].

The nine separate phases are shown in Figure 3.2; they are survey, analysis, design implementation, acceptance test generation, quality assurance, procedure description database preparation, and installation. Contrasting the Yourdan structured software development lifecycle model with the waterfall software development lifecycle model there is no difference in the stages specified, other than minor semantic ones [Sage & Palmer, 90].

The structured evolutionary rapid prototyping approach

The lifecycle activities of the structured evolutionary rapid prototyping appro: [Connell & Shafer, 89] are derived from a structured development lifecycle witl number of necessary changes and consists of rapid analysis, prototyping, design, tuni and testing. As opposed to the structured software development lifecycle discus: above, the structured evolutionary rapid prototyping approach does not requ exhaustive analysis and design activities before implementation activity takes place ε also differs from traditional throwaway prototyping [Lee, 93]. The evolutionary ra| prototyping process is shown Figure 3.3.

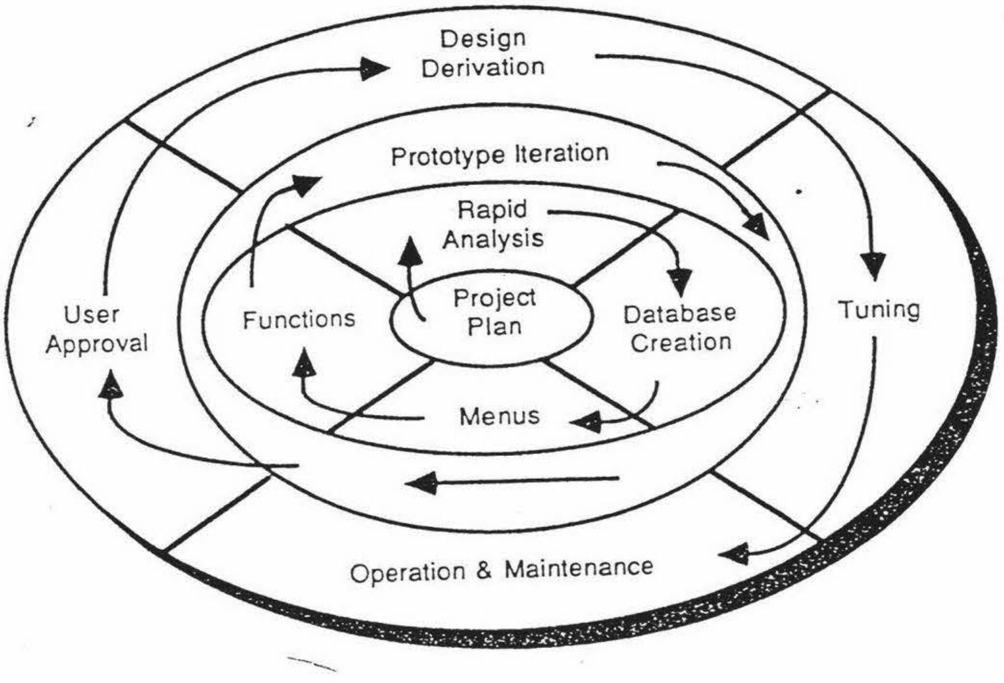


Figure 3.3. The evolutionary rapid prototyping process [Connell & Shafer, 89].

In the structured evolutionary rapid prototyping approach, the requirements document produced to specify user profile and the initial user task model to be realized in the initi prototype. The initial prototype is constructed right after rapid analysis activity withoı going through any formal design activity and detailed design will be derived from a use: approved prototype. The testing activity is not clearly defined in the structure evolutionary rapid prototyping approach because a prototype has effectively undergon system testing throughout the lifecycle. The key to a successful structured evolutionar rapid prototyping development is access to good software tools that allow rapi construction and iterative evolution of prototypes. With its early prototyping activity an iterations of user feedback and developer implement, the structured evolutionary rapi prototyping approach shortens the development lifecycle span, increases user satisfaction, and lowers development and maintenance costs. Fourth-generation languages (4GL) provide high-level interpretive description of program functionality and are widely adopted for prototyping database applications [Lee, 93].

The spiral model

The spiral model [Boehm, 88] of the software development lifecycle differs from the waterfall model in that the waterfall model is a specifications-driven model but the spiral model is a risk-oriented or risk-based model. An outline of the spiral model is shown in Figure 3.4.

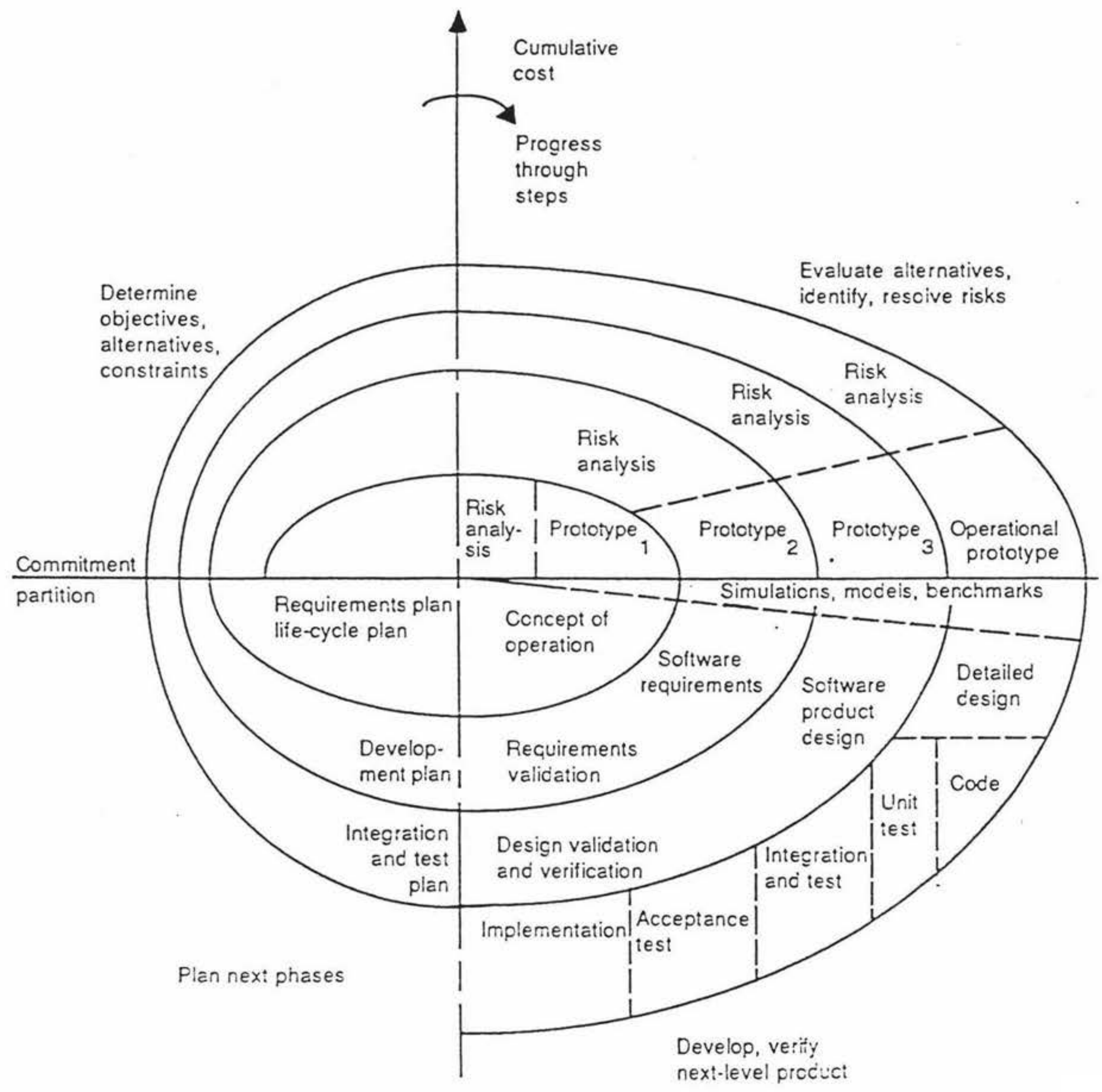


Figure 3.4. The spiral model [Boehm, 88].

The key characteristic of the spiral model is an assessment of management risk items at regular stages in the project and the initiation of actions to counteract these risks. Before each cycle, a risk analysis is initiated and at the end of each cycle, a review procedure assesses whether to move on to the next cycle of the spiral. In Figure 3.4 the radial dimension reflects the cost parameter as these costs accumulate over the several phases completed at the time of use of the model; the angular dimension represents the progress made in the development at a particular cycle. The spiral model has been designed to accommodate all other models of development and a risk-oriented spiral model of process management forces the consideration of all alternatives and risks [Sommerville, 92]. The advantages of the spiral model are the introduction of the notion of risk assessment and risk management into the software development process. It is able to incorporate the best attributes of other software development lifecycle models and is well suited for iterative process[Sage & Palmer, 90].

The object-oriented software development lifecycle

The object-oriented software development lifecycle [Lee, 93] shares some characteristics with the rapid prototyping approach. Since object classes are abstract data types with encapsulated data and methods, individual classes can be regularly refined while creating new classes through inheritance mechanisms. The object-oriented software development lifecycle is iterative and evolutionary. Figure 3.5 shows a object-oriented software development lifecycle proposed by The Object Management Group (OMG).

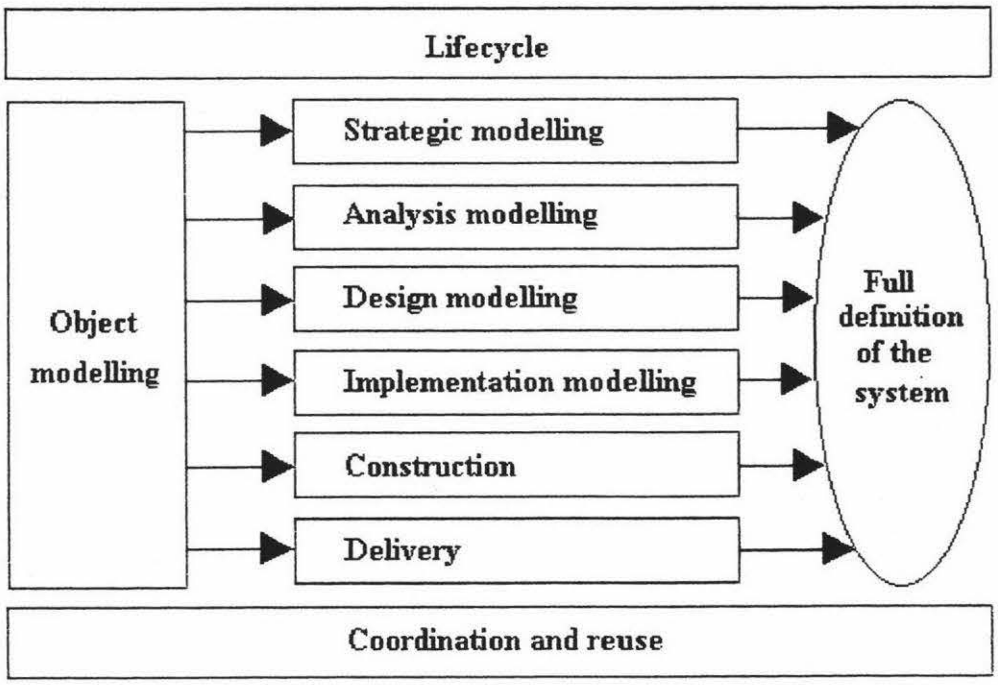


Figure 3.5. The OO software development lifecycle proposed by OMG [Yourdon, 94].

The object-oriented approach encapsulates data with corresponding operations, and employs polymorphic mechanisms such as class inheritance to allow incremental software development. With data encapsulation, object classes can be conveniently reused, modified, tested, and extended. As a software development approach, more recent developments have focused on object-oriented analysis and object-oriented design activities [Booch, 94; Yourdon, 94].

3.2. Graphical User Interface (GUI) Development Lifecycle

Graphical user interface development consists of several phases: the analysis of graphical user interface requirements, graphical user interface design, graphical user interface prototyping, graphical user interface evaluation, and graphical user interface implementation. The graphical user interface development lifecycle generally follows that of software development, so some models of software development lifecycle discussed in section 3.1 can be adopted for graphical user interface development. But designing and developing the graphical user interface is an iterative process of refinement that persists as long as there is sufficient motivation to modify an interactive system. Thus, any implementation should be based on a software architecture that supports modification of the user interface [Bass & Coutaz, 91]. The key to the success of graphical user interfaces is user-centred design [Lee, 93]; it is not acceptable for designers simply to impose their view of an acceptable user interface on users, and the user must take part in the interface design process [Sommerville, 92] and give timely user evaluation and feedback.

When a waterfall lifecycle model or a structured software development lifecycle model is used for graphical user interface development, testing is the only stage to allow meaningful user feedback, any errors and problems introduced during the analysis or design stage will have to be found and corrected only at a very late stage, it can be costly if extensive design modifications are required. Although iterations are always included in various descriptions of the waterfall lifecycle model, it remains unclear when and according to what criteria such iterations are to be carried out [Bischofberger & Pomberger, 92]. These make the waterfall lifecycle model and the structured software development lifecycle model less appealing for graphical user interface development, where timely user evaluation and feedback are critical to success.

The prototyping-oriented software development lifecycle model [Bischofberger & Pomberger, 92; Lee, 93; Martin, 88; Sage & Palmer, 90; Smith, 91; Sommerville, 92] has its strengths where the waterfall lifecycle model and the structured software development lifecycle model display serious weaknesses. It can be used for graphical user interface development because it supports the requirements definition. The user interface specification is the most important part of the requirement definition and its quality is decisive. The user interface specification and its underlying functional behaviour can be represented much better and tested more effectively with an executable prototype than in the case with other means of represents. A user interface prototype can provide an ideal basis for communication between the user interface developers and the users.

The "star" user interface development lifecycle (Figure 3.6) [Hix & Hartson, 93] shows that evaluation is to be applied to every user interface development activity, not just to design.

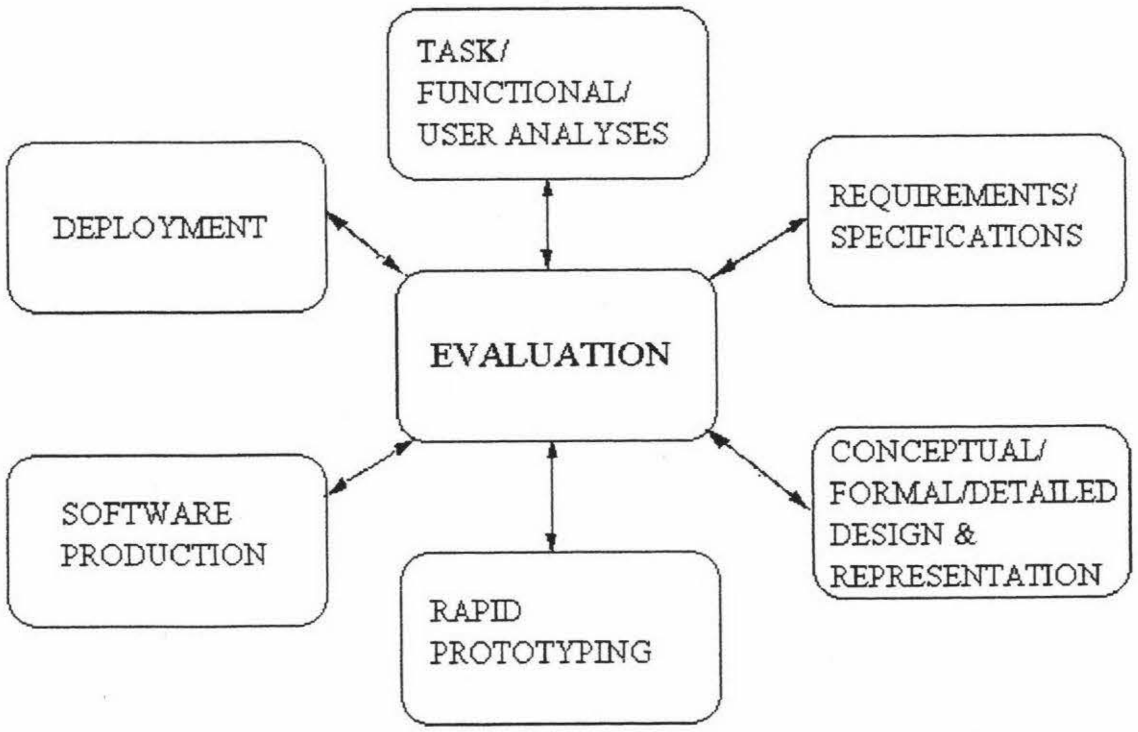


Figure 3.6. The "star" lifecycle of user interface development [Hix & Hartson, 93].

The star lifecycle also shows that there is not a single starting point, nor is there a prescribed order for development activities. Each development activity should be followed by evaluation of some sort.

The object-oriented software development lifecycle model [Lee, 93] treats the graphical user interface development as an integrated object-oriented software development process. A task model based on the GOMS (Goals, Operators, Methods and Selection) model will guide the object-model construction. The task model and the object model are the high-level abstraction of a graphical user interface application. Developing a graphical user interface for a specific style guide becomes a straightforward mapping process.

The user interface development lifecycle proposed by Perlman [Perlman, 88] includes three main stages: design, evaluation, and implementation (Figure 3.7) . The sources of design and redesign are personal experiences with previous development and popular system, guidelines and standards provided by basic and applied research and experts, and feedback based on evaluation of similar existing system and of prototype system. Formal models make predictions of the goodness of a user interface based on a formal specification of the design, without requiring implementation. Usage data contribute to an empirical evaluation after implementation of a preliminary version or a prototype. The user interface development lifecycle shown in Figure 3.7 is often iterative, with many small changes in design being implemented and evaluated over its brief period [Perlman, 88].

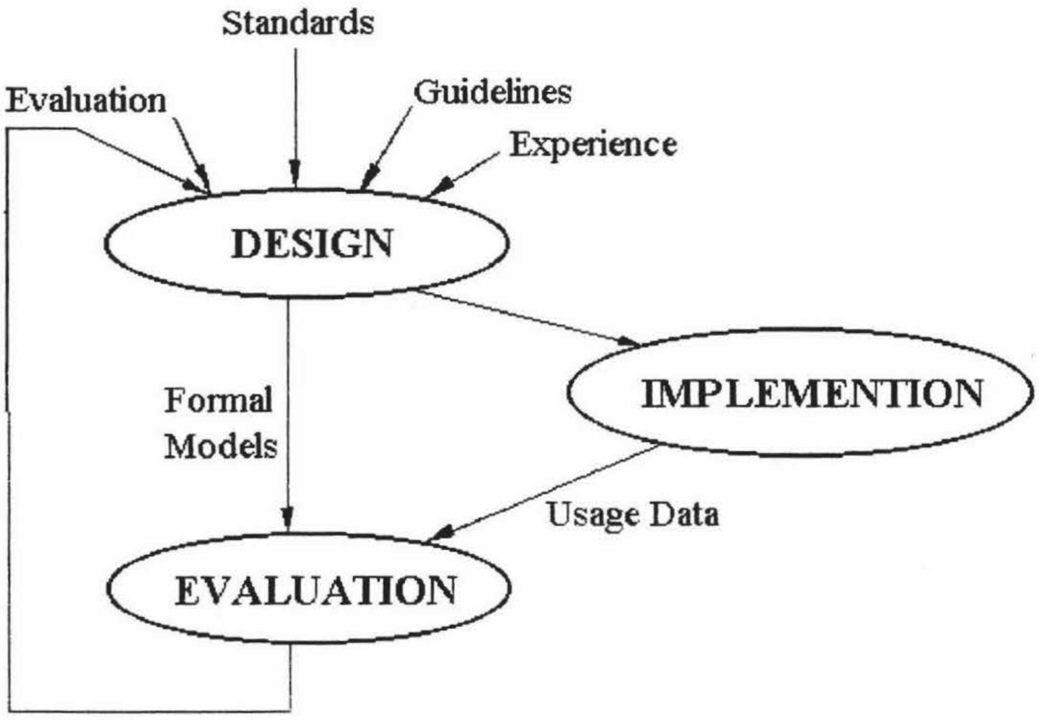


Figure 3.7. The user interface development lifecycle proposed by Perlman [Perlman, 88].

3.3. Current Graphical Environments

There are many graphical environments which provide hardware and software support for graphical user interface development and application. Generally speaking, the hardware support of a graphical environment includes user input devices (e.g., mouse, keyboard) and graphics-display output devices (high-resolution graphical monitors). The software support of a graphical environment includes graphical operating system (e.g., OpenWindows™, Microsoft Windows™) and layered abstraction of software libraries (e.g., Xlib, Xt Intrinsics, Motif, OPEN LOOK, XView, Macintosh Toolbox, Microsoft Windows Libraries) which are used to assist the development of application software. In different graphical environments, different layers based on different abstractions allow graphical application software to access various services provided by that graphical environment.

In the X Window System, there are a number of primitive layers. The Xlib contains all the necessary low level primitives to construct X Window applications. This would be the lowest level that an application developer would work at. The Xt Intrinstics collects together the Xlib primitives and provides a basic framework to construct and use widgets. A number of style-specific toolkits are available with different layering architecture. The Athena widget set (Xaw), the Motif widget set, and the OPEN LOOK Intrinsics Toolkit (OLIT) are Xt-based widget sets, each of which is implemented as a software layer above the Xt Intrinsics. The XView toolkit is a Xlib-based toolkit and it implements the OPEN LOOK graphical user interface. In the X Window System, the typical software development environments are shown in Figure 3.8, and Figure 3.9.

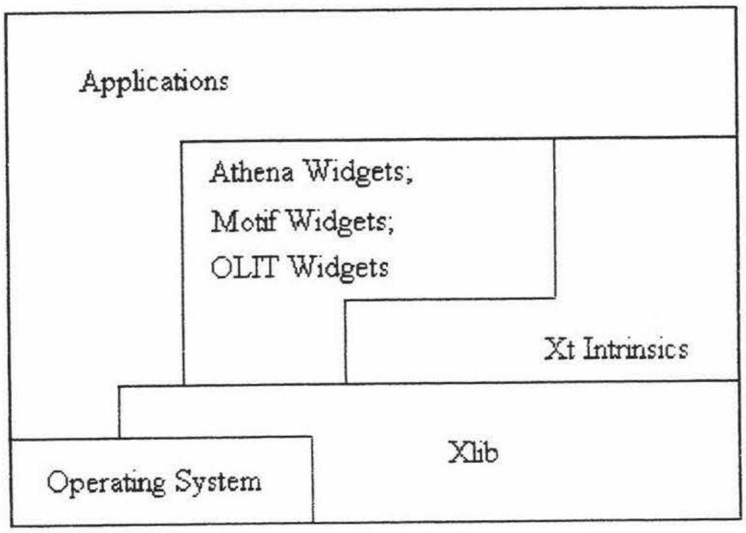


Figure 3.8. The software development environment using Xt-based widget set.

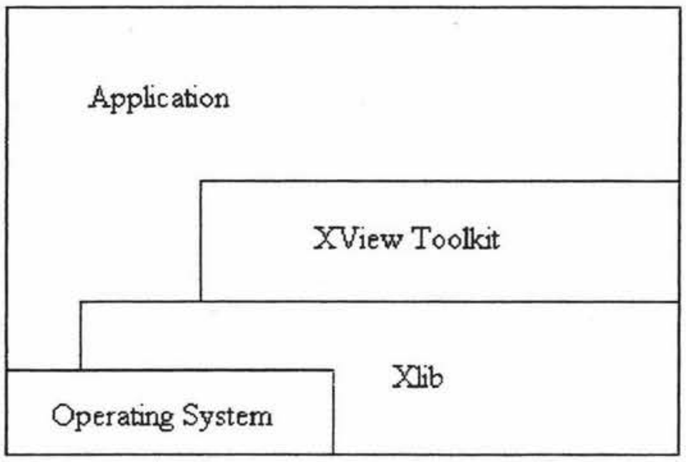


Figure 3.9. The software development environment using XView.

In the Apple Macintosh graphical environment, graphical user interfaces are developed by using Macintosh User Interface Toolbox which provides various resource services to support a Macintosh-style graphical user interface implementation. The Macintosh software development environment is shown Figure 3.10.

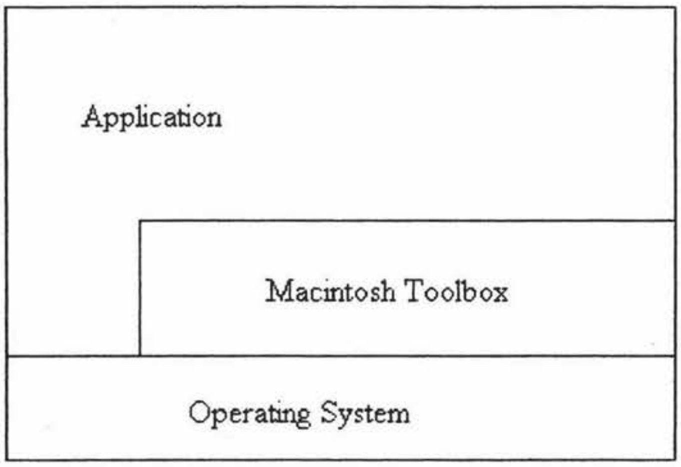


Figure 3.10. The Macintosh software development environment.

In the Microsoft Windows 3.x software development environment, there are a number of developing tools that support the development of graphical user interfaces. The most common tools for Windows programming are the Microsoft C compiler and the Microsoft Windows Software Development Kit (SDK), Borland C++ and Resource Workshop, and Microsoft Visual Basic. The application program interface (API) of Microsoft Windows consists of all functions that Windows programs may call. All the Windows functions are declared in a large header file named WINDOWS.H, which is included in the Windows Software Development Kit (SDK). An import library included in the SDK is used by dynamic linking to create an executable Windows program. The three major dynamic link libraries included in Windows are a kernel library, a user library, and a graphics device interface (GDI) library. The kernel library contains the tasking and memory management functions, the user library contains the windowing and user interface functions, and GDI library contains the Windows Graphics Device Interface functions. The Microsoft Windows software development environment is shown in Figure 3.11.

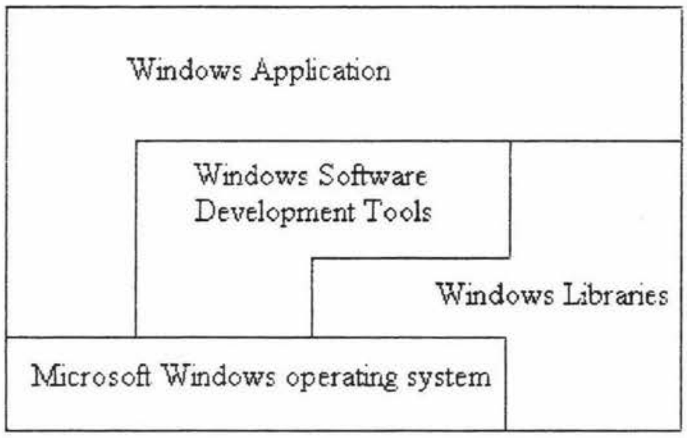

Windows Application
Windows Software Development Tools
Windows Libraries
Microsoft Windows operating system

Figure 3.11. The Microsoft Windows software development environment.

3.4. Notations for Dialogue Specification

There are many notations for dialogue specification. Existing techniques can be grouped into two classes; textual notations and graphical notations.

3.4.1. Textual Notations

The textual notations are special-purpose languages used by the interface designer to describe dialogue specification and control. These special-purpose languages are context-free grammars, declarative languages, event languages, object-oriented languages and command descriptions. The details of these special-purpose languages can be found from [Alty & Mullin, 89; Apollo, 88; Hill, 86; Olsen, 92; Schmucker, 86; Sibert et al., 86; Singh & Green, 91; Yap & Scott, 90]. Although these special-purpose languages have been used in most UIMSs for interface specification and some of them have proved to be very useful in user interface development. The user interface specifications based on these special-purpose languages are difficult to create, debug, modify and understand. The languages used by many UIMSs to specify the interface are poorly structured in software-engineering sense. Most UIMSs require that the designer learn a new special-purpose language and some of them are real programming environments (e.g., MacApp, Mickey), so they are inaccessible to nonprogrammers [Myers, 89].

3.4.2. Graphical Notations

The graphical notations incorporate special-purpose symbols used by the interface designer to describe dialogue specification and control in a graphical manner. For example, the Lean Cuisine graphical notation [Apperley & Spence, 89] is based on tree diagrams used to describe the structure and the behaviour of menu systems. Transition networks based on a set of nodes (states) and the links between them (state transitions) have been extensively used as a dialogue description technique, by both designers and implementors, and are probably the most popular form of dialogue description currently in use [Apperley & Spence, 89]. There are other graphical notations used in the context of dialogue specification and user interface management systems, these graphical notations are Petri nets [Bastide & Palanque, 90], event graphs [Roudaud et al., 90], and dialogue nets [Janssen et al., 93]. Event graphs can be used to describe the dynamic visibility of objects within a graphical-interactive application. Dialogue nets are similar to event graphs in modelling the dynamic visibility of objects needed for the coarse-grain dialogue specification. Petri net objects are more powerful since general object flow and object manipulation can be described. However, they are more complex than event graphs and less suited for the earlier design phases [Janssen et al., 93]. The following discussion focus attention on Transition networks and Lean Cuisine graphical notations.

3.4.2.1. Transition Networks

Transition networks, also known as finite state machine diagrams or state transition diagrams, consist of nodes that correspond to a state and directed arcs that connect nodes. In a transition network, the dialogue is shown as being in a particular state awaiting the next interaction with the user [Jones, 91b]. Arcs out of each state are labelled with the input token that will cause a transition to the state at the other end of the arc. In addition to input tokens, the arcs in some systems are labelled with application procedures to be called and output to be displayed [Myers, 89]. Transition networks often show the dynamics of a dialogue in a graphical manner.

Some software tools have been developed to create and interpret transition networks. For example, Rapid/Use [Wasserman et al., 82; Wasserman, 85; Wasserman et al., 86] is based on graphical specification of the required dialogue using transition networks. The dialogue is specified based on diagrams created using a graphical tool -- Transition Diagram Editor (TDE). A typical transition diagram created using the TDE is shown in Figure 3.12.

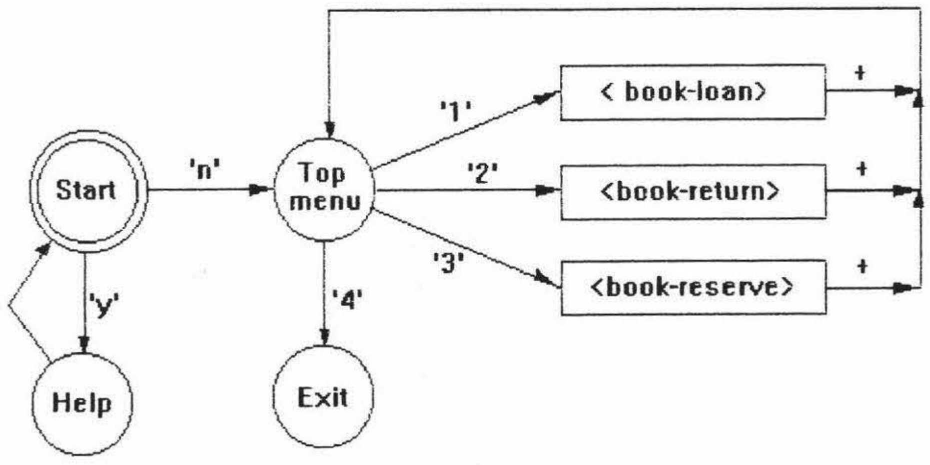
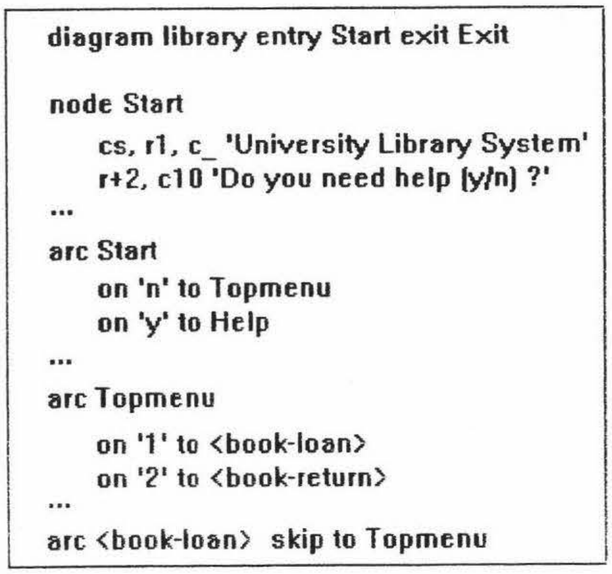


Figure 3.12. A Rapid/USE transition diagram [Jones, 91a].

A text file can then be compiled from the transition diagram created using the TDE, or alternatively the text file can be created directly by using a special language (the transition diagram language) entered with a conventional text editor. The textual representation of above Rapid/USE transition diagram is shown in Figure 3.13.



```
diagram library entry Start exit Exit

node Start
    cs, r1, c_ 'University Library System'
    r+2, c10 'Do you need help [y/n] ?'
...
arc Start
    on 'n' to Topmenu
    on 'y' to Help
...
arc Topmenu
    on '1' to <book-loan>
    on '2' to <book-return>
...
arc <book-loan>  skip to Topmenu
```

Figure 3.13. Textual representation of the Rapid/USE transition diagram [Jones, 91a].

Based on the text file of the Rapid/USE transition diagram, a menu system can be rapidly generated by using Transition Diagram Interpreter (TDI) and can be used to demonstrate and evaluate the dialogue. The similar system based on transition networks can be found from [Jacob, 85].

Although transition networks have been extensively used as a dialogue description technique, by both designers and implementors, and are probably the most popular form of dialogue description currently in use, this graphical notation has obvious shortcoming when attempting to describe a menu containing a number of interrelated subgroups [Apperley & Spence, 89]. Transition networks can become an incomprehensible maze of arcs as the interface becomes large, and can be very difficult to understand and edit [Myers, 89]. A typical example is shown in Figure 3.14.

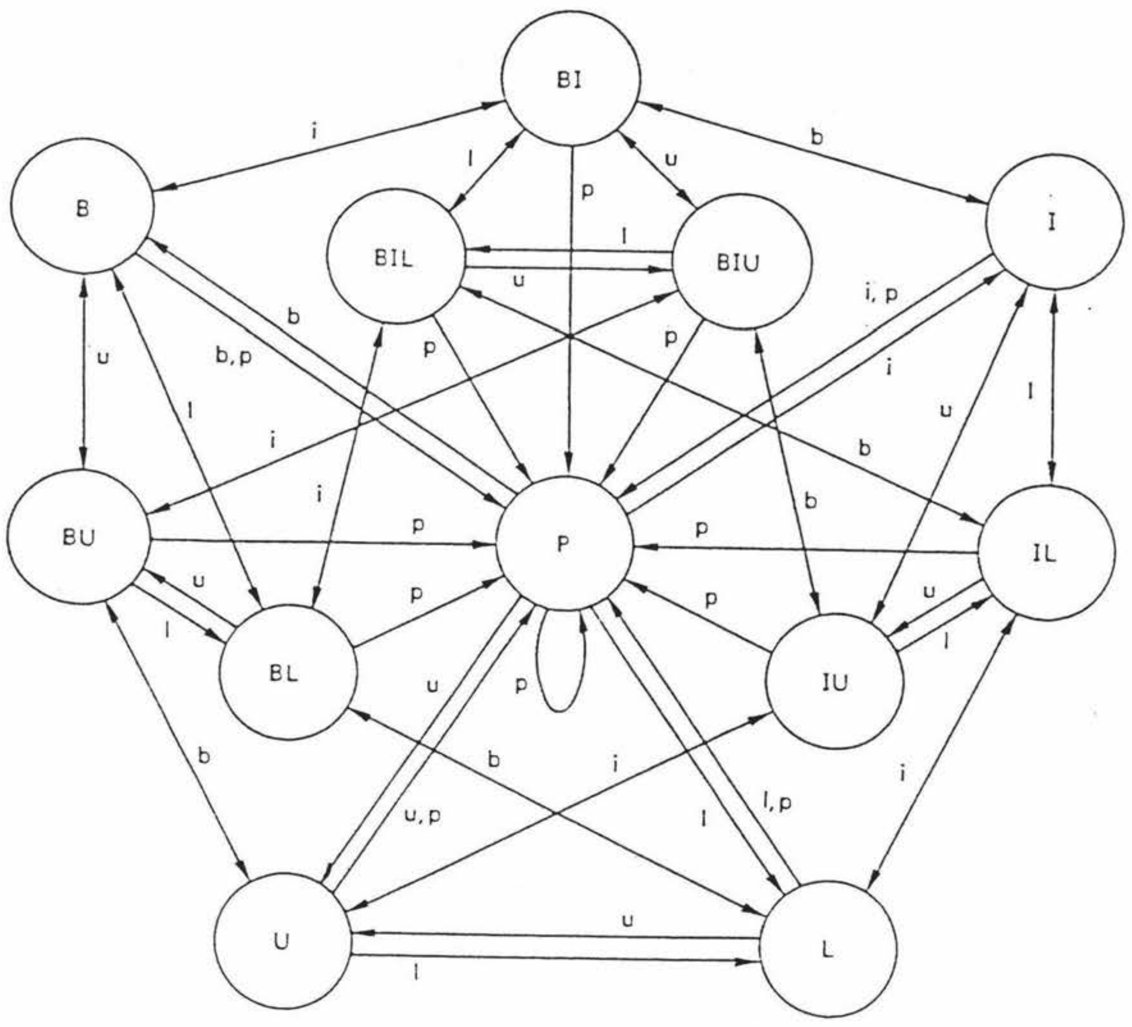


Figure 3.14. Transition network description of the Macintosh Style menu [Apperley & Spence, 89].

In Figure 3.14, the behaviour of a five meneme menu is described using a transition network; states in the diagram are used to represent twelve possible menu states, and arcs are used to represent the possible input actions in each state. Although the diagram does completely and accurately describe the behaviour of this particular menu, it is neither simple to produce nor to understand [Apperley & Spence, 89].

3.4.2.2. Lean Cuisine Notation

Lean Cuisine [Apperley & Spence, 89] is a visual notation used to describe the behaviour of hierarchical menu systems. Two basic tree diagrams are used to describe mutually exclusive (1-from-n) menu groups and mutually compatible (m-from-n) menu groups. In the Lean Cuisine description of a menu system, a menu consists of a set of menemes which may be terminal (leaf) menemes, real non-terminal menemes, or virtual menemes. Terminal (leaf) menemes represent specific actions or parameters; real non-terminal menemes are headers to other menus, and virtual menemes are used to partition a single menu, and are not presented to the user. Also in the Lean Cuisine description of a menu system, many special characters are used as menu and meneme modifiers. Figure 3.15 shows the Lean Cuisine description of the reduced five meneme *Style* menu which has been described using a transition network in Figure 3.14.

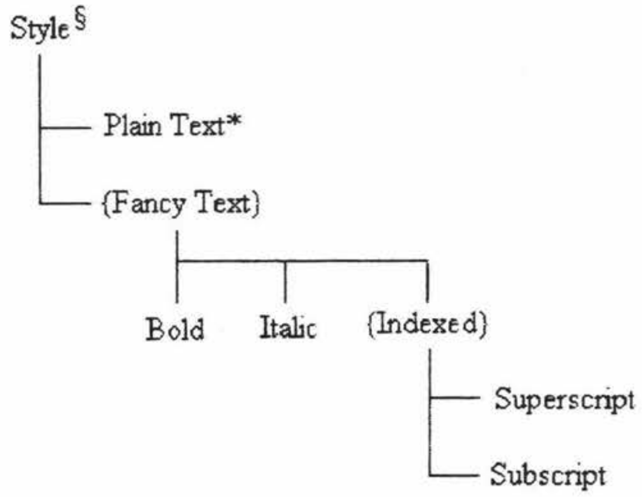


Figure 3.15. The Lean Cuisine description of the reduced five meneme *Style* menu.

In Figure 3.15, *Style* is a required choice group marked § and offers a mutually exclusive choice between *Plain Text*, which is marked * as the default choice, and the virtual meneme *{Fancy Text}*. This virtual meneme in turn represents the mutually compatible choice of *Bold*, *Italic* and *{Indexed}*, the latter another virtual meneme offering a mutually exclusive choice between *Superscript* and *Subscript*.

The Lean Cuisine notation has been used as a diagrammatic design tool to design a user interface from specification, through preliminary analysis, to the final design [Apperley, 88], and its textual form has been used as the basis for an interface prototyping environment which provides for the direct implementation of the action and control layers from a Lean Cuisine description [Anderson & Apperley, 90]. The prototyping environment implemented by Anderson & Apperley on an Apple Macintosh provides two software tools: the first tool generates the interface definition file (IDF) based on a Lean Cuisine textual specification and the second tool -- an interpreter then uses the interface definition file (IDF) to simulate the interface. Figure 3.16 shows the translation of a textual Lean Cuisine specification (b) to an intermediate presentation specification (c). The graphical specification of the simple menu is shown in (a) and the resultant menu is shown in (d).

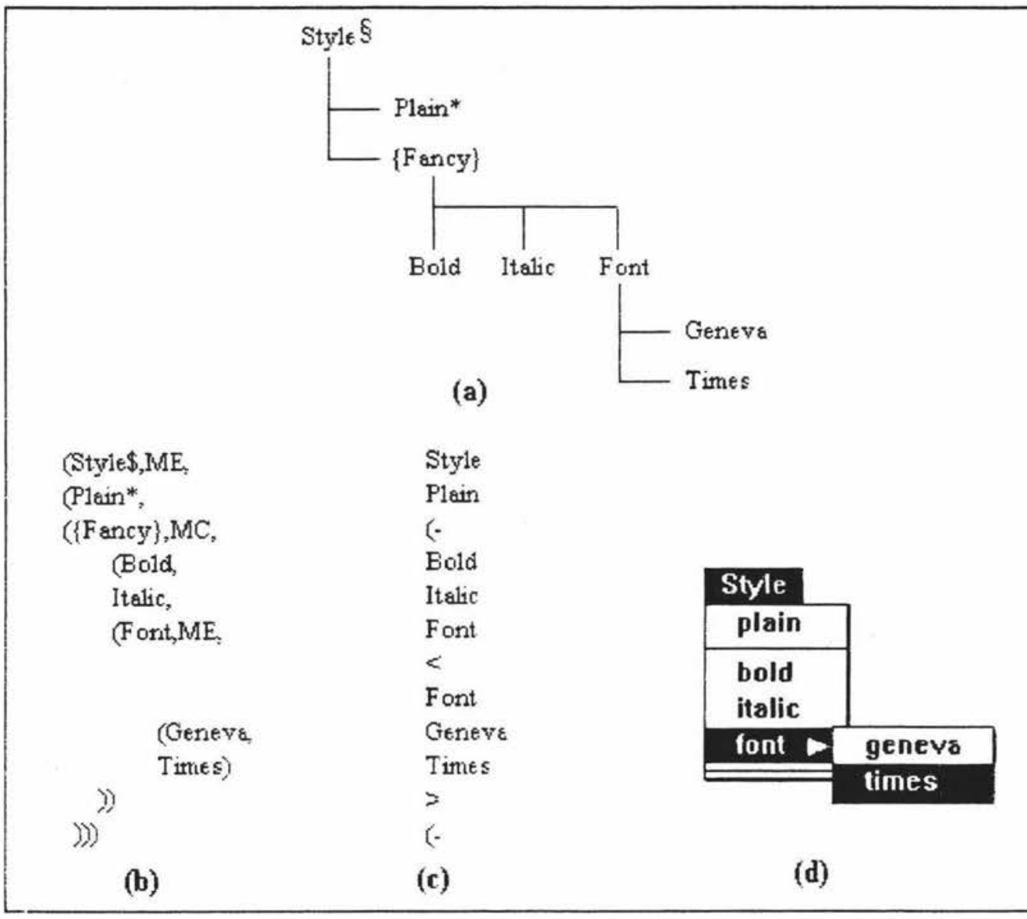


Figure 3.16. **(a)** Lean Cuisine specification, and **(b)** its translation from a textual description to: **(c)** equivalent presentation model and **(d)** resultant menu [Anderson & Apperley, 90].

Lean Cuisine offers a clear, concise and compact notation, which in its present form can be used to describe both the action and control layers of hierarchical menu systems [Anderson & Apperley, 90], and as a graphical design tool, it is more powerful than other graphical notations (e.g. transition networks et al.) for descriptions of complex menu systems. The specification of a menu-based user interface is much easier with the LC graphical design tool than with a textual language. If software tools are available in current graphical environments, the Lean Cuisine graphical notation could be used as the basis for an integrated interface developing environment in which menu-based interfaces will be very quickly built without programming.

3.5. Problems with Existing User Interface Developing Tools

Many current user interface management systems (UIMSs) provide a set of implementation tools, but little in the way of design aids. Most UIMSs use a textual specification and often require the user interface designer to learn a special purpose programming-like language, and with an ordinary text editor, it is not easy to create, debug, and modify a textual specification file for a complex user interface. Textual specifications are too complicated to be understand and used by non-programmers, so these language-based UIMSs are very hard to use [Myers, 88]. Very few UIMSs can provide software tools to support user interface development from user interface design, user interface simulation to direct implementation based on graphical description, and support the graphical user interface development lifecycle. Existing user interface development tools are not suitable for both interface designers and application developers.

3.6. An Integrated Interface Development Environment

In order to solve these problems and make user interface development much easier and quicker, research has been carried out to develop software tools used to support the Lean Cuisine graphical notation. The result of this research is an integrated interface development environment based on Lean Cuisine graphical notation. The integrated interface development environment is a combination of software tools used to support user interface design, simulation and implementation. The Lean Cuisine graphical editor (Elc) provides an interactive design environment for graphical specifications of menu-based interfaces. The simulation tool (Slc) provides an automatic prototype environment and allows user interface designers to simulate user interfaces based on a partial or whole dialogue specification at any design stage by a direct mapping process. A tool called user interface generator (Glc) is used to generate basic interface source code files for a user interface from its Lean Cuisine graphical specification file and allows the user interfaces to be implemented without programming. The menu-based user interface development process based on these three supporting tools is shown in Figure 3.17.

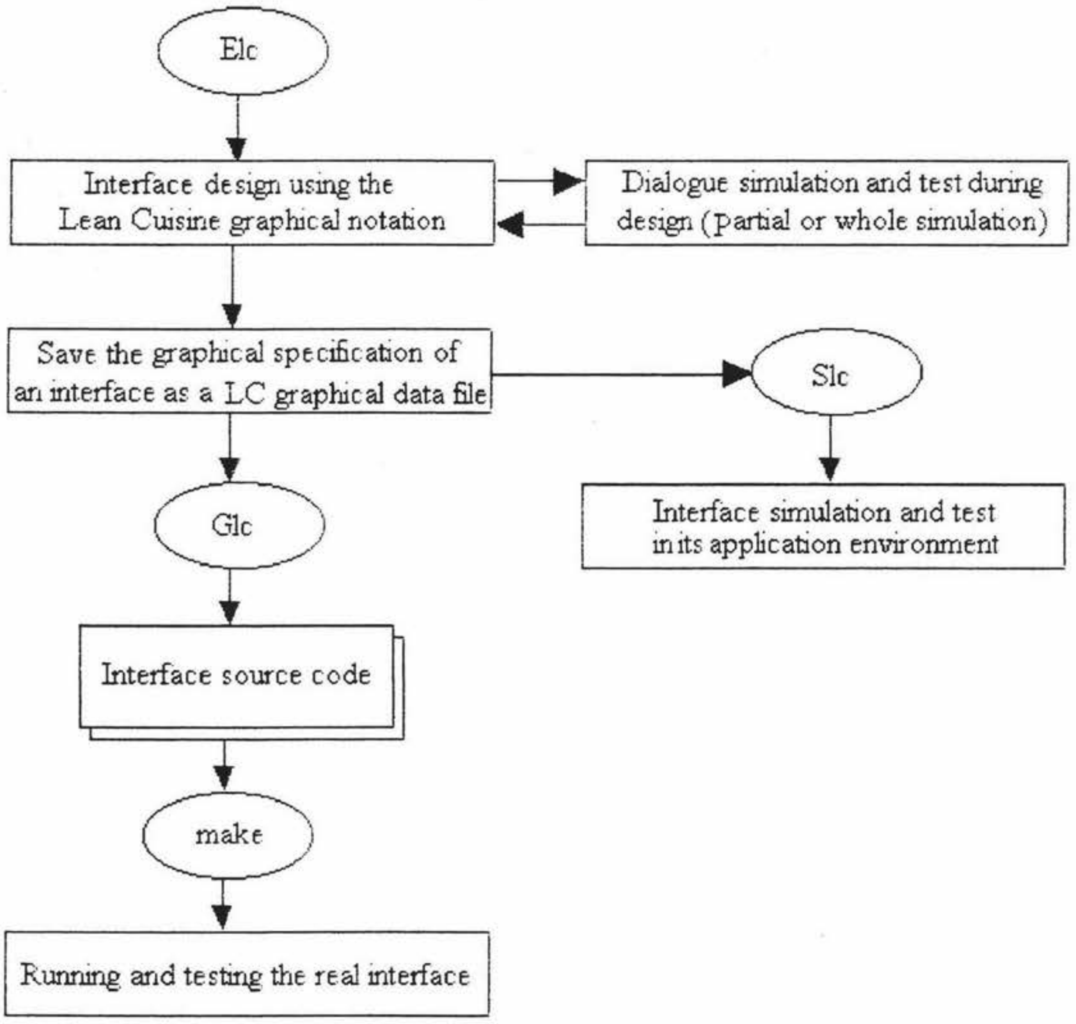

Elc
Interface design using the Lean Cuisine graphical notation
Dialogue simulation and test during design (partial or whole simulation)
Save the graphical specification of an interface as a LC graphical data file
Slc
Glc
Interface simulation and test in its application environment
Interface source code
make
Running and testing the real interface

Figure 3.17. The menu-based user interface development process based on three supporting tools.

The characteristic of the integrated interface development environment based on the Lean Cuisine graphical notation is that it provides software tools which make user interface development much easier and quicker from design and prototyping to implementation. The details of three supporting tools will be discussed in following chapters.

Chapter 4

The Lean Cuisine Graphical Editor

4.1. Introduction

The menu interface development environment is based on the Lean Cuisine visual notation [Apperley & Spence, 89]. Three supporting tools, Elc, Slc and Glc, have been developed for graphical dialogue specification, simulation and automatic generation of a menu-based interface. Elc is a graphical editor used by the interface designer to create a Lean Cuisine graphical description of a menu interface; Slc is a user interface simulator and Glc is a user interface generator which can automatically generate a user interface from its Lean Cuisine graphical specification.

This chapter describes the basic structure of Elc and discusses how a menu interface designer interactively designs the graphical dialogue specification by using Elc. The user interface simulation at the design stage, and the implementation environment of Elc will also be discussed.

Before describing the basic structure and functions of Elc, it is necessary to review the Lean Cuisine visual notation and examine an example of a Lean Cuisine description of a system of menus.

4.2. Menu Structures in the Lean Cuisine Visual Notation

As discussed in section 3.4.2.2, Lean Cuisine [Apperley & Spence, 89] is a visual notation which can be used as a diagrammatic user interface design tool. Two basic diagrams shown in Figure 4.1 are used to describe mutually exclusive (1-from-n) menu groups and mutually compatible (m-from-n) menu groups.

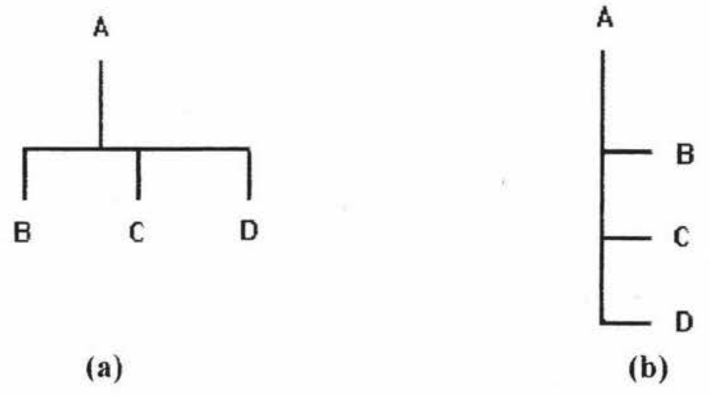


Figure 4.1. Two basic Lean Cuisine tree diagrams [Apperley, 88].
(a). The tree diagram for a mutually compatible menu group, and
(b). the tree diagram for a mutually exclusive menu group.

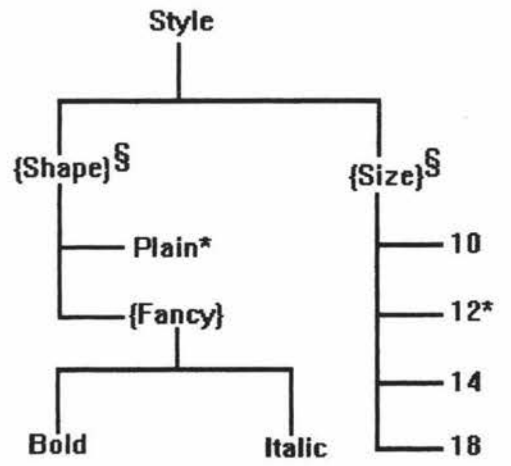
As summarised in [Apperley, 88], a menu is a set of selectable representations of actions, parameters, objects (which may be other menus), states, and other attributes, in which selections may be logically related and/or constrained.

A meneme is defined as an individual selectable representation within a menu, and has two basic states: selected and not selected. A meneme's state may be changed either by direct excitation, or by indirect modification. Menemes may be terminal (leaf) menemes, in which case they represent specific actions or parameters, or they may be non-terminal menemes, in which case they are themselves headers to other menus.

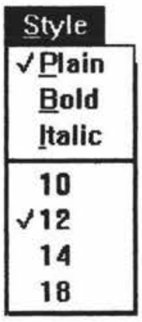
In order to partition a single menu into constituent syntactic sub-groups, virtual menemes are used and correspond to nodes in the Lean Cuisine tree. However they are not presented to, and not directly accessible to, the user. Virtual menemes are shown by enclosing their names in braces. A virtual meneme can be considered to be in the selected state if there is a valid selection present in the sub-section of the menu that it represents.

In the Lean Cuisine diagrammatic notation, many special characters are used as menu and meneme modifiers to indicate virtual menemes, required choices[†], default choices, dynamic defaults, bistable menemes, select-only menemes, deselect-only menemes, monostable menemes and passive menemes. Figure 4.2 shows a typical example of a menu taken from the TaP [Apperley, 88] interactive application. The Lean Cuisine description of the text style and size facilities is shown in Figure 4.2(a), and a possible presentation of the *Style* menu is shown in Figure 4.2(b).

[†] In the Elc system, a special character ^ is used to indicate a required choice.



(a)



(b)

Figure 4.2. (a). The Lean Cuisine description of the text style and size facilities, and **(b).** a possible presentation of the *Style* menu.

In the Lean Cuisine description of the *Style* menu, *Shape*, *Size* and *Fancy* are virtual menemes which represent three menu sub-groups. So this Lean Cuisine diagram describes a single menu. In the *Size* sub-group, only one size is possible at any one time, so the Lean Cuisine tree diagram for a mutually exclusive group is used to describe different text size (*10,12,14,18*). In the *Shape* sub-group, the possible text shape combinations are plain, bold, italic, and bold-italic. So *Bold* and *Italic* are described by a Lean Cuisine tree diagram for a mutually compatible group, and *Plain* and *{Fancy}* are described by a Lean Cuisine tree for a mutually exclusive group.

A size and a shape are both always required, so a special character § is marked on *Shape* and *Size* to indicate a required choice. The character * which is marked on menemes *Plain* and *12* is used in Figure 4.2(a) to indicate a default choice in the *Shape* and *Size* sub-groups.

The main aim of developing the Elc program is to implement the Lean Cuisine notation described in [Apperley & Spence, 89] in a practical computer environment and to provide an interactive Lean Cuisine menu interface design tool which is independent of the implementation and suitable for both non-programming and programming interface designers. The computer description of the Lean Cuisine specification of a menu interface in the Elc system can be used by Slc to simulate the user interface and by Glc to generate the user interface code files. The details of Slc and Glc will be discussed in Chapter 5 and Chapter 6.

4.3. Overview of Elc

Elc is a window based graphical editor. To a user interface designer, it seems like an interactive drawing package such as AutoCad [Omura, 92], CATIA's DRAFTING [Dassault, 87], and MacDraw [Dilback, 1991]. In general CAD software systems, the basic drawing elements are lines, curves and so on, but in the Elc system, the basic drawing elements are the two basic Lean Cuisine tree structures shown in Figure 4.1.

The Elc graphical editor has two modes, the design mode and the simulation mode. In the design mode, the base window of the Elc system includes the menu-bar which contains menu buttons and command buttons, the canvas pane, the vertical scrollbar and the horizontal scrollbar, as shown in Figure 4.3.

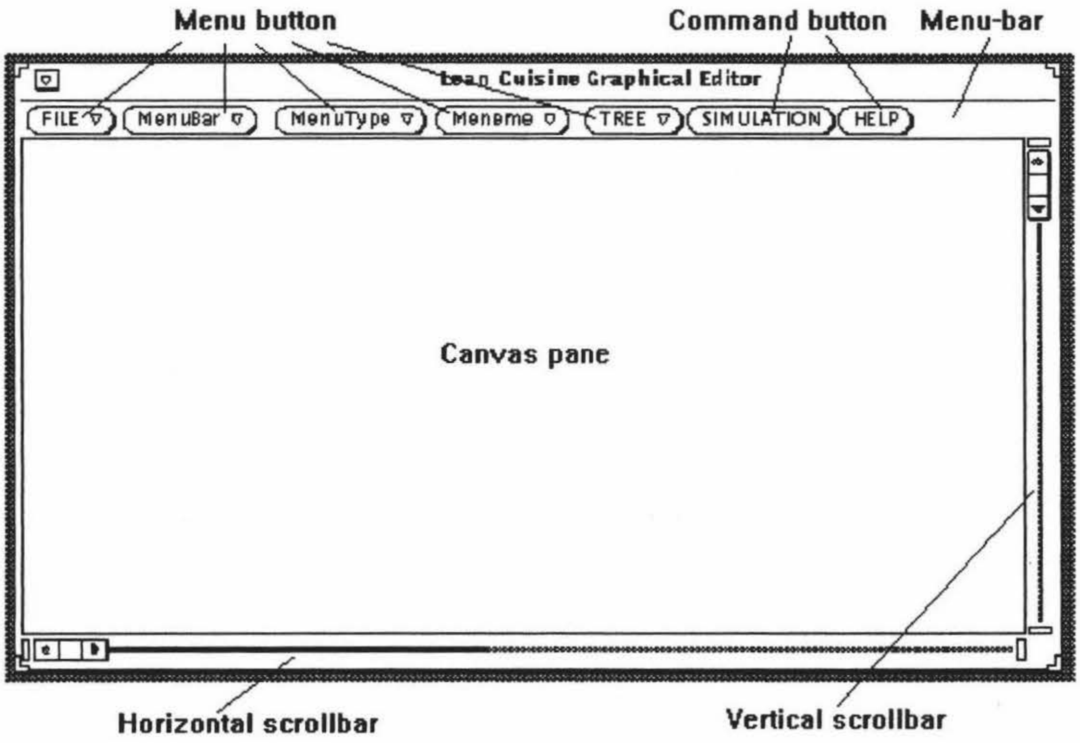


Figure 4.3. The base window of Elc.

In the base window shown above, the menu-bar provides the pull-down menus and the command buttons which will be used by the user interface designer to give commands for performing the basic functions of Elc; these pull-down menus and command buttons are shown in Figure 4.4a and Figure 4.4b.

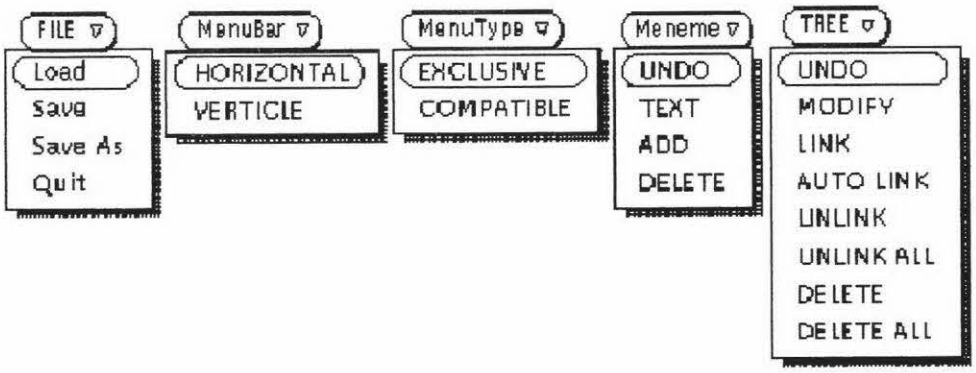


Figure 4.4a. The pull-down menus of the design mode of Elc.

Figure 4.4b. The command buttons of the design mode of Elc.

In the simulation mode of Elc, all menu buttons and command buttons are grayed (disabled) except the *HELP* and *GO TO DESIGN* command buttons. Figure 4.5a shows the menu-bar of the Elc system which is in the simulation mode.

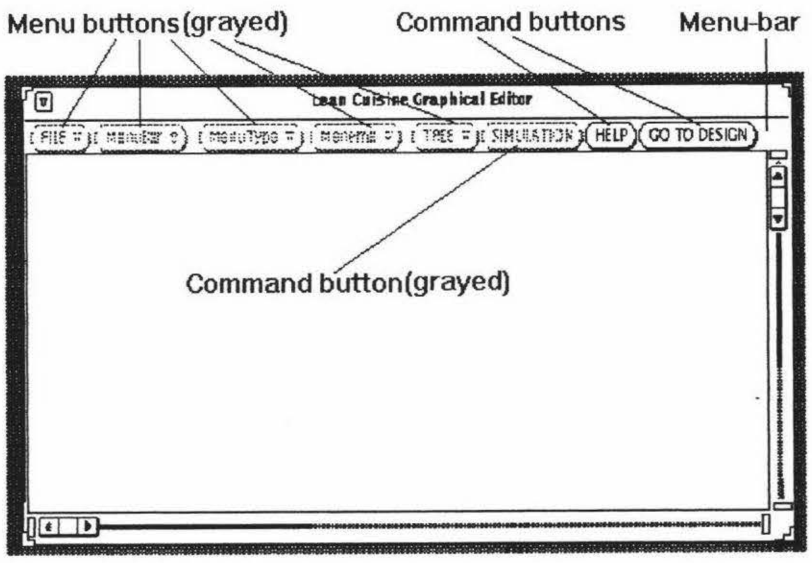


Figure 4.5a. The menu-bar of the simulation mode of the Elc system.

Two command buttons which are available in the simulation mode of Elc are shown in Figure 4.5b.

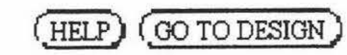


Figure 4.5b. The command buttons of the simulation mode of Elc.

The canvas pane provides a drawing surface where the Elc system displays Lean Cuisine trees and where an interface designer can create new trees and edit existing trees for graphical specification of an interface. The vertical scrollbar and horizontal scrollbar are used to scroll a large Lean Cuisine diagrams in the pane.

In the canvas pane, mouse buttons are used to select (left button) a Lean Cuisine tree which will become the currently selected Lean Cuisine tree, or to drag (middle button) the currently selected Lean Cuisine tree to an appropriate position, or to indicate (right button) a new position for the currently selected Lean Cuisine tree, which will directly move to this new position.

The accurate position of the mouse selection on a Lean Cuisine tree will be recorded by the Elc system when this Lean Cuisine tree becomes the currently selected Lean Cuisine tree. The meneme and tree operations on the currently selected Lean Cuisine tree will use this position message to determine their operation position when the designer wants to name a meneme, delete a selected meneme, to add a meneme to an appropriate position of the currently selected Lean Cuisine tree (Section 4.6), or to modify the local geometric shape of the currently selected Lean Cuisine tree (section 4.7), etc.

To define a menu-based interface, the first step is to create different Lean Cuisine trees, then edit them and finally link them to form the graphical dialogue specification of a menu interface. To do this, the designer selects different Lean Cuisine tree types (horizontal or vertical menu-bar, exclusive or compatible menu) from the pull-down menu and inputs a parameter (the meneme number of a Lean Cuisine tree) from a pop-up input window, which appears after the designer selected a related menu command and disappears after a parameter has been input. By using related menu commands, mouse selection operations and keyboard inputs, the designer can add text to the meneme as its name or modifier, delete a meneme from the currently selected Lean Cuisine tree or insert a meneme into the menemes of the currently selected Lean Cuisine tree. Lean Cuisine trees created in a design window of Elc can be linked to form a graphical dialogue specification of a menu interface.

The designer can modify dialogue specifications by using the menu commands or by creating new Lean Cuisine trees and linking new Lean Cuisine trees into a dialogue specification of a user interface. The geometric shape of a Lean Cuisine tree can be modified (section 4.7) for general layout of all Lean Cuisine trees. Some tree group operation commands are available for rapid tree operations. During graphical dialogue specification or after finishing graphical dialogue specification for a menu-based user interface, the designer can simulate the whole or the parts of the menu system by using the Elc simulation function.

The Lean Cuisine graphical specification created in an Elc design window can be saved as a Lean Cuisine graphical specification file with the extension ".lc", and a previously saved Lean Cuisine graphical specification file can be loaded into an Elc design window.

All of the menu and button commands will be discussed in this and subsequent chapters.

4.4. The Basic Graphical Primitives

The Elc system provides two basic graphical primitives shown in Figure 4.6 to represent the Lean Cuisine tree diagrams for a mutually compatible menu groups and a mutually exclusive menu groups.

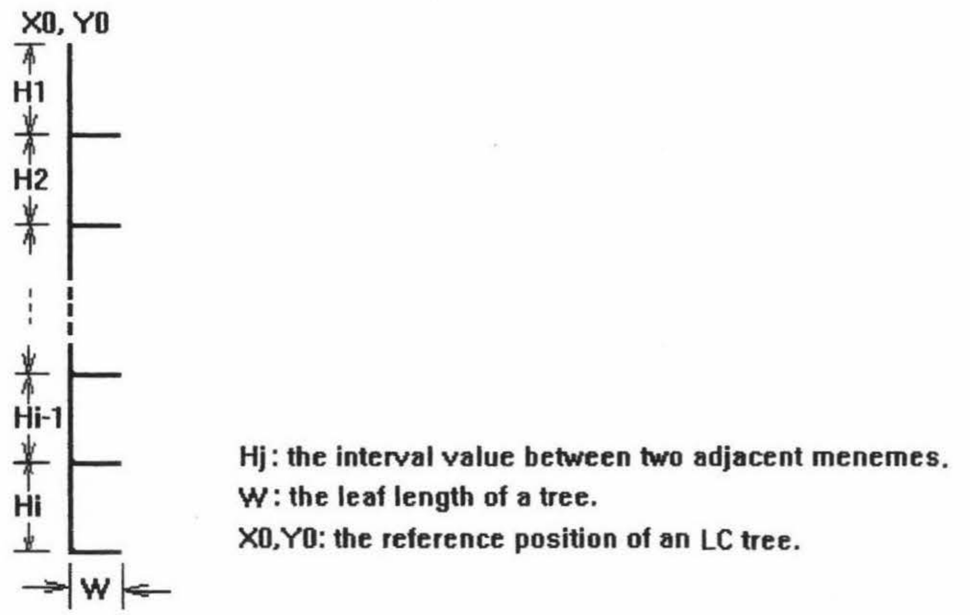
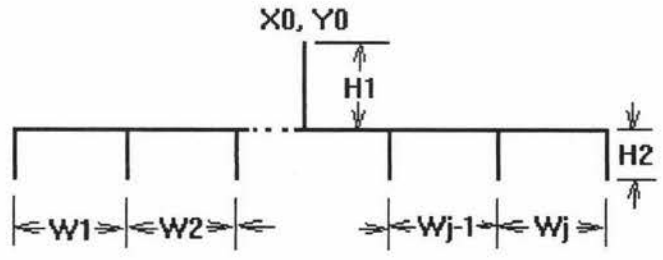

X0, Y0
H1
H2
Hi-1
Hi
W
Hj: the interval value between two adjacent menemes.
W: the leaf length of a tree.
X0,Y0: the reference position of an LC tree.

Figure 4.6a. The graphical primitive representing a mutually exclusive menu group.



H1: the root length of a tree.
H2: the leaf length of a tree.
Wi: the interval value between two adjacent menemes.
X0,Y0: the reference position of an LC tree.

Figure 4.6b. The graphical primitive representing a mutually compatible menu group.

Each graphical primitive has various attributes which are under the control of the interface designer. Some of the attributes of a graphical primitive are initialized by the Elc system when this graphical primitive is created. These attributes are tree type, tree reference position, tree leaf length, interval value between two adjacent menemes, and colour. Table 4.1 lists all the attributes of a graphical primitive and their initial values.

Attribute name	Initial value
Tree type	Number (0, 1, 2, or 3)
Tree reference position	Constant
Tree leaf length	Constant
Meneme number of a tree	Number (1~20)
Interval value between two adjacent menemes (array)	Constant
Subtrees (pointer array)	Nil
Parent tree pointer	Nil
Meneme names (pointer array)	Nil
Meneme modifier	Nil
Colour	Black

Table 4.1. The attributes of a graphical primitive and their initial values.

Some of the attributes of a graphical primitive can be set by the designer. These attributes are parent tree pointer, subtree pointers, meneme names and meneme modifiers. Other attributes like meneme number can be input from a pop-up window before a tree is created. All of these attributes can be modified by the designer during graphical dialogue specification.

When basic Lean Cuisine trees are linked to construct more complex dialogue specifications, some attributes (except colour) of each Lean Cuisine tree will be automatically modified by the Elc system according to the designer's action.

Based on the two basic graphical primitives described above, four different basic Lean Cuisine tree diagrams are used in Elc to represent the mutually exclusive menu group, the mutually compatible menu group, the horizontal menu-bar and the vertical menu-bar. The examples of these four basic Lean Cuisine tree diagrams are shown in Figure 4.7.

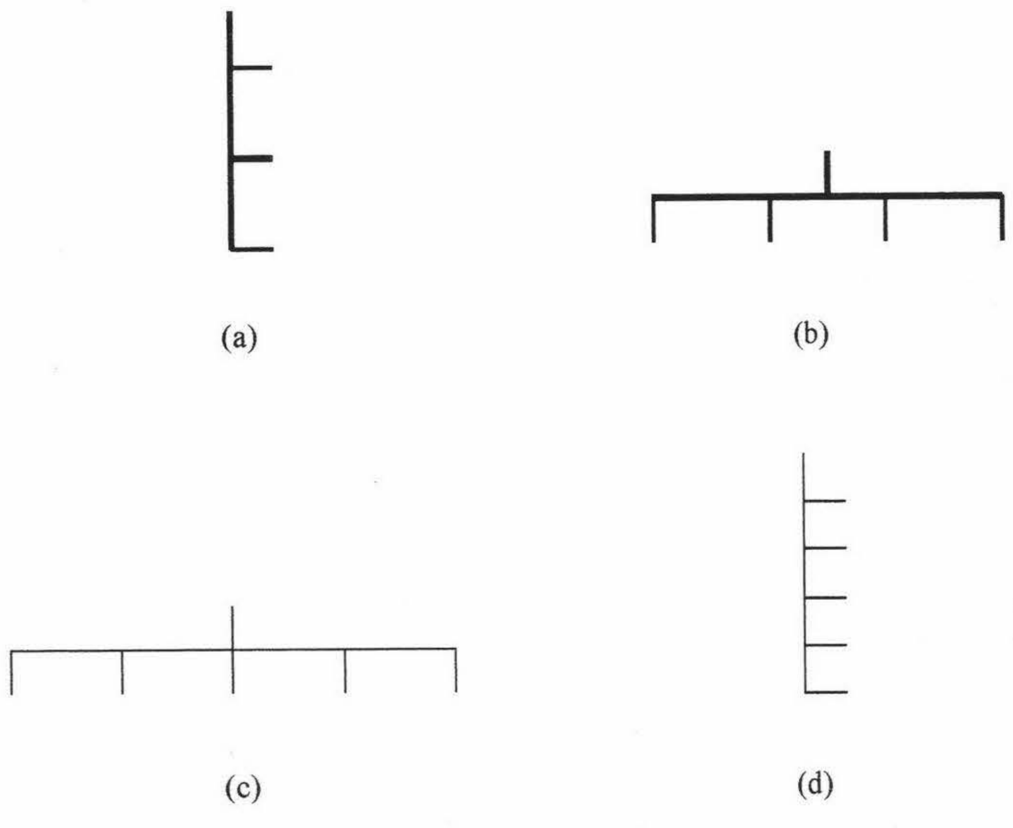


Figure 4.7. The examples of the four basic Lean Cuisine tree diagrams used in Elc. **(a)**: A vertical menu-bar (tree type = 0, meneme number = 3); **(b)**: a horizontal menu-bar (tree type = 1, meneme number = 4); **(c)**: a mutually compatible menu group (tree type = 2, meneme number = 5); and **(d)**: a mutually exclusive menu group (tree type =3, meneme number = 5).

4.5. Creating Basic Lean Cuisine Trees

In order to produce the graphical dialogue specification for a menu-based interface, the first step is to create different Lean Cuisine trees in an Elc design window, then edit them and finally link them to form the graphical dialogue specification of a menu interface. A Lean Cuisine tree is created based on its type and meneme number. After a Lean Cuisine tree type has been selected from the pull-down menu *MenuBar* or *MenuType* and a meneme number of a Lean Cuisine tree has been input from the pop-up input window, a new Lean Cuisine tree will be displayed on the Elc editing window. Figure 4.8 shows the procedure used to create a LC tree.

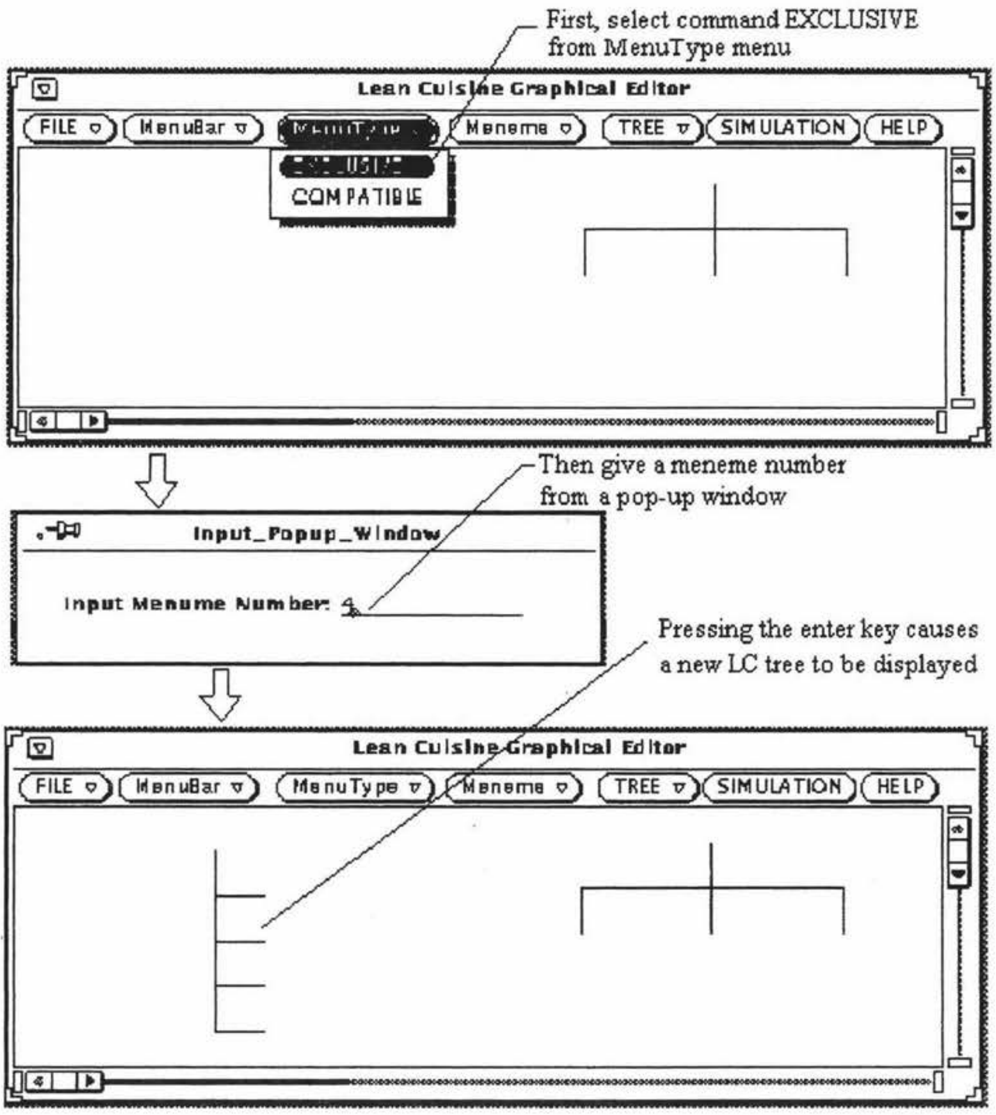

First, select command EXCLUSIVE from MenuType menu
Lean Cuisine Graphical Editor
FILE
MenuBar
MenuType
Meneme
TREE
SIMULATION
HELP
EXCLUSIVE
COMPATIBLE
Then give a meneme number from a pop-up window
Input_Popup_Window
Input Meneme Number: 4
Pressing the enter key causes a new LC tree to be displayed
Lean Cuisine Graphical Editor
FILE
MenuBar
MenuType
Meneme
TREE
SIMULATION
HELP

Figure 4.8. The procedure used to create a LC tree.

All of Lean Cuisine trees are free trees before the designer adds constraints to them. The designer can put a Lean Cuisine tree at any new position in the editing window by selecting it and indicating a new position for it. If the designer want to move a Lean Cuisine tree which has been linked to other Lean Cuisine trees, for example, it has parent tree or subtrees; the Elc system will launch a dialogue box which includes a constraint message of the currently selected Lean Cuisine tree. The command buttons *CANCEL* and *UNLINK_AND_MOVE* displayed in this dialogue box provide two possible operations. If the designer selects the *CANCEL* button from this dialogue box, the move operation will be cancelled. If the designer selects the *UNLINK_AND_MOVE* button from this dialogue box, the Elc system will remove all link constraints from the currently selected Lean Cuisine tree and modify attributes of the currently selected Lean Cuisine tree, attributes of its parent tree and attributes of its subtrees. After finishing attribute modification, the currently selected Lean Cuisine tree moves to a new position and becomes a free Lean Cuisine tree.

4.6. Basic Meneme Operations

Basic interactive operations on each meneme are provided by the Elc system. The interface designer can interactively add a meneme to the currently selected Lean Cuisine tree at an appropriate position, delete a meneme from the currently selected Lean Cuisine tree, and attach a meneme name and semantic message to each meneme. The *ADD* command and the *DELETE* command are very useful when the designer wants to modify an existing Lean Cuisine tree.

To add a meneme to a Lean Cuisine tree, first select a meneme from a Lean Cuisine tree which will become the currently selected Lean Cuisine tree after selection, and afterwards select the menu command *ADD* from the pull-down menu *Meneme*. A new meneme will be inserted after the selected meneme. Some of attributes of the currently selected Lean Cuisine tree will be modified by the Elc system after new meneme's insertion. It would not be difficult to modify the Elc system to allow the designer to insert a new meneme before a meneme or insert a new meneme at selection point which is on a tree trunk. Figure 4.9 shows the procedure used to add a meneme.

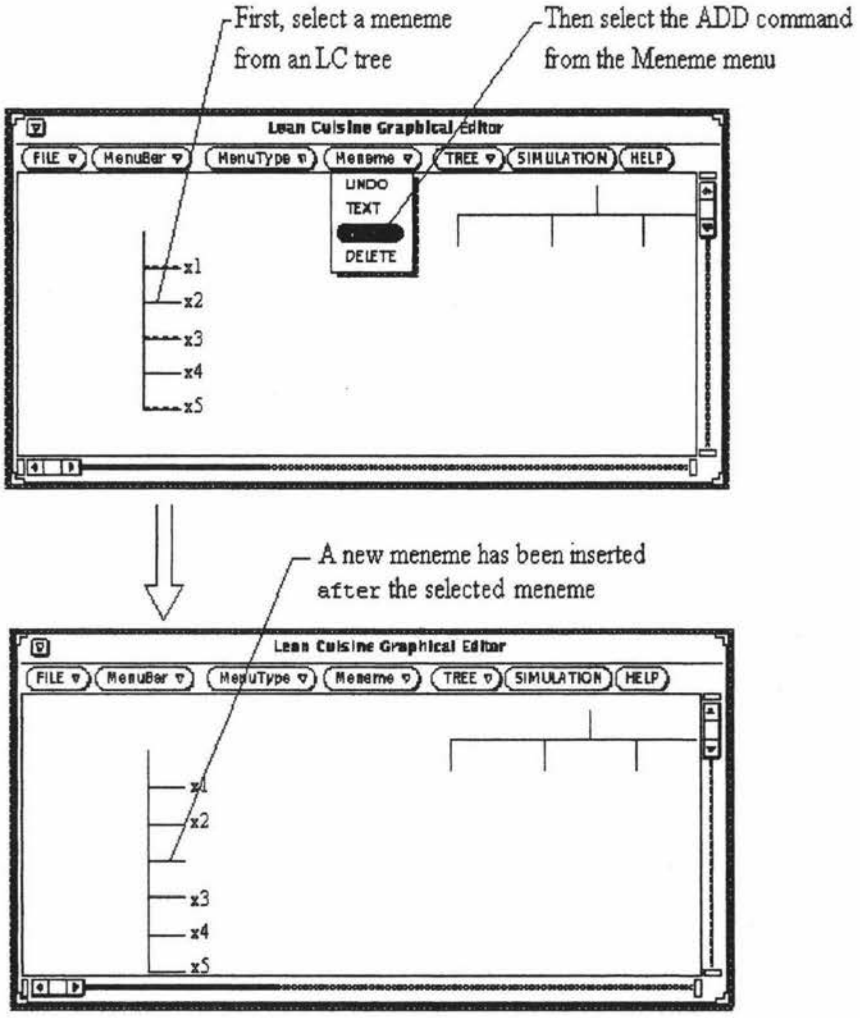

First, select a meneme from an LC tree
Then select the ADD command from the Meneme menu
Lean Cuisine Graphical Editor
FILE
MenuBar
MenuType
Meneme
TREE
SIMULATION
HELP
UNDO
TEXT
DELETE
x1
x2
x3
x4
x5
A new meneme has been inserted after the selected meneme

Figure 4.9. The procedure used to add a meneme.

To delete a meneme from a Lean Cuisine tree, first select the meneme to be deleted and then select the menu command *DELETE* from the pull-down menu *Meneme*; the selected meneme will be deleted from the currently selected Lean Cuisine tree. Similar to a meneme *ADD* operation, related attributes of the currently selected Lean Cuisine tree will be modified by the Elc system after a meneme deletion. The Elc system will automatically delete a Lean Cuisine tree from the graphical specification of a menu interface if all of its menemes have been deleted by the designer.

The menemes of a Lean Cuisine tree are pure geometric representations when the interface designer creates the Lean Cuisine tree by using menu commands selected from pull-down menu *MenuBar* or *MenuType*. The meneme name, meneme modifier (it also is submenu modifier if a meneme has a submenu) and the application link message must be added to the menemes by the designer. To do this, first select a meneme, and then select the menu command *TEXT* from the pull-down menu *Meneme*. A pop-up window which is for text input appears and the designer can input a meneme name, or a meneme modifier, or an application link message from this pop-up window. After finishing the typing of text, pressing the enter key causes the pop-up window to disappear and the attributes of the currently selected Lean Cuisine tree to be modified by the Elc system, and the content of the Elc design window to be redrawed. Figure 4.10 shows the procedure used to add a meneme name.

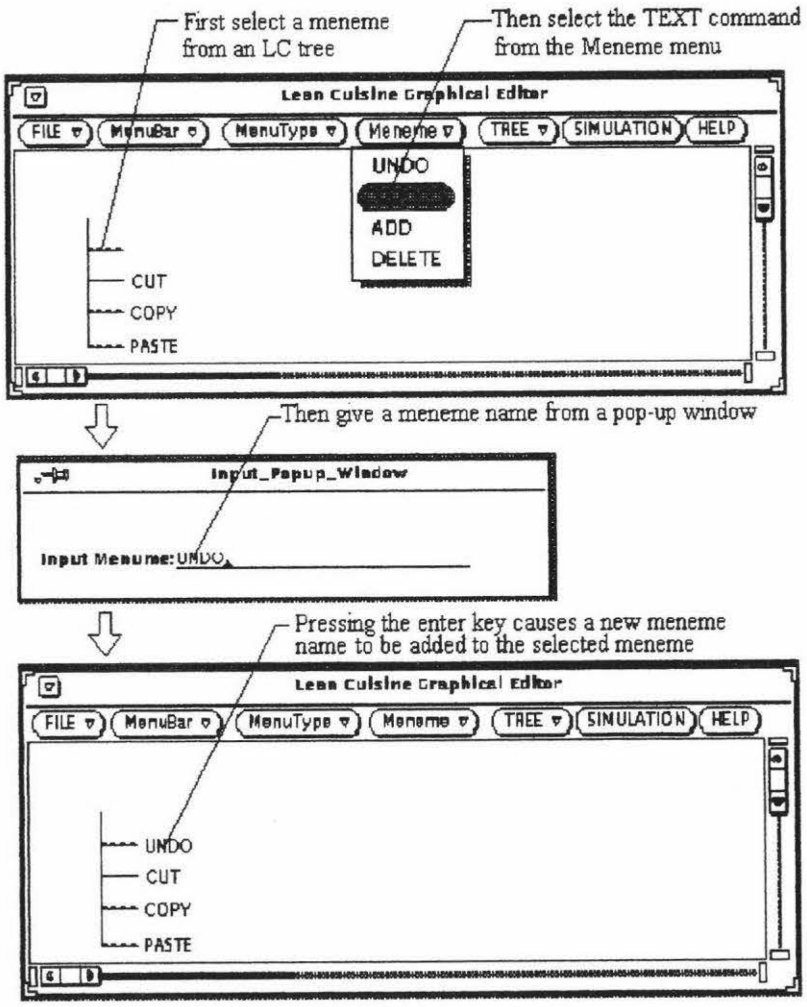

First select a meneme from an LC tree
Then select the TEXT command from the Meneme menu
Lean Cuisine Graphical Editor
FILE
MenuBar
MenuType
Meneme
TREE
SIMULATION
HELP
UNDO
ADD
DELETE
CUT
COPY
PASTE
Then give a meneme name from a pop-up window
Input_Popup_Window
Input Meneme: UNDO
Pressing the enter key causes a new meneme name to be added to the selected meneme
Lean Cuisine Graphical Editor
FILE
MenuBar
MenuType
Meneme
TREE
SIMULATION
HELP
UNDO
CUT
COPY
PASTE

Figure 4.10. The procedure used to add a meneme name.

The pop-up window for text input can be kept on the screen for repeated operations. To do this, the designer clicks the SELECT button on the pushpin to pop it into the hole. The pop-up window is then pinned to the workspace, and it remains on the screen until the designer either unpins it, or dismisses it using the window menu. In this case, the designer does not need to select the menu command *TEXT* from the pull-down menu *Meneme* after meneme selection when he/she wants to continuously add text strings to different menemes.

The *UNDO* command which is in pull-down menu *Meneme* is available to cancel *ADD* a meneme or *DELETE* a meneme operation.

4.7. Basic Tree Operations

Basic tree operations are provided by the Elc system to modify interactively the tree shape for the layout of a menu specification and to link trees to form a complex menu structure. Menu commands on a single tree, which is the currently selected Lean Cuisine tree, and menu commands on all Lean Cuisine trees are available so the designer can have different command selections for efficient tree operations.

The basic geometric shape of a Lean Cuisine tree is determined by the Elc system which uses system default (initial) values to determine the geometric attributes of a graphical primitive. The local geometric shape of a Lean Cuisine tree can be modified at any time after its creation to avoid Lean Cuisine trees overlapping when linking them to form a complex graphical specification of a menu system.

To modify the shape of a Lean Cuisine tree, first select the part which will be modified, and then select the menu command *MODIFY* from pull-down menu *TREE*. A pop-up window appears on the screen after the *MODIFY* menu command selection, and a parameter P for the tree modification is typed into the input field of the pop-up window. The parameter input procedure is similar to the text string input procedure; pressing the enter key causes the Elc system to check the input parameter; if it is valid, the pop-up window is dismissed, and the shape of the selected part of the Lean Cuisine tree is changed according to the input parameter, and the redrawing procedure happens in the Elc design window. If it is invalid, an error message will be displayed on the screen. Repeated modifying operations can be done when the pop-up window is kept on the screen.

For a Lean Cuisine tree which represents a mutually exclusive menu group, its shape can be modified according to its tree leaf length or the interval value between two adjacent menemes (including the tree root).

To modify the leaf length of a tree, select one of the leaves of the tree, and select the menu command *MODIFY* from the pull-down menu *TREE*, then type a valid parameter P into the input field of the pop-up window. All the leaves of the tree will be modified at the same time according to the parameter P.

To modify an interval value between two adjacent menemes (including the tree root) of a tree, from the tree, select the line segment which is between two adjacent menemes ,and select the menu command *MODIFY* from pull-down menu *TREE*, then type a valid parameter P into the input field of the pop-up window; the interval value which was the length of the selected line segment will be modified according to the parameter P so the shape of the tree will be modified.

The valid range of the parameter P is greater than 0 when the modified LC tree represents a mutually exclusive menu group. Figure 4.11 shows the original shape of a Lean Cuisine tree (tree type = 3, meneme number = 5) which represents a mutually exclusive menu group and its possible shapes after modification.

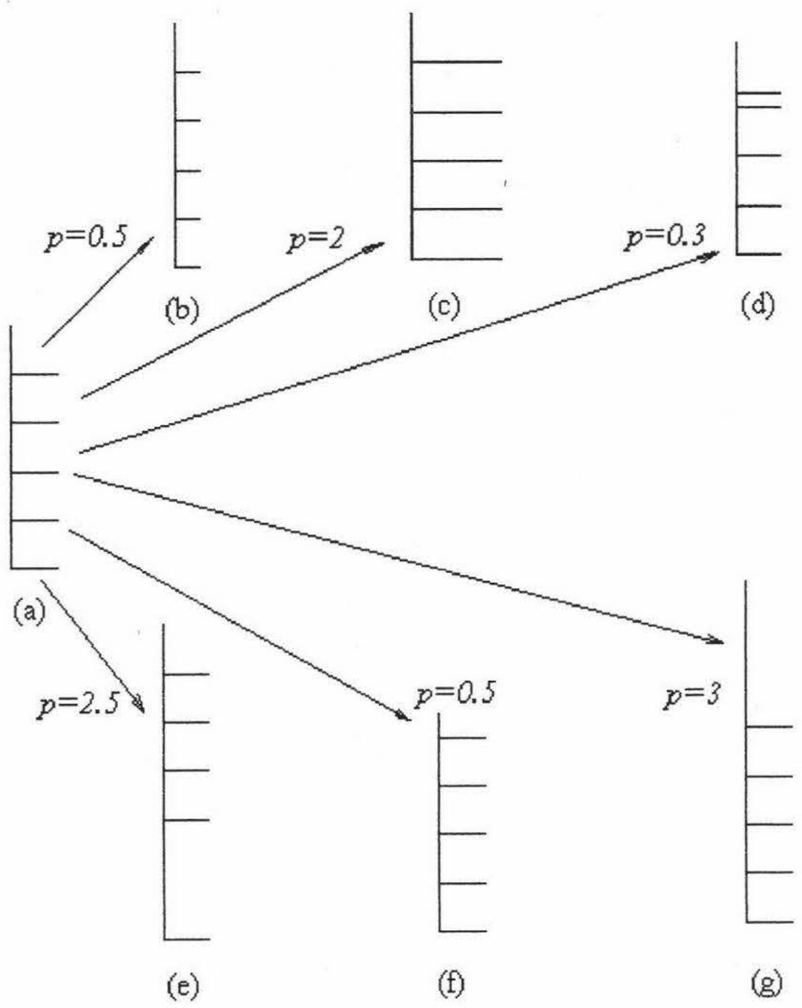


Figure 4.11. The original shape of a Lean Cuisine tree which represents a mutually exclusive menu group and its possible shapes after modification. **(a).** The original shape; **(b)(c).** after changing its tree leaf length; **(d)(e).** after changing one of the interval value between two adjacent menemes; **(f)(g).** after changing the interval value of the tree root.

For a Lean Cuisine tree which represents the mutually compatible menu group, except tree root the modification of its shape according to its tree leaf length or the interval value between two adjacent menemes is in the same way as the modification for a Lean Cuisine tree which represents a mutually exclusive menu group; to modify the root of a tree which represents the mutually compatible menu group, select the root of the tree first, then select the menu command *MODIFY* from the pull-down menu *TREE*, and type a valid parameter P into the input field of the pop-up window, the root of the tree will be moved in horizontal direction according to the parameter P. The valid range of the parameter P is greater than -2000 and less than 2000, if the parameter P is negative, the root of the tree will be horizontally moved to the left of the original position of the tree according to the absolute value of the parameter P; if the parameter P is positive, the root of the tree will be horizontally moved to the right of the original position of the tree according to the parameter P. Figure 4.12 shows the original shape of a Lean Cuisine tree (tree type = 2, meneme number = 4) which represents a mutually compatible menu group and its possible shapes after modification.

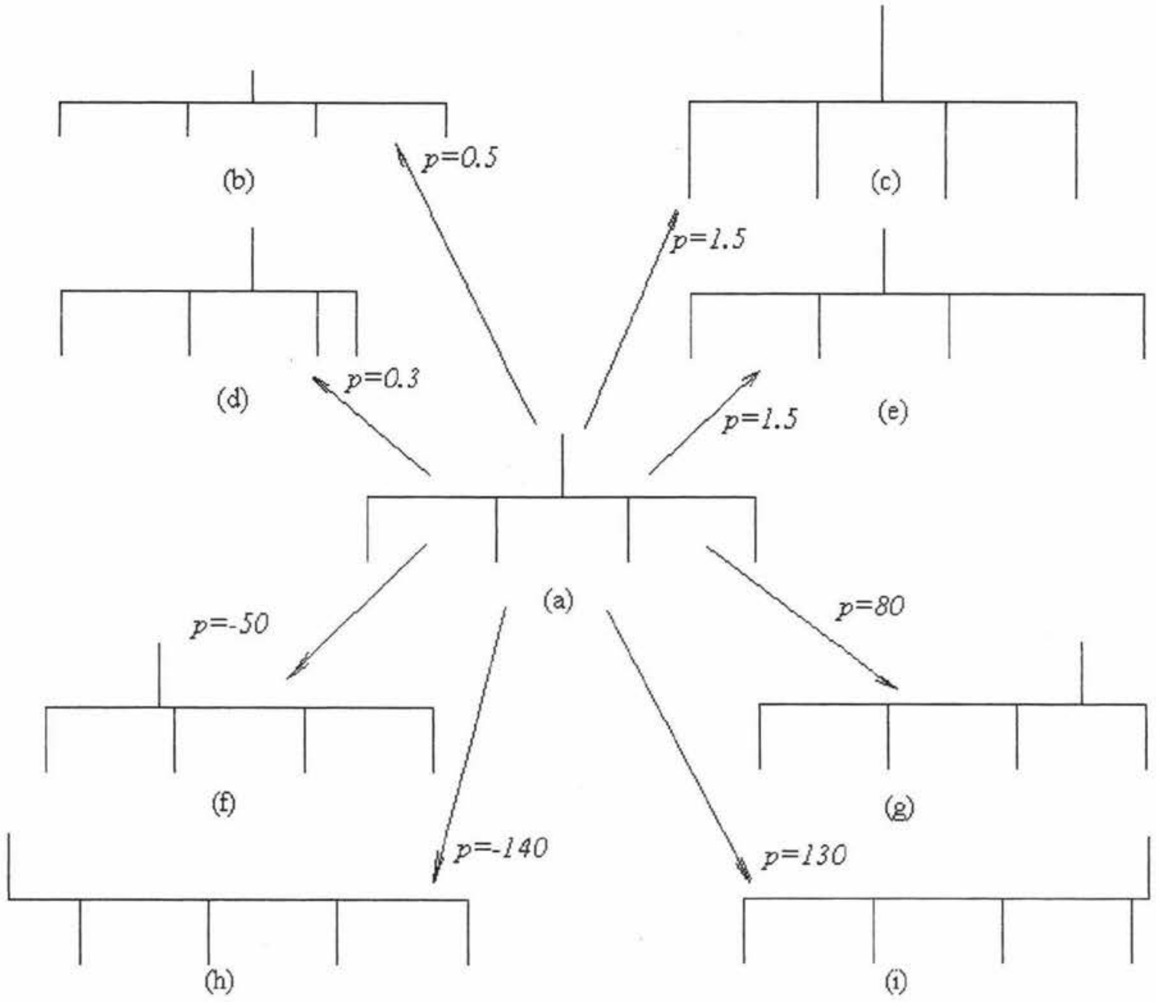


Figure 4.12. The original shape of a Lean Cuisine tree which represents a mutually compatible menu group and its possible shapes after modification. **(a)**. The original shape of a LC tree; **(b)(c).** after changing its tree leaf length; **(d)(e).** after changing one of the interval value between two adjacent menemes; **(f)(g)(h)(i).** after changing the position of the tree root.

The *LINK* menu command selected from the pull-down menu *TREE* is used to create a relationship between a parent tree and a subtree to form a complex menu structure. A subtree can be linked to its parent tree only when its reference position is close to a meneme of its parent tree. If the link operation is successful, the linked subtree will be highlighted and some attributes of both the subtree and its parent tree will be modified by the Elc system, or an error message will be displayed on the screen. If a meneme and its subtree cannot be linked, the position or shape of the trees should be modified to satisfy the link condition. Figure 4.13 shows a design example, an LC graphical specification of a menu-based interface is formed in an Elc design window and ready to be simulated after linking related LC trees.

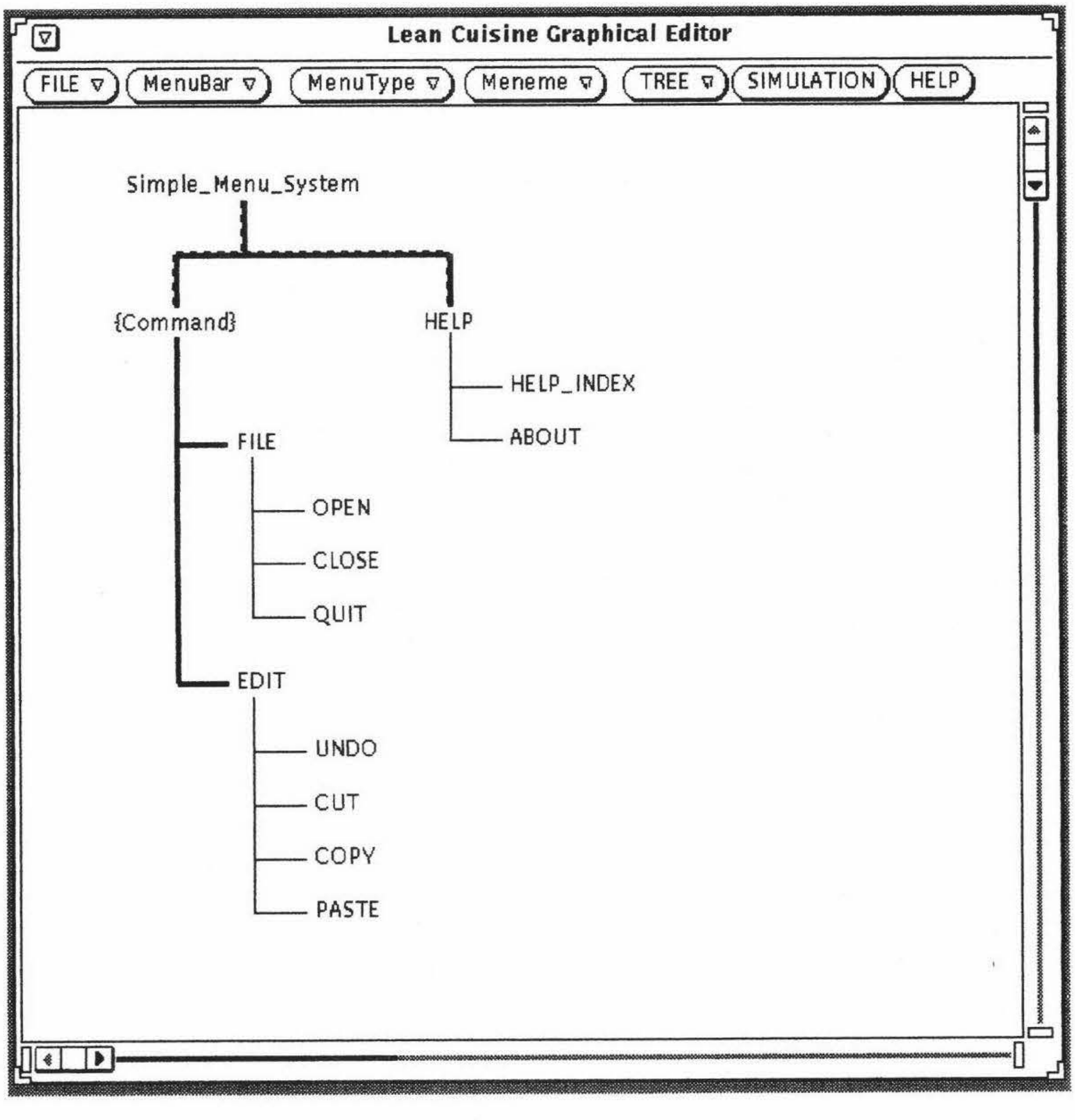


Figure 4.13. The LC graphical specification of a user interface.

The *UNLINK* menu command is used to destroy the relationship created by the *LINK* command between two trees. After an *UNLINK* operation, some attributes of the related trees will be modified by the Elc system.

The *DELETE* menu command is used to delete the currently selected Lean Cuisine tree. If the *DELETE* command is selected, the currently selected Lean Cuisine tree will disappear from the design window. The *DELETE ALL* menu command is a group operation command which is used to delete all Lean Cuisine trees from the design window, the all trees will disappear from the design window after the *DELETE ALL* operation.

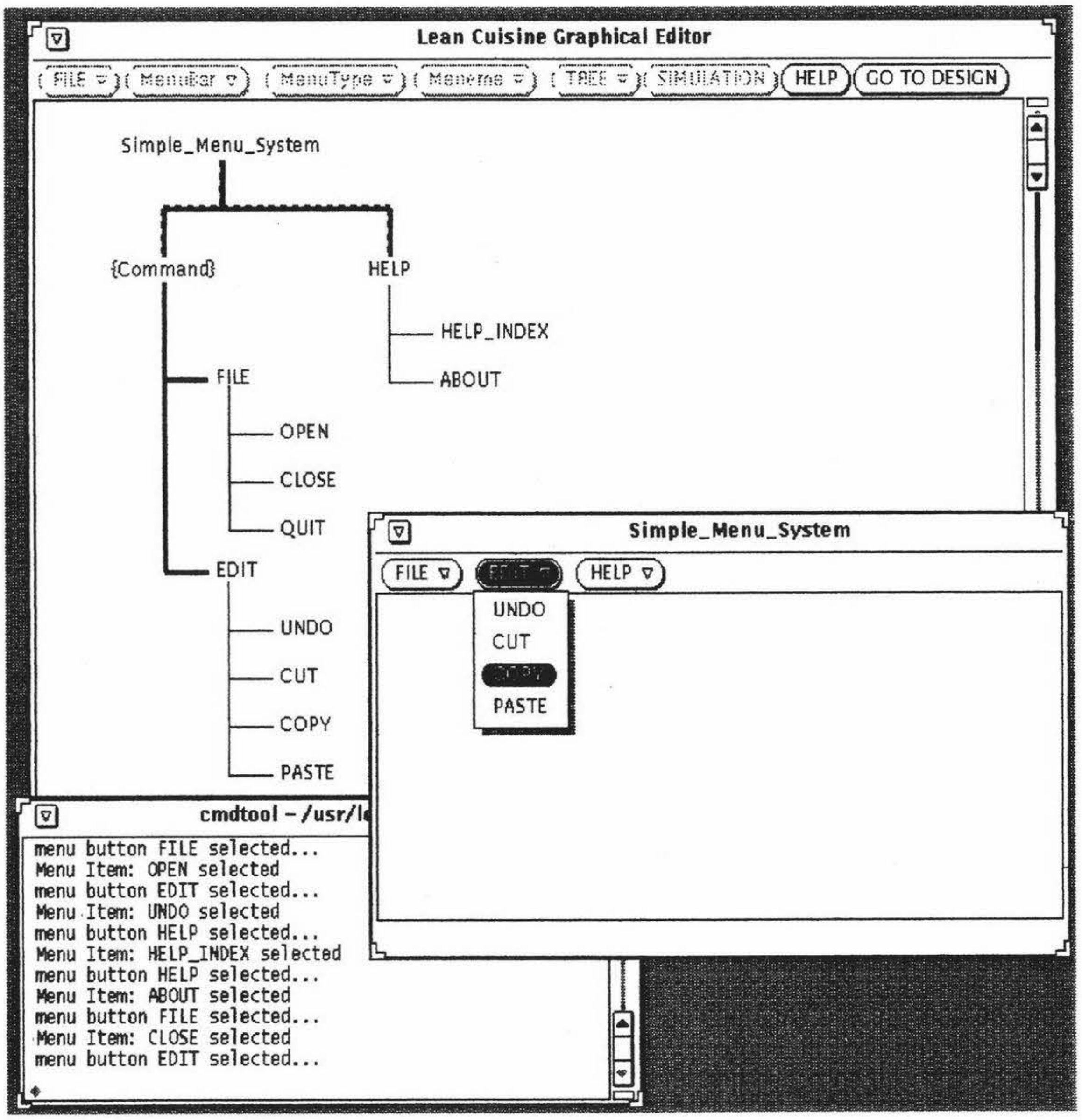
If no Lean Cuisine tree has been created since the tree deletion, the deleted tree or trees can be restored by using the *UNDO* command selected from the pull-down menu *TREE*, the restored tree or trees will be redisplayed on the design window if the restoration procedure is successful. Also the *UNDO* menu command can be used to cancel the last *MODIFY* operation.

The *AUTO LINK* command and the *UNLINK ALL* command are other group operation commands; the *AUTO LINK* command is used to create all possible links between trees according to their positions; and the *UNLINK ALL* command is used to destroy the all link relationships from existent graphical specification of a menu-based user interface. The aim of using the group operation commands *AUTO LINK*, *UNLINK ALL* and *DELETE ALL* is to provide rapid tree operations for a complex menu structure specification.

4.8. Menu Interface Simulation at the Design Stage

Rapid prototyping within a UIMS environment is probably the most efficient method for supporting the design-build-test nature of a user interface construction process [Wilson, 88]. The Elc system provides a simulation environment in which the designer can simulate and test a menu-based user interface which has been designed in an Elc design window by using the Lean Cuisine graphical notation. When starting Elc, the default mode of Elc is the design mode. The designer can change the mode of Elc from the design mode to the simulation mode by selecting the command button *SIMULATION* from the menu-bar. In simulation mode, the lc_run_time kernel is used to generate a user interface prototype from its Lean Cuisine graphical specification, and dynamically create, delete and manage menu structures according to its Lean Cuisine graphical specification and the user's actions.

The menu structure of the user interface designed in an Elc design window can be partially or wholly simulated. Before selecting the *SIMULATION* button, a Lean Cuisine tree which describes a menu-bar must be selected. If it is a root of the Lean Cuisine trees, the whole menu interface will be simulated in the simulation window (whole simulation); if it is a subtree, only this tree and its subtrees can be simulated in the simulation window (partial simulation). Figure 4.14 shows an example of interface simulation based on the graphical dialogue specification shown in Figure 4.13. The details of interface simulation and lc_run_time kernel will be discussed in Chapter 5.



```
menu button FILE selected...
Menu Item: OPEN selected
menu button EDIT selected...
Menu Item: UNDO selected
menu button HELP selected...
Menu Item: HELP_INDEX selected
menu button HELP selected...
Menu Item: ABOUT selected
menu button FILE selected...
Menu Item: CLOSE selected
menu button EDIT selected...
```

Figure 4.14. The direct simulation of a menu interface from its Lean Cuisine description created in an editing window of Elc.

In Figure 4.14, the front window is the simulation window of a user interface; the back window is the design window which is in the simulation mode and contains the Lean Cuisine visual specification of the user interface; and the middle window is the command window used to display the feedback message when testing the interface.

A menu tree which describes a menu structure and is not attached to a menu-bar cannot be simulated because in the current Elc system, the precondition of setting the simulation mode of Elc is that a Lean Cuisine tree which represents a menu-bar must be selected. But it is easy to modify the Elc system in the future to simulate this type menu tree as a pop-up menu or a pull-down menu if it has been selected for simulation.

4.9. Lean Cuisine Graphical Specification File

A special file format is used by the Elc system to store the Lean Cuisine graphical specifications of user interfaces. The Lean Cuisine graphical specification file of a user interface is a text file which contains the attributes of all Lean Cuisine trees created in a design window of Elc. Because it stores an interface without reference to a specific windowing environment or programming language, the Lean Cuisine graphical specification file of a user interface can be used in other windowing environment--as long as the supporting tools, Elc, Slc and Glc, exist in that windowing environment.

With the supporting tools, Elc, Slc and Glc, available in different windowing environment, it will be possible to design, modify and simulate a user interface in a windowing environment, save it in a Lean Cuisine graphical specification file, then in other windowing environment, use the supporting tools to load it into the Elc system, modify or simulate this user interface in current windowing environment. Based on the Lean Cuisine graphical specification file of a user interface, the interface source code for different windowing environments and programming languages will be generated by using the available code generators.

4.10. The Development Environment of Elc

The Elc graphical editor has been implemented as a XView™ application in the OpenWindows™ application environment that operates on Sun™ workstations. XView™ is based upon Xlib and is a user interface toolkit to support interactive, graphics-based applications running under the X Window System™, an important feature of the XView™ toolkit is that it implements the OPEN LOOK™ Graphical User Interface. Figure 4.15 shows the development environment of Elc.

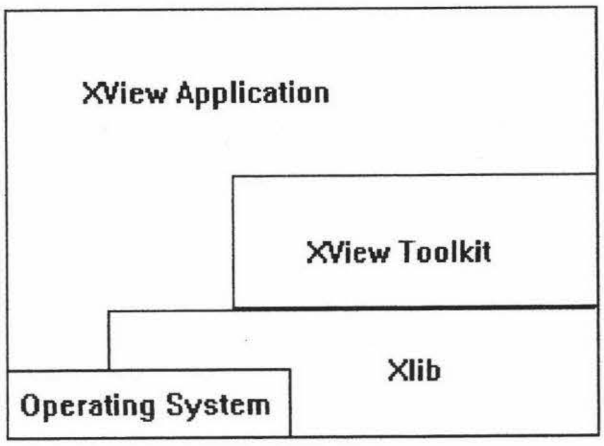


Figure 4.15. The development environment of Elc.

Chapter 5

The User Interface Simulator

5.1. Interface Simulation and Test

User interface development is an iterative process, after creating the specification of a user interface or parts of an interface, the designer may want to see how its elements feel when the end user uses them. One way to do that is to generate the source code for the interface, compile and link the code, then run executive file. Perhaps this testing method is a very effective and reliable interface testing method because the interface simulation is based on its source code, but this type interface testing is not suitable for early development of an interface because it involves several steps and is not convenient. The specification of a user interface needs to be modified time after time and the user interface should be simulated or tested in a quicker and more effective way.

5.2. Some Existing Interface Testing Tools

Some interface testing tools have been developed to support quick and convenient user interface simulation in the user interface design environment or in user interface management systems. The following is the survey of these interface testing tools.

Devguide [Sun, 90b] uses the interface setting to determine the way user interface elements on the workspace behave. When the first setting, Build, is set, the user interface can be built and selected user interface element can be edited. The second setting, Test, helps the interface designer tests the feel of the user interface which has been designed. When Test is set, selected button highlights, menus display, and so forth, user interface elements act as if they were part of a finished user interface. Of course, there are no function calls in place, so no anticipated program functions happen as user interface elements are used. It is very easy to change interface setting from Build mode to Test mode, and vice versa.

SCENARIOO [Roudaud et al., 90] provides the interface designer a test environment based on its Run_time_kernel which is connected either to the simulation data base or to the actual application. The simulation database is used during prototyping and early debugging, and the connection to the actual application is made for final debugging and tests.

In the Borland Resource Workshop [Borland, 93], menus can be created as a resource script in the Workshop menu editor. The Workshop menu editor provides three different views of the menus which are being edited: the Outline pane shows the menu's pseudo-code (resource script), the Attribute pane is used to customize the currently highlighted line in the menu resource script, and the Test Menu pane is used to test menu system. Top level menu items and top level pop-up menus are displayed in the menu bar for the window. The Menu editor automatically updates the test menu system when its attributes are modified or its resource script is edited by the designer. The Workshop menu editor provides an integrated menu design, modify and test environment.

UofA* UIMS [Singh & Green, 91] provides the simulation subsystem in the vu environment which is used to refine the graphical presentation components generated by Chisel, the simulation subsystem uses the work window and enables the designer to rehearse the presentation components being created, the design created in workshop subsystem is passed to the simulation subsystem which creates windows and activates menus and other interaction techniques defined in the presentation components. The menus and other interaction techniques can be tested by designer just as the end user interacts with them. The designer can go back to the workshop subsystem from the simulation subsystem to modify the design and then reenter the simulation subsystem to test the new design.

U-face [Kanba & Hashimoto, 91] is a design system mainly for using with menu-driven application software on generic terminal screens. The different views, design views, simulation view and verification views, are provided in U-face user interface design system. The design views consist of a screen layout view and an operation view, the simulation view which is automatically produced from a designed interface is used to test each operation step by step such as runtime. The verification views are also produced from a designed interface and include displays of a screen transition network, a mode sequence diagram and a key-binding graph. With the simulation view, the designers can check their design work and demonstrate the user interface, and the end users can test the user interface and check its suitability to their requirements.

5.3. Overview of Slc

Slc is a software tool which is used to simulate a user interface from its graphical specification in the menu interface development environment based on the Lean Cuisine visual notation. It can be used by the interface designer to simulate the menu interface which s/he has designed at different design stages. A menu interface can be simulated in its design environment or in its application environment. When a menu interface is simulated in its design environment, it can be partially or wholly simulated from its Lean Cuisine description created in the Elc editing window; any invalid link relationship between a parent tree and a subtree can be dynamically modified by the Slc system during the simulation. When a menu interface is simulated in its application environment, it is wholly simulated based on its Lean Cuisine graphical specification file, and it can give the evaluators a real feel of how the menu-based user interface works.

5.4. Interface Simulation in Its Design Environment

As discussed in section 4.8, Slc can be embedded in the Elc system to provide a user interface simulation environment in which a user interface can be directly simulated from its Lean Cuisine visual specification created in a design window of Elc. The details of interface simulation in the Elc design environment will be discussed in the following sections.

5.4.1. Change the Mode of Elc

During the design, the designer can easily change the mode of the Elc system from the design mode to the simulation mode when the precondition for interface simulation has been satisfied, or change the mode of the Elc system from the simulation mode to the design mode without precondition. The precondition for interface simulation is that a Lean Cuisine tree which represents a menu-bar must be selected (so it is the currently selected Lean Cuisine tree) before selecting the *SIMULATION* button which will change the Elc system from the design mode to the simulation mode. If the precondition for interface simulation has not been satisfied when the *SIMULATION* button is selected, a alert box which gives prompt message will be displayed. This alert box is dismissed when the designer selects the *OK* button from the alert box, in this case, the mode of the Elc system is still in design mode. If the precondition for interface simulation has been satisfied when the *SIMULATION* button is selected, the mode of the Elc system will be changed from the design mode to the simulation mode, and a window which simulates the part or the whole of the menu interface designed in the Elc window appears. Figure 5.1 shows the design mode and the simulation mode of the Elc system.

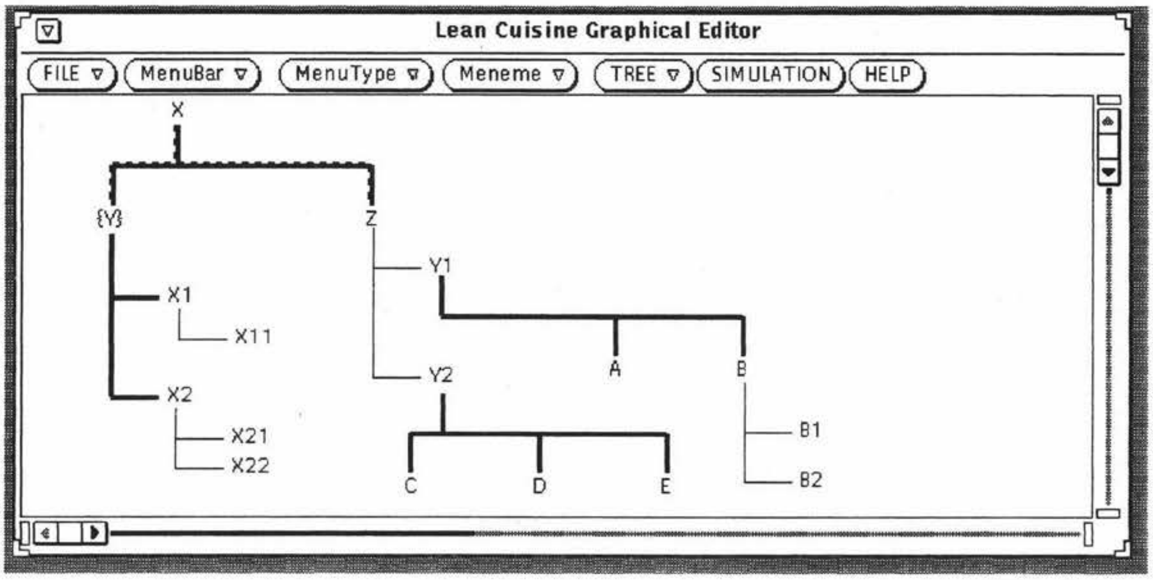


Figure 5.1a. A design window of Elc in design mode.

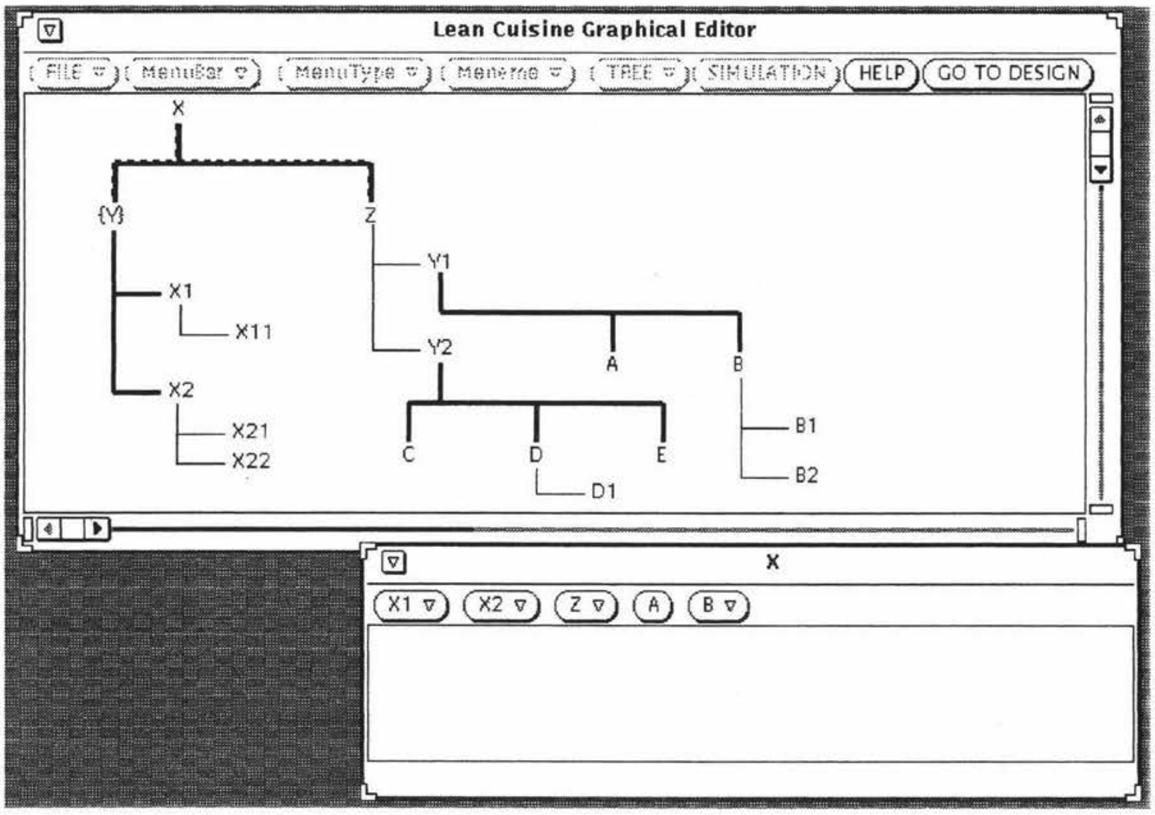


Figure 5.1b. A design window of Elc in simulation mode.

In the simulation mode, the Elc design window which displays the Lean Cuisine graphical specification of a menu interface provides a graphical reference for menu interface simulation, but its menu buttons and command buttons are grayed (disabled) except the command button *HELP* and the command button *GO TO DESIGN*, so the Lean Cuisine graphical specification displayed in the Elc design window cannot be directly modified by the designer during simulation. If the design should be modified according to the simulation result, the menu buttons and command buttons of the Elc system can be enabled by selecting the *GO TO DESIGN* button; the selection of the *GO TO DESIGN* button will unconditionally terminate the simulation procedure, close the simulation window and reset the design mode of the Elc system.

The command button *SIMULATION* is often used to change the mode of Elc from the design mode to the simulation mode and the command button *GO TO DESIGN* is used to change the mode of Elc from the simulation mode to the design mode. To avoid confusing, the Elc system has been developed to provide the enabled command button *SIMULATION* in the design mode and the grayed (disabled) command button *SIMULATION* in the simulation mode, so the command button *SIMULATION* can only be selected when the Elc system is in the design mode; the command button *GO TO DESIGN* is dynamically created by the Elc system when the Elc system changes its mode from the design mode to the simulation mode and destroyed by the Elc system when the command button *GO TO DESIGN* has been selected and the mode of Elc will be changed from the simulation mode to the design mode, so the command button *GO TO DESIGN* can only be selected when the Elc system is in the simulation mode. By using these command buttons, the mode of the Elc system can be easily changed and a menu interface can be easily created in the design-simulation-modification-simulation way in the Elc graphical editor.

5.4.2. Partial or Whole Simulation of a Menu System

When a menu interface is simulated in its design environment, according to the designer's decision, it can be partially or wholly simulated from its Lean Cuisine description created in the Elc editing window. Before selecting the *SIMULATION* button to use the Slc tool, a Lean Cuisine tree which describes a menu-bar must be selected from the design window of Elc. If it is the root of the Lean Cuisine trees, the whole menu system will be simulated in the simulation window (whole simulation). If it is a subtree or an isolated tree, only this tree and its subtrees can be simulated in the simulation window (partial simulation). Partial simulation of a menu system is very useful during interface design; it gives the designer a chance to design many different specification for a same menu system in a same design window of Elc, and select the better one by simulating them one by one. The examples of partial simulation and whole simulation based on the current selection are shown in Figure 5.2 and Figure 5.3.

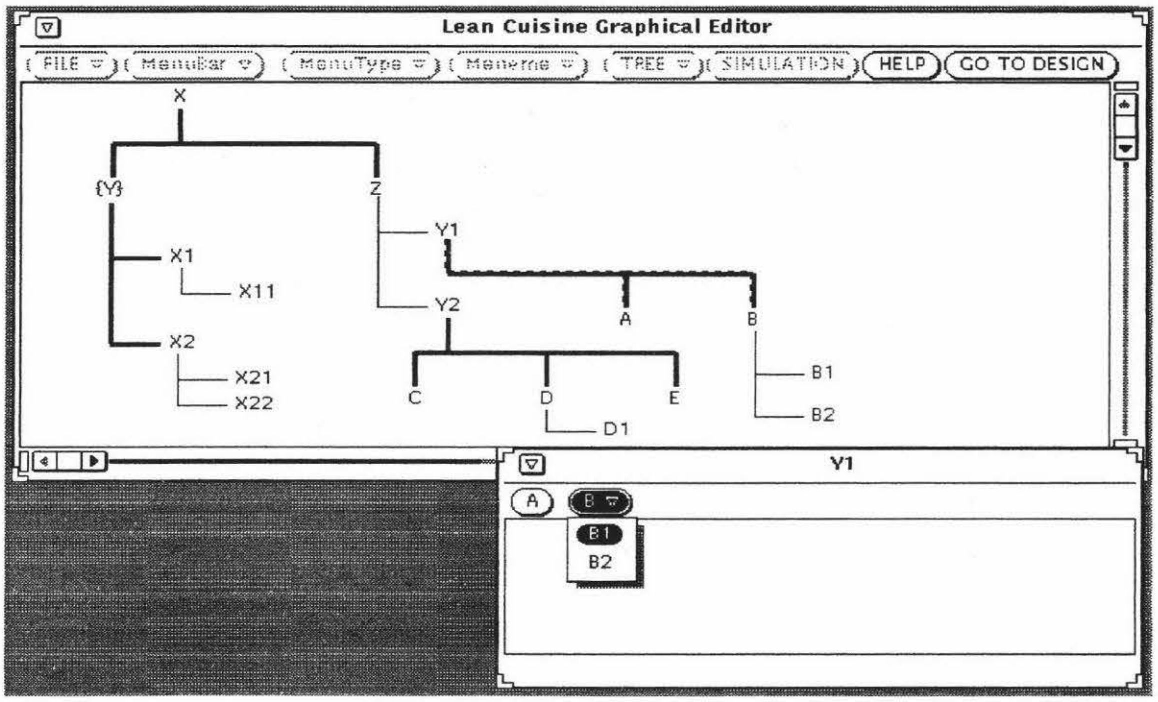


Figure 5.2. An example of partial simulation based on the current selection.

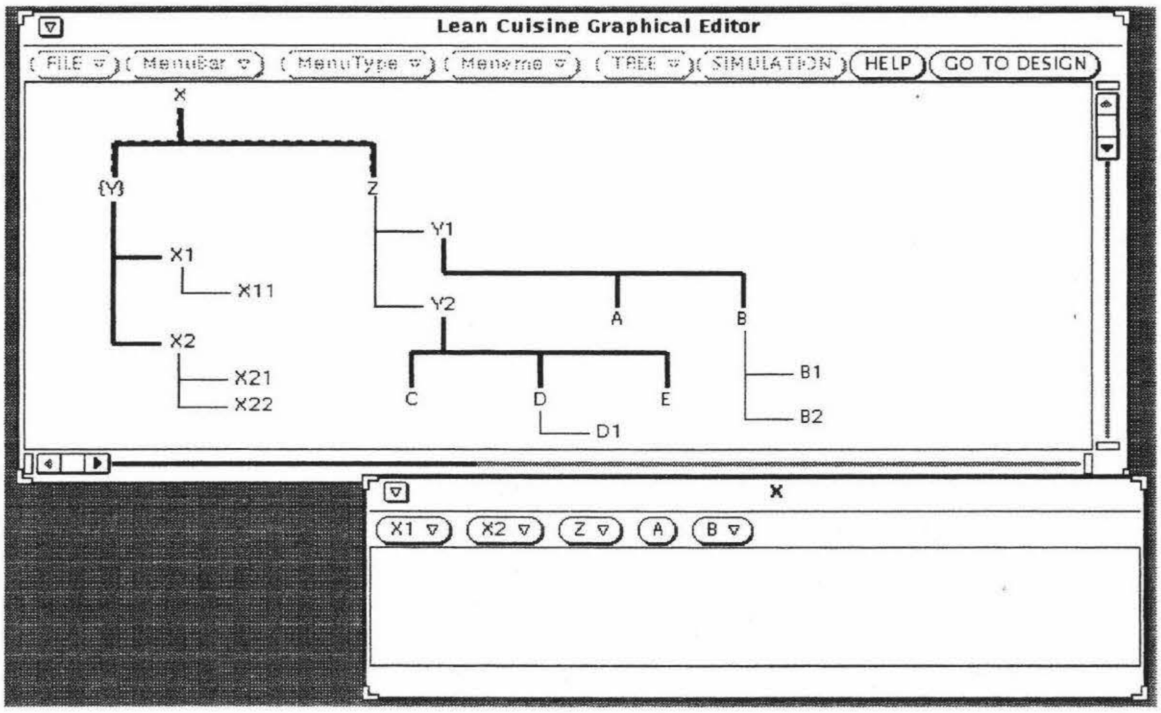


Figure 5.3. An example of whole simulation based on the current selection.

5.4.3. Modifying Invalid Menu Structures During Simulation.

An invalid link relationship between a parent tree and a subtree can be dynamically modified by the Slc system during interface simulation. Several rules are used by the Slc system to check up whether the link relationship between every subtree and its parent tree is valid before simulating a menu-based user interface. For example, suppose that a subtree describes menu-bar items, and if its parent tree is a menu-bar tree, the link relationship between them is valid when a parent tree's meneme which is directly linked to this subtree is a virtual meneme. However if its parent tree is a menu tree, the link relationship between them is valid when a parent tree's meneme which is directly linked to this subtree is a real meneme. In case a subtree is a menu tree, if its parent tree is a menu-bar tree, the link relationship between them is valid when a parent tree's meneme which is directly linked to this subtree is a real meneme; if its parent tree is a menu tree, the link relationship between them is always valid. Some examples of the invalid link between a parent tree and a subtree are shown in Figure 5.4.

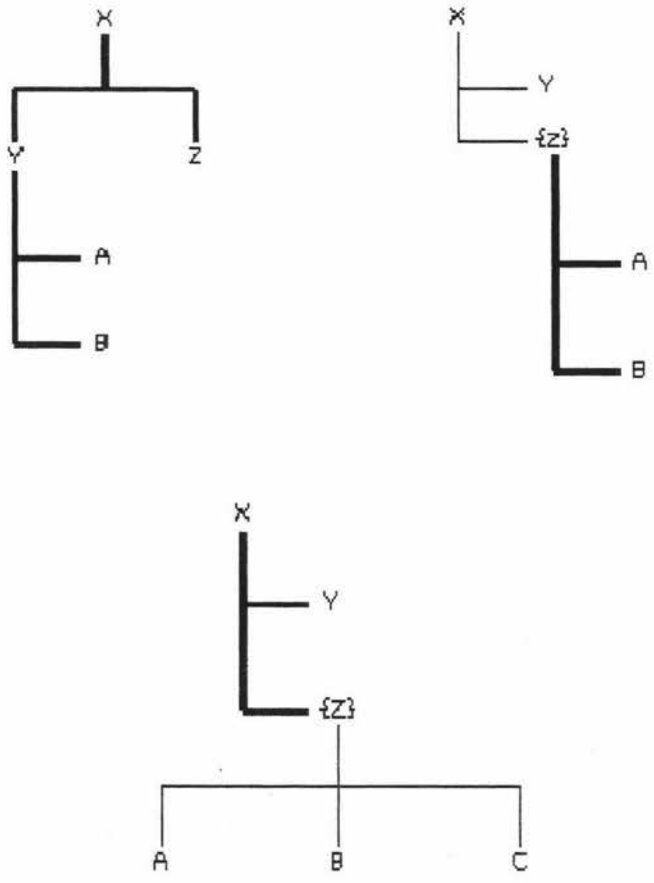
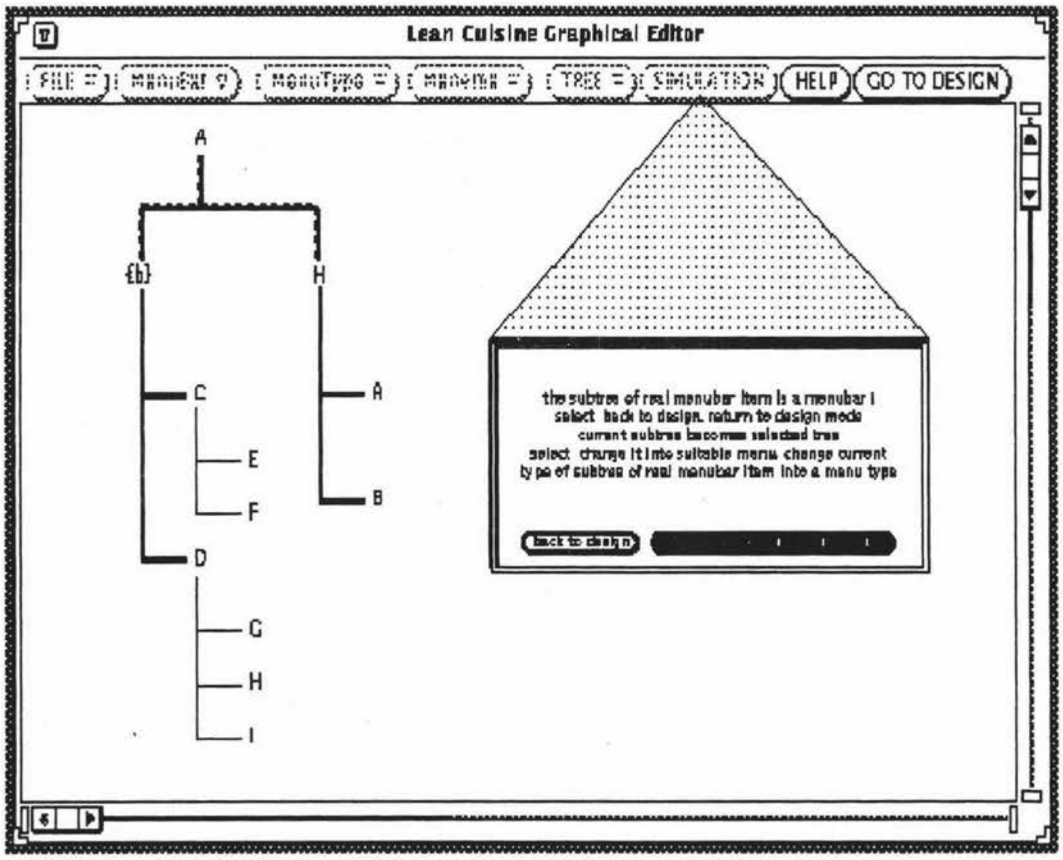


Figure 5.4. Some examples of the invalid link between a parent tree and a subtree.

If an invalid tree link is found by the Slc system, a alert box which includes prompt message and command buttons will be displayed. This alert box is dismissed when the designer selects the *GO TO DESIGN* button from the alert box, and wants to modify the invalid tree link in the Elc design mode. The mode change is same as the change after the *GO TO DESIGN* button is selected from the menu-bar: The simulation procedure will be unconditionally terminated, the simulation window will be closed and the design mode of the Elc system will be reset. Figure 5.5 shows an invalid tree link is found by the Slc system during simulation.



the subtree of real menubar item is a menubar !
select back to design. return to design mode
current subtree becomes selected tree
select change it into suitable menu. change current
type of subtree of real menubar item into a menu type

Figure 5.5. An invalid tree link is found by the Slc system.

The type of a Lean Cuisine tree (or the type of a meneme) which causes invalid link between a parent tree and a subtree can be dynamically modified in the Elc simulation mode. If the command button *MODIFY* is selected from above alert box, the type of the Lean Cuisine tree (or the type of the meneme) will be changed into a suitable type based on the rules created in the Slc system so the invalid link becomes valid and Slc can simulate the presentation components of the menu-based user interface from these Lean Cuisine tree description. In this case, the mode of the Elc system is still in the simulation mode but the modified result will be displayed in the Elc design window. Figure 5.6 shows the result after dynamic modification.

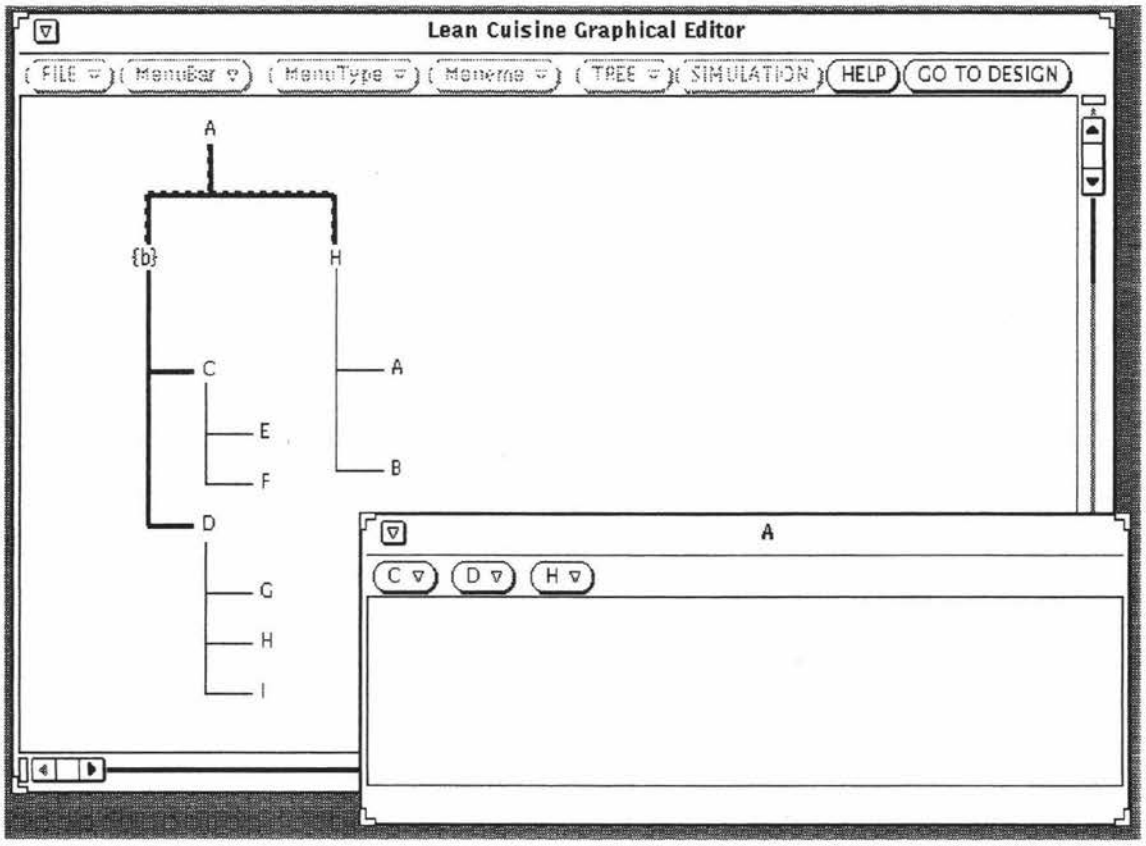
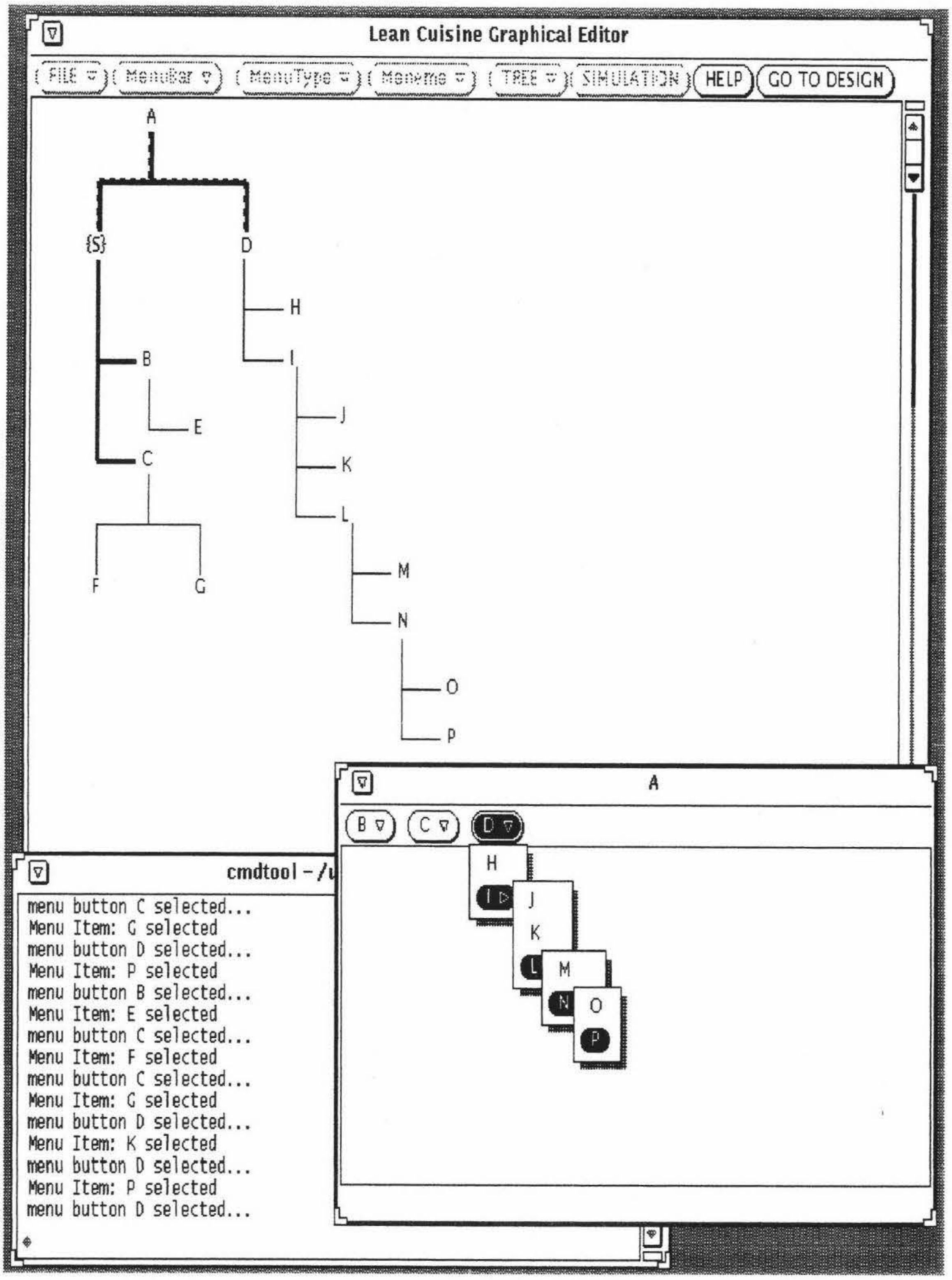


Figure 5.6. The valid LC description and its simulation window after dynamic modification.

5.4.4. Testing the Interface

In the simulation window, the designer can test the menus and get a feel for the way the menu interface works in use. When the designer selects a menu button, a menu item, or a command button from the simulation window, the selected item is highlighted and the feedback message is displayed in the default Console window if Elc was started from the File Manager or in the command window (the Command Tool window or the Console window) if Elc was started from the command line of this window. An example of the test of an interface is shown in Figure 5.7.

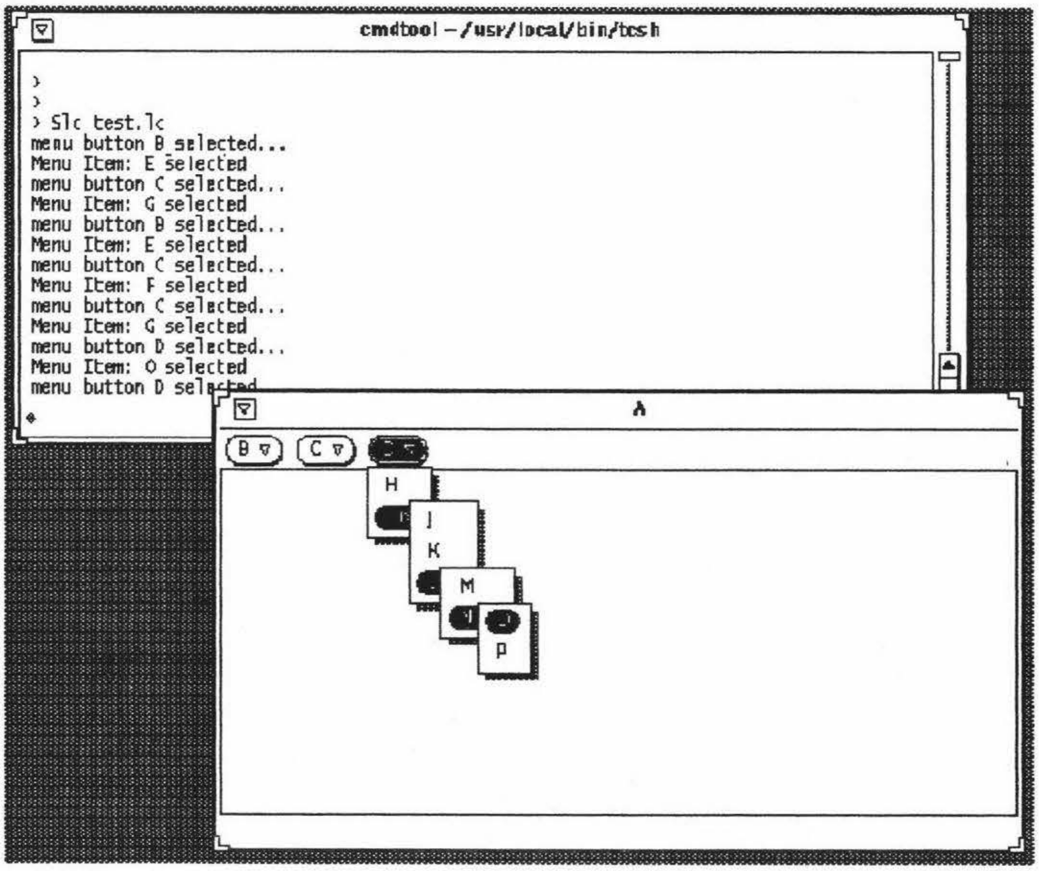


```
menu button C selected...
Menu Item: G selected
menu button D selected...
Menu Item: P selected
menu button B selected...
Menu Item: E selected
menu button C selected...
Menu Item: F selected
menu button C selected...
Menu Item: G selected
menu button D selected...
Menu Item: K selected
menu button D selected...
Menu Item: P selected
menu button D selected...
```

Figure 5.7. Testing the menu system of an interface in the simulation window. **Front**: the simulation window of the interface; **back**: the Elc design window which is in the simulation mode contains the Lean Cuisine visual specification of the interface; **middle**: the command window which is used to display the feedback message when testing the interface.

5.5. Interface Simulation in Its Application Environment

Slc can also be used out of the Elc system, in this case, Slc is an independent simulation tool. Based on the Lean Cuisine graphical specification file which was saved from the Elc design window, the Slc system uses the lc_run_time kernel to generate a user interface prototype and dynamically create and delete menu structures according to its Lean Cuisine specification and user's selection. Before creating presentation components of a user interface, the Slc system searches the root of the Lean Cuisine trees and checks up whether it describes a menu-bar. If the search is successful, the Slc tool simulates the menu-based user interface based on the root of the Lean Cuisine tree and its subtrees, or the error message will be displayed in the command window. In the simulation window, the menu buttons, menu items, or command buttons can be tested by the interface evaluators (end users, application program developers or interface designers), the selected item is highlighted and the feedback message is displayed in the command window from which Slc was started. Figure 5.8 shows an example of direct simulation of an interface in its application environment.



```
>
>
> Slc test.lc
menu button B selected...
Menu Item: E selected
menu button C selected...
Menu Item: G selected
menu button B selected...
Menu Item: E selected
menu button C selected...
Menu Item: F selected
menu button C selected...
Menu Item: G selected
menu button D selected...
Menu Item: O selected
menu button D selected
```

Figure 5.8. The direct simulation of an interface in its application environment.

When Slc is independently used to simulate a menu-based user interface from its Lean Cuisine graphical specification file, it can give evaluators a real feel of how this menu-based user interface works. Besides the simulation and test of the menu system, keyboard inputs and mouse actions can be handled by the Slc tool. A keyboard event takes place when a key is pressed or released. Slc will display the message (the ASCII and decimal code of a key which is pressed or released and the current mouse position) on the simulation window when the keyboard events take place. Mouse events take place when a button (left, middle, right) of the mouse is pressed or released, the mouse pointer moves or the mouse pointer moves while SELECT (left button), ADJUST (middle button), or MENU (right button) is held down. Slc will display the message (the event type and the current mouse position) on the simulation window when the mouse events take place. Slc cannot perform its partial simulation and dynamic modification functions when it is independently used.

Generally speaking, the interface designer does not need to run the Slc system out of the Elc graphical editor because the Slc system embedded in Elc provides a complete user interface simulation and modification environment and covers all simulation functions, except keyboard events and mouse events simulation, which Slc provides when it is run independently. But when Slc is used out of the Elc system, it provides a simulated application environment where the user interface works. It is convenient for application program developers who want to understand the user interface structure before developing and linking the application to it and is convenient for end users who want to evaluate the user interface in a real application environment.

5.6. Functions of the Lc_run_time Kernel

Slc uses the lc_run_time kernel to create and manage menu system based on Lean Cuisine graphical specification by direct mapping process. At the beginning, the basic menu system is created by lc_run_time kernel according to the LC specification of the default mode of a menu system, and names of LC trees which correspond to current menu-bar items are stored in the list of active LC trees. Then, when the mode of the menu system is changed based on a menu item selection, the lc_run_time kernel will modify the list of active LC trees, delete menu-bar items from the menu-bar and add new menu-bar items to menu-bar according to the LC specification of the current mode of a menu system. When menu-bar items are dynamically deleted from the menu-bar according to a mode change, their menus and submenus will be automatically deleted, and when new menu-bar items are dynamically added to menu-bar, their menus and submenus are dynamically created from their Lean Cuisine graphical specification at the same time. If a selected menu item corresponds to one of the terminal (leaf) menemes of Lean Cuisine trees, in order to respond the menu item selection, the lc_run_time kernel will display the feedback message in the message window by calling a pseudo-application procedure during interface simulation. In fact, when a selected menu item corresponds to a real non-terminal meneme, the lc_run_time kernel will also display the feedback message in the message window.

5.7. Summary

Based on the Lean Cuisine visual specification of a menu system, a very effective and reliable interface testing tool, Slc, has been developed and can be used in the Lean Cuisine menu interface development environment to support quick and convenient user interface simulation. A menu interface can be simulated in its design environment or in its application environment. From its design environment, a menu interface can be partially or wholly simulated based on its Lean Cuisine description, and any invalid link relationship between a parent tree and a subtree can be dynamically modified by the Slc system during the simulation. When a menu interface is simulated in its application environment, it is wholly simulated based on its Lean Cuisine graphical specification file, and it can give evaluators a real feel of how the menu-based user interface works.

Chapter 6

The User Interface Generator

6.1. Overview of Glc

In the integrated interface development environment based on the Lean Cuisine graphical notation, Glc is the user interface generator, a software tool used to generate basic interface source code files for a user interface from its Lean Cuisine graphical specification file. By using Glc, a user interface can be automatically generated without the need for any programming.

6.2. Using Glc to Generate Interface Source Code Files

Suppose that a user interface called Test has been created in an Elc editing window by using the Lean Cuisine graphical notation, and based on its Lean Cuisine graphical specification, the Test user interface has been simulated in both the design environment and the application environment, and the Lean Cuisine graphical specification of the Test user interface has been saved from the Elc design window as a Lean Cuisine graphical specification file, Test.lc. To generate the Test user interface source code files, in a command window, the Glc command is followed by the name of the Lean Cuisine graphical specification file of Test user interface, Test.lc.

```
>Glc  Test.lc
```

Based on the Lean Cuisine graphical specification file Test.lc, the Glc tool generates the following four interface source code files and a Makefile which can be used with the make command to compile these interface source code files.

Test_main.c
Test_link_app.c
Test_app.c
Test_app.h
Makefile

During the generation of interface source code, the Glc command names four generated source code files after the original Lean Cuisine graphical specification file name, Test.lc, stripping off the .lc extension and adding new extensions to identify each file.

Now an executable file of the Test user interface can be generated based on above four source code files and Makefile. To do this, the interface designer simply enters the command "make" in the command window, the compiler goes to work and uses the Makefile parameters for the compile. When it finishes it leaves an executable file named "Test" according to the default file name setting in above Makefile. The real Test user interface can now be run by entering "Test" in the command window and its menu system can be tested. Selected menu items will be highlighted, the feedback messages will be displayed in the command window, and keyboard events and mouse events will be also handled.

6.3. Files Generated by Glc

As shown in above example, based on a Lean Cuisine graphical specification file, four interface source code files and a Makefile are generated by Glc. The four source code filename extensions, _main.c, _link_app.c, _app.c, and _app.h, are used in the integrated interface development environment based on Lean Cuisine graphical notation to identify different interface source code file.

The _main.c file

The _main.c file includes a main() procedure which is created by Glc and contains basic user interface elements (window, scroll-bar etc), calls used to load the Lean Cuisine graphical specification file and run the lc_run_time kernel which is used to create and manage the menu system, and calls used to initialize XView and start the main loop.

The _link_app.c file

The _link_app.c is the link socket between a user interface and its application programs. It includes the event handling procedures used to receive events and call application procedures when the events occur. The original application procedures generated by the Glc tool are included in the _app.c file.

The _app.c file

The _app.c file includes the original application procedures generated by the Glc tool and each of these procedures is designed to respond an event and display the feedback message in a command window when the event occurs. Application programmers will develop real application procedures based on these original application procedures.

The _app.h file

The _app.h file includes declarations of all application procedures generated by the Glc tool.

The Makefile

The Makefile is generated by Glc to control the source code compile. It is used with the make command to compile and link the source code files. An executable file of a user interface can be easily generated according to the instructions in the Makefile and named according to the default file name setting in the Makefile.

6.4. The Development of Real Application Procedures

The real application procedures will be developed on the basis of the original application procedures generated by Glc. During the course of the development of the real application, the new executable file of the user interface can always be generated by using "make" command after linking some of the real application procedures to the user interface. The new linked application procedures can be tested and modified based on the working model of the user interface.

Besides the development of the real application routines which respond to menu selections, other very important application routines, which are invoked by keyboard events, mouse motion events and mouse button events, will be also developed according to the requirements of the real application.

6.5. Summary

By using Glc, the basic interface source code files of a user interface can be automatically generated from its Lean Cuisine graphical specification file. A working model of a user interface can be easily and quickly created without programming.

The combination of the three software tools, Elc, Slc, and Glc, forms an integrated interface development environment which makes user interface development much easier and quicker from design and simulation to implementation. The integrated interface development environment based on Lean Cuisine graphical notation is shown Figure 6.1.

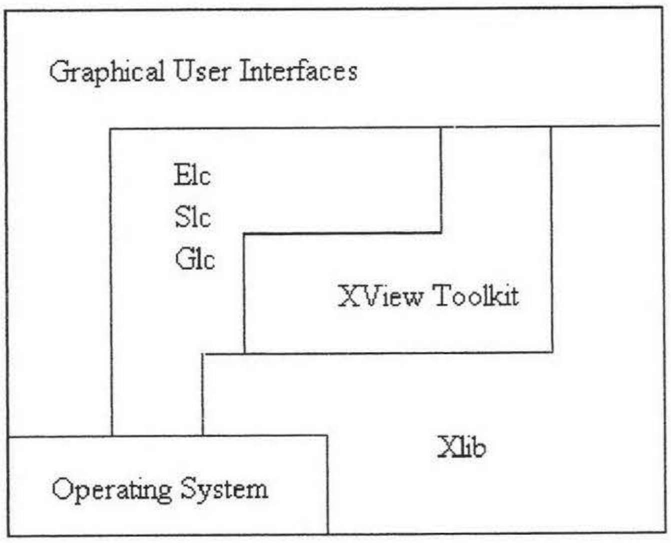


Figure 6.1. The integrated interface development environment based on Lean Cuisine.

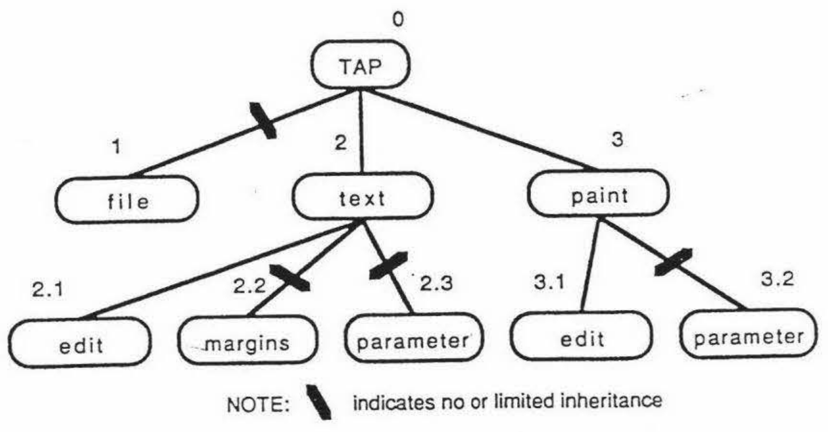
Chapter 7

Detailed Examples of Application Development

7.1. Development of TaP Interactive Application

7.1.1. About TaP

TaP is a simple interactive application which provides a basic text-processing capability together with a painting facility. It has been used to demonstrate a methodology for menu-based user interface design using the Lean Cuisine notation [Apperley, 88].



Mode 0 (root):	(a)	new, open, close, save, print
	(b)	copy, cut, paste, clear
	(c)	text, paint
Mode 1 (file):	(d)	file operation sub-dialogues
Mode 2 (text):	(e)	on-screen editing
	(f)	adjust margins
	(g)	select style and size
Mode 2.1:	(h)	enter text, select text
Mode 2.2:	(i)	margin adjust sub-dialogues
Mode 2.3:	(j)	style and size sub-dialogues
Mode 3 (paint):	(k)	on-screen editing
	(l)	select tool and/or properties
Mode 3.1:	(m)	use current tool
Mode 3.2:	(n)	tool and property sub-dialogues

Figure 7.1.1. The fully refined mode-tree of TaP and a summary of the facilities available in each mode [Apperley, 88].

The general specification of TaP can be found from [Apperley, 88]. Figure 7.1.1 shows the fully refined mode-tree of TaP and a summary of the facilities available in each mode, and Figure 7.1.2 shows the complete Lean Cuisine description for TaP [Apperley, 88].

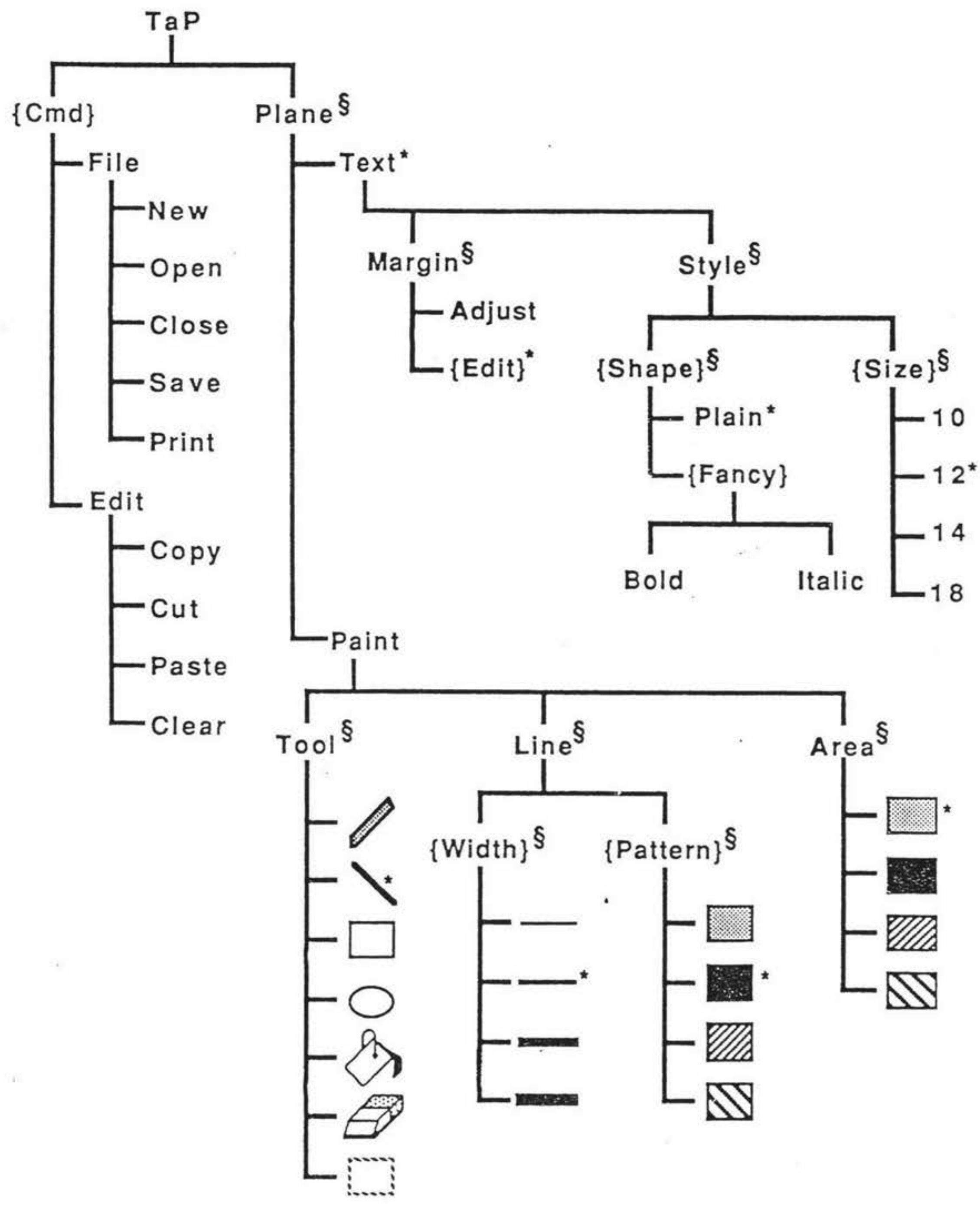


Figure 7.1.2. The complete Lean Cuisine description for TaP [Apperley, 88].

7.1.2. Using Elc to Create the LC Graphical Specification of TaP

The Lean Cuisine graphical description shown in Figure 7.1.2 can be interactively created in an Elc editing window. Each Lean Cuisine tree will be created according to its type and meneme number. Then the meneme names and meneme modifiers will be added to the menemes. In order to avoid Lean Cuisine trees overlapping when linking all Lean Cuisine trees to form the graphical specification of the menu system of TaP, the geometric shapes of some Lean Cuisine trees will be adjusted. At the different design stages, the design result can be simulated based on the Lean Cuisine graphical description. The following paragraphs give the details of the design and the simulation of the TaP user interface at the different stages.

The Design of the Basic Menu-bar

According to the design steps described in [Apperley, 88], a menu-bar which contains the basic items *File*, *Edit* and *Plane* will be designed first, Figure 7.1.3 shows the LC description of the basic menu-bar.

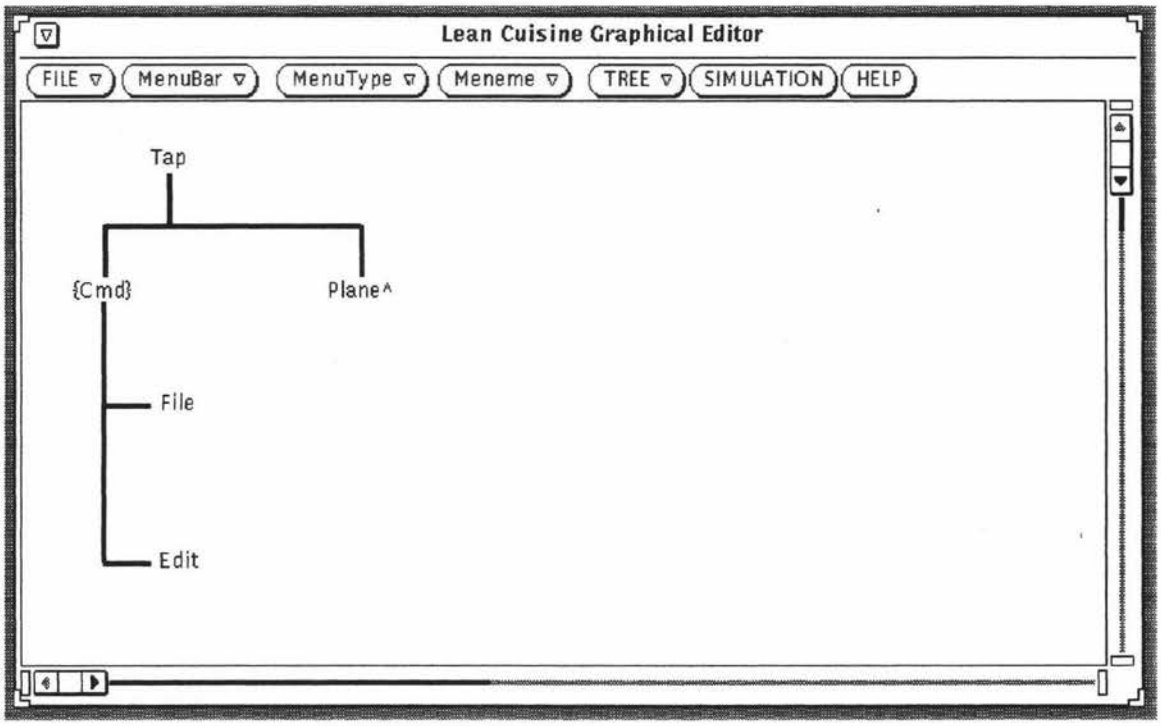
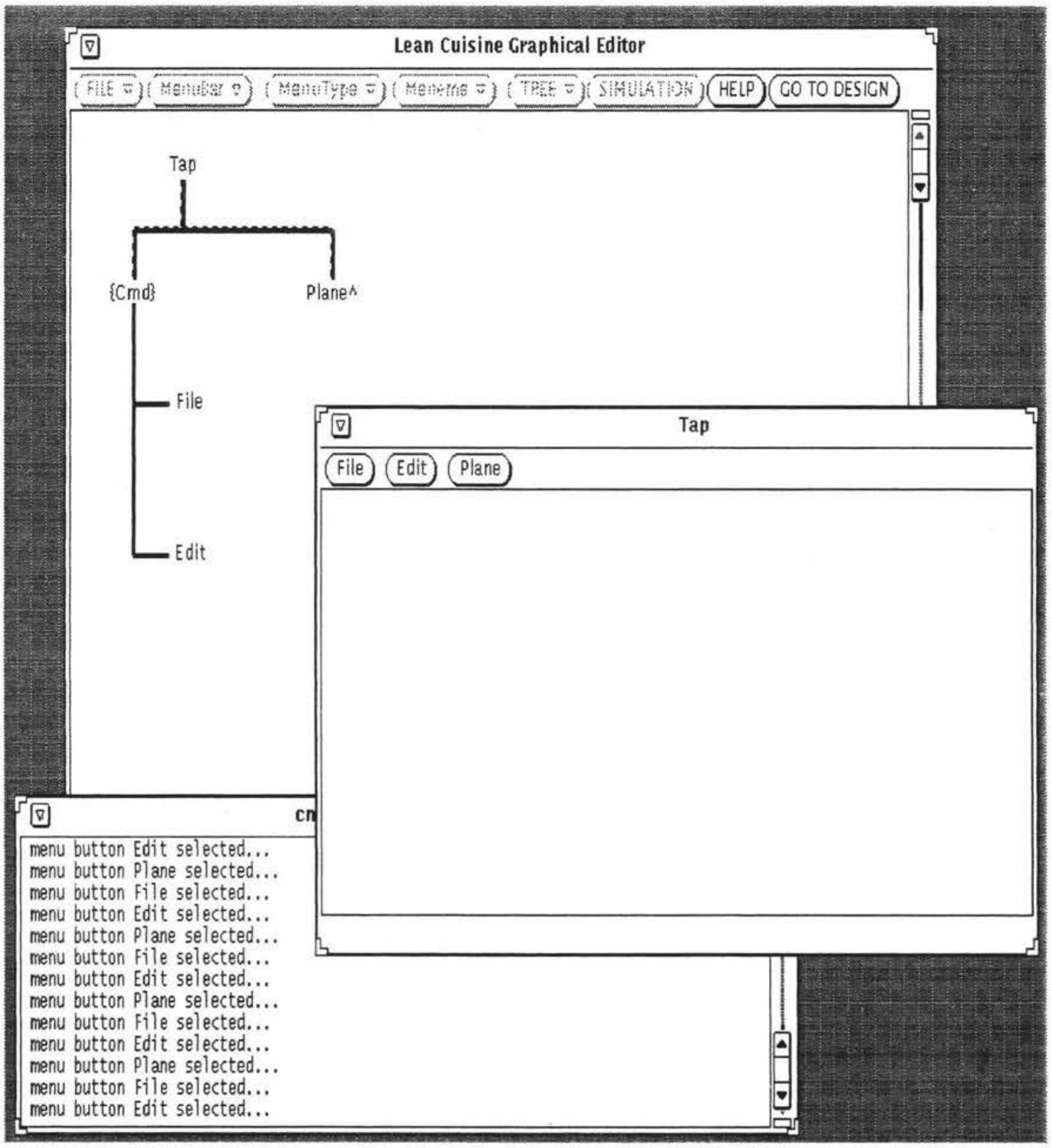


Figure 7.1.3. The LC description of the basic menu-bar.

Based on the LC description shown in Figure 7.1.3, an interface which contains the basic menu-bar can be simulated and tested in the simulation environment provided by the Elc system. The direct simulation of the basic menu-bar from its Lean Cuisine description is shown in Figure 7.1.4.



```
menu button Edit selected...
menu button Plane selected...
menu button File selected...
menu button Edit selected...
menu button Plane selected...
menu button File selected...
menu button Edit selected...
menu button Plane selected...
menu button File selected...
menu button Edit selected...
menu button Plane selected...
menu button File selected...
menu button Edit selected...
```

Figure 7.1.4. The direct simulation of the basic menu-bar from its Lean Cuisine description. **Front**: the simulation window of the TaP interface which contains the basic menu-bar; **back**: the Elc design window which is in the simulation mode contains the Lean Cuisine visual specification of the basic menu-bar of TaP; **middle**: the command window which is used to display the feedback message when testing the interface.

Each of these menu-bar items has an appropriate pull-down menu. The LC trees which represent these pull-down menus are then created. When linking these Lean Cuisine trees to menu-bar items, the geometric shapes of Lean Cuisine trees which represent the basic menu-bar have been adjusted to avoid some Lean Cuisine trees overlapping. Figure 7.1.5 shows the LC description of the basic menu-bar and its pull-down menus.

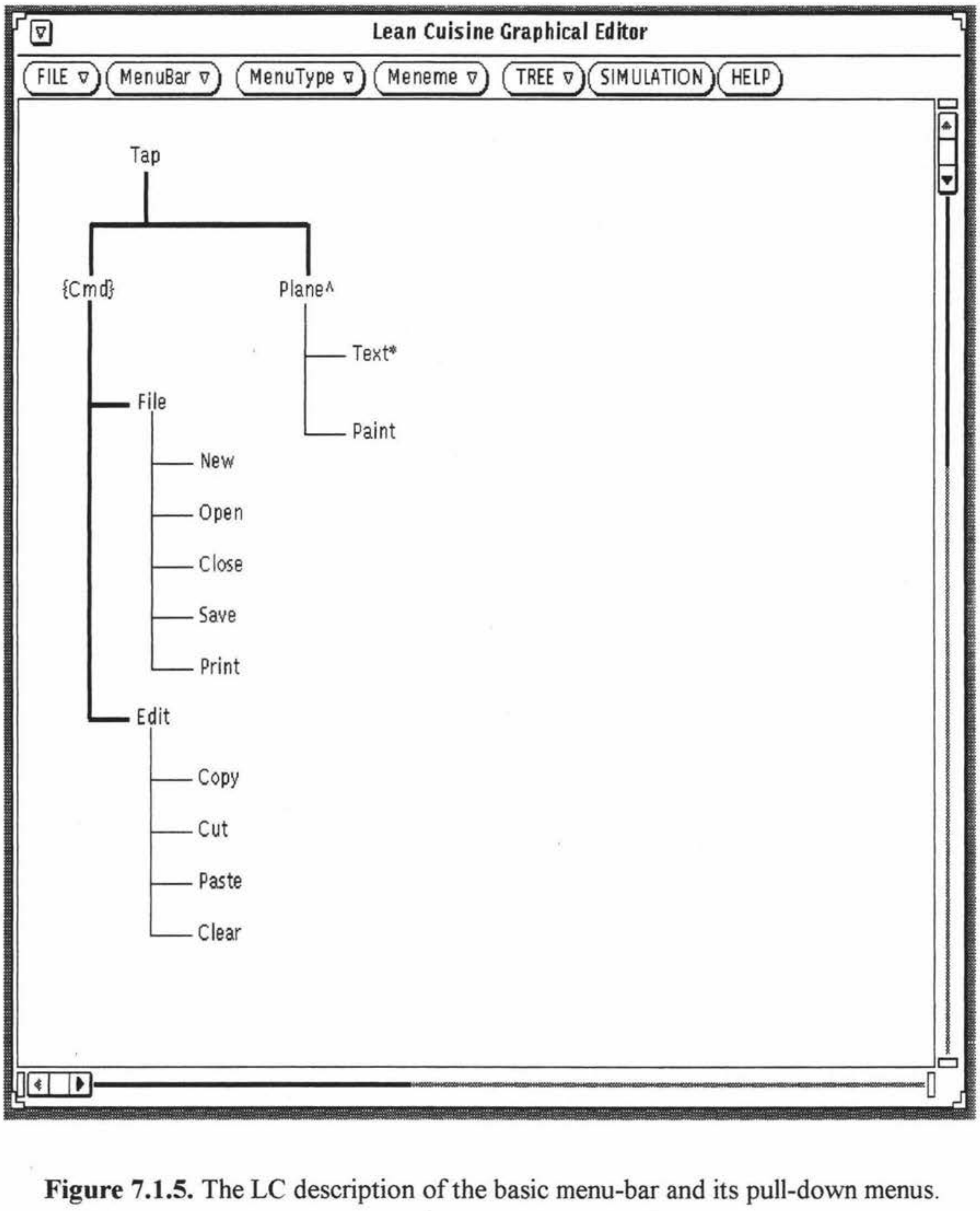
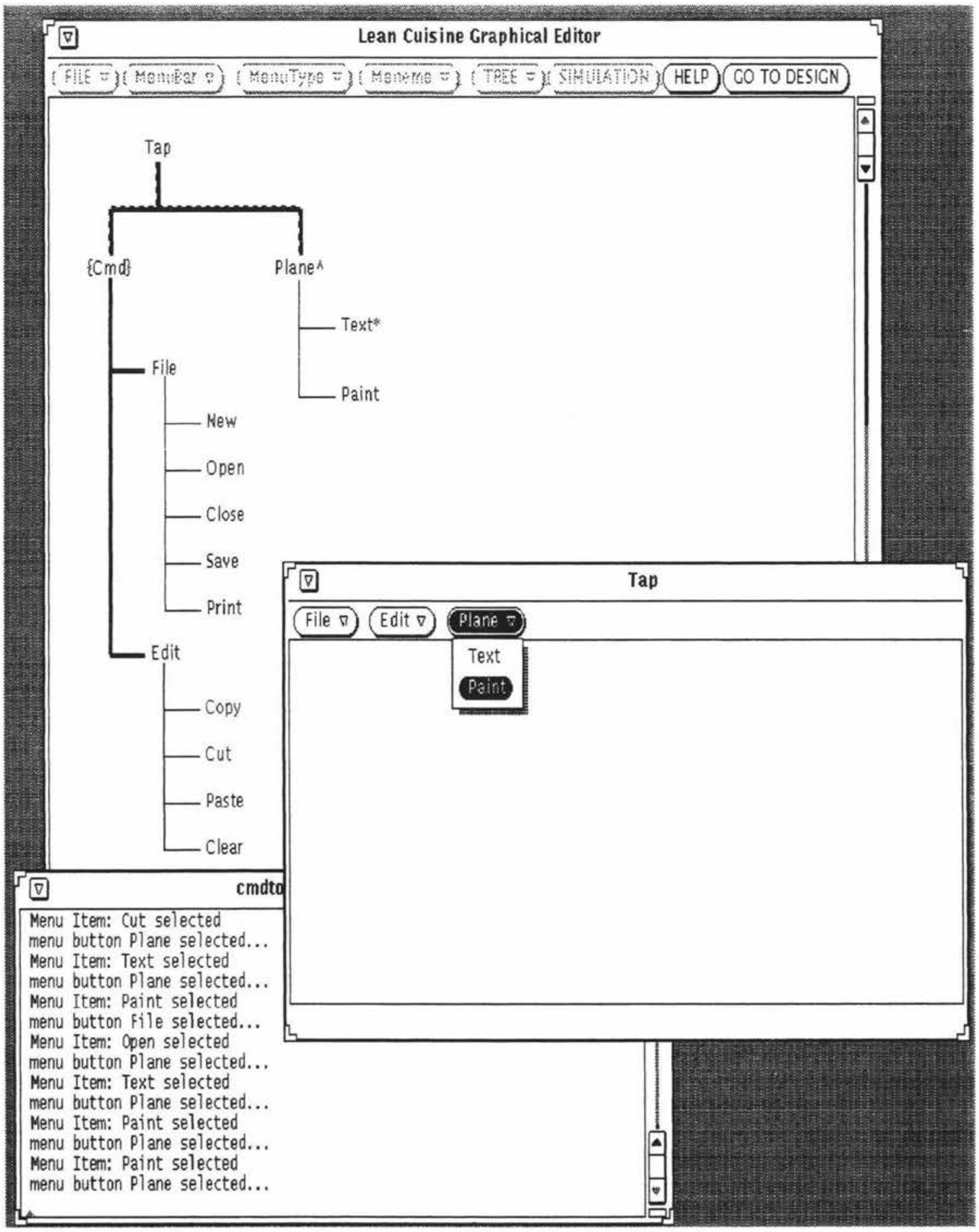


Figure 7.1.5. The LC description of the basic menu-bar and its pull-down menus.

Based on the LC description shown in Figure 7.1.5, the basic menu-bar and its pull-down menus can be simulated and tested in the Elc simulation environment, Figure 7.1.6 shows the direct simulation of the basic menu-bar and its pull-down menus from its Lean Cuisine description.



```
Menu Item: Cut selected
menu button Plane selected...
Menu Item: Text selected
menu button Plane selected...
Menu Item: Paint selected
menu button File selected...
Menu Item: Open selected
menu button Plane selected...
Menu Item: Text selected
menu button Plane selected...
Menu Item: Paint selected
menu button Plane selected...
Menu Item: Paint selected
menu button Plane selected...
```

Figure 7.1.6. The direct simulation of the basic menu-bar and its pull-down menus from its Lean Cuisine description. **Front**: the simulation window of the TaP interface which contains the basic menu-bar and pull-down menus; **back**: the Elc design window which is in the simulation mode contains the Lean Cuisine visual specification of the basic menu-bar and its pull-down menus; **middle**: the command window which is used to display the feedback message when testing the interface.

The Design of the *Text* Sub-mode Facilities

Figure 7.1.7 shows that the Lean Cuisine description of the *Text* sub-mode facilities of TaP has been created in an Elc editing window and some LC trees have been adjusted to avoid Lean Cuisine trees overlapping.

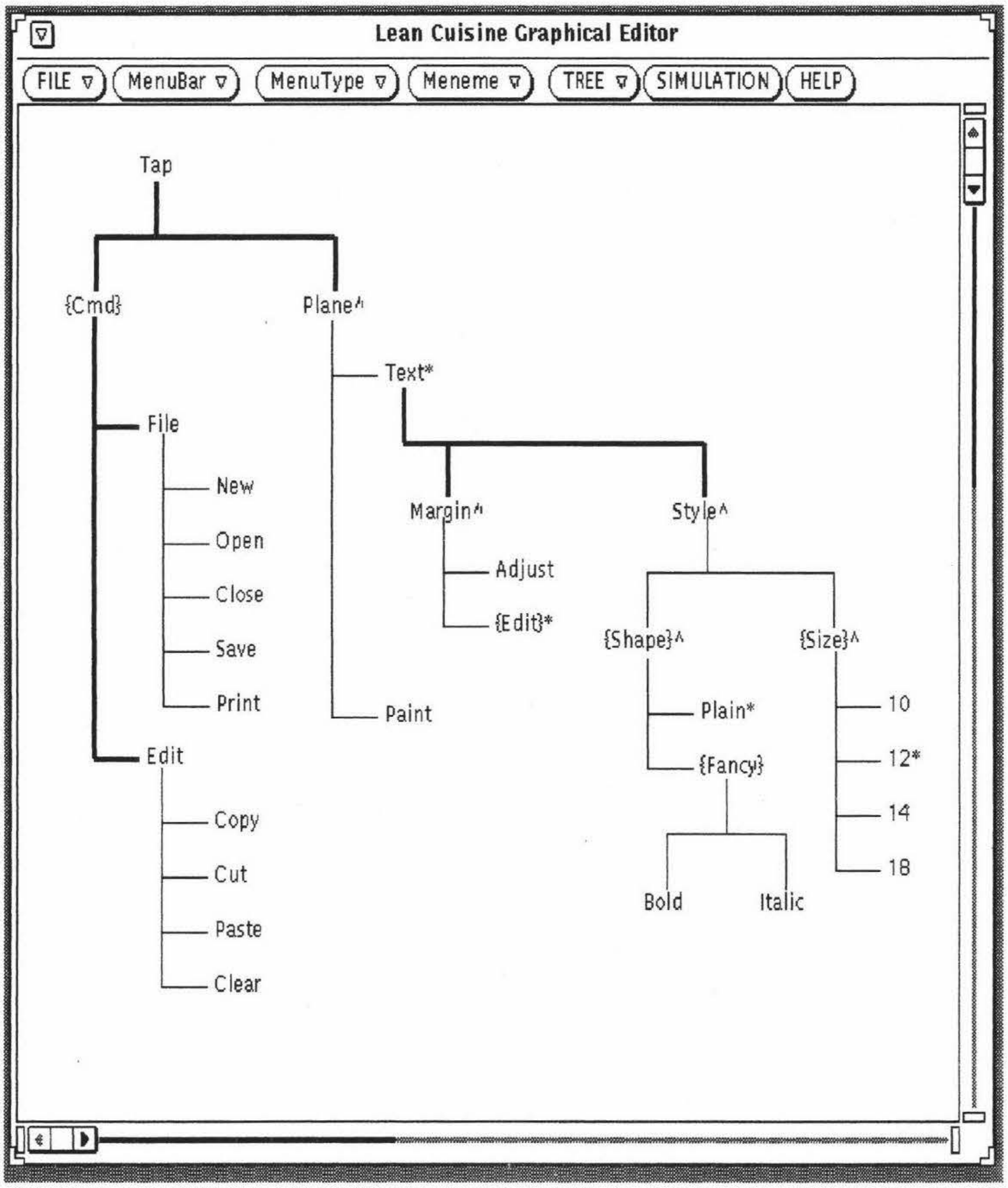
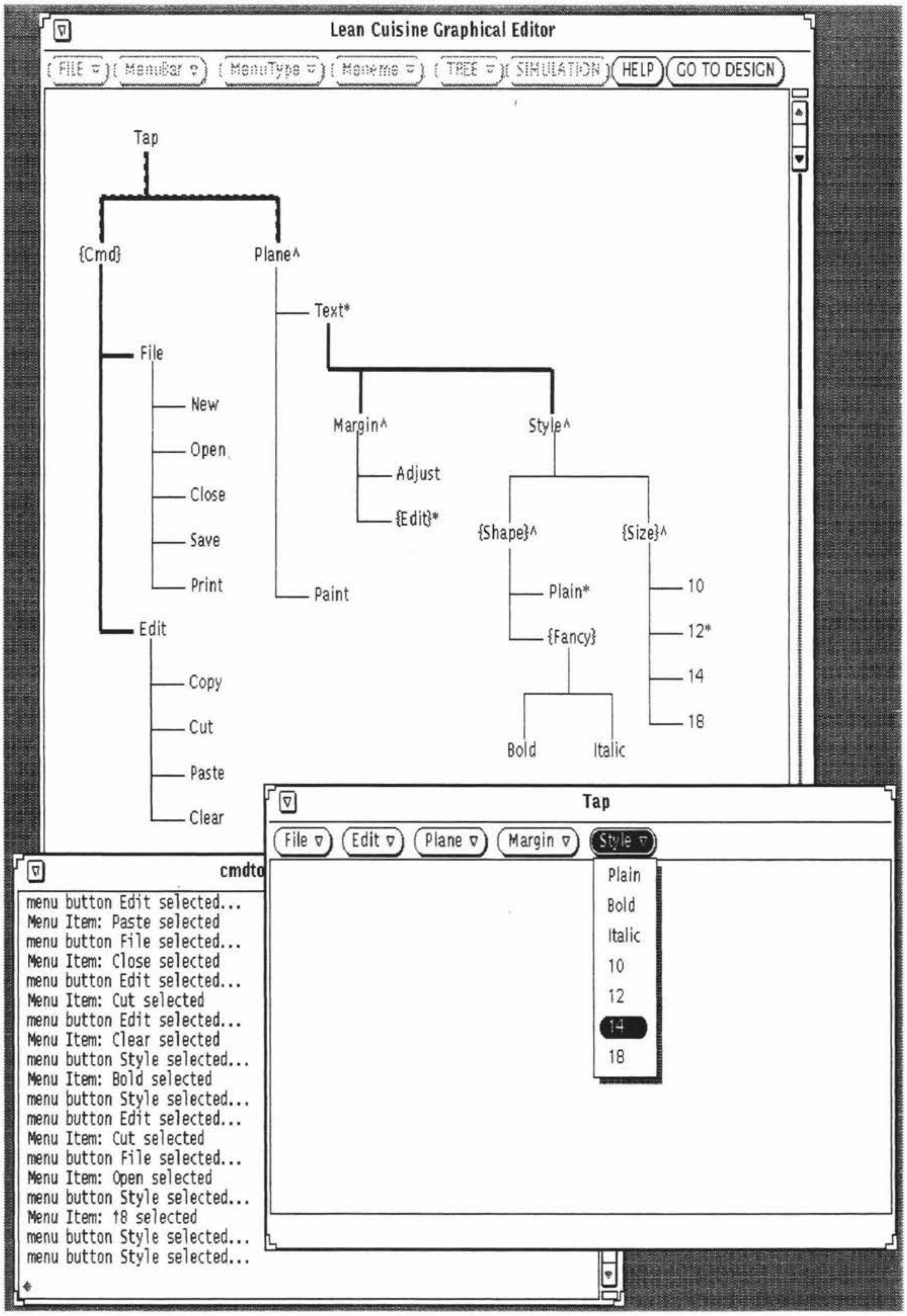


Figure 7.1.7. The Lean Cuisine description of the basic menu-bar with its pull-down menus and the *Text* sub-mode facilities of TaP created in an Elc editing window.

Based on the Lean Cuisine description shown in Figure 7.1.7, the TaP interface can be simulated and tested in the Elc simulation environment and is shown in Figure 7.1.8.



```
menu button Edit selected...
Menu Item: Paste selected
menu button File selected...
Menu Item: Close selected
menu button Edit selected...
Menu Item: Cut selected
menu button Edit selected...
Menu Item: Clear selected
menu button Style selected...
Menu Item: Bold selected
menu button Style selected...
menu button Edit selected...
Menu Item: Cut selected
menu button File selected...
Menu Item: Open selected
menu button Style selected...
Menu Item: 18 selected
menu button Style selected...
menu button Style selected...
```

Figure 7.1.8. Elc simulation environment showing the direct simulation of the *Text* mode of TaP from its Lean Cuisine description of Figure 7.1.7.

In the simulation window (the front window shown in Figure 7.1.8), two menu-bar items *Margin* and *Style* have been added to the menu-bar because the default mode of TaP is the *Text* sub-mode. The pull-down menu items can be selected and tested. The selected menu item is highlighted and the feedback message is displayed in the feedback message window (the middle window shown in Figure 7.1.8) which is a default Console window if Elc was started from the File Manager or is a command window (a Command Tool window or a Console window) if Elc was started from the command line of this window.

When testing the interface in the simulation window, the *Paint* sub-mode can be set by selecting the menu item *Paint* from the pull-down menu *Plane*. When the menu item *Paint* is selected, the menu-bar items *Margin* and *Style* will be deleted from the menu-bar and TaP is in the *Paint* sub-mode, but no new menu-bar items will be added to the menu-bar for the *Paint* sub-mode of TaP because the Lean Cuisine description of these menu-bar items of the *Paint* sub-mode has not been created in the Elc design window (the back window shown in Figure 7.1.8).

The Design of the *Paint* Sub-mode Facilities

The Lean Cuisine description of the *Paint* sub-mode facilities of TaP is then created in an Elc editing window and when linking these Lean Cuisine trees, the geometric shapes of the Lean Cuisine trees have been adjusted to avoid Lean Cuisine trees overlapping. Figure 7.1.9 shows the Lean Cuisine description of TaP after adding the *Paint* sub-mode facilities to it.

Based on the Lean Cuisine description created in the Elc editing window shown in Figure 7.1.9, the TaP interface can be simulated and tested in the simulation environment provided by the Elc system, the direct simulation of the TaP menu-based interface from its Lean Cuisine description is shown in Figure 7.1.10.

In the simulation window (the front window shown in Figure 7.1.10) of the TaP menu system, the default mode of TaP is the *Text* mode. When testing the interface in the simulation window, the *Paint* sub-mode of TaP can be set by selecting the menu item *Paint* from the pull-down menu *Plane*; three menu-bar items *Tool*, *Line* and *Area* have been added to the menu-bar after the *Paint* sub-mode has been set. The pull-down menu items can be selected and tested. The selected menu item is highlighted, and the feedback message is displayed in the feedback message window (the back window shown in Figure 7.1.10).

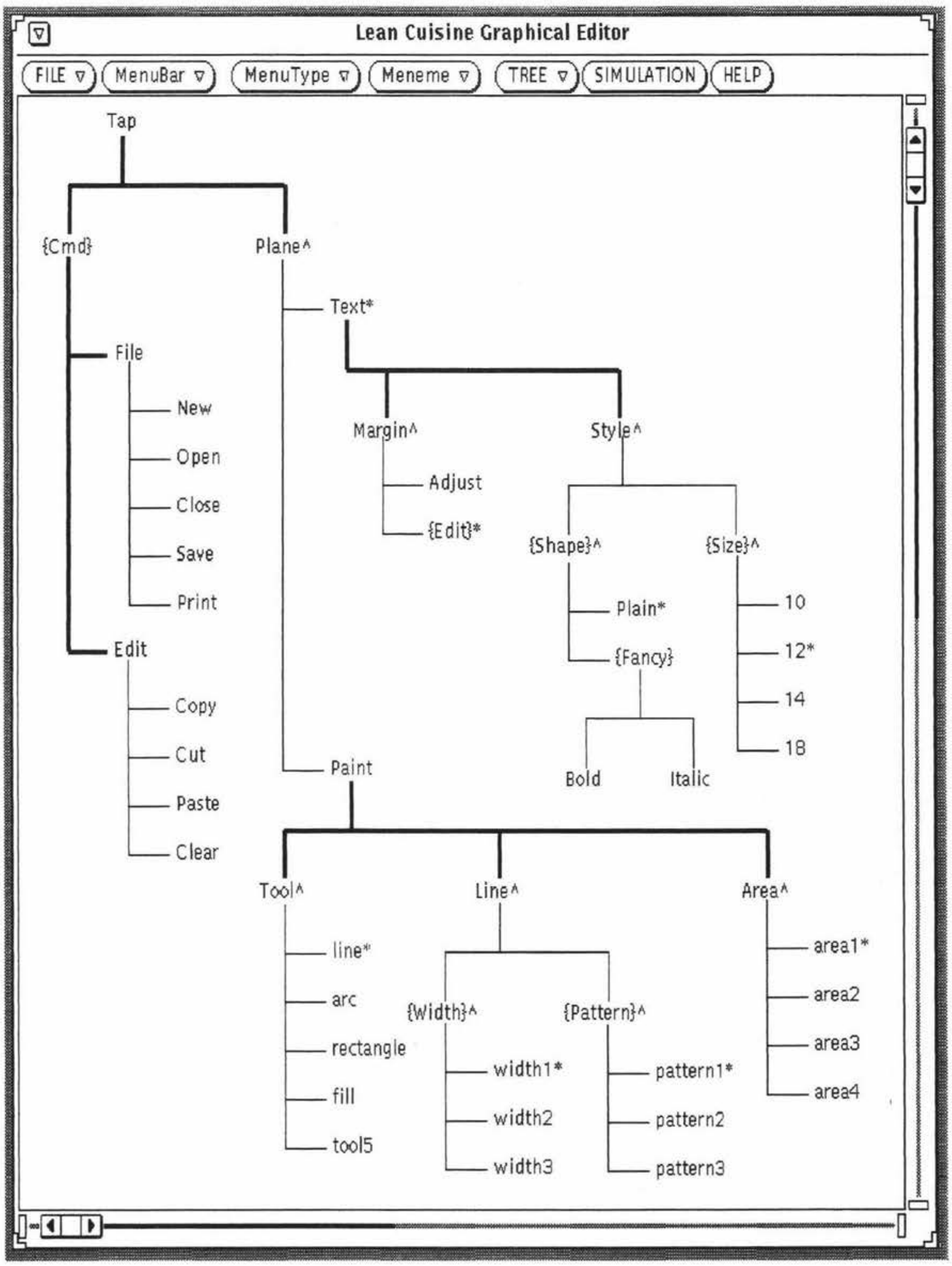
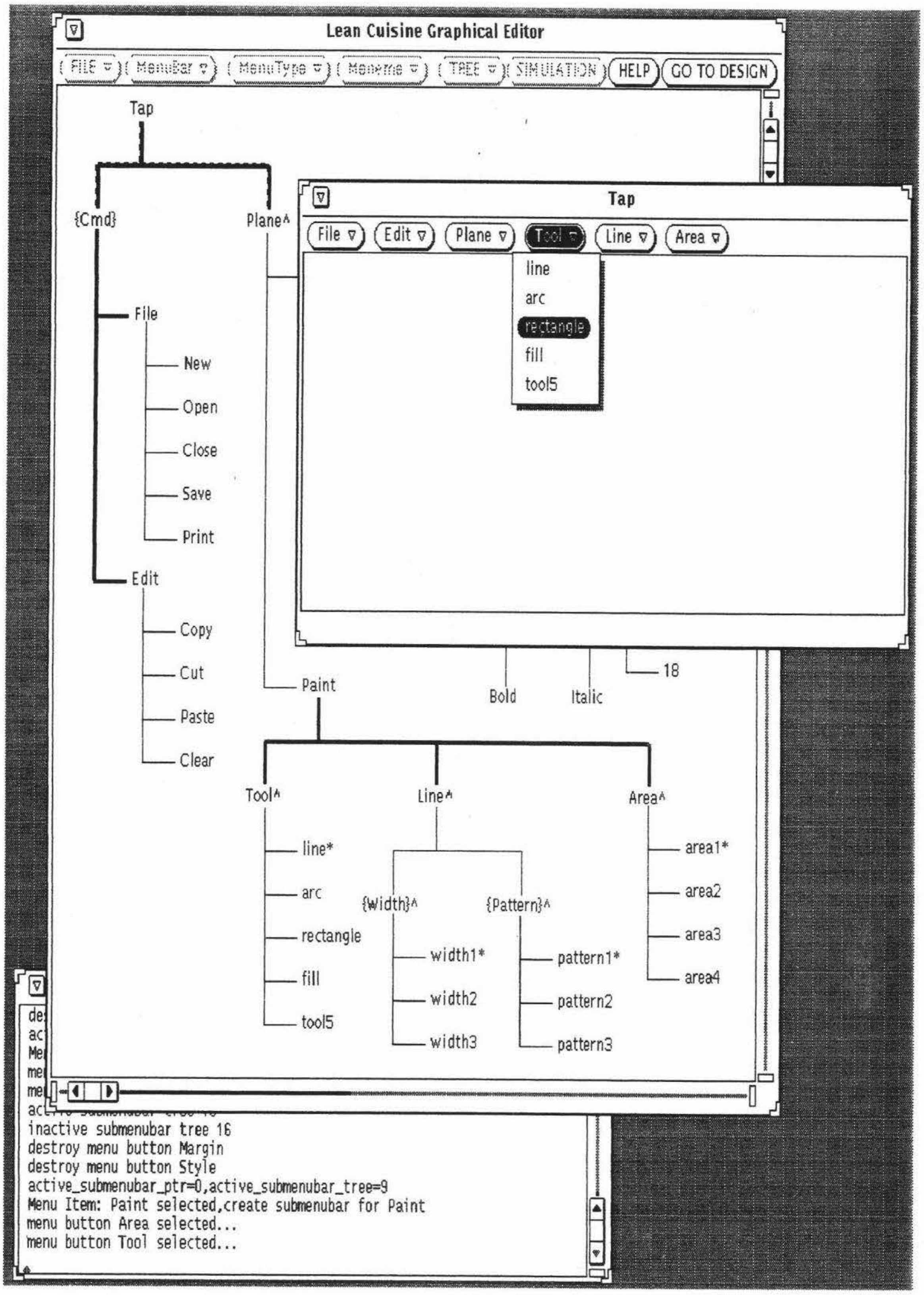


Figure 7.1.9. The Lean Cuisine description of TaP after adding the *Paint* sub-mode facilities.



```
inactive submenubar tree 16
destroy menu button Margin
destroy menu button Style
active_submenubar_ptr=0,active_submenubar_tree=9
Menu Item: Paint selected,create submenubar for Paint
menu button Area selected...
menu button Tool selected...
```

Figure 7.1.10. Elc simulation environment showing the direct simulation of the *Paint* sub-mode of the TaP menu system from its Lean Cuisine description of Figure 7.1.9.

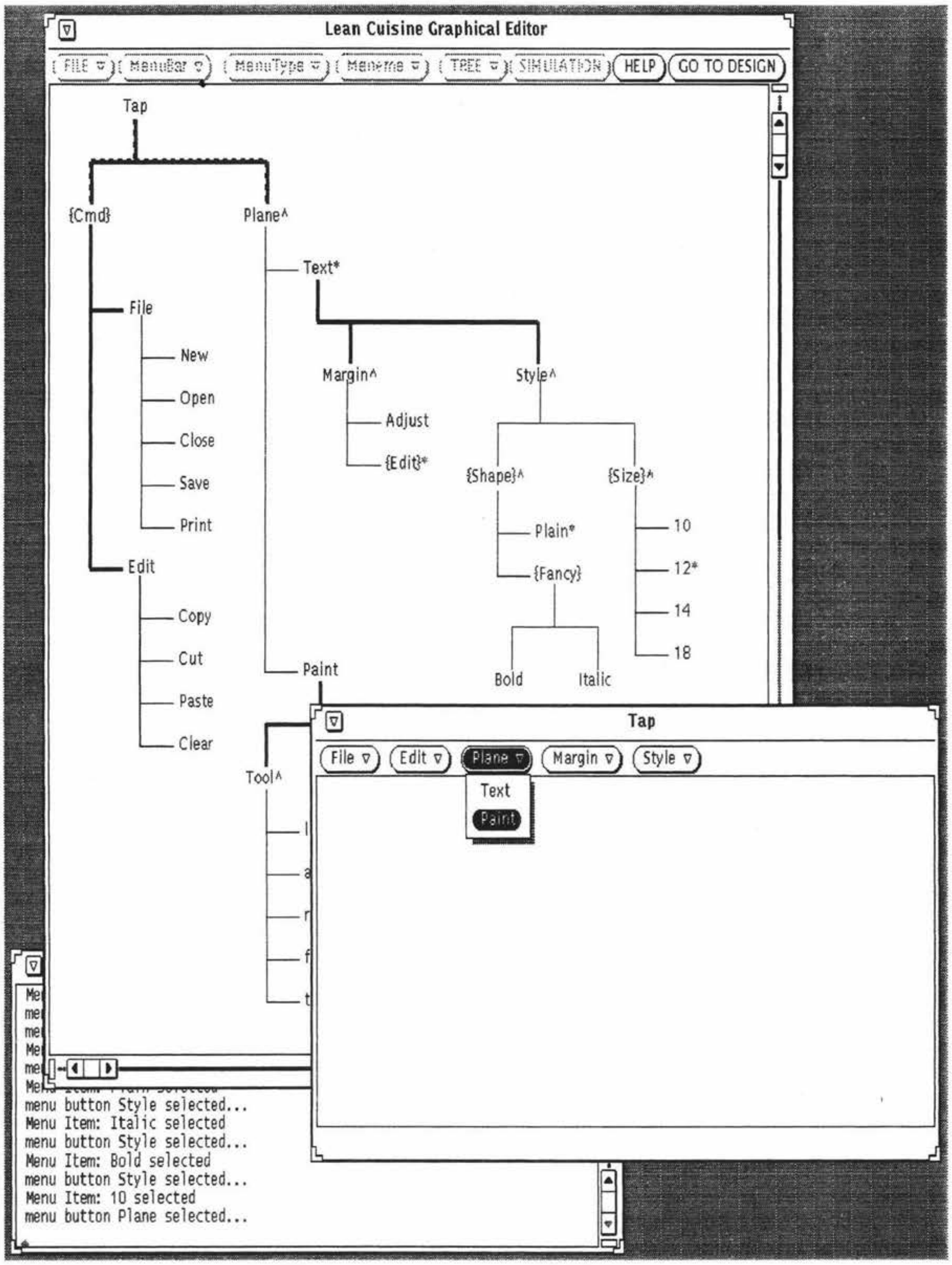
The Dynamic Simulation of the Menu-bar of TaP

Figure 7.1.9 shows an Elc editing window which contains the complete Lean Cuisine graphical specification of TaP. Based on this complete Lean Cuisine graphical specification, the mode of TaP can be dynamically simulated. The TaP application system must always be in one of the two modes, the *Text* sub-mode or the *Paint* sub-mode. According to the complete Lean Cuisine description for TaP [Apperley, 88] shown in Figure 7.1.2 and Figure 7.1.9, the default mode of TaP is the *Text* sub-mode, so when the TaP application is started, two menu-bar items *Margin* and *Style* will be added to the menu-bar for the default *Text* sub-mode (Figure 7.1.11). When the *Paint* sub-mode is set, the menu-bar items *Margin* and *Style* will be deleted from the menu-bar before three new menu-bar items *Tool*, *Line* and *Area* are added to the menu-bar (Figure 7.1.12); similarly, when the *Text* sub-mode is reset, the menu-bar items *Tool*, *Line* and *Area* will be deleted from the menu-bar before the new menu-bar items *Margin* and *Style* are added to the menu-bar.

In the simulation environment provided by the Elc system, based on the Lean Cuisine description, it is very easy to add or delete menu-bar items and the pull-down menus dynamically in the simulation window of a user interface. This dynamic simulation technique used in the Lean Cuisine menu-based interface development environment has not been found in other visual programming tools. Figure 7.1.11 and Figure 7.1.12 show the dynamic simulation of the menu-bar of TaP according to the user's selection.

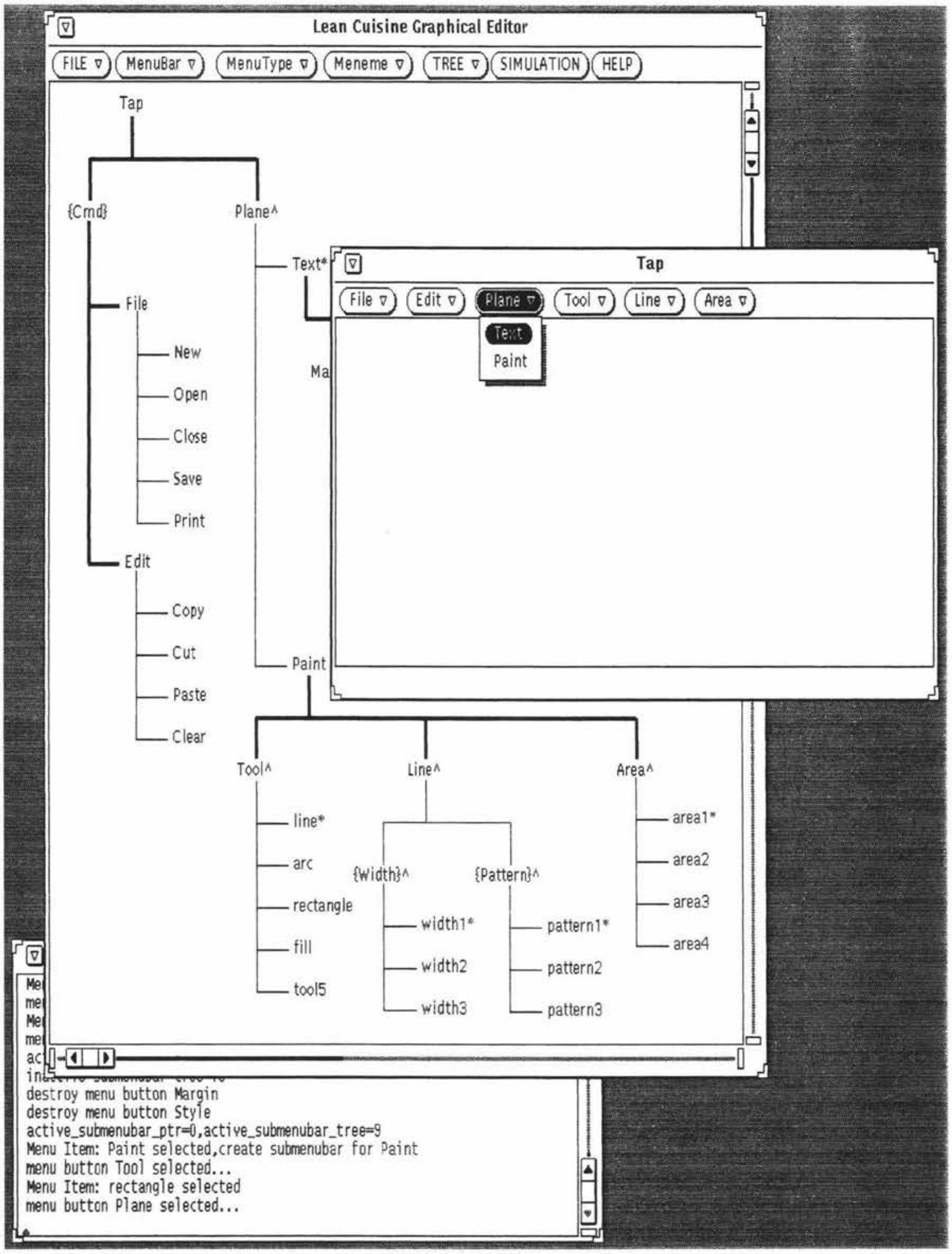
In Figure 7.1.11, the current mode of the simulation window (the front window) of the TaP user interface is the *Text* sub-mode. The Lean Cuisine description (the basic menu-bar and the *Text* sub-mode facilities of TaP) for the current mode is shown in the Elc design window (the middle window, Elc is in simulation mode). When the user selects the menu item *Paint* from the menu *Plane* in the simulation window of TaP, the menu-bar for the *Paint* sub-mode of TaP will appear in the simulation window (the front window shown in Figure 7.1.12).

In Figure 7.1.12, the current mode of the simulation window (the front window) of the TaP user interface is the *Paint* sub-mode. The Lean Cuisine description (the basic menu-bar and the *Paint* sub-mode facilities of TaP) for the current mode is shown in the Elc design window (the middle window, Elc is in simulation mode). When the user selects the menu item *Text* from the menu *Plane* in the simulation window of TaP, the menu-bar for the *Text* sub-mode of TaP will appear in the simulation window (the front window shown in Figure 7.1.11).



```
menu button Style selected...
Menu Item: Italic selected
menu button Style selected...
Menu Item: Bold selected
menu button Style selected...
Menu Item: 10 selected
menu button Plane selected...
```

Figure 7.1.11. The dynamic simulation of the menu-bar of TaP according to the user's selection. The current mode of the simulation window (the front window) of TaP is the *Text* sub-mode, the menu-bar for the *Paint* sub-mode of TaP (shown in the front window of Figure 7.1.12) will appear when the user selects the menu item *Paint* from the menu *Plane* in the simulation window of TaP.



```
destroy menu button Margin
destroy menu button Style
active_submenubar_ptr=0,active_submenubar_tree=9
Menu Item: Paint selected,create submenubar for Paint
menu button Tool selected...
Menu Item: rectangle selected
menu button Plane selected...
```

Figure 7.1.12. The dynamic simulation of the menu-bar of TaP according to the user's selection. The current mode of the simulation window (the front window) of TaP is the *Paint* sub-mode, the menu-bar for the *Text* sub-mode of TaP (shown in the front window of Figure 7.1.11) will appear when the user selects the menu item *Text* from the menu *Plane* in the simulation window of TaP.

An Alternative Design Step of TaP

Alternately, the menu-bar of the TaP application can be simulated based on its basic Lean Cuisine description at an early design stage. Figure 7.1.13 shows the alternative design of the Lean Cuisine description for the menu-bar of TaP.

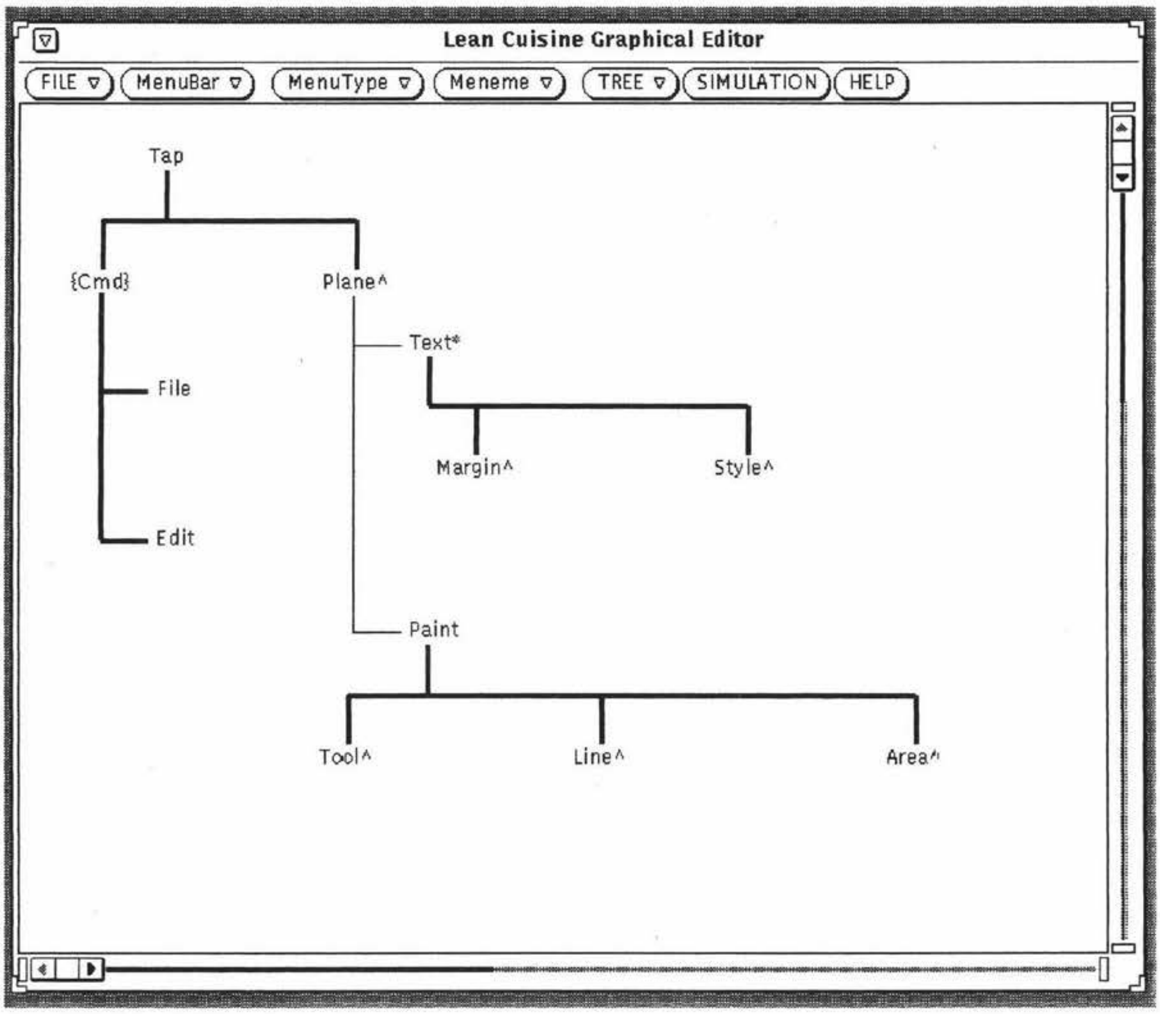
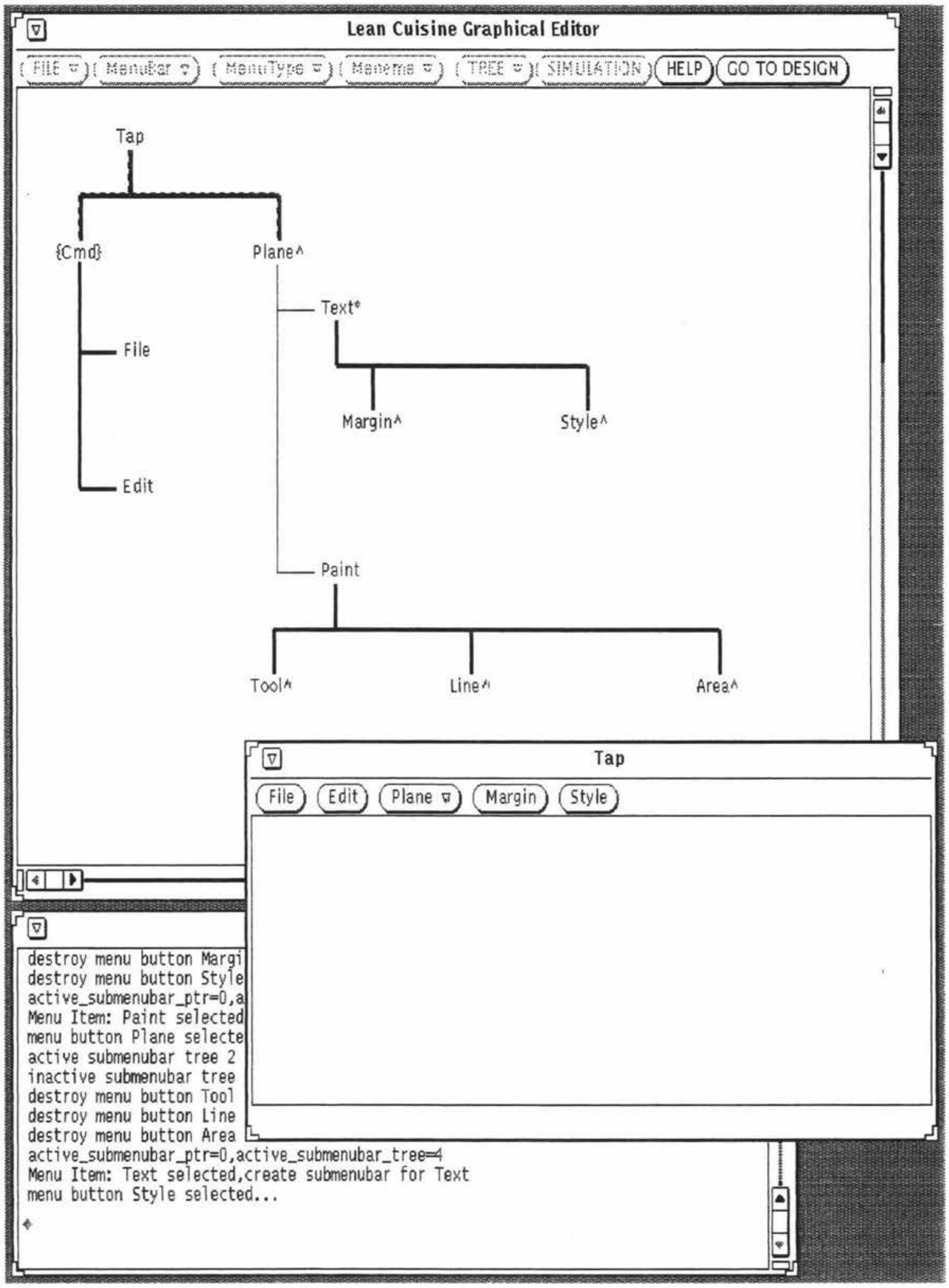


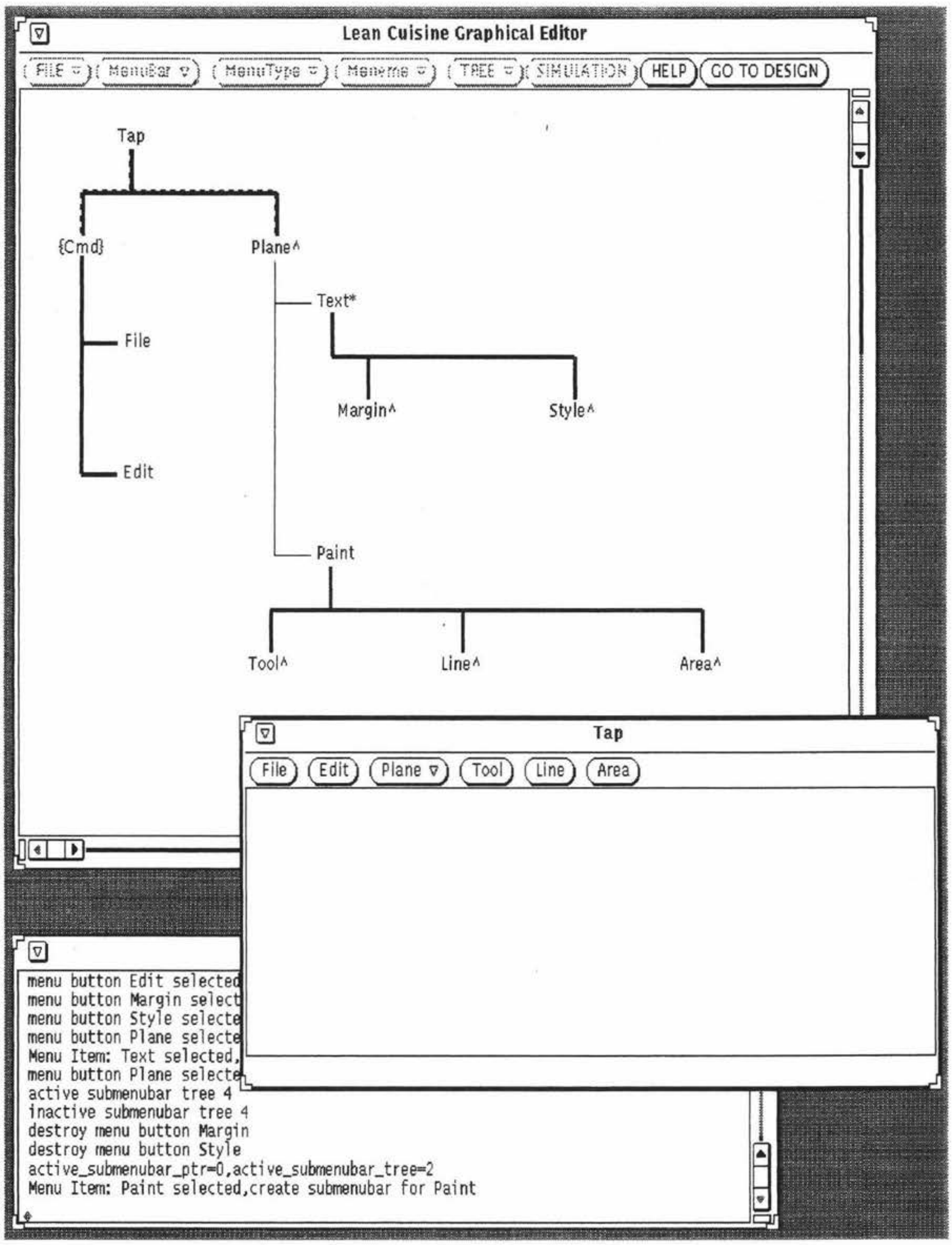
Figure 7.1.13. An alternative design step for TaP.

The basic Lean Cuisine description shown in Figure 7.1.13 is very simple because the Lean Cuisine description of all pull-down menus, except those used to link submenu-bars, are not created in this Elc design window, so it is very easy to modify. This alternative design step is suitable for frequent refinement of the menu-bar before linking all pull-down menus to those menu-bar items. Based on the basic Lean Cuisine description shown in Figure 7.1.13, the TaP interface can be simulated and in the simulation window (the front window shown in Figure 7.1.14 and Figure 7.1.15), the *Text* sub-mode and the *Paint* sub-mode of the menu-bar can be activated according to the user's selection. The dynamic simulation of the menu-bar of TaP according to the user's selection is similar to the procedures shown in Figure 7.1.11 and Figure 7.1.12 but the pull-down menus are not available to be tested at the moment.



```
destroy menu button Margi
destroy menu button Style
active_submenubar_ptr=0,a
Menu Item: Paint selected
menu button Plane selecte
active submenubar tree 2
inactive submenubar tree
destroy menu button Tool
destroy menu button Line
destroy menu button Area
active_submenubar_ptr=0,active_submenubar_tree=4
Menu Item: Text selected,create submenubar for Text
menu button Style selected...
```

Figure 7.1.14. The dynamic simulation of the menu-bar of TaP according to the user's selection (the current mode is the *Text* sub-mode).



```
menu button Edit selecte
menu button Margin select
menu button Style selecte
menu button Plane selecte
Menu Item: Text selected,
menu button Plane selecte
active submenubar tree 4
inactive submenubar tree 4
destroy menu button Margin
destroy menu button Style
active_submenubar_ptr=0,active_submenubar_tree=2
Menu Item: Paint selected,create submenubar for Paint
```

Figure 7.1.15. The dynamic simulation of the menu-bar of TaP according to the user's selection (the current mode is the *Paint* sub-mode).

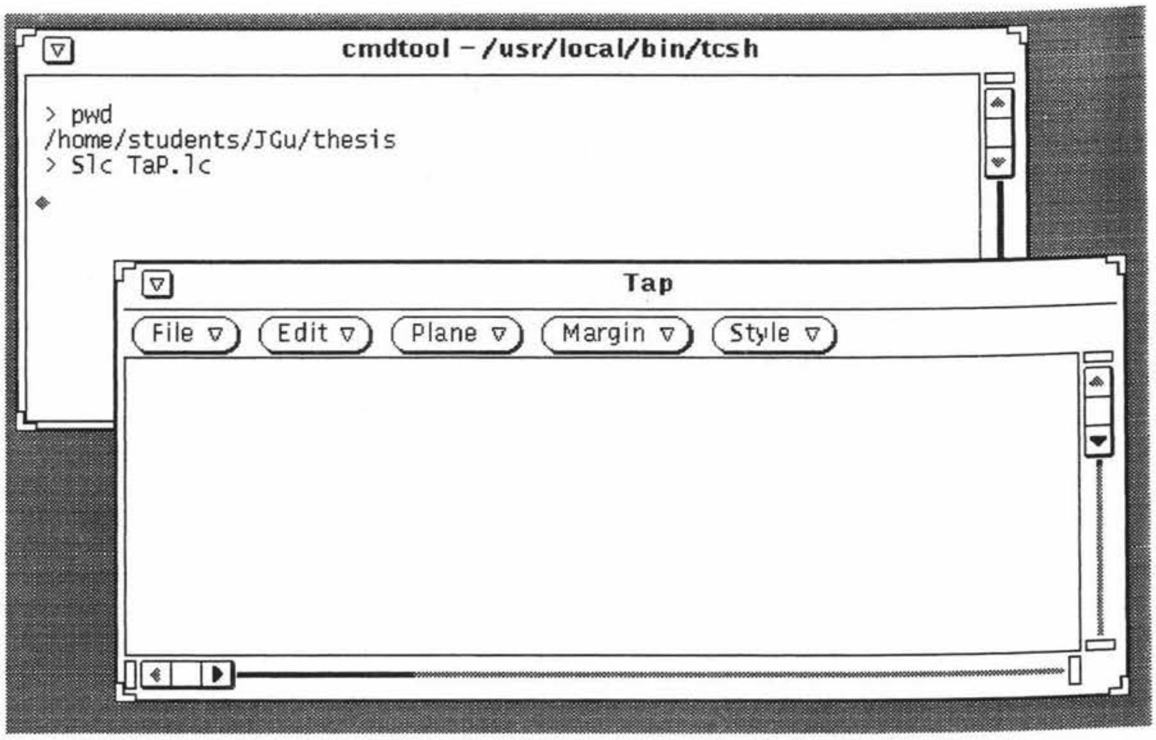
In Figure 7.1.14, the *Text* sub-mode of TaP shown in the front window (the simulation window of TaP) will be changed to the *Paint* sub-mode of TaP (shown in the front window of Figure 7.1.15) when the user selects the menu item *Paint* from the menu

Plane in the simulation window of TaP. Also in Figure 7.1.15, the *Paint* sub-mode of TaP shown in the simulation window of TaP (the front window) will be changed to the *Text* sub-mode of TaP (shown in the front window of Figure 7.1.14) when the user selects the menu item *Text* from the menu *Plane* in the simulation window of TaP.

7.1.3. The Direct Simulation Based on the LC Graphical Specification File

The TaP user interface was simulated during the course of design by changing the mode of the Elc system from the design mode to the simulation mode. It can also be directly simulated in its application environment. The direct simulation is based on its Lean Cuisine graphical specification file which was saved from the Elc design window. To simulate the TaP user interface from its Lean Cuisine graphical specification file, in a command window, simply type the Slc command followed by the name of the Lean Cuisine graphical specification file:

>Slc TaP.lc

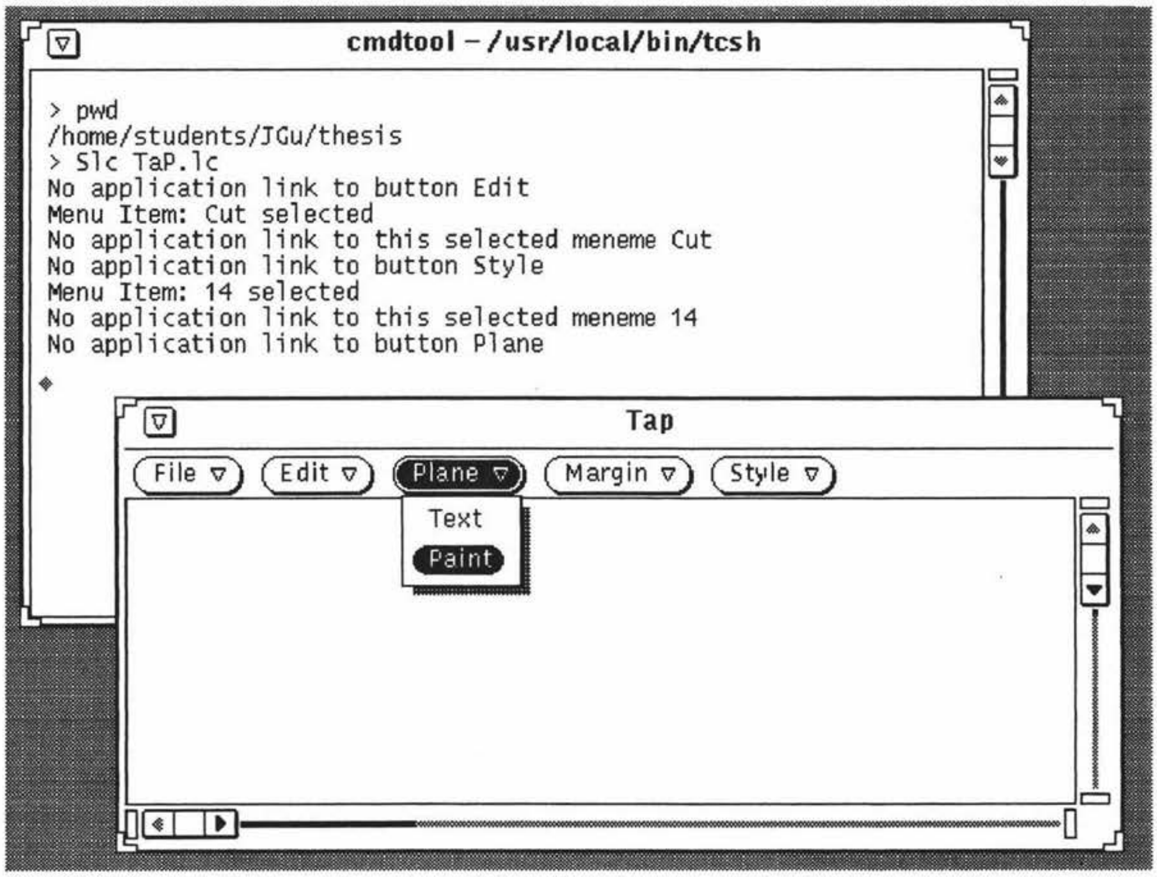


```
> pwd
/home/students/JGu/thesis
> Slc TaP.lc
```

Figure 7.1.16. The direct simulation of TaP in its application environment. **Front**: the simulation window of TaP; **back**: the command window used to display the feedback message.

Here TaP.lc is the name of the Lean Cuisine graphical specification file saved from the Elc design window for the TaP user interface. Figure 7.1.16 shows the direct simulation of the TaP user interface in its application environment, the simulation window of TaP is in the *Text* sub-mode.

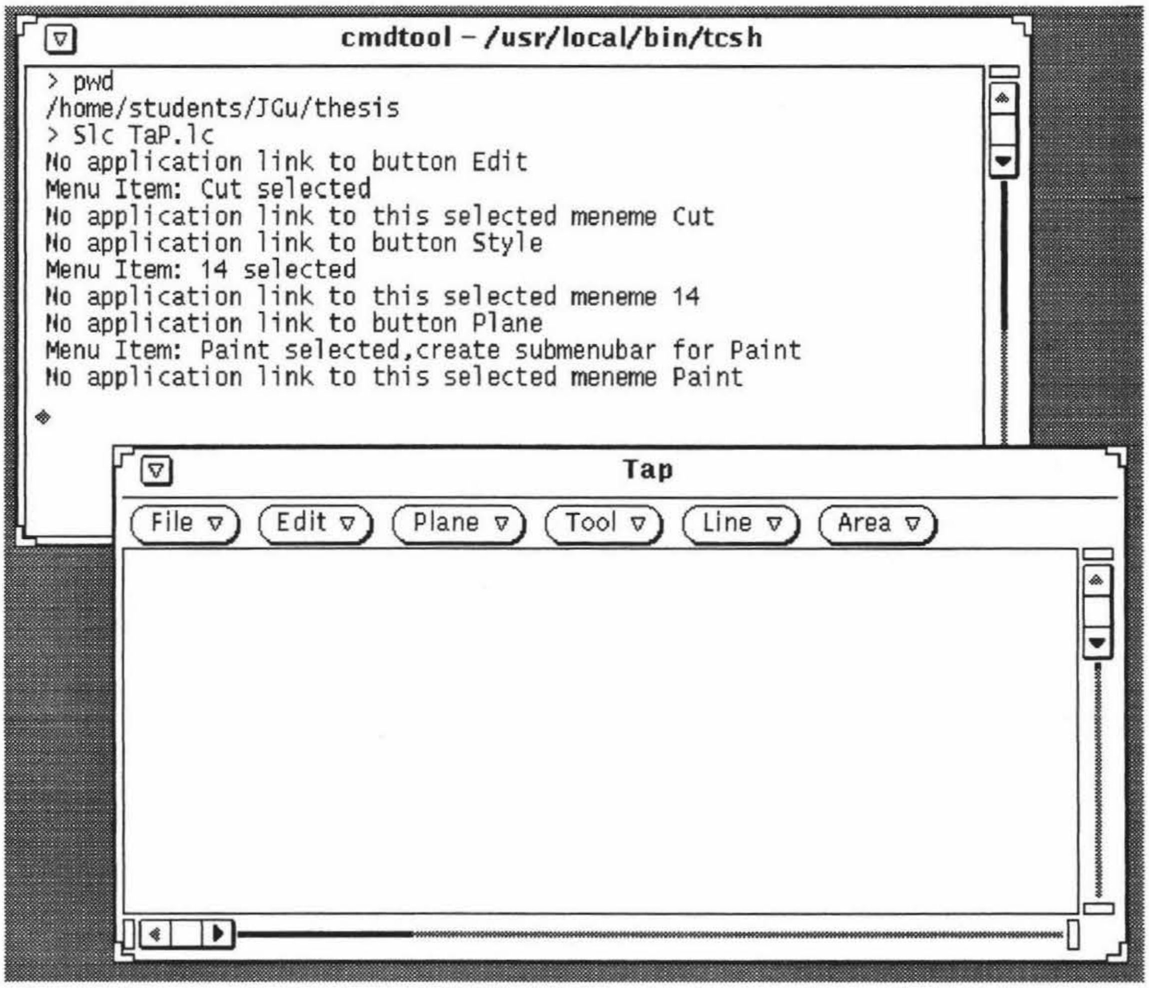
In the simulation window (the front window shown in Figure 7.1.17), the TaP user interface can be tested, the selected menu item is highlighted and the feedback message is displayed in the command window (the back window shown in Figure 7.1.17).



```
> pwd
/home/students/JGu/thesis
> Slc TaP.lc
No application link to button Edit
Menu Item: Cut selected
No application link to this selected meneme Cut
No application link to button Style
Menu Item: 14 selected
No application link to this selected meneme 14
No application link to button Plane
```

Figure 7.1.17. The simulation and test of the menu system of TaP in the application environment. **Front**: the simulation window of TaP; **back**: the command window used to display the feedback message.

In Figure 7.1.17, the current mode of the simulation window of TaP is the *Text* mode, the mode of the simulation window of TaP will be changed to the *Paint* mode according to the current menu selection: select the menu item *Paint* from the menu *Plane*. After the menu selection, in the simulation window of TaP, the menu-bar items (*Margin*, *Style*) of the *Text* mode will be destroyed and the menu-bar items (*Tool*, *Line*, *Area*) will be added. The *Paint* mode of the simulation window of TaP is shown in Figure 7.1.18, the feedback message displayed in the command window (the back window shown in Figure 7.1.18) also shows this mode change.

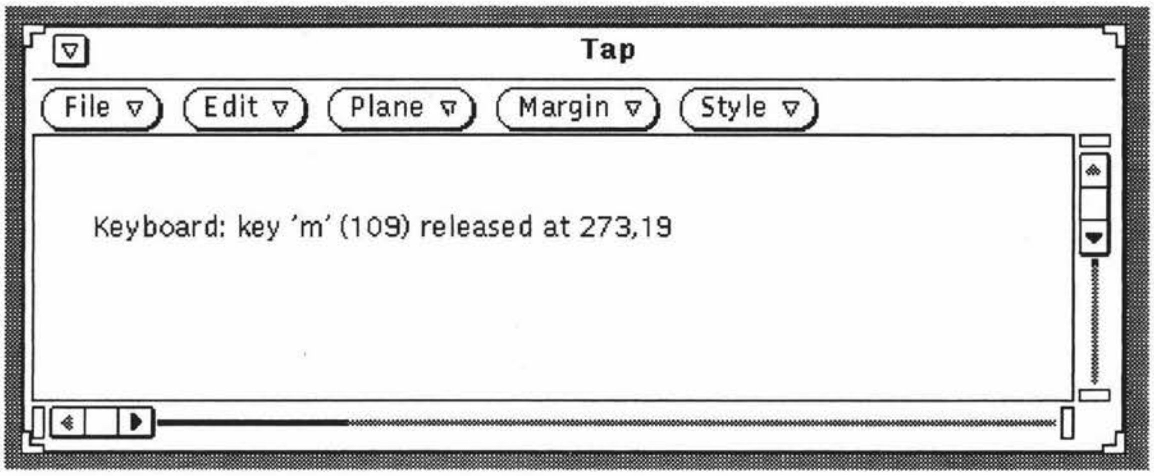


```
> pwd
/home/students/JGu/thesis
> Slc TaP.lc
No application link to button Edit
Menu Item: Cut selected
No application link to this selected meneme Cut
No application link to button Style
Menu Item: 14 selected
No application link to this selected meneme 14
No application link to button Plane
Menu Item: Paint selected,create submenubar for Paint
No application link to this selected meneme Paint
```

Figure 7.1.18. The *Paint* mode of TaP simulated in the application environment. **Front**: the simulation window of TaP; **back**: the command window used to display the feedback message.

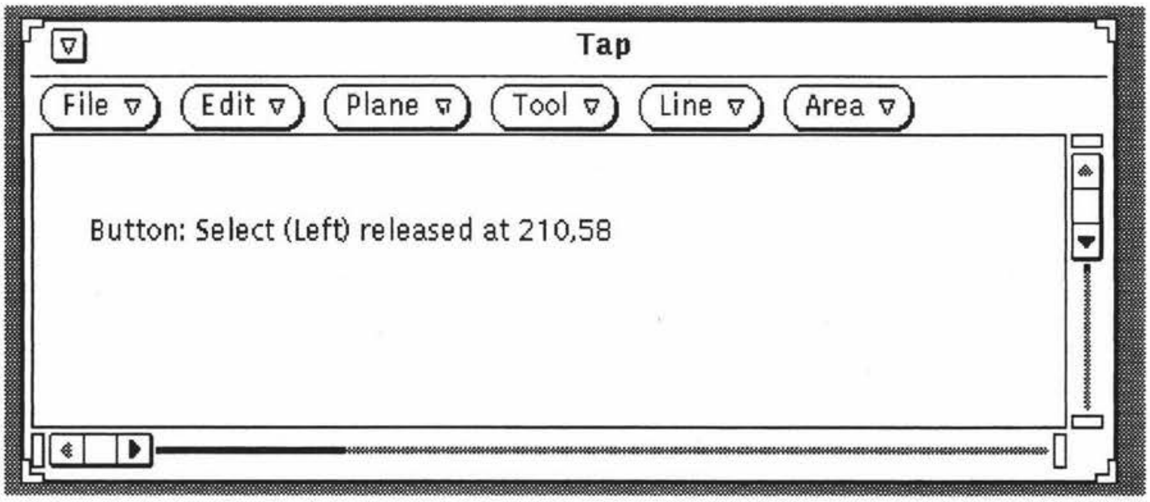
When the TaP interface is directly simulated in its application environment based on its Lean Cuisine graphical specification file, besides the simulation and test of its menu system, the keyboard input and mouse actions can be handled by the Slc tool. A keyboard event takes place when a key is pressed or released; Slc will display the message (the ASCII and decimal code of a key which is pressed or released and the current mouse position) on the simulation window when the keyboard event takes place, Figure 7.1.19 shows how Slc handles the keyboard input in the simulation window of TaP. A mouse event takes place when a button (left, middle right) of the mouse is pressed or released, the mouse pointer moves, or the mouse pointer moves while

SELECT (left button), ADJUST (middle button), or MENU (right button) is held down, Slc will display the message (the event type and the current mouse position) on the simulation window when the mouse event takes place. Figure 7.1.20 shows how Slc handles the mouse event in the simulation window of TaP.



```
Keyboard: key 'm' (109) released at 273,19
```

Figure 7.1.19. A keyboard input in the simulation window of TaP



```
Button: Select (Left) released at 210,58
```

Figure 7.1.20. A mouse event in the simulation window of TaP.

As discussed in section 5.5, it is not necessary to run the Slc system in TaP application environment for simulating the TaP user interface because the Slc system embedded in Elc provides a complete user interface simulation and modification environment and covers all simulation functions, except keyboard events and mouse events simulation, which Slc provides when it is run independently. But when Slc is used out of the Elc system, it simulates TaP in its application environment where the TaP interface will work and gives the evaluators a real feel of how the TaP user interface works.

7.1.4. Using Glc to Generate TaP User Interface Source Code Files

So far the Lean Cuisine graphical specification for the TaP user interface has been created in an Elc editing window, and based on its Lean Cuisine graphical specification the TaP user interface has been simulated in both the design environment and the application environment. The next step is to generate the TaP user interface source code files by using the Glc command followed by the name of its Lean Cuisine graphical specification file, TaP.lc. In a command window, enter the command:

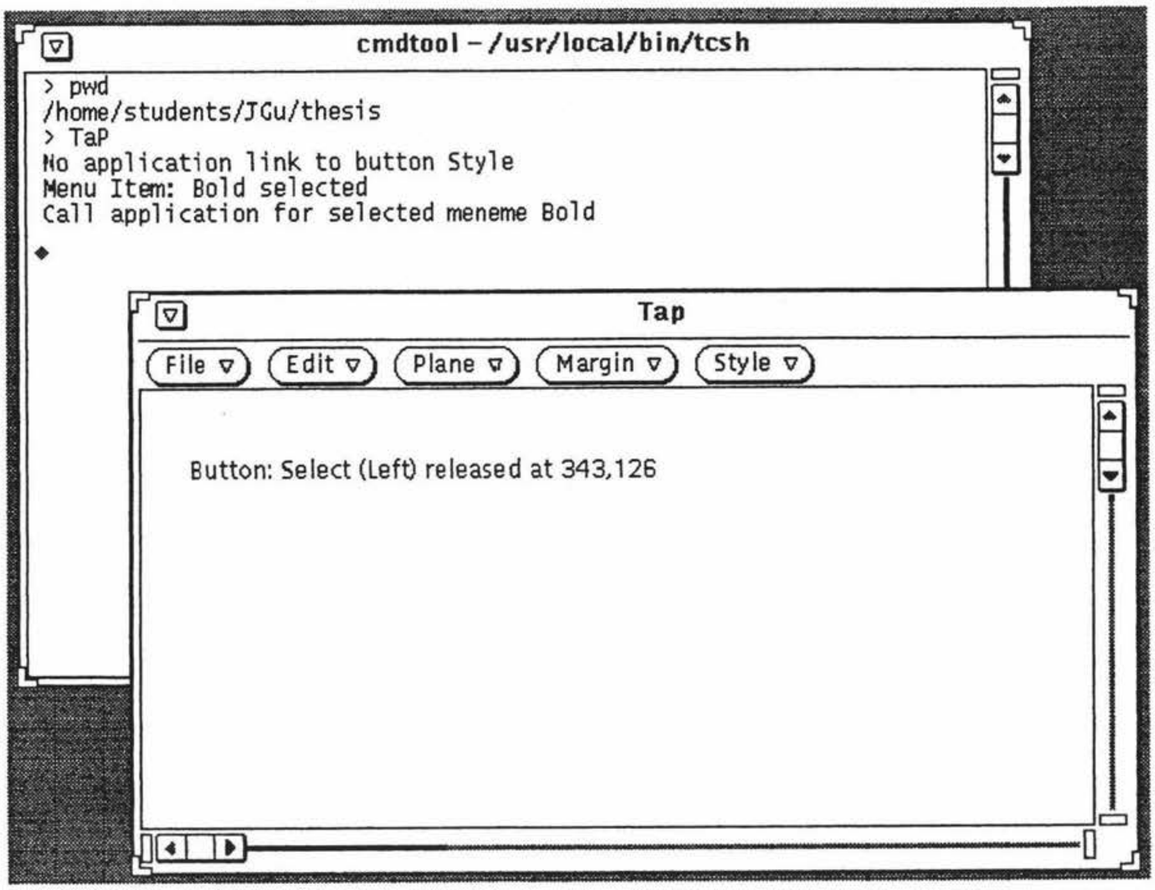
>Glc TaP.lc

The Glc tool generates four source code files and a Makefile which can be used with the make command to compile the source code files. As discussed in section 6.3, The Glc command names its four generated source code files after the original Lean Cuisine graphical specification file name, stripping off the .lc extension and adding new extensions to identify each file. In this example, the command "Glc TaP.lc" generates the following files:

TaP_main.c
TaP_link_app.c
TaP_app.c
TaP_app.h
Makefile

Based on above four source code files and Makefile, an executable file for the TaP user interface can be generated. To do this, simply enter command "make" in the command window. The compiler goes to work and uses the Makefile parameters for the compile, when it finishes it leaves an executable file named "TaP" according to the default file name setting in above Makefile. The real TaP user interface can now be run by entering "TaP" in the command window (the back window shown in Figure 7.1.21), the TaP user interface (the front window shown in Figure 7.1.21) has the same appearance as is shown in Figure 7.1.16 but it is a real working model generated from the interface source code.

Based on this real working model of TaP, The mode of the TaP user interface can be changed, the menu system can be tested, the selected item will be highlighted and the feedback messages will be displayed in the command window, and the keyboard events and mouse events will be handled.



```
> pwd
/home/students/JGu/thesis
> TaP
No application link to button Style
Menu Item: Bold selected
Call application for selected meneme Bold
```

```
Button: Select (Left) released at 343,126
```

Figure 7.1.21. The TaP user interface (real working model) generated from the interface source code. **Front**: the *Text* mode of the TaP user interface, **back**: the command window used to run the TaP real working model and display the feedback message.

7.1.5. Link Application to TaP User Interface

The semantic links with the application routines are described in [Apperley, 88] and this semantic description is shown in Table 7.1.

Meneme	Type	Action
TaP	NT	Invokes an initialisation **TaPInit** routine in the application.
{Cmd}	V	No application action
File	NT	No application action; provides access to file operation menu.
New	T	Invokes system **newfile** routine.
Open	T	Invokes system **openfile** routine.
Close	T	Invokes system **closefile** routine.
Save	T	Invokes system **savefile** routine.
Print	T	Invokes system **printfile** routine.

Table 7.1a. The application links [Apperley, 88].

Meneme	Type	Action
Edit	NT	No application action; provides access to edit operation menu.
Copy	T	Invokes application routine **copy**, to copy current selection to clipboard.
Cut	T	Invokes application routine **cut**, to cut current selection and copy it to clipboard.
Paste	T	Invokes application routine **paste**, to paste clipboard into current selection.
Clear	T	Invokes application routine **clear**, to cut current selection without saving it in the clipboard.
Plane	NT	No application action; provides access to text/paint menu.
Text	NT	Invokes application routine **setplane** to indicate to application that edit plane has been redefined.
Margin	NT	No application action.
Adjust	T	Invokes application routine **margin** to indicate to application that margin adjust operation is to be carried out on-screen. System enters modal sub-dialogue.
{Edit}	V	No application action.
Style	NT	No application action; provides access to style menu.
{Shape}	V	No application action.
Plain	T	Invokes application routine **Tshape** to indicate to application that a change of text shape has been made.
{Fancy}	V	No application action.
Bold	T	Invokes 'application routine **Tshape** to indicate to application that a change of text shape has been made.
Italic	T	Invokes application routine **Tshape** to indicate to application that a change of text shape has been made.
{Size}	V	No application action.
size n	T	(four menemes) Each of these menemes should invoke **Tsize** to indicate to the application that a change of text size has been made.
Paint	NT	Invokes application routine **setplane** to indicate to application that edit plane has been redefined.
Tool	NT	No application action; provides access to tool menu.
tool n	T	(seven menemes) Each of these menemes should invoke **tool** to indicate to the application that a change of tool has been made.
Line	NT	No application action; provides access to line menu.
{Width}	V	No application action.
width n	T	(four menemes) Each of these menemes should invoke **linewidth** to indicate to the application that a change of line width has been made.
{Pattern}	V	No application action.
pattern n	T	(four menemes) Each of these menemes should invoke **penpat** to indicate to the application that a change of pen pattern has been made.
Area	NT	No application action; provides access to area pattern menu.
area n	T	(four menemes) Each of these menemes should invoke **areapat** to indicate to the application that a change of fill pattern has been made.

Table 7.1b. The application links, continued [Apperley, 88].

In Table 7.1, the application or system routines are shown in **bold**. Each of the terminal menemes invokes its application routine. No application action has been suggested for each of the virtual menemes and the non-terminal menemes except the non-terminal meneme TaP. The semantic description, shown in Table 7.1, is a preliminary to the full specification of the application routines.

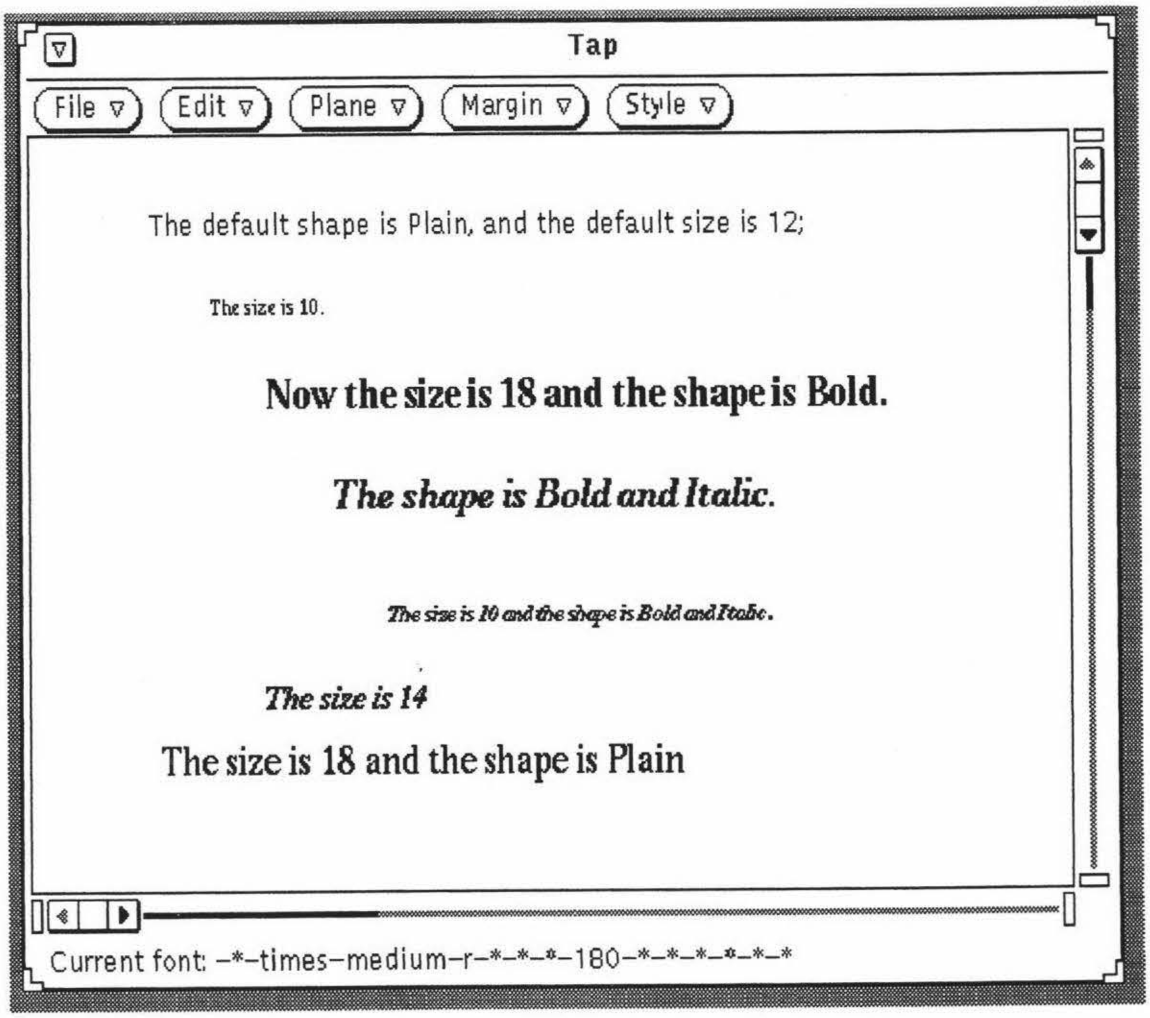
The TaP_link_app.c is the link socket between the TaP user interface and its application programs. It includes the event handling procedures used to receive events and call application procedures when the events occur. The original application procedures generated by the Glc tool are included in the TaP_app.c file and each of these procedures is designed to display the feedback message in the command window when they are called.

The real application procedures were developed on the basis of the original application procedures. During the course of the development of the real application, the new executable file of the TaP application was always generated by using the "make" command after linking some of the real application procedures to the TaP user interface. The new linked application procedures were tested and modified based on this working model of the TaP application at different stage.

Besides the implementation of the application routines shown in Table 7.1, other very important application routines, which are invoked by keyboard events, mouse motion events and mouse button events, are also developed according to the requirements of the real TaP application.

7.1.6. TaP in Use

Now the "complete" TaP application can show its basic text-processing capability and its painting function. When the TaP application is started, its default mode is the *Text* mode. The menu-bar items are *File*, *Edit*, *Plane*, *Margins* and *Style* and the default text shape is plain and the default text size is 12, the other shapes (bold, italic, and bold-italic) and sizes (10, 14, and 18) can be set by selecting the menus. Figure 7.1.22 shows that in a TaP application window, the characters have the possible shape combinations and different sizes.



The default shape is Plain, and the default size is 12;

The size is 10.

Now the size is 18 and the shape is Bold.

The shape is Bold and Italic.

The size is 10 and the shape is Bold and Italic.

The size is 14

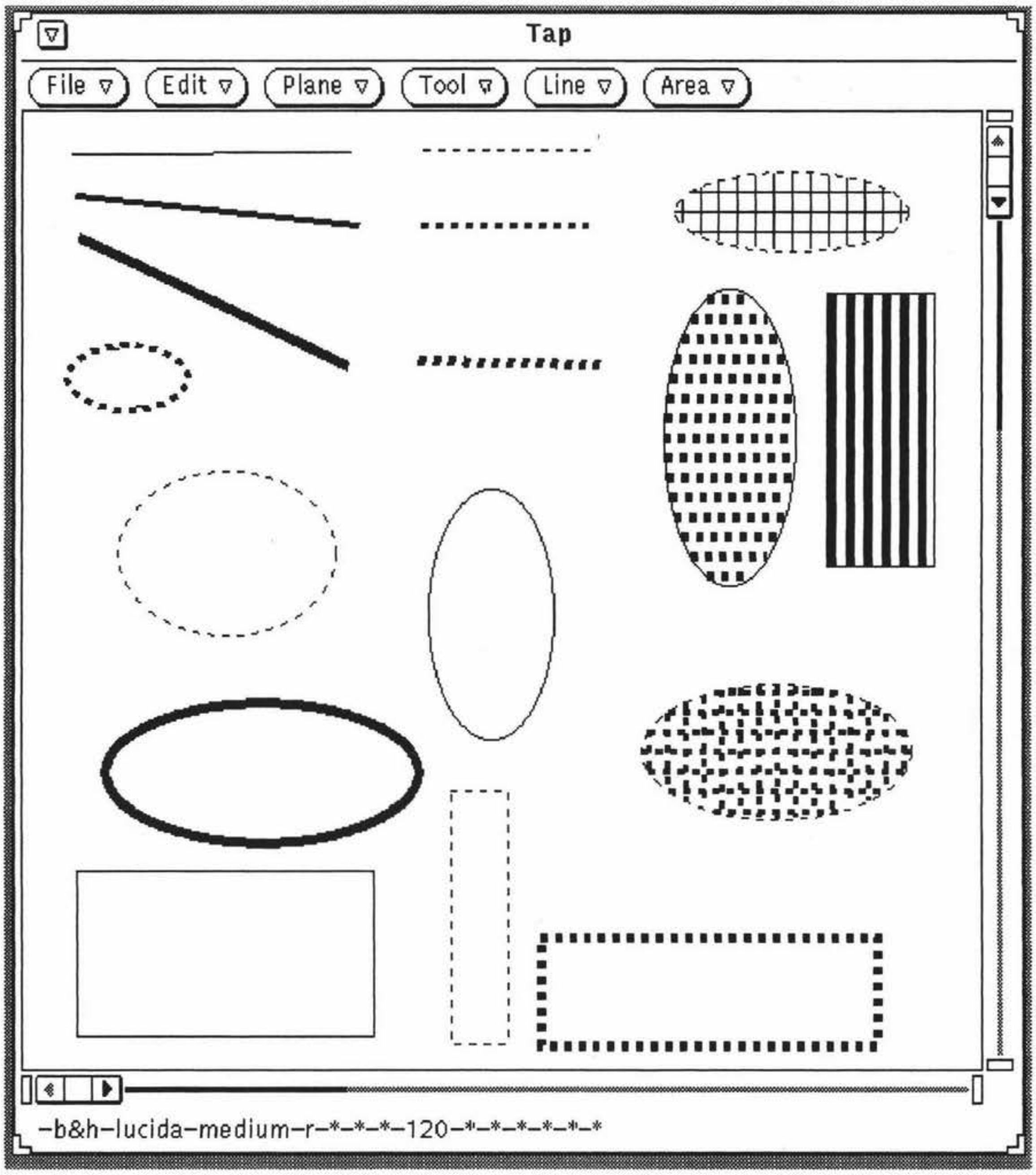
The size is 18 and the shape is Plain

Current font: -*-times-medium-r-*-*-*-180-*-*-*-*-*-*

Figure 7.1.22. The text with possible shape combinations and different sizes.

When changing the mode of a TaP application from the *Text* mode to the *Paint* mode, the menu-bar items *Margins* and *Style* are deleted from the menu-bar and the menu-bar items *Tool*, *Line*, and *Area* are added to the menu-bar.

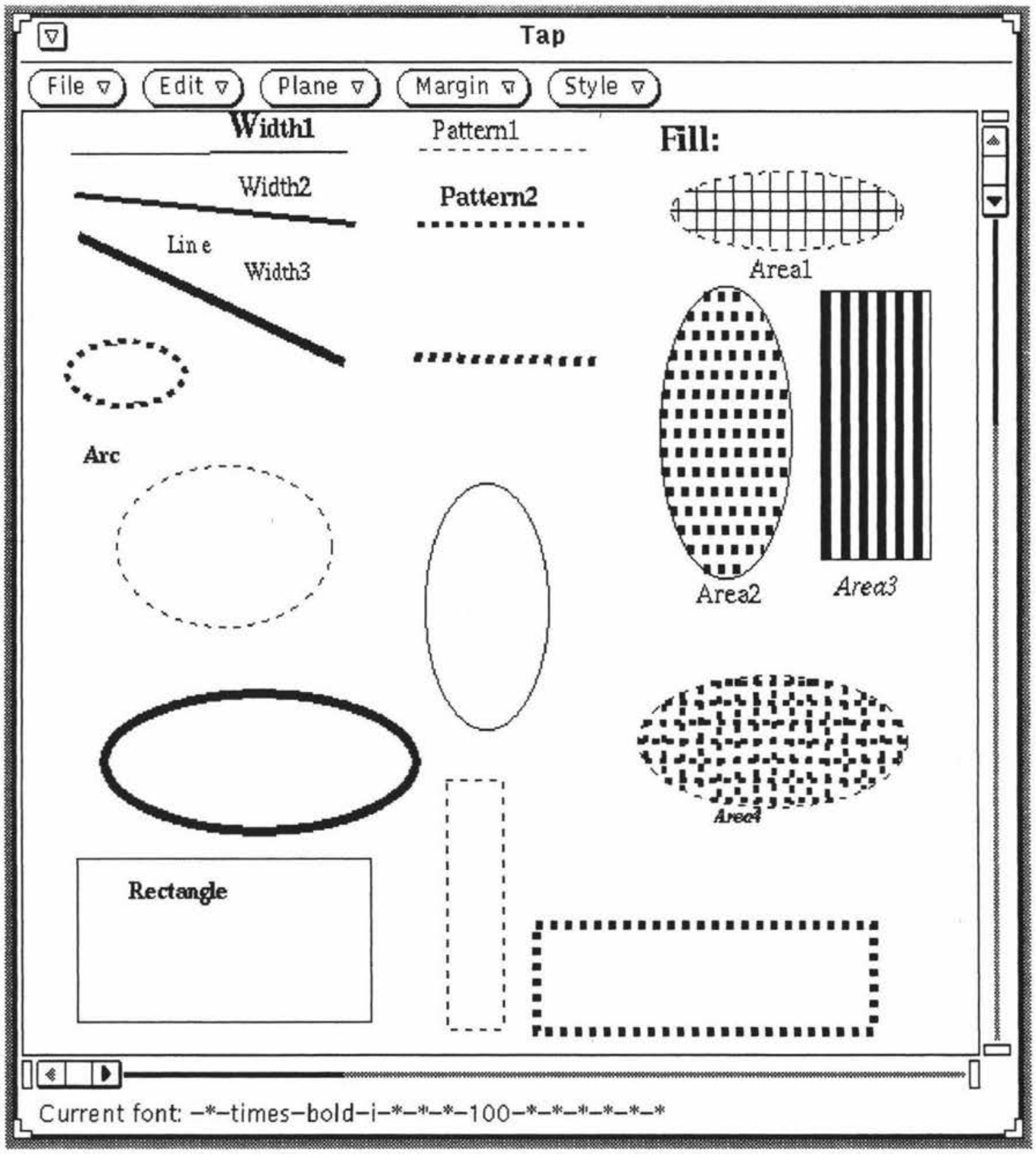
In the *Paint* mode, the available tools are *line*, *arc*, *rectangle* and *fill*, the default setting of the tool is *line*; four line widths and three line styles are provided for the *line*, *arc* and *rectangle* tools; and four fill styles are provided for the *fill* tool. The lines, ellipses and rectangles displayed in a TaP application show the available line widths, line styles, and fill styles, as shown in Figure 7.1.23.



-b&h-lucida-medium-r-*-*-*-120-*-*-*-*-*-*

Figure 7.1.23. The examples show the available line widths, line styles, and fill styles used in a TaP application.

It is possible to create text and drawing alternately in a TaP application window. Based on the mode exchange, new graphics can be added to existing text and new text can be added to existing graphics. Figure 7.1.24 shows a TaP application window which includes both text created in the text mode and drawing created in the painting mode.



Current font: -*-times-bold-i-*-*-*-100-*-*-*-*-*-*

Figure 7.1.24. A TaP application window including both text and drawing.

7.2. Development of the ELC Lean Cuisine Graphical Editor

7.2.1. About ELC

The ELC system is a window based Lean Cuisine graphical editor which will have the same functions as the Elc graphical editor has and it will be developed in the menu interface development environment [Chapter 4, 5, 6] based on the Lean Cuisine visual notation [Apperley & Spence, 89]. Three supporting tools, Elc, Slc and Glc, will be used during the ELC development. First, the Lean Cuisine graphical dialogue specification for the ELC user interface will be created in an Elc design window; second, based on the Lean Cuisine graphical dialogue specification, the menu interface of ELC will be simulated in both the design environment and the application environment by using the Slc tool; third, by using the Glc tool, the source code files of the ELC user interface will be generated from its Lean Cuisine graphical specification file.

Because the ELC system will have the same functions as the Elc graphical editor has, so the application programs which have been developed for the Elc graphical editor will be linked to the ELC user interface. The user interface of the ELC graphical editor will be similar to that of the Elc system, the Elc user interface was created by programming but the ELC user interface will be generated from its Lean Cuisine graphical specification.

The aim of the development of ELC is to try out another example of software development in the menu interface development environment based on the Lean Cuisine visual notation.

7.2.2. The Lean Cuisine Description of the Elc System

As mentioned above, the ELC graphical editor will have the same functions as the Elc graphical editor has, and the ELC system will use the same menu system as the Elc system. Based on the details of the Elc graphical editor discussed in Chapter 4, the Lean Cuisine description of the Elc system will be given in the following paragraphs.

The Mode of Elc
The Elc graphical editor has two modes, the design mode and the simulation mode. The design and the simulation of a user interface cannot be carried out at a time in the Elc system, so the commands used in the design mode of the Elc system and the commands

used in the simulation mode of the Elc system are mutually exclusive commands. The Lean Cuisine description of the mode of Elc is shown in Figure 7.2.1, the default mode of the Elc system is the design mode.

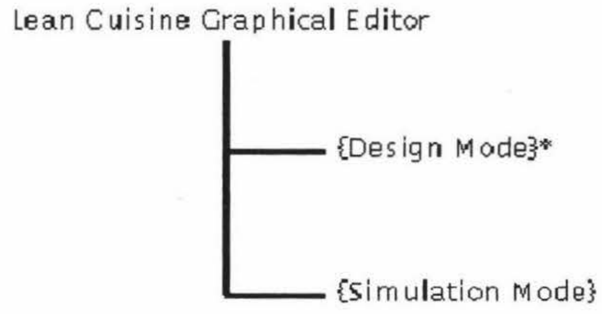


Figure 7.2.1. The Lean Cuisine description of the mode of the Elc system.

The Design Mode of Elc

In the Elc system, the menus and commands which are available in the design mode are *FILE*, *MenuBar*, *MenuType*, *Meneme*, *TREE*, *SIMULATION* and *HELP*. Figure 7.2.2 shows the menu-bar, the menu buttons and their submenus, and the command buttons of the Elc system which is in the design mode.

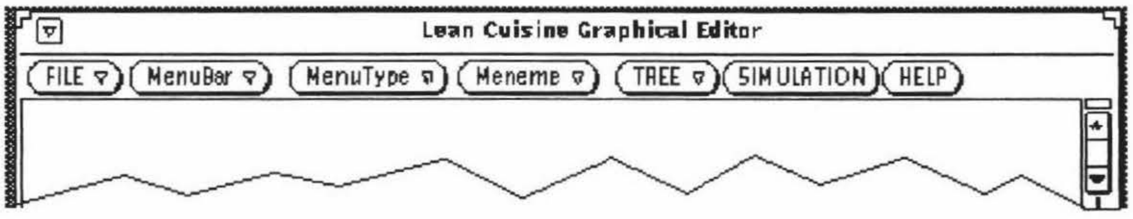


Figure 7.2.2a. The menu-bar of the design mode of Elc.

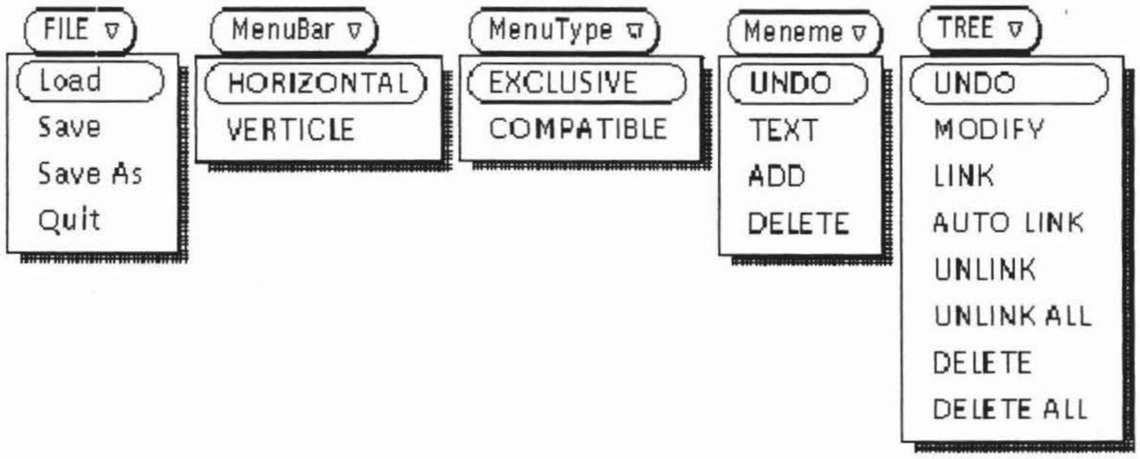


Figure 7.2.2b. The menu buttons and pull-down menus in the design mode of Elc.

Figure 7.2.2c. The command buttons in the design mode of Elc.

On the supposition that the ELC graphical editor will have the same functions as the Elc graphical editor has, the same menus and commands shown in Figure 7.2.2 will be used in the design mode of the ELC system, the following paragraphs show how to create the Lean Cuisine description of these menus and commands of Elc.

The commands in the *MenuBar* menu and the *MenuType* menu are used to select the type of a Lean Cuisine tree which will be created, only one LC tree, which may be the description of a part of a menu-bar or a menu, can be created at a time, so the commands in the *MenuBar* menu and the *MenuType* menu form two mutually exclusive command sets, and the *MenuBar* menu and the *MenuType* menu are mutually exclusive menus. Figure 7.2.3 shows the Lean Cuisine description of the *MenuBar* menu and the *MenuType* menu.

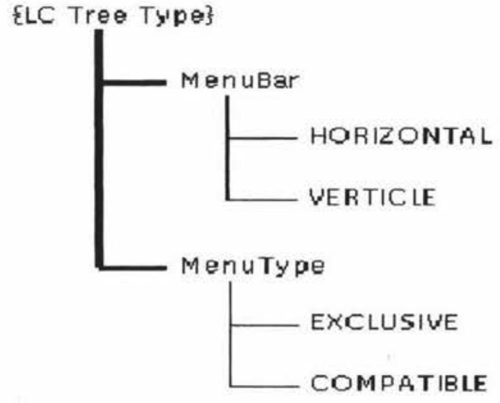


Figure 7.2.3. The LC description of the *MenuBar* menu and the *MenuType* menu.

The commands in the *Meneme* menu are used to perform meneme operations on the currently selected Lean Cuisine tree, and the commands in the *TREE* menu are used to perform tree operations on the currently selected Lean Cuisine tree. Of the *Meneme* and *TREE* commands, only one can be carried out at a time, so the commands in the *Meneme* menu and the commands in the *TREE* menu form two mutually exclusive command sets, and the *Meneme* menu and the *TREE* menu must be mutually exclusive menus. Figure 7.2.4 shows the Lean Cuisine description of the *Meneme* menu and the *TREE* menu.

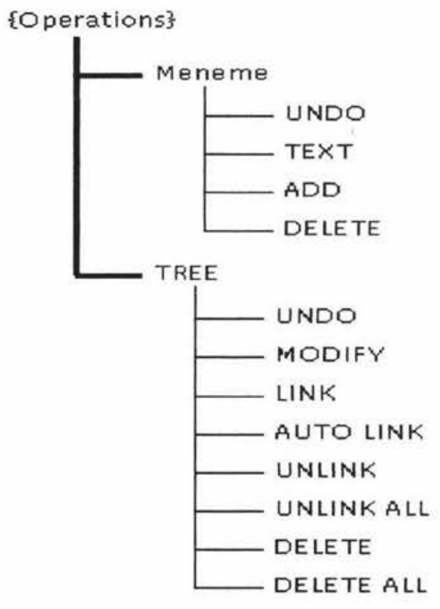


Figure 7.2.4. The Lean Cuisine description of the *Meneme* menu and the *TREE* menu.

Figure 7.2.3 and Figure 7.2.4 can be combined into a single LC diagram (Figure 7.2.5) which is the Lean Cuisine description of the *{Design Commands}* of the Elc system.

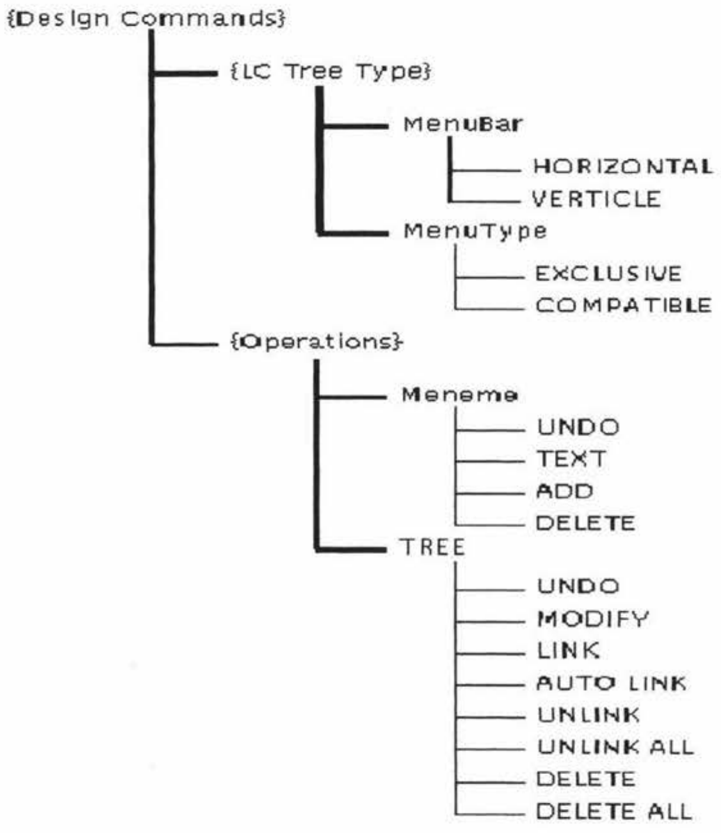


Figure 7.2.5. The LC description of the *{Design Commands}* of the Elc system.

In Figure 7.2.5, the commands used to select the type of a Lean Cuisine tree and the commands used to perform operations on the currently selected Lean Cuisine tree are mutually exclusive.

The commands in the *FILE* menu are mutually exclusive, however, these commands are mutually compatible with the design commands shown in Figure 7.2.5. Figure 7.2.6 shows the Lean Cuisine description of the *{Basic Commands}* of the Elc system.

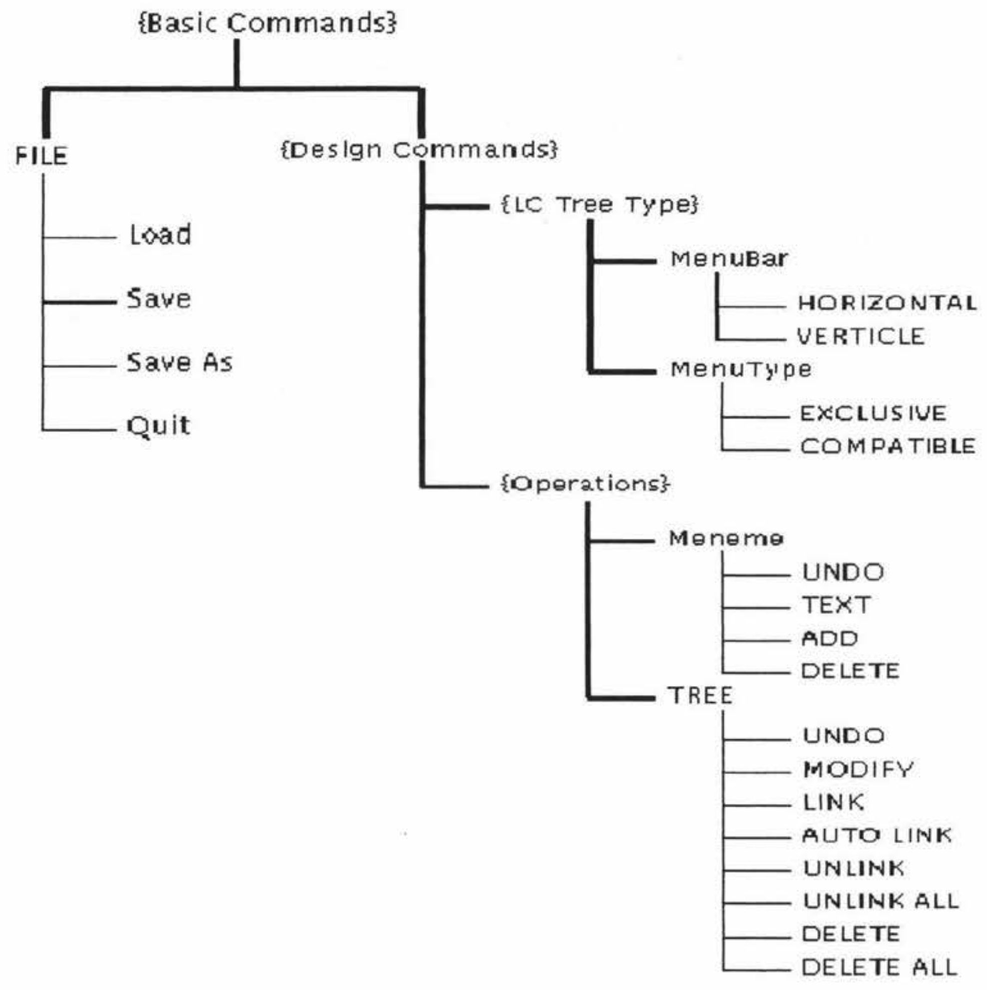


Figure 7.2.6. The Lean Cuisine description of the *{Basic Commands}* of the Elc system.

The command *SIMULATION* is used to set the simulation mode of the Elc system and is mutually exclusive with the *{Basic Commands}* shown in Figure 7.2.6, both the *{Basic Commands}* and the command *SIMULATION* are mutually compatible with the *HELP* command, Figure 7.2.7 shows the Lean Cuisine description of the commands available in the design mode of Elc.

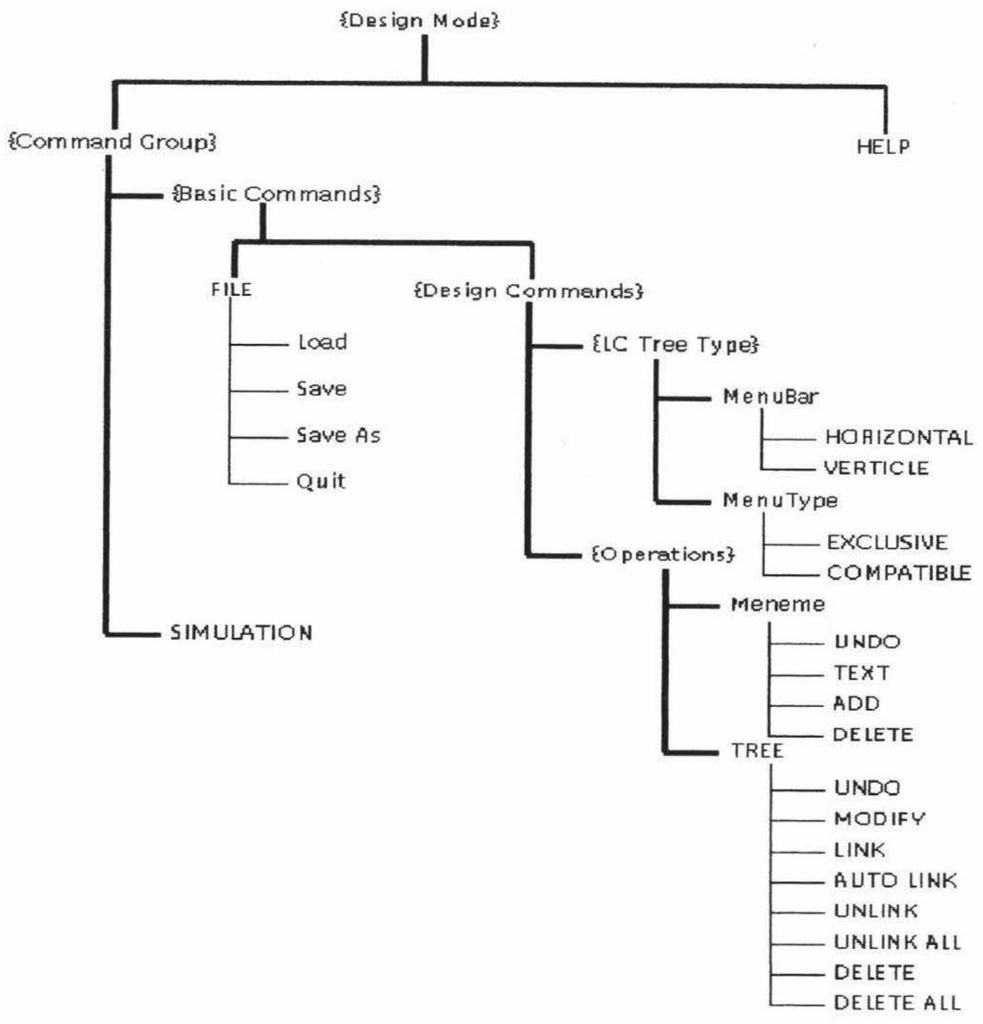


Figure 7.2.7. The LC description of the commands for the design mode of Elc.

The Simulation Mode of Elc

In the simulation mode of Elc, only the command *HELP* and the command *GO TO DESIGN* are available (Other menu buttons and command buttons are grayed.), they are mutually compatible. Figure 7.2.8 shows the menu-bar of the Elc system which is in the simulation mode and Figure 7.2.9 shows the Lean Cuisine description of the commands available in the simulation mode of Elc.

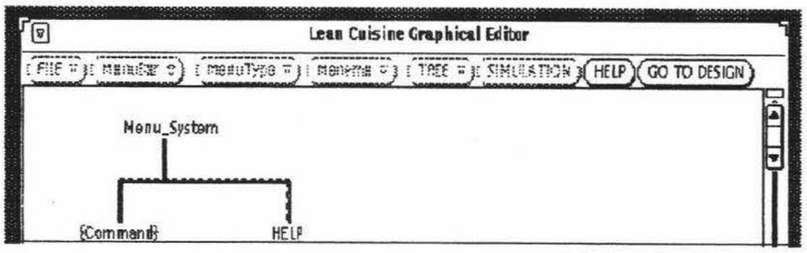


Figure 7.2.8. The menu-bar of the simulation mode of the Elc system.

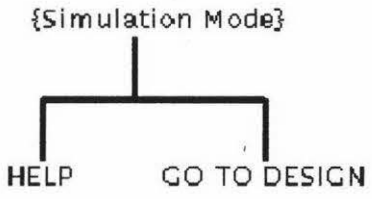


Figure 7.2.9. The LC description of the commands for the simulation mode of Elc.

Finally, a complete Lean Cuisine description of the Elc system has been created by combining above Lean Cuisine diagrams into a single diagram which is shown in Figure 7.2.10.

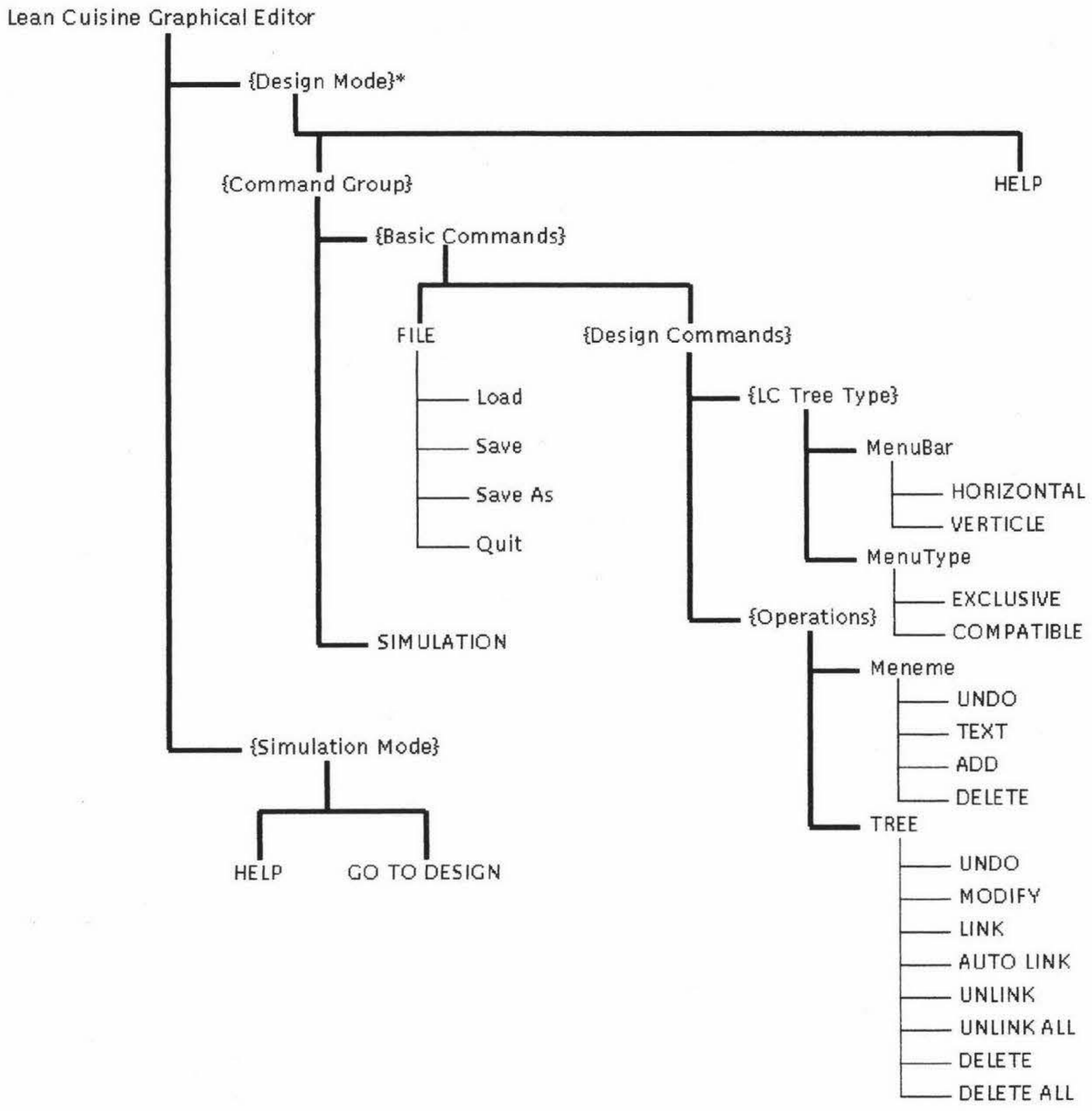


Figure 7.2.10. The complete Lean Cuisine description of the Elc system.

7.2.3. Using Elc to Create the Lean Cuisine Description of ELC

On the supposition that the ELC graphical editor will have the same functions as the Elc graphical editor has, the ELC system will use the same menu system as the Elc system, so the complete Lean Cuisine description shown in Figure 7.2.10 for the menu system of Elc can also be used as the Lean Cuisine description of the menu system of ELC.

In order to develop the ELC system in the Lean Cuisine menu interface development environment, the first step is to create its Lean Cuisine description in an Elc editing window for its menu system, the Lean Cuisine description shown in Figure 7.2.10 can be available for reference, in an Elc editing window, each Lean Cuisine tree will be interactively created according to its type and meneme number, the meneme names and meneme modifiers will be added to the menemes. In order to make a distinction between the Elc graphical editor and the ELC system, each blank character in meneme names shown in Figure 7.2.10 will be replaced by the symbol '_', for example, the meneme name '*Lean Cuisine Graphical Editor*' will be replaced by the new meneme name '*Lean_Cuisine_Graphical_Editor*'. The positions of some Lean Cuisine trees will be changed and the geometric shapes of some Lean Cuisine trees will be adjusted when linking Lean Cuisine trees to form the graphical specification of the menu system of ELC. At the different design stages, the design result will be simulated based on the Lean Cuisine graphical description created in an Elc editing window. The following paragraphs give the details of the design and the simulation of the ELC user interface at the different stages.

The Design of the Menu-bar

In a design window of the Elc system, according to the Lean Cuisine description shown in Figure 7.2.10, the Lean Cuisine description of the menu-bar of the ELC system has been created and is shown in Figure 7.2.11.

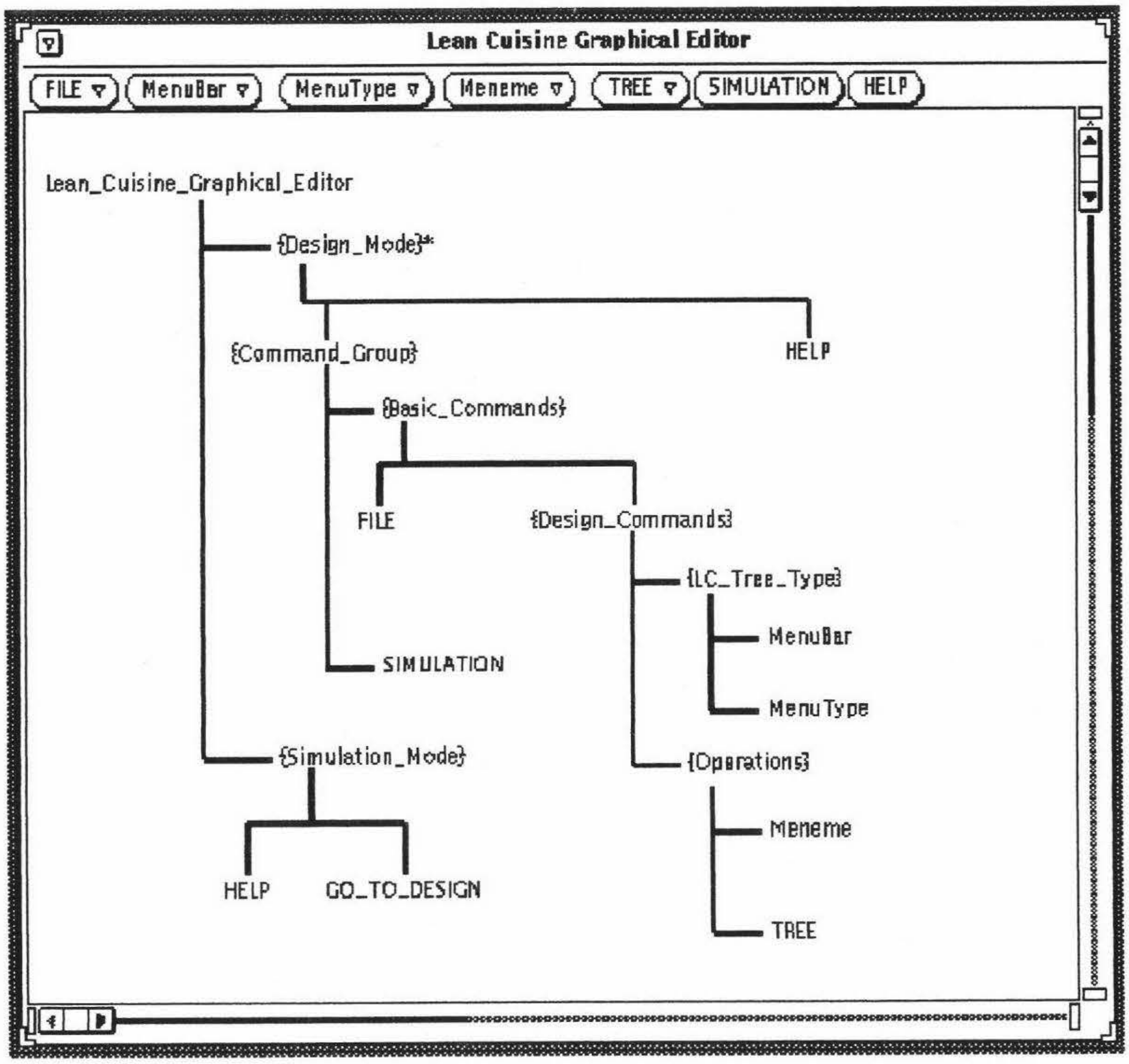
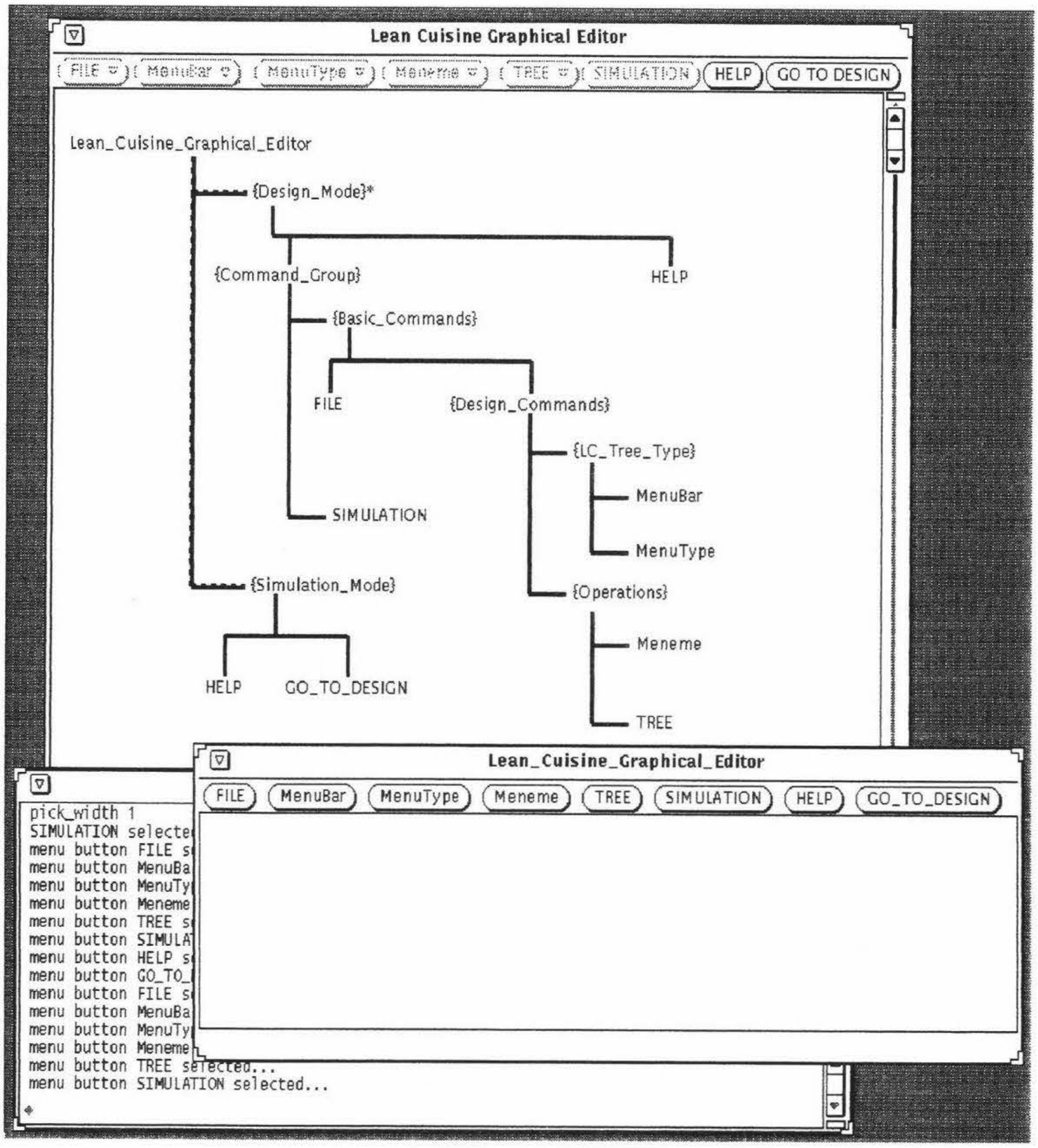


Figure 7.2.11. The LC description of the menu-bar of the ELC system.

From the above editing window of Elc, based on the current selection, the menu-bar of the ELC user interface can be partially or wholly simulated. Figure 7.2.12 shows the direct simulation of the menu-bar of the ELC menu interface from its Lean Cuisine description.



```
pick_width 1
SIMULATION selecte
menu button FILE s
menu button MenuBa
menu button MenuTy
menu button Meneme
menu button TREE s
menu button SIMULA
menu button HELP s
menu button GO_TO_
menu button FILE s
menu button MenuBa
menu button MenuTy
menu button Meneme
menu button TREE selected...
menu button SIMULATION selected...
```

Figure 7.2.12. The direct simulation of the menu-bar of the ELC menu interface from its Lean Cuisine description. **Front**: the simulation window of ELC; **back**: the Elc design window which is in the simulation mode contains the Lean Cuisine visual specification of the menu-bar of ELC; **middle**: the command window which is used to display the feedback message when testing the interface.

The Design of the Pull-down Menus

After the direct simulation of the menu-bar, the Lean Cuisine description of all pull-down menus are then created and linked to menu-bar items to form complete Lean Cuisine description of the ELC system, when linking these Lean Cuisine trees to menu-bar items, the geometric shapes of Lean Cuisine trees which represent the menu-bar have been adjusted to avoid some Lean Cuisine trees overlapping. Figure 7.2.13 shows the complete Lean Cuisine description of ELC.

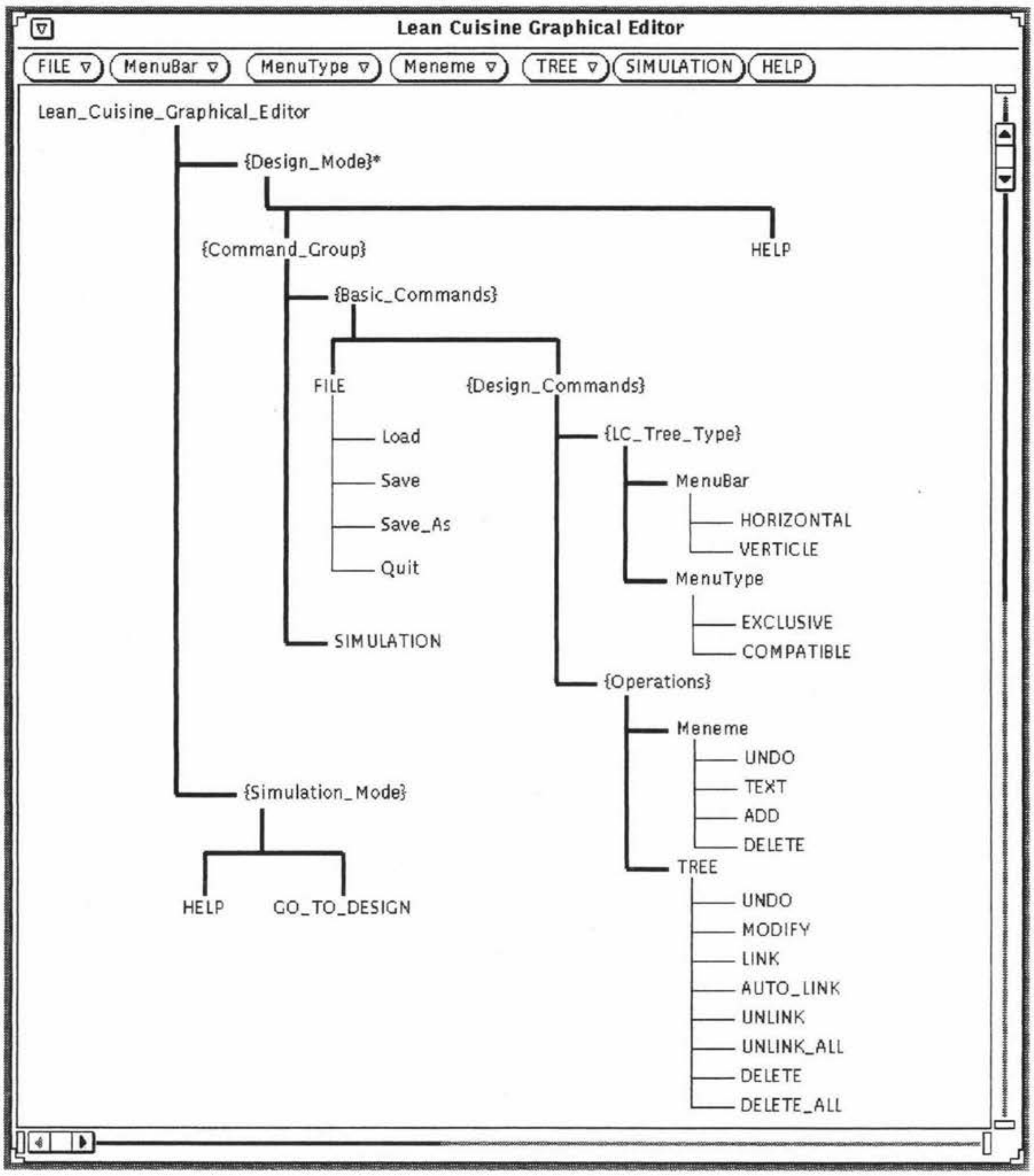
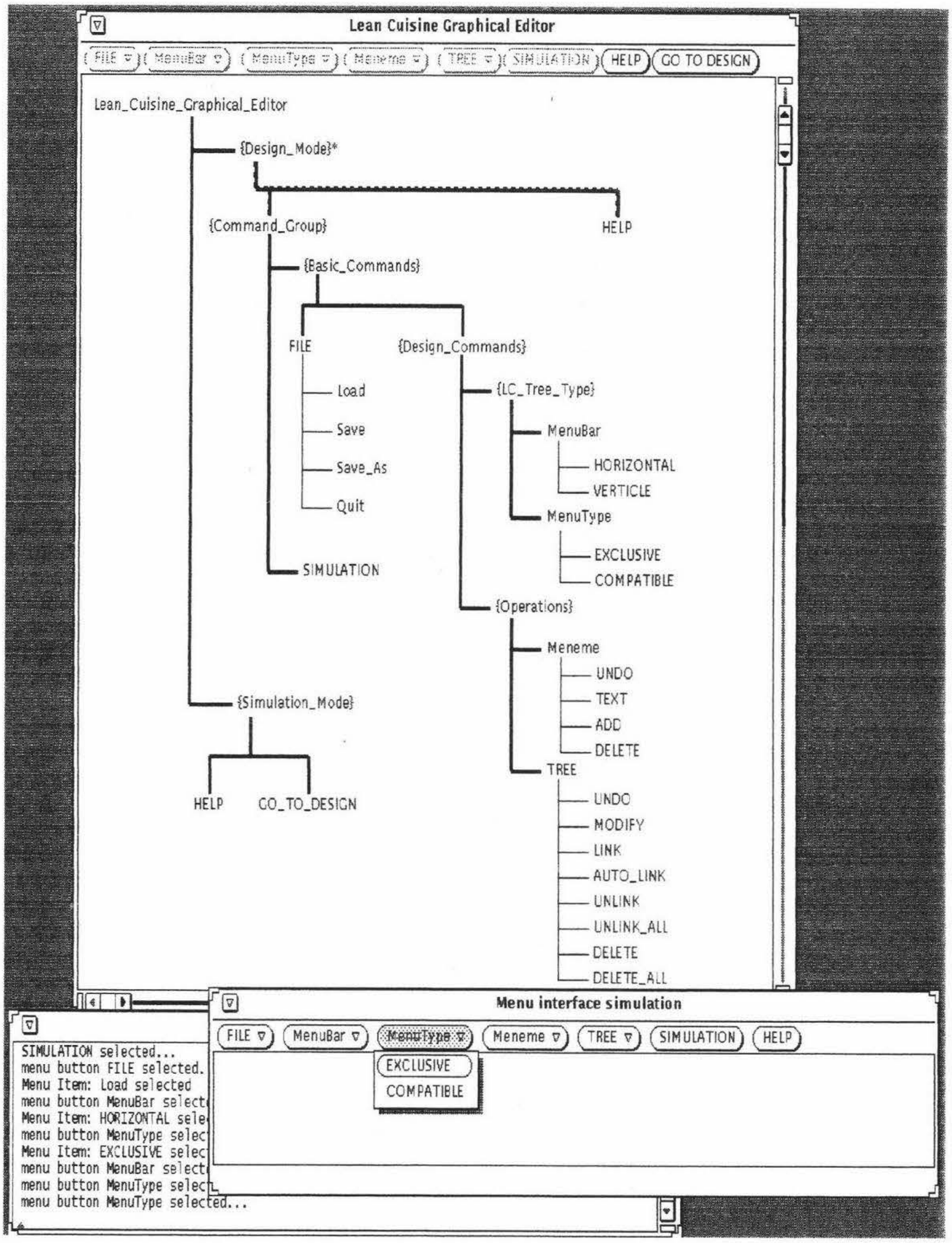


Figure 7.2.13. The complete Lean Cuisine description of ELC.

Based on the Lean Cuisine graphical specification shown in Figure 7.2.13, the ELC menu interface can be partially or wholly simulated from the editing window of Elc. Figure 7.2.14 shows the direct simulation of the design mode of the ELC menu interface from its Lean Cuisine description created in the editing window of Elc.



```
SIMULATION selected...
menu button FILE selected.
Menu Item: Load selected
menu button MenuBar select
Menu Item: HORIZONTAL sele
menu button MenuType selec
Menu Item: EXCLUSIVE selec
menu button MenuBar select
menu button MenuType selec
menu button MenuType selected...
```

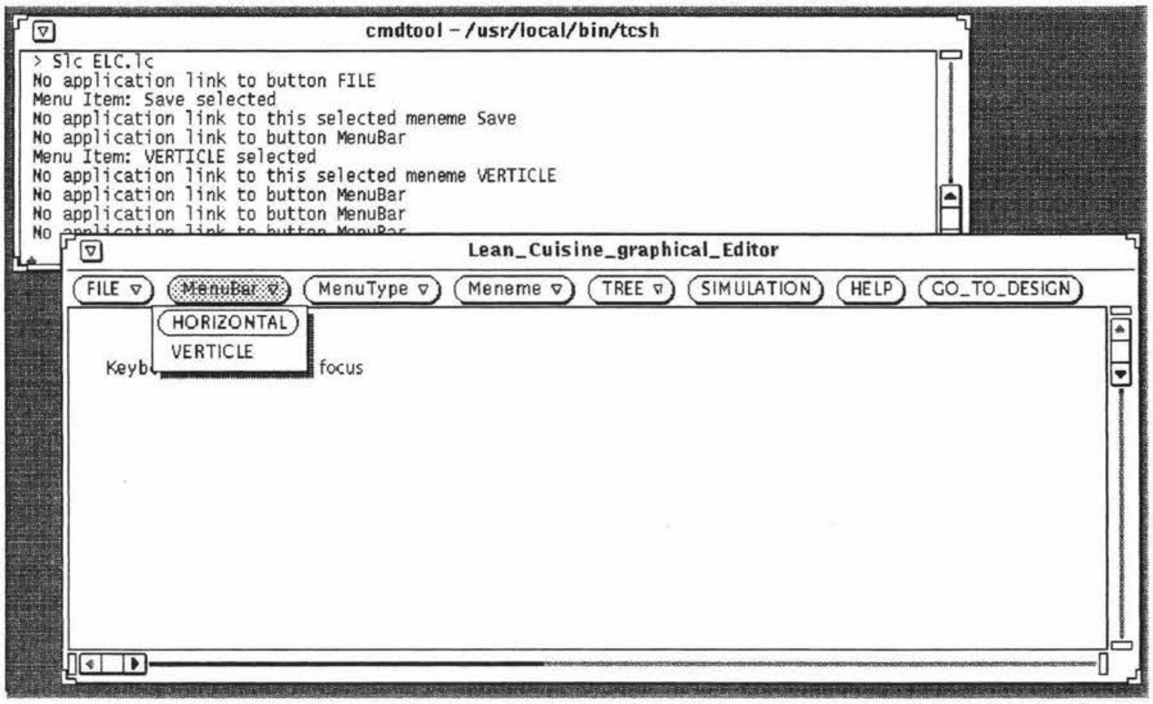
Figure 7.2.14. The direct simulation of the design mode of the ELC menu interface from its Lean Cuisine description created in an editing window of Elc. **Front**: the simulation window of ELC which is in design mode; **back**: the Elc design window which is in the simulation mode contains the Lean Cuisine visual specification of ELC; **middle**: the command window which is used to display the feedback message when testing the interface.

7.2.4. The Direct Simulation Based on the LC Graphical Specification File

The direct simulation of the ELC system in its application environment is based on its Lean Cuisine graphical specification file, ELC.lc, which was saved from the Elc editing window shown in Figure 7.2.13. To simulate the ELC user interface from its Lean Cuisine graphical specification file, in a command window, simply type the Slc command followed by the name of the Lean Cuisine graphical specification file of ELC:

>Slc ELC.lc

The simulation window of ELC is launched and can be tested. Figure 7.2.15 shows the direct simulation of the ELC user interface in its application environment.

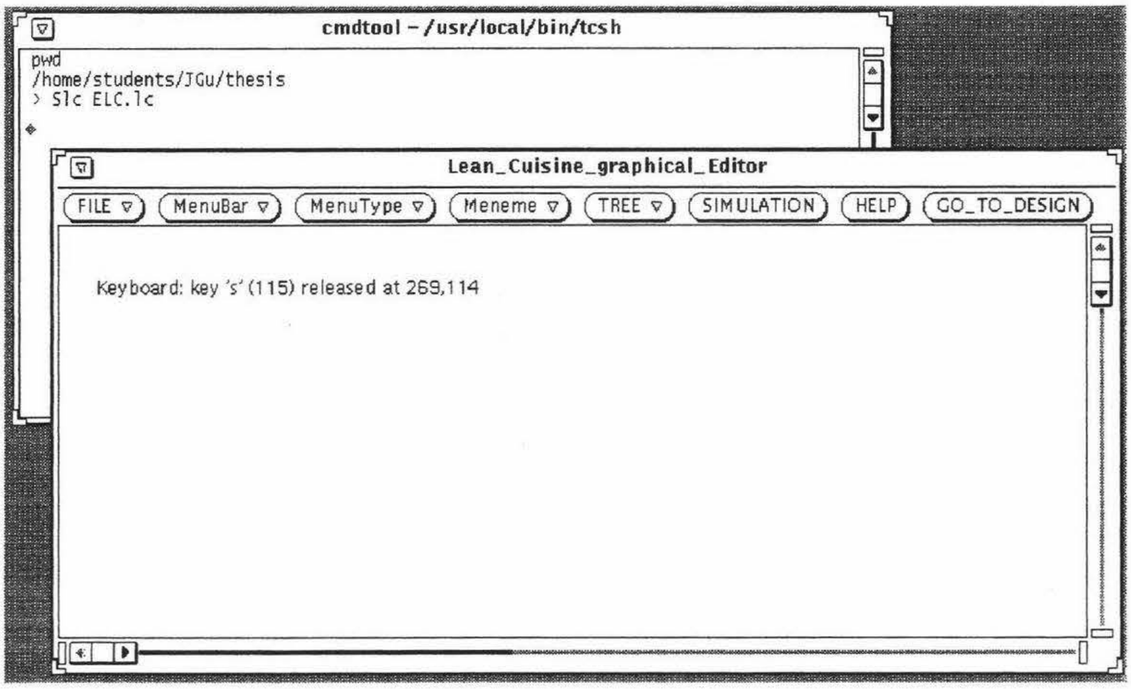


```
> Slc ELC.lc
No application link to button FILE
Menu Item: Save selected
No application link to this selected meneme Save
No application link to button MenuBar
Menu Item: VERTICLE selected
No application link to this selected meneme VERTICLE
No application link to button MenuBar
No application link to button MenuBar
```

Figure 7.2.15. The direct simulation and test of the menu system of ELC in its application environment. **Front**: the simulation window of ELC; **back**: the command window used to display the feedback message.

In the simulation window shown in Figure 7.2.15, the ELC user interface can be tested, the selected item is highlighted and the feedback message is displayed in the feedback message window.

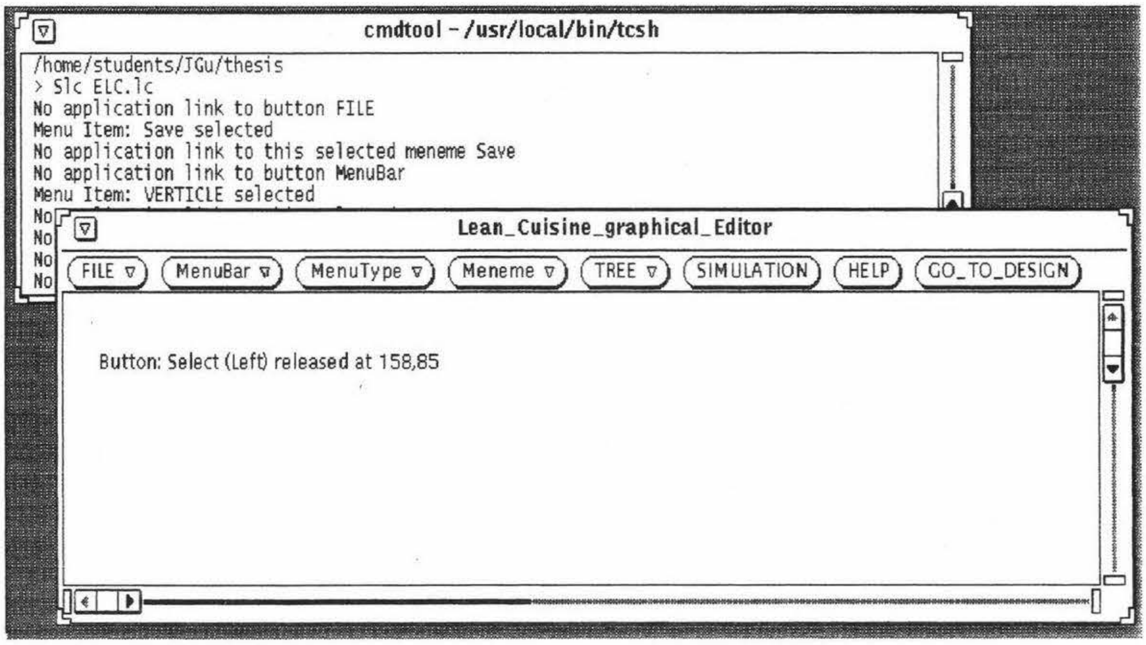
The mouse events and the keyboard events can be handled when these events take place in the simulation window of ELC, if a key is pressed or released, a button of the mouse is pressed or released, the mouse pointer moves or the mouse pointer moves while a button is held down, the message will be displayed on the simulation window of ELC. Figure 7.2.16 shows how Slc handle the mouse events and the keyboard events in the simulation window of ELC.



```
pwd
/home/students/JGu/thesis
> Slc ELC.lc
```

```
Keyboard: key 's' (115) released at 269,114
```

Figure 7.2.16a. A keyboard input in the simulation window of ELC.



```
/home/students/JGu/thesis
> Slc ELC.lc
No application link to button FILE
Menu Item: Save selected
No application link to this selected meneme Save
No application link to button MenuBar
Menu Item: VERTICLE selected
No
No
No
No
```

```
Button: Select (Left) released at 158,85
```

Figure 7.2.16b. A mouse event in the simulation window of ELC.

7.2.5. Using Glc to Generate ELC User Interface Source Code Files

Up to now the Lean Cuisine graphical specification for the ELC user interface has been created in an Elc editing window, and based on its Lean Cuisine graphical specification, the ELC user interface has been simulated in both the design environment and the application environment. The next step is to generate the ELC user interface source code files by using the Glc command followed by the name of its Lean Cuisine graphical specification file, ELC.lc. In a command window, enter the command:

```
>Glc  ELC.lc
```

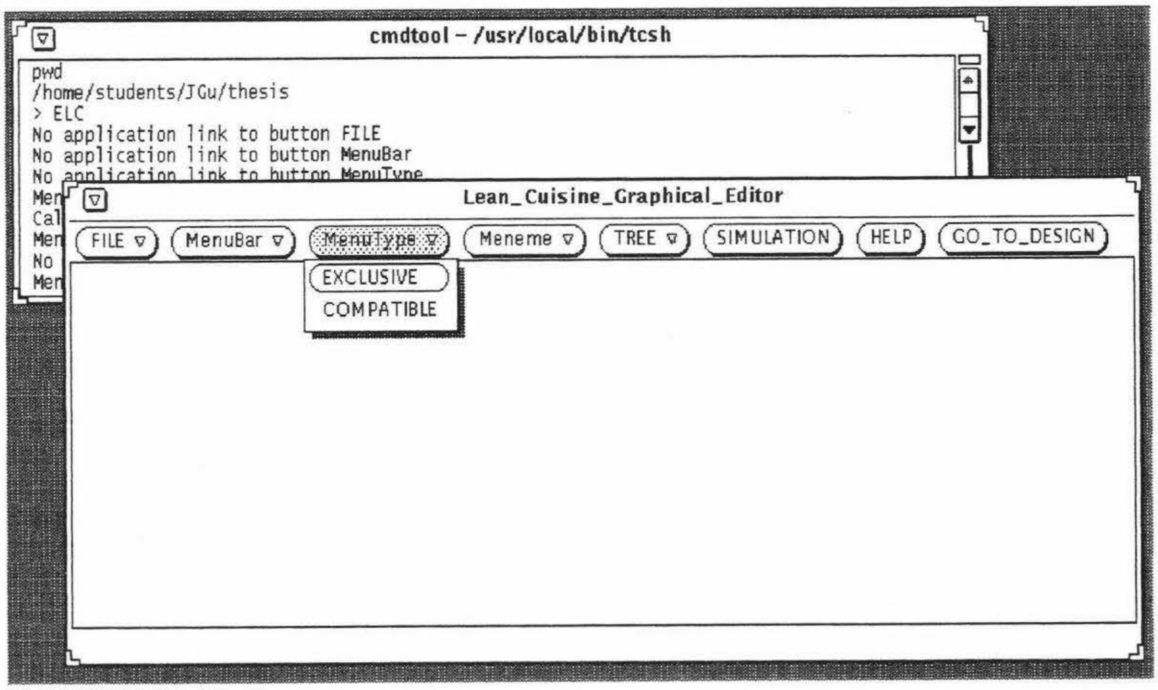
The Glc command generates four source code files and a Makefile which can be used with the make command to compile the source code files. As discussed in section 6.3, Glc names its four generated source code files after the original Lean Cuisine graphical specification filename, stripping off the .lc extension and adding new extensions to identify each file. In this example, the command "Glc ELC.lc" generates the following files:

ELC_main.c
ELC_link_app.c
ELC_app.c
ELC_app.h
Makefile

Based on above four source code files and Makefile, an executable file for the ELC user interface can be generated. To do this, simply enter command "make" in the command window. The compiler goes to work and uses the Makefile parameters for the compile, when it finishes it leaves an executable file named "ELC" according to the default filename setting in above Makefile. The real ELC user interface can now be run by entering "ELC" in the command window (the back window shown in Figure 7.2.17):

```
>ELC
```

The editing window (the front window shown in Figure 7.2.17) of ELC is launched and has the same appearance as is shown in Figure 7.2.16, but it is a real working model generated from the interface source code. Based on this real working model of ELC, the menu system of ELC can be tested, the selected item is highlighted and the feedback messages are displayed in the command window from which ELC was started, and the keyboard events and mouse events can be handled.



```
pwd
/home/students/JGu/thesis
> ELC
No application link to button FILE
No application link to button MenuBar
No application link to button MenuType
Men
Cal
Men
No
Men
```

Figure 7.2.17. The ELC user interface (real working model) generated from the interface source code. **Front**: the user interface of ELC; **back**: the command window used to run ELC real working model and display the feedback message.

7.2.6. Link Application to ELC User Interface

The ELC_link_app.c generated by the Glc tool includes the event handling procedures used to receive events and call application procedures when the events occur, it is the link socket between the ELC user interface and its application programs. The original application procedures generated by the Glc tool are included in the ELC_app.c and each of these procedures is designed to display the feedback message in the command window when they are called.

The application programs developed for the Elc graphical editor will be used in the ELC system to perform the same functions as the Elc graphical editor has. The linking procedure is to replace the original application procedures generated by the Glc tool with the correspondent real application procedures developed for the Elc graphical editor. Besides the replacement of the application routines which are invoked by the menu selections, other original application procedures, which are invoked by keyboard events, mouse motion events and mouse button events, are also replaced by the correspondent real application procedures developed for the Elc graphical editor.

The ELC system has two modes: the design mode and the simulation mode. The command button *SIMULATION* will be used to change the ELC mode from the design mode to the simulation mode and the command button *GO_TO_DESIGN* will be used to change the ELC mode from the simulation mode to the design mode. When ELC is started, the default mode is the design mode, the button *GO_TO_DESIGN* is dynamically created when the ELC system changes its mode from the design mode to the simulation mode and destroyed when the ELC system changes its mode from the simulation mode to the design mode. In simulation mode, all menu buttons and command buttons of ELC are grayed (disabled) except the *HELP* and *GO_TO_DESIGN* command buttons, so the Lean Cuisine graphical specification displayed in the ELC design window cannot be modified by the designer during the simulation. The grayed (disabled) menu buttons and command buttons are enabled and appear in black instead of gray when the design mode is reset.

The implementation of above mode changes of ELC is different from the implementation of the mode changes of TaP. When ELC is started, it invokes application routine ELCinit( ) to set the default mode of ELC, then the command button *SIMULATION* is used to invoke application routine *SIMULATION_proc( )* to set the ELC mode to the simulation mode and the command button *GO_TO_DESIGN* is used to invoke application routine *GO_TO_DESIGN_proc( )* to reset the ELC mode to the design mode. This means that the behaviour of a menu interface, which is developed in the LC menu interface development environment and controlled by lc_run_time kernel based on its Lean Cuisine description, can also be modified by the application programmers if they want.

The new executable file of the ELC application can always be generated by using the "make" command after linking some of the real application procedures to the ELC user interface. The new linked application procedures can be tested and modified based on this executable model of the ELC application.

7.2.7. ELC in Use

After having linked all the application procedures which was developed for Elc to its menu interface, the ELC graphical editor has the same functions as the Elc graphical editor has. These functions are: open an existing LC graphical specification file and display the Lean Cuisine description of a menu interface in an ELC editing window; create or modify Lean Cuisine trees and link them to form the Lean Cuisine graphical specification of a menu interface; simulate a menu interface from the LC graphical specification displayed in an ELC editing window and test the interface in simulation window; save the interface and so on. Figure 7.2.18 shows the design mode of the ELC graphical editor and its menu system is shown in Figure 7.2.19.

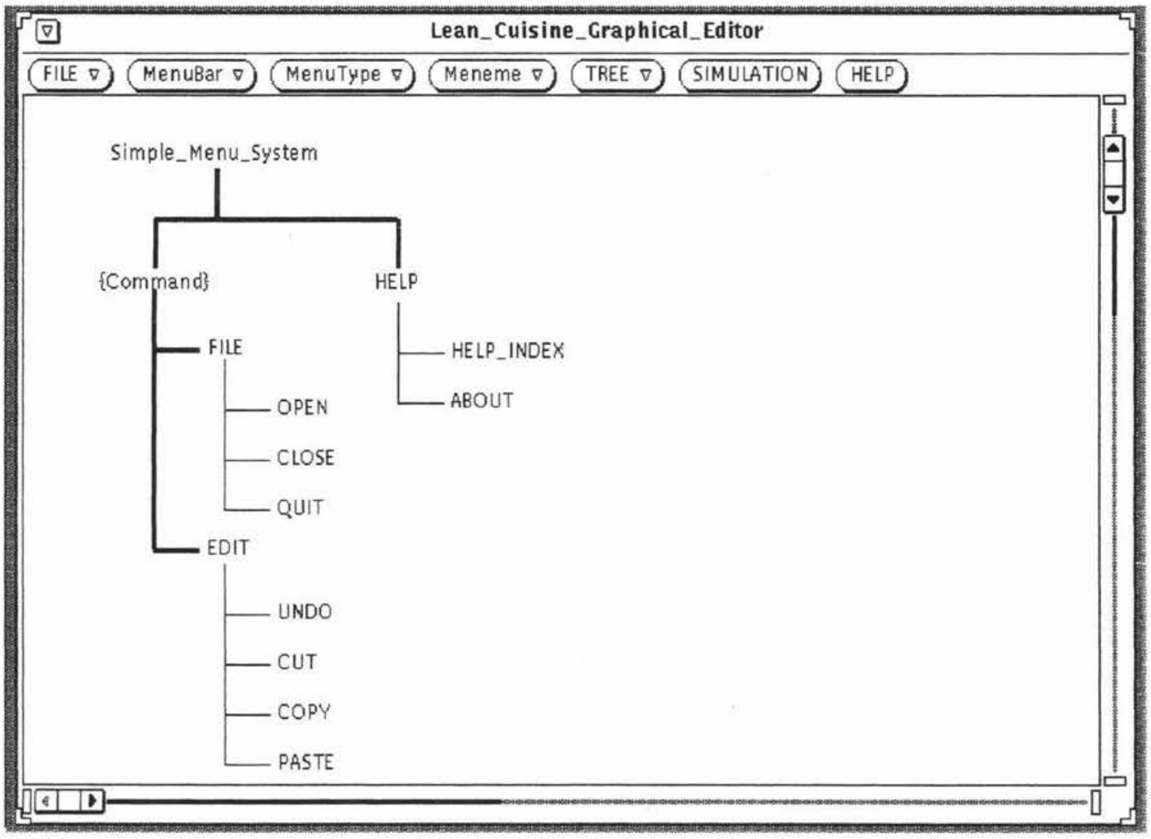


Figure 7.2.18. The design mode of the ELC graphical editor.

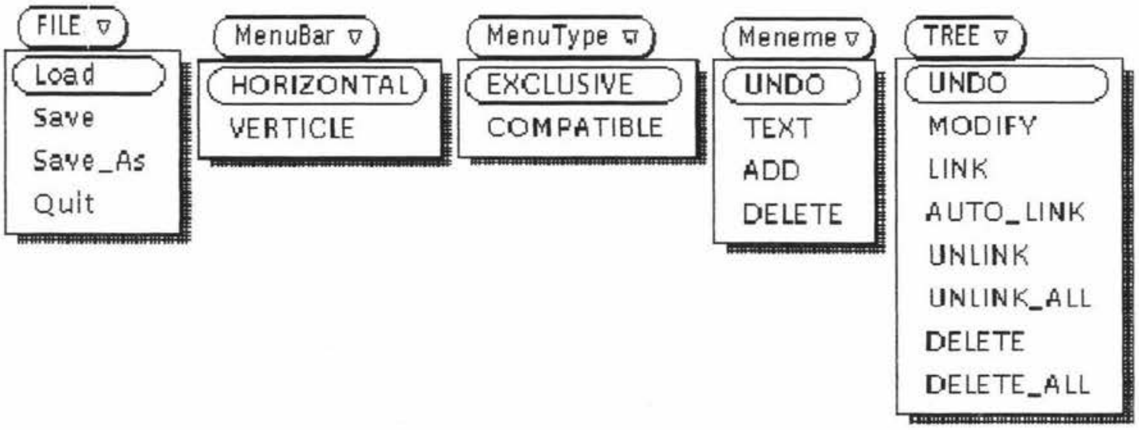


Figure 7.2.19. The menu buttons and pull-down menus in the design mode of ELC.

Figure 7.2.20 shows the simulation mode of the ELC graphical editor.

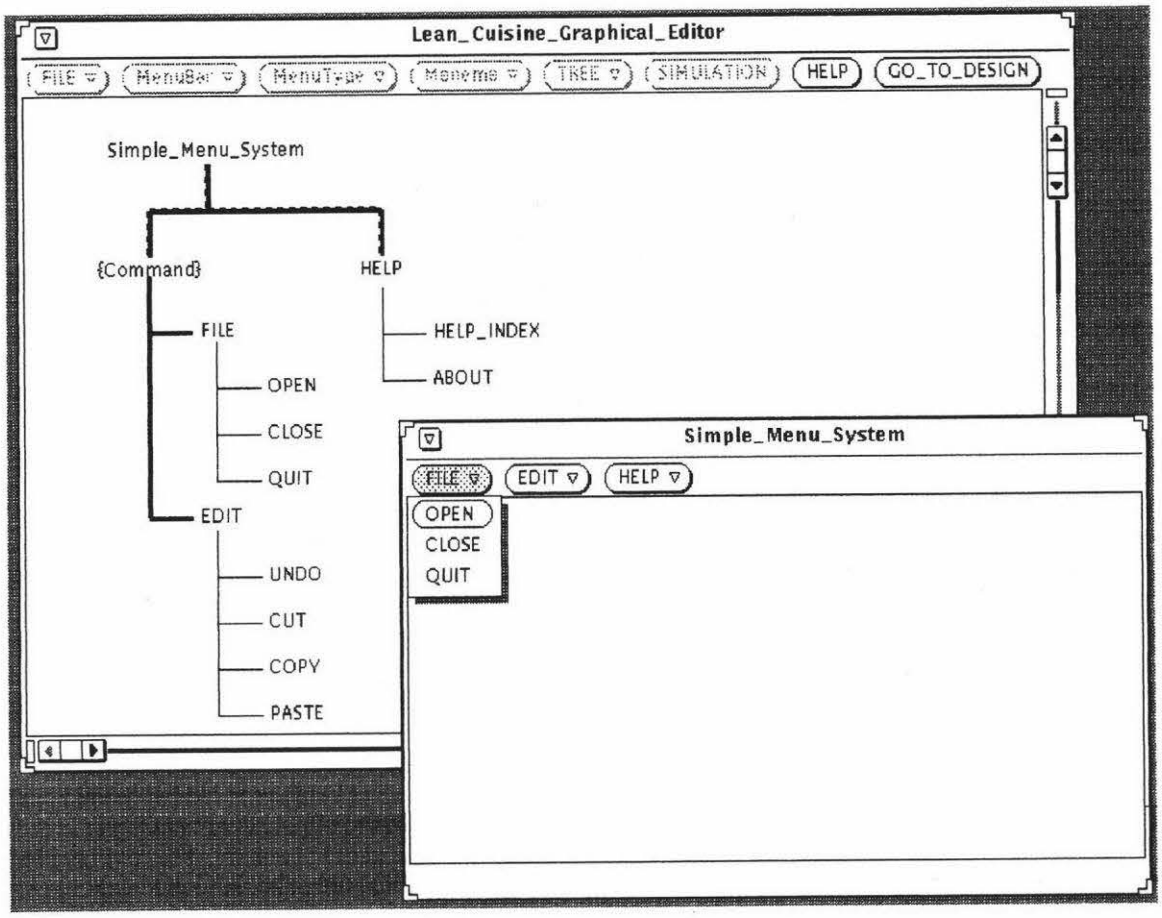


Figure 7.2.20. The simulation mode of the ELC graphical editor.

7.3. Summary

In the Lean Cuisine [Apperley & Spence, 89] menu interface development environment supported by Elc, Slc and Glc three software tools, the successful development of the TaP application and the ELC application shows that in this menu interface development environment, a menu based user interface can be rapidly simulated, modified and implemented based on its Lean Cuisine visual specification.

The Lean Cuisine menu interface development environment can cut down tremendously on the time and effort required for developing menu based user interfaces. For the TaP application, it only took 30 minutes to implement its interface using the three tools. Elc and ELC have a similar menu based interface for the same application, it took 24.5 hours to implement the interface of Elc by using programming language but in the Lean Cuisine menu interface development environment, it only took 35 minutes to implement the interface of ELC by using the three supporting tools and a speedup of 4200% was achieved.

Chapter 8

Conclusions

8.1. Review of the Integrated Interface Development Environment

The integrated user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] is a combination of software tools used to support user interface development from initial design, rapid prototyping through to direct implementation.

The Lean Cuisine Graphical Editor
The Lean Cuisine graphical editor (Elc) provides an interactive design environment for graphical specifications of menu-based interfaces and shows that the Lean Cuisine notation described in [Apperley & Spence, 89] has been implemented in a practical computer environment as an interactive interface design tool, which is independent of the user interface implementation and suitable for both non-programming and programming interface designers. The computer description of the Lean Cuisine specification of a menu-based interface in the Elc system can be directly used by Slc to simulate the user interface and by Glc to generate the user interface source code files. A special file format is used by the Elc system to store the Lean Cuisine graphical specifications of user interfaces. The Lean Cuisine graphical specification file of a user interface is a text file which contains the attributes of all Lean Cuisine trees created in a design window of Elc. Because Lean Cuisine graphical specification files store user interfaces without reference to a specific windowing environment or programming language, it is possible to use the Lean Cuisine graphical specification files of user interfaces in other windowing environment--as long as the supporting tools, Elc, Slc, Glc and the lc_run_time kernel exist in that windowing environment.

The User Interface Simulator
The user interface simulator (Slc) is a very effective and reliable interface simulating and testing tool which supports quick and convenient user interface simulation. Using Slc, a menu interface can be simulated in its design environment or in its application environment. When it is used in design environment to simulate a user interface, Slc is embedded in the Elc system to provide a user interface simulation environment where a

user interface can be directly simulated from its Lean Cuisine visual specification created in a design window of Elc. According to the designer's decision, a menu-based interface can be partially or wholly simulated from a design window of Elc and invalid menu structures can be dynamically modified during the interface simulation. When it is used in application environment to simulate a user interface, Slc is an independent simulation tool. Based on the Lean Cuisine graphical specification file which was saved from a design window of Elc, the Slc system uses the lc_run_time kernel to generate a user interface prototype and dynamically create, delete menu structures according to its Lean Cuisine specification and the user's selection. The user interface is wholly simulated based on its Lean Cuisine graphical specification file, and gives evaluators (end users, application program developers or interface designers) a real feel of how this menu-based user interface works in its application environment. Besides the simulation and test of the menu system, keyboard inputs and mouse actions can be handled by the Slc tool when it is independently used.

The User Interface Generator

The user interface generator (Glc) is used to generate basic interface source code files for a user interface from its Lean Cuisine graphical specification file and a working model of a user interface can be easily and quickly implemented without programming.

The Lc_run_time Kernel

The lc_run_time kernel is used by the Slc tool and real user interfaces to create and manage menu system based on Lean Cuisine graphical specification by direct mapping process. The basic menu system is created by the lc_run_time kernel according to the Lean Cuisine specification of the default mode of a menu system when starting Slc or a real interface. Then, when the mode of the menu system is changed based on a menu item selection, the lc_run_time kernel will dynamically delete or add menus and submenus according to the Lean Cuisine specification of the current mode of a menu system. The behaviour of a menu system can be wholly controlled by the lc_run_time kernel based on its Lean Cuisine specification. The more important function of the lc_run_time kernel is that it ensures a true separation of concerns between the application and the user interface. For example, during interface simulation, when a selected menu item corresponds to one of the terminal (leaf) menemes of Lean Cuisine trees, the lc_run_time kernel calls a pseudo-application procedure to display the feedback message in a command window; while running a real interface, in order to respond a menu item selection, it invokes an application procedure by sending an event to the link socket (_link_app.c which includes the event handling procedures used to receive events and call application procedures when the events occur.) between a user interface and its application programs.

8.2. Conclusions

The integrated user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] successfully achieves the goals listed in section 1.3, and provides software tools to support user interface development from initial design, rapid prototyping through to direct implementation. It can make menu-based user interface development much easier and quicker than other existing user interface development tools, and is suitable for both non-programming interface designers and application developers.

The integrated user interface development environment based on the Lean Cuisine graphical notation can cut down tremendously on the time and effort required for developing menu based user interfaces. The successful development of the TaP application and the ELC application in the integrated user interface development environment supported by Elc, Slc and Glc shows that in the menu interface development environment based on Lean Cuisine graphical notation, the graphical specification of a menu based user interface can be easily created and modified, and based on the Lean Cuisine graphical specification, the interface can be rapidly simulated, and direct implemented. For the TaP application, based on the original Lean Cuisine description [Apperley, 88], it took 30 minutes to design and implement Tap interface using the three tools. Elc and ELC have a similar menu based interface for the same application, it took 24.5 hours to implement the interface of Elc by using programming language but in the Lean Cuisine menu interface development environment, it only took 35 minutes to create and implement the interface of ELC by using the three supporting tools and a speedup of 4200% (a factor of 42 times faster) was achieved.

The integrated user interface development environment supports multi-platform development of user interfaces and interface portability. Many different graphical interface development environments discussed in section 3.3 are suitable for using the integrated user interface development environment to develop user interfaces based on Lean Cuisine graphical notation. With the supporting tools, Elc, Slc and Glc, and the lc_run_time kernel available in different windowing environments, it could be possible to design, modify, and simulate a user interface in a windowing environment, and then redesign, modify, and simulate the user interface in other windowing environments.

Based on the Lean Cuisine graphical specification file of a user interface, the interface source code for different windowing environments and programming languages could be easily generated by using the available code generators.

In addition to the contributions discussed above, the integrated user interface development environment based on the Lean Cuisine graphical notation [Apperley & Spence, 89] successfully integrates a graphical notation, the visual programming technique with an existing programming toolkit and offers advantages over other User Interface Programming Toolkits, language-based UIMSs and current Visual Programming Tools. It supports three main phases (design, prototyping and implementation) of the graphical user interface development lifecycle. This approach has not been found in previous user interface development tools and user interface management systems.

REFERENCES

Alty, J. and Mullin, J. (1989) "Dialogue Specification in the GRADIENT Dialogue System", People and Computers V: Proceedings of the fifth conference of the British Computer Society Human-Computer Interaction Specialist Group, edited by Sutcliffe, A. and Macaulay, L., Cambridge University Press.

Anderson, P. S. and Apperley, M. D. (1990) "An Interface Prototyping System Based on Lean Cuisine", Interacting with Computers, Vol. 2, No. 2, 217-226.

Anderson, P. S. and Apperley, M. D. (1991) Dialogue Activation: A Design Approach for Direct Manipution Interfaces, Massey Computer Science Report 91/2.

Apollo (1988) Open Dialogue Reference, Apollo Computers Inc.

Apperley, M. D. (1988) TaP: A Menu Interface Design Study Using the Lean Cuisine Notation, Information Engineering Report #88/2, Department of Electrical Engineering, Imperial College, London, UK.

Apperley, M. D. and Spence, R. (1989) "Lean Cuisine: A Low-Fat Notation for Menus", Interacting with computer, Vol. 1, No. 1, 43-68.

Apple (1985) Inside Macintosh Vol. 1-2, Addison-Wesley.

Barn, B. S. (1992) "User Interface Development: Our Experience with HP Interface Architect", CASE: Current Practice, Future Prospects, Spurr, K. and Layzell P. (Editors), John Wiley & Sons Ltd.

Bass, L. (1993) "Architectures for Interactive Software Systems: Rationale and Design", User Interface Software, edited by Bass and Dewan, John Wiley & Sons Ltd.

Bass, L. and Coutaz, J. (1991) Developing Software for the User interface, Addison-Wesley Publishing Company, Inc.

Bastide, R. and Palanque, P. (1990) "Petri Net Objects for the Design, Validation and Prototyping of User-Driven Interface", Human-Computer Interaction---INTERACT'90, Diaper, D. et al. (Editors), Elsevier Science Publishers B.V. (North-Holland).

Bischofberger, W. and Pomberger, G. (1992) Prototyping-Oriented Software Development , Springer-Verlag.

Boehm, B. W. (1988) "A Spiral Model of Software Development and Enhancement", COMPUTER, May 1988, pp 249-260.

Booch, G. (1994) Object-Oriented Analysis and Design, The Benjamin/Cummings Publishing Company, Inc.

Borland (1993) Borland C++ Programmer's Guide, Borland International, Inc.

Borland (1993) ObjectWindows for C++, Borland International, Inc.

Brown, C. M. (1988) Human-Computer Interface Design Guidelines, Ablex Publishing Corporation.

Bullinger, H. J. and Fahnrich, K. P. (1991) "User Interface Management - The Strategic View", Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals, edited by Bullinger, H. J., Elsevier Science Publishers.

Buxton, W. et al. (1983) "Toward a Comprehensive User-Interface Management System", Computer Graphics, July 1983.

Cattell, R. (1991) Object Data Management, Addison-Wesley Publishing Company.

Chang, Shi-Kuo (1987) "Visual Languages: A Tutorial and Survey", IEEE Software, January 1987, pp 29-39.

Coad, P. and Yourdon, E. (1991a) Object-Oriented Analysis, Prentice Hall, 1991.

Coad, P. and Yourdon, E. (1991b) Object-Oriented Design, Prentice Hall, 1991.

Connell, J. and Shafer, L. (1989) Structure Rapid Prototyping: An Evolutionary Approach to Software Development. Prentice-Hall.

Cockton, G. (1991) "Human Factors and Structured Software Development: the Importance of Software Structure", People and Computers VI, edited by Diaper, D. and Hammond, N., Cambridge University Press.

Coutaz, J. (1989) "UIMS: Promises, Failures and Trends", People and Computers V: Proceedings of the fifth conference of the British Computer Society Human-Computer Interaction Specialist Group, edited by Sutcliffe, A. and Macaulay, L., Cambridge University Press.

Cox, K. and Walker, D. (1990) User-Interface Design. Advanced Education Software.

Dassault (1987) CATIA: User's manual, Dassault System.

Dearnley, P. and Mayhew, P. (1984) "On the Use of Software Development Tools in the Construction of Data Processing System Prototypes", Approaches to Prototyping, edited by Budde, R., Springer-Verlag.

Dilbeck, J. and Fink, N. (1991) A Macintosh Journey : with guided projects for Microsoft Word 4, Microsoft Excel 2.2, HyperCard 1.2, FileMaker II, MacDraw II, MacPaint 2.0, Benjamin/cummings Pub. Co.

Dix, A. J. (1991) Formal Methods for Interactive Systems, Academic, 1991.

Edmonds, E. and Hagiwara, N. (1990) "An Experiment in Interactive Architectures", Human-Computer Interaction---INTERACT'90, Diaper, D. et al. (Editors), Elsevier Science Publishers B.V. (North-Holland).

England, D (1988) "Graphical Prototyping of Graphical Tools", People and Computers IV, edited by Jones, D. and Winder, R., Cambridge University Press.

England, D (1990) "MUD: Multiple-view User Interface Design", Human-Computer Interaction- INTERACT'90, 613-618.

Fairley, R. (1985) Software Engineering Concepts, McGRAW-HALL Book Company.

Foley, J., Gibbs, C., Kim, W. and Kovacevic, S. (1988) "A Knowledge-based User Interface Management System", CHI'88. 67-72.

Foley, J., Kim W. and Murray K. (1991) "UIDE: An Intelligent User Interface Design Environment", Intelligent User Interfaces, 339-384, Addison-Wesley.

Frakes, W. B., Fox, C. J. and Nejmeh, B. A. (1991) Software Engineering in the UNIX/C Environment, Prentice Hall.

Galitz, W. O. (1993) User Interface Screen Design, QED Publishing Group.

Green, M (1985) "The University of Alberta User Interface Management System", SIGGRAPH '85, San Francisco, July 22-26.

Green, M. (1986) "A Survey of Three Dialogue Models", ACM Transactions on Graphics, Vol. 5, No. 3, July 1986, pp 244-275.

Heller, D. (1990) XView Programming Manual, O'Reilly & Associates, Inc.

Henderson, D. A. Jr. (1986) "The Trillium User-Interface Design Environment", Proc. SIGCHI86, ACM, New York, 1986, pp. 221-227.

Herczeg, J., Hohl, H. and Schwab,T. (1991) "Xit-A Multi-Layered Tool for User Interface Design", Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals, edited by Bullinger, H. J., Elsevier Science Publishers B.V., 679-683.

Hill, R. D. (1986) "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction--The Sassafras UIMS", ACM Transactions on Graphics, Vol. 5, No. 3, July 1986, pp 179-210.

Hix, D. and Hartson, H. R. (1993) "Formative Evaluation: Ensuring Usability in User Interface", User Interface Software, edited by Bass and Dewan, John Wiley & Sons Ltd.

Jacob, R. (1985) "A State Transition Diagram Language for Visual Programming", IEEE Computer, 18(7), 51-59.

Jacob, R. (1986) "A Specification Language for Direct-Manipulation User Interface", ACM Transactions on Office Information System, 5(4), 283-317.

Janssen, C., Weisbecker, A. and Ziegler, J. (1993) "Generating User Interface from Data Models and Dialogue Net Specifications", INTERCHI' 93, 418-423.

Jones, P. (1991a) "Dialogue Delivery Systems", Engineering the Human-Computer Interface, edited by Downton, A., McGraw-Hill Book Company, 1991.

Jones, P. (1991b) "Dialogue Specification", Engineering the Human-Computer Interface, edited by Downton, A., McGraw-Hill Book Company, 1991.

Kanba, T. and Hashimoto, O. (1991) " U-face: A User Interface Design System Based on Multiview Model", Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals, edited by Bullinger, H. J., Elsevier Science Publishers B.V., 684-688.

Laplante, P. A. (1993) Real-time Systems Design and Analysis, IEEE Press.

Lee, G. (1993) Object-Oriented GUI Application Development, Prentice-Hall, Inc.

Lieberman, H. (1985) "There's More to Menu System Than Meets the Screen", SIGGRAPH'85, Computer Graphics, Vol. 19, No. 3, 1985, pp 181-189.

Linton, M. A. (1993) "Making User Interfaces Easy-to Build", User Interface Software, edited by Bass and Dewan, John Wiley & Sons Ltd.

Luo, P., Szekely, P. and Neches, R. (1993) "Management of Interface Design in HUMANOID", INTERCHI' 93, 107-114.

Maddix, F. (1990) Human-Computer Interaction , Ellis Horwood.

Martin, C. F. (1988) "User-Centred Requirements Analysis", Prentice Hall.

Myers, B. A. (1988) Creating User Interface by Demonstration, Academic Press, Inc.

Myers, B. A. (1989) "User-Interface Tools: Introduction and Survey", IEEE Software, January 1989, 15-24.

Myers, B. A. (1991) "Demonstrational Interface: A Step Beyond Direct Manipulation", People and Computers VI, edited by Diaper, D. and Hammond, N., Cambridge University Press.

Myers, B. A. et al. (1993) " Marquise: Creating Complete User Interfaces by Demonstration", INTERCHI' 93, 293-300.

Nelson, D. (1984) "A Software Development Environment Emphasizing Rapid Prototyping", Approaches to Prototyping, edited by Budde, R., Springer-Verlag.

Norman, K. L. (1991) The Psychology of Menu Selection, Ablex Publishing Corporation.

Nye, A. (1990) Xlib Programming Manual, O'Reilly & Associates, Inc.

Olsen, D.R. Jr. (1986) "MIKE: The Menu Interaction Kontrol Environment", ACM Trans. Graphics, 318-344.

Olsen, D.R. Jr.(1989) "A Programming Language Basis for User Interface Management", CHI'89, 171-175.

Olsen, D.R. Jr. (1992) User Interface Management Systems: Models and Algorithms, Morgan Kaufmann Publishers, Inc.

Omura, G. (1992) Mastering AutoCAD Release 12, SYBEX Inc., 1992.

Papageorgiou, P. (1991) "CIAO: Representing Graphical and Application Structural Relationships in Object Oriented User Interface", Object Oriented Approach in

Information systems, Assche, F. V., Moulin, B. and Rolland, C. (Editors). Elsevier Science Publishers B.V. (North-Holland).

Paulisch, F. N. (1993) The Design of an Extendible Graph Editor, Springer-Verlag.

Perlman, G. (1988) "Software Tools for User Interface Development", Handbook of Human-Computer Interaction, Helander, M. (ed.). Elsevier Science Publishers B.V. (North-Holland), 1988.

Peterson, M. (1992) Borland C++ Developer's Bible, Waite Group Press, Inc.

Phillips, C. H. E. (1991) "Towards a Notation for Describing the Behaviour of Direct Manipulation Interface", New Zealand Journal of Computing, 3(1), 1991, 11-25.

Phillips, C. H. E. (1992) Extending Lean Cuisine: A Case Study in Reverse Engineering, Massey University, Information Science Report 92/1.

Pugh, Ted (1993) C User Interface Library, Sigma, 1993.

Reese, J., Twiddy, R., Buchanan, L., Tarka, M. and Leung, K.C. (1985) "GUIDES: A Tool for Rapid Prototyping of User -Computer Interfaces", Proceedings of the 1985 ACM Computer Science Conference-Agenda for Computing Research: The Challenge for Creativity, 1985 March 12-14.

Roudaud, B., Lavigne, V., Lagneau, O. and Minor, E. (1990) "SCENARIOO: A New Generation UIMS", Human-Computer Interaction---INTERACT'90, Diaper, D.et al. (Editors), Elsevier Science Publishers B.V. (North-Holland).

Sage, A. P. and Palmer, J. D. (1990) Software Systems Engineering, John Wiley & Sons.

Schmucker, K.J. (1986) "MacApp: An Application Framework", Byte, Aug. 1986, pp.189-193

Shan, Yen-Ping (1990) "An Object-Oriented UIMS for Rapid Prototyping", Human-Computer Interaction---INTERACT'90, Diaper, D. et al. (Editors), Elsevier Science Publishers B.V. (North-Holland), 633-638.

Shneiderman, B. (1983) "Direct Manipulation: A Step Beyond Programming Languages", IEEE Computer, 16(8), 57-69.

Sibert, J. et al. (1986) "An Object-Oriented User-Interface Management System", Computer Graphics, Aug. 1986.

Simons, G. (1987) Introducing Software Engineering, NCC Publications.

Singh, G. and Green , M. (1989a) "A High Level User Interface Management System", CHI'89 Proceedings.

Singh, G. and Green , M. (1989b) "Generating Graphical Interface From High-Level Descriptions", Graphics Interface' 89.

Singh, G. and Green , M. (1991) "Automating the Lexical and Syntactic Design of Graphical User Interface: The UofA* UIMS", ACM Transactions on Graphics, Vol. 10, No.3, July 1991, 213-254.

Smith, M. F. (1991) Software Prototyping , McGRAW-HIll Book Company.

Sommerville, I. (1992) Software Engineering, Addison-Wesley Publishing Company.

Sukaviriya, P., Foley, J. D. and Griffith, T. (1993) "A Second Generation User Interface Design Environment: The Model and Runtime Architecture", INTERCHI' 93.

Sun (1989a) OpenLook Graphical User Interface Application Style Guidelines, Addison-Wesley Publishing Company, Inc.

Sun (1989b) OpenLook Graphical User Interface Functional Specification, Addison-Wesley Publishing Company, Inc.

Sun (1990a) OpenLook 1.0 Users guide, Sun Microsystems, Inc.

Sun (1990b) OpenWindows Developer's Guide User's Manual, Sun Microsystems, Inc.

Sutcliffe A. G. (1988) Human-Computer Interface Design, Macmillan Education Ltd.

Szekely, P., Luo, P. and Neches, R. (1993) "Beyond Interface Builders: Model-Based Interface Tools", INTERCHI' 93, 383-390.

Wasserman, A. and Shewmake, D. (1982) "Rapid Prototying of Interactive Information systems", ACM SIGSoft Software Eng. Notes, Vol. 7, No. 5, Dec. 1982.

Wasserman, A. (1985) "Extending State Transition Diagrams for the Specification of Human-Computer Interaction", IEEE Transactions on Software Engineering, 11(8).

Wasserman, A., Pircher, P., Shewmake, D. and Kersten, M. (1986) "Developing Interactive Information Systems with the User Software Engineering Methodology", IEEE Transactions on Software Engineering, SE-12(2).

Wilson, J. and Rosenbery, D. (1988) "Rapid Prototyping for User Interface Design", Handbook of Human-Computer Interaction, edited by Helander, M., 1988.

Yap, S. and Scott, M. (1990) "PENGUIN: A Language for Reactive Graphical User Interface Programming", Human-Computer Interaction---INTERACT'90, Diaper, D. et al. (Editors), Elsevier Science Publishers B.V. (North-Holland).

Young, D.A. (1989) X Window Systems Programming and Applications With Motif, Prentic Hall, 1989.

Yourdon, E. (1982) Managing the System Life Cycle: A Software Development Methodology Overview, Yourdon Press.

Yourdon, E. (1994) Object-Oriented System Design, Yourdon Press.

Appendix I

A Summary of the Lean Cuisine Notation [Apperley, 88]

1. Menu Definitions:

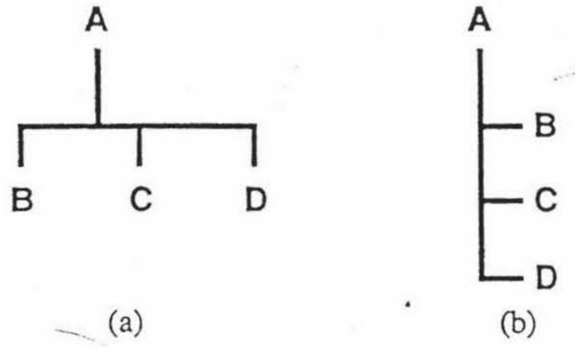
A **menu** is a set of selectable representations of actions, parameters, objects (which may be other menus), states, and other attributes, in which selections may be logically related and/or constrained.

A **meneme** is defined as an individual selectable representation (of an action, parameter, object, menu, or other attribute) within a menu, the minimum or basic unit of information in the two-way dialogue between the user and the application.

A meneme has just two possible **states**, selected and not selected, and the state may be changed either by direct excitation, or by indirect modification.

2. Menu Sub-Group Structures:

Within a menu, menemes are clustered into syntactic sub-groups that are either **mutually exclusive** (1-from-N) or **mutually compatible** (M-from-N). In the Lean Cuisine notation these structures are represented diagrammatically as follows:



(a) The tree diagram for a mutually compatible group, and
(b) that for a mutually exclusive group.

3. Lean Cuisine Definitions:

Menemes may be **terminal** (leaf) menemes, in which case they represent specific actions or parameters, or they may be **non-terminal** menemes, in which case they are themselves headers to other menus.

Virtual menemes are used to partition a single menu into constituent syntactic sub-groups. They correspond to nodes in the Lean Cuisine tree, but are not presented to, nor directly accessible to, the user.

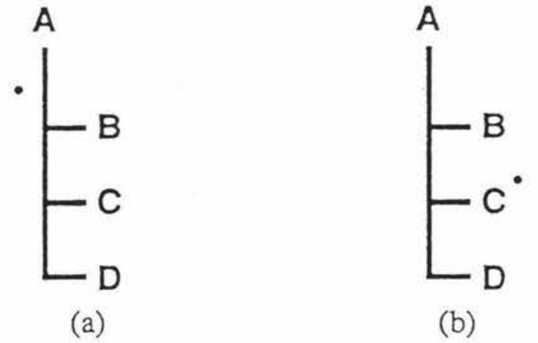
The menu corresponding to a real non-terminal meneme in a Lean Cuisine diagram consists of all of the real menemes with which the target meneme is connected by downward directed branches, either directly or via virtual menemes. If there is a sequence associated with the menemes, it is as scanned from left to right and from top to bottom.

4. Menu and Meneme Modifiers:

A menu or sub-group header may be tagged as a **required-choice** group; this places the additional constraint on the relationships between the menemes of that group, that a valid selection is always required. An initial default choice must be shown under these circumstances.

An **initial default** choice indicates a meneme that is to be initially in the selected state. It may subsequently be deselected, directly or indirectly, according to the constraints and interrelationships that apply.

A **dynamic default** is a default which takes on the value of the last user selection from that group. It may have an initial assignment, or it may be initially unassigned, in which case the first user choice from that group becomes its first value (see figure below).



(a) An unassigned dynamic default applying to the sub-group BCD, and
(b) a dynamic default applying to the same group, but initially assigned to C.

Menemes may be bistable, select-only, deselect-only, monostable or passive. The **type** is normally determined by the sub-group constraints. However, the default type may be explicitly overridden by specifying one of the other types (see special characters).

5. Special Characters:

The following special characters are used in the Lean Cuisine diagrammatic notation:

{ }	Virtual meneme; the meneme name appears between braces.
§	Required choice; normally associated with a menu or sub-group header.
*	Default choice (fixed); normally associated with a meneme.
•	Dynamic default; may be associated with a meneme (assigned) or a group (unassigned)
↑↓	Bistable meneme; used to explicitly override the default type.
↑	Select-only meneme; used to explicitly override the default type.
↓	Deselect-only meneme; used to explicitly override the default type.
⊥	Monostable meneme; used to explicitly override the default type.
⊗	Passive meneme; used to explicitly override the default type.

Appendix II Program Listing

```
/** Elc.c -- The Lean Cuisine Graphical editor **/

#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/scrollbar.h>
#include <xview/rectlist.h>
#include <xview/xv_xrect.h>
#include <xview/panel.h>
#include <xview/openmenu.h>
#include <xview/icon_load.h>
#include <xview/svrimage.h>
#include <xview/notice.h>
#include <math.h>

#define H_width 80
#define H_hight 40
#define V_width 30
#define V_hight 60
#define Tree_xo 150
#define Tree_yo 150

Frame           frame, frame_s;
Frame           subframe_1,subframe_2,subframe_f,subframe_t;
Panel           panel,subpanel,subpanel_1,subpanel_2,subpanel_f,subpanel_t;
Panel           panel_s;
Panel_itempanel_file_name_item , panel_tree_modify_item ;
Panel_itempanel_meneme_num_item, panel_meneme_text_item ;
Canvas    canvas, canvas_s;

char            msg[128];
char            Textstring [1000][50];

int     xx0,xx1;
int     yy0,yy1;
int     i, t_x, t_y;
int     sub_menubar_ptr;
int     sub_menubar_data[40][2];
int     active_submenubar_ptr;
int     active_submenubar_tree[10];
int     LC_Tree[100][8], H_and_W[100][3][20],Max_Min_Box[100][4];
int     draw_tree_ptr=-1,old_draw_tree_ptr;
int     text_ptr=-1,old_text_ptr;
int     pick_tree=0;
int     select_ptr=-100, old_select_ptr;
int     pick_hight,pick_width;
int     create_type,file_type;
int     simulating_ptr;
int     design_state=1;

void    events(), canvas_repaint_proc(), init_split(), join_split();

main(argc, argv)
int argc;
char *argv[];
{
        Xv_Window       view;
        Rect            *rect;
        Menu
        file_menu,menubar_menu,menutype_menu,meneme_operate_menu,tree_operate_menu;
        int             file_selected();
        int             menubar_selected();
        int             menutype_selected();
        int             meneme_operate_menu_selected();
        int             tree_operate_selected();
        int             simulation_selected();
        int             go_to_design_selected();
        int             help_selected();
        int             inputed();
        int             treat_file_by_name();
        int             create_tree_by_menume_number();
        int             add_text_to_selected_menume();
        int             modify_tree_by_scale();
        int             which_line_selected();
        int             Draw_Verticle_tree();
        int             Draw_Horizontal_tree();
        int             Create_Verticle_tree();
        int             Create_Horizontal_tree();
        int             link_subtree_to_Horizontal_tree_branch();
        int             link_subtree_to_Verticle_tree_branch();
        int             text_rect_width();

        extern double           sin(/* double x */);
        extern double           cos(/* double x */);

        void            menubar_menu_proc();
        void            file_menu_proc();
        void            menutype_menu_proc();
        void            meneme_operate_menu_proc();
        void            tree_operate_proc();

xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

/* Create a frame that's 600 wide by 800 high -- give it a titlebar */

frame = (Frame)xv_create(XV_NULL, FRAME,
        XV_WIDTH,       800,
        XV_HEIGHT,      600,
        /*FRAME_LABEL,      argv[0],*/
        FRAME_LABEL,        "Lean Cuisine Graphical Editor",
        NULL);

        panel = (Panel)xv_create(frame, PANEL,
                PANEL_LAYOUT, PANEL_HORIZONTAL,
                XV_X, 0,
                XV_Y, 0,
                /*XV_WIDTH, 400,*/
                XV_WIDTH, WIN_EXTEND_TO_EDGE,
                XV_HEIGHT, 25,
                WIN_BORDER, FALSE,
                NULL);


subframe_f = (Frame)xv_create(frame, FRAME_CMD,
                XV_WIDTH,       400,
                XV_HEIGHT,      65,
                FRAME_LABEL,    "Input_Popup_Window",
                FRAME_SHOW_FOOTER,    TRUE,
                NULL);

subpanel_f = (Panel)xv_get(subframe_f, FRAME_CMD_PANEL ,
                PANEL_LAYOUT,       PANEL_VERTICAL,
                NULL);

panel_file_name_item = (Panel_item)xv_create(subpanel_f, PANEL_TEXT,
                XV_X, 24,
                XV_Y, 30,
                PANEL_LABEL_STRING,    "File Name:",
                PANEL_LAYOUT,       PANEL_HORIZONTAL,
                PANEL_VALUE_DISPLAY_LENGTH, 20,
                PANEL_NOTIFY_PROC,     treat_file_by_name,
                NULL);

subframe_t = (Frame)xv_create(frame, FRAME_CMD,
                XV_WIDTH,       400,
                XV_HEIGHT,      65,
                FRAME_LABEL,    "Input_Popup_Window",
                FRAME_SHOW_FOOTER,    TRUE,
                NULL);

subpanel_t = (Panel)xv_get(subframe_t, FRAME_CMD_PANEL ,
                PANEL_LAYOUT,       PANEL_VERTICAL,
                NULL);

panel_tree_modify_item = (Panel_item)xv_create(subpanel_t, PANEL_TEXT,
                XV_X, 24,
                XV_Y, 30,
                PANEL_LABEL_STRING,    "Modify scale:",
                PANEL_LAYOUT,       PANEL_HORIZONTAL,
                PANEL_VALUE_DISPLAY_LENGTH, 20,
                PANEL_NOTIFY_PROC,     modify_tree_by_scale,
                NULL);

subframe_1 = (Frame)xv_create(frame, FRAME_CMD,
                XV_WIDTH,       400,
                XV_HEIGHT,      65,
                FRAME_LABEL,    "Input_Popup_Window",
                FRAME_SHOW_FOOTER,    TRUE,
                NULL);

subpanel_1 = (Panel)xv_get(subframe_1, FRAME_CMD_PANEL ,
                PANEL_LAYOUT,       PANEL_VERTICAL,
                NULL);

panel_meneme_num_item = (Panel_item)xv_create(subpanel_1, PANEL_TEXT,
                XV_X, 24,
                XV_Y, 30,
                PANEL_LABEL_STRING,    "Input Menume Number:",
                PANEL_LAYOUT,       PANEL_HORIZONTAL,
                PANEL_VALUE_DISPLAY_LENGTH, 20,
                PANEL_NOTIFY_PROC,     create_tree_by_menume_number,
                NULL);

subframe_2 = (Frame)xv_create(frame, FRAME_CMD,
                XV_WIDTH,       400,
                XV_HEIGHT,      65,
                FRAME_LABEL,    "Text_Input_Window",
                FRAME_SHOW_FOOTER,    TRUE,
                    NULL);

subpanel_2 = (Panel)xv_get(subframe_2, FRAME_CMD_PANEL ,
                    PANEL_LAYOUT,         PANEL_VERTICAL,
                    NULL);

panel_meneme_text_item = (Panel_item)xv_create(subpanel_2, PANEL_TEXT,
                    XV_X, 24,
                    XV_Y, 30,
                    PANEL_LABEL_STRING,   "Input Meneme Text:",
                    PANEL_LAYOUT,         PANEL_HORIZONTAL,
                    PANEL_VALUE_DISPLAY_LENGTH, 20,
                    PANEL_NOTIFY_PROC,    add_text_to_selected_menume,
                    NULL);

/* (void) xv_create(subpanel, PANEL_BUTTON,
    PANEL_LABEL_STRING, "Input OK",
    PANEL_NOTIFY_PROC,  inputed,
    NULL);*/

/* Create the menu _before_ the panel button */
file_menu = (Menu)xv_create(NULL, MENU,
                    MENU_NOTIFY_PROC,     file_menu_proc,
                    MENU_STRINGS,         "Load", "Save", "Save As","Quit", NULL,
                    NULL);

(void) xv_create(panel, PANEL_BUTTON,
                    XV_X, 4,
                    XV_Y, 4,
                    XV_WIDTH, 10,
                    XV_HEIGHT, 10,
                    PANEL_LABEL_STRING,   "FILE",
                    PANEL_NOTIFY_PROC,    file_selected,
                    PANEL_ITEM_MENU,      file_menu, /* attach menu to button */
                    NULL);

/* Create the menu _before_ the panel button */
menubar_menu = (Menu)xv_create(NULL, MENU,
                    MENU_NOTIFY_PROC,     menubar_menu_proc,
                    MENU_STRINGS,         "HORIZONTAL", "VERTICLE",NULL,
                    NULL);

(void) xv_create(panel, PANEL_BUTTON,
                    XV_X, 64,
                    XV_Y, 4,
                    XV_WIDTH, 10,
                    XV_HEIGHT, 10,
                    PANEL_LABEL_STRING,   "MenuBar",
                    PANEL_NOTIFY_PROC,    menubar_selected,
                    PANEL_ITEM_MENU,      menubar_menu, /* attach menu to button */
                    NULL);

/* Create the menu _before_ the panel button */
menutype_menu = (Menu)xv_create(NULL, MENU,
                    MENU_NOTIFY_PROC,     menutype_menu_proc,
                    MENU_STRINGS,         "CHOICE", "EXCLUSIVE", "COMPATIBLE",NULL,
                    NULL);

(void) xv_create(panel, PANEL_BUTTON,
                    XV_X, 160,
                    XV_Y, 4,
                    XV_WIDTH, 10,
                    XV_HEIGHT, 10,
                    PANEL_LABEL_STRING,   "MenuType",
                    PANEL_NOTIFY_PROC,    menutype_selected,
                    PANEL_ITEM_MENU,      menutype_menu, /* attach menu to button */
                    NULL);



/* Create the menu _before_ the panel button */
meneme_operate_menu = (Menu)xv_create(NULL, MENU,
                    MENU_NOTIFY_PROC,     meneme_operate_menu_proc,
                    MENU_STRINGS,         "UNDO","TEXT", "ADD", "DELETE",NULL,
                    NULL);

(void) xv_create(panel, PANEL_BUTTON,
                    XV_X, 262,
                    XV_Y, 4,
                    XV_WIDTH, 10,
                    XV_HEIGHT, 10,
                    PANEL_LABEL_STRING,   "Meneme",
                    PANEL_NOTIFY_PROC,    meneme_operate_menu_selected,
                    PANEL_ITEM_MENU,      meneme_operate_menu, /* attach menu to button */
                    NULL);

/* Create the menu _before_ the panel button */
tree_operate_menu = (Menu)xv_create(NULL, MENU,
MENU_NOTIFY_PROC,    tree_operate_proc,
                    MENU_STRINGS,         "UNDO","MODIFY", "LINK","AUTO
LINK","UNLINK","UNLINK
```

```
                                        ALL","DELETE","DELETE ALL",NULL,
                NULL);

(void) xv_create(panel, PANEL_BUTTON,
                XV_X, 356,
                XV_Y, 4,
                XV_WIDTH, 10,
                XV_HEIGHT, 10,
                PANEL_LABEL_STRING,    "TREE",
                PANEL_NOTIFY_PROC,    tree_operate_selected,
                PANEL_ITEM_MENU,      tree_operate_menu, /* attach menu to button */
                NULL);

(void) xv_create(panel, PANEL_BUTTON,
                XV_X, 420,
                XV_Y, 4,
                XV_WIDTH, 10,
                XV_HEIGHT, 10,
                PANEL_LABEL_STRING,    "SIMULATION",
                PANEL_NOTIFY_PROC,     simulation_selected,
                XV_SHOW,               TRUE,
                NULL);

(void) xv_create(panel, PANEL_BUTTON,
                XV_X, 516,
                XV_Y, 4,
                XV_WIDTH, 10,
                XV_HEIGHT, 10,
                PANEL_LABEL_STRING,    "HELP",
                PANEL_NOTIFY_PROC,    help_selected,
                NULL);

(void) xv_create(panel, PANEL_BUTTON,
                XV_X, 566,
                XV_Y, 4,
                PANEL_LABEL_STRING,    "GO TO DESIGN",
                PANEL_NOTIFY_PROC,    go_to_design_selected,
                XV_SHOW,              FALSE,
                NULL);

canvas = (Canvas)xv_create(frame, CANVAS,
                XV_X, 0,
                XV_Y, 25,
                CANVAS_WIDTH,        1600,
                CANVAS_HEIGHT,       1500,
                CANVAS_AUTO_SHRINK,    FALSE,
                CANVAS_AUTO_EXPAND,    FALSE,
                /* CANVAS_REPAINT_PROC,    repaint_proc,*/
                CANVAS_REPAINT_PROC,    canvas_repaint_proc,
                CANVAS_X_PAINT_WINDOW, TRUE,
                NULL);

/* Install the callback for events on the first (and only, so far)
paint window.  We'll use the default events provided by the canvas.
*/
/*  xv_set(canvas_paint_window(canvas),
WIN_EVENT_PROC,         events,
WIN_CONSUME_EVENTS,     ACTION_SELECT, ACTION_ADJUST, NULL,
NULL);*/

xv_set(canvas_paint_window(canvas),
                WIN_CONSUME_EVENTS,
                WIN_NO_EVENTS,
                WIN_ASCII_EVENTS, KBD_USE, KBD_DONE,
                LOC_DRAG, /*LOC_WINENTER, LOC_WINEXIT,*/ WIN_MOUSE_BUTTONS,
                NULL,
                WIN_EVENT_PROC, events,
                NULL);

/*
* There's only one viewport since multi-views cannot be created
* when creating a canvas.  Install "init" and "destroy" callbacks
* in the canvas object.  See the corresponding routines for specifics.
*/
xv_set(canvas,
                OPENWIN_SPLIT,
                OPENWIN_SPLIT_INIT_PROC,    init_split,
                OPENWIN_SPLIT_DESTROY_PROC, join_split,
                NULL,
                NULL);

/*
* Attach scrollbars to the canvas.
*/
xv_create(canvas, SCROLLBAR,
                SCROLLBAR_SPLITTABLE,   TRUE,
                SCROLLBAR_DIRECTION,    SCROLLBAR_VERTICAL,
                NULL);

xv_create(canvas, SCROLLBAR,
                    SCROLLBAR_SPLITTABLE,  TRUE,
                    SCROLLBAR_DIRECTION,   SCROLLBAR_HORIZONTAL,
                    NULL);

frame_s = (Frame)xv_create(frame, FRAME,
                    XV_WIDTH,          800,
                    XV_HEIGHT,         450,
                    FRAME_LABEL,          argv[1]? argv[1] : "Menu interface simulation",
                    FRAME_SHOW_FOOTER,    TRUE,
                    NULL);

panel_s = (Panel)xv_create(frame_s, PANEL,
                    PANEL_LAYOUT, PANEL_HORIZONTAL,
                    XV_X, 0,
                    XV_Y, 0,
                    /*XV_WIDTH, 400,*/
                    XV_WIDTH, WIN_EXTEND_TO_EDGE,
                    XV_HEIGHT, 25,
                    WIN_BORDER, FALSE,
                    NULL);

canvas_s = (Canvas)xv_create(frame_s, CANVAS,
                    FRAME_LABEL,   argv[0],
                    XV_WIDTH,      1600,
                    XV_HEIGHT,     800,
                    NULL);


/* window_fit(panel); window_fit(frame);*/

xv_main_loop(frame);
exit(0);
}



void
canvas_repaint_proc(canvas, paint_window, dpy, xwin, xrects)
Canvas        canvas;        /* unused */
Xv_Window     paint_window;  /* unused */
Display       *dpy;
Window        xwin;
Xv_xrectlist *xrects;        /* unused */
{
 GC gc;
 int j;

gc = DefaultGC(dpy, DefaultScreen(dpy));
if(design_state==1)
{
 XClearWindow(dpy, xwin);
 for(j=0;j<=draw_tree_ptr;j++)

 {
  if(LC_Tree[j][1]<=2)
      {
      if(LC_Tree[j][1]==0)
          XSetLineAttributes(dpy,gc,3,LineSolid,CapButt,JoinMiter);
      else
          XSetLineAttributes(dpy,gc,1,LineSolid,CapButt,JoinMiter);
      Draw_Verticle_tree(j,dpy,xwin,gc);
      }
  else
      {
      if(LC_Tree[j][1]==3)
          XSetLineAttributes(dpy,gc,3,LineSolid,CapButt,JoinMiter);
      else
          XSetLineAttributes(dpy,gc,1,LineSolid,CapButt,JoinMiter);
      Draw_Horizontal_tree(j,dpy,xwin,gc);
      }
}

if(select_ptr>=0)
  {
  j=select_ptr;
  if(LC_Tree[j][1]<=2)
      {
      if(LC_Tree[j][1]==0)
          XSetLineAttributes(dpy,gc,4,LineOnOffDash,CapButt,JoinMiter);
      else
          XSetLineAttributes(dpy,gc,2,LineOnOffDash,CapButt,JoinMiter);
      Draw_Verticle_tree(j,dpy,xwin,gc);
      }
   else
      {
      if(LC_Tree[j][1]==3)
          XSetLineAttributes(dpy,gc,4,LineOnOffDash,CapButt,JoinMiter);
      else
          XSetLineAttributes(dpy,gc,2,LineOnOffDash,CapButt,JoinMiter);
      Draw_Horizontal_tree(j,dpy,xwin,gc);
```

```
          }

    }
  }
   /* XDrawString(dpy, xwin, gc, 25, 45, msg, strlen(msg));*/
}



void
events(pw, event)
Xv_Window pw;
Event *event;
{
int j,k, result;
 register char *p = msg;

   int code = event_action(event);
   Xv_Window view;
   int i = (int)xv_get(canvas, OPENWIN_NVIEWS);

   *p = 0;
   /* Not interested in button up events */
  /* if (win_inputnegevent(event))
      return;*/


   /* test to see if a function key has been hit */
   if (event_is_key_left(event))
      sprintf(p, "(L%d) ", event_id(event) - KEY_LEFTFIRST + 1);
   else if (event_is_key_top(event))
      sprintf(p, "(T%d) ", event_id(event) - KEY_TOPFIRST + 1);
   else if (event_is_key_right(event))
      sprintf(p, "(R%d) ", event_id(event) - KEY_RIGHTFIRST + 1);
   else if (event_id(event) == KEY_BOTTOMLEFT)
      strcpy(p, "bottom left ");
   else if (event_id(event) == KEY_BOTTOMRIGHT)
      strcpy(p, "bottom right ");
   p += strlen(p);



   /* Determine which paint window this event happened in. */
   while (pw != xv_get(canvas, CANVAS_NTH_PAINT_WINDOW, --i) && i > 0)
      ;
```

```
  /* The paint window number is "i" -- get the "i"th view window */
  view = xv_get(canvas, OPENWIN_NTH_VIEW, i);


 if (event_is_ascii(event))
    {
    /*
     * note that shift modifier is reflected in the event code by
     * virtue of the char printed is upper/lower case.
     */
    sprintf(p, "Keyboard: key '%c' (%d) %s at %d,%d",
       event_action(event), event_action(event),
       event_is_down(event)? "pressed" : "released",
       event_x(event), event_y(event));
  } else switch (event_action(event)) {
    case ACTION_CLOSE :
       xv_set(frame, FRAME_CLOSED, TRUE, NULL);
       break;
    case ACTION_OPEN :
       strcpy(p, "frame opened up");
       break;
    case ACTION_HELP :
       strcpy(p, "Help (action ignored)");
       break;
    case ACTION_SELECT :



 xx1=event_x(event);
 yy1=event_y(event);

 pick_tree=0;

 for(j=0;j<=draw_tree_ptr;j++)
    {
    if(LC_Tree[j][1]<=2)
      {
        if(xx1>LC_Tree[j][2]+Max_Min_Box[j][0]-5 && xx1<LC_Tree[j][2]+Max_Min_Box[j][1]
               && yy1>LC_Tree[j][3]+ Max_Min_Box[j][2]&& yy1<LC_Tree[j][3]+
Max_Min_Box[j][3]+5)
           {
           pick_tree=1;
           select_ptr=j;
           which_line_selected(j,xx1,yy1);
           break;
           }
        }
    else
        {
            if(xx1>LC_Tree[j][2]+Max_Min_Box[j][0]-5 && xx1<LC_Tree[j][2]+Max_Min_Box[j][1]+5
                    && yy1>LC_Tree[j][3]+ Max_Min_Box[j][2]&& yy1<LC_Tree[j][3]+
Max_Min_Box[j][3])
            {
            pick_tree=1;
            select_ptr=j;
            which_line_selected(j,xx1,yy1);
            break;
            }

        }
    }
 if(pick_tree==0)
      select_ptr =-100;


      sprintf(p, "Button: Select (Left) %s at %d,%d",
         event_is_down(event)? "pressed" : "released",
         event_x(event), event_y(event));
      break;
    case ACTION_ADJUST :
      sprintf(p, "Button: Adjust (Middle) %s at %d,%d",
         event_is_down(event)? "pressed" : "released",
         event_x(event), event_y(event));
      break;
    case ACTION_MENU :

 if(event_is_down(event))

 {

 xx1=event_x(event);
 yy1=event_y(event);

 if(pick_tree==1)
  {
    k=0;
    for(j=0;j<LC_Tree[select_ptr][5];j++)
       {
        if(H_and_W[select_ptr][2][j]>=0)
           {
            k=1;
            break;
           }
     }

    if(k==1&& LC_Tree[select_ptr][6]>=0)
    {
         result = notice_prompt(panel, NULL,
         NOTICE_FOCUS_XY,     t_x, t_y,
NOTICE_MESSAGE_STRINGS, "can not move this tree because it has parent tree and subtree(s)?", NULL,
         NOTICE_BUTTON_YES,    "dislink parent tree,subtree(s)and move it",
         NOTICE_BUTTON_NO,     "cancel",
         NULL);

         if (result == NOTICE_YES)
          {
           for(j=0;j<LC_Tree[select_ptr][5];j++)
               {
                if(H_and_W[select_ptr][2][j]>=0)
                    {
                     LC_Tree[H_and_W[select_ptr][2][j]][6]=-1000;
                     H_and_W[select_ptr][2][j]=-1000;
                    }

               }
           for(k=0;k<LC_Tree[ LC_Tree[select_ptr][6] ][5];k++)
                   {
                    if ( H_and_W[   LC_Tree[select_ptr][6]  ][2][k]==select_ptr)
                      {
                       H_and_W[ LC_Tree[select_ptr][6]   ][2][k]=-1000;
                       break;
                      }
                   }
           LC_Tree[select_ptr][6]=-1000;
           LC_Tree[select_ptr][2]=xx1;
           LC_Tree[select_ptr][3]=yy1;
          }
    }

    else if(k==1)
        {
         result = notice_prompt(panel, NULL,
         NOTICE_FOCUS_XY,     t_x, t_y,
         NOTICE_MESSAGE_STRINGS, "can not move this tree because it has subtree(s)?", NULL,
         NOTICE_BUTTON_YES,    "dislink subtree(s)",
         NOTICE_BUTTON_NO,     "cancel",
         NULL);
```

```
          if (result == NOTICE_YES)
            {
             for(j=0;j<LC_Tree[select_ptr][5];j++)
               {
                if(H_and_W[select_ptr][2][j]>=0)
                    {
                     LC_Tree[H_and_W[select_ptr][2][j]][6]=-1000;
                     H_and_W[select_ptr][2][j]=-1000;
                    }

               }

            LC_Tree[select_ptr][2]=xx1;
            LC_Tree[select_ptr][3]=yy1;

            }
       }


    else if( LC_Tree[select_ptr][6]>=0)
     {
       result = notice_prompt(panel, NULL,
          NOTICE_FOCUS_XY,        t_x, t_y,
          NOTICE_MESSAGE_STRINGS, "can not move this tree because it has parent_tree(s)?", NULL,
          NOTICE_BUTTON_YES,      "dislink parenttree(s)",
          NOTICE_BUTTON_NO,       "cancel",
          NULL);

          if (result == NOTICE_YES)
            {
                 for(k=0;k<LC_Tree[ LC_Tree[select_ptr][6] ][5];k++)
                   {
                    if ( H_and_W[   LC_Tree[select_ptr][6]  ][2][k]==select_ptr)
                      {
                       H_and_W[ LC_Tree[select_ptr][6]  ][2][k]=-1000;
                       break;
                      }
                   }
          LC_Tree[select_ptr][6]=-1000;

          LC_Tree[select_ptr][2]=xx1;
          LC_Tree[select_ptr][3]=yy1;
            }

     }
```

```
      else {
         LC_Tree[select_ptr][2]=xx1;
         LC_Tree[select_ptr][3]=yy1;
         }
   }

   /*  sprintf(p, "Button: Menu (Right) %s at %d,%d",
           event_is_down(event)? "pressed" : "released",
           event_x(event), event_y(event));
        break;*/

         sprintf(p, "Button: Menu (Right) %s at %d,%d","pressed",event_x(event), event_y(event));
         break;
}
else return;

      case SHIFT_RIGHT :
        sprintf(p, "Keyboard: right shift %s",
           event_is_down(event)? "pressed" : "released");
        break;



      case SHIFT_LEFT :
        sprintf(p, "Keyboard: left shift %s",
           event_is_down(event)? "pressed" : "released");
        break;
      case SHIFT_LEFTCTRL : case SHIFT_RIGHTCTRL :
        sprintf(p, "Keyboard: control key %s",
           event_is_down(event)? "pressed" : "released");
        break;
      case SHIFT_META :
        sprintf(p, "Keyboard: meta key %s",
           event_is_down(event)? "pressed" : "released");
        break;
      case SHIFT_ALT :
        sprintf(p, "Keyboard: alt key %s",
           event_is_down(event)? "pressed" : "released");
        break;
      case KBD_USE:
        sprintf(p, "Keyboard: got keyboard focus");
        break;
     /* case KBD_DONE:
        sprintf(p, "Keyboard: lost keyboard focus");
        break;   */
```

```
    case LOC_MOVE:
    sprintf(p, "Pointer: moved to %d,%d",
            event_x(event),event_y(event));
      break;
    case LOC_DRAG:

/* xx1=event_x(event);
 yy1=event_y(event);

 if(pick_tree==1)
  {
  LC_Tree[select_ptr][2]=xx1;
  LC_Tree[select_ptr][3]=yy1;
  }*/


      sprintf(p, "Pointer: dragged to %d,%d",
            event_x(event), event_y(event));
      break;
/*    case LOC_WINENTER:
      win_set_kbd_focus(pw, xv_get(pw, XV_XID));
      sprintf(p, "Pointer: entered window at %d,%d",
            event_x(event), event_y(event));
      break;
    case LOC_WINEXIT:
      sprintf(p, "Pointer: exited window at %d,%d",
            event_x(event), event_y(event));
      break;   */
    case WIN_RESIZE :
    case WIN_REPAINT :
      return;
    default :
      /* There are too many ACTION events to trap -- ignore the
       * ones we're not interested in.
       */
      return;

  }

  canvas_repaint_proc(canvas, pw,
    xv_get(canvas, XV_DISPLAY), xv_get(pw, XV_XID), NULL);

  /* indicate which paint window and view window ID's */
 /* printf("win %x, view: %x\n", pw, view);*/
}
```

```
/*
 * notify this routine whenever two views are joined.
 */
void
join_split(view)
Xv_Window view;
{
  puts("joined view");
}

/*
 * Notify this routine whenever a view is split. The new view is
 * created and its position is indicated. This is the first time
 * the new view can be accessed by the program. Immediately install
 * the callback for events for the new paint window.
 */
void
init_split(oldview, newview, pos)
Xv_Window oldview, newview;
int pos;
{
  xv_set(xv_get(newview, CANVAS_VIEW_PAINT_WINDOW),
    WIN_EVENT_PROC,        events,
    WIN_CONSUME_EVENT,     ACTION_SELECT, ACTION_ADJUST, NULL,
    NULL);
}

int
file_selected(item, event)
Panel_item item;
Event *event;
{
  t_x= event_x(event);
  t_y= event_y(event);

  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
}

void
file_menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
 int       result;
```

```
if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Load"))
 {
 if(draw_tree_ptr >=0)
    {
       result = notice_prompt(panel, NULL,
       NOTICE_FOCUS_XY,        t_x, t_y,
       NOTICE_MESSAGE_STRINGS, "Loading file will clear current tree",
                 "Please confirm", NULL,
       NOTICE_BUTTON_YES,      "Confirm loading",
       NOTICE_BUTTON_NO,       "Cancel",
       NULL);
   if (result == NOTICE_YES)
       {
      xv_set( panel_file_name_item, PANEL_VALUE, "", NULL);

      file_type=1;
      printf("Menu Item: load  file \n");
      xv_set(subframe_f, XV_SHOW, TRUE, NULL);

       }
    }
 else
    {
    xv_set( panel_file_name_item, PANEL_VALUE, "", NULL);
    file_type=1;
    printf("Menu Item: load  file \n");
    xv_set(subframe_f, XV_SHOW, TRUE, NULL);

    }

}
else    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Save"))
         {
         xv_set( panel_file_name_item, PANEL_VALUE, "", NULL);
         file_type=2;
         printf("Menu Item: save file\n");
         xv_set(subframe_f, XV_SHOW, TRUE, NULL);

         }

else    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Save As"))
         {
         xv_set( panel_file_name_item, PANEL_VALUE, "", NULL);
         file_type=3;
         printf("Menu Item: save as file\n");
xv_set(subframe_f, XV_SHOW, TRUE, NULL);

         }
   else    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Quit"))
         {
         printf("Menu Item: Quit\n");

   result = notice_prompt(panel, NULL,
     NOTICE_FOCUS_XY,        t_x, t_y,
     NOTICE_MESSAGE_STRINGS, "Do you really want to quit?", NULL,
     NOTICE_BUTTON_YES,      "Yes",
     NOTICE_BUTTON_NO,       "No",
     NULL);

   if (result == NOTICE_YES)
     exit(0);

            }
}

int
menubar_selected(item, event)
Panel_item item;
Event *event;
{
  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
}

void
menubar_menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{

 xv_set( panel_meneme_num_item, PANEL_VALUE, "", NULL);

 /* printf("Menu Item: %s\n", xv_get(menu_item, MENU_STRING));*/


 if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "HORIZONTAL"))

        {
         create_type =1;
xv_set(subframe_1, XV_SHOW, TRUE, NULL);
printf("Menu Item: HORIZONTAL \n");
            }

  else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "VERTICLE"))

              {
          create_type =2;
          xv_set(subframe_1, XV_SHOW, TRUE, NULL);


                  printf("Menu Item: VERTICLE \n");
              }

}


int
menutype_selected(item, event)
Panel_item item;
Event *event;
{
  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
}

void
menutype_menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{


  /*printf("Menu Item: %s\n", xv_get(menu_item, MENU_STRING));*/

 xv_set( panel_meneme_num_item, PANEL_VALUE, "", NULL);




 if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "CHOICE"))

        {  create_type =3;
           xv_set(subframe_1, XV_SHOW, TRUE, NULL);
   printf("Menu Item: CHOICE \n");
        }
```

```
else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "EXCLUSIVE"))

              {
          create_type =4;
          xv_set(subframe_1, XV_SHOW, TRUE, NULL);


                  printf("Menu Item: EXCLUSIVE \n");
              }

 else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "COMPATIBLE"))

              {
          create_type =5;
          xv_set(subframe_1, XV_SHOW, TRUE, NULL);

                  printf("Menu Item: COMPATIBLE \n");
              }
}

int
help_selected(item, event)
Panel_item  item;
Event    *event;
{
  int       result;

  result = notice_prompt(panel, NULL,
     NOTICE_FOCUS_XY,       event_x(event), event_y(event),
     NOTICE_MESSAGE_STRINGS, "Do you really want to quit?", "Do you really want to quit?",
          "Do you really   want to quit?", "Do you reall   want to quit?", "Do you really want to quit?",
          "Do you reall want to quit?", "Do you reall   want to quit?", "Do you really want to quit?",
          "Do you really want to quit?", "Do you really want to quit?", "Do you really want to quit?",
          "Do you really want to quit?", "Do you really want to quit?", NULL,
     NOTICE_BUTTON_YES,    "Yes",
     NOTICE_BUTTON_NO,     "No",
     NULL);
}

int  Draw_Verticle_tree(j,dpy,xwin,gc)
Display    *dpy;
Window      xwin;

GC gc;
int j;
```

```
{
 int k,xo,yo,x,y;
 char *txt_ptr;
 xo=LC_Tree[j][2];
 yo=LC_Tree[j][3];
 if(LC_Tree[j][7]>=0 && LC_Tree[j][6]<0)
         {
         txt_ptr=&Textstring[LC_Tree[j][7]][0];
         /* x= strlen(txt_ptr)*5/2;*/
          x= (text_rect_width(txt_ptr)-2*text_rect_width(txt_ptr)/(strlen(txt_ptr)+2))/2;
         XDrawString(dpy, xwin, gc, xo-x, yo-5,txt_ptr, strlen(txt_ptr));
         }

 x=LC_Tree[j][2]+LC_Tree[j][4];
 y=LC_Tree[j][3];

 for(k=0;k<LC_Tree[j][5];k++)
   {
    y+=H_and_W[j][0][k];
    XDrawLine(dpy, xwin, gc,xo,yo,xo,y);
    XDrawLine(dpy, xwin, gc,xo,y,x,y);
    if(H_and_W[j][1][k]>=0)
      {
      txt_ptr=&Textstring[H_and_W[j][1][k]][0];
      XDrawString(dpy, xwin, gc, x+5, y+3,txt_ptr, strlen(txt_ptr));
      }
    yo=y;
   }
/* printf(" Draw_Verticle_tree %d \n",j);*/

 }


int  Draw_Horizontal_tree(j,dpy,xwin,gc)
Display    *dpy;
Window      xwin;
GC gc;


 int j;
 {
  int k,xo,yo,x,y,x00;
  char *txt_ptr;

  xo=LC_Tree[j][2];
  yo=LC_Tree[j][3];
  if(LC_Tree[j][7]>=0 && LC_Tree[j][6]<0)
         {
         txt_ptr=&Textstring[LC_Tree[j][7]][0];
        /* x00= strlen(txt_ptr)*5/2;*/
         x00= (text_rect_width(txt_ptr)-2*text_rect_width(txt_ptr)/(strlen(txt_ptr)+2))/2;

         XDrawString(dpy, xwin, gc, xo-x00, yo-5,txt_ptr, strlen(txt_ptr));
         }
 y=yo+LC_Tree[j][4];
 XDrawLine(dpy, xwin, gc,xo,yo,xo,y);
 yo=y;

 if(H_and_W[j][0][0]>0)

   {
   x=xo;
   y=yo+LC_Tree[j][4];

   for(k=0;k<LC_Tree[j][5];k++)
     {
     xo=x;
     x+=H_and_W[j][0][k];
     XDrawLine(dpy, xwin, gc,xo,yo,x,yo);
     XDrawLine(dpy, xwin, gc,x,yo,x,y);
     if(H_and_W[j][1][k]>=0)
         {
         txt_ptr=&Textstring[H_and_W[j][1][k]][0];
        /*x00= strlen(txt_ptr)*5/2;*/
         x00= (text_rect_width(txt_ptr)-2*text_rect_width(txt_ptr)/(strlen(txt_ptr)+2))/2;

         XDrawString(dpy, xwin, gc, x-x00, y+12,txt_ptr, strlen(txt_ptr));
         }
     }
   }

 else
   {
   xo=xo+H_and_W[j][0][0];
   x=xo;
   y+=LC_Tree[j][4];
   XDrawLine(dpy, xwin, gc,xo,yo,xo,y);
   if(H_and_W[j][1][0]>=0)
         {
         txt_ptr=&Textstring[H_and_W[j][1][0]][0];
```

```
            /*x00= strlen(txt_ptr)*5/2; */
            x00= (text_rect_width(txt_ptr)-2*text_rect_width(txt_ptr)/(strlen(txt_ptr)+2))/2;
            XDrawString(dpy, xwin, gc, x-x00, y+12,txt_ptr, strlen(txt_ptr));
            }

      for(k=1;k<LC_Tree[j][5];k++)
       {
       xo=x;
       x+=H_and_W[j][0][k];
       XDrawLine(dpy, xwin, gc,xo,yo,x,yo);
       XDrawLine(dpy, xwin, gc,x,yo,x,y);
       if(H_and_W[j][1][k]>=0)
            {
            txt_ptr=&Textstring[H_and_W[j][1][k]][0];
            /*x00= strlen(txt_ptr)*5/2;*/
            x00= (text_rect_width(txt_ptr)-2*text_rect_width(txt_ptr)/(strlen(txt_ptr)+2))/2;

            XDrawString(dpy, xwin, gc, x-x00, y+12,txt_ptr, strlen(txt_ptr));
            }
       }
       if(x<LC_Tree[j][2])
            XDrawLine(dpy, xwin, gc,x,yo,LC_Tree[j][2],yo);
    }
/*printf(" Draw_Horizontal_tree %d \n",j);*/


  }


int create_tree_by_menume_number(item,event)   /*change_tree_by_factor(item, event)*/

Panel_item item;
Event *event;
{
   int result,j,k;
   char buf[128];
   int input;
   char *name = (char *)xv_get(item, PANEL_VALUE);
   Frame frame = (Frame)xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);

   input=atoi(name);
  printf("create_tree_by_menume_number create_type %d input %d \n",create_type,input);

  if(input<=0)
       {
```

```
     result = notice_prompt(panel, NULL,
     NOTICE_FOCUS_XY,      t_x, t_y,
     NOTICE_MESSAGE_STRINGS, "Menume number must be input ",
                   "before creating menubar or menu ","        ",
                   "Select O K to continue.", NULL,
     NOTICE_BUTTON_YES,     "O K",
       NULL);
       }
 else if( create_type==1)
       {
 sprintf(buf, " Create horizontal menubar");
   xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_1, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_1, XV_SHOW, FALSE, NULL);

     Create_Horizontal_tree(input,3);
       }
else if( create_type==2)
       {
 sprintf(buf, " Create verticle menubar");
   xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_1, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_1, XV_SHOW, FALSE, NULL);
     Create_Verticle_tree(input,0);
       }
else if( create_type==3)
       {
 sprintf(buf, " Create choice menu");
   xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_1, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_1, XV_SHOW, FALSE, NULL);

     Create_Verticle_tree(input,1);
       }
else if( create_type==4)
       {
 sprintf(buf, " Create exclusive menu");
   xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_1, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_1, XV_SHOW, FALSE, NULL);

     Create_Verticle_tree(input,2);
       }
else if( create_type==5)
```

```
        {
  sprintf(buf, " Create compatible menu");
    xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_1, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_1, XV_SHOW, FALSE, NULL);

     Create_Horizontal_tree(input,4);

     }


     return PANEL_NONE;

 }

int  Create_Verticle_tree(input,type)
       int input,type;
       {
         int k,hight;
         draw_tree_ptr++;

         if(type==0)
           hight=V_hight;
         else
           hight=V_hight/2;
         LC_Tree[draw_tree_ptr][0]=0;
         LC_Tree[draw_tree_ptr][1]=type;
         LC_Tree[draw_tree_ptr][2]=Tree_xo;
         LC_Tree[draw_tree_ptr][3]=Tree_yo;
         LC_Tree[draw_tree_ptr][4]=V_width;
         LC_Tree[draw_tree_ptr][5]=input;
         LC_Tree[draw_tree_ptr][6]=-100;
         LC_Tree[draw_tree_ptr][7]=-100;

         for(k=0;k<input;k++)
           {
            H_and_W[draw_tree_ptr][0][k]=hight;
            H_and_W[draw_tree_ptr][1][k]=-100;
            H_and_W[draw_tree_ptr][2][k]=-100;
           }
         Max_Min_Box[draw_tree_ptr][0]= 0;
         Max_Min_Box[draw_tree_ptr][1]= V_width;
         Max_Min_Box[draw_tree_ptr][2]= 0;
         Max_Min_Box[draw_tree_ptr][3]= input*hight;
         select_ptr=draw_tree_ptr;
         pick_tree=1;
            printf(" create_Verticle_tree select_ptr: %d \n", select_ptr);
      printf(" type= %d xx1= %d yy1= %d V_width= %d menume number= %d
\n",type,xx1,yy1,V_width,input);
         for(k=0;k<=7;k++)
         printf("LC-Tree[ %d ] [%d] %d \n", draw_tree_ptr,k, LC_Tree[draw_tree_ptr][k]);

         for(k=0;k<LC_Tree[draw_tree_ptr][5];k++)
           printf("H_and_W[%d][0][%d]=%d,H_and_W[%d][1][%d]=%d,H_and_W[%d][2][%d]=%d\n",
           draw_tree_ptr,k,H_and_W[draw_tree_ptr][0][k],draw_tree_ptr,k,H_and_W[draw_tree_ptr][1][k],
           draw_tree_ptr,k,H_and_W[draw_tree_ptr][2][k]);

         for(k=0;k<=3;k++)
         printf("max_min_box[ %d ] [%d] %d \n", draw_tree_ptr,k,Max_Min_Box[draw_tree_ptr][k]);
        }
int  Create_Horizontal_tree(input,type)
       int input,type;
       {
         int k, j ,hight;
         draw_tree_ptr++;
         if(type==3)
           hight=H_hight-10;
         else
           hight=H_hight-10;
         LC_Tree[draw_tree_ptr][0]=0;
         LC_Tree[draw_tree_ptr][1]=type;
         LC_Tree[draw_tree_ptr][2]=Tree_xo;
         LC_Tree[draw_tree_ptr][3]=Tree_yo;
         LC_Tree[draw_tree_ptr][4]=hight;
         LC_Tree[draw_tree_ptr][5]=input;
         LC_Tree[draw_tree_ptr][6]=-100;
         LC_Tree[draw_tree_ptr][7]=-100;
         j=(input-1)*H_width/2;
         H_and_W[draw_tree_ptr][0][0]=-j;
         H_and_W[draw_tree_ptr][1][0]=-100;
         H_and_W[draw_tree_ptr][2][0]=-100;
         for(k=1;k<input;k++)
           {
            H_and_W[draw_tree_ptr][0][k]=H_width;
            H_and_W[draw_tree_ptr][1][k]=-100;
            H_and_W[draw_tree_ptr][2][k]=-100;
           }
         Max_Min_Box[draw_tree_ptr][0]= -j;
         Max_Min_Box[draw_tree_ptr][1]= j;
         Max_Min_Box[draw_tree_ptr][2]= 0;
         Max_Min_Box[draw_tree_ptr][3]= 2*hight;
         select_ptr=draw_tree_ptr;
         pick_tree=1;
         printf(" create_Horizontal_tree %d \n", select_ptr);
printf(" type= %d xx1= %d yy1= %d V_width= %d menume number= %d  \n",type,xx1,yy1,V_width,input);
         for(k=0;k<=7;k++)
           printf("LC-Tree[ %d ] [%d] %d \n", draw_tree_ptr,k, LC_Tree[draw_tree_ptr][k]);
         for(k=0;k<LC_Tree[draw_tree_ptr][5];k++)
           printf("H_and_W[%d][0][%d]=%d,H_and_W[%d][1][%d]=%d,H_and_W[%d][2][%d]=%d\n",
           draw_tree_ptr,k,H_and_W[draw_tree_ptr][0][k],draw_tree_ptr,k,H_and_W[draw_tree_ptr][1][k],
           draw_tree_ptr,k,H_and_W[draw_tree_ptr][2][k]);
         for(k=0;k<=3;k++)
         printf("max_min_box[ %d ] [%d] %d \n", draw_tree_ptr,k,Max_Min_Box[draw_tree_ptr][k]);

       }

int  meneme_operate_menu_selected(item, event)
 Panel_item item;
 Event *event;
{
  t_x= event_x(event);
  t_y= event_y(event);
  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
}

void  meneme_operate_menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
int      result,i,j,k;


   if(select_ptr<0)
      {
    result = notice_prompt(panel, NULL,
    NOTICE_FOCUS_XY,      t_x, t_y,
    NOTICE_MESSAGE_STRINGS, "A tree must be selected before",
                  "having a operation on it ","         ",
                  "Select O K to continue.", NULL,
    NOTICE_BUTTON_YES,     "O K",
      NULL);
      }
  else if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "TEXT"))
        {
```

```
       xv_set( panel_meneme_text_item, PANEL_VALUE, "", NULL);
       if(LC_Tree[select_ptr][1]<=2)
          {
          if(pick_width >=0)
             {
               xv_set(subframe_2, XV_SHOW, TRUE, NULL);
               printf("Menu Item: add text to meneme of verticle tree \n");
             }
          else if (pick_hight==0 )
             {
             xv_set(subframe_2, XV_SHOW, TRUE, NULL);
             printf("Menu Item: add text to root of verticle tree \n");
             }

          else
             {
             result = notice_prompt(panel, NULL,
             NOTICE_FOCUS_XY,      t_x, t_y,
NOTICE_MESSAGE_STRINGS, "A horizontal line(meneme) or root of the tree  must be selected ",
                   "before adding a text to it ","        ",
                   "Select O K to continue.", NULL,
             NOTICE_BUTTON_YES,     "O K",
             NULL);
             }

          }
     else
        {
          xv_set( panel_meneme_text_item, PANEL_VALUE, "", NULL);

          if(pick_hight>=0 && pick_hight<100)
             {
              xv_set(subframe_2, XV_SHOW, TRUE, NULL);
              printf("Menu Item: add text to menume of horizontal tree\n");
             }

          else if(pick_hight==100)
             {
              xv_set(subframe_2, XV_SHOW, TRUE, NULL);
              printf("Menu Item: add text to root of horizontal tree\n");
             }

          else
             {
              result = notice_prompt(panel, NULL,
                NOTICE_FOCUS_XY,       t_x,t_y,
NOTICE_MESSAGE_STRINGS, "A verticle line(meneme or root of the tree ) must be selected ",
                        "before adding a text to it ","          ",
                        "Select O K to continue.", NULL,
                NOTICE_BUTTON_YES,    "O K",
                NULL);
                }


          }
      }

  else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "ADD"))

               {


                  printf("Menu Item: add meneme \n");
               }

  else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "DELETE"))

    {

    result = notice_prompt(panel, NULL,
    NOTICE_FOCUS_XY,     t_x,t_y,
    NOTICE_MESSAGE_STRINGS, "Do you really want to DELETE this meneme ?", NULL,
    NOTICE_BUTTON_YES,     "Yes",
    NOTICE_BUTTON_NO,     "No",
    NULL);

  if (result == NOTICE_YES)

             {
             printf("Menu Item: delete   meneme\n");
             }

    }
  else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "UNDO"))
        {
          printf("Menu Item: undo\n");
        }
}
int   tree_operate_selected(item, event)
  Panel_item item;
  Event *event;
```

```
{
printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
t_x= event_x(event);
t_y= event_y(event);
return XV_OK;
}
 void   tree_operate_proc(menu, menu_item)
 Menu menu;
 Menu_item menu_item;
  {
  static int tree_operation, link_line,link_ptr,subtree_ptr;
  int   result,i,j,k,link;
  if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "AUTO LINK"))
             {
                tree_operation=1;
                printf("Menu Item: auto link subtree \n");
             }
 else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "UNLINK ALL"))
             {
                tree_operation=2;
                printf("Menu Item: unlink all subtree \n");
             }
 else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "DELETE ALL"))
  {
   result = notice_prompt(panel, NULL,
    NOTICE_FOCUS_XY,     t_x,t_y,
    NOTICE_MESSAGE_STRINGS, "Do you really want to DELETE ALL trees or subtrees?", NULL,
    NOTICE_BUTTON_YES,     "Yes",
    NOTICE_BUTTON_NO,     "No",
    NULL);
   if (result == NOTICE_YES)
            {     old_draw_tree_ptr=draw_tree_ptr;
                old_select_ptr=select_ptr;
                old_text_ptr=text_ptr;
                text_ptr=-1;
                draw_tree_ptr=-1;
                select_ptr=-100;
                printf("Menu Item: delete   ALL   trees and clear the screen\n");
                tree_operation=3;
            }
  }
else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "UNDO"))
       {
         if   (tree_operation==3)
              {
                draw_tree_ptr= old_draw_tree_ptr;
                select_ptr= old_select_ptr;
                printf("Menu Item: undo :restore  ALL  trees deleted by last delete operation \n");
                }
        else if (tree_operation==7)
                {
                 if(select_ptr==link_ptr && (pick_width==link_line || pick_hight == link_line) )
                    {
                     LC_Tree[ subtree_ptr ][6]=link_ptr;
                     H_and_W[select_ptr][2][link_line] = subtree_ptr ;
                    }
                  else
                    {
                    result = notice_prompt(panel, NULL,
                    NOTICE_FOCUS_XY,        t_x, t_y,
                    NOTICE_MESSAGE_STRINGS, "impossible undo operation",
                   "three is(are) other operation(s) between unlink and undo unlink operation !","          ",
                    "Select O K to continue.", NULL,
                    NOTICE_BUTTON_YES,      "O K",
                    NULL);
                    }
                }
        else if (tree_operation==8)
                {
                draw_tree_ptr+=1;
                select_ptr=draw_tree_ptr;
                printf("Menu Item: undo: restore a tree deleted by last operation \n");
                if( (LC_Tree[draw_tree_ptr][6])>=0)
                   H_and_W[ LC_Tree[draw_tree_ptr][6]    ][2][ link_line]= draw_tree_ptr;
                for(k=0;k< LC_Tree[draw_tree_ptr][5];k++)
                   if ( H_and_W[draw_tree_ptr][2][k]>=0)
LC_Tree[ H_and_W[draw_tree_ptr][2][k]][6]=draw_tree_ptr; /*H_and_W[draw_tree_ptr][2][k]*/
                }
        tree_operation=4;
      }
 else if(select_ptr<0)
    {
   result = notice_prompt(panel, NULL,
   NOTICE_FOCUS_XY,        t_x, t_y,
   NOTICE_MESSAGE_STRINGS, "A tree must be selected before",
                  "having a operation on it ","          ",
                  "Select O K to continue.", NULL,
   NOTICE_BUTTON_YES,      "O K",
     NULL);
    }
```

```
else if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "MODIFY"))
      {
        xv_set( panel_tree_modify_item, PANEL_VALUE, "", NULL);

        if(pick_hight>=0 || pick_width>=0)
            {
            printf("Menu Item: modify \n");
            xv_set(subframe_t, XV_SHOW, TRUE, NULL);
            tree_operation=5;
            }
        else
            {
            result = notice_prompt(panel, NULL,
            NOTICE_FOCUS_XY,        t_x, t_y,
            NOTICE_MESSAGE_STRINGS, "A line of the tree must be selected ",
                  "before scaling it ","          ",
                  "Select O K to continue.", NULL,
            NOTICE_BUTTON_YES,      "O K",
            NULL);
            }
      }

else    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "LINK"))

      {
        if( pick_width>=0)
            {
             link=link_subtree_to_Verticle_tree_branch(select_ptr,pick_width);

printf("link=%d\n",link);

             if(link==0)
                  {
                result = notice_prompt(panel, NULL,
                NOTICE_FOCUS_XY,        t_x, t_y,
                NOTICE_MESSAGE_STRINGS, "can not find subtree near this branch ",
                 "modify and move subtree to this branch, please!","          ",
                 "Select O K to continue.", NULL,
                NOTICE_BUTTON_YES,      "O K",
                NULL);
                  }
             tree_operation=6;
             printf("Menu Item: link subtree \n");
            }
        else  if(pick_hight>=0 )
        {

          if(pick_hight<100)
            {
              link=link_subtree_to_Horizontal_tree_branch();
              printf("link=%d\n",link);
              if(link==0)
                {
                  result = notice_prompt(panel, NULL,
                  NOTICE_FOCUS_XY,      t_x,t_y,
                  NOTICE_MESSAGE_STRINGS, "can not find subtree near this branch ",
                  "modify and move subtree to this branch, please!","          ",
                  "Select O K to continue.", NULL,
                  NOTICE_BUTTON_YES,     "O K",
                  NULL);
                }
              tree_operation=6;
              printf("Menu Item: link subtree \n");
            }
          else    /*if(pick_hight==100)*/
            {
              result = notice_prompt(panel, NULL,
              NOTICE_FOCUS_XY,      t_x,t_y,
              NOTICE_MESSAGE_STRINGS, "impossible link ! ",
              "only a menume can link a subtree","          ",
              "Select O K to continue.", NULL,
              NOTICE_BUTTON_YES,     "O K",
              NULL);
            }
        }
      else
        {
        result = notice_prompt(panel, NULL,
        NOTICE_FOCUS_XY,      t_x,t_y,
        NOTICE_MESSAGE_STRINGS, "A line of the tree must be selected ",
              "before link it to subtree","          ",
              "Select O K to continue.", NULL,
        NOTICE_BUTTON_YES,     "O K",
        NULL);
        }
    }
else    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "UNLINK"))

    {
      if(pick_width>=0 )
        {
         if ( H_and_W[select_ptr][2][pick_width]>=0)
           {
             link_line=pick_width;
             link_ptr=select_ptr;
             subtree_ptr= H_and_W[select_ptr][2][pick_width] ;
             LC_Tree[  H_and_W[select_ptr][2][pick_width]  ][6]=-1000;
             H_and_W[select_ptr][2][pick_width] =-1000;
             tree_operation=7;
             printf("Menu Item: unlink subtree \n");
                        }
         else
           {
             result = notice_prompt(panel, NULL,
             NOTICE_FOCUS_XY,      t_x,t_y,
             NOTICE_MESSAGE_STRINGS, "impossible unlink operation",
             "no subtree linked with this selected menume","          ",
             "Select O K to continue.", NULL,
             NOTICE_BUTTON_YES,     "O K",
             NULL);
             tree_operation=0;
           }
        }
      else if(pick_hight>=0 )
        {
          if(pick_hight<100)
            {
             if ( H_and_W[select_ptr][2][pick_hight]>=0)
               {
                 link_line=pick_hight;
                 link_ptr=select_ptr;
                 subtree_ptr= H_and_W[select_ptr][2][pick_width] ;
                 LC_Tree[  H_and_W[select_ptr][2][pick_hight]  ][6]=-1000;
                 H_and_W[select_ptr][2][pick_hight] =-1000;
                 tree_operation=7;
                 printf("Menu Item: unlink subtree \n");
               }
             else
               {
                 result = notice_prompt(panel, NULL,
                 NOTICE_FOCUS_XY,      t_x,t_y,
                 NOTICE_MESSAGE_STRINGS, "impossible unlink operation",
                 "no subtree linked with this selected menume","          ",
                 "Select O K to continue.", NULL,
                 NOTICE_BUTTON_YES,     "O K",
                    NULL);
                    tree_operation=0;
                    }
                }
            else   /*if((pick_hight==100)*/
                {
                 result = notice_prompt(panel, NULL,
                 NOTICE_FOCUS_XY,       t_x, t_y,
                 NOTICE_MESSAGE_STRINGS, "impossible link ! ",
                 "only a menume can link a subtree","           ",
                 "Select O K to continue.", NULL,
                 NOTICE_BUTTON_YES,     "O K",
                 NULL);
                 tree_operation=0;
                 }
            }
        else
            {
             result = notice_prompt(panel, NULL,
             NOTICE_FOCUS_XY,       t_x, t_y,
             NOTICE_MESSAGE_STRINGS, "A line of the tree must be selected ",
                "before unlink its subtree ","          ",
                "Select O K to continue.", NULL,
             NOTICE_BUTTON_YES,     "O K",
             NULL);
             tree_operation=0;
            }
    }

else   if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "DELETE"))

  {

  result = notice_prompt(panel, NULL,
  NOTICE_FOCUS_XY,       t_x, t_y,
  NOTICE_MESSAGE_STRINGS, "Do you really want to DELETE this tree or subtree?", NULL,
  NOTICE_BUTTON_YES,     "Yes",
  NOTICE_BUTTON_NO,      "No",
  NULL);

  if (result == NOTICE_YES)

            {
             printf("Menu Item: delete  tree\n");
             LC_Tree[draw_tree_ptr+1][0]= LC_Tree[select_ptr][0];
             LC_Tree[draw_tree_ptr+1][1]= LC_Tree[select_ptr][1];
             LC_Tree[draw_tree_ptr+1][2]= LC_Tree[select_ptr][2];
             LC_Tree[draw_tree_ptr+1][3]= LC_Tree[select_ptr][3];
             LC_Tree[draw_tree_ptr+1][4]= LC_Tree[select_ptr][4];
             LC_Tree[draw_tree_ptr+1][5]= LC_Tree[select_ptr][5];
             LC_Tree[draw_tree_ptr+1][6]= LC_Tree[select_ptr][6];
             LC_Tree[draw_tree_ptr+1][7]= LC_Tree[select_ptr][7];
                for(k=0;k<LC_Tree[select_ptr][5];k++)
              {
               H_and_W[draw_tree_ptr+1][0][k]= H_and_W[select_ptr][0][k];
               H_and_W[draw_tree_ptr+1][1][k]= H_and_W[select_ptr][1][k];
               H_and_W[draw_tree_ptr+1][2][k]= H_and_W[select_ptr][2][k];
              }
             Max_Min_Box[draw_tree_ptr+1][0]= Max_Min_Box[select_ptr][0];
             Max_Min_Box[draw_tree_ptr+1][1]= Max_Min_Box[select_ptr][1];
             Max_Min_Box[draw_tree_ptr+1][2]= Max_Min_Box[select_ptr][2];
             Max_Min_Box[draw_tree_ptr+1][3]= Max_Min_Box[select_ptr][3];
             if( (LC_Tree[draw_tree_ptr+1][6])>=0)
               {
                 for(k=0;k<LC_Tree[  LC_Tree[draw_tree_ptr+1][6]  ][5];k++)
                   {
                    if ( H_and_W[  LC_Tree[draw_tree_ptr+1][6]   ][2][k]==select_ptr)
                      {
                       H_and_W[ LC_Tree[draw_tree_ptr+1][6]   ][2][k]=-1000;
                       link_line=k;
                       break;
                      }
                   }

                 if( (LC_Tree[draw_tree_ptr+1][6]) > select_ptr )    LC_Tree[draw_tree_ptr+1][6]-=1;
               }

                 for(k=0;k< LC_Tree[draw_tree_ptr+1][5];k++)
                   {
                    if ( H_and_W[draw_tree_ptr+1][2][k]>=0)
                      {
                       LC_Tree[   H_and_W[draw_tree_ptr+1][2][k]   ][6]=-1000;
if(   H_and_W[draw_tree_ptr+1][2][k]   > select_ptr )    H_and_W[draw_tree_ptr+1][2][k] -=1;
                      }
                   }
             for(j=select_ptr;j<=draw_tree_ptr+1;j++)
               {
                LC_Tree[j][0]= LC_Tree[j+1][0];
                LC_Tree[j][1]= LC_Tree[j+1][1];
                LC_Tree[j][2]= LC_Tree[j+1][2];
```

```
                    LC_Tree[j][3]= LC_Tree[j+1][3];
                    LC_Tree[j][4]= LC_Tree[j+1][4];
                    LC_Tree[j][5]= LC_Tree[j+1][5];
                    LC_Tree[j][6]= LC_Tree[j+1][6];
                    LC_Tree[j][7]= LC_Tree[j+1][7];
                    for(k=0;k<LC_Tree[j+1][5];k++)
                      {
                       H_and_W[j][0][k]= H_and_W[j+1][0][k];
                       H_and_W[j][1][k]= H_and_W[j+1][1][k];
                       H_and_W[j][2][k]= H_and_W[j+1][2][k];
                      }
                    Max_Min_Box[j][0]= Max_Min_Box[j+1][0];
                    Max_Min_Box[j][1]= Max_Min_Box[j+1][1];
                    Max_Min_Box[j][2]= Max_Min_Box[j+1][2];
                    Max_Min_Box[j][3]= Max_Min_Box[j+1][3];
                  }

                draw_tree_ptr-=1;
                  for(j=0;j<=draw_tree_ptr;j++)
                    {
                     if( (LC_Tree[j][6]) > select_ptr ) LC_Tree[j][6]-=1;
                     for(k=0;k<LC_Tree[j][5];k++)
                          if(  H_and_W[j][2][k]  > select_ptr )   H_and_W[j][2][k] -=1;
                    }
                select_ptr=-100;
                tree_operation=8;
               }

     }


 }

int which_line_selected(ptr,x1,y1)
 int ptr,x1,y1;
 {
  int x0,y0,x2,y2,j;
   pick_hight=-1;
   pick_width=-1;
  if(LC_Tree[ptr][1]<=2)
    {
     x0=LC_Tree[ptr][2];
     y0=LC_Tree[ptr][3];
     x2=x0+LC_Tree[ptr][4];
     y2=y0;
```

```
    for(j=0;j<LC_Tree[ptr][5];j++)
      {
       y2+=H_and_W[ptr][0][j];
       if(x1>=x0-5 && x1<= x0+5 && y1>=y0 && y1 <=y2)
         {
          pick_hight=j;
          printf("pick_hight %d\n", j);
          break;
         }
       else if(x1 >= x0 && x1 <= x2 && y1 >= y2-5 && y1 <= y2+5)
         {
          pick_width=j;
          printf("pick_width %d\n", j);
          break;
         }
      }
    }
  else
    {
     x0=LC_Tree[ptr][2];
     y0=LC_Tree[ptr][3];
     y2=y0+LC_Tree[ptr][4];
     if(x1>=x0-5 && x1<= x0+5 && y1>=y0 && y1 <=y2)
         {
          pick_hight=100;
          printf("pick_hight %d\n", pick_hight);
         }
     else  if( H_and_W[ptr][0][0]>=0)
         {  x2=x0;
           y0=y2;
           y2+=LC_Tree[ptr][4];
           for(j=0;j<LC_Tree[ptr][5];j++)
            {
             x0=x2;
             x2+=H_and_W[ptr][0][j];
             if(x1>=x0 && x1<= x2 && y1>=y0-5 && y1 <=y0+5)
               {
                pick_width=j;
                printf("pick_width %d\n", pick_width);
                break;
               }
             else if(x1>=x2-5 && x1<= x2+5 && y1>=y0 && y1 <=y2)
               {
                pick_hight=j;
                printf("pick_hight %d\n", pick_hight);
```

```
                    break;
                    }
                 }

              }
    else
      {
        x0+=H_and_W[ptr][0][0];
        x2=x0;
        y0=y2;
        y2+=LC_Tree[ptr][4];
        if(x1>=x0-5 && x1<= x0+5 && y1>=y0 && y1 <=y2)
          {
           pick_hight=0;
           printf("pick_hight %d\n", pick_hight);
          }

        else
          {
           for(j=1;j<LC_Tree[ptr][5];j++)
             {
              x0=x2;
              x2+=H_and_W[ptr][0][j];
              if(x1>=x0 && x1<= x2 && y1>=y0-5 && y1 <=y0+5)
                {
                 pick_width=j;
                 printf("pick_width %d\n", pick_width);
                 break;
                 }

              else if(x1>=x2-5 && x1<= x2+5 && y1>=y0 && y1 <=y2)
                {
                 pick_hight=j;
                 printf("pick_hight %d\n", pick_hight);
                 break;
                 }
             }
             if( pick_hight<0 && pick_width<0)
               {
                if(x1>=x2 && x1<=LC_Tree[ptr][2]&& y1>=y0-5 && y1 <=y0+5)
                   {
                    pick_width=0;
                    printf("pick_width %d\n", pick_width);
                    }
               }
          }
```

```
          }
       }
}
int add_text_to_selected_menume(item,event)
  Panel_item item;
  Event *event;
  {
  int result,j,k;
  char *my_text_ptr;
  char buf[128];
  char *name = (char *)xv_get(item, PANEL_VALUE);
  Frame frame = (Frame)xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
printf("add_text_to_selected_menume select_ptr %d pick_hight %d pick_width %d text %s\n",

                                          select_ptr,pick_hight,pick_width,name );
  if(LC_Tree[select_ptr][1]<=2)
     {
     if(pick_hight==0)
         {
           sprintf(buf, " Add text to a selected root of verticle tree ");
           xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
           if ((int)xv_get(subframe_2, FRAME_CMD_PUSHPIN_IN) == FALSE)
           xv_set(subframe_2, XV_SHOW, FALSE, NULL);
           text_ptr++;
           LC_Tree[select_ptr][7]=text_ptr;
           my_text_ptr=&Textstring[text_ptr][0];
           strcpy(my_text_ptr,name);
           printf("add %s to root of verticle tree \n",my_text_ptr);
         }
     else if(pick_width>=0)
       {
           sprintf(buf, " Add text to a selected meneme ");
           xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
           if ((int)xv_get(subframe_2, FRAME_CMD_PUSHPIN_IN) == FALSE)
           xv_set(subframe_2, XV_SHOW, FALSE, NULL);
           text_ptr++;
           H_and_W[select_ptr][1][pick_width]=text_ptr;
           my_text_ptr=&Textstring[text_ptr][0];
           strcpy(my_text_ptr,name);
           printf("add %s to selected_menume of verticle tree \n",my_text_ptr);
       }
     }
  else
     {
     if(pick_hight==100)
         {
            sprintf(buf, " Add text to a selected  root of  horizontal tree ");
            xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
            if ((int)xv_get(subframe_2, FRAME_CMD_PUSHPIN_IN) == FALSE)
            xv_set(subframe_2, XV_SHOW, FALSE, NULL);
            text_ptr++;
            LC_Tree[select_ptr][7]=text_ptr;
            my_text_ptr=&Textstring[text_ptr][0];
            strcpy(my_text_ptr,name);
            printf("add %s to root of  horizontal tree \n",my_text_ptr);
        }
      else  if(pick_hight>=0)
        {
            sprintf(buf, " Add text to a selected meneme ");
            xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
            if ((int)xv_get(subframe_2, FRAME_CMD_PUSHPIN_IN) == FALSE)
            xv_set(subframe_2, XV_SHOW, FALSE, NULL);
            text_ptr++;
            H_and_W[select_ptr][1][pick_hight]=text_ptr;
            my_text_ptr=&Textstring[text_ptr][0];
            strcpy(my_text_ptr,name);
            printf("add %s to selected_menume of horizontal tree \n",my_text_ptr);
        }
    }
    return PANEL_NONE;

   }


int   treat_file_by_name(item,event)
Panel_item item;
Event *event;
{
   FILE  *fp,*fopen();
   char  *filename, *type;
   int  result,i,j,k;
   char buf[128];
   char *sp;
   int input;
   char *name = (char *)xv_get(item, PANEL_VALUE);
   Frame frame = (Frame)xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
   printf("treat file by file name  file_type %d file_name %s \n",file_type,name);
   input=strlen(name);
   if(input<=0)
      {
    result = notice_prompt(panel, NULL,
```

```
    NOTICE_FOCUS_XY,        t_x, t_y,
    NOTICE_MESSAGE_STRINGS, "Impossible file operation  ",
                  "File name  must  be  input ",
                  "before loading or saving file ","           ",
                  "Select O K to continue.", NULL,
    NOTICE_BUTTON_YES,      "O K",
     NULL);
    }
  else if( file_type==1)
    {
     sprintf(buf, " Loading file");
   . xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
     if ((int)xv_get(subframe_f, FRAME_CMD_PUSHPIN_IN) == FALSE)
     xv_set(subframe_f, XV_SHOW, FALSE, NULL);
         filename=name;
         type="r";
         printf("filename %s type %s\n",filename,type);
         if((fp=fopen(filename,type))==NULL)
         printf("open file feiled\n");
         else
            {
             select_ptr=-1;
             pick_hight=-1;
             pick_width=-1;
             draw_tree_ptr=-1;
             text_ptr=-1;
             fscanf(fp,"%d\n",&draw_tree_ptr);
             printf("draw_tree_ptr=%d\n",draw_tree_ptr);

             for(i=0;i<=draw_tree_ptr;i++)
                 {
                  for(j=0;j<=7;j++)
                      {
                       fscanf(fp,"%d ",&LC_Tree[i][j]);
                       printf("Lc_tree[%d][%d]= %d\n",i,j,LC_Tree[i][j]);

                      }
                  for(j=0;j<LC_Tree[i][5];j++)
                      {
                      /* printf("input%d\n",j);*/
                       fscanf(fp,"%d %d %d
",&H_and_W[i][0][j],&H_and_W[i][1][j],&H_and_W[i][2][j]);
printf("H_and_W[%d][0][%d]=%d,H_and_W[%d][1][%d]=%d,H_and_W[%d][2][%d]=%d\n",i,j,

        H_and_W[i][0][j],i,j,H_and_W[i][1][j],i,j,H_and_W[i][2][j]);
```

```
                    }
                for(j=0;j<=3;j++)
                    {
                    fscanf(fp,"%d ",&Max_Min_Box[i][j]);
                    printf("max_min_box[%d][%d]=%d\n",i,j,Max_Min_Box[i][j]);

                    }
                }
            fscanf(fp,"\n");
            fscanf(fp,"%d\n",&text_ptr);
            printf("text_ptr=%d\n",text_ptr);

            if(text_ptr>=0)
                {
                for(k=0;k<=text_ptr;k++)
                    {
                    sp=&Textstring[k][0];
                    fscanf(fp,"%s ",sp);
                    printf("%D     %s\n",k,sp);
                    }
                }
            fscanf(fp,"\n");
            fscanf(fp,"%d %d %d\n",&select_ptr,&pick_hight,&pick_width);
            printf("select_ptr=%d,pick_hight=%d,pick_width=%d\n",select_ptr,pick_hight,pick_width);

            fclose(fp);
            }
    }
else if( file_type==2)
    {
    sprintf(buf, " Saving file");
    xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
    if ((int)xv_get(subframe_f, FRAME_CMD_PUSHPIN_IN) == FALSE)
    xv_set(subframe_f, XV_SHOW, FALSE, NULL);
    /* saving file */

    }
else if( file_type==3)
    {
    sprintf(buf, " saving as a new file");
    xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
    if ((int)xv_get(subframe_f, FRAME_CMD_PUSHPIN_IN) == FALSE)
    xv_set(subframe_f, XV_SHOW, FALSE, NULL);

    /* save as a new file*/
```

```
        filename=name;
        type="w";
        if((fp=fopen(filename,type))==NULL)
          printf("open file feiled\n");
        else
          {
           fprintf(fp,"%d\n",draw_tree_ptr);
           printf("draw_tree_ptr=%d\n",draw_tree_ptr);

           for(i=0;i<=draw_tree_ptr;i++)
              {
               for(j=0;j<=7;j++)
                  {
                   fprintf(fp,"%d ",LC_Tree[i][j]);
                   printf("Lc_tree[%d][%d]= %d\n",i,j,LC_Tree[i][j]);
                  }
               for(j=0;j<LC_Tree[i][5];j++)
                  {
                    fprintf(fp,"%d %d %d ",H_and_W[i][0][j],H_and_W[i][1][j],H_and_W[i][2][j]);
printf("H_and_W[%d][0][%d]=%d,H_and_W[%d][1][%d]=%d,H_and_W[%d][2][%d]=%d\n",i,j,

          H_and_W[i][0][j],i,j,H_and_W[i][1][j],i,j,H_and_W[i][2][j]);
                  }
               for(j=0;j<=3;j++)
                  {
                  fprintf(fp,"%d ",Max_Min_Box[i][j]);
                   printf("max_min_box[%d][%d]=%d\n",i,j,Max_Min_Box[i][j]);
                  }
              }
           fprintf(fp,"\n");
           fprintf(fp,"%d\n",text_ptr);
            printf("text_ptr=%d\n",text_ptr);

               if(text_ptr>=0)
                  {
                  for(k=0;k<=text_ptr;k++)
                     {
                      sp=&Textstring[k][0];
                      fprintf(fp,"%s ",sp);
                       printf("%D     %s\n",k,sp);

                     }
                  }
           fprintf(fp,"\n");
           fprintf(fp,"%d %d %d\n",select_ptr,pick_hight,pick_width);
            printf("select_ptr=%d,pick_hight=%d,pick_width=%d\n",select_ptr,pick_hight,pick_width);
          fclose(fp);
        }
      }

      return PANEL_NONE;

  }

int  modify_tree_by_scale(item,event)
   Panel_item item;
   Event *event;
   {
    int  result,j,k;
    float scale;
    char buf[128];
    char *name = (char *)xv_get(item, PANEL_VALUE);
    Frame frame = (Frame)xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
    scale=atof(name);
printf("modify_selected_line  select_ptr %d pick_hight %d pick_width %d scale %f\n",

                                select_ptr,pick_hight,pick_width,scale);
    sprintf(buf, " modify a selected line by scale ");
    xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
    if ((int)xv_get(subframe_t, FRAME_CMD_PUSHPIN_IN) == FALSE)
    xv_set(subframe_t, XV_SHOW, FALSE, NULL);
    if(LC_Tree[select_ptr][1]<=2)
      {
        if(pick_hight>=0)
              {
              H_and_W[select_ptr][0][pick_hight]*=scale;
              Max_Min_Box[select_ptr][3]=0;
              for(k=0;k<LC_Tree[select_ptr][5];k++)
                    Max_Min_Box[select_ptr][3]+= H_and_W[select_ptr][0][k];
              }
        else
              {
              LC_Tree[select_ptr][4]*=scale;
              Max_Min_Box[select_ptr][1]= LC_Tree[select_ptr][4];
              }

printf(" verticle tree modified : pick_hight %d pick_width %d scale %f \n",pick_hight,pick_width,scale);
      }
    else
      {
       if(pick_width>=0)
              {
                if(pick_width==0 && H_and_W[select_ptr][0][0]<0)
                {
                 result= H_and_W[select_ptr][0][0];
                 for(k=1;k<LC_Tree[select_ptr][5];k++)
                   result+= H_and_W[select_ptr][0][k];
                 H_and_W[select_ptr][0][0]= H_and_W[select_ptr][0][0]-result +result*scale;
                 printf(" h_w[0][0] modified : result %d pick_width %d scale %f
\n",result,pick_width,scale);

                }
                else
                 H_and_W[select_ptr][0][pick_width]*=scale;
               Max_Min_Box[select_ptr][0]=H_and_W[select_ptr][0][0];
               if(Max_Min_Box[select_ptr][0]>=0)
                 {
                  Max_Min_Box[select_ptr][0]=0;
                  Max_Min_Box[select_ptr][1]=Max_Min_Box[select_ptr][0];
                  for(k=0;k<LC_Tree[select_ptr][5];k++)
                    Max_Min_Box[select_ptr][1]+= H_and_W[select_ptr][0][k];
                 }
               else
                 {
                  Max_Min_Box[select_ptr][1]=Max_Min_Box[select_ptr][0];
                  for(k=1;k<LC_Tree[select_ptr][5];k++)
                    Max_Min_Box[select_ptr][1]+= H_and_W[select_ptr][0][k];
                  if(Max_Min_Box[select_ptr][1]<0)
                    Max_Min_Box[select_ptr][1]=0;
                 }



              }
       else if(pick_hight>=0 && pick_hight<100)
              {
               LC_Tree[select_ptr][4]*=scale;
               Max_Min_Box[select_ptr][3]= 2*LC_Tree[select_ptr][4];
              }
       else
          {

               LC_Tree[select_ptr][2]+=scale;
               H_and_W[select_ptr][0][0]-=scale;
               Max_Min_Box[select_ptr][0]=H_and_W[select_ptr][0][0];
               if(Max_Min_Box[select_ptr][0]>=0)
                 {
                  Max_Min_Box[select_ptr][0]=0;
                  Max_Min_Box[select_ptr][1]=Max_Min_Box[select_ptr][0];
```

```
            for(k=0;k<LC_Tree[select_ptr][5];k++)
              Max_Min_Box[select_ptr][1]+= H_and_W[select_ptr][0][k];
            }
          else
            {
            Max_Min_Box[select_ptr][1]=Max_Min_Box[select_ptr][0];
            for(k=1;k<LC_Tree[select_ptr][5];k++)
              Max_Min_Box[select_ptr][1]+= H_and_W[select_ptr][0][k];
            if(Max_Min_Box[select_ptr][1]<0)
              Max_Min_Box[select_ptr][1]=0;
            }
        }
printf(" horizontal tree modified: pick_hight %d pick_width %d scale %f \n",pick_hight,pick_width,scale);
      }
    return PANEL_NONE;

  }




int  link_subtree_to_Verticle_tree_branch()
  {
  int k,x,y,link;
  link=0;
  x=LC_Tree[select_ptr][2]+LC_Tree[select_ptr][4];
  y=LC_Tree[select_ptr][3];
  printf("xo=%d,yo=%d,\n", LC_Tree[select_ptr][2],LC_Tree[select_ptr][3]);
  for(k=0;k<=pick_width;k++)
        {
         y+=H_and_W[select_ptr][0][k];
         printf("x=%d,Y=%d\n",x,y);
        }
  for(k=0;k<=draw_tree_ptr;k++)
    {
    if(LC_Tree[k][2]>=x && LC_Tree[k][2]<= x+25 && LC_Tree[k][3]>=y && LC_Tree[k][3]<=y +25)
        {
        link=1;
        H_and_W[select_ptr][2][pick_width]=k;
        LC_Tree[k][6]=select_ptr;
        printf(" link subtree [%d] to Verticle_tree [%d]'s branch [%d] \n",k,select_ptr,pick_width);
        select_ptr=k;
        break;
        }
```

```
      }
    return(link);
  }


int  link_subtree_to_Horizontal_tree_branch()

    {
    int k,x,y,link;
    link=0;
    x=LC_Tree[select_ptr][2];
    y=LC_Tree[select_ptr][3]+2*LC_Tree[select_ptr][4];
    printf("xo=%d,yo=%d,\n", LC_Tree[select_ptr][2],LC_Tree[select_ptr][3]);
    for(k=0;k<=pick_hight;k++)
                {
                x+=H_and_W[select_ptr][0][k];
                printf("x=%d,Y=%d\n",x,y);
                }

  for(k=0;k<=draw_tree_ptr;k++)
    {
     if(LC_Tree[k][2]>=x-15 && LC_Tree[k][2]<= x+15 && LC_Tree[k][3]>=y && LC_Tree[k][3]<=y
+20)
        {
        link=1;
        H_and_W[select_ptr][2][pick_hight]=k;
        LC_Tree[k][6]=select_ptr;
        printf(" link subtree [%d] to horizontal_tree [%d]'s branch [%d] \n",k,select_ptr,pick_hight);
        select_ptr=k;
        break;
        }
    }
    return(link);
  }

int text_rect_width(text_ptr)
char *text_ptr;
  {
    Panel_item item;
    Rect      *item_rect;
    int       width;
    item= xv_create(panel_s, PANEL_BUTTON,PANEL_LABEL_STRING, text_ptr, NULL);
    item_rect = (Rect *)xv_get(item, XV_RECT);
    width=item_rect->r_width;
    xv_destroy(item);
    return(width);
```

```
}


int
go_to_design_selected(item, event)
Panel_item item;
Event *event;
{
  Panel_item button1;
  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  xv_set(frame_s, XV_SHOW, FALSE, NULL);

  PANEL_EACH_ITEM(panel,button1)

        if   (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO DESIGN"))
                      xv_set(button1, XV_SHOW, FALSE, NULL);
        else if (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                      ;
        else          xv_set(button1, PANEL_INACTIVE,FALSE, NULL);
        PANEL_END_EACH

  PANEL_EACH_ITEM(panel_s,button1)
  xv_destroy(button1);
  PANEL_END_EACH

  design_state=1;


  return XV_OK;
}
int panel_s_button_selected(item, event)

 Panel_item item;
 Event *event;

 {

   printf("menu button %s selected...\n", xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
 }


int simulation_selected(item, event)
Panel_item item;
Event *event;
{
  Panel_item button1;
  int      result;
  int      create_simulate_menubar();
  void      defult_submenubar_simulation();

  char     *txt_ptr;
  printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));

  if(select_ptr<0)
     {
     result = notice_prompt(panel, NULL,
     NOTICE_FOCUS_XY,      t_x, t_y,
     NOTICE_MESSAGE_STRINGS, "A menubar which will be simulated",
                  "must be selected before "."           ",
                  "Select O K to continue.", NULL,
     NOTICE_BUTTON_YES,     "O K",
     NULL);
     return XV_OK;

     }

 else if( LC_Tree[select_ptr][1]==0 || LC_Tree[select_ptr][1]==3 )

     {
     PANEL_EACH_ITEM(panel,button1)

       if   (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO DESIGN"))
                 xv_set(button1, XV_SHOW, TRUE, NULL);
       else if (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                 ;
       else           xv_set(button1, PANEL_INACTIVE, TRUE, NULL);

     PANEL_END_EACH
     design_state=0;
     sub_menubar_ptr=-1;
     active_submenubar_ptr=-1;
     simulating_ptr=select_ptr;
     xv_set(frame_s, FRAME_LABEL, "Menu interface simulation",NULL);

     if(LC_Tree[simulating_ptr][7]>=0)
        {
        txt_ptr=&Textstring[LC_Tree[simulating_ptr][7]][0];
        xv_set(frame_s, FRAME_LABEL,txt_ptr,NULL);
        }
        xv_set(frame_s, XV_SHOW, TRUE, NULL);
        create_simulate_menubar( simulating_ptr);
        defult_submenubar_simulation();
        return XV_OK;

      }

  else
      {
        result = notice_prompt(panel, NULL,
        NOTICE_FOCUS_XY,       t_x, t_y,
        NOTICE_MESSAGE_STRINGS, "A menubar which will be simulated",
                    "must be selected before ","          ",
                    "Select O K to continue.", NULL,
        NOTICE_BUTTON_YES,     "O K",
        NULL);
        return XV_OK;
        }

}

/* defult submenubar simulation*/

void defult_submenubar_simulation()
{

  if(sub_menubar_ptr>=0)
    {

        active_submenubar_ptr++;
        active_submenubar_tree[active_submenubar_ptr]=sub_menubar_data[0][1];
                                            create_simulate_menubar(sub_menubar_data[0][1]);
                          }
}
/*****************************/

int create_simulate_menubar(simulate_ptr)

int simulate_ptr;

     {
       int  create_simulate_menubar_item();
       int  k,result;
       char *txt_ptr;
       int  virtual_item();
```

```
      Panel_item button1;

      for(k=0;k<LC_Tree[simulate_ptr][5];k++)
        {
             if(H_and_W[simulate_ptr][1][k]>=0)
          {
           txt_ptr=&Textstring[H_and_W[simulate_ptr][1][k]][0];
           if(virtual_item(txt_ptr)==0)

             create_simulate_menubar_item(txt_ptr,simulate_ptr,k);

           else if(virtual_item(txt_ptr)==2 || virtual_item(txt_ptr)==1)
             {
              if(H_and_W[simulate_ptr][2][k]>=0)
                {
if( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==0 || LC_Tree[H_and_W[simulate_ptr][2][k]][1]==3 )
                  create_simulate_menubar(H_and_W[simulate_ptr][2][k]);
                else
                   {
                    result = notice_prompt(panel, NULL,
                    NOTICE_FOCUS_XY,       t_x, t_y,
NOTICE_MESSAGE_STRINGS, "the subtree of virtual menubar item is not a menubar !",
                              "select  back to design, return to design mode",
                              "current subtree becomes selected tree",
                              "select  change it into suitable menubar, change current",
                              "type of subtree of virtual menubar item into a menubar type",
                               NULL,
                    NOTICE_BUTTON_YES,     "back to design",
                    NOTICE_BUTTON_NO,      "change it into suitable menubar",
                    NULL);

                    if (result == NOTICE_YES)
                      {
                       select_ptr=H_and_W[simulate_ptr][2][k];

                       printf("back tree %d to design button selected..\n",select_ptr);
                       xv_set(frame_s, XV_SHOW, FALSE, NULL);

                       PANEL_EACH_ITEM(panel,button1)

                       if   (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO
DESIGN"))
                      xv_set(button1, XV_SHOW, FALSE, NULL);
                      else if  (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                      ;
                      else          xv_set(button1, PANEL_INACTIVE,FALSE, NULL);
```

```
                        PANEL_END_EACH
                        PANEL_EACH_ITEM(panel_s,button1)
                        xv_destroy(button1);
                        PANEL_END_EACH
                        }

                      else if (result == NOTICE_NO)
                        {
                         select_ptr=H_and_W[simulate_ptr][2][k];
                         printf("change tree %d into suitable menubar button selected..\n",select_ptr);
if ( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==1 || LC_Tree[H_and_W[simulate_ptr][2][k]][1]==2)
                              LC_Tree[H_and_W[simulate_ptr][2][k]][1]=0;
                         else if( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==4)
                              LC_Tree[H_and_W[simulate_ptr][2][k]][1]=3;
                         create_simulate_menubar(H_and_W[simulate_ptr][2][k]);
                        }
                      }
                    }


                 }
              }

           }
        return;
        }

int virtual_item(t)
 char *t;
  {

    if(*t=='{')
      {
       while( *t != '\0')
           t++;
       if(*(t-1)=='}')
         return(2);

       else
         return(1);
      }
    else

       return(0);

  }
```

```
int create_simulate_menubar_item(item_name,simulate_ptr,k)

char *item_name;
int simulate_ptr;
int  k;

{
  Panel_item  pi,button1;
  Menu      simulate_menu,create_simulate_menus();
  int       panel_s_button_selected();
  int       result;
  /* loop thru all panel items and check for item with same name */

  PANEL_EACH_ITEM(panel_s, pi)

    if (!strcmp(item_name, (char *)xv_get(pi, PANEL_LABEL_STRING))) return PANEL_NONE;

  PANEL_END_EACH


    pi=(Panel_item) xv_create(panel_s,  PANEL_BUTTON,
    PANEL_LABEL_STRING,    item_name,
    PANEL_NOTIFY_PROC,     panel_s_button_selected,
    NULL);

   if(H_and_W[simulate_ptr][2][k]>=0)
        {

        if( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==0 ||
LC_Tree[H_and_W[simulate_ptr][2][k]][1]==3 )
             {
               result = notice_prompt(panel, NULL,
               NOTICE_FOCUS_XY,      t_x, t_y,
               NOTICE_MESSAGE_STRINGS, "the subtree of real menubar item is a menubar !",
                                 "select  back to design, return to design mode",
                                 "current subtree becomes selected tree",
                                 "select  change it into suitable menu, change current",
                                 "type of subtree of real menubar item into a menu type",
                                  NULL,
               NOTICE_BUTTON_YES,     "back to design",
               NOTICE_BUTTON_NO,      "change it into suitable menutree",
               NULL);
               if (result == NOTICE_YES)
                      {
                        select_ptr=H_and_W[simulate_ptr][2][k];
```

```
                    printf("back tree %d to design button selected..\n",select_ptr);
                    xv_set(frame_s, XV_SHOW, FALSE, NULL);

                    PANEL_EACH_ITEM(panel,button1)

                    if    (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO
DESIGN"))
                    xv_set(button1, XV_SHOW, FALSE, NULL);
                    else if (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                      ;
                    else          xv_set(button1, PANEL_INACTIVE,FALSE, NULL);
                    PANEL_END_EACH

                    PANEL_EACH_ITEM(panel_s,button1)
                    xv_destroy(button1);
                    PANEL_END_EACH
                  }

              else if (result == NOTICE_NO)
                  {
                    select_ptr=H_and_W[simulate_ptr][2][k];
                    printf("change tree %d into suitable menutree button selected..\n",select_ptr);
                    if ( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==0)
                       LC_Tree[H_and_W[simulate_ptr][2][k]][1]=1;
                    else  if( LC_Tree[H_and_W[simulate_ptr][2][k]][1]==3)
                          LC_Tree[H_and_W[simulate_ptr][2][k]][1]=4;
simulate_menu  = create_simulate_menus(H_and_W[simulate_ptr][1][k],H_and_W[simulate_ptr][2][k]);
                    if( simulate_menu != NULL)
                    xv_set(pi,  PANEL_ITEM_MENU, simulate_menu, NULL);
                   }
              }
        else
              {
simulate_menu  = create_simulate_menus(H_and_W[simulate_ptr][1][k],H_and_W[simulate_ptr][2][k]);
              if( simulate_menu != NULL)
              xv_set(pi,  PANEL_ITEM_MENU, simulate_menu, NULL);
              }
            }
    return PANEL_NEXT;
}

Menu create_simulate_menus(menu_root_title_ptr,menu_ptr)

int menu_root_title_ptr;
int menu_ptr;
{
   Menu_item          mi,create_simulate_menu_item();
   Menu                 menu;
   int              k,has_menu,result;
   char              *txt_ptr;
   int              create_virtual_menus();
   Panel_item     button1;
          if( LC_Tree[menu_ptr][1]==0 || LC_Tree[menu_ptr][1]==3 )
          /* sub menubar */
                        {

   for(k=0;k<=sub_menubar_ptr;k++)
          if( sub_menubar_data[k][1]==menu_ptr)

return NULL;
                                   sub_menubar_ptr++;
   sub_menubar_data[sub_menubar_ptr][0]= menu_root_title_ptr;
   sub_menubar_data[sub_menubar_ptr][1]= menu_ptr;
printf("sub_menubar_ptr=%d,menu_root_title_ptr=%d,menu_ptr=%d\n",sub_menubar_ptr,


          menu_root_title_ptr,menu_ptr );
   return NULL;
  }

          has_menu=0;
 menu = (Menu)xv_create(XV_NULL, MENU, NULL);
 for(k=0;k<LC_Tree[menu_ptr][5];k++)
   {
      if(H_and_W[menu_ptr][1][k]>=0)
        {
         txt_ptr=&Textstring[H_and_W[menu_ptr][1][k]][0];
         if(virtual_item(txt_ptr)==0)
           {
            has_menu++;
            mi=create_simulate_menu_item(txt_ptr,menu_ptr,k);
            xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
           }
         else  if(virtual_item(txt_ptr)==2 || virtual_item(txt_ptr)==1)
            {
            if(H_and_W[menu_ptr][2][k]>=0)
              {
 if( LC_Tree[H_and_W[menu_ptr][2][k]][1]==1 || LC_Tree[H_and_W[menu_ptr][2][k]][1]==2 || LC_Tree
                                                     [H_and_W[menu_ptr][2][k]][1]==4 )
                       has_menu= create_virtual_menus(H_and_W[menu_ptr][2][k],menu);
```

```
                else
                    {
                     result = notice_prompt(panel, NULL,
                     NOTICE_FOCUS_XY,      t_x, t_y,
                     NOTICE_MESSAGE_STRINGS, "the subtree of virtual menu item is not a menutree
!",
                                   "select  back to design, return to design mode",
                                   "current subtree becomes selected tree",
                                   "select  change it into suitable menutree, change current",
                                   "type of subtree of virtual menu item into a menutree type",
                                    NULL,
                     NOTICE_BUTTON_YES,     "back to design",
                     NOTICE_BUTTON_NO,      "change it into suitable menubar",
                     NULL);

                     if (result == NOTICE_YES)
                        {
                         select_ptr=H_and_W[menu_ptr][2][k];

                         printf("back tree %d to design button selected..\n",select_ptr);
                         xv_set(frame_s, XV_SHOW, FALSE, NULL);

                         PANEL_EACH_ITEM(panel,button1)

 if    (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO DESIGN"))
                         xv_set(button1, XV_SHOW, FALSE, NULL);
                         else if  (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                         ;

                        else             xv_set(button1, PANEL_INACTIVE,FALSE, NULL);

                        PANEL_END_EACH

                        PANEL_EACH_ITEM(panel_s,button1)
                        xv_destroy(button1);
                        PANEL_END_EACH
                        }

                     else if (result == NOTICE_NO)
                        {
                          select_ptr=H_and_W[menu_ptr][2][k];
                          printf("change tree %d into suitable menutree button selected..\n",select_ptr);
                          if ( LC_Tree[H_and_W[menu_ptr][2][k]][1]==0 )
                               LC_Tree[H_and_W[menu_ptr][2][k]][1]=1;
                          else  if( LC_Tree[H_and_W[menu_ptr][2][k]][1]==3)
                               LC_Tree[H_and_W[menu_ptr][2][k]][1]=4;
                          has_menu= create_virtual_menus(H_and_W[menu_ptr][2][k],menu);
                         }
                       }
                     }

                 }
             }
}

  if( has_menu==0)
      {
      xv_destroy(menu);
      return NULL;
      }

  else
      return menu;

  }


int create_virtual_menus(menu_ptr,menu)
int     menu_ptr;
Menu    menu;

   {
    int  k,result;
    char *txt_ptr;
    int  has_menu;
    int  virtual_item();
    int  create_virtual_menus();
    Panel_item      button1;
    Menu_item           mi;

    has_menu=0;
    for(k=0;k<LC_Tree[menu_ptr][5];k++)
        {
         if(H_and_W[menu_ptr][1][k]>=0)
            {

             txt_ptr=&Textstring[H_and_W[menu_ptr][1][k]][0];
             if(virtual_item(txt_ptr)==0)
                {
                 has_menu=1;
            mi=create_simulate_menu_item(txt_ptr,menu_ptr,k);
            xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
            }
        else  if(virtual_item(txt_ptr)==2 || virtual_item(txt_ptr)==1)
            {
            if(H_and_W[menu_ptr][2][k]>=0)
             {
             if( LC_Tree[H_and_W[menu_ptr][2][k]][1]==1 ||
LC_Tree[H_and_W[menu_ptr][2][k]][1]==2 || LC_Tree[H_and_W[menu_ptr][2][k]][1]==4 )
                  has_menu= create_virtual_menus(H_and_W[menu_ptr][2][k],menu);
              else
                 {
                  result = notice_prompt(panel, NULL,
                  NOTICE_FOCUS_XY,      t_x, t_y,
                  NOTICE_MESSAGE_STRINGS, "the subtree of virtual menu item is not a menutree
!",
                              "select  back to design, return to design mode",
                              "current subtree becomes selected tree",
                              "select  change it into suitable menutree, change current",
                              "type of subtree of virtual menu item into a menutree type",
                               NULL,
                  NOTICE_BUTTON_YES,      "back to design",
                  NOTICE_BUTTON_NO,       "change it into suitable menubar",
                  NULL);

                  if (result == NOTICE_YES)
                     {
                     select_ptr=H_and_W[menu_ptr][2][k];

                     printf("back tree %d to design button selected..\n",select_ptr);
                     xv_set(frame_s, XV_SHOW, FALSE, NULL);

                     PANEL_EACH_ITEM(panel,button1)

                     if   (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "GO TO
DESIGN"))
                     xv_set(button1, XV_SHOW, FALSE, NULL);
                     else if  (!strcmp( (char *)xv_get(button1, PANEL_LABEL_STRING), "HELP"))
                     ;
                    else          xv_set(button1, PANEL_INACTIVE,FALSE, NULL);
                    PANEL_END_EACH
                    PANEL_EACH_ITEM(panel_s,button1)
                    xv_destroy(button1);
                    PANEL_END_EACH
                     }
```

```
                           else if (result == NOTICE_NO)
                              {
                               select_ptr=H_and_W[menu_ptr][2][k];
                               printf("change tree %d into suitable menutree button selected..\n",select_ptr);
                               if ( LC_Tree[H_and_W[menu_ptr][2][k]][1]==0 )
                                   LC_Tree[H_and_W[menu_ptr][2][k]][1]=1;
                               else  if( LC_Tree[H_and_W[menu_ptr][2][k]][1]==3)
                                   LC_Tree[H_and_W[menu_ptr][2][k]][1]=4;
                               has_menu=  create_virtual_menus(H_and_W[menu_ptr][2][k],menu);
                              }
                         }
                      }
                  }
              }
          }
   return has_menu;

    }

Menu_item create_simulate_menu_item(txt_ptr,menu_ptr,k)

  char  *txt_ptr;
  int  menu_ptr;
  int  k;


{
Menu_item          mi;
Menu              menu ,create_simulate_menus();
void              simulate_menu_action_proc();

        mi = xv_create(XV_NULL, MENUITEM,
        MENU_STRING,      txt_ptr,
        MENU_RELEASE,
        MENU_RELEASE_IMAGE,
        MENU_NOTIFY_PROC,  simulate_menu_action_proc,
        NULL);
     if(H_and_W[menu_ptr][2][k]>=0)
            {
             menu  = create_simulate_menus(H_and_W[menu_ptr][1][k],H_and_W[menu_ptr][2][k]);
             if( menu != NULL)
                     xv_set(mi, MENU_PULLRIGHT,menu, NULL);
            }
return mi;
 }
```

```
void simulate_menu_action_proc(menu, menu_item)

Menu menu;
Menu_item menu_item;

{
  int  j,k,l;
  int  parent_tree_ptr;
  int  is_menu_item=1;
  int  is_active_submenubar;
  int  is_inactive_submenubar;
  int  temp_active_submenubar_ptr;
  int  temp_active_submenubar_tree[15];
  int  inactive_submenubar_ptr;
  int  inactive_submenubar_tree[20];
  void   destroy_inactive_submenubar();
  char   *txt_ptr;


  if(sub_menubar_ptr>=0)
    {
    for(k=0;k<=sub_menubar_ptr;k++)
        {
        txt_ptr=&Textstring[sub_menubar_data[k][0]][0];

        if (!strcmp((char *)xv_get(menu_item, MENU_STRING),txt_ptr))
                {
                  is_menu_item=0;
                  is_active_submenubar=0;
                  if(active_submenubar_ptr>=0)
                    {
                    for(j=0;j<=active_submenubar_ptr;j++)
                      {
                        if(active_submenubar_tree[j]==sub_menubar_data[k][1])
                          {
                            is_active_submenubar=1;
                            printf("Menu Item: %s selected,tree %d is a active submenubar ! \n",
                                  xv_get(menu_item, MENU_STRING),active_submenubar_tree[j]);
                            break;
                          }

                      }


                    }
```

```
              if( is_active_submenubar==0)
                  {
                          parent_tree_ptr= LC_Tree[sub_menubar_data[k][1]][6];

                          temp_active_submenubar_ptr=-1;
                          inactive_submenubar_ptr=-1;

                          while( parent_tree_ptr!=simulating_ptr)

                              {
                              if( LC_Tree[ parent_tree_ptr][1]==0 || LC_Tree[ parent_tree_ptr][1]==3 )
                                  {
                                   printf("parent_tree %d \n", parent_tree_ptr);
                                   temp_active_submenubar_ptr++;
                                   temp_active_submenubar_tree[temp_active_submenubar_ptr]=
parent_tree_ptr;
                                  }

                              parent_tree_ptr= LC_Tree[ parent_tree_ptr][6];
                              }

                          for(j=0;j<=active_submenubar_ptr;j++)
                          printf("active submenubar tree %d\n",active_submenubar_tree[j]);
                          for(j=0;j<=temp_active_submenubar_ptr;j++)
                          printf("temp_active submenubar tree %d\n",temp_active_submenubar_tree[j]);

          for(j=0;j<=active_submenubar_ptr;j++)
            {
             is_inactive_submenubar=1;
             for(l=0;l<=temp_active_submenubar_ptr;l++)
                if( active_submenubar_tree[j]== temp_active_submenubar_tree[l])
                    {
                     is_inactive_submenubar=0;
                     break;
                    }
             if( is_inactive_submenubar==1)
                 {
                  inactive_submenubar_ptr++;
                  inactive_submenubar_tree[inactive_submenubar_ptr]=active_submenubar_tree[j];
                 }
            }
          for(j=0;j<=inactive_submenubar_ptr;j++)
          printf("inactive submenubar tree %d\n",inactive_submenubar_tree[j]);

          for(j=0;j<=inactive_submenubar_ptr;j++)
                destroy_inactive_submenubar(inactive_submenubar_tree[j]);
```

```
        for(j=0;j<=temp_active_submenubar_ptr;j++)
              active_submenubar_tree[j]== temp_active_submenubar_tree[j];
        active_submenubar_ptr=temp_active_submenubar_ptr;

        active_submenubar_ptr++;
        active_submenubar_tree[active_submenubar_ptr]=sub_menubar_data[k][1];
        printf("active_submenubar_ptr=%d,active_submenubar_tree=%d\n",
                   active_submenubar_ptr,sub_menubar_data[k][1]);
        create_simulate_menubar( sub_menubar_data[k][1]);
        printf("Menu Item: %s selected,create submenubar for %s \n",
             xv_get(menu_item, MENU_STRING),txt_ptr);
                      break;
                    }
               }
        }

    }

if(is_menu_item==1)
printf("Menu Item: %s selected \n", xv_get(menu_item, MENU_STRING));


}

void destroy_inactive_submenubar(inactive_submenubar_tree_ptr)
            int inactive_submenubar_tree_ptr;
            {
                        int                         k;
                        char          *txt_ptr;
                        Panel_itempi;

for(k=0;k<LC_Tree[inactive_submenubar_tree_ptr][5];k++)
                        {
                        if(H_and_W[inactive_submenubar_tree_ptr][1][k]>=0)
                        {
                        txt_ptr=&Textstring[H_and_W[inactive_submenubar_tree_ptr][1][k]][0];
                        PANEL_EACH_ITEM(panel_s, pi)
                        if (!strcmp(txt_ptr, (char *)xv_get(pi, PANEL_LABEL_STRING)))    xv_destroy(pi);
                                    PANEL_END_EACH
                                    printf("destroy menu button %s\n",txt_ptr);
                        }
            }
}

/**** The end of Elc *****/
```

```
/* Slc -- The User Interface Simulator    */

#include "/home/sis-server/student/JGu/imple/include/lc_xview.h"
#include "/home/sis-server/student/JGu/imple/include/lc_basic.h"
/*#include "/home/sis-server/student/JGu/imple/include/lc_app.h" */

main(argc, argv)
int argc;
char *argv[];

{
int         draw_tree_ptr;
int         Basic_gen_get_interface_specification_file();
int         i;
void        Basic_gen_menu_user_interface_simulation();
int         simulating_ptr;

if(argc != 2)
     {
     printf(" Input format error, stop(1)\n");
     exit(1);
     }
   else
     {
     if((draw_tree_ptr=Basic_gen_get_interface_specification_file(argv[1]))<0)
        {
        printf("nill L C Tree , stop(2)\n");
        exit(2);
        }
}

simulating_ptr=-100;

for(i=0;i<=draw_tree_ptr;i++)
   if(( Basic_gen_LC_Tree[i][1]==0 || Basic_gen_LC_Tree[i][1]==3 )
                                                   &&( Basic_gen_LC_Tree[i][6]<0))
           {
             simulating_ptr=i;
             break;
           }
   if(simulating_ptr<0)
        {
        printf("no menubar can be simulated ! stop(3)\n");
        exit(0);
```

```
        }
  xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
   Basic_gen_frame = (Frame)xv_create(XV_NULL, FRAME,
    XV_WIDTH,           800,
    XV_HEIGHT,          500,
    FRAME_LABEL,        argv[0],
    NULL);

  Basic_gen_panel = (Panel)xv_create(Basic_gen_frame, PANEL,
        PANEL_LAYOUT, PANEL_HORIZONTAL,
             XV_X, 0,
                  XV_Y, 0,
                  /*XV_WIDTH, 400,*/
             XV_WIDTH, WIN_EXTEND_TO_EDGE,
                  XV_HEIGHT, 25,
                  WIN_BORDER, FALSE,
        NULL);

  Basic_gen_canvas = (Canvas)xv_create(Basic_gen_frame, CANVAS,
          XV_X, 0,
          XV_Y, 25,
    CANVAS_WIDTH,        1600,
    CANVAS_HEIGHT,       1500,
    CANVAS_AUTO_SHRINK,   FALSE,
    CANVAS_AUTO_EXPAND,   FALSE,
    CANVAS_REPAINT_PROC,   Basic_gen_canvas_repaint_proc,
    CANVAS_X_PAINT_WINDOW, TRUE,

    NULL);

xv_set(canvas_paint_window(Basic_gen_canvas),
          WIN_CONSUME_EVENTS,
          WIN_NO_EVENTS,
          WIN_ASCII_EVENTS, KBD_USE, KBD_DONE,
          LOC_DRAG, /*LOC_WINENTER, LOC_WINEXIT,
          */ WIN_MOUSE_BUTTONS,
          NULL,
          WIN_EVENT_PROC, Basic_gen_events,
          NULL);

  xv_set(Basic_gen_canvas,
    OPENWIN_SPLIT,
      OPENWIN_SPLIT_INIT_PROC,   Basic_gen_init_split,
      OPENWIN_SPLIT_DESTROY_PROC, Basic_gen_join_split,
      NULL,
    NULL);

   xv_create(Basic_gen_canvas, SCROLLBAR,
    SCROLLBAR_SPLITTABLE,  TRUE,
    SCROLLBAR_DIRECTION,   SCROLLBAR_VERTICAL,
    NULL);
  xv_create(Basic_gen_canvas, SCROLLBAR,
    SCROLLBAR_SPLITTABLE,  TRUE,
    SCROLLBAR_DIRECTION,   SCROLLBAR_HORIZONTAL,
    NULL);


        Basic_gen_menu_user_interface_simulation(simulating_ptr);

  window_fit(Basic_gen_frame);
  xv_main_loop(Basic_gen_frame);
  exit(0);
}



void
Basic_gen_canvas_repaint_proc(canvas, paint_window, dpy, xwin, xrects)
Canvas      canvas;        /* unused */
Xv_Window   paint_window;  /* unused */
Display     *dpy;
Window      xwin;
Xv_xrectlist *xrects;      /* unused */
{

 void draw_application_data_on_canvas();
/* GC gc;
 int j;

 gc = DefaultGC(dpy, DefaultScreen(dpy));
 XClearWindow(dpy, xwin);*/

 /* call real draw subroutines of application*/

   draw_application_data_on_canvas(canvas, paint_window, dpy, xwin, xrects);
/* XDrawString(dpy, xwin, gc, 25, 45, Basic_gen_msg, strlen(Basic_gen_msg));*/
}

void
Basic_gen_events(pw, event)
Xv_Window pw;
Event *event;
{
int j,k, result;
int event_ACTION_SELECT_proc();
int event_ACTION_MENU_proc();

 register char *p = Basic_gen_msg;

  int code = event_action(event);
  Xv_Window view;
  int i = (int)xv_get(Basic_gen_canvas, OPENWIN_NVIEWS);

  *p = 0;
  /* Not interested in button up Basic_gen_events */
 /* if (win_inputnegevent(event))
     return;*/


  /* test to see if a function key has been hit */
  if (event_is_key_left(event))
     sprintf(p, "(L%d) ", event_id(event) - KEY_LEFTFIRST + 1);
  else if (event_is_key_top(event))
     sprintf(p, "(T%d) ", event_id(event) - KEY_TOPFIRST + 1);
  else if (event_is_key_right(event))
     sprintf(p, "(R%d) ", event_id(event) - KEY_RIGHTFIRST + 1);
  else if (event_id(event) == KEY_BOTTOMLEFT)
     strcpy(p, "bottom left ");
  else if (event_id(event) == KEY_BOTTOMRIGHT)
     strcpy(p, "bottom right ");
 p += strlen(p);
  /* Determine which paint window this event happened in. */
  while (pw != xv_get(Basic_gen_canvas, CANVAS_NTH_PAINT_WINDOW, --i) && i > 0)
     ;
  /* The paint window number is "i" -- get the "i"th view window */
  view = xv_get(Basic_gen_canvas, OPENWIN_NTH_VIEW, i);


 if (event_is_ascii(event))

    {
    /*
     * note that shift modifier is reflected in the event code by
     * virtue of the char printed is upper/lower case.
     */
    sprintf(p, "Keyboard: key '%c' (%d) %s at %d,%d",
      event_action(event), event_action(event),
      event_is_down(event)? "pressed" : "released",
      event_x(event), event_y(event));
    }
  else switch (event_action(event))
   {
    case ACTION_CLOSE :
      xv_set(Basic_gen_frame, FRAME_CLOSED, TRUE, NULL);
      break;
    case ACTION_OPEN :
      strcpy(p, "frame opened up");
      break;
    case ACTION_HELP :
      strcpy(p, "Help (action ignored)");
      break;
    case ACTION_SELECT :
          event_ACTION_SELECT_proc( event_x(event),event_y(event));
      sprintf(p, "Button: Select (Left) %s at %d,%d",
        event_is_down(event)? "pressed" : "released",
        event_x(event), event_y(event));
      break;
    case ACTION_ADJUST :
      sprintf(p, "Button: Adjust (Middle) %s at %d,%d",
        event_is_down(event)? "pressed" : "released",
        event_x(event), event_y(event));
      break;
    case ACTION_MENU :

     if(event_is_down(event))
          {
           event_ACTION_MENU_proc(event_x(event), event_y(event));
           sprintf(p, "Button: Menu (Right) %s at %d,%d",
                   "pressed",event_x(event), event_y(event));
           break;
          }
     else
          {
          sprintf(p, "Button: Menu (Right) %s at %d,%d",
          event_is_down(event)? "pressed" : "released",
          event_x(event), event_y(event));
          break;
          }
    case SHIFT_RIGHT :
      sprintf(p, "Keyboard: right shift %s",
```

```
        event_is_down(event)? "pressed" : "released");
      break;
    case SHIFT_LEFT :
      sprintf(p, "Keyboard: left shift %s",
        event_is_down(event)? "pressed" : "released");
      break;
    case SHIFT_LEFTCTRL : case SHIFT_RIGHTCTRL :
      sprintf(p, "Keyboard: control key %s",
        event_is_down(event)? "pressed" : "released");
      break;
    case SHIFT_META :
      sprintf(p, "Keyboard: meta key %s",
        event_is_down(event)? "pressed" : "released");
      break;
    case SHIFT_ALT :
      sprintf(p, "Keyboard: alt key %s",
        event_is_down(event)? "pressed" : "released");
      break;
    case KBD_USE:
      sprintf(p, "Keyboard: got keyboard focus");
      break;
  /* case KBD_DONE:
      sprintf(p, "Keyboard: lost keyboard focus");
      break;   */
    case LOC_MOVE:
    sprintf(p, "Pointer: moved to %d,%d",
          event_x(event),event_y(event));
      break;
    case LOC_DRAG:
      sprintf(p, "Pointer: dragged to %d,%d",
          event_x(event), event_y(event));
      break;
/*    case LOC_WINENTER:
      win_set_kbd_focus(pw, xv_get(pw, XV_XID));
      sprintf(p, "Pointer: entered window at %d,%d",
          event_x(event), event_y(event));
      break;
    case LOC_WINEXIT:
      sprintf(p, "Pointer: exited window at %d,%d",
          event_x(event), event_y(event));
      break;   */
    case WIN_RESIZE :
    case WIN_REPAINT :
      return;
    default :
        /* There are too many ACTION Basic_gen_events to trap -- ignore the
         * ones we're not interested in.
         */
        return;
   }
   Basic_gen_canvas_repaint_proc(Basic_gen_canvas, pw,
      xv_get(Basic_gen_canvas, XV_DISPLAY), xv_get(pw, XV_XID), NULL);

  /* indicate which paint window and view window ID's */
 /* printf("win %x, view: %x\n", pw, view);*/
}



void
Basic_gen_join_split(view)
Xv_Window view;
{
   puts("joined view");
}



void
Basic_gen_init_split(oldview, newview, pos)
Xv_Window oldview, newview;
int pos;
{
   xv_set(xv_get(newview, CANVAS_VIEW_PAINT_WINDOW),
      WIN_EVENT_PROC,        Basic_gen_events,
      WIN_CONSUME_EVENT,     ACTION_SELECT, ACTION_ADJUST, NULL,
      NULL);
}



  int Basic_gen_get_interface_specification_file(filename)
  char *filename;

  {
   FILE *fp,*fopen();
   char *type;
   int result,i,j,k;
   char buf[128];
   char *sp;
```

```
int draw_tree_ptr=-1;
int text_ptr=-1;

int select_ptr=-100;
int pick_hight,pick_width;

if( Basic_gen_print_contral!=0)
                         printf("input specification file by file name  %s \n",filename);

 if((strlen(filename))<=0)
   {
    printf("File name is nill stop[2]\n");
    exit(2);
   }

 else

   {
    type="r";
 if( Basic_gen_print_contral!=0) printf("filename %s type %s\n",filename,type);
    if((fp=fopen(filename,type))==NULL)
        {
          printf("open file feiled,stop[3]\n");
          exit(3);
        }
    else
        {
          select_ptr=-1;
          pick_hight=-1;
          pick_width=-1;
          draw_tree_ptr=-1;
          text_ptr=-1;
          fscanf(fp,"%d\n",&draw_tree_ptr);
 if( Basic_gen_print_contral!=0) printf("draw_tree_ptr=%d\n",draw_tree_ptr);

              for(i=0;i<=draw_tree_ptr;i++)
                 {
                  for(j=0;j<=7;j++)
                      {
                       fscanf(fp,"%d ",&Basic_gen_LC_Tree[i][j]);
         if( Basic_gen_print_contral!=0) printf("Lc_tree[%d][%d]= %d\n",i,j,Basic_gen_LC_Tree[i][j]);
                      }
                  for(j=0;j<Basic_gen_LC_Tree[i][5];j++)
                      {
                      /* printf("input%d\n",j);*/
```

```
                        fscanf(fp,"%d %d %d",&Basic_gen_H_and_W[i][0][j],
                       &Basic_gen_H_and_W[i][1][j],&Basic_gen_H_and_W[i][2][j]);
                        if( Basic_gen_print_contral!=0)
printf("Basic_gen_H_and_W[%d][0][%d]=%d,Basic_gen_H_and_W[%d][1][%d]=%d,
                                                Basic_gen_H_and_W[%d][2][%d]=%d\n",

i,j,Basic_gen_H_and_W[i][0][j],i,j,Basic_gen_H_and_W[i][1][j],
                                                        i,j,Basic_gen_H_and_W[i][2][j]);
                        }
                    for(j=0;j<=3;j++)
                        {
                        fscanf(fp,"%d ",&Basic_gen_Max_Min_Box[i][j]);
  if( Basic_gen_print_contral!=0) printf("max_min_box[%d][%d]=%d\n",i,j,Basic_gen_Max_Min_Box[i][j]);

                        }
                    }
                fscanf(fp,"\n");
                fscanf(fp,"%d\n",&text_ptr);
                if( Basic_gen_print_contral!=0) printf("text_ptr=%d\n",text_ptr);

                if(text_ptr>=0)
                    {
                    for(k=0;k<=text_ptr;k++)
                        {
                        sp=&Basic_gen_Textstring[k][0];
                        fscanf(fp,"%s ",sp);
                        if( Basic_gen_print_contral!=0) printf("%D      %s\n",k,sp);
                        }
                    }
                fscanf(fp,"\n");
                fscanf(fp,"%d %d %d\n",&select_ptr,&pick_hight,&pick_width);
                if( Basic_gen_print_contral!=0) printf("select_ptr=%d,
                                                pick_hight=%d,pick_width=%d\n",
                                                select_ptr,pick_hight,pick_width);

                fclose(fp);
               }

        }
  return(draw_tree_ptr);
 }

void
Basic_gen_menu_user_interface_simulation(simulating_ptr)
int simulating_ptr;
```

```
{
  int      Basic_gen_create_simulate_menubar();
  void     Basic_gen_defult_submenubar_simulation();

  char     *txt_ptr;
  if( Basic_gen_print_contral!=0)
                        printf("menu user interface simulation %d \n", simulating_ptr);
      Basic_gen_sub_menubar_ptr=-1;
      Basic_gen_active_submenubar_ptr=-1;
      xv_set(Basic_gen_frame, FRAME_LABEL,
                                      "untitled Menu interface simulation",NULL);
      if(Basic_gen_LC_Tree[simulating_ptr][7]>=0)
          {
txt_ptr=&Basic_gen_Textstring[Basic_gen_LC_Tree[simulating_ptr][7]][0];
          xv_set(Basic_gen_frame, FRAME_LABEL,txt_ptr,NULL);
          }

      Basic_gen_create_simulate_menubar( simulating_ptr);
      Basic_gen_defult_submenubar_simulation();

}


void
Basic_gen_defult_submenubar_simulation()

{


  if(Basic_gen_sub_menubar_ptr>=0)
    {


        Basic_gen_active_submenubar_ptr++;
        Basic_gen_active_submenubar_tree[Basic_gen_active_submenubar_ptr]
                                                =Basic_gen_sub_menubar_data[0][1];
        Basic_gen_create_simulate_menubar( Basic_gen_sub_menubar_data[0][1]);


    }
}

int
Basic_gen_create_simulate_menubar(simulate_ptr)

int simulate_ptr;
```

```
    {
      int  Basic_gen_create_simulate_menubar_item();
      int  k,result;
      char  *txt_ptr;
      int  Basic_gen_virtual_item();
      Panel_item button1;

      for(k=0;k<Basic_gen_LC_Tree[simulate_ptr][5];k++)
        {

          if(Basic_gen_H_and_W[simulate_ptr][1][k]>=0)
            {

  txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[simulate_ptr][1][k]][0];
              if(Basic_gen_virtual_item(txt_ptr)==0)

  Basic_gen_create_simulate_menubar_item(txt_ptr,simulate_ptr,k);

 else if(Basic_gen_virtual_item(txt_ptr)==2 || Basic_gen_virtual_item(txt_ptr)==1)
                {
                  if(Basic_gen_H_and_W[simulate_ptr][2][k]>=0)
                    {
  if( Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==0 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==3 )
    Basic_gen_create_simulate_menubar(Basic_gen_H_and_W[simulate_ptr][2][k]);
                    else
                      {
            printf( "the subtree of virtual menubar item is not a menubar !\n");

                      }
                    }
                }
            }
        }
      return;
    }

int
Basic_gen_virtual_item(t)
char *t;
 {

   if(*t=='{')
     {
      while( *t != '\0')
         t++;
      if(*(t-1)=='}')
        return(2);
      else
        return(1);
      }
    else
      return(0);

 }



int
Basic_gen_create_simulate_menubar_item(item_name,simulate_ptr,k)

char *item_name;
int simulate_ptr;
int  k;


{
  Panel_item  pi,button1;
  Menu        simulate_menu,Basic_gen_create_simulate_menus();
  int         Basic_gen_panel_button_selected();
  int         result;
  /* loop thru all panel items and check for item with same name */

  PANEL_EACH_ITEM(Basic_gen_panel, pi)

    if (!strcmp(item_name, (char *)xv_get(pi, PANEL_LABEL_STRING)))
                                                          return PANEL_NONE;

  PANEL_END_EACH
    pi=(Panel_item) xv_create(Basic_gen_panel,  PANEL_BUTTON,
    PANEL_LABEL_STRING,     item_name,
    PANEL_NOTIFY_PROC,      Basic_gen_panel_button_selected,
    NULL);

    if(Basic_gen_H_and_W[simulate_ptr][2][k]>=0)
        {

          if( Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==0 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==3 )
                printf( "the subtree of real menubar item is a menubar !\n");
          else
```

```
            {
            simulate_menu = Basic_gen_create_simulate_menus
                                        (Basic_gen_H_and_W[simulate_ptr]
                                        [1][k],Basic_gen_H_and_W[simulate_ptr][2][k]);
              if( simulate_menu != NULL)
              xv_set(pi, PANEL_ITEM_MENU, simulate_menu, NULL);
            }
        }
    return PANEL_NEXT;
}


 Menu
 Basic_gen_create_simulate_menus(menu_root_title_ptr,menu_ptr)

 int menu_root_title_ptr;
 int menu_ptr;
  {
    Menu_item       mi,Basic_gen_create_simulate_menu_item();
    Menu           menu;
    int          k,has_menu,result;
    char          *txt_ptr;
    int          Basic_gen_create_virtual_menus();
    Panel_item     button1;


if( Basic_gen_LC_Tree[menu_ptr][1]==0 || Basic_gen_LC_Tree[menu_ptr][1]==3 )
     /* sub menubar */
    {
     for(k=0;k<=Basic_gen_sub_menubar_ptr;k++)
          if( Basic_gen_sub_menubar_data[k][1]==menu_ptr)
               return NULL;

Basic_gen_sub_menubar_ptr++;
Basic_gen_sub_menubar_data[Basic_gen_sub_menubar_ptr][0]= menu_root_title_ptr;
Basic_gen_sub_menubar_data[Basic_gen_sub_menubar_ptr][1]= menu_ptr;
if( Basic_gen_print_contral!=0)
printf("Basic_gen_sub_menubar_ptr=%d,menu_root_title_ptr=%d,menu_ptr=%d\n",
Basic_gen_sub_menubar_ptr,menu_root_title_ptr,menu_ptr );

     return NULL;
    }
  has_menu=0;
  menu = (Menu)xv_create(XV_NULL, MENU, NULL);

  for(k=0;k<Basic_gen_LC_Tree[menu_ptr][5];k++)
```

```
    {
     if(Basic_gen_H_and_W[menu_ptr][1][k]>=0)
       {

        txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[menu_ptr][1][k]][0];

        if(Basic_gen_virtual_item(txt_ptr)==0)
          {
           has_menu++;
           mi=Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k);
           xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
          }
        else if(Basic_gen_virtual_item(txt_ptr)==2 ||
                                        Basic_gen_virtual_item(txt_ptr)==1)
          {
          if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)

      if( Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==1 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==2
         || Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==4 )
         has_menu= Basic_gen_create_virtual_menus(Basic_gen_H_and_W[menu_ptr][2][k],menu);
                 else
       printf("the subtree of virtual menu item is not a menutree !\n");

          }
       }
    }
  if( has_menu==0)
     {
     xv_destroy(menu);
     return NULL;
     }

  else
     return menu;

  }


int
Basic_gen_create_virtual_menus(menu_ptr,menu)
int    menu_ptr;
Menu    menu;

 {
    int  k,result;
    char *txt_ptr;
    int  has_menu;
    int  Basic_gen_virtual_item();
    int  Basic_gen_create_virtual_menus();
    Panel_item    button1;
    Menu_item         mi;

    has_menu=0;
    for(k=0;k<Basic_gen_LC_Tree[menu_ptr][5];k++)
      {
       if(Basic_gen_H_and_W[menu_ptr][1][k]>=0)
         {

  txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[menu_ptr][1][k]][0];

          if(Basic_gen_virtual_item(txt_ptr)==0)
             {
              has_menu=1;
              mi=Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k);
              xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
             }
else if(Basic_gen_virtual_item(txt_ptr)==2 || Basic_gen_virtual_item(txt_ptr)==1)
             {
               if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)
                {
  if( Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==1 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==2 ||
                      Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==4 )
         has_menu= Basic_gen_create_virtual_menus(Basic_gen_H_and_W[menu_ptr][2][k],menu);
                 else
                      printf( "the subtree of virtual menu item is not a menutree !\n");
                }
             }
         }
     }
   return has_menu;

 }


 Menu_item
 Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k)

 char *txt_ptr;
```

```
int  menu_ptr;
int  k;

  {

    Menu_item        mi;
    Menu             menu ,Basic_gen_create_simulate_menus();
    void             Basic_gen_simulate_menu_action_proc();

    mi = xv_create(XV_NULL, MENUITEM,
      MENU_STRING,     txt_ptr,
      MENU_RELEASE,
      MENU_RELEASE_IMAGE,
      MENU_NOTIFY_PROC,  Basic_gen_simulate_menu_action_proc,
      NULL);

    if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)
           {
             menu = Basic_gen_create_simulate_menus(Basic_gen_H_and_W[menu_ptr][1][k],
Basic_gen_H_and_W[menu_ptr][2][k]);
             if( menu != NULL)
                    xv_set(mi, MENU_PULLRIGHT,menu, NULL);
           }
   return mi;
  }


void
Basic_gen_simulate_menu_action_proc(menu, menu_item)

Menu menu;
Menu_item menu_item;

{
  int  j,k,l;
  int  parent_tree_ptr;
  int  is_menu_item=1;
  int  is_active_submenubar;
  int  is_inactive_submenubar;
  int  temp_active_submenubar_ptr;
  int  temp_active_submenubar_tree[15];
  int  inactive_submenubar_ptr;
  int  inactive_submenubar_tree[20];
  void  Basic_gen_destroy_inactive_submenubar();
  void  meneme_application_links_proc();
  char  *txt_ptr;


  if(Basic_gen_sub_menubar_ptr>=0)
   {
    for(k=0;k<=Basic_gen_sub_menubar_ptr;k++)
     {
      txt_ptr=&Basic_gen_Textstring[Basic_gen_sub_menubar_data[k][0]][0];

      if (!strcmp((char *)xv_get(menu_item, MENU_STRING),txt_ptr))
                {
                 is_menu_item=0;
                 is_active_submenubar=0;
                 if(Basic_gen_active_submenubar_ptr>=0)
                   {
                    for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
                      {
if(Basic_gen_active_submenubar_tree[j]==Basic_gen_sub_menubar_data[k][1])
                         {
                          is_active_submenubar=1;
        printf("Menu Item: %s selected,tree %d is a active submenubar ! \n",
                                 xv_get(menu_item,
MENU_STRING),Basic_gen_active_submenubar_tree[j]);
 meneme_application_links_proc( xv_get(menu_item, MENU_STRING));
                          break;
                         }

                      }



                   }

           if(  is_active_submenubar==0)
                {
parent_tree_ptr= Basic_gen_LC_Tree[Basic_gen_sub_menubar_data[k][1]][6];

                    temp_active_submenubar_ptr=-1;
                    inactive_submenubar_ptr=-1;

                  /*  while( parent_tree_ptr!=simulating_ptr) */
                    while( parent_tree_ptr>=0)
                      {
if( Basic_gen_LC_Tree[ parent_tree_ptr][1]==0 ||
                                 Basic_gen_LC_Tree[ parent_tree_ptr][1]==3 )
                          {
 if( Basic_gen_print_contral!=0) printf("parent_tree %d \n", parent_tree_ptr);
```

```
                                 temp_active_submenubar_ptr++;
 temp_active_submenubar_tree[ temp_active_submenubar_ptr]= parent_tree_ptr;
                                 }
if( (Basic_gen_LC_Tree[Basic_gen_LC_Tree[ parent_tree_ptr][6]][6])>=0 )
parent_tree_ptr= Basic_gen_LC_Tree[ parent_tree_ptr][6];
                          else
                            break;
                          }
                       if( Basic_gen_print_contral!=0)
                         for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
 printf("active submenubar tree %d\n",Basic_gen_active_submenubar_tree[j]);
                       if( Basic_gen_print_contral!=0)
                         for(j=0;j<= temp_active_submenubar_ptr;j++)
 printf("temp_active submenubar tree %d\n",temp_active_submenubar_tree[j]);

          for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
           {
            is_inactive_submenubar=1;
            for(l=0;l<= temp_active_submenubar_ptr;l++)
 if( Basic_gen_active_submenubar_tree[j]== temp_active_submenubar_tree[l])
                 {
                  is_inactive_submenubar=0;
                  break;
                 }
            if( is_inactive_submenubar==1)
               {
                inactive_submenubar_ptr++;
inactive_submenubar_tree[ inactive_submenubar_ptr]=Basic_gen
                                                 _active_submenubar_tree[j];
               }
           }


          if( Basic_gen_print_contral!=0)
            for(j=0;j<= inactive_submenubar_ptr;j++)
              printf("inactive submenubar tree %d\n",inactive_submenubar_tree[j]);

          for(j=0;j<= inactive_submenubar_ptr;j++)
           Basic_gen_destroy_inactive_submenubar(inactive_submenubar_tree[j]);
          for(j=0;j<= temp_active_submenubar_ptr;j++)
        Basic_gen_active_submenubar_tree[j]== temp_active_submenubar_tree[j];
          Basic_gen_active_submenubar_ptr= temp_active_submenubar_ptr;

          Basic_gen_active_submenubar_ptr++;
          Basic_gen_active_submenubar_tree[Basic_gen_active_submenubar_ptr]=
                                                 Basic_gen_sub_menubar_data[k][1];
        if( Basic_gen_print_contral!=0)
              printf("Basic_gen_active_submenubar_ptr=%d,
                                        Basic_gen_active_submenubar_tree=%d\n",
           Basic_gen_active_submenubar_ptr,Basic_gen_sub_menubar_data[k][1]);
      Basic_gen_create_simulate_menubar( Basic_gen_sub_menubar_data[k][1]);


        printf("Menu Item: %s selected,create submenubar for %s \n",
            xv_get(menu_item, MENU_STRING),txt_ptr);
        meneme_application_links_proc( xv_get(menu_item, MENU_STRING));

        break;
                 }
              }
          }

     }

  if(is_menu_item==1)
     {
     printf("Menu Item: %s selected \n", xv_get(menu_item, MENU_STRING));
     meneme_application_links_proc( xv_get(menu_item, MENU_STRING));
     }

}

 void
 Basic_gen_destroy_inactive_submenubar(inactive_submenubar_tree_ptr)
 int inactive_submenubar_tree_ptr;
    {
    int         k;
    char        *txt_ptr;
    Panel_item   pi;

      for(k=0;k<Basic_gen_LC_Tree[inactive_submenubar_tree_ptr][5];k++)
        {

          if(Basic_gen_H_and_W[inactive_submenubar_tree_ptr][1][k]>=0)
            {

txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[inactive_submenubar_tree_ptr][1][k]][0];

             PANEL_EACH_ITEM(Basic_gen_panel, pi)

             if (!strcmp(txt_ptr, (char *)xv_get(pi, PANEL_LABEL_STRING)))    xv_destroy(pi);
```

```
             PANEL_END_EACH
             if( Basic_gen_print_contral!=0)
                printf("destroy menu button %s\n",txt_ptr);
             }

           }
        }



int
Basic_gen_panel_button_selected(item, event)

Panel_item item;
Event *event;

 {
  void button_application_links_proc();

 /* printf("menu button %s selected...\n", xv_get(item, PANEL_LABEL_STRING));*/

  button_application_links_proc( xv_get(item, PANEL_LABEL_STRING));
  return XV_OK;
 }



void
meneme_application_links_proc(active_meneme)

char *active_meneme;

{
   /* application include file
   void load_file_proc();
   void save_file_proc();
   void save_as_file_proc();
   void quit_proc();
   void create_horizontal_menubar_proc();
   void create_verticle_menubar_proc();
   void create_choice_menu_proc();
   void create_exclusive_menu_proc();
   void create_compatible_menu_proc();
   void add_meneme_text_proc();
   void undo_tree_operation_proc();
void modify_tree_proc();
void link_tree_proc();
void unlink_tree_proc();
void delete_tree_proc();
void delete_all_tree_proc();


if (!strcmp(active_meneme,"Load"))
        {
        printf("Call application for selected meneme Load \n");
        load_file_proc();
        }
else  if (!strcmp(active_meneme,"Save"))
        {
        printf("Call application for selected meneme Save \n");
        save_file_proc();
        }
else  if (!strcmp(active_meneme,"Save_As"))
        {
        printf("Call application for selected meneme Save_As \n");
        save_as_file_proc();
        }
else  if (!strcmp(active_meneme,"Quit"))
        {
        printf("Call application for selected meneme Quit \n");
        quit_proc();

        }
else  if (!strcmp(active_meneme,"HORIZONTAL"))
        {
        printf("Call application for creating  horizontal menubar \n");
        create_horizontal_menubar_proc();
        }
else  if (!strcmp(active_meneme,"VERTICLE"))
        {
        printf("Call application for creating verticle menubar \n");
        create_verticle_menubar_proc();
        }
else  if (!strcmp(active_meneme,"CHOICE"))
        {
        printf("Call application for creating verticle menu:choice type \n");
       create_choice_menu_proc();
        }
else  if (!strcmp(active_meneme,"EXCLUSIVE"))
        {
        printf("Call application for creating verticle menu:exclusive type \n");
        create_exclusive_menu_proc();
        }
else  if (!strcmp(active_meneme,"COMPATIBLE"))
        {
        printf("Call application for creating compatible menu \n");
        create_compatible_menu_proc();
        }
else  if (!strcmp(active_meneme,"TEXT"))
        {
        printf("Call application for creating meneme text operation \n");
        add_meneme_text_proc();
        }
else  if (!strcmp(active_meneme,"UNDO"))
        {
        printf("Call application for undo tree operation \n");
         undo_tree_operation_proc();
        }

   else  if (!strcmp(active_meneme,"MODIFY"))
        {
        printf("Call application for modify a selected tree \n");
        modify_tree_proc();
        }
else  if (!strcmp(active_meneme,"LINK"))
        {
        printf("Call application for link tree \n");
        link_tree_proc();
        }
else  if (!strcmp(active_meneme,"UNLINK"))
        {
        printf("Call application for unlinktree \n");
        unlink_tree_proc();
        }
else  if (!strcmp(active_meneme,"DELETE"))
        {
        printf("Call application for delete  selected tree \n");
        delete_tree_proc();
        }
else  if (!strcmp(active_meneme,"DELETE_ALL"))
        {
        printf("Call application for delete all tree \n");
        delete_all_tree_proc();
        }
  else  */
```

```
            printf("No application link to this selected meneme %s \n", active_meneme);


}


/* application draw start point */


void
draw_application_data_on_canvas(canvas, paint_window, dpy, xwin, xrects)
Canvas        canvas;       /* unused */
Xv_Window     paint_window;  /* unused */
Display       *dpy;
Window        xwin;
Xv_xrectlist *xrects;       /* unused */
{
 GC gc;
 int j;

  gc = DefaultGC(dpy, DefaultScreen(dpy));
  XClearWindow(dpy, xwin);
  XDrawString(dpy, xwin, gc, 25, 45, Basic_gen_msg, strlen(Basic_gen_msg));
}


/* application draw end point */

/* button application link proc start point */


void
button_application_links_proc(active_button)

char *active_button;

{

 /* application include file
    int help_button_proc();
    int simulation_button_proc();
    int go_to_design_button_proc();


    if (!strcmp(active_button,"SIMULATION"))
            {
            printf("Call application for selected button simulation \n");
            simulation_button_proc();
            }

  else if (!strcmp(active_button,"HELP"))
            {
            printf("Call application for selected button help \n");
            help_button_proc();
            }
  else if (!strcmp(active_button,"GO_TO_DESIGN"))
            {
            printf("Call application for selected button go_to_design \n");
            go_to_design_button_proc();
            }

  else    */

  printf("No application link to button %s \n", active_button);
}
/* event proc start point  */


  /*  case ACTION_SELECT :*/



int
event_ACTION_SELECT_proc(xx1,yy1)
 int xx1;
 int yy1;
{
  return;
}
  /*    case ACTION_MENU :  if(event_is_down(event))   */

int
event_ACTION_MENU_proc(xx1,yy1)
 int xx1,yy1;
 {
 return;
 }
/* event proc end point  */

/******  The end of Slc  **********/
```

```
/* Glc - The User Interface Generator */

#include <stdio.h>
#include <strings.h>
FILE *fplc,*fopen();
char *lc_filename, *lc_type,string[6][20];
char *ta,*th,*tl,*tm,*mi;


main(argc, argv)
int argc;
char *argv[];

{

 void generate_Makefile();
 void generate_main();
 void generate_link();
 void generate_application();
 void generate_head_file();


  if(argc != 2)
     {
     printf(" Input format error, stop(1)\n");
     exit(1);
     }
   else
    {

     lc_filename=&string[0][0];
     strcpy(lc_filename,argv[1]);
     strcat(lc_filename,".lc");
     lc_type="r";
     if((fplc=fopen(lc_filename,lc_type))==NULL)
         printf("%s dose not exist stop(2)\n",lc_filename);
     else
        {
        mi="menu_interface";
        tm=&string[1][0];
        tl=&string[2][0];
        ta=&string[3][0];
        th=&string[4][0];
        strcpy(tm,argv[1]);
        strcpy(tl,argv[1]);
        strcpy(ta,argv[1]);
        strcpy(th,argv[1]);
        strcat(tm,"_main");
        strcat(tl,"_link_app");
        strcat(ta,"_application");
        strcat(th,"_application");
        generate_Makefile(argv[1]);
        strcat(tm,".c");
        strcat(tl,".c");
        strcat(ta,".c");
        strcat(th,".h");
        generate_main(argv[1]);
        generate_link(argv[1]);
        generate_application(argv[1]);
        generate_head_file(argv[1]);

      }
    }
}



void
generate_Makefile(name)
char *name;
{
   FILE *fp,*fopen(),*fclose();
   char *filename, *type;
   filename="Makefile";
   type="w";
   if((fp=fopen(filename,type))==NULL)
        printf("Makefile generation for %s feiled\n",name);
   else
     {

       fprintf(fp,"#\n");
       fprintf(fp,"# @(#)Makefile is generated by glc from lc specification file %s.lc\n",name);
       fprintf(fp,"#\n");
       fprintf(fp,"# menu interface specification file %s.lc is created by Lean Cuisine Graphical
Editor\n",name);
       fprintf(fp,"#\n");
       fprintf(fp,"# do not modify this file by hand\n");
       fprintf(fp,"\n");
       fprintf(fp,"\n");
       fprintf(fp,"INCLUDE          = -I${OPENWINHOME}/include \n");
```

```
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"#\n");
        fprintf(fp,"# If you want to compile for debugging, change '-O' to '-g' \n");
        fprintf(fp,"#\n");
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"CFLAGS     = ${INCLUDE} -O  \n");
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"#\n");
        fprintf(fp,"# if you want special to pass special loader options to ld, set\n");
        fprintf(fp,"# LDFLAGS= ...\n");
        fprintf(fp,"#\n");
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"XVIEW_LIBS     = -L${OPENWINHOME}/lib -lxview -lolgx -lX11 -lm\n");
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"#\n");
        fprintf(fp,"# CFILES = %s_main.c %s_link_app.c %s_application.c\n",name,name,name);
        fprintf(fp,"#\n");
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fprintf(fp,"%s: %s.o %s.o %s.o \n",name,tm,tl,ta);
        fprintf(fp,"         ${CC} ${CFLAGS} ${LDFLAGS} -o %s   %s.o  %s.o %s.o %s.o
${XVIEW_LIBS}\n",name,tm,tl,ta,mi);
        fprintf(fp,"\n");
        fprintf(fp,"\n");
        fclose(fp);
     }
}

void
generate_main(name)
char *name;
{
   FILE  *fp,*fopen(),*fclose();
   char  *filename, *type;
   filename=tm;
   type="w";
   if((fp=fopen(filename,type))==NULL)
        printf(" %s generation  feiled\n",filename);
   else
      {
```

```
fprintf(fp,"/* %s  generated by glc  */\n",filename);
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_xview.h\"\n");
fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_basic2.h\"\n");
fprintf(fp,"#include \"%s_application.h\"\n",name);
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp,"main(argc, argv)\n");
fprintf(fp,"int argc;\n");
fprintf(fp,"char *argv[];\n");
fprintf(fp," \n");
fprintf(fp,"{\n");
fprintf(fp,"    int        draw_tree_ptr;\n");
fprintf(fp,"    int        i;\n");
fprintf(fp,"    int        simulating_ptr;\n");
fprintf(fp,"    char       *spec_file;\n");
fprintf(fp," \n");
fprintf(fp,"    spec_file=\"%s\";\n",lc_filename);
fprintf(fp,"    if((draw_tree_ptr=Basic_gen_get_interface_specification_file(spec_file))<0)\n");
fprintf(fp,"        {\n");
fprintf(fp,"         printf(\"nill L C Tree , stop(2)\\n\");\n");
fprintf(fp,"         exit(2);\n");
fprintf(fp,"        }\n");
fprintf(fp,"   simulating_ptr=-100;\n");

fprintf(fp,"   for(i=0;i<=draw_tree_ptr;i++)\n");
fprintf(fp,"         if(( Basic_gen_LC_Tree[i][1]==0 || Basic_gen_LC_Tree[i][1]==3 )&&(
Basic_gen_LC_Tree[i][6]<0))\n");
fprintf(fp,"             {\n");
fprintf(fp,"               simulating_ptr=i;\n");
fprintf(fp,"               break;\n");
fprintf(fp,"             }\n");

fprintf(fp,"   if(simulating_ptr<0)\n");
fprintf(fp,"        {\n");
fprintf(fp,"         printf(\"no menubar can be simulated ! stop(3)\\n\");\n");
fprintf(fp,"         exit(0);\n");
fprintf(fp,"        }\n");

fprintf(fp,"   xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);\n");

fprintf(fp,"    Basic_gen_frame = (Frame)xv_create(XV_NULL, FRAME,\n");
fprintf(fp,"     XV_WIDTH,            800,\n");
fprintf(fp,"     XV_HEIGHT,           500,\n");
fprintf(fp,"     FRAME_LABEL,          argv[0],\n");
fprintf(fp,"     FRAME_SHOW_FOOTER,    TRUE,\n");

fprintf(fp,"     NULL);\n");

fprintf(fp,"   Basic_gen_panel = (Panel)xv_create(Basic_gen_frame, PANEL,\n");
fprintf(fp,"     PANEL_LAYOUT, PANEL_HORIZONTAL,\n");
fprintf(fp,"    XV_X, 0,\n");
fprintf(fp,"     XV_Y, 0,\n");
fprintf(fp,"    XV_WIDTH, WIN_EXTEND_TO_EDGE,\n");
fprintf(fp,"    XV_HEIGHT, 25,\n");
fprintf(fp,"    WIN_BORDER, FALSE,\n");
fprintf(fp,"     NULL);\n");
fprintf(fp,"   Basic_gen_canvas = (Canvas)xv_create(Basic_gen_frame, CANVAS,\n");
fprintf(fp," XV_X, 0,\n");
fprintf(fp," XV_Y, 25,\n");
fprintf(fp,"     CANVAS_WIDTH,         1600,\n");
fprintf(fp,"     CANVAS_HEIGHT,        1500,\n");
fprintf(fp,"     CANVAS_AUTO_SHRINK,    FALSE,\n");
fprintf(fp,"     CANVAS_AUTO_EXPAND,    FALSE,\n");
fprintf(fp,"     CANVAS_REPAINT_PROC,   Basic_gen_canvas_repaint_proc,\n");
fprintf(fp,"     CANVAS_X_PAINT_WINDOW, TRUE,\n");

fprintf(fp,"     NULL);\n");

fprintf(fp,"   xv_set(canvas_paint_window(Basic_gen_canvas),\n");
fprintf(fp,"     WIN_CONSUME_EVENTS,\n");
fprintf(fp,"     WIN_NO_EVENTS,\n");
fprintf(fp,"     WIN_ASCII_EVENTS, KBD_USE, KBD_DONE,\n");
fprintf(fp,"     LOC_DRAG, /*LOC_WINENTER, LOC_WINEXIT,*/ WIN_MOUSE_BUTTONS,\n");
fprintf(fp,"     NULL,\n");
fprintf(fp,"     WIN_EVENT_PROC, Basic_gen_events,\n");
fprintf(fp,"     NULL);\n");

fprintf(fp,"   xv_set(Basic_gen_canvas,\n");
fprintf(fp,"     OPENWIN_SPLIT,\n");
fprintf(fp,"     OPENWIN_SPLIT_INIT_PROC,   Basic_gen_init_split,\n");
fprintf(fp,"     OPENWIN_SPLIT_DESTROY_PROC, Basic_gen_join_split,\n");
fprintf(fp,"     NULL,\n");
fprintf(fp,"     NULL);\n");

fprintf(fp,"   xv_create(Basic_gen_canvas, SCROLLBAR,\n");
fprintf(fp,"     SCROLLBAR_SPLITTABLE,  TRUE,\n");
fprintf(fp,"     SCROLLBAR_DIRECTION,   SCROLLBAR_VERTICAL,\n");
fprintf(fp,"     NULL);\n");
fprintf(fp,"   xv_create(Basic_gen_canvas, SCROLLBAR,\n");
fprintf(fp,"     SCROLLBAR_SPLITTABLE,  TRUE,\n");
fprintf(fp,"     SCROLLBAR_DIRECTION,   SCROLLBAR_HORIZONTAL,\n");
fprintf(fp,"     NULL);\n");



fprintf(fp," /* modify main: add user specification . start point */\n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");

fprintf(fp,"  /* modify main: add user specification . end   point */\n");


fprintf(fp,"    Basic_gen_menu_user_interface_simulation(simulating_ptr);\n");
fprintf(fp,"    %s_init();\n",name);
fprintf(fp,"    window_fit(Basic_gen_frame);\n");
fprintf(fp,"    xv_main_loop(Basic_gen_frame);\n");
fprintf(fp,"    exit(0);\n");
fprintf(fp," }\n");

fclose(fp);
     }
 }

void
generate_link(name)
char *name;
{
   FILE  *fp,*fopen(),*fclose();
   char *filename, *type;
   filename=tl;
   type="w";
   if((fp=fopen(filename,type))==NULL)
        printf(" %s generation  feiled\n",filename);
   else
      {
fprintf(fp,"/*  %s  generated by glc  */\n",filename);
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_xview.h\"\n");
fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_basic2.h\"\n");
fprintf(fp,"#include \"%s_application.h\"\n",name);
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp,"/* button application link proc  */\n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," void\n");
fprintf(fp," button_application_links_proc(active_button)\n");
fprintf(fp," char *active_button;\n");

fprintf(fp," { \n");

fprintf(fp," /* void   aaa_button_proc();\n");
fprintf(fp,"    void   bbb_button_proc();\n");


fprintf(fp,"    if (!strcmp(active_button,\"aaa\"))\n");
fprintf(fp,"          aaa_button_proc();\n");
fprintf(fp,"    else if (!strcmp(active_button,\"bbb\"))\n");
fprintf(fp,"          bbb_button_proc();\n");
fprintf(fp,"    else                      */\n");

fprintf(fp,"   printf(\"No application link to button %%s\\n\", active_button);\n");
fprintf(fp," }\n");

fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");

fprintf(fp," /* menume application link proc  */\n");


fprintf(fp," void \n");
fprintf(fp," meneme_application_links_proc(active_meneme) \n");

fprintf(fp," char *active_meneme;\n");
```

```
fprintf(fp," { \n");
fprintf(fp,"  /*    void    xxxx_application_proc();\n");
fprintf(fp,"        void    yyyy_application_proc();\n");


fprintf(fp,"    if (!strcmp(active_meneme,\"xxxx\"))\n");
fprintf(fp,"          xxxx_application_proc();\n");
fprintf(fp,"    else  if (!strcmp(active_meneme,\"yyyy\"))\n");
fprintf(fp,"          yyyy_application_proc();\n");
fprintf(fp,"    else                          */\n");
fprintf(fp,"          printf(\"No application link to this selected meneme %%s \\n\", active_meneme);\n");


fprintf(fp," }\n");


fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");


fprintf(fp," /* events  application link proc    */\n");

fprintf(fp," void\n");
fprintf(fp," Basic_gen_events(pw, event)\n");
fprintf(fp," Xv_Window pw;\n");
fprintf(fp," Event *event;\n");

fprintf(fp," {\n");

fprintf(fp,"  register char   *p = Basic_gen_msg;\n");
fprintf(fp,"  int code        = event_action(event);\n");
fprintf(fp,"  Xv_Window       view;\n");
fprintf(fp,"  int i           = (int)xv_get(Basic_gen_canvas, OPENWIN_NVIEWS);\n");
fprintf(fp,"  Window xwin     = (Window)xv_get(pw, XV_XID);\n");

fprintf(fp,"    *p = 0;\n");

fprintf(fp,"   if (event_is_key_left(event))\n");
fprintf(fp,"      sprintf(p, \"(L%%d) \", event_id(event) - KEY_LEFTFIRST + 1);\n");
fprintf(fp,"   else if (event_is_key_top(event))\n");
fprintf(fp,"      sprintf(p, \"(T%%d) \", event_id(event) - KEY_TOPFIRST + 1);\n");
fprintf(fp,"   else if (event_is_key_right(event))\n");
fprintf(fp,"      sprintf(p, \"(R%%d) \", event_id(event) - KEY_RIGHTFIRST + 1);\n");
fprintf(fp,"   else if (event_id(event) == KEY_BOTTOMLEFT)\n");
fprintf(fp,"      strcpy(p, \"bottom left \");\n");
fprintf(fp,"   else if (event_id(event) == KEY_BOTTOMRIGHT)\n");
fprintf(fp,"      strcpy(p, \"bottom right \");\n");
fprintf(fp,"   p += strlen(p);\n");

fprintf(fp,"   while (pw != xv_get(Basic_gen_canvas,
CANVAS_NTH_PAINT_WINDOW, --i) && i > 0)\n");
fprintf(fp,"      ;\n");
fprintf(fp,"   view = xv_get(Basic_gen_canvas, OPENWIN_NTH_VIEW, i);\n");

fprintf(fp,"   if (event_is_ascii(event))\n");

fprintf(fp,"    {\n");
fprintf(fp,"        sprintf(p, \"Keyboard: key '%%c' (%%d) %%s at %%d,%%d\",\n");
fprintf(fp,"        event_action(event), event_action(event),\n");
fprintf(fp,"        event_is_down(event)? \"pressed\" : \"released\",\n");
fprintf(fp,"        event_x(event), event_y(event));\n");
fprintf(fp,"    } \n");
fprintf(fp,"   else switch (event_action(event))\n");
fprintf(fp,"    {\n");
fprintf(fp,"     case ACTION_CLOSE :\n");
fprintf(fp,"        xv_set(Basic_gen_frame, FRAME_CLOSED, TRUE, NULL);\n");
fprintf(fp,"        break;\n");
fprintf(fp,"     case ACTION_OPEN :\n");
fprintf(fp,"        strcpy(p, \"frame opened up\");\n");
fprintf(fp,"        break;\n");
fprintf(fp,"     case ACTION_HELP :\n");
fprintf(fp,"        strcpy(p, \"Help (action ignored)\");\n");
fprintf(fp,"        break;\n");
fprintf(fp,"     case ACTION_SELECT :\n");
fprintf(fp,"           sprintf(p, \"Button: Select (Left) %%s at %%d,%%d\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\",\n");
fprintf(fp,"          event_x(event), event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case ACTION_ADJUST :\n");
fprintf(fp,"           sprintf(p, \"Button: Adjust (Middle) %%s at %%d,%%d\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\",\n");
fprintf(fp,"           event_x(event), event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case ACTION_MENU :\n");
fprintf(fp,"           sprintf(p, \"Button: Menu (Right) %%s at %%d,%%d\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\",\n");
```

```
fprintf(fp,"           event_x(event), event_y(event));\n");
fprintf(fp,"           break; \n");
fprintf(fp,"     case SHIFT_RIGHT :\n");
fprintf(fp,"           sprintf(p, \"Keyboard: right shift %%s\",\n");
fprintf(fp,"          event_is_down(event)? \"pressed\" : \"released\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case SHIFT_LEFT :\n");
fprintf(fp,"           sprintf(p, \"Keyboard: left shift %%s\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case SHIFT_LEFTCTRL : case SHIFT_RIGHTCTRL :\n");
fprintf(fp,"           sprintf(p, \"Keyboard: control key %%s\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"    case SHIFT_META :\n");
fprintf(fp,"           sprintf(p, \"Keyboard: meta key %%s\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case SHIFT_ALT :\n");
fprintf(fp,"           sprintf(p, \"Keyboard: alt key %%s\",\n");
fprintf(fp,"           event_is_down(event)? \"pressed\" : \"released\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case KBD_USE:\n");
fprintf(fp,"           sprintf(p, \"Keyboard: got keyboard focus\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case KBD_DONE:\n");
fprintf(fp,"           sprintf(p, \"Keyboard: lost keyboard focus\");\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case LOC_MOVE:\n");
fprintf(fp,"           sprintf(p, \"Pointer: moved to %%d,%%d\",\n");
fprintf(fp,"           event_x(event),event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"    case LOC_DRAG:\n");
fprintf(fp,"           sprintf(p, \"Pointer: dragged to %%d,%%d\",\n");
fprintf(fp,"           event_x(event), event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case LOC_WINENTER:\n");
fprintf(fp,"           win_set_kbd_focus(pw, xv_get(pw, XV_XID));\n");
fprintf(fp,"           sprintf(p, \"Pointer: entered window at %%d,%%d\",\n");
fprintf(fp,"           event_x(event), event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"     case LOC_WINEXIT:\n");
fprintf(fp,"           sprintf(p, \"Pointer: exited window at %%d,%%d\",\n");
fprintf(fp,"           event_x(event), event_y(event));\n");
fprintf(fp,"           break;\n");
fprintf(fp,"      case WIN_RESIZE :\n");
fprintf(fp,"      case WIN_REPAINT :\n");
fprintf(fp,"            return;\n");
fprintf(fp,"      default :\n");
fprintf(fp,"            return;\n");
fprintf(fp,"    }\n");
fprintf(fp,"    Basic_gen_canvas_repaint_proc(Basic_gen_canvas, pw,\n");
fprintf(fp,"      xv_get(Basic_gen_canvas, XV_DISPLAY), xv_get(pw, XV_XID), NULL);\n");
fprintf(fp," }\n");


fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");
fprintf(fp," \n");


fprintf(fp,"/* application draw proc    */\n");


fprintf(fp,"void \n");
fprintf(fp,"draw_application_data_on_canvas(canvas, paint_window, dpy, xwin, xrects)\n");
fprintf(fp,"Canvas       canvas;       /* unused */\n");
fprintf(fp,"Xv_Window    paint_window;  /* unused */\n");
fprintf(fp,"Display     *dpy;\n");
fprintf(fp,"Window       xwin;\n");
fprintf(fp,"Xv_xrectlist *xrects;       /* unused */\n");
fprintf(fp,"{\n");
fprintf(fp," GC           gc;\n");

fprintf(fp," gc = DefaultGC(dpy, DefaultScreen(dpy));\n");
fprintf(fp," XClearWindow(dpy, xwin);\n");

fprintf(fp," XDrawString(dpy, xwin, gc, 25, 45, Basic_gen_msg, strlen(Basic_gen_msg));\n");
fprintf(fp,"}\n");
fclose(fp);
     }
 }

void
generate_application(name)
char *name;
{
  FILE  *fp,*fopen(),*fclose();
  char  *filename, *type;
  filename=ta;
  type="w";
  if((fp=fopen(filename,type))==NULL)
       printf(" %s generation  feiled\n",filename);
  else
     {
     fprintf(fp,"/* %s  generated by glc   */\n",filename);
     fprintf(fp," \n");
     fprintf(fp," \n");
     fprintf(fp," \n");
     fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_xview.h\"\n");
     fprintf(fp,"#include \"/home/sis-server/student/JGu/demo/include/lc_basic2.h\"\n");
     fprintf(fp,"#include \"%s_application.h\"\n",name);
     fprintf(fp," \n");
     fprintf(fp," \n");
     fprintf(fp," \n");
     fprintf(fp,"void        %s_init()\n",name);
     fprintf(fp,"            {\n");
     fprintf(fp,"             ;\n");
     fprintf(fp,"            }\n");
     fclose(fp);
     }
}

void
generate_head_file(name)
char *name;
{
  FILE  *fp,*fopen(),*fclose();
  char  *filename, *type;
  filename=th;
  type="w";
  if((fp=fopen(filename,type))==NULL)
       printf(" %s generation  feiled\n",filename);
  else
     {
      fprintf(fp,"/* %s  generated by glc   */\n",filename);
     fprintf(fp,"\n");
     fprintf(fp,"\n");
     fprintf(fp,"void        %s_init();\n",name);
     fclose(fp);
     }
}              /* The end of Glc  */
```

```
/* Basic functions of the Lc_run_time Kernel*/


#include "/home/sis-server/student/JGu/demo/include/lc_xview.h"
#include "/home/sis-server/student/JGu/demo/include/lc_basic2.h"


void
Basic_gen_canvas_repaint_proc(canvas, paint_window, dpy, xwin, xrects)
Canvas      canvas;       /* unused */
Xv_Window    paint_window;  /* unused */
Display    *dpy;
Window      xwin;
Xv_xrectlist *xrects;       /* unused */
{

void draw_application_data_on_canvas();
 /* GC gc;
  int j;

  gc = DefaultGC(dpy, DefaultScreen(dpy));
  XClearWindow(dpy, xwin);*/

  /* call real draw subroutines of application*/

    draw_application_data_on_canvas(canvas, paint_window, dpy, xwin, xrects);


 /* XDrawString(dpy, xwin, gc, 25, 45, Basic_gen_msg, strlen(Basic_gen_msg));*/
}


void
Basic_gen_join_split(view)
Xv_Window view;
{
  puts("joined view");
}



void
Basic_gen_init_split(oldview, newview, pos)
Xv_Window oldview, newview;
int pos;
```

```
{
  xv_set(xv_get(newview, CANVAS_VIEW_PAINT_WINDOW),
    WIN_EVENT_PROC,         Basic_gen_events,
    WIN_CONSUME_EVENT,    ACTION_SELECT, ACTION_ADJUST, NULL,
    NULL);
}



 int Basic_gen_get_interface_specification_file(filename)
 char *filename;

 {
 FILE *fp,*fopen();
 char *type;
 int result,i,j,k;
 char buf[128];
 char *sp;


 int draw_tree_ptr=-1;
 int text_ptr=-1;

 int select_ptr=-100;
 int pick_hight,pick_width;

 if( Basic_gen_print_contral!=0) printf("input specification file by file name %s \n",filename);

   if((strlen(filename))<=0)
     {
       printf("File name is nill stop[2]\n");
       exit(2);
     }

   else

     {
       type="r";
       if( Basic_gen_print_contral!=0) printf("filename %s type %s\n",filename,type);
       if((fp=fopen(filename,type))==NULL)
           {
             printf("open file feiled,stop[3]\n");
             exit(3);
           }
       else
```

```
            {
             select_ptr=-1;
             pick_hight=-1;
             pick_width=-1;
             draw_tree_ptr=-1;
             text_ptr=-1;
             fscanf(fp,"%d\n",&draw_tree_ptr);
             if( Basic_gen_print_contral!=0) printf("draw_tree_ptr=%d\n",draw_tree_ptr);

                for(i=0;i<=draw_tree_ptr;i++)
                  {
                   for(j=0;j<=7;j++)
                      {
                       fscanf(fp,"%d ",&Basic_gen_LC_Tree[i][j]);
  if( Basic_gen_print_contral!=0) printf("Lc_tree[%d][%d]= %d\n",i,j,Basic_gen_LC_Tree[i][j]);

                      }
                   for(j=0;j<Basic_gen_LC_Tree[i][5];j++)
                      {
                     /* printf("input%d\n",j);*/
                      fscanf(fp,"%d %d %d
",&Basic_gen_H_and_W[i][0][j],&Basic_gen_H_and_W[i][1][j],&Basic_gen_H_and_W[i][2][j]);
                      if( Basic_gen_print_contral!=0) printf("Basic_gen_H_and_W[%d][0][%d]=%d,
                     Basic_gen_H_and_W[%d][1][%d]=%d,Basic_gen_H_and_W[%d][2][%d]=%d\n",

i,j,Basic_gen_H_and_W[i][0][j],i,j,Basic_gen_H_and_W[i][1][j],i,j,Basic_gen_H_and_W[i][2][j]);

                      }
                   for(j=0;j<=3;j++)
                      {
                      fscanf(fp,"%d ",&Basic_gen_Max_Min_Box[i][j]);
                      if( Basic_gen_print_contral!=0)
                     printf("max_min_box[%d][%d]=%d\n",i,j,Basic_gen_Max_Min_Box[i][j]);

                      }
                  }
               fscanf(fp,"\n");
               fscanf(fp,"%d\n",&text_ptr);
               if( Basic_gen_print_contral!=0) printf("text_ptr=%d\n",text_ptr);

               if(text_ptr>=0)
                  {
                   for(k=0;k<=text_ptr;k++)
                      {
                       sp=&Basic_gen_Textstring[k][0];
                       fscanf(fp,"%s ",sp);
                       if( Basic_gen_print_contral!=0) printf("%D     %s\n",k,sp);
                      }
                  }
               fscanf(fp,"\n");
               fscanf(fp,"%d %d %d\n",&select_ptr,&pick_hight,&pick_width);
               if( Basic_gen_print_contral!=0) printf("select_ptr=%d,pick_hight=%d,
                                  pick_width=%d\n",select_ptr,pick_hight,pick_width);

               fclose(fp);
              }

       }
   return(draw_tree_ptr);
  }



void
Basic_gen_menu_user_interface_simulation(simulating_ptr)
int simulating_ptr;

{
  int      Basic_gen_create_simulate_menubar();
  void      Basic_gen_defult_submenubar_simulation();

  char      *txt_ptr;
  if( Basic_gen_print_contral!=0) printf("menu user interface simulation %d \n", simulating_ptr);



      Basic_gen_sub_menubar_ptr=-1;
      Basic_gen_active_submenubar_ptr=-1;

      xv_set(Basic_gen_frame, FRAME_LABEL, "untitled Menu interface simulation",NULL);

      if(Basic_gen_LC_Tree[simulating_ptr][7]>=0)
        {
         txt_ptr=&Basic_gen_Textstring[Basic_gen_LC_Tree[simulating_ptr][7]][0];
         xv_set(Basic_gen_frame, FRAME_LABEL,txt_ptr,NULL);
        }

      Basic_gen_create_simulate_menubar( simulating_ptr);
      Basic_gen_defult_submenubar_simulation();
}  /*********/
```

```
void
Basic_gen_defult_submenubar_simulation()

{


  if(Basic_gen_sub_menubar_ptr>=0)
    {

        Basic_gen_active_submenubar_ptr++;
Basic_gen_active_submenubar_tree[Basic_gen_active_submenubar_ptr]
                                        =Basic_gen_sub_menubar_data[0][1];
        Basic_gen_create_simulate_menubar( Basic_gen_sub_menubar_data[0][1]);



    }


}
/****************************/


int
Basic_gen_create_simulate_menubar(simulate_ptr)

int simulate_ptr;


     {
       int  Basic_gen_create_simulate_menubar_item();
       int  k,result;
       char *txt_ptr;
       int  Basic_gen_virtual_item();
       Panel_item button1;

       for(k=0;k<Basic_gen_LC_Tree[simulate_ptr][5];k++)
         {

           if(Basic_gen_H_and_W[simulate_ptr][1][k]>=0)
             {

             txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[simulate_ptr][1][k]][0];
             if(Basic_gen_virtual_item(txt_ptr)==0)

               Basic_gen_create_simulate_menubar_item(txt_ptr,simulate_ptr,k);

             else if(Basic_gen_virtual_item(txt_ptr)==2 || Basic_gen_virtual_item(txt_ptr)==1)
               {
                if(Basic_gen_H_and_W[simulate_ptr][2][k]>=0)
                  {
                   if( Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==0 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==3 )
                       Basic_gen_create_simulate_menubar(Basic_gen_H_and_W[simulate_ptr][2][k]);
                  else
                    {
                     printf( "the subtree of virtual menubar item is not a menubar !\n");

                    }
                  }

               }

           }

        }
       return;
      }

int
Basic_gen_virtual_item(t)
char *t;
  {

    if(*t=='{')
      {
       while( *t != '\0')
           t++;
       if(*(t-1)=='}')
         return(2);
       else
         return(1);
      }
    else
      return(0);

  }



int
Basic_gen_create_simulate_menubar_item(item_name,simulate_ptr,k)
 char *item_name;
```

```
int simulate_ptr;
int  k;

{
   Panel_item  pi,button1;
   Menu       simulate_menu,Basic_gen_create_simulate_menus();
   int        Basic_gen_panel_button_selected();
   int        result;
   /* loop thru all panel items and check for item with same name */

   PANEL_EACH_ITEM(Basic_gen_panel, pi)

      if (!strcmp(item_name, (char *)xv_get(pi, PANEL_LABEL_STRING))) return PANEL_NONE;

   PANEL_END_EACH
      pi=(Panel_item) xv_create(Basic_gen_panel,  PANEL_BUTTON,
      PANEL_LABEL_STRING,    item_name,
      PANEL_NOTIFY_PROC,     Basic_gen_panel_button_selected,
      NULL);

     if(Basic_gen_H_and_W[simulate_ptr][2][k]>=0)
          {

          if( Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==0 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[simulate_ptr][2][k]][1]==3 )
               printf( "the subtree of real menubar item is a menubar !\n");
          else
               {
simulate_menu  = Basic_gen_create_simulate_menus(Basic_gen_H_and_W[simulate_ptr][1][k],
                                   Basic_gen_H_and_W[simulate_ptr][2][k]);
                 if( simulate_menu != NULL)
                 xv_set(pi,  PANEL_ITEM_MENU, simulate_menu, NULL);
               }
          }
     return PANEL_NEXT;
}


Menu
Basic_gen_create_simulate_menus(menu_root_title_ptr,menu_ptr)

int menu_root_title_ptr;
int menu_ptr;
  {
     Menu_item        mi,Basic_gen_create_simulate_menu_item();
     Menu             menu;
     int              k,has_menu,result;
     char             *txt_ptr;
     int              Basic_gen_create_virtual_menus();
     Panel_item       button1;


   if( Basic_gen_LC_Tree[menu_ptr][1]==0 || Basic_gen_LC_Tree[menu_ptr][1]==3 )        /* sub menubar
*/


     {

       for(k=0;k<=Basic_gen_sub_menubar_ptr;k++)
            if( Basic_gen_sub_menubar_data[k][1]==menu_ptr)
                 return NULL;

       Basic_gen_sub_menubar_ptr++;
       Basic_gen_sub_menubar_data[Basic_gen_sub_menubar_ptr][0]= menu_root_title_ptr;
       Basic_gen_sub_menubar_data[Basic_gen_sub_menubar_ptr][1]= menu_ptr;
if( Basic_gen_print_contral!=0)
                      printf("Basic_gen_sub_menubar_ptr=%d,menu_root_title_ptr=%d,menu_ptr=%d\n",
                                         Basic_gen_sub_menubar_ptr,menu_root_title_ptr,menu_ptr );

       return NULL;
     }
   has_menu=0;
   menu = (Menu)xv_create(XV_NULL, MENU, NULL);

   for(k=0;k<Basic_gen_LC_Tree[menu_ptr][5];k++)
     {
       if(Basic_gen_H_and_W[menu_ptr][1][k]>=0)
         {

           txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[menu_ptr][1][k]][0];
           if(Basic_gen_virtual_item(txt_ptr)==0)
             {
              has_menu++;
              mi=Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k);
              xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
             }
           else  if(Basic_gen_virtual_item(txt_ptr)==2 || Basic_gen_virtual_item(txt_ptr)==1)

              {
              if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)
                              if( Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==1 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==2
                                   || Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==4 )
   has_menu= Basic_gen_create_virtual_menus(Basic_gen_H_and_W[menu_ptr][2][k],menu);
                  else

                        printf("the subtree of virtual menu item is not a menutree !\n");


               }

           }
       }
   if( has_menu==0)
       {
       xv_destroy(menu);
       return NULL;
       }

   else
       return menu;

   }


int
Basic_gen_create_virtual_menus(menu_ptr,menu)
int    menu_ptr;
Menu    menu;

 {
    int  k,result;
    char *txt_ptr;
    int  has_menu;
    int  Basic_gen_virtual_item();
    int  Basic_gen_create_virtual_menus();
    Panel_item     button1;
    Menu_item         mi;

    has_menu=0;
    for(k=0;k<Basic_gen_LC_Tree[menu_ptr][5];k++)
       {
        if(Basic_gen_H_and_W[menu_ptr][1][k]>=0)
           {

             txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[menu_ptr][1][k]][0];
             if(Basic_gen_virtual_item(txt_ptr)==0)
```

```
               {
                has_menu=1;
                mi=Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k);
                xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
               }
          else  if(Basic_gen_virtual_item(txt_ptr)==2 || Basic_gen_virtual_item(txt_ptr)==1)
               {
                 if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)
                  {
   if( Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==1 ||
Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==2 ||
                                 Basic_gen_LC_Tree[Basic_gen_H_and_W[menu_ptr][2][k]][1]==4 )
                   has_menu=
Basic_gen_create_virtual_menus(Basic_gen_H_and_W[menu_ptr][2][k],menu);
                   else
                         printf( "the subtree of virtual menu item is not a menutree !\n");
                  }
               }
          }
     }
   return has_menu;

 }


  Menu_item
  Basic_gen_create_simulate_menu_item(txt_ptr,menu_ptr,k)

  char *txt_ptr;
  int  menu_ptr;
  int  k;

   {


     Menu_item        mi;
     Menu             menu ,Basic_gen_create_simulate_menus();
     void            Basic_gen_simulate_menu_action_proc();

     mi = xv_create(XV_NULL, MENUITEM,
        MENU_STRING,      txt_ptr,
        MENU_RELEASE,
        MENU_RELEASE_IMAGE,
        MENU_NOTIFY_PROC,  Basic_gen_simulate_menu_action_proc,
        NULL);

     if(Basic_gen_H_and_W[menu_ptr][2][k]>=0)
```

```
            {
              menu = Basic_gen_create_simulate_menus(Basic_gen_H_and_W[menu_ptr][1][k],
                                                     Basic_gen_H_and_W[menu_ptr][2][k]);
              if( menu != NULL)
                    xv_set(mi, MENU_PULLRIGHT,menu, NULL);
            }


     return mi;



  }




void
Basic_gen_simulate_menu_action_proc(menu, menu_item)

Menu menu;
Menu_item menu_item;

{
  int  j,k,l;
  int  parent_tree_ptr;
  int  is_menu_item=1;
  int  is_active_submenubar;
  int  is_inactive_submenubar;
  int  temp_active_submenubar_ptr;
  int  temp_active_submenubar_tree[15];
  int  inactive_submenubar_ptr;
  int  inactive_submenubar_tree[20];
  void  Basic_gen_destroy_inactive_submenubar();
  void  meneme_application_links_proc();
  char  *txt_ptr;


  if(Basic_gen_sub_menubar_ptr>=0)
    {
     for(k=0;k<=Basic_gen_sub_menubar_ptr;k++)
        {
        txt_ptr=&Basic_gen_Textstring[Basic_gen_sub_menubar_data[k][0]][0];

        if (!strcmp((char *)xv_get(menu_item, MENU_STRING),txt_ptr))
                {
                 is_menu_item=0;
                 is_active_submenubar=0;
                 if(Basic_gen_active_submenubar_ptr>=0)
                   {
                   for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
                     {
                      if(Basic_gen_active_submenubar_tree[j]==Basic_gen_sub_menubar_data[k][1])
                        {
                          is_active_submenubar=1;
                         printf("Menu Item: %s selected,tree %d is a active submenubar ! \n",
                               xv_get(menu_item,
MENU_STRING),Basic_gen_active_submenubar_tree[j]);
                              meneme_application_links_proc( xv_get(menu_item, MENU_STRING));
                         break;
                        }

                     }


                   }

         if( is_active_submenubar==0)
            {
                 parent_tree_ptr= Basic_gen_LC_Tree[Basic_gen_sub_menubar_data[k][1]][6];

                  temp_active_submenubar_ptr=-1;
                  inactive_submenubar_ptr=-1;

               /* while( parent_tree_ptr!=simulating_ptr) */
                  while( parent_tree_ptr>=0)
                       {
         if( Basic_gen_LC_Tree[ parent_tree_ptr][1]==0 || Basic_gen_LC_Tree[ parent_tree_ptr][1]==3 )
                          {
                           if( Basic_gen_print_contral!=0) printf("parent_tree %d \n", parent_tree_ptr);
                           temp_active_submenubar_ptr++;
     temp_active_submenubar_tree[ temp_active_submenubar_ptr]= parent_tree_ptr;
                          }
                     if( (Basic_gen_LC_Tree[Basic_gen_LC_Tree[ parent_tree_ptr][6]][6])>=0 )
                        parent_tree_ptr= Basic_gen_LC_Tree[ parent_tree_ptr][6];
                     else
                        break;
                     }
                  if( Basic_gen_print_contral!=0)
                    for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
                     printf("active submenubar tree %d\n",Basic_gen_active_submenubar_tree[j]);
                  if( Basic_gen_print_contral!=0)
```

```
                    for(j=0;j<= temp_active_submenubar_ptr;j++)
                     printf("temp_active submenubar tree %d\n",temp_active_submenubar_tree[j]);

        for(j=0;j<=Basic_gen_active_submenubar_ptr;j++)
          {
           is_inactive_submenubar=1;
           for(l=0;l<= temp_active_submenubar_ptr;l++)
              if( Basic_gen_active_submenubar_tree[j]== temp_active_submenubar_tree[l])
                 {
                  is_inactive_submenubar=0;
                  break;
                 }
           if( is_inactive_submenubar==1)
              {
               inactive_submenubar_ptr++;
              inactive_submenubar_tree[
inactive_submenubar_ptr]=Basic_gen_active_submenubar_tree[j];
              }


          }

        if( Basic_gen_print_contral!=0)
           for(j=0;j<= inactive_submenubar_ptr;j++)
             printf("inactive submenubar tree %d\n",inactive_submenubar_tree[j]);

        for(j=0;j<= inactive_submenubar_ptr;j++)
             Basic_gen_destroy_inactive_submenubar(inactive_submenubar_tree[j]);
        for(j=0;j<= temp_active_submenubar_ptr;j++)
             Basic_gen_active_submenubar_tree[j]== temp_active_submenubar_tree[j];
        Basic_gen_active_submenubar_ptr= temp_active_submenubar_ptr;

        Basic_gen_active_submenubar_ptr++;
Basic_gen_active_submenubar_tree[Basic_gen_active_submenubar_ptr]=
                                                              Basic_gen_sub_menubar_data[k][1];
        if( Basic_gen_print_contral!=0)
            printf("Basic_gen_active_submenubar_ptr=%d,Basic_gen_active_submenubar_tree=%d\n",
                 Basic_gen_active_submenubar_ptr,Basic_gen_sub_menubar_data[k][1]);
        Basic_gen_create_simulate_menubar( Basic_gen_sub_menubar_data[k][1]);


        printf("Menu Item: %s selected,create submenubar for %s \n",
             xv_get(menu_item, MENU_STRING),txt_ptr);
       meneme_application_links_proc( xv_get(menu_item, MENU_STRING));
        break;
                }
```

```
            }
        }
    }
 if(is_menu_item==1) {
    printf("Menu Item: %s selected \n", xv_get(menu_item, MENU_STRING));
    meneme_application_links_proc( xv_get(menu_item, MENU_STRING));
    }
}

 void
 Basic_gen_destroy_inactive_submenubar(inactive_submenubar_tree_ptr)
 int inactive_submenubar_tree_ptr; {
     int       k;
     char      *txt_ptr;
     Panel_item  pi;
       for(k=0;k<Basic_gen_LC_Tree[inactive_submenubar_tree_ptr][5];k++)
         {
           if(Basic_gen_H_and_W[inactive_submenubar_tree_ptr][1][k]>=0)
             {

txt_ptr=&Basic_gen_Textstring[Basic_gen_H_and_W[inactive_submenubar_tree_ptr][1][k]][0];
               PANEL_EACH_ITEM(Basic_gen_panel, pi)
               if (!strcmp(txt_ptr, (char *)xv_get(pi, PANEL_LABEL_STRING)))    xv_destroy(pi);
               PANEL_END_EACH
               if( Basic_gen_print_contral!=0)
                  printf("destroy menu button %s\n",txt_ptr);
             }
         }
     }


 int
 Basic_gen_panel_button_selected(item, event)
 Panel_item item;
 Event *event;

 {
   void button_application_links_proc();

  /* printf("menu button %s selected...\n", xv_get(item, PANEL_LABEL_STRING));*/

   button_application_links_proc( xv_get(item, PANEL_LABEL_STRING));
   return XV_OK;
 }

/* The end of the Lc_run_time Kernel*/
```