The Automatic Generation and Execution of Lean Cuisine+ Specifications

A thesis presented in partial fulfilment of the requirements
for the degree of
Master of Information Science
in
Computer Science
at Massey University, Palmerston North, New Zealand

Lei LI (李磊)

2003

ABSTRACT

Lean Cuisine+ (Phillips, 1995), a semi-formal graphical dialogue notation for describing the behaviour of event based direct manipulation GUIs, was developed at Massey University in the early 1990s. More recently, a software environment, SELCU (Scogings, 2003) has been built for Lean Cuisine+ which permits dialogue models to be manually constructed and edited using a drag and drop approach. The aim of the research presented in this thesis is to develop extensions to SELCU, which include the automatic generation of Lean Cuisine+ diagrams, and their execution.

A shortcoming of current prototyping tools and user interface builders is that although they permit the designer to construct a mock up of the look and feel of the interface, they provide no model of the interaction. The Auto-Generation Software is a tool which can automatically generate a Lean Cuisine+ diagram for a graphical user interface developed using Delphi. The generated description is represented as a text file, and in a format compatible with the SELCU system.

The Lean Cuisine+ Execution Environment is embedded in the SELCU application. It supports the execution of Lean Cuisine+ specifications, including meneme selection and task action sequence, and also takes account of triggers.

The SELCU extensions successfully integrate a graphical dialogue notation (Lean Cuisine+), an object oriented development environment (Delphi), and an existing support environment (SELCU). This offers a more complete environment for the early stages of the design of graphical user interfaces.

ACKNOWLEDGEMENTS

First of all, I wish to gratefully acknowledge my supervisor Dr. Chris Phillips, for introducing me to this exciting research field, and for his extremely helpful insights and guidance. As an international student, this research year in New Zealand has meant so much more to me because of him, and his professional, unconditional and dedicated encouragement and support.

Gratitude must also express to the following people:

- Mr. Chris Scogings, for his co-supervision, and for his helpful suggestions in regard to the SELCU environment.

- Dr. Ray Kemp, for explanation of StateChart software, and for representing the big picture of execution environments to me.

- Dr Judy Brown, for providing an introduction to Panorama, and for her friendship and assistance.

Finally, I give thanks to my parents, for their undying love and encouragement throughout my life.

CONTENTS

ABSTRACT I

ACKNOWLEDGMENTS III

CONTENTS V

FIGURES IX

TABLES XIII

INTRODUCTION 1

1.1 Human Computer Interaction Concepts 2

1.2 Dialogue Concepts 4

1.3 Project Goal 5

1.4 Structure of the Thesis 7

LITERATURE REVIEW 9

2.1 Dialogue Notations 9

2.1.1 Textual Dialogue Notations 10

2.1.2 Graphical Dialogue Notations 11

2.2 Support Environments for Dialogue Notations 23

2.2.1 State Charts 23

2.2.2 Petri Nets 28

2.3 Early Prototyping Tools and IDEs 31

2.4 Summary 39

THE LEAN CUISINE+ NOTATION AND ITS SUPPORT ENVIRONMENT..... 41

3.1 The Lean Cuisine+ Notation..... 41

3.1.1 Basic Definitions..... 42

3.1.2 Column Attributes Selector Case Study..... 44

3.1.3 Preconditions..... 47

3.1.4 Selection Triggers..... 48

3.1.5 Task Action Sequence..... 49

3.2 Support Environment for Lean Cuisine+..... 51

3.2.1 SELCU Interface Introduction..... 52

3.2.2 SELCU Source File Format..... 55

3.2.3 Main Functions of SELCU..... 63

3.3 Proposed Extensions of SELCU..... 67

3.4 Summary..... 69

CONCEPTUAL DESIGN OF SELCU EXTENSIONS..... 71

4.1 Design Decisions..... 71

4.2 Auto-Generation Software Interface Components..... 75

4.3 Lo-Fi Prototyping of Execution Environment..... 80

4.3.1 The Extended SELCU Interface within Execution Function..... 81

4.3.2 The Main Interface of SELCU Execution Environment..... 81

4.3.3 Preview Mode of Execution Environment..... 83

4.3.4 Selecting a Meneme..... 84

4.3.5 Viewing Tasks in the Execution Mode..... 86

4.3.6 Error Feedback..... 89

4.3.7 Trigger View in Execution Mode..... 90

4.4 Summary..... 91

IMPLEMENTATION OF THE SELCU EXTENSIONS..... 93

5.1 Overview of the SELCU Extensions..... 93

5.2 Language Choice..... 94

5.2.1 Decision..... 95

5.2.2 Brief Introduction to the Selected Languages..... 96

5.3 Implementation Issues of The Auto Generation Software..... 97
5.3.1 Basic Tree Structure Function..... 97
5.3.2 The Lean Cuisine+ Attributes Function..... 100
5.4 Implementation Issues of the Execution Environment..... 103
5.4.1 Meneme Selection..... 103
5.4.2 Meneme Preview Mode..... 104
5.4.3 Task View Mode..... 106
5.4.4 Triggers View Mode..... 107
5.5 Summary..... 108

CASE STUDIES..... 109

6.1 Case Study One: Timetabling System..... 109
6.1.1 Main Functions of Timetabling System..... 110
6.1.2 Main Interface Components in Timetabling System GUI..... 111
6.1.3 Using the Auto-Generation Software..... 112
6.1.4 Using the Execution Environment..... 117
6.2 Case Study Two: Library Catalogue System..... 122
6.2.1 Library Catalogue System GUI Description..... 122
6.2.2 Using the Auto-Generation Software..... 125
6.2.3 Using the Execution Environment..... 129
6.3 Summary..... 142

CONCLUSIONS..... 143

7.1 Review of the Auto-Generation Software..... 143
7.2 Review of the Execution Environment..... 145
7.3 Conclusion..... 148

REFERENCES..... 149

APPENDIX A – USER MANUAL..... 159

A1 The Auto Generation Software..... 159
A2 The Execution Environment..... 162

FIGURES

Figure 1-1 The General Interaction Frame Work..... 2
Figure 1-2 Translations Between Components..... 3
Figure 1-3 Project Processing Sequence..... 6

Figure 2-1 Possible Interface for Style Menu Example..... 12
Figure 2-2 STN Diagram to Describe Style Menu Example..... 13
Figure 2-3 Using Petri Nets to Describe Style Example..... 16
Figure 2-4 Using State Charts to Present Style Example..... 18
Figure 2-5 (a) Mutually Compatible Sub-Dialogue..... 19
(b) Mutually Exclusive Sub-Dialogue..... 19
Figure 2-6 Using Lean Cuisine to Present Style Example..... 22
Figure 2-7 A Screen Shot of the StateCharts Program in Action..... 24
Figure 2-8 Graphic Editor of Stateflow..... 26
Figure 2-9 Example of VisualState..... 27
Figure 2-10 Main Interface of PetriSim..... 29
Figure 2-11 Example of HPSim..... 30
Figure 2-12 Early Interface Design in PowerPoint..... 33
Figure 2-13 User Interface Design Using the Delphi IDE..... 35
Figure 2-14 Visual C++ 6.0 Constructor..... 37

Figure 3-1 (a) Mutually Compatible Fork..... 43
(b) Mutually Exclusive Fork..... 43
Figure 3-2 (a) Original Lean Cuisine+ Diagram..... 44
(b) The Lean Cuisine+ Diagram after Hide "Children"..... 44
(c) The Lean Cuisine+ Diagram after Hide "Parents"..... 44
Figure 3-3 Column Attributes Selector..... 45
Figure 3-4 Lean Cuisine+ Tree Diagram for Column Attributes Example..... 47
Figure 3-5 Lean Cuisine+ Diagram with Conditions..... 48
Figure 3-6 Lean Cuisine+ Diagram Following Triggers..... 49

Figure 3-7 Task Overlay for Column Example..... 50
Figure 3-8 SELCU Interface..... 52
Figure 3-9 System Pull Down Menus..... 53
Figure 3-10 SELCU Tool Bar..... 54
Figure 3-11 Lean Cuisine+ Diagram with Conditions, Triggers and Tasks..... 55
Figure 3-12 Add First Meneme on Diagram..... 63
Figure 3-13 Add Mutually Compatible Child Menemes..... 64
Figure 3-14 Add Mutually Exclusive Meneme Group..... 65
Figure 3-15 Add a New Trigger..... 65
Figure 3-16 Add a Task..... 66
Figure 3-17 Add Task Name..... 67

Figure 4-1 Extended SELCU User Interface..... 81
Figure 4-2 Main Interface in Execution Mode..... 82
Figure 4-3 Preview of "Submit" Selection..... 84
Figure 4-4 Left Click on "Submit" Meneme..... 85
Figure 4-5 How to Access Task List in Execution Mode..... 86
Figure 4-6 Task List Display Sub-Window..... 87
Figure 4-7 Single Step Control Task View..... 88
Figure 4-8 System Presents All Task Steps in Sequence..... 89
Figure 4-9 Error Message Interface..... 89
Figure 4-10 Select a Trigger View..... 90
Figure 4-11 Triggers View in Execution Mode..... 91

Figure 5-1 Delphi Interface..... 97
Figure 5-2 Main Part of Example Code in "dfm" File..... 98
Figure 5-3 Basic Tree Structure for Delphi Interface..... 100
Figure 5-4 A Piece of Example Core Code in "pas" File..... 101
Figure 5-5 Main Auto-Generation Software Structure..... 102
Figure 5-6 Selecting a Meneme..... 103
Figure 5-7 Selecting a Monostable Meneme..... 103
Figure 5-8 Mutually Compatible Group..... 104
Figure 5-9 Mutually Exclusive Group..... 104

Figure 5-10 Error Feedback..... 104
Figure 5-11 Preview and Original States..... 105
Figure 5-12 Original States Before Preview..... 106
Figure 5-13 Preview Showing Trigger..... 106
Figure 5-14 Main Structure of the Execution Environment..... 108

Figure 6-1 GUI of Timetabling System..... 110
Figure 6-2 Generation of the Lean Cuisine+ Source File 113
Figure 6-3 The Generated Lean Cuisine+ Diagram for Case Study One 115
Figure 6-4 Lean Cuisine+ Diagram with Trigger Displayed 116
Figure 6-5 A Lean Cuisine+ Diagram in the SELCU System 117
Figure 6-6 Execution Environment for Case Study One 118
Figure 6-7 Preview for Selection of "Semester 2" Meneme 119
Figure 6-8 The Lean Cuisine+ Diagram after Release of the Right Button 120
Figure 6-9 Diagram after Left Clicking (Selecting) "List Clashes" Meneme 121
Figure 6-10 GUI of Library Catalogue System (Before Submit Request)..... 123
Figure 6-11 Library Catalogue System GUI (After Submit Request)..... 124
Figure 6-12 Lean Cuisine+ Diagram of Library Catalogue System..... 126
Figure 6-13 Lean Cuisine+ Diagram with Triggers for Library System..... 127
Figure 6-14 Lean Cuisine+ Diagram with Submit Book Request Task..... 128
Figure 6-15 Lean Cuisine+ Diagram in Execution Mode..... 129
Figure 6-16 Selecting a Monostable Meneme (First Three Seconds)..... 130
Figure 6-17 Meneme Preview Showing Trigger..... 131
Figure 6-18 Selection Propagation 132
Figure 6-19 System Interface (After Clicking the "Close" Meneme)..... 133
Figure 6-20 Selecting Triggers Function from Menu..... 134
Figure 6-21 Triggers View in Execution Mode..... 135
Figure 6-22 First Step of "Submit Book Request" Task..... 136
Figure 6-23 Second Step of "Submit Book Request" Task..... 137
Figure 6-24 Third Step of "Submit Book Request" Task..... 137
Figure 6-25 Fourth Step of "Submit Book Request" Task..... 138
Figure 6-26 Fifth Step of "Submit Book Request" Task..... 138
Figure 6-27 Sixth Step of "Submit Book Request" Task..... 139

Figure 6-28 Last Step of "Submit Book Request" Task..... 140
Figure 6-29 Error Feedback when the User Clicks an Unavailable Meneme..... 141
Figure 6-30 Error Feedback when the User Clicks a Virtual Meneme..... 141

Figure 7-1 Tandem Manipulation of the Lean Cuisine+ Diagram and the Interface. 146
Figure 7-2 Executing a Task both in Lean Cuisine+ and Delphi Interface..... 147

Figure A-1 Auto-Generation Software Interface..... 160
Figure A-2 Error Feedback..... 161
Figure A-3 Automatically Generate a Lean Cuisine+ Diagram Source File..... 161
Figure A-4 Interface of Execution Environment..... 162
Figure A-5 View Pull Down Menu..... 163
Figure A-6 Meneme Preview Interface..... 164
Figure A-7 Meneme Selection Interface..... 165
Figure A-8 Task Display Interface..... 165
Figure A-9 Task Execution Interface..... 166
Figure A-10 Task Views Interface..... 167

TABLES

Table 2-1 Relationship between Options..... 12
Table 2-2 Meneme Modifiers..... 21

Table 3-1 Trigger Actions..... 48
Table 3-2 The Meaning of Basic Meneme Description..... 59
Table 3-3 The Meaning of the Meneme Condition Description..... 60
Table 3-4 The Trigger Description..... 61
Table 3-5 The Meaning of the Basic Task Description..... 62
Table 3-6 Single Meneme Description in Task Sequence..... 63

Chapter 1

INTRODUCTION

Dialogue models can provide a useful input to user interface design. They provide a behavioral description of the dialogue at a level removed from the visible user interface. This description can be subjected to analysis, for example to check that all tasks uncovered at the requirements stage are supported by the system and that they can be carried out in an efficient manner. Lean Cuisine+ is a dialogue notation developed at Massey University (Phillips, 1995; Scogings, 2000). It is a semi-formal graphical notation for describing the behavior of event-based GUIs.

A shortcoming of current prototyping tools and user interface builders, is that although they permit the designer to construct the "look and feel" of the user interface, they provide no model of the interaction (Phillips and Scogings, 1997). Prototyping tools could be extended to produce a dialogue model as a by-product of the construction of the interface (Scogings and Phillips, 1998). In addition to supporting analysis of the interaction, the dialogue model would provide useful documentation.

In section 1.1 and section 1.2 of the chapter, some human-computer interaction and dialogue concepts are briefly reviewed. Section 1.3 introduces the goal of this project. The structure of the whole thesis is described in section 1.4.

1.1 Human-Computer Interaction Concepts

For most users, the interface is the computer, the system, or the application. What the user sees, hears, feels, are the artifacts of the user interface. Interaction models help the user to understand what is going on in the interaction between user and system. They address the translations between what the user wants and what the system does.

The purpose of an interactive system is to aid a user in accomplishing goals from some application domain. A domain defines an area of expertise and knowledge in some real-world activity. A domain consists of concepts that highlight its important aspects.

Tasks are operations to manipulate the concepts of a domain. A goal is the desired output from a performed task, and an intention is a specific action required to meet the goal. The task analysis involves the identification of the problem space for the user of an interactive system in terms of the domain, goals, intentions and tasks.

Figure 1-1 presents an interaction framework (Abowd and Beale, 1991). It provides a complete description of the interaction by including the system explicitly, and breaks it into four main components.

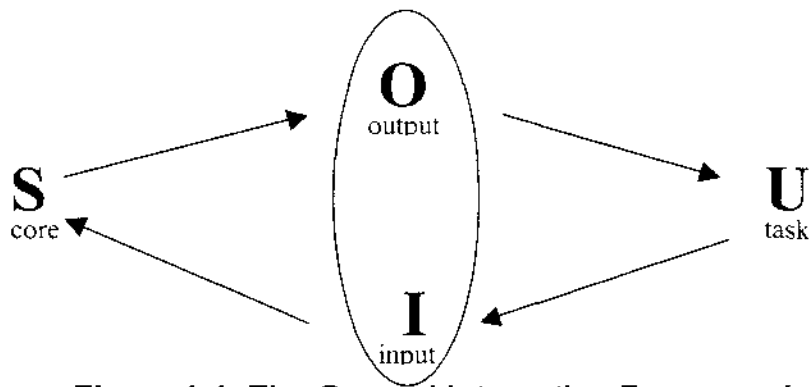


Figure 1-1: The General Interaction Frame work

The nodes represent the four major components in an interactive system --- the system (S), the user (U), the input (I) and the output (O). Each component has its own language. In addition to the uers's task language and the system's core language, there are languages for both the input and output components to represent these separate, though possibly overlapping, components. Input and output together form the interface.

Figure 1-2 presents the translations between the components from figure 1-1. As the interface sits between the user and the system, there are four steps in the interactive cycle, each corresponding to a translation from one component to another.

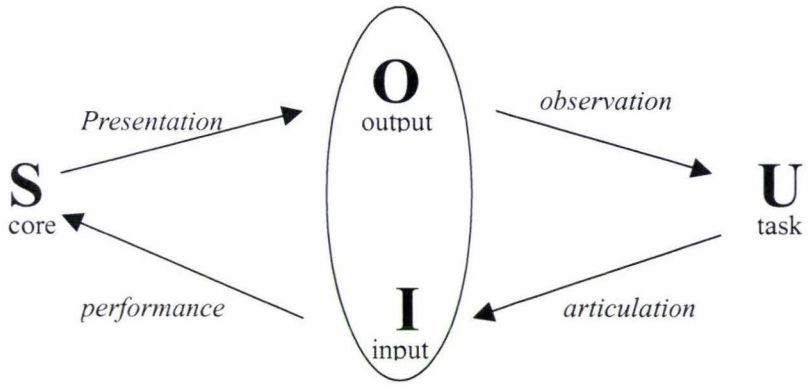


Figure 1-2: Translations Between Components

Four main translations are involved in the interaction: articulation, performance, presentation and observation. The user (U) begins the interactive cycle with the formulation of a goal and a task to achieve that goal. The only way the user can manipulate the machine is through the input (I), and so the task must be articulated within the input language. The input language is translated into the core language as operations to be performed by the system (S). The system (S) then transforms itself as described by the operation translated from the input (I); the execution phase of the cycle is complete and the evaluation phase now begins.

The system (S) is in a new state, which must now be communicated to the user (U). The current values of system attributes are rendered as concepts or features of the output (O). It is then up to the user (U) to observe the output (O) and assess the result of the interaction relative to the original goal, ending the evaluation phase and, hence, the interactive cycle.

1.2 Dialogue Concepts

Dialog is the syntactic level of human computer interaction; it is rather like the script of a play, except the user, and sometimes the computer, has more choices. The dialog between user and system is influenced by the style of the interface. It is a conversation between two or more parties. It has also come to imply a level of cooperation or at least intent to resolve conflict. The dialog is linked to the semantics of the system to present "what it does", and also is linked to the presentation of the system to describe "how it looks".

The user can look at computer language at three levels:

- **The Lexical Level**

  This is the lowest level, and is concerned with the physical tokens exchanged. For example, the shape of icons on the screen; the actual keys pressed and so on.

- **The Syntactic Level**

  This is the middle level. It focuses on the order and structure of inputs and outputs which make up the dialogue. In human language, it presents the grammar of sentence construction.

- **The Semantic Level**

  This is the highest level. The meaning of the conversation in terms of its effect on the computer's internal data structures and/or the external world. In human language, it ascribed by the different participants to the conversation.

In user interfaces, the term dialog is often taken to be almost synonymous with the syntactic level. However, the lexical/syntactic barrier is somewhat fuzzy and actual use of dialog description often includes some lexical features. The Lean Cuisine+ notation, which forms the basis of this project, focuses on the syntactic level.

1.3 Project Goal

A software environment (SELCU) has been developed for Lean Cuisine+ (Scogings, 2003) which permits dialogue models to be manually constructed and edited using a drag and drop approach.

This project is concerned with extending SELCU. It will involve:

- Design and implementation of the automatic generation of Lean Cuisine+ descriptions for user interface prototypes constructed using a conventional user interface builder (Delphi). The descriptions produced will be in a format compatible with the SELCU application.

- Extension of SELCU to support the execution of Lean Cuisine+ specifications, whether developed manually or via a prototyping tool.

The overall sequence of activities to achieve this is shown in figure 1-3.

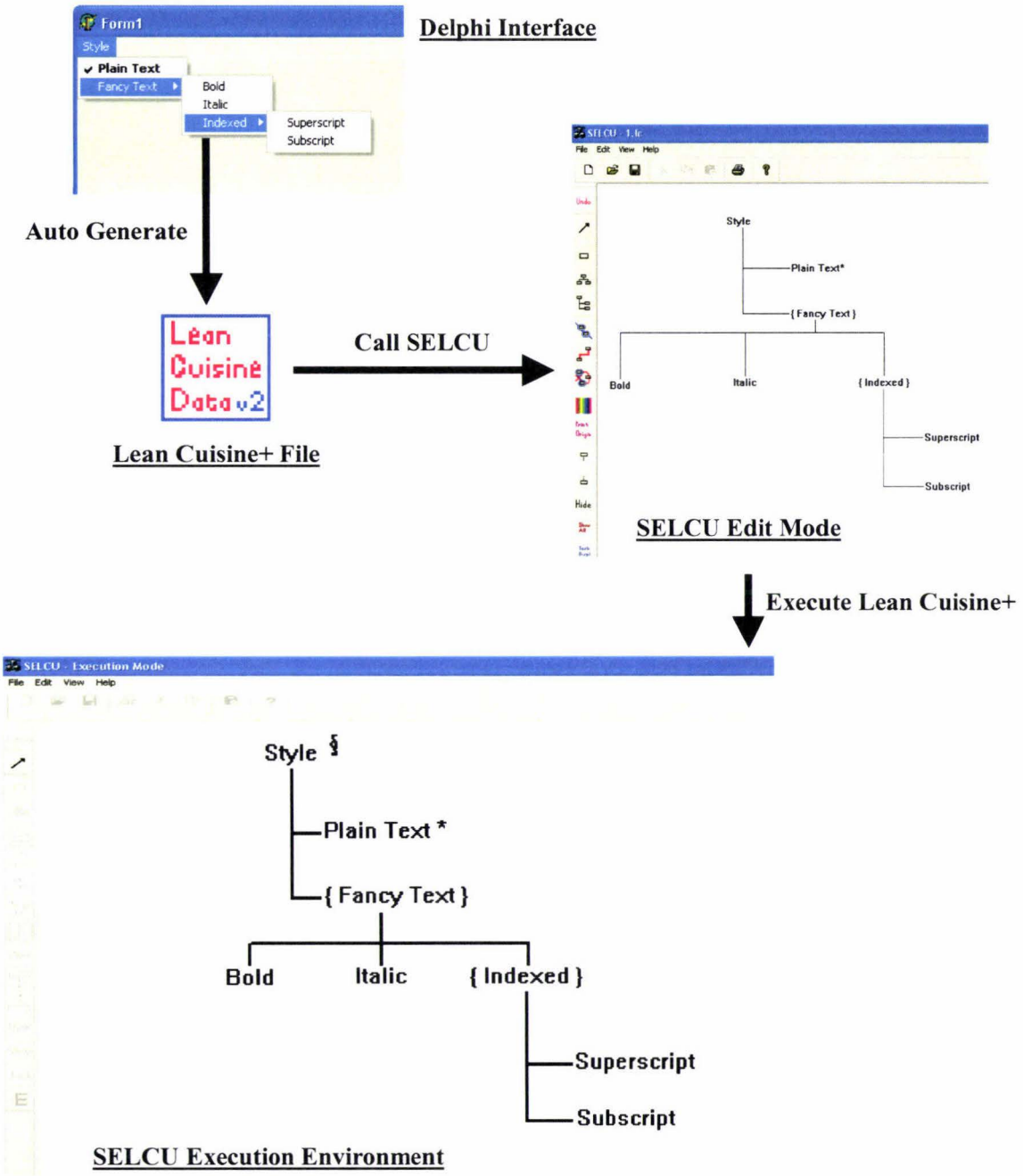


Figure 1-3: Project Processing Sequence

In figure 1-3, the user designs the interface in a Delphi IDE, and then uses the Auto-Generation software to generate a SELCU compatible Lean Cuisine+ file. After that, the user can call SELCU to present the generation file in a Lean

Cuisine+ diagram. In SELCU, the user changes to the execution environment to execute the notation.

Detailed functional requirements are presented in section 3.3

1.4 Structure of the Thesis

This thesis is structured into seven chapters including this one. The remaining chapters are organized as follows:

- **Chapter two** reviews dialogue notations with a focus on graphical notations. Support environments, a range of early prototyping tools and IDEs are also described in this chapter.

- **Chapter three** introduces the main principles of the Lean Cuisine+ notation. The support environment (SELCU) for Lean Cuisine+ is described in detail. The proposed extensions of SELCU (automatic generation and execution environment) are presented.

- **Chapter four** describes the conceptual design of the SELCU extensions, including design decisions to satisfy the functional requirements. Lo-Fi prototyping and all transferable Delphi interface components are discussed.

- **Chapter five** discusses the implementation of the Auto-Generation software and Execution Environment. These are developed in Delphi and C++. Several important implementation issues are discussed in this chapter.

- **Chapter six** presents two detailed case studies to show the software in action.

- **Chapter seven** presents a set of conclusions. Proposals for future enhancements and extensions are also considered in this chapter.

Chapter 2

LITERATURE REVIEW

In this chapter, we will review some essential human computer interaction knowledge. The chapter will cover dialogue notations, their support environments, prototyping tools and integrated development environments.

In section 2.1, we will review dialogue notations, with a focus on graphical notations, and explain their advantages and shortcomings.

In section 2.2, our attention will move to the support environments for dialogue notations. The principles of the support environments will be discussed, and some examples described.

In section 2.3, a range of today's early prototyping tools and IDEs are reviewed. Examples of software development tools will be described, and a general observation on them will be taken.

2.1 Dialogue Notations

Dialog is the syntactic level of human computer interaction. It is the structure of the conversation between the user and the computer system. Dialogue notations describe the dialogue structure in terms of events, sequence, hierarchy, states and other elements.

Usually, the dialog will be linked to the semantics of the system (what it does) and the presentation of the system (how it looks) (Calder and Thomas, 1998).

The ease of analysis and separation of the interface elements of the program from the semantic is the main justification for a dialog description (Baecker, Grudin and Buxton, 1995). The dialog description will allow the user to analyze the proposed structure, or perhaps use a prototyping tool to execute the dialog before a program is written.

Notations used for dialog description can be separated into two main groups, which are textual notations and graphical notations (also referred to as diagrammatic notations). The textual notations are typically based on formal grammars (such as BNF) to describe the system, which provide for formal analysis (Harel, 1988). The graphical notations use visual pictures or diagrams to present the dialog, making it more accessible to users.

2.1.1 Textual Dialogue Notations

The textual dialogue notation is one kind of special purpose and context free language which is used to describe the interface. Formal grammars and production rules are two popular textual notations.

Formal grammars have been applied to describing programming languages and interactive dialogues. Regular expressions are heavily used in editors to describe complex textual search criteria. One advantage of using a formal grammar is that it is readily executed using existing tools. This kind of tool is now also readily available on PCs. The tools are probably more suitable for parsing text commands than for graphical input, but this kind of formal grammar doesn't

really show sequence well at a surface level. It can be deduced but is not readily apparent.

2.1.2 Graphical Dialogue Notations

Graphical notations are also called diagrammatic notations. They provide a visual representation of dialogue structure, and are used in the dialog design area (Brown and Marshall, 1998). They allow the user to see the structure of the dialogue. Graphical notations are the main focus of this thesis.

To be effective, graphical notations should describe the external behaviour precisely, and match the user's model (Phillips, 1994). Ideally, graphical notations should be directly executable, should separate function from implementation and should not constrain the implementation (Jacob, 1985). Graphical notations should be easier to understand and produce than conventional symbolic code (Phillips, 1990).

In recent years, a variety of graphical notations have been developed to represent dialogs. We can separate them into three groups: sequence based (e.g. State transition networks, Petri nets), state based (e.g. State chart diagrams) and object based (e.g. Lean Cuisine and Lean Cuisine+). In this section we will select some representative notations from these three groups to review the graphical notations world.

State Transition Networks

State transition networks (STNs) have long been used for dialog description, and they are based on a set of nodes and links between them. The nodes represent states which the system can reach. When we use the STN to describe a system,

each link inside the STN will represent a transition between states and is associated with an input event which triggers it. The transition may be conditional. Each diagram defines the legal sequences of input events for some dialogue. When we describe dialogues, enhancements have been made to provide for hierarchies of diagrams, to allow the system actions to be attached to arcs, and to encode some states within internal registers. STNs provide an easily displayed and manipulated network representation based on a small number of constructs.

For example, suppose we have a menu called "Style", offering five options: *Plain Text*, *Bold*, *Italic*, *Superscript* and *Subscript*. The relationship between these options is presented in table 2-1, a possible interface in figure 2-1, and STN diagram in figure 2-2.

	Plain Text	**Bold**	**Italic**	**Superscript**	**Subscript**
Plain Text		E	E	E	E
Bold	E		C	C	C
Italic	E	C		C	C
Superscript	E	C	C		E
Subscript	E	C	C	E	

E = "Mutual Exclusive" C = "Mutual Compatible"

Table 2-1 Relationship between Options

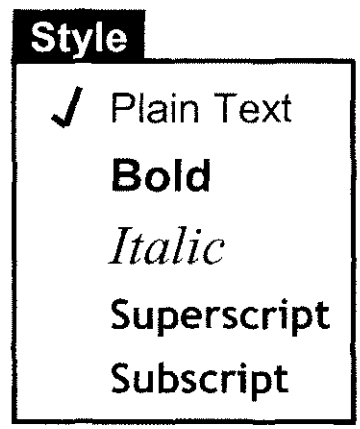


Figure 2-1 Possible Interface of Style Menu Example
(after Apperley and Spence, 1989)

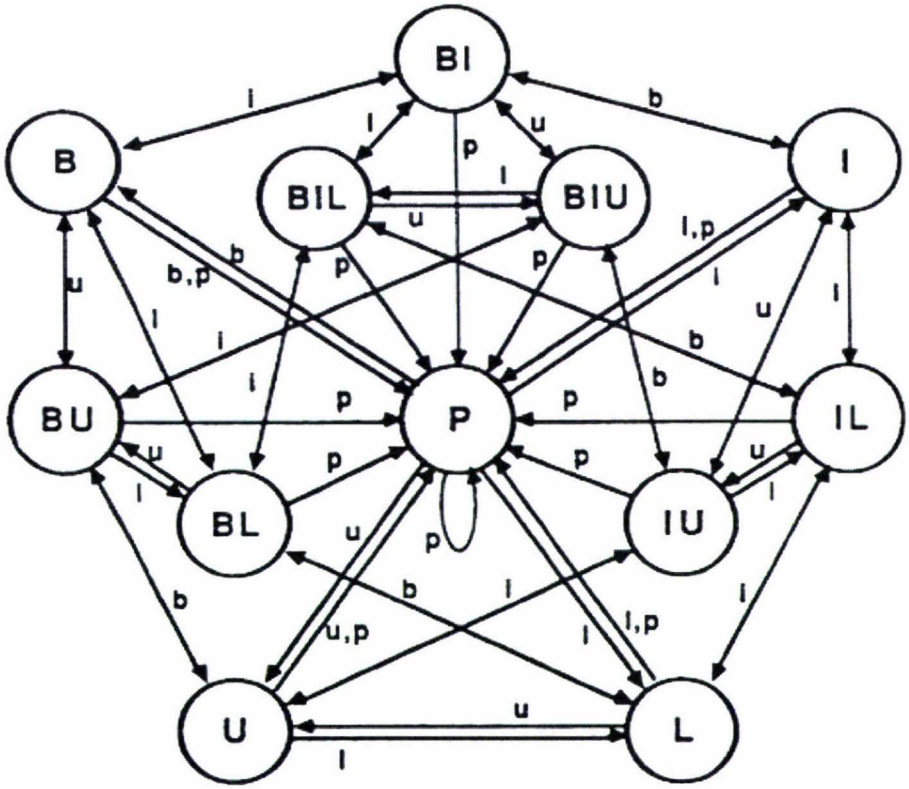


Figure 2-2 STN diagram to describe style menu example
(Apperley and Spence, 1989)

There are several shortcomings in STNs. They predate direct manipulation interfaces, and suffer from a number of shortcomings in this respect, notably that they are based on a "single event / single state" model, which can lead to an exponential growth in states and transitions in asynchronous dialogues, resulting in large and very complex diagrams. In their basic form, they are flat (lack depth) and do not capture the structure of complex dialogues well. STNs are inherently sequential in nature, it means that it do not support parallelism. They are best suited to sequential dialogues with limited choice, where they can be very effective.

Petri Nets

Petri nets were originally developed in response to the limited modeling power of finite state machines. The Petri Net is a graphical formalism designed for reasoning about concurrent activities (Kjeld and Heiko, 2002). Petri nets define possible sequences of events in a modeled system, and they also show the system states (combinations of conditions) which result from the occurrence of these events (Zurawski and Zhou, 1994). They are abstract virtual machines with a well defined behaviour, which have been used to specify process synchrony during the design phase of time-critical applications.

In recent years, Petri nets have been used by several researchers to specify aspects of single user and multi-user systems (Murata, 1989). Petri nets are represented as network graphs comprising two types of node: circles (places) and bars (transitions) connected by directed arcs. They model the static properties of a system, but also have dynamic properties that result from their execution. In a STN the system is always at exactly one state (Peterson, 1981) (In the Petri net, indeed, we can simulate the behaviour of the system by moving a counter around the STN following arcs). A Petri net is more powerful in that the system can have several "states" at once.

Like STDs, Petri nets define possible sequences of events in a modeled system. In the modeling of systems, places represent conditions, and transitions represent events. Places and transitions may be labeled to indicate the intent of the model, but Petri nets are un-interpreted models. For an event to occur, it may be necessary for certain "pre-conditions" to hold. During execution, tokens (the black dots) move from place to place by the firing of the transitions of the net.

Here we use Petri Nets to describe the same example as for the STN, as shown in figure 2-3. There are two main problems in this figure which need to be considered.

- **Problem One: It consists of 3 separate fragments**

  This problem addressed by looking at the correct mathematical definition of a Petri net. A Petri net is composed of 4 parts: a set of places P, a set of transitions T, an input function I and an output function O. A Petri net graph is the graphical representation of the Petri net and follows the rules of graph theory. It is possible to produce a graph that graphically consists of separate fragments. In fact, such graphs even have a name – they are called disconnected (or unconnected) graphs. Thus a Petri net graph as a formal mathematical theory can have separate fragments. It is possible that a convention exists which states "when modelling with Petri nets, all graphs should be connected".

- **Problem Two: The condition "Plain is selected" is not represented.**

  There is an easy way to solve it and that is to make a Petri net that is an exact duplicate of the STD in Figure 2-2 (apart from slight graphical discrepancies between the two notations). Peterson explicitly discusses how to turn STDs into Peri nets. Interestingly enough, in these cases he seems quite happy that places in the Petri net represent states and not conditions.

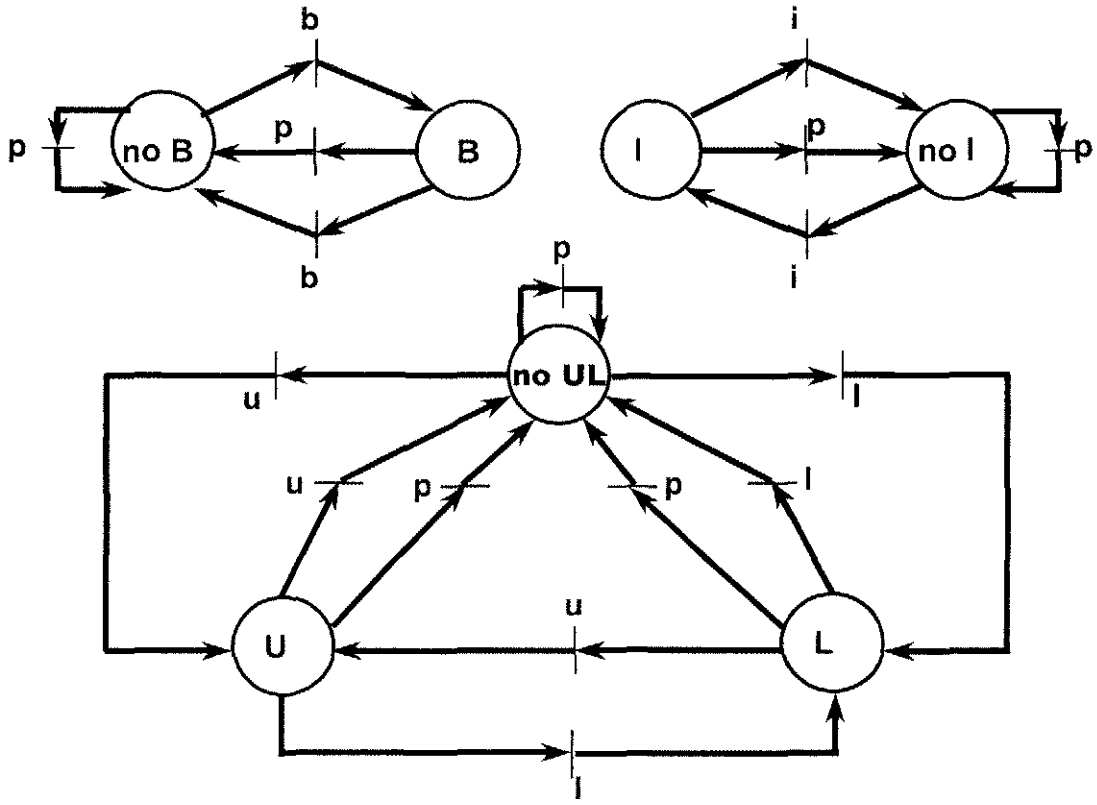


Figure 2-3: Using Petri Nets to describe Style example

Although a Petri net is a quite good tool to describe the dialog, it still has several shortcomings we cannot cope with. Firstly, the objects are represented only indirectly by the sum of their states, which means that object and their behaviours are not well defined. Inter-object relationships are not presented at all (Wolfgang,1985). The latter problem is compounded to an extent by the requirement to represent separately, selected and deselected object states, which leads to additional transitions. The need to show all transitions explicitly clutters and detracts from the diagrams (Jensen, 1992). This is especially problematic where a number of transitions relate to a particular object. Lastly, sub-states are not supported by Petri nets.

State Charts

State charts were developed for the specification of complex reactive systems, which are event driven and continuously react to stimuli (internal and external). Such systems are described in terms of the state changes resulting from events, with events grouped in connection with change of state (Harel, 1987).

State charts are a higraph-based extension of the STNs formalism and are state-based. Areas represent states and labeled directed arcs represent transitions. They were developed for describing the behavior of complex reactive systems. State Charts provide the following extensions to traditional STNs:

- Hierarchy, the ability to group states into super-states to any level of nesting (which provides a basis for iterative refinement);
- Orthogonality, the ability to describe the transition of several independent states concurrently;
- Communication, the ability to trigger several transitions by broadcasting an event to several states.

The concepts that state charts add to basic STNs: hierarchy, orthogonality and broadcast events, provide expressive power and reduce the size of the resulting specification. The hierarchy in state charts is used within a single diagram to add structure, and to show which parts represent alternative states and which represent concurrent activity (Harel and Politi, 1998).

Statecharts have been criticized as being too difficult for the designer. The need to show transitions explicitly remains, and is compounded to an extent by the requirement for "null" states to be shown. Figure 2-4 shows a State Charts description of the "Style Menu" example.

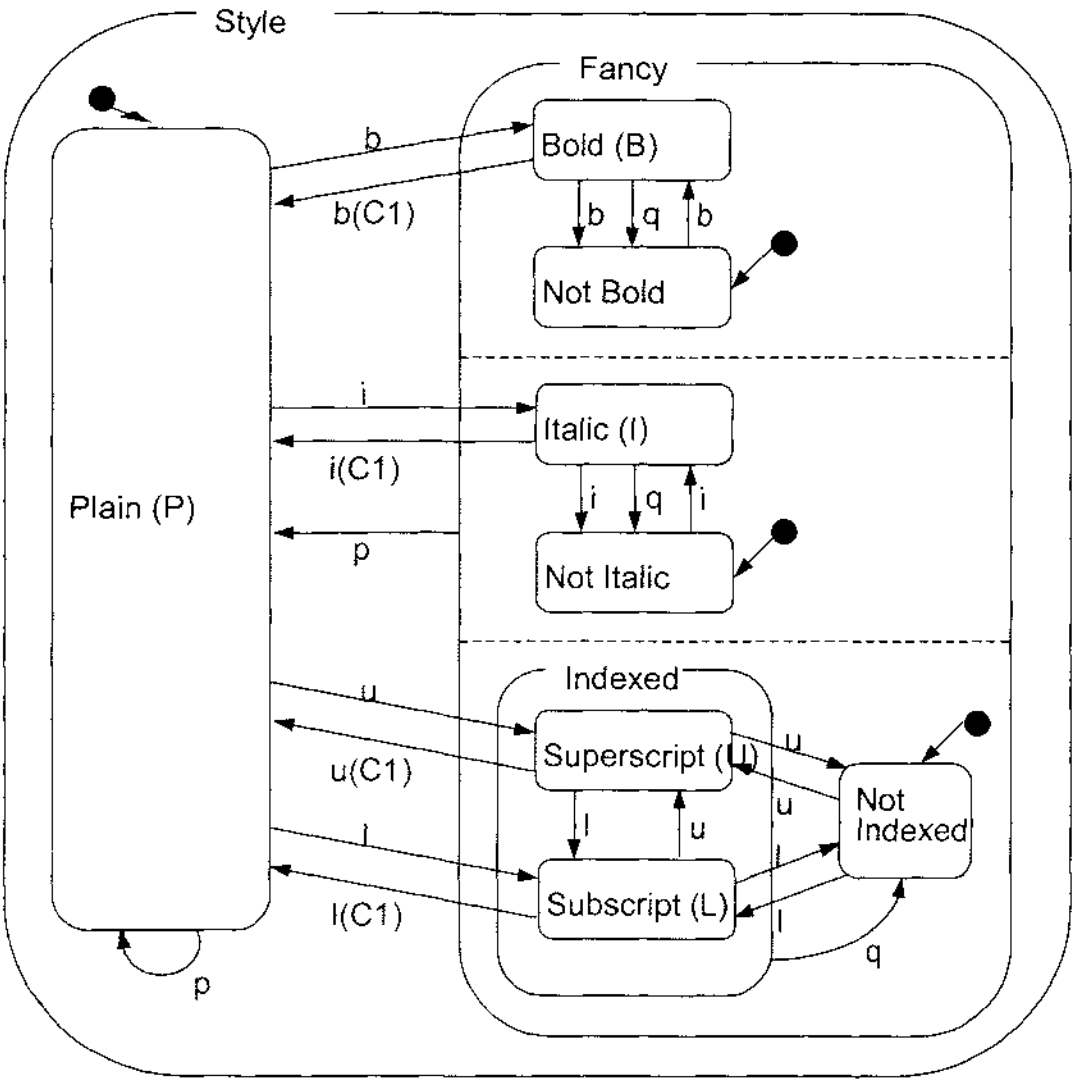


Figure 2-4: Using State Charts to Present Style Example

From the above discussion we know that the state charts were not developed for dialogue description, but are suited to describing asynchronous event-driven user interfaces. They capture the event sequences, but are not inherently sequence-based in the way that STNs are, and they also capture the structure of complex dialogues reasonably well through the nesting of states (Fung, Kemp, R.H., and Kemp, E.A, 1998). State charts consist of multiple orthogonal inter-communicating STNs, and therefore support multiple active states, but on

the other hand, they are state-based rather than object-based, which can lead to complexity in representing direct multiple interfaces.

Lean Cuisine

Lean Cuisine (Apperley and Spence, 1989) is a graphical notation based on the use of tree diagrams to describe systems of menus. A menu is viewed as a set of selectable representations (called menemes) of objects, actions, states, parameters and so on, which may be logically related or constrained. Lean Cuisine was developed explicitly for dialogue description.

- **Menu**

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

- **Sub-Menu**

  Within a dialogue, 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 figure 2-5.

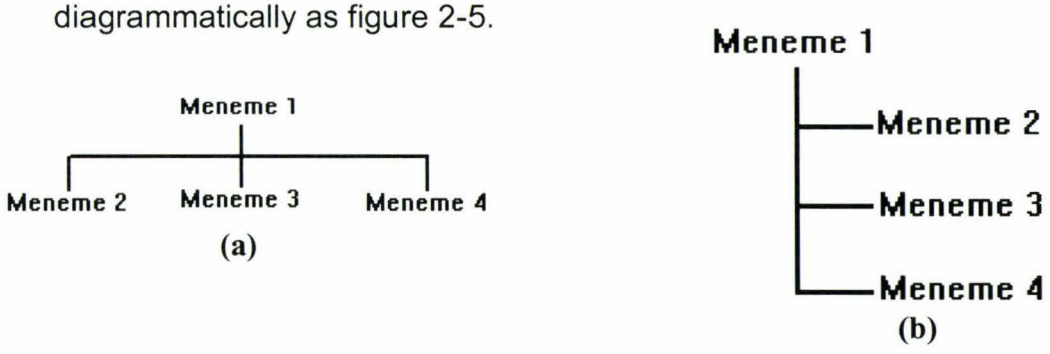


Figure 2-5 (a): Mutually Compatible Sub-Dialogue

(b): Mutually Exclusive Sub-Dialogue

- **Menemes**

  The meneme is an individual selectable representation within a menu, it is the minimum or basic unit of information in the two way dialogue between the user and the application (Apperley, 1988). A meneme is defined as having four possible states "available and selected", "available but not selected", "not available but selected" and "not available and not selected". Its state may be change either by direct excitation, or by indirect modification (that means as a result of the excitation of another meneme) (Phillips and Apperley, 1990).

- **Real or Virtual Menemes**

  Mnemes may be "real" or "virtual". Associated with each real meneme is an implicit event with may trigger its selection or de-selection. Real menemes may be terminal (leaf) nodes, in which case they represent specific selectable objects, actions or parameters (Anderson and Apperley, 1990). If the real menemes are non-terminal nodes, they are selectable headers to other menus. Virtual non-terminal menemes are used to partition a single menu, and are not directly accessible to the user. They can be indirectly selected as a result of a valid selection having been made in the menu subsection they represent.

- **Meneme Designators**

  A meneme can have modifiers, which capture other aspect of a dialogue. These include the following:

  (1) 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.

(2) 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.

(3) 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 values.

(4) 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 the table 2-2).

Icon	Meaning
§	***Required choice***, which is present on a group header where a valid choice must exist within the group
*	***Default choice***, which is present on a terminal node within a required choice group
•	***Dynamic default***, which is used at terminal node or group header levels, where the default is the last selection within the group
↑	***Select-only***, which is always shown on a terminal node, and means that user can select but cannot de-select
↓	***De-select-only***, which is on a terminal node, and means that user can deselect but cannot select
⊥	***Monostable***, which is used on a terminal node representing a finite action, and means that the meneme is automatically deselected on action completion
⊗	***Passive***, which is present on the terminal node, and means that the user cannot select this option

Table 2-2 Meneme Modifiers

By using the above modifiers, the Lean Cuisine notation can show meneme states in a simple and clear way. Figure 2-6 is the Lean Cuisine diagram used to describe the Style menu interface. Compared with STN, Petri Net or State Chart representations, the conciseness of Lean Cuisine notation is shown very clearly in this example.

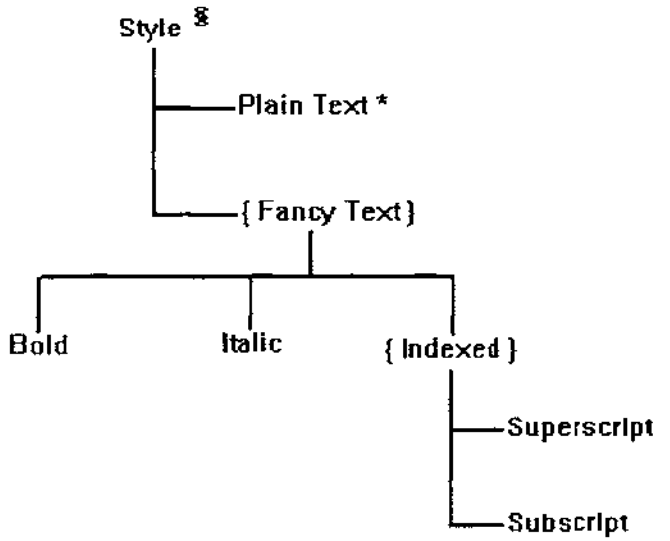


Figure 2-6 Using Lean Cuisine to Present Style example
(after Apperley and Spence, 1989)

Lean Cuisine offers a clear, concise, and compact graphical notation for describing asynchronous aspects of menu-based dialogues, but is not able to describe those aspects which include sequence (Cockton, 1990). The state of a dialogue at any stage is represented by the sum of the individual meneme states (Phillips, 1992). The events in Lean Cuisine are not directly represented but are implicitly associated with menemes. The objects are directly represented, but not visually distinguished from other elements, and the grouping of actions is around objects. Event sequences are not explicitly captured, except via hierarchy. Finally, the Lean Cuisine diagram does not describe how the menemes are selected, or the dynamics of menu use.

Lean Cuisine+

Lean Cuisine+ is a development and extension of the Lean Cuisine notation. It is an executable semi-formal graphical notation for specifying the underlying behaviour of event-based direct manipulation interfaces (Phillips, 1994). It supports the early design phase of the interface development cycle. In a Lean Cuisine+ specification, an interface is described in terms of a basic dialogue tree plus additional constraints and dependencies (Phillips, 1995).

This project is based on the Lean Cuisine+ notation; which will be described in detail in chapter 3.

2.2 Support Environments for Dialogue Notations

The Lean Cuisine+, Statechart and Petri Net notations have been reviewed in section 2.1. Several environments have been developed to support these executable notations. SELCU (Scogings and Phillips, 1999), a Lean Cuisine+ support environment is described in chapter 3. Support environments for State Charts and Petri Nets are reviewed below.

2.2.1 State charts

StateCharts

Statecharts is a statechart design program. It supports the construction of statecharts (Harel, 1987), and provides some method of generalizing the use of statecharts for purposes such as device tuition. Figure 2-7 shows a screen shot of the *Statecharts* program in action.

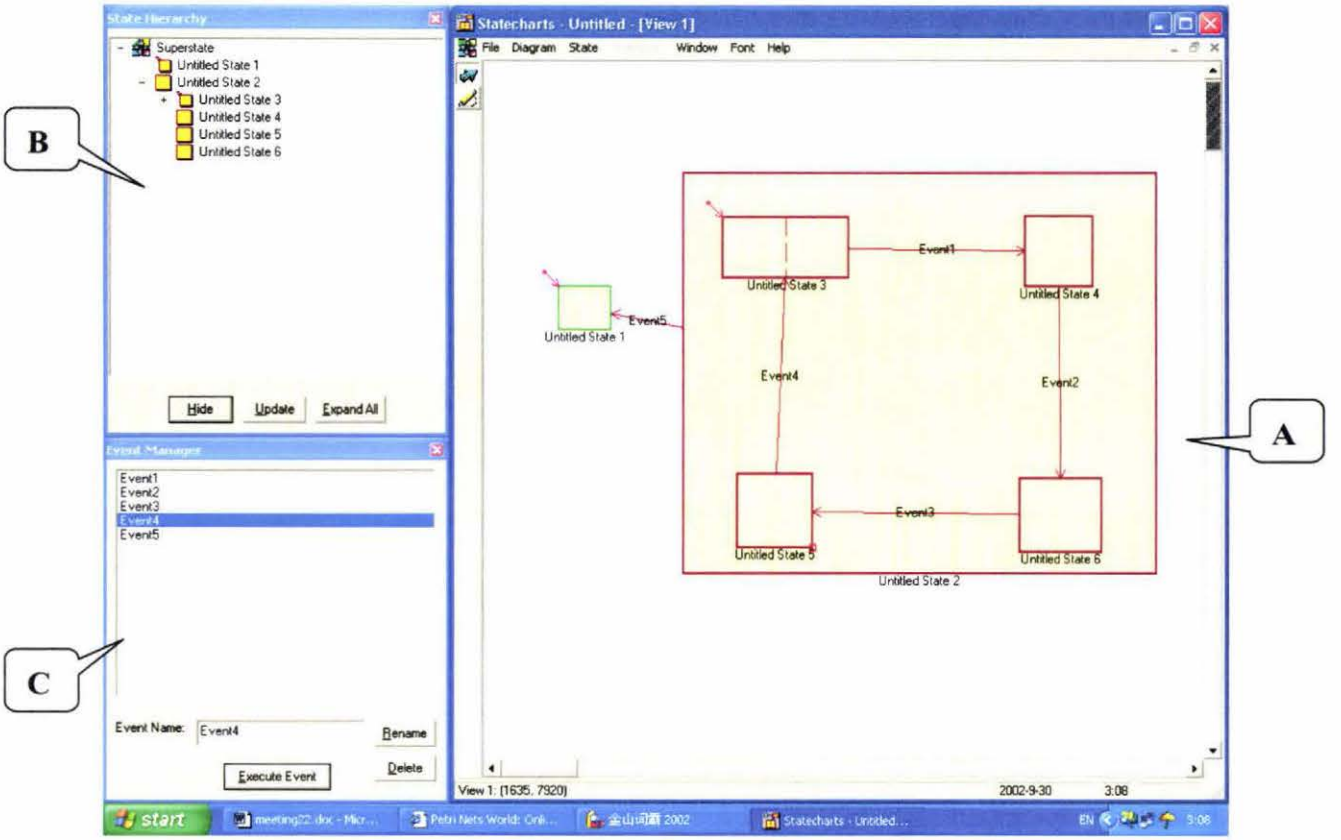


Figure 2-7: A screen Shot of the *Statecharts* program in action

There are three work areas in Figure 2-7.

Area A includes the menu bar, tool bar and main view window of this software. All the system functions are presented in the menu bar, and this software also uses the tool bar to give a fast way to access select and new state functions. The entire diagram is shown in the main view window. The user draws or modifies state chart diagrams in this area.

Area B is a state hierarchy window. It displays the hierarchy of states in a tree form, with the default state at each level at the top of its branch and with a different icon.

Area C is an event manager. It controls the execution of the diagram. From the list of events in the event manager, the modeller can select any number of events to assert, and then click the "Execute Event" button to run the state chart.

Stateflow

Stateflow is an interactive design tool for modeling and simulating event-driven systems. It provides for designing embedded systems that contain supervisory logic. Its combination of graphical modeling and animated simulation brings system specification and design closer together.

Stateflow is based on a combination of traditional statechart diagrams and control flow diagrams. *Stateflow* charts enable graphical representation of hierarchical and parallel states and the event-driven transitions between them. However, *Stateflow* also augments traditional statecharts with the innovative capabilities of control flow, graphical functions, temporal operators, directed-event broadcasting, and modeling support for legacy C code.

Figure 2-8 presents the Graphic Editor of *Stateflow*, and it features a number of built-in functions which include the following:

- Area A is a graphical Stateflow Explorer. It lets the user draw, browse, define, and modify objects in this window.
- Area B is a tool bar. It includes the most popular functions in this software.
- Area C includes shortcut bars, navigation and command menus, zoom controls, and undo/redo capability in this area.

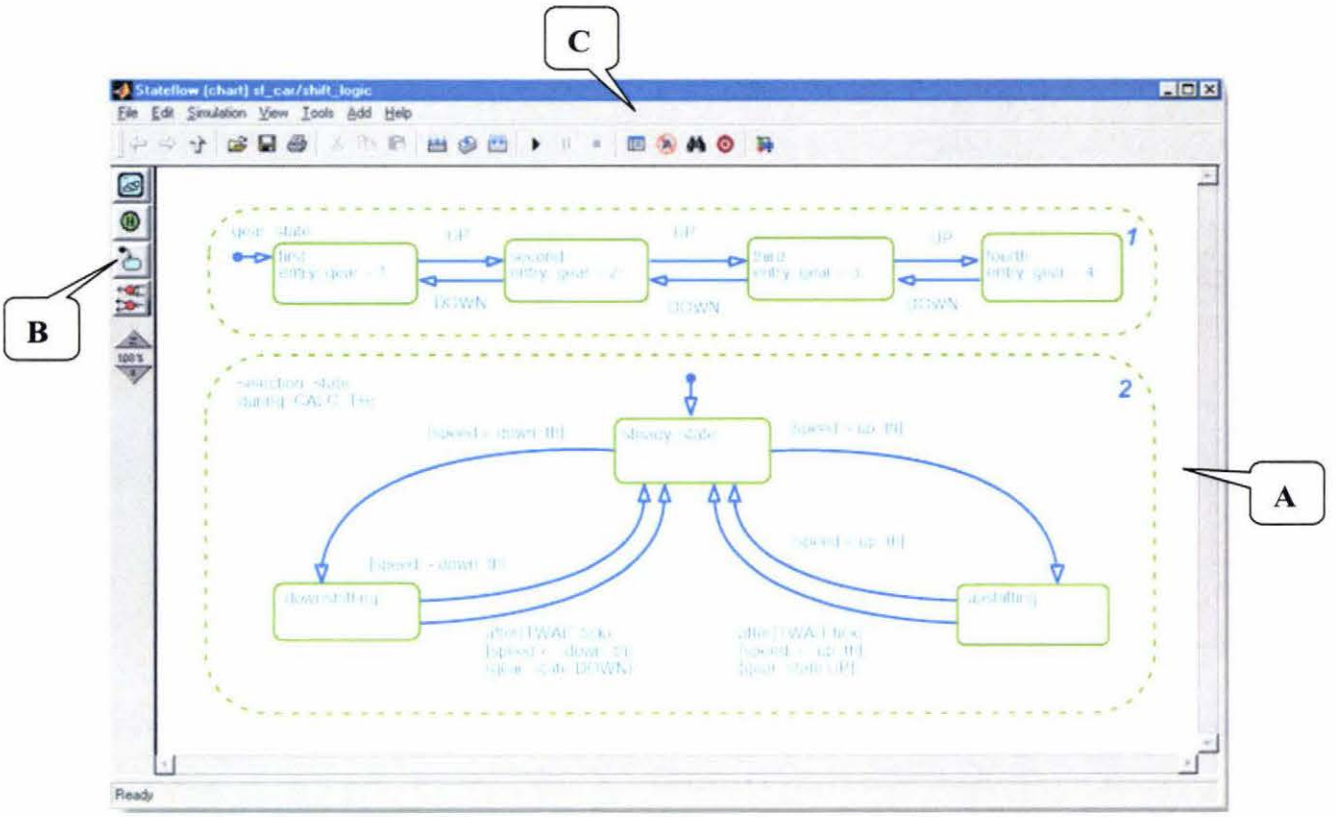


Figure 2-8 Graphic Editor of *Stateflow*

Using *Stateflow*, the user can develop visual models of event-driven systems that incorporate state transition diagrams without knowing finite state machine theory. Users can also generate, embeddable C code from their models with *Stateflow* Coder. These features make the *Stateflow* environment suitable for developing embedded systems in automotive, aerospace, and telecommunications design applications.

VisualState

VisualState is a graphical programming tool based on Statecharts and Flowcharts. With graphical specification, automatic code generation, graphical debugging, and round- trip engineering. Figure 2-9 is a screenshot of the *visualState*.

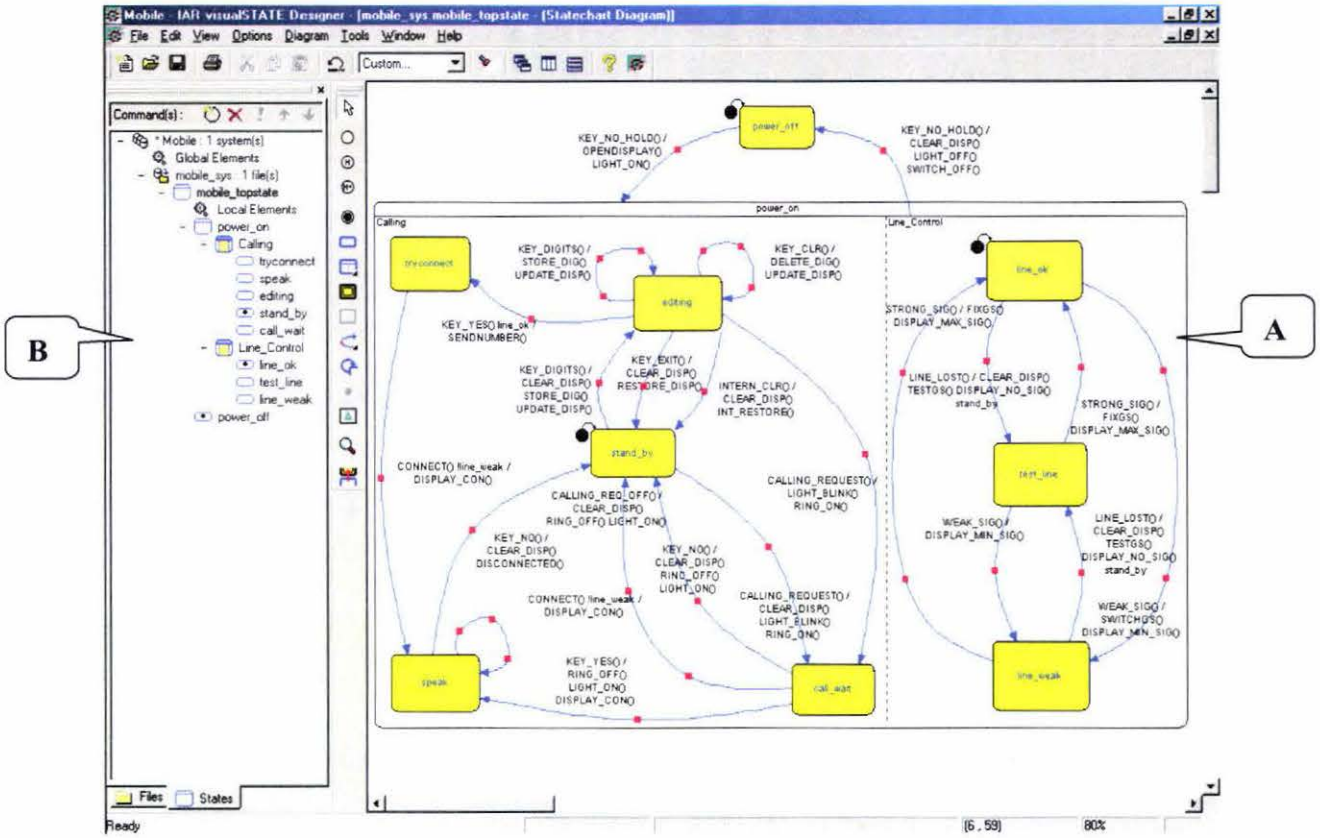


Figure 2-9 Example of *VisualState*

There are two main areas in above figure:

In area A, users can create their model using states, transitions, events, actions, initial states, variables, assignments, concurrent regions, unit states, history states, deep history states, guards, signals, parameters, entry, exit and do reactions - according to the UML notation.

Area B is a tree browser. It allows the user to overview and navigate through a project. All elements of the system are visible and accessible through the browser, and a double-click on the desired element brings up the appropriate drawing on the screen.

VisualState provides a direct and formal link between user requirements and software implementation by allowing the user to create a complete, executable

specification. Operating on an engineering workstation or PC, *VisualState* creates a visual, graphical specification that represents the intended functions and behavior of the system being specified. This specification may be executed, or graphically simulated, so the system engineer can explore "what if" scenarios to determine if the behavior and the interactions between system elements are correct.

2.2.2 Petri Nets

PetriSim

PetriSim is a generator of discrete simulation models that can also be used as an editor and simulator of Petri networks. Simulation models are created by adding Pascal code snippets into the network created by *PetriSim*. It enables creation of discrete simulation models in a commonly known language with the simplicity typical for simulation languages.

PetriSim was created as a tool to support the teaching of:

- Petri Networks
- Discrete Simulation
- Object Oriented Programming

The main interface of the system in action is shown in figure 2-10.

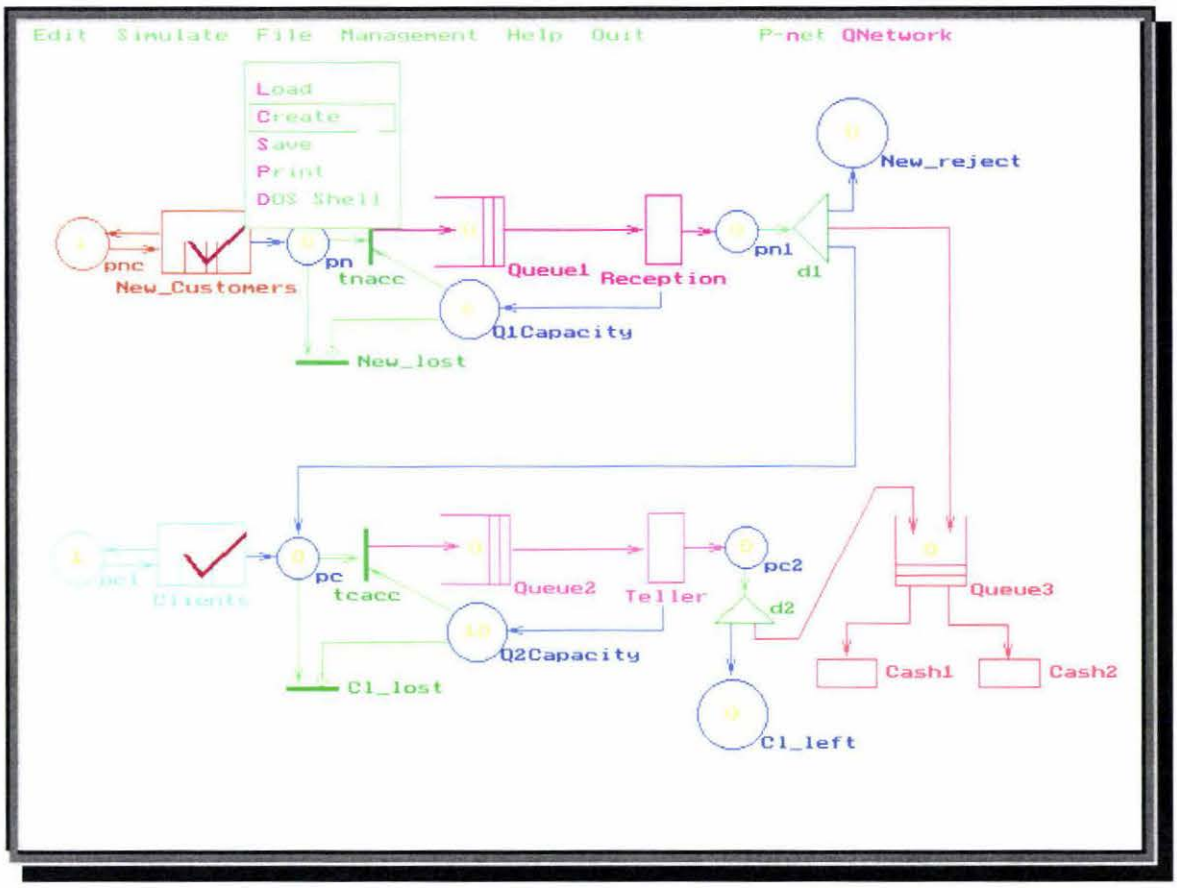


Figure 2-10 Main Interface of *PetriSim*

Because PetriSim is based on the Dos system, its interface looks quite simple. In the above figure, the main screen is this software's basic working area. The user can use the mouse and keyboard to design a Petri Net diagram on this area. A set of menu bars are at the top of the window. At any time, the user can use the mouse to click the menu bar to display a pop down sub-menu (for example the user can find a ***create*** new file function by single clicking ***File*** menu option). On the other hand, the user also can use a shortcut key to launch the function.

HPSim

HPSim is a high-level Petri net editor and simulator that achieves the integration of Petri nets and the Java programming language. Petri nets can implement

methods and can be treated as first-class Java objects. On the other hand, Java code can be accessed from nets easily. Figure 2-11 presents a screenshot of the operation of *HPSim*.

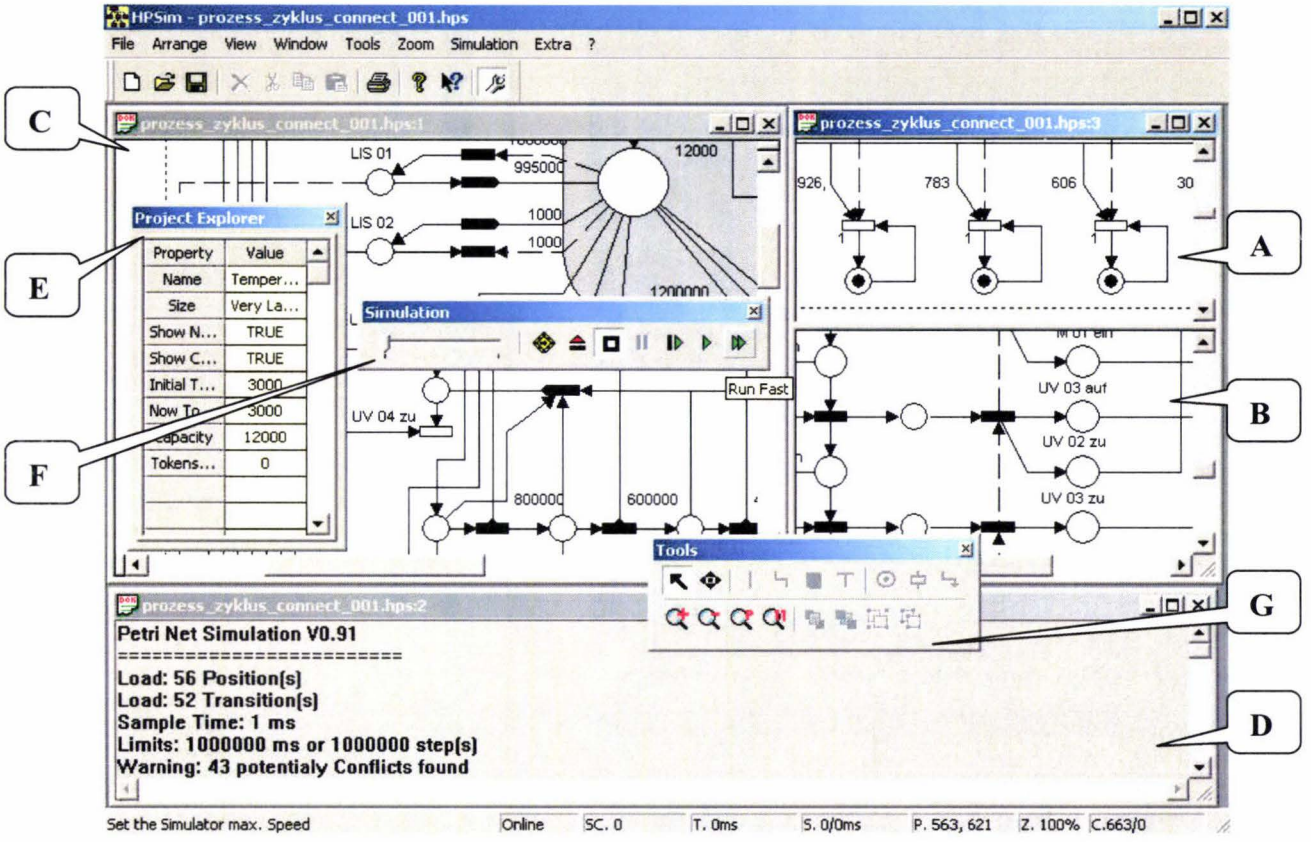


Figure 2-11 Example of *HPSim*

In figure 2-11, several documents have been sketched and simulated in a common main window (areas A, B and C). The graphic objects in the diagram can be positioned, moved, and deleted. The text and geometrical objects are available for annotations (area D). The project explorer presents all the diagram components' property and value (area E). Users can click on the contents and modify them. Area F is a simulator. A multi-level zoom function control window is also presented in this figure (area G).

Several support environments for State Chart and Petri Nets have been introduced in this section. *StateCharts*, *Stateflow* and *visualState* are used to

construct state chart diagrams. The common features include the design, editing and modification of the basic structure of state chart diagrams. In addition, *StateCharts* provides an execution function for the diagram; *Stateflow* offers the generation of embeddable C code from the models with the diagram; *visualState* supplies the creation of a visual, specification to graphically simulate the state chart diagram.

PetriSim and *HPSim* are designed to support Petri Nets. Both of them support the editing and simulation of Petri Net diagrams. *PetriSim* is a DOS based system and the simulation function only recognizes Pascal code. The *HPSim* simulator can analyze Java code and create executable workflows.

2.3 Early Prototyping Tools and IDEs

Prototyping tools help the designer to build and implement the system interface. Interface prototyping tools and system Integrated Development Environments (IDE) have become more and more important in the software development process (Myers Brad A., 1995). For the early stage of interface sketching, *Microsoft Paint* or *PowerPoint* can replace paper and pencil to present the designer's ideas in a clearer and more lively fashion. Sometimes we also describe early prototyping software as "Façade tools".

Most of the popular programming languages have their own comprehensive integrated development environments (Calvert Charlie, 1999). For example, *Delphi IDE, J Builder, Visual Basic IDE, Microsoft Visual C++ IDE* (Ford and Topp, 1996). Most of these tools help the software designers reduce the

amount of code that they need to produce when creating a user interface, and by using these tools, the user interfaces can be created more quickly.

Another important advantage to using the IDE, is it help achieve a consistent look and feel, when different developers use the same IDE to design their different products (Myers, 1995). For most designers, the consistency of interfaces makes it possible for them to understand and transfer interface products between different applications and platforms easily. When they know one graphical user interface, it will be possible for them to simply move to others (Myers, Scott. and Randy, 1999). On the other hand, when the end user is presented with software developed by the IDEs, it also will be easier for them to master the applications, because most of the components in the different IDEs are very similar.

In this section, we will briefly review four interface development tools and analyze their characteristics and differences. *PowerPoint* is an early prototyping (Facade) tool. It is easy for the user to learn, includes some basic graphical functions, and is suited to presenting the designer's initial ideas. But most of the interface components have to be drawn by the user, and it can't generate any source code for the program.

The complicated IDEs like *Delphi environment, Visual C++ 6.0 IDE* make it easier to build the interfaces. Most of the components can be found in the IDE library, and they also support the generation of code to reduce the designer's work (Bennett, 1997). By using an IDE, the user can quickly create quality interfaces and code. IDEs also help the designer to maintain the consistency of system interfaces (Calvert, 1999). But compared with PowerPoint, leaning how to master this kind of IDE is more difficult and requires programming ability (Deitel and Deitel, 1994).

PowerPoint

PowerPoint is a basic prototyping tool to support the early design of the user interface. The picture in figure 2-12 shows a mockup of part of the SELCU interface in PowerPoint.

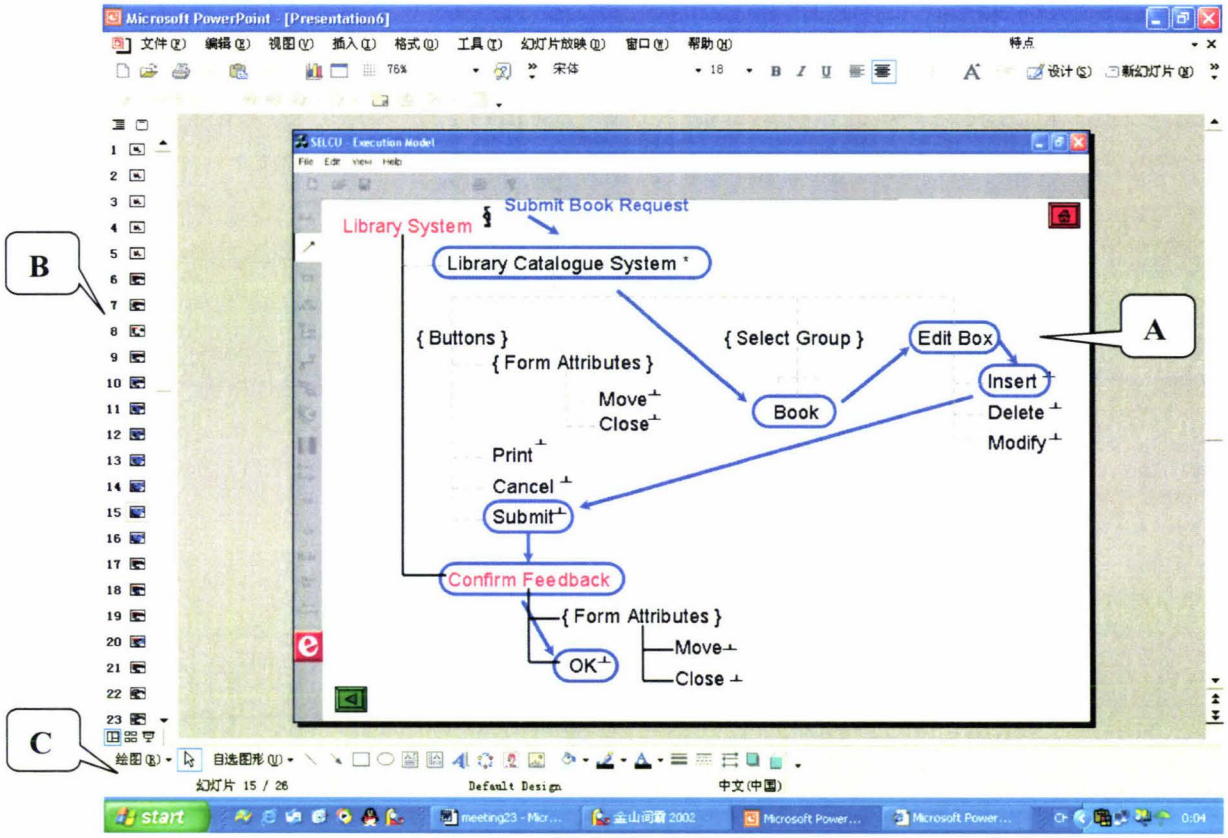


Figure 2-12 Early Interface Design in PowerPoint

Although we have lots of complicated IDEs to support graphical interface design, sometime we still feel the lack of primary prototyping tools to support very early sketching. Microsoft PowerPoint is a capable of generating on-screen interface mocks ups.

In figure 2-12, area A is the main design window of PowerPoint. Users edit their slides in this area. The components that the user wants to draw can be selected from shortcut menus at the bottom (area C). Area B is an overview of the whole

design work. It presents all the slides as a list and gives users an easier way to navigate the design options. These options include design templates, color schemes, and animation schemes—all of them can be previewed at the time a selection is made.

By using PowerPoint, the developer can mock up their design ideas on the screen. All the colors, styles, interface components, animation effects are easy for the user to input to the computer and also easy to modify. This helps the user to create a high quality interface mockup in a short time (Myers A. Personal Web, 2002). But compared with IDEs, PowerPoint only provides a mock up of the look and feel, and cannot auto generate any code for future software development (Swan Tom, 1998). The developers have to design most of the interface components by themselves.

The Delphi IDE

The Delphi IDE includes an Object Pascal compiler, interface design toolkit and extensive facilities for debugging software (Jacobs, 1999). The interface design toolkit typically provides both a library of interactive components, and an architectural framework to manage the operation of interfaces made up of those components (Lischner, 2000). Compared with PowerPoint, this kind of IDE employs an established framework and a library of reusable components to support user interface construction (Pacheco and Teixeira, 1998). Most of the interface components are provided in the IDE library, but in the simple prototyping tools like PowerPoint or Paint, the users have to draw components by themselves.

Fig 2-13 presents a user interface that design by the Delphi IDE.

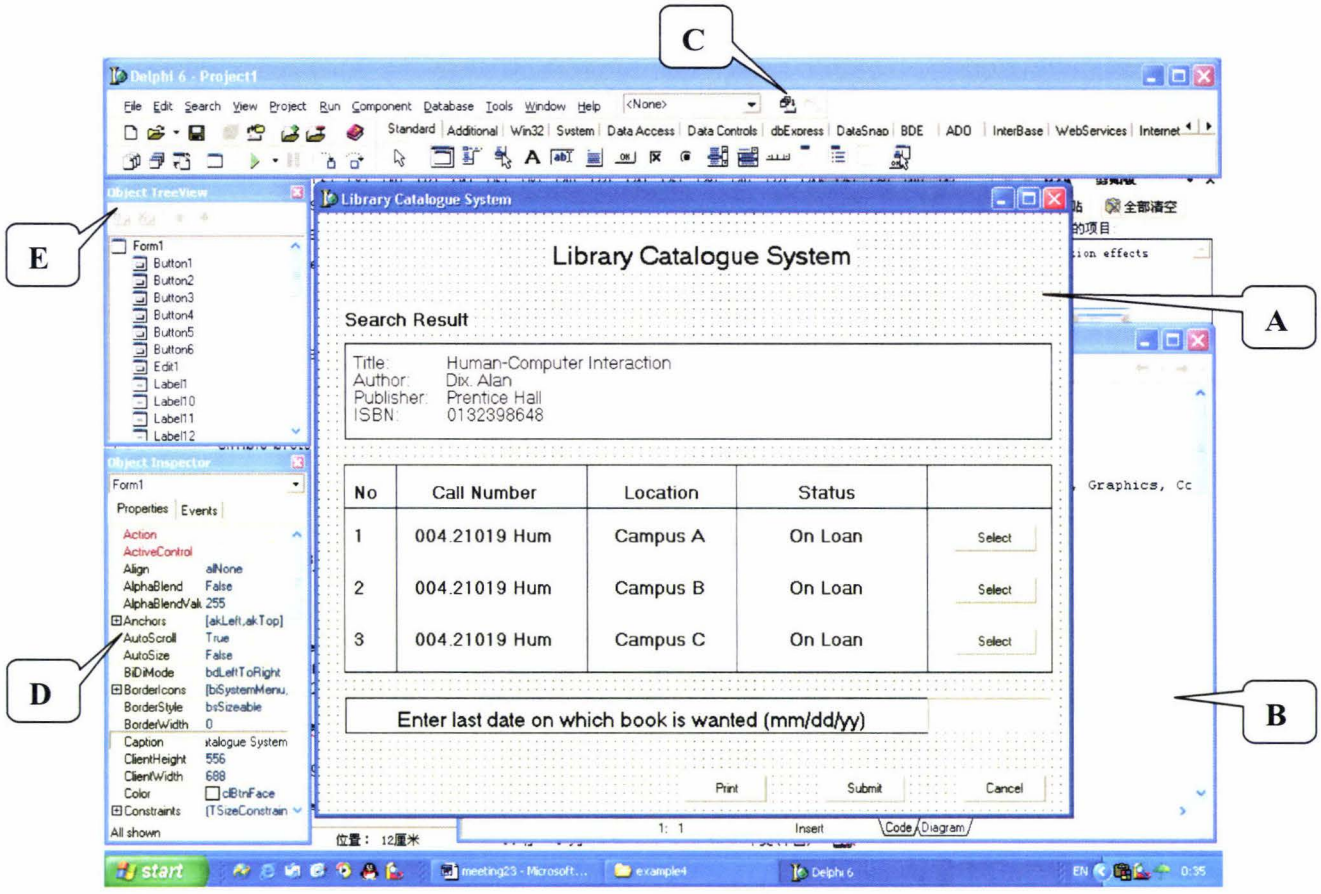


Fig 2-13: User Interface Design Using the Delphi IDE

The above Delphi IDE interface can be separated into five areas:

- **Area A : Form Designer**

  Used to design the basic window where the users specify the layout of the components of the interface.

- **Area B : Code Editer**

  Allows the user to access the Pascal code that provides the functionality of the interface components in form designer.

- **Area C: Menu/Toolbar**

  Includes the Delphi applications (such as Open, Save and so on), and all the Delphi user interface components.

- **Area D: Object Inspector**

  Has two tabbed sections, one showing the properties of a form and the other showing its events. All the components in the form will present their properties and events here.

- **Area E: Object Tree View Window**

  Presents all the components in a tree structure. It helps the user to have a big picture of the whole design work and understand the relationship between these objects.

These IDEs use a technology called RAD (Rapid Application Development), to assist programmers to develop code (Rachele, 2001). Generally, when a user finishes the design of the visual interface, the RAD system will automatically produces the skeleton of a corresponding program in the particular programming language (Roy, Smith and Lyons, 2000). In this skeleton, the program will include a list of names, following the correct syntax and some of the activity programs (Santos, 1999). The users just need to fill in the blanks with other code to specify how to perform each of the activities.

The IDE systems (like the Delphi IDE) are very popular in today's interface design because they map well to the direct manipulation style of graphical user interface. These systems generate the basic skeleton code for each user activity (Shannon, 1999). Most of the interface design components can be found from the library, and the user can simply use them and needn't design again. It saves the user's time and makes it easy to build a consistent look for all the applications.

Visual C++ Constructor

Visual C++ 6.0 IDE is another important interactive graphical tool to support prototyping. This kind of builder is similar to the Delphi IDE. (Dorfman, 1995) The constructors allow interactive components to be placed using a mouse to create windows and dialog boxes (Kruglinski, Wingo and Shepherd, 1998). In C++ constructor, the system also supports the code generation. Figure 2-14 shows the main screen of the Visual C++ 6.0 Constructor environment.

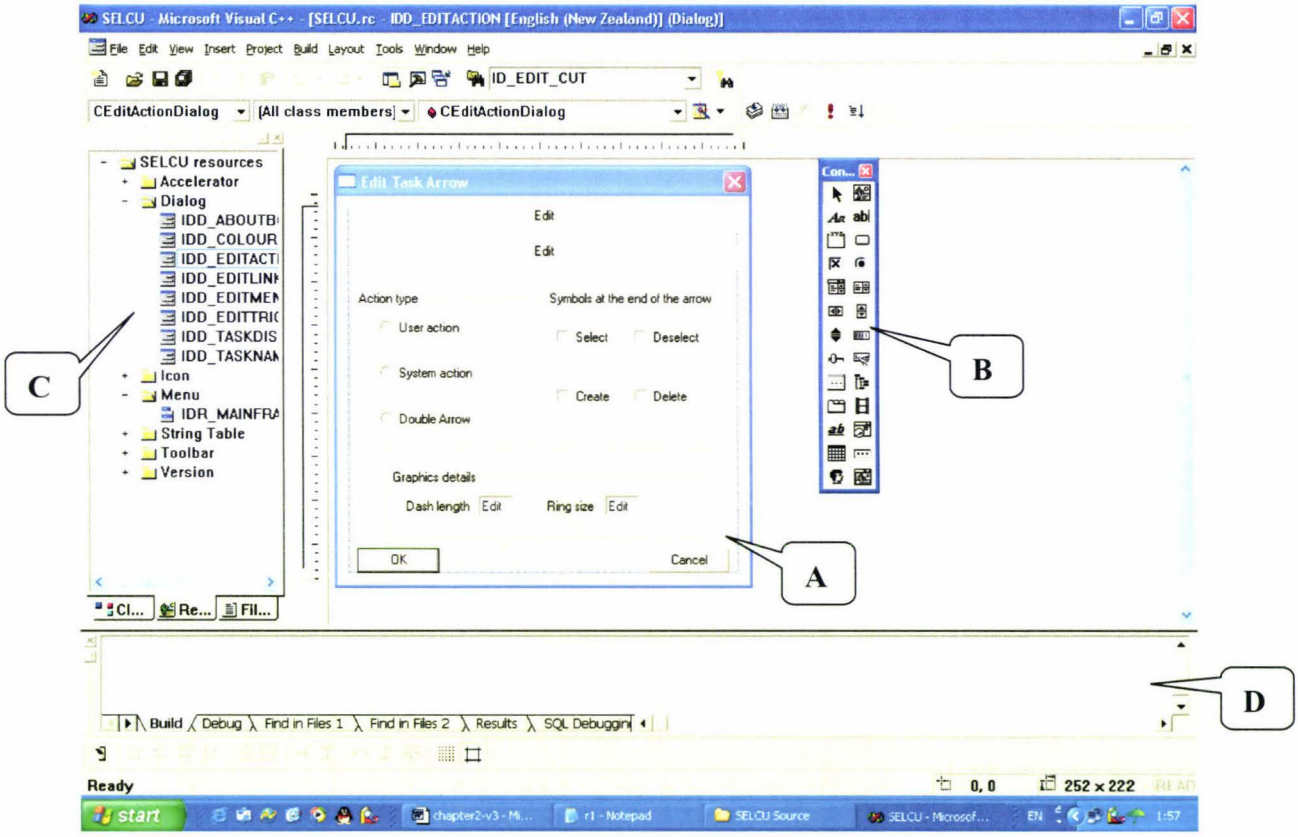


Figure 2-14: Visual C++ 6.0 Constructor

There are five main areas in figure 2-14,

- **Area A** is a dialogue box design area. It is like the form designer in Delphi IDE. Users can directly move interface components from the controls toolbar (area B) to the dialogue box.

- **Area B** is a controls toolbar. The interface components users need to design a dialogue box are all taken from here. To add a new component, just click the control's icon in here and then drag it to the dialogue box.

- **Area C** is a viewer window. It includes a class view, file view and resource view. By clicking the tab, a user can easily switch between three view modes. It presents all the files, classes or resources in tree structures, and the user can have an overview of the whole project.

- **Area D** is the output window area. When users build the project files, they will see messages in this area. It will list all the errors or problems found by the system. By double clicking an error messenger in this area, the user can easily find where the error has occurred and the line that contains the error.

In the constructor, by moving some aspects of user interface implementation from conventional code into an interactive specification system, these aspects of interface implementation are made available to those who are not conventional programmers (Horton Ivor, 1997). This advantage has allowed many visual design professionals to become more involved in creating the appearance of interfaces.

In summary, the facade tools provide special purpose scripting languages, and support the execution of early prototypes. Hence they provide the illusion of interaction with the application under development. This is valuable in acquiring knowledge of the system, in defining requirements, and in obtaining early feedback from users.

Interface builders on the other hand are capable of producing industrial strength applications, and are commonly based on general purpose (often object-

oriented) programming languages. They provide access to a toolkit of widgets, and increasingly support visual programming, which provides designers with immediate feedback on the look of the interface (Scogings and Phillips, 1998).

2.4 Summary

This chapter has reviewed dialogue notations, their support environments, and early prototyping tools and IDEs. The focus has been on graphical notations. A consistent example was used to review and compare **Petri Nets**, **STN**, **State Charts**, and **Lean Cuisine**. A simple introduction was also presented for **Lean Cuisine+**.

Support environments reviewed include ***Statecharts***, ***Stateflow***, ***VisualState*** (used to support State Chart); and ***PetriSim***, ***HPSim*** (used to support Petri Nets). The **SELCU** software environment developed to support Lean Cuisine+ is described in detail in the next chapter.

Early prototyping tools like **Microsoft PowerPoint** and advanced IDEs like the **Delphi environment** and **Microsoft Visual C++ 6.0 IDE** have also been reviewed.

Chapter 3

THE LEAN CUISINE+ NOTATION AND ITS SUPPORT ENVIRONMENT

This chapter is concerned with the Lean Cuisine+ notation and its support environment. In section 3.1, the main principles of the Lean Cuisine+ notation are introduced, and a Lean Cuisine+ description of the behavior of the Delphi example interface is produced. A Lean Cuisine+ support environment (SELCU) is described in section 3.2. Section 3.3 describes a proposed extension of SELCU. It includes automatic generation and execution of the Lean Cuisine+ notation.

3.1 The Lean Cuisine+ Notation

A review of graphical dialogue notations, including STNs, Petri Nets, State Charts and Lean Cuisine was carried out in chapter 2. Shortcomings were uncovered in all the above the notations in relation to describing direct manipulation GUIs (Phillips, 1994), in that none could describe satisfactorily both asynchronous and sequential aspects of the dialogue. In particular, these notations had difficulties either in the capturing of event sequences or in relating event sequences to higher level tasks.

Lean Cuisine+ (Phillips, 1995) is a semi-formal graphical notation for describing the behavior of event-based direct manipulation GUIs, and is an extension of

Lean Cuisine. The Lean Cuisine+ notation's strengths can be summarized as follows:

- It has been specifically developed as an interface design tool
- It overcomes the limitations of Lean Cuisine
- It fits easily into an object-oriented approach
- It brings together task and dialogue modeling
- A software tool has been developed to support Lean Cuisine+

3.1.1 Basic Definitions

- **Dialogue**

  A dialogue is described by a set of selectable primitives, called menemes, arranged in a tree structure, and a set of constraints over these primitives. The total dialogue state at any time is the sum of all the meneme states plus the values of any state variables.

- **Sub-Dialogue**

  Menemes may be grouped into sub-dialogues to any level of nesting. A sub-dialogue is a logical grouping of menemes within the body of a Lean Cuisine+ dialogue. The grouping of menemes within a sub-dialogue places constraints on their behavior. The grouping may be either mutually exclusive (1 from N) or mutually compatible (M from N).

- **Menemes**

  The meneme is the dialogue primitive, and is an individual selectable representation of an object, operation, state, or value. A Meneme may be real or virtual:

(1) Real menemes represent specific selectable options, and may be terminal or non-terminal. Non-terminal real menemes are headers to future sub-dialogues.

(2) Virtual menemes are always non-terminal and are used to partition a dialogue or sub-dialogue into further constituent syntactic meneme groupings. They are not available for selection. It is possible to depict the selection of a virtual meneme in a task action sequence (see section 3.1.5), but this indicates that one or more of the real menemes in the sub-dialogue may be selected at that point in the task.

- **Meneme States**

  A meneme is either available or not available for selection, and either selected or not selected. This gives rise to four possible meneme states. The state may be changed either by direct excitation, or by indirect modification, subject to any constraints applying.

- **Fork**

  Where a group of options constitute a homogeneous set, they may be represented graphically using a fork symbol, as shown in figure 3-1.

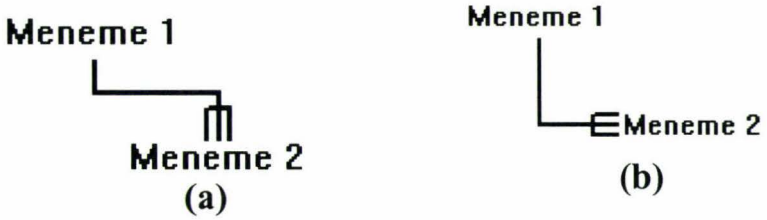


Figure 3-1 (a): Mutually Compatible Fork

(b): Mutually Exclusive Fork

- **Stepwise Refinement**

  To support stepwise refinement, a dialogue or sub-dialogue tree may be represented visually by an ellipsis to indicate that the detail is developed in a separate diagram. In the case of a sub-dialogue, the lower level diagram is headed by an ellipsis in order to visually complete the association. The example for stepwise refinement is shown in figure 3-2.

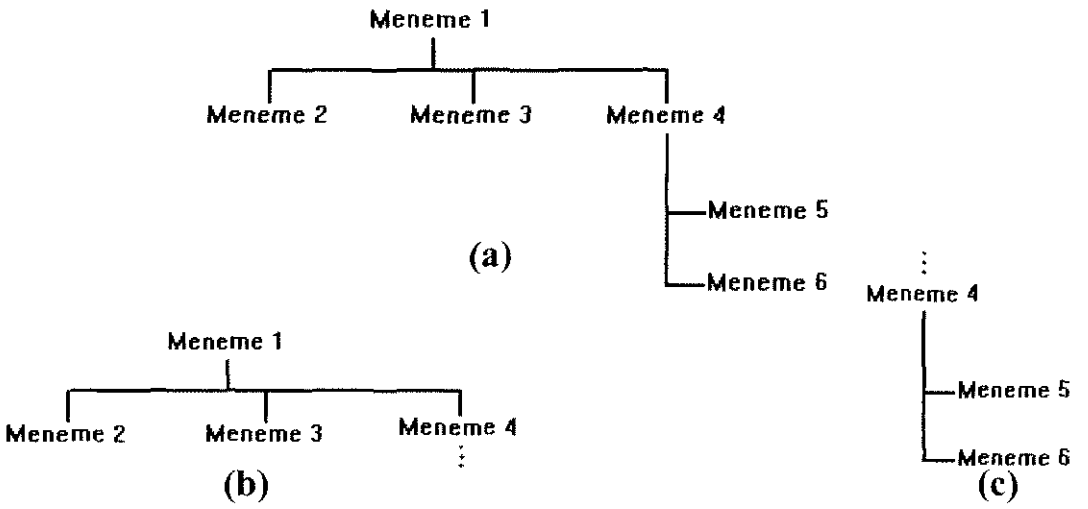


Figure 3-2 (a): Original Lean Cuisine+ Diagram

(b): The Lean Cuisine+ Diagram after "Hide Children"

(c): The Lean Cuisine+ Diagram after "Hide Parents"

- **Meneme Designators**

  Refer to Table 2-2, "meneme modifiers".

3.1.2 Column Attributes Selector Case Study

A case study is introduced at this point. It will be used through the remainder of section 3.1 to illustrate features of the Lean Cuisine+ notation. Column attributes selector is a sub-application of Microsoft Word 2000. It allows the user to design the column numbers, set the width and spacing, setup the active area and

preview the selection result. The interface of column attributes selector is presented in figure 3-3.

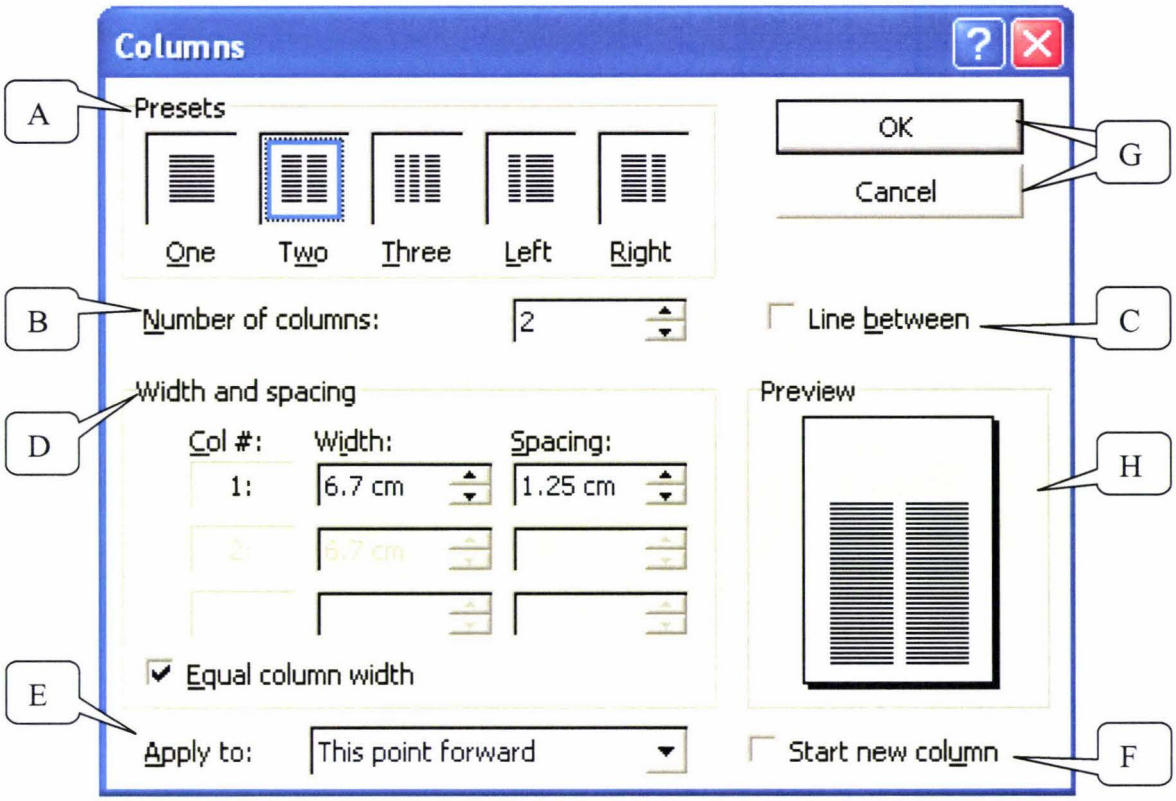


Figure 3-3 Column Attributes Selector

Eight main interface element groups are shown in figure 3-3:

- **A** is used to select and present the basic column options. It allows the user to set the column number which is less than four. If the column numbers (*One*, *Two* or *Three*) is set in **A**, the *Number of columns* is also automatically set to the corresponding number in **B**.

  Left and *Right* attributes are only used for *Two* columns option. *Left* means the left column is shorter than right one. *Right* means the right column is shorter than left one.

- **B** is used to set the column number. The user can select the number of columns from one to forty five. If the column number is less than four, the corresponding option is automatically set in **A**.

- **C** is an option which lets the user put lines between columns. Only if the column number is more than two, does this selection become available.

- **D** is used to set the *width* and *spacing* for the columns. The default state is *Equal column width*.

- **E** is used to setup the column attribute selection's active area. The default setting is *Whole document*.

- **F** allows the user to *Start a new column*, it is only available when the active area is set to *This point forward* in **E**.

- **G** is a button group. *Ok* is used to confirm and save all the selections, then it closes the application. *Cancel* means ignore all the previous settings and exit the column attributes selector program.

- **H** presents a preview window. All the selections results are shown in here.

Figure 3-4 is a Lean Cuisine+ diagram used to describe this example.

- Brackets { } are used to indicate virtual menemes (for example *{Buttons}*, *{Present}*, *{Width and Spacing}* and *{Apply to}*).

- When the user clicks buttons such as *Ok* and *Cancel*, they will not stay in the selected state (and will go back to an unselected state automatically). They are represented by Monostable menemes ($\perp$).

- Since the user doesn't know the *number of columns* the mutually exclusive fork is used.

- The asterisk (*) is used to represent default menemes (for example *One*, *Equal column width* and *Whole document*).

- The meneme ***column*** is used to represent the ***Col#*** in **D** of figure 3-3. Because it is only selected by the system, designator ***passive*** (⊗) is used.

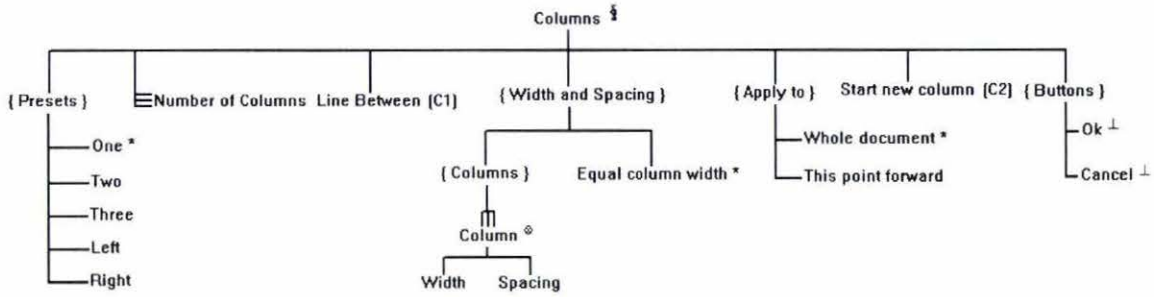


Figure 3-4 Lean Cuisine+ Tree Diagram for Column Attributes Example

The main Lean Cuisine+ extensions are described below.

3.1.3 Preconditions

The availability of an option for selection can be governed by a precondition attaching to the meneme which represents it. This provides for description of the unavailability of selectable options at appropriate points in a dialogue.

In figure 3-3, the *Line between* becomes available only the *Number of columns* is more than one. Additionally, the *Start new column* option is available only if the *This point forward* is selected. The representation of this in Lean Cuisine+ is shown in figure 3-5, the precondition being enclosed in square brackets ([ ]) following the meneme name.

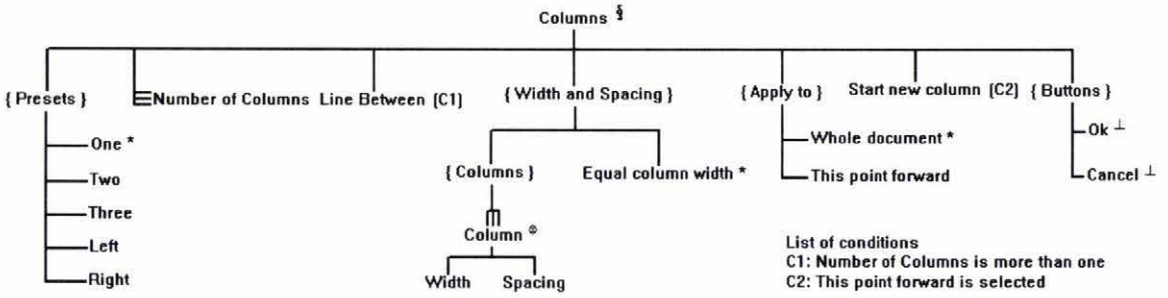


Figure 3-5: Lean Cuisine+ Diagram with Conditions

3.1.4 Selection Triggers

Selection Triggers are used to connect menemes in order to describe system responses to user events involving selection. They capture the fact that the *selection* or *de-selection* of an object or option may trigger the *selection* or *de-selection* of other object or options. Triggers may also create or delete object instances (Scogings and Phillips, 2000).

Selection triggers are annotated as shown in Table 3-1.

Trigger	Action
a → b (△)	Select
a → b (▽)	Deselect
a → b (⊕)	Create
a → b (⊖)	Delete
a → b (|)	Constrained
a → b	Unconstrained
a → b Condition	Condition

Table 3-1 Trigger Actions

Select and deselect actions mean this trigger will cause a selection or de-selection for meneme **b**. Create or delete actions mean that trigger will create or delete meneme **b**. Triggers are either unconstrained, in which case further knock-on effects are possible, or constrained, in which case the behavior of the selected or deselected meneme is modified, inhibiting further links, and preventing side effects such as cycles.

In figure 3-6, the red lines show triggers for **Column Attributes** example.

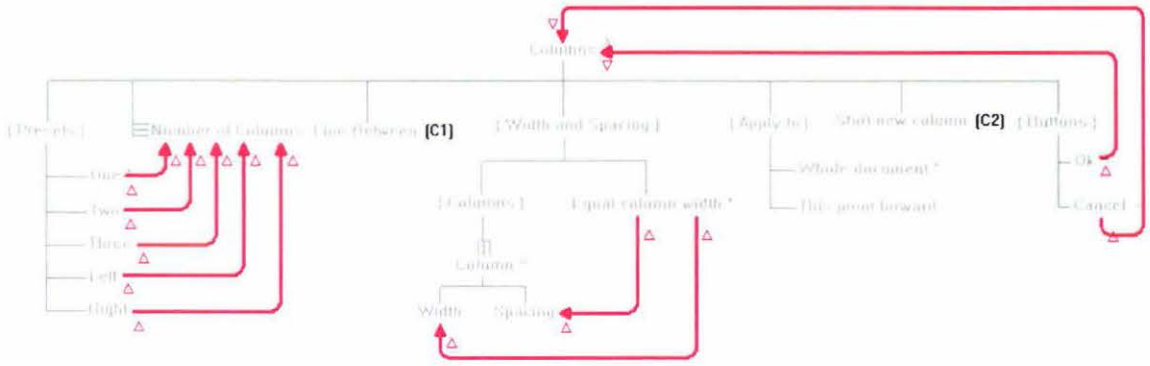


Figure 3-6 Lean Cuisine+ Diagram Following Triggers

In above diagram:

- If the user selects *One*, *Two*, *Three*, *Left* or *Right* meneme, the *Number of Columns* automatically changes corresponding value.
- If the *Equal column width meneme* is selected, all the *Width Value* and *Spacing Value* automatically changes to same.
- If the user clicks *Ok* or *Cancel* button, *Column* window automatically closes.

3.1.5 Task Action Sequences

A major advantage of the notation is that task action sequences can be represents in the context of the interface dialogue. A task action sequence describes a high level user task in terms of primitive actions, capturing temporal

relationships between them. Each task or group of related tasks is represented as an overlay on the base diagram. Each task overlay consists of a series of linked nodes superimposed on the tree diagram.

Figure 3-7 shows the *Select Two Columns Task* sequence overland on the basic tree diagram for the column example. There are six steps in this task:

(1) The user sets column number as two (selects *Two* meneme).

(2) The user puts a line between columns (selects *Line Between* meneme).

(3) The user equals all the columns width and spacing in same value (selects *Equal column width* meneme).

(4) The user actives the setting in forward area (selects *This point forward* meneme).

(5) The user saves the setting (selects *Ok* meneme).

(6) The system closes the columns dialogue (deselects *Columns* meneme). Note that system actions are shown dotted.

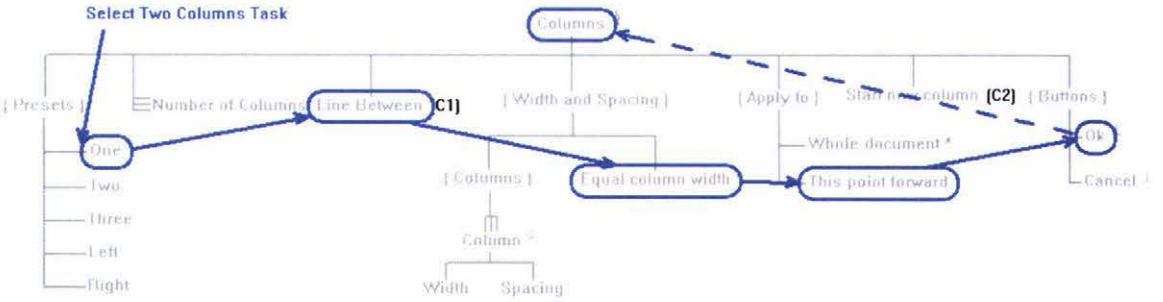


Figure 3-7 Task Overlay for Column Example

The advantages of the Lean Cuisine+ task overlay are (Scogings and Phillips, 1999):

1. The task is represented at the correct level required for interface design
2. The task is shown within the context of the interface
3. System responses are represented as well as the user actions

4. It can be used as an interface analysis tool

In summary, Lean Cuisine+ is a concise graphical notation which clearly shows dialogue structure (as a hierarchy of sub-dialogues). It allows designers to represent the structure of an interface dialogue. It is able to combine static and dynamic interface modeling in a coherent manner. The dialogue structure captures constraints and inter-relationships between dialogue components. Tasks in Lean Cuisine+ are presented in the context of this dialogue.

3.2 Support Environment for Lean Cuisine+

(Phillips, 1995) defines a support environment for Lean Cuisine+ with the following functionality:

- Support for the construction, browsing and display of Lean Cuisine+ specifications, through the provision of diagram editors supported by a system dictionary.

- Support for the execution of Lean Cuisine+ specifications, providing a simulation of meneme behaviour as "selections" are made.

A system called "**S**oftware **E**nvironment support for the **L**ean **C**Uisine+ notation and **U**ML" (SELCU) has been developed (Scogings, 2003). SELCU is a tool that is used to help the designer to create, edit, save or print the Lean Cuisine+ diagram in a systemic software environment.

The SELCU system supports direct manipulation for most of the Lean Cuisine+ diagram objects. A meneme can be moved by selecting it with the mouse and dragging it to another place. Most of the other diagram objects can be moved in

a similar way. Most of the objects can be modified by double clicking, or be selected by single clicking.

The whole SELCU system has been developed in Microsoft Visual C++ 6.0 using the Microsoft Foundation Classes environment. It works on a Windows 2000 based operation system or similar (like Windows 98/ Me/ NT/ XP).

3.2.1 SELCU Interface Introduction

Figure 3-8 illustrates the SELCU interface of standard windows toolbar options.

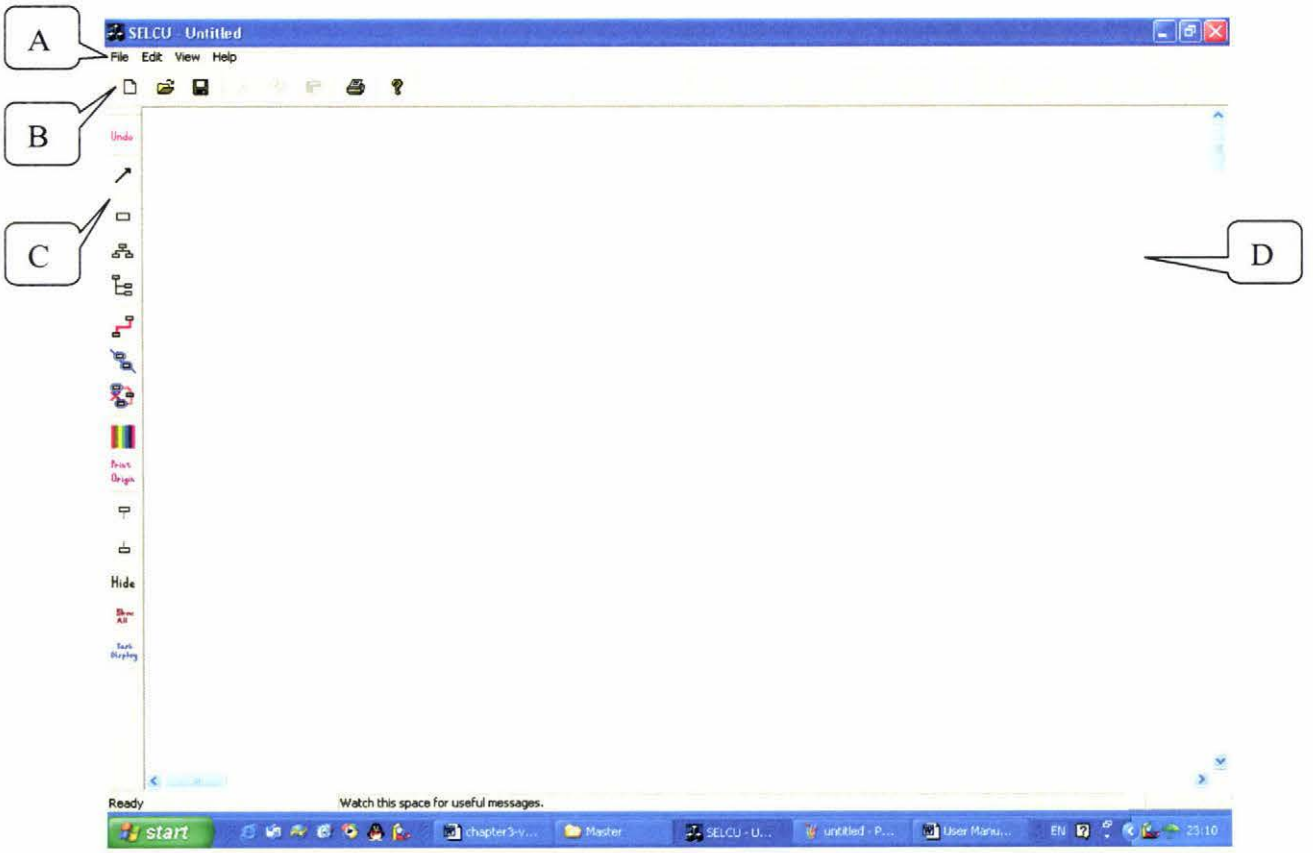


Figure 3-8 SELCU Interface

There are four main areas in above screen layout:

- **A** is a set of system pull down menus. It provides standard *File*, *Edit* and *Help* menus, plus a view menu discussed below.

- **B** is a system tool bar. It presents several Windows standard function shortcuts (which are *New, Open, Save, Print* and *Help*).
- **C** is the SELCU tool bar. Most of SELCU main functions all have a shortcut button in here.
- **D** is main editor area. The Lean Cuisine+ diagram is constructed in here.

View Sub-Menu (A of figure 3-8)

The *View* sub-menu provides *Menemes only* (display only the basic dialogue tree of linked menemes), *Unhide menemes* (display all menemes), *Conditions* (display all meneme conditions and list of them), *Unhide triggers* (display all the triggers), *Tasks* (present a list of tasks from which can be selected for display), *All tasks* (display all task sequences), *Toolbar* (toggles display of the standard Windows toolbar) and *Status bar* (toggles display of the standard Windows status bar).

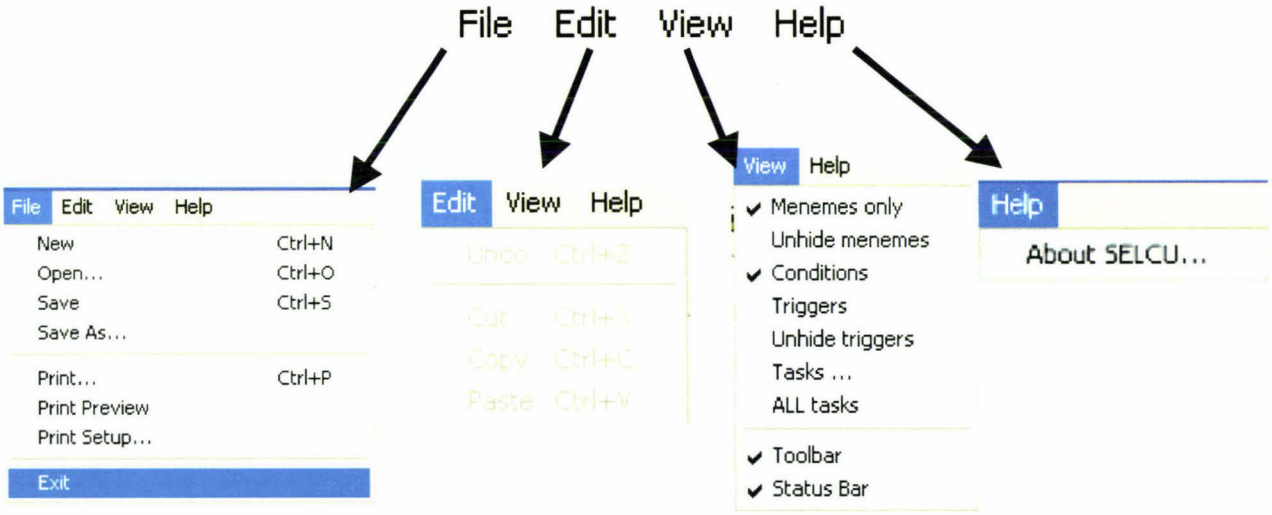
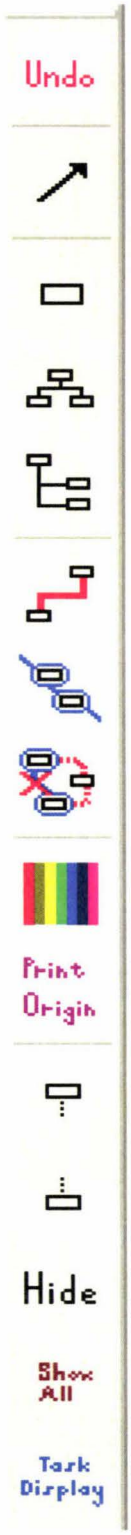


Figure 3-9: System Pull Down Menus

SELCU Tool Bar (C of figure 3-8)

The SELCU tool bar provides a series of editing functions. Figure 3-10 presents the meaning of SELCU tool bar buttons and their effect.



→ *Undo* (Undo previous deletion)

→ *Select object* (Cancel the current mode and select a meneme)

→ *Insert Meneme* (Add a new meneme)

→ *Insert Compatible* (Add a mutually compatible meneme group)

→ *Insert Exclusive* (Add a mutually exclusive meneme group)

→ *Construct Trigger* (Add, delete or modify a trigger)

→ *Construct Task* (Add or delete a task)

→ *Modify Task* (Change order of sequences, add or remove meneme)

→ *Change Colours* (Change SELCU system colours and font)

→ *Print Origin* (Set print origin)

→ *Hide Children* (Replace all the children menemes by an ellipsis)

→ *Hide Parent* (Replace all the parent menemes by an ellipsis)

→ *Hide* (Hide selected object)

→ *Show All* (Display all the menemes, tasks, triggers and conditions)

→ *Task Display* (Select tasks to display from the list)

Figure 3-10: SELCU Tool Bar

3.2.2 SELCU Source File Format

The source file for a Lean Cuisine+ diagram is stored in text format:

- the text file can be directly edited which means that, if necessary, a Lean Cuisine+ diagram can be modified without using the SELCU package.

- graphical data following this format can be generated by another application and subsequently viewed as a Lean Cuisine+ diagram

Each data file must have the extension ".lc" and an example is presented below.

An example data file

Using the "Column Attributes Selector" (which is introduced in section 3.1.2) as an example. The Lean Cuisine+ diagram with conditions, triggers and tasks is shown in figure 3-11.

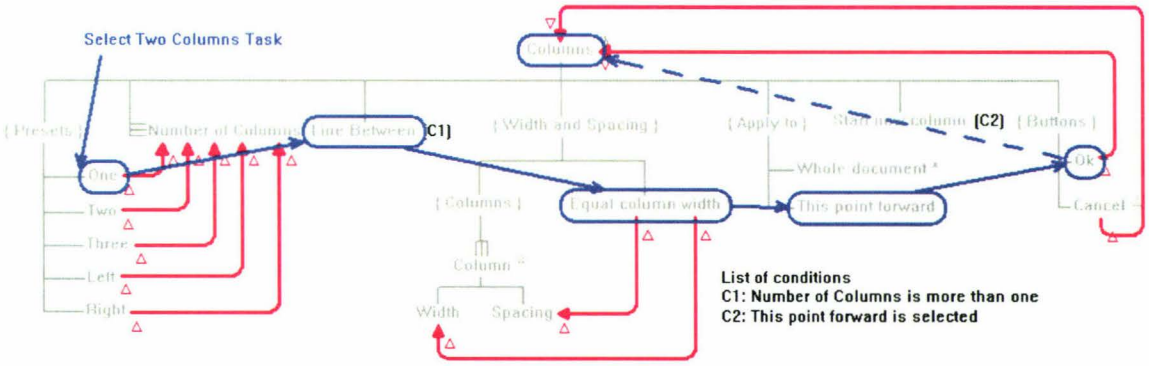


Figure 3-11: Lean Cuisine+ Diagram with Conditions, Triggers and Tasks

The text source file of above Lean Cuisine+ diagram is listed below:

```
System, 16, 7, 0, 0, 700, 0, 0, 0, 134, 1, 2, 2, 34, 10,
1, [553:-77,631:-99]Vs(0)<r>Nf; ; ;CL[0,0:0,0][39,-22:39,-42][-495,-42:550,-
42] ;Columns
2,{}[12:-162,104:-184]Vs(1)<>Nf; ; ;EL[46,43:46,0][45,-22:45,-198][0,0:0,0] ;Presets
4, [329:-162,445:-184]Vs(1)<>Nf; ; ;nL[58,43:58,0][0,0:0,0][0,0:0,0]*;Line Between
2:C1;Number of Columns is more than one
5,{}[518:-156,698:-178]Vs(1)<>Nf; ; ;CL[74,37:74,0][75,-22:75,-47][-11,-47:160,-
47] ;Width and Spacing
6,{}[759:-154,853:-176]Vs(1)<>Nf; ; ;EL[47,35:47,0][47,-22:47,-98][0,0:0,0] ;Apply to
7, [870:-151,1015:-173]Vs(1)<>Nf; ; ;nL[72,32:72,0][0,0:0,0][0,0:0,0]*;Start new
column
2:C2;This point forward is selected
8,{}[1057:-152,1149:-174]Vs(1)<>Nf; ; ;EL[46,33:46,0][46,-22:46,-
99][0,0:0,0] ;Buttons
9, [99:-209,139:-231]Vs(2)<e>Nf; ; ;nL[-42,-11:0,-11][0,0:0,0][0,0:0,0] ;One
10, [98:-246,138:-268]Vs(2)<>Nf; ; ;nL[-41,-11:0,-11][0,0:0,0][0,0:0,0] ;Two
11, [97:-280,151:-302]Vs(2)<>Nf; ; ;nL[-40,-11:0,-11][0,0:0,0][0,0:0,0] ;Three
12, [98:-315,136:-337]Vs(2)<>Nf; ; ;nL[-41,-11:0,-11][0,0:0,0][0,0:0,0] ;Left
13, [97:-349,147:-371]Vs(2)<>Nf; ; ;nL[-40,-11:0,-11][0,0:0,0][0,0:0,0] ;Right
16,{}[457:-237,557:-259]Vs(5)<>Nf; ; ;EL[50,34:50,0][51,-22:51,-
50][0,0:0,0] ;Columns
18, [597:-239,760:-261]Vs(5)<e>Nf; ; ;nL[81,36:81,0][0,0:0,0][0,0:0,0] ;Equal column
width
19, [832:-202,974:-224]Vs(6)<e>Nf; ; ;nL[-26,-11:0,-11][0,0:0,0][0,0:0,0] ;Whole
document
20, [832:-241,983:-263]Vs(6)<>Nf; ; ;nL[-26,-11:0,-11][0,0:0,0][0,0:0,0] ;This point
forward
21, [1115:-193,1144:-215]Vs(8)<m>Nf; ; ;nL[-12,-11:0,-11][0,0:0,0][0,0:0,0] ;Ok
```

22, [1114:-240,1176:-262]Vs(8)<m>Nf; ; ;nL[-11,-11:0,-11][0,0:0,0][0,0:0,0] ;Cancel

3, [162:-162,328:-184]Vs(1)<>Ef; ; ;nL[-15,43:-15,-9][56,-22:56,-61][0,0:0,0] ;Number of Columns

23, [457:-302,545:-324]Vs(16)<p>Cf; ; ;CL[51,15:52,15][51,-22:51,-33][7,-33:95,-33] ; Column

25, [437:-352,492:-374]Vs(23)<>Nf; ; ;nL[27,17:27,0][0,0:0,0][0,0:0,0] ;Width

26, [516:-352,588:-374]Vs(23)<>Nf; ; ;nL[36,17:36,0][0,0:0,0][0,0:0,0] ;Spacing

A. Triggers ###

8[9,R,13<s>][3,B,17<s>] (166,-218:166,-218)0:;

(80,-9)

9[10,R,14<s>][3,B,43<s>] (201,-255:201,-255)0:;

(107,-8)

10[11,R,12<s>][3,B,70<s>] (244,-290:244,-290)0:;

(135,-10)

11[12,R,15<s>][3,B,99<s>] (276,-322:276,-322)0:;

(163,-7)

12[13,R,10<s>][3,B,137<s>] (295,-361:295,-361)0:;

(202,-12)

15[21,R,17<s>][1,R,11<d>] (1154,-198:1154,-198)0:;

(45,-5)(45,105)

17[22,B,32<s>][1,T,38<d>] (1142,-279:1142,-279)0:;

(32,-39)(75,-39)(75,198)(-523,198)

20[18,B,71<s>][26,R,12<s>] (668,-361:668,-361)0:;

(71,-123)

21[18,B,131<s>][25,B,27<s>] (728,-397:728,-397)0:;

(131,-170)(-133,-170)

B. Tasks ###

1[-68:96,-94:306]B20;Select Two Columns Task
[9(L,7)(R,13)U;6;20;][4(L,27)(B,105)U;6;20;][18(T,45)(R,17)U;6;20;][20(L,16)(T,127)U;6;20;]+
[21(L,19)(L,13)U;6;20;][1(R,25)(B,10)S;6;20;]#

Explanation of the text file format

- **The first line of the file**

System, 16, 7, 0, 0,700, 0, 0, 0, 134, 1, 2, 2, 34, 10,

The above text is the first line of the example source file. It contains the font information for the file as: Font name (**System**), Height (**16**), Width (**7**), Escapement (**0**), Orientation (**0**), Weight (**700**), Italic (**0**), Underline (**0**), Strikeout (**0**), Char set (**134**), Out precision (**1**), Clip precision (**2**), Quality (**2**), Pitch & family (**34**) and Font size (**10**),

- **The meneme**

4, [329:-162,445:-184]Vs(1)<>Nf; ; ;nL[58,43:58,0][0,0:0,0][0,0:0,0]*;Line Between 2:C1;Number of Columns is more than one

The above text is used to describe a meneme. It includes two lines of content. The first line shows basic meneme information, and the second line provides the selection condition for the meneme (if the meneme doesn't include a condition, it only has first line description).

The information in the first line is shown in table 3-2:

Text	Meaning
4,	The number is the meneme ID
O	If a space in here means a real meneme, or { } in here means a virtual meneme
[329:-162,445:-184]	Four numbers [left:top,right:bottom], are the pixel coordinates of the meneme border

Text	Meaning
Vs	Vs means a visible meneme; Hd means Hidden or Dt means Dots
(1)	The number means header meneme ID (zero if there is no header)
< >	<designators> s = Select, d = Deselect, p = passive, e = Default, M = Monostable, r = Required, l = Modal link, t = Unassigned default for subgroup, space = no attributes
Nf;	Nf means No Fork, Cf means compatible fork and Ef means exclusive fork
; ;	; ; means no ellipsis present, ;U; means parent ellipsis present and ;D; means children ellipsis present
nL	nL means no Link, CL means Compatible Link or EL means Exclusive Link
[58,43:58,0]	Four numbers [x1,y1:x2,y2], are the pixel coordinates of 2 points forming the line "into" the meneme
[0,0:0,0]	Four numbers [x3,y3:x4,y4], are the pixel coordinates of 2 points forming the line "out of" the meneme
[0,0:0,0]	Four numbers [x5,y5:x6,y6], are 2 points forming the "crossbar" if the meneme has a compatible subdialogue *Note: (x1,y1) to (x6,y6) are given as coordinates* ***relative*** *to the top-left corner of the meneme.*
***;**	*; means a meneme with a condition (has second line), or ; means a meneme without a condition (doesn't have second line)
Line Between	The meneme name

Table 3-2: The Meaning of Basic Meneme Description

Only if a meneme includes a condition, does it have a second line description. The information in the second line is shown in table 3-3:

Text	Meaning
2:	The number of characters in the abbreviation. If a condition doesn't have an abbreviation, the number is zero
C1;	The abbreviation. If a condition doesn't have an abbreviation, it appears as a space
Number of Columns is more than one	The condition

Table 3-3: The Meaning of the Meneme Condition Description

- **The trigger**

If there are triggers in the file, the following header line appears after the menemes:

A. Triggers ###

After this header, each trigger appears on two lines. The following text is a trigger description from the example:

❶ ❷

8[9,R,13<s>][3,B,17<s>] (166,-218:166,-218)0:';
(80,-9)

The information in the trigger text description is shown in table 3-4:

Text	Meaning
8	The trigger ID number
[9,	[First meneme ID in trigger
R,	First edge, it includes R (right), T (top), L (Left) or B (button)
13	Distance in pixels from top-left corner of first meneme to the point where the trigger starts

Text	Meaning
<s>]	<trigger designators for the first meneme>], it includes s (select), d (deselect), c (create) or d (delete)
[3,	[Id number of the last meneme
B,	Last edge, it includes T (top), L (left), R (right) or B (bottom)
17	Distance in pixels from top-left corner of last meneme to the point where the trigger ends
<s>]	<trigger designators for the first meneme>], it includes s (select), d (deselect), c (create) or d (delete)
(166,-218:166,-218)	(left,top:right,bottom), pixel coordinates of the position of the condition
0:	The number of characters in the abbreviation. If the trigger doesn't have a condition, the number is zero :
❶;	The abbreviation. If the trigger doesn't have a condition, it is a space;
❷	The condition content. If the trigger doesn't have a condition, it is a space
(80,-9)	The turning point pixel for the trigger. If it is a straight trigger, it appears *no points*.

Table 3-4: The Trigger Description

- **The task sequence format**

If there are tasks in the file, the following header line appears:

B. Tasks ###

After this, each task sequence appears on two or more lines. The following text is a text description from the example:

1[-68:96,-94:306]B20;Select Two Columns Task
[9(L,7)(R,13)U;6;20;][4(L,27)(B,105)U;6;20;][18(T,45)(R,17)U;6;20;][20(L,16)(T,127)U;6;20;]+
[21(L,19)(L,13)U;6;20;][1(R,25)(B,10)S;6;20;]#

The information in the first line of the above text is presented in table 3-5:

Text	Meaning
1	The task sequence ID number
[-68:96,-94:306]	[left:top,right:bottom], the pixel coordinates of the task name
B	Arrow edge, includes B (bottom), T (top), L (left) or R (right).
20;	Distance in pixels from top-left corner of task name to the point where the arrow starts;
Select Two Columns Task	The task name

Table 3-5: The Meaning of the Basic Task Description

The information in the second line includes the sequence of the task. Every meneme in the task contains a group of text in a square bracket "[ ]". The following text is the last meneme description of the task:

[1(R,25)(B,10)S;6;20;]#

The meneme description in the task sequence is shown in table 3-6:

Text	Meaning
[1	[The meneme ID number
(R,25)	(edge1,d1) – edge and distance from top-left of where an arrow arrives at the meneme
(B,10)	(edge2,d2) – edge and distance from top-left of where an arrow leaves the meneme
S;	Arrow type; it can be U (user action), S (system action) or D (double-headed);
6;	ring-offset; distance in pixels from the meneme border to the "ring" which is drawn around the meneme

20;]	dash-length; is the length in pixels of dashes used to draw a system action
#	Task sequence finish symbol

Table 3-6: Single Meneme Description in Task Sequence

3.2.3 Main Functions of SELCU

SELCU has been designed to be intuitive and any part of the Lean Cuisine+ diagram can be directly selected. In this system, single menemes can be placed at any point in the main work area, and new diagrams can be rapidly constructed and expanded by the addition of sub-dialogues where a mouse click places each meneme in the sub-dialogue and links are automatically constructed back to the header meneme.

The user can add the first meneme *(Meneme 1)* of a Lean Cuisine+ diagram by selecting the *Construct a new meneme* shortcut from the SELCU toolbar, and then clicking anywhere in the work area. The screen layout is presented in figure 3-12:

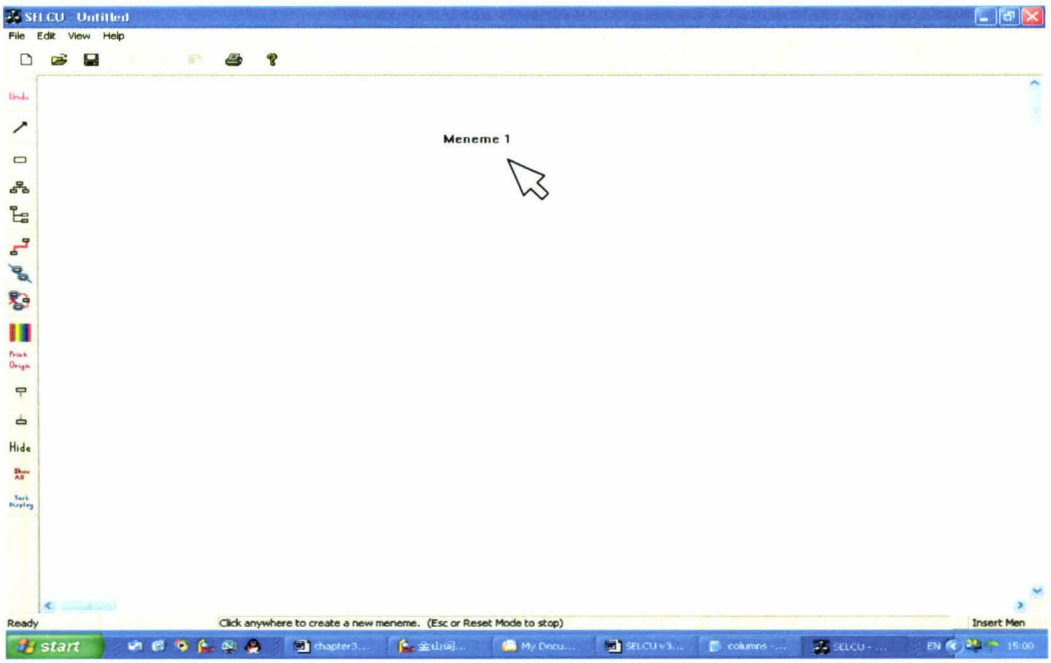


Figure 3-12: Add First Meneme on Diagram

If the user wants to add mutually compatible child menemes, they select *Construct a mutually compatible subdialogue* shortcut from the SELCU toolbar first. Then select the existing meneme (*meneme 1*), and click at points in the work area to add child menemes (*Meneme 2, 3, 4, 5*). The interface is shown in figure 3-13.

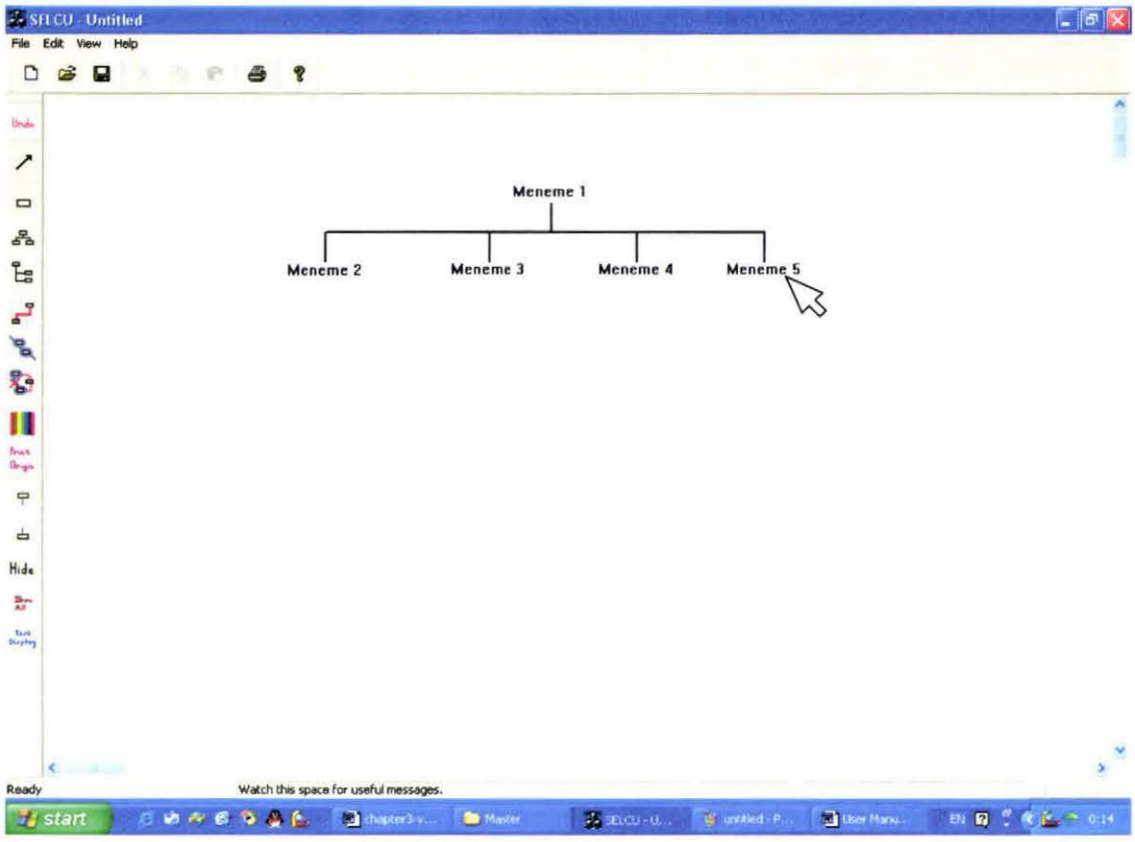


Figure 3-13: Add Mutually Compatible Child Menemes

If the user wants to add mutually exclusive menemes on the diagram, they need to select *Construct a mutually exclusive subdialogue* shortcut from the SELCU toolbar first. Then, select the parent meneme (*Meneme 4*) for the new subdialogue. Then, click anywhere in the work area. Each click will create a new meneme (*Meneme 6, 7, 8*) with a link back to the parent. The screen looks like figure 3-14.

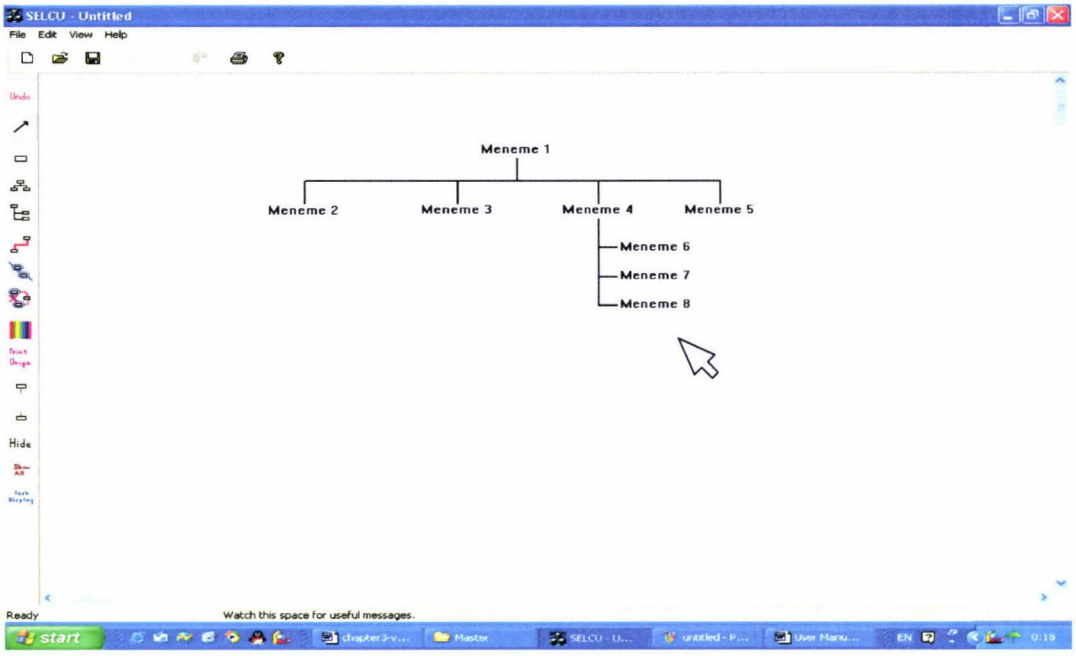


Figure 3-14: Add Mutually Exclusive Meneme Group

To add a trigger on the diagram, the user selects the *Construct a new trigger* shortcut from the SELCU toolbar first. Then, select any existing meneme (e.g. *Meneme 2*) as the source meneme for the trigger. After that, the user selects another meneme (*Meneme 8*) as the target meneme for the trigger. The screen layout of the trigger is shown in figure 3-15:

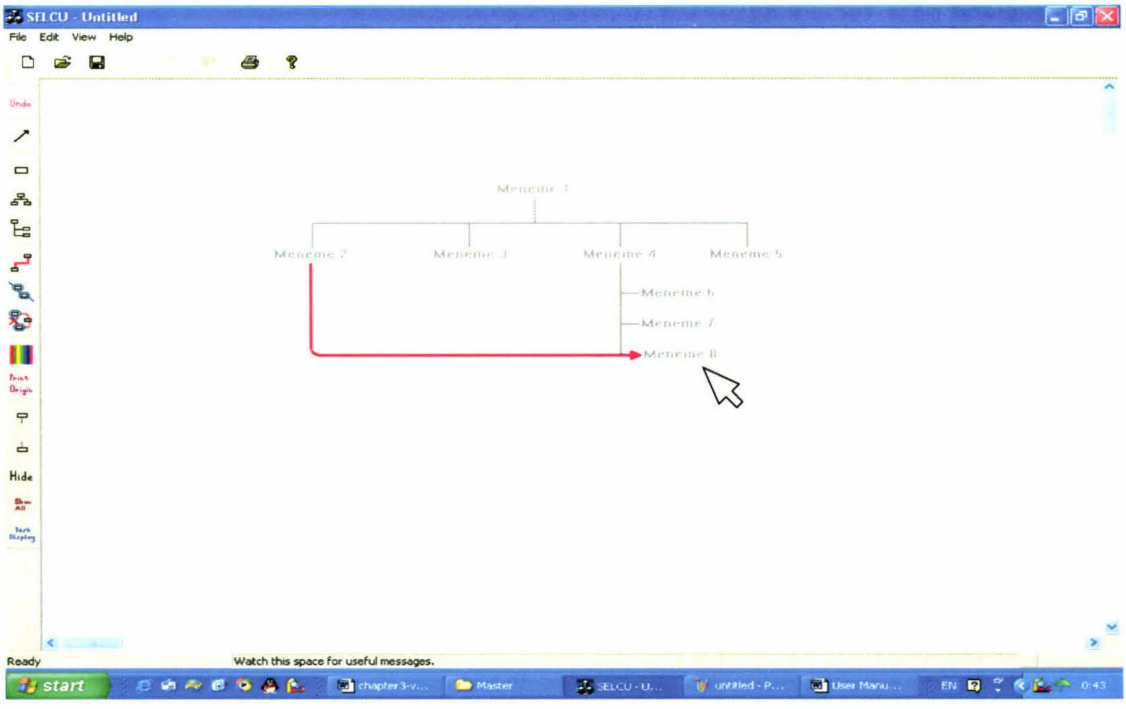


Figure 3-15: Add a New Trigger

To add a new task on the diagram, the user selects the *Construct a new task* shortcut from the SELCU toolbar. Then, the user selects several existing menemes (*Meneme 1, 3, 7*) to set the task sequence. A click on an empty area will create a new meneme in that area as part of the task sequence. The final task is presented in figure 3-16:

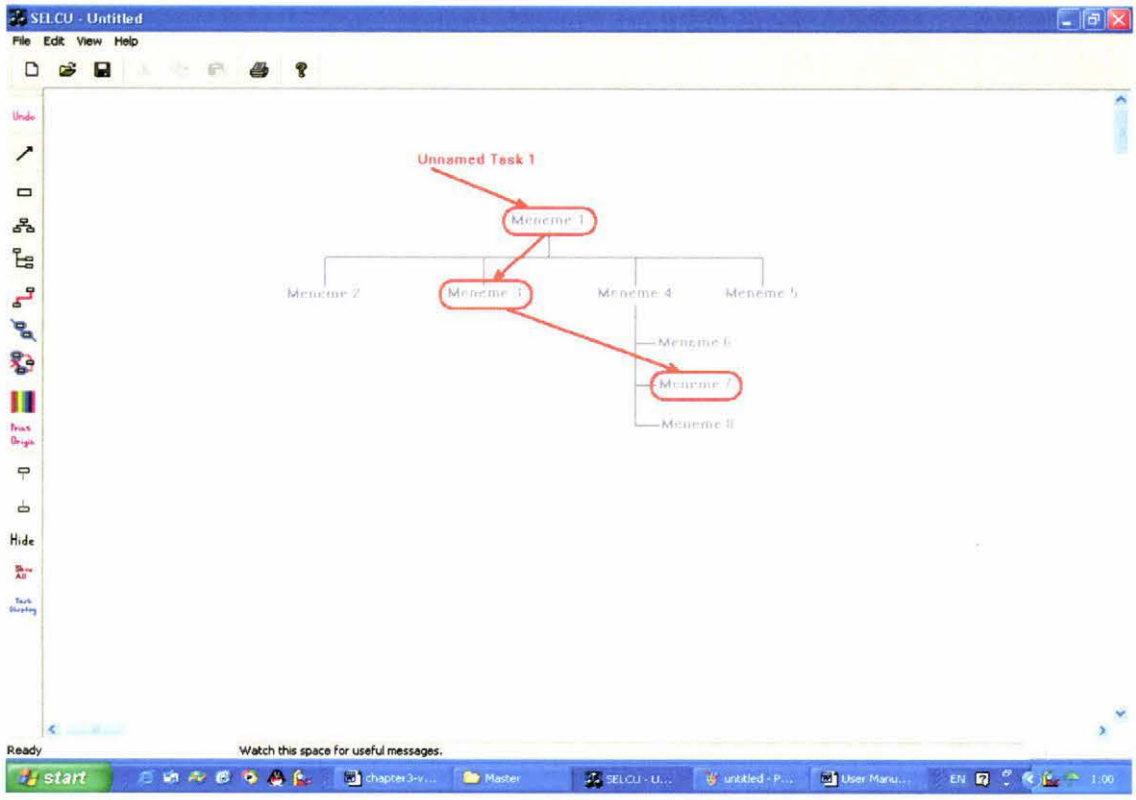


Figure 3-16: Add a Task

When the user designs a basic task sequence on the diagram, they can click and drag the task name to the desired final position. Then they can double click on the task name to open a new "Task Name" dialogue box to allow the addition of a task name (like Figure 3-17).

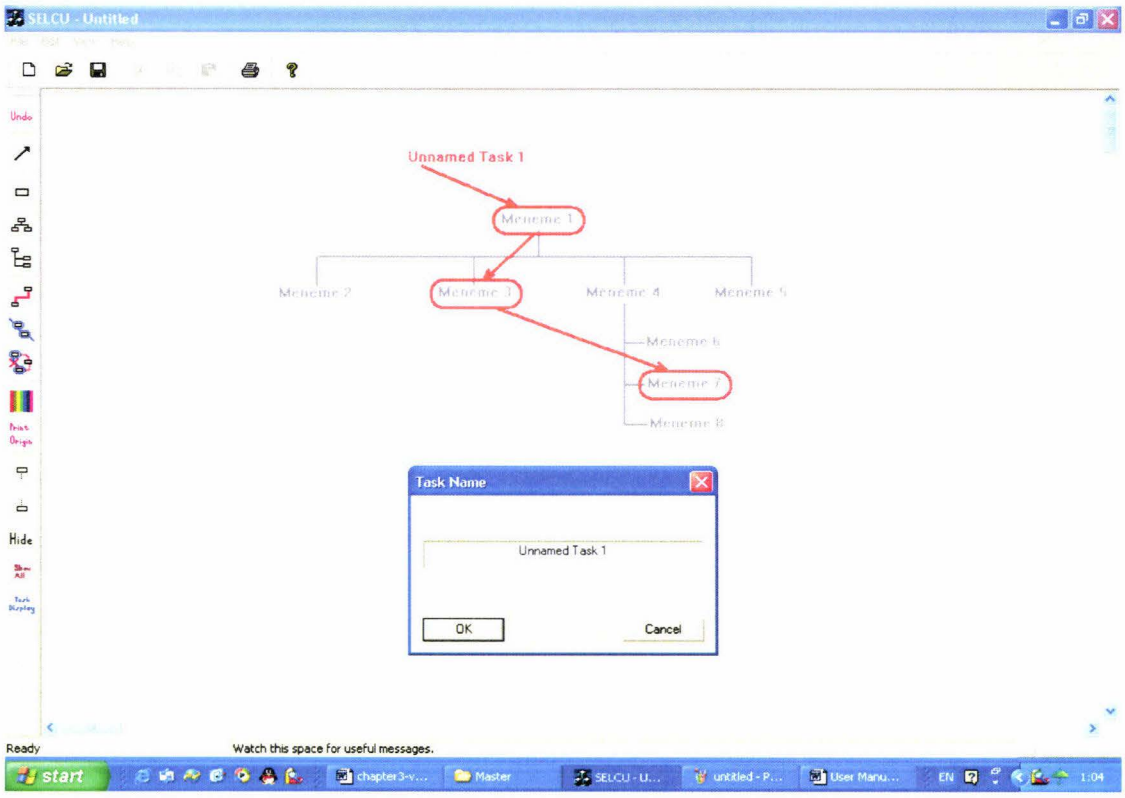


Figure 3-17: Add Task Name

In SELCU, any part of the diagram can be directly selected. Menemes, triggers and tasks can be edited by selecting the object and changing their attributes. A detailed description of SELCU is provided in (Scogings, 2003).

3.3 Proposed Extensions of SELCU

The SELCU extensions include the automatic generation of Lean Cuisine+ diagrams, and their execution environment.

More specifically,

- For the automatic generation , the requirements are:

  (1) The software will take Delphi source code as input

  (2) The software will analysis the Delphi source code and extract code for the interface components, including Lean Cuisine+ "trigger" structures.

(3) The software will analysis the Delphi interface code and translate it into a Lean Cuisine+ structure represented as a SELCU source file

(4) The user can use SELCU to view or modify the file that is generated by this software

(5) When the user attempts illegal operations, the system will give clear error feedback.

- For the execution environment, the requirements are:

(1) The execution environment must sit inside the SELCU software. The user must be able to change to execution mode from the SELCU main interface in a single operation.

(2) The execution environment interface needs to clearly indicate that the user is in execution mode instead of standard SELCU edit mode

(3) Modification of the Lean Cuisine+ diagram is not permitted

(4) The user can select and deselect any available meneme

(5) The system needs to clearly present different states for a meneme (selected & available; not selected & available; selected & not available; not selected & not available)

(6) The system needs to clearly show special attributes for menemes (for example monostable attributes)

(7) A preview mode must be provided so that the user can check out the effect of manipulating any meneme

(8) At any time, the user can request the display of triggers on Lean Cuisine+ diagram

(9) When the user selects a task view of the Lean Cuisine+ diagram, they can view any single task or all the tasks

(10) At any time, the user can request the display of tasks on a Lean Cuisine+ diagram

--- (10.1) the user can single step control the task

--- (10.2) the user can let the system automatically run the task

--- (10.3) at anytime, the system will allow user to change between single step control task and system automatic running task freely

(11) When the user requests an invalid process, the system will give clear error feedback.

(12) At anytime, the user can easily change between preview; task view; trigger view and normal view inside the execution environment

(13) At anytime, the user can switch easily from execution mode change back to SELCU ordinary model

3.4 Summary

Lean Cuisine+, a semi-formal graphical notation for describing the behavior of event-based direct manipulation GUIs, has been described. A software environment (SELCU) is used to support the browsing and construction of Lean Cuisine+ diagrams. This software's GUIs, main functions, and Lean Cuisine+ file format have been discussed. The functional requirements for the proposed extensions of SELCU have been presented.

Chapter 4

CONCEPTUAL DESIGN OF SELCU EXTENSIONS

Two supporting tools for the SELCU system, Auto-Generation Software and Execution Environment are to be developed for Lean Cuisine+. The Auto-Generation Software is a user interface generator which can automatically generate a Lean Cuisine+ diagram from a Delphi interface. The Execution Environment is embedded in SELCU. It will allow users to execute a Lean Cuisine+ model.

This chapter discusses some conceptual design issues relating to the SELCU extensions. Section 4.1 will explain how the proposed SELCU extensions satisfy the functional requirements. In section 4.2, all the transferable Delphi interface components for the Auto-Generation Software are described. Some Lo-Fi prototyping layouts of the Execution Environment are described in section 4.3.

4.1 Design Decisions

The functional requirements have been defined in section 3.3. In this section, design decisions will be related to these requirements.

Auto Generation Software

The Auto Generation Software meets the functional requirements set down.

Specifically:

- This software uses the Delphi IDE as the development environment
- The software can read a Delphi source file and extract the required information
- The final generated file is SELCU compatible
- The software includes adequate error messages

Execution Environment design decisions:

Design decisions are related to requirements:

Requirement 1: The execution environment must sit inside the SELCU software. The user must be able to change to the execution mode from the SELCU main interface in a single operation

Decision: The execution environment is embedded in SELCU. There is an "Execution Mode" button in SELCU. The user clicks this button to change to the execution environment.

Requirement 2: The execution environment interface needs to clearly indicate that the user is in execution mode instead of the ordinary SELCU edit mode

Decision: In the execution mode, the cursor changes to another style, the text "execution environment" appears in the windows title, and most of the function buttons become unavailable. Only the "meneme view" and "open" buttons are available.

Requirement 3: When the user is in execution mode, the Lean Cuisine+ diagram cannot be modified.

Decision: The Lean Cuisine+ diagram is opened as "read only". The system only allows the user to view or execute the notation.

Requirement 4: The user can execute the Lean Cuisine+ diagram

Decision: The execution mode allows users to select menemes and tasks, and it takes account of triggers.

Requirement 5: The system needs to clearly present the four possible states for a meneme

Decision: Text color and line types are used to separate four different states for the meneme.

- The text color **red** means the meneme is **selected**.
- The text color **black** means the meneme is **not selected**.
- A **solid** line means that a meneme is **available**.
- A **dim broken** line means that a meneme is **not available.**

Thus, we will have four states for a meneme:

	Red Name	**Black Name**
Solid Line	Selected & Available	Not Selected & Available
Dim Broken Line	Selected & Not Available	Not Selected & Not Available

Requirement 6: The behavior of monostable menemes must be shown

Decision: Monostable menemes are shown as selected for three seconds, then return to unselected

Requirement 7: A preview mode is required so that the user can check out the effect of manipulating any meneme

Decision: If a meneme name is presented **with underline,** it means the user is in **Preview Mode** (the meneme is selected by pressing the right mouse button).

Requirement 8: At any time, the user must be able to request the display of triggers on the Lean Cuisine+ diagram

Decision: The user can select the "trigger view" function from the system menu at any time

Requirement 9: The user must be able to view any single task, or all the tasks

Decision: The user can select the **task view** function from the system menu at any time. A sub-menu appears to allow users to decide to view a single task or all the tasks.

Requirement 10: At any time, the user must be able to request the display of tasks on the Lean Cuisine+ diagram

10.1 The users can single step control the task

Decision: There is a single step control menu in task view mode.

10.2 The user can let the system automatically run the task

Decision: There is an automatic task button in task view mode

10.3 At anytime, the system must allow the user to change between single step control task and system automatic running task freely

Decision: The single step control menu and the automatic task button appear in the same screen

Requirement 11 When the user requests an invalid process, the system must provide clear error feedback

Decision: This software includes adequate error messages

Requirement 12 At anytime, the user must be able to easily change between preview; task view; trigger view and normal view inside the execution environment

Decision: The preview, task view, trigger view and normal view functions are all selected via a single click.

Requirement 13 At anytime, the user must be able to move easily between execution mode and edit mode

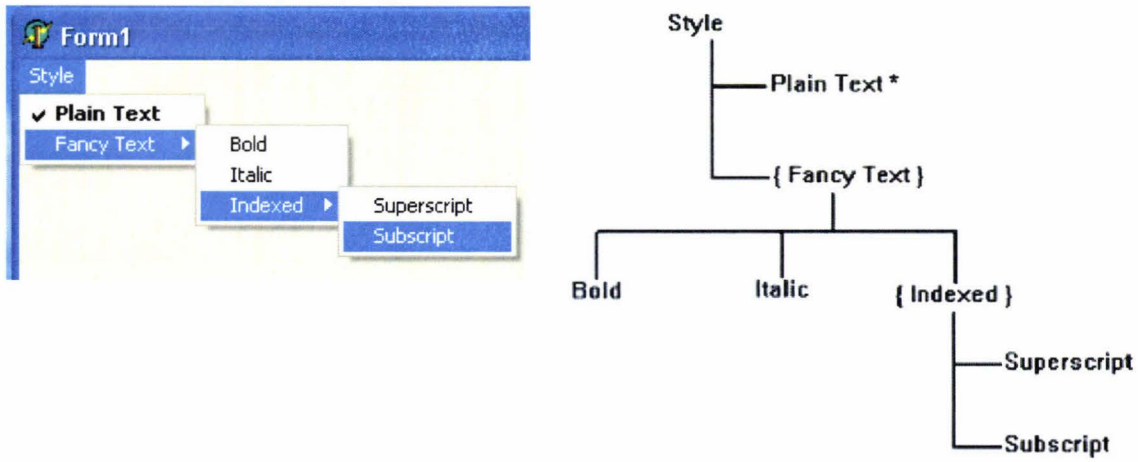
Decision: Mode can be switched via a single button at any time.

4.2 Auto-Generation Software Interface Components

The Auto-Generation Software is a tool used to generate a basic Lean Cuisine+ diagram from a Delphi user interface. After generation, the resulting Lean Cuisine+ file is compatible with SELCU. The following list presents all the transferable Delphi interface components.

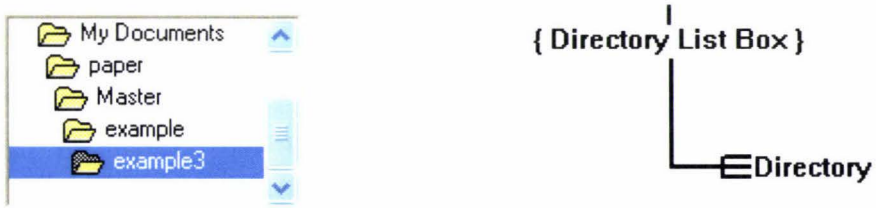
- **Menu**

  The head of the menu (style) is selectable and is represented by a real meneme. The menu sub-headers (fancy text, indexed) are gates to further selectable options, and are represented by virtual menemes. The leaf-nodes are real menemes.



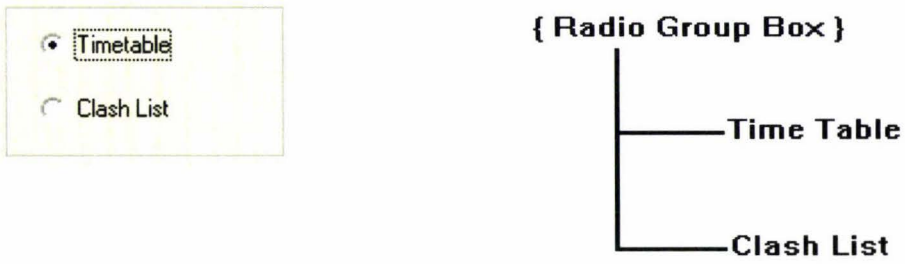
- **Directory List Box**

  The directory list box itself is an access mechanism, represented as a virtual meneme. The options inside the directory are homogeneous and selectable and the number of items is unknown, so here we use the fork structure with a generic real meneme.



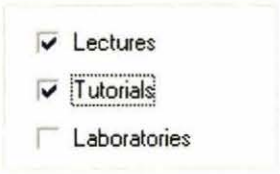
- **Radio Group**

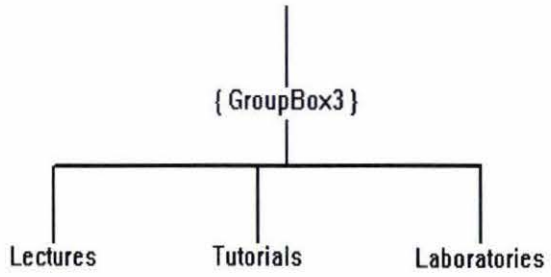
  As for the directory radio group, the radio group header is represented as a virtual meneme, and the options inside the radio group box are represented by real menemes. The options are mutually exclusive.



- **Check Box**

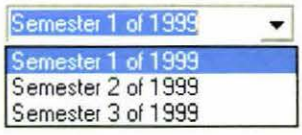
  As for the check box, the check box header is represented by a virtual meneme, and the options inside the check box are represented by real menemes. The options inside the check box are mutually compatible.

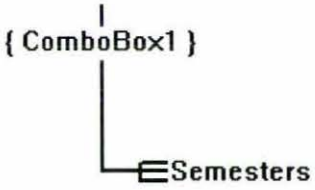




- **Combo Box**

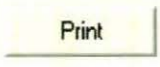
  As for the combo box, the combo box header is represented as a virtual meneme, and the contents inside the combo box are represented by real menemes, and they are mutually exclusive. The options are homogeneous and selectable, and the number of items is unknown, so a generic real meneme with a fork structure is used.





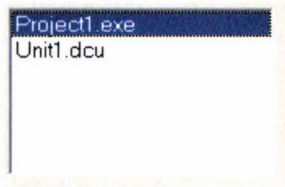
- **Buttons**

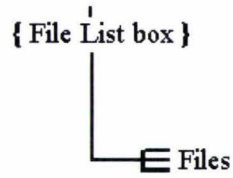
  All the buttons in the system are represented as real menemes. Note that in some situations we can see a button, but we can not select it. Some buttons, after we click them, do not remain in a selected state. They are represented by Monostable menemes (as in the case of Print below).



- **File List Box**

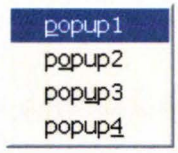
  The File List box is represented as a virtual meneme. The options inside the File List box are homogeneous and selectable, and the number of items is unknown so a fork structure with a generic real meneme structure is used to represent it, and they are mutually exclusive.

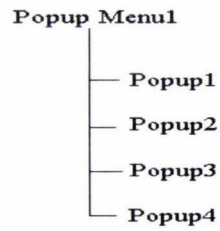




- **Popup Menu**

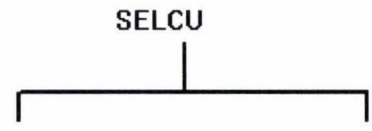
  The header of the popup menu (popup menu1) is selectable and is represented by a real meneme. The contents are mutually exclusive. The leaf-nodes of popup menu are represented as real menemes.





- **Forms**

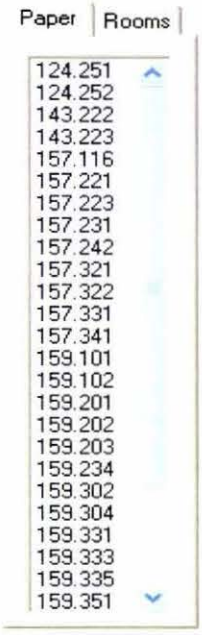
  All the forms are real in the system. In a multi-forms system, one form will generate another new form/window (but we can not see it until the first form generates it). The second form is also represented by a real meneme, because it can directly be selected.

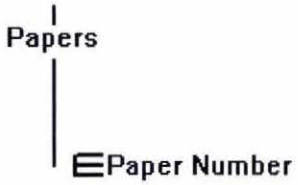


- **Page Control**

  Because we can select the page control name, it is represented by a real meneme. The numbers of items inside is uncertain, so we use a fork

structure. The contents inside the page could be mutually exclusive or mutually compatible, it depends on the requirements.

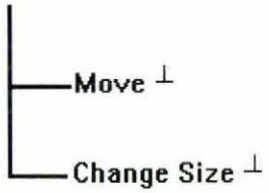




- **Actions**

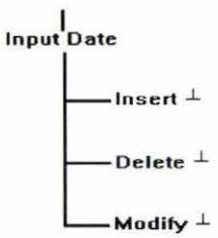
  Actions are represented by real menemes.

  For example, Window's move and change size attributes (which are transient – hence are monostable)



- **Input box**

  The input box is represented by a real meneme. Actions like insert, delete or modify are represented by real menemes, because the user can select these actions. All these actions are represented by Monostable, because they are transient.



4.3 Lo-Fi Prototyping of the Execution Environment

Low-fidelity (lo-fi) prototyping is characterised by a quick and easy translation of high-level design concepts into tangible and testable artefacts. Lo-fi is also know as low-tech, as the means required for such an implementation consists, most of the time, of a mixture of paper, cardboard, post-it notes, acetone sheets etc.

A Lo-Fi prototype is a paper and pencil model of a proposed system. Because it is fast to develop and easy to modify, a Lo-Fi prototype allows designers to refine the system using iterative user centered design procedures. Successful lo-fi test procedures involve the application of ethics, and the use of role playing to encourage the user to interact with the model.

A clear advantage of lo-fi prototyping is its extremely low cost and the fact that non-programmers can actively be part of the idea-crystallisation process. There are several more advantages to lo-fi:

- Using a lo-fi model, it is fast and easy to make changes to the system.
- Because lo-fi is so much easier to create than a coded model, it can be tested much sooner in the design stage.
- Also, because the lo-fi model can be changed relatively quickly, more time is available to try different ideas and repeat tests with users.
- Lo-fi focuses on design issues rather than implementation details.
- Designers are much less reluctant to make minor, or even major, changes to a paper model than they would be to a prototype that took many hours to code.
- Repeated testing builds the designer's familiarity with the system and also builds user confidence.

4.3.1 The Extended SELCU Interface within Execution Function

A detailed description of the SELCU interface appears in section 3.2. The extended SELCU user interface, showing the execution function, is presented below:

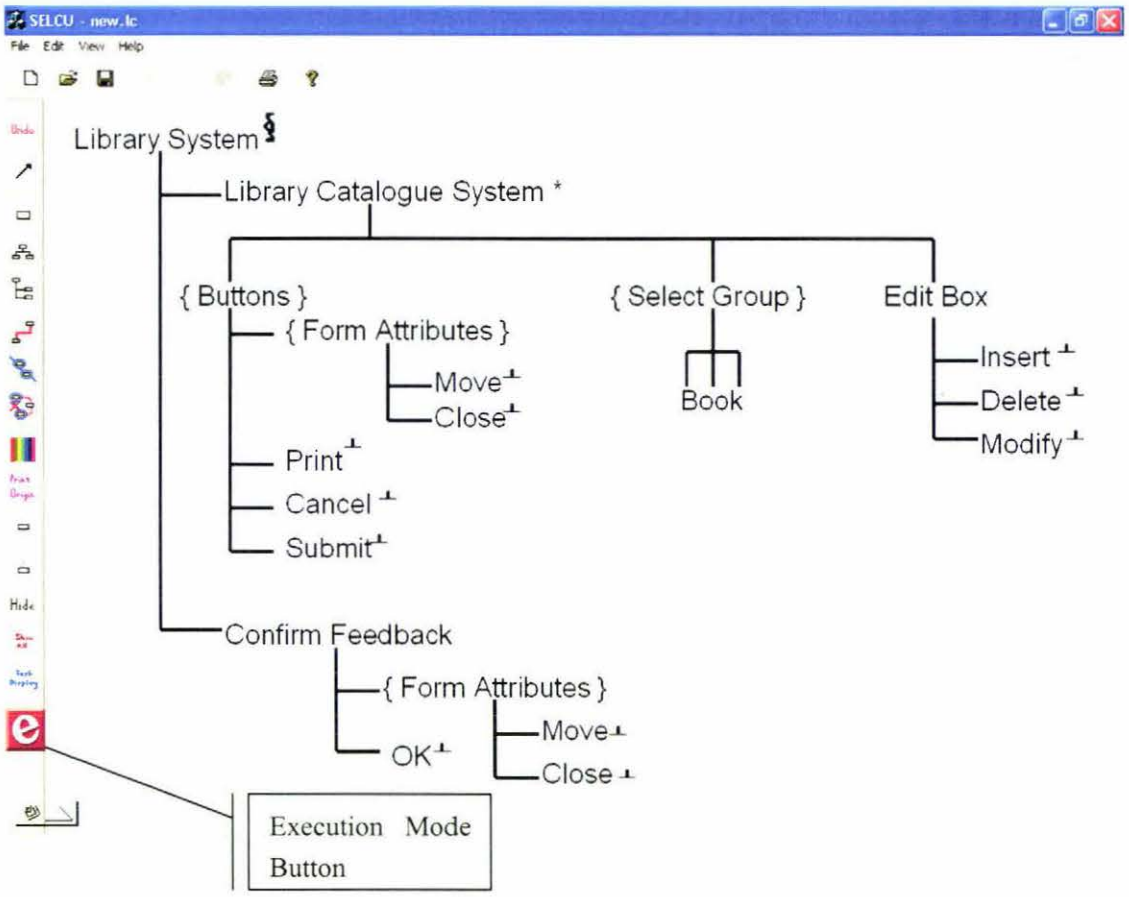


Figure 4-1: Extended SELCU User Interface

In figure 4-1, a red Execution Mode Button has been added to the SELCU main interface. The user can change to Execution Mode by simply clicking that button.

4.3.2 The Main Interface of the SELCU Execution Environment

If the user clicks the red execution mode button in figure 4-1, the software will change to the Execution Mode.

Figure 4-2 shows the main interface after selection of the Execution Mode Button.

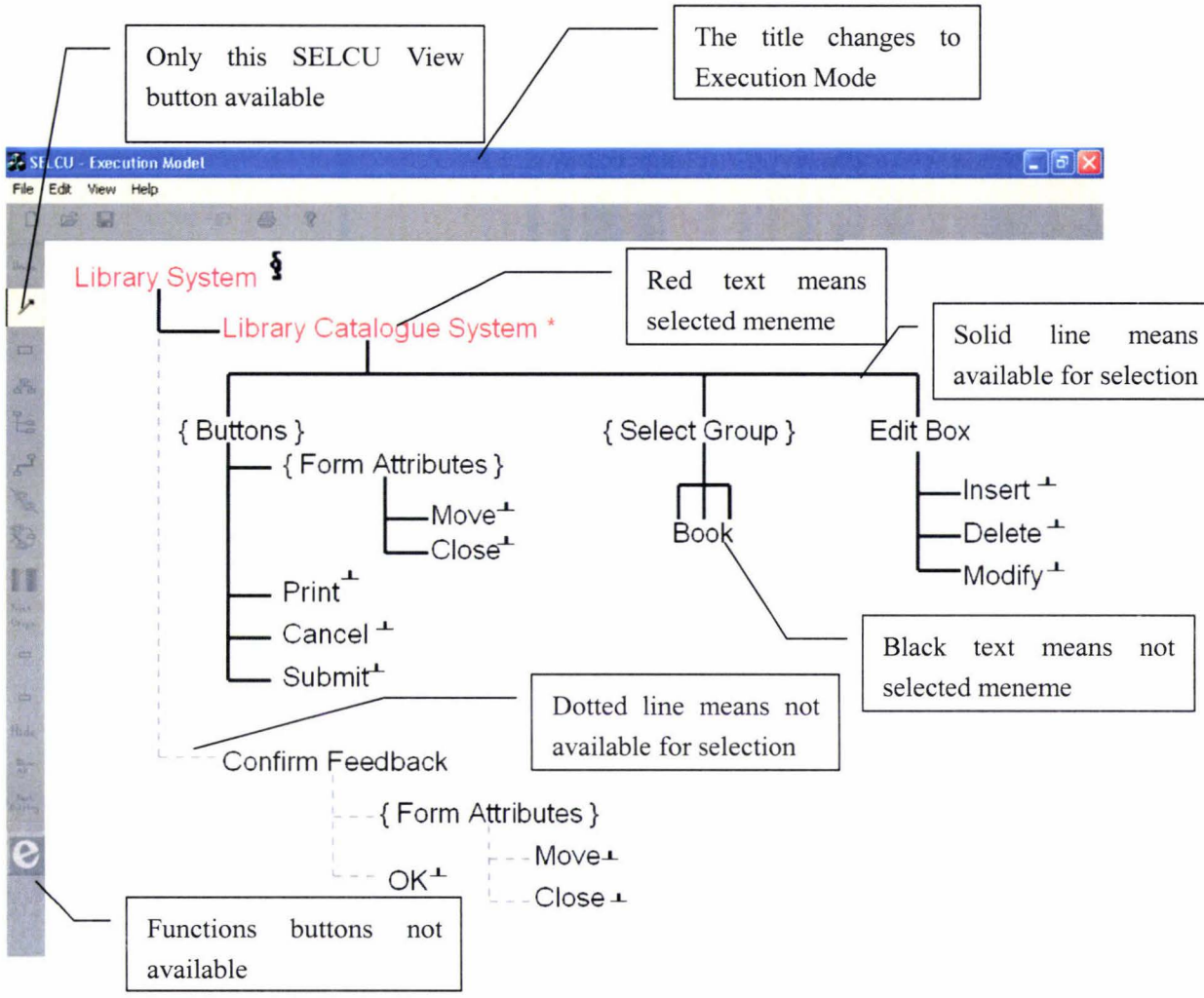


Figure 4-2 Main interface in Execution Mode

In figure 4-2, there are four main indicators which remind the user that they are in the Execution Mode instead of SELCU edit mode.

(1) The title changes to "Execution Mode"

(2) Most of the function buttons become not available, only the SELCU View button remains available (The user can click this button anytime to quit execution mode and return to SELCU edit mode)

(3) The user can not make any changes to the Lean Cuisine+ diagram.

(4) The state of each meneme is shown. Some menemes are red (selected), some menemes are black (not selected), some lines are solid (available) and some lines are dotted (not available).

4.3.3 Preview Mode of Execution Environment

In execution mode, the system reacts differently, if we right click or left click the meneme name. Figure 4-3 presents the interface when the user presses the right mouse button. When the user presses the right button and holds it down the meneme changes to bold which means the user is in the Preview Mode. The preview mode uses animation to show the user what will happen if they select or deselect a particular meneme.

There are five indicators to remind the user that they are in the preview mode.

(1) The selected and relevant meneme names flash in sequence (The selected meneme name flash for two seconds, then each relevant meneme flashes in turn for two seconds)
(2) The selected and relevant meneme names change color in sequence (The selected meneme name changes to red, then all the relevant meneme names change to red)
(3) The selected and relevant meneme names are shown underlined
(4) The line type is changed (black real line or dimmed broken line)to present the meneme state (available or not available)
(5) The triggers are represented by blue arrowhead lines

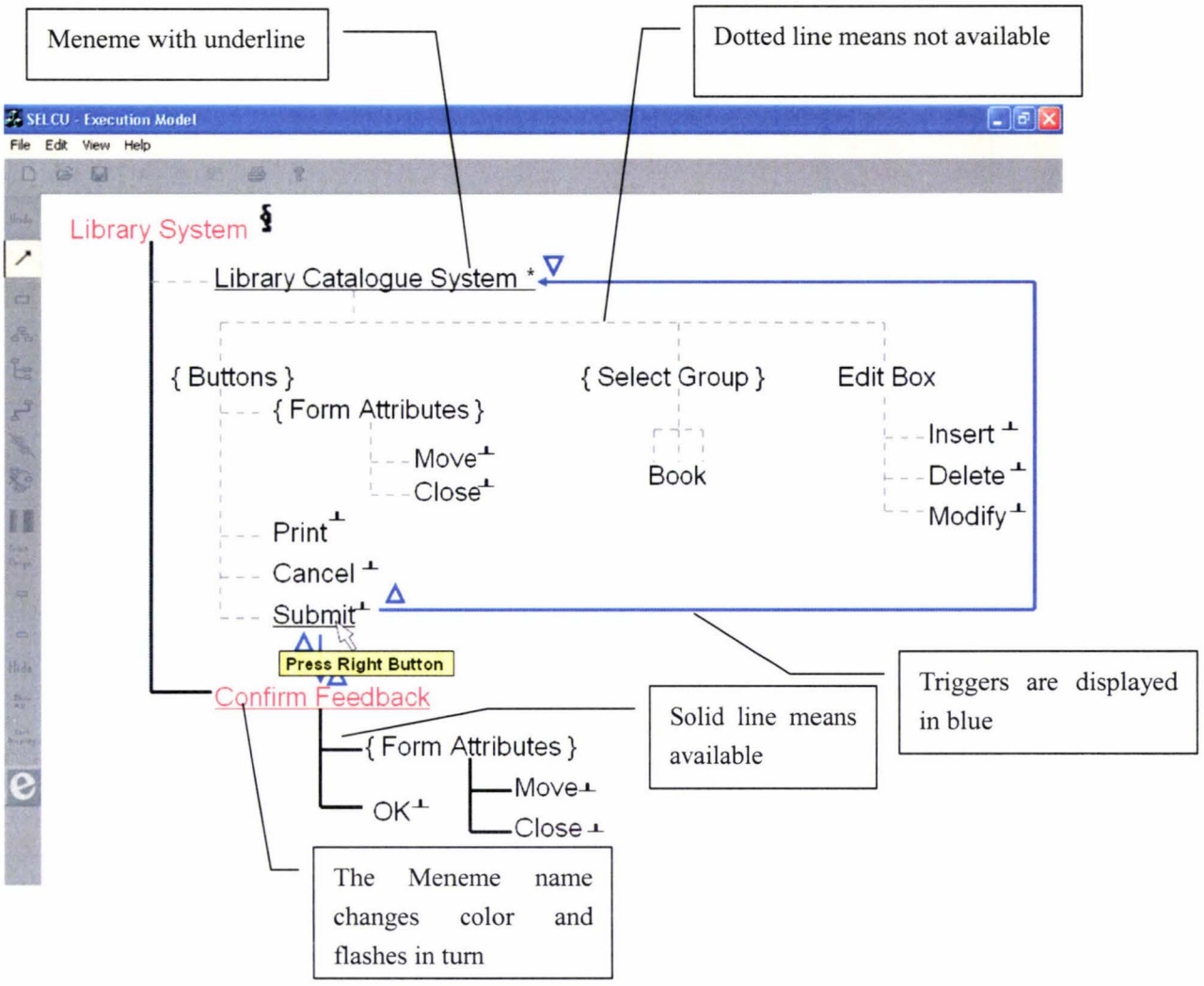


Figure 4-3 Preview of "Submit" Selection

4.3.4 Selecting a Meneme

When the user presses the right button the system provides a preview. If they release the right button, the system will go back to the current state.

If the user does want to change the meneme's state, they can left click the meneme. The system then directly presents the resulting state to the user. At this time, the meneme name directly changes color. Figure 4-4 shows the state of the model following a left click on "Submit".

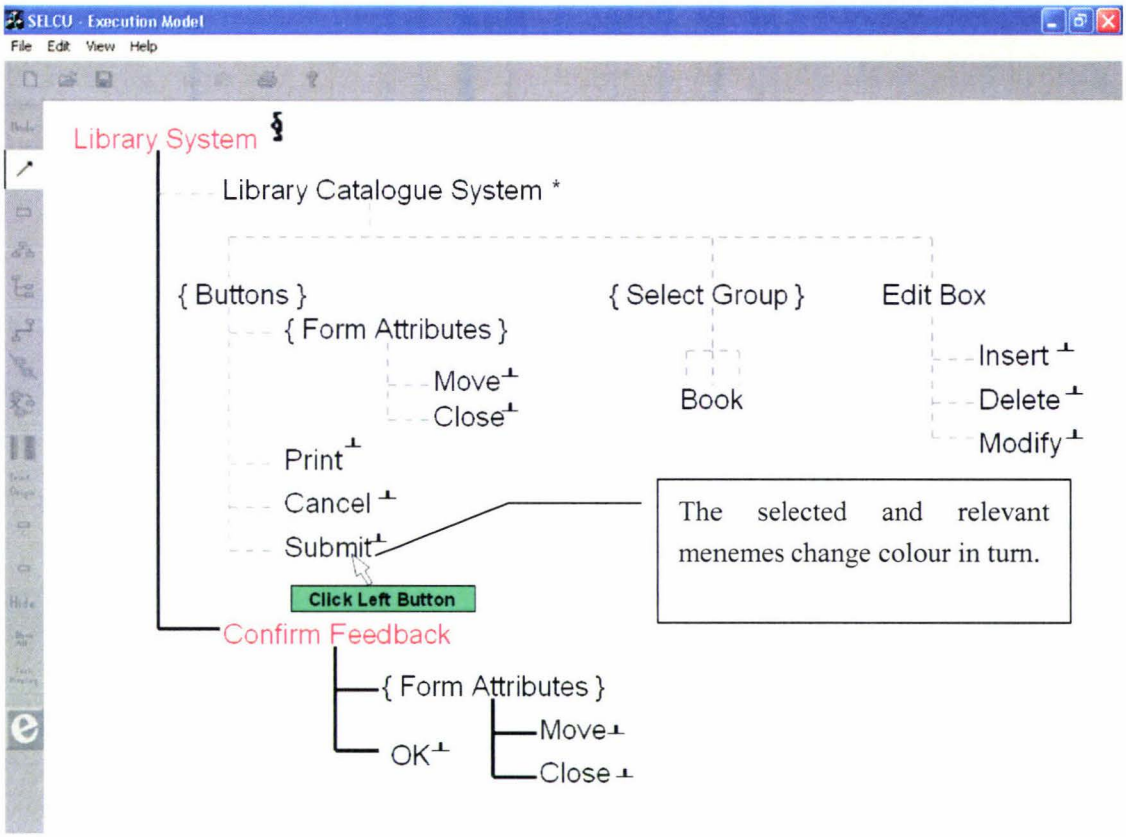


Figure: 4-4 Left Click on "Submit" Meneme

4.3.5 Viewing Tasks in the Execution Mode

If the user wants to see the tasks super-imposed on the Lean Cuisine+ diagram, they can click the "View" menu in the menu bar. After that, a pop down menu list will be presented to the user. The user can select "Task..." to see the task list (Figure 4-5).

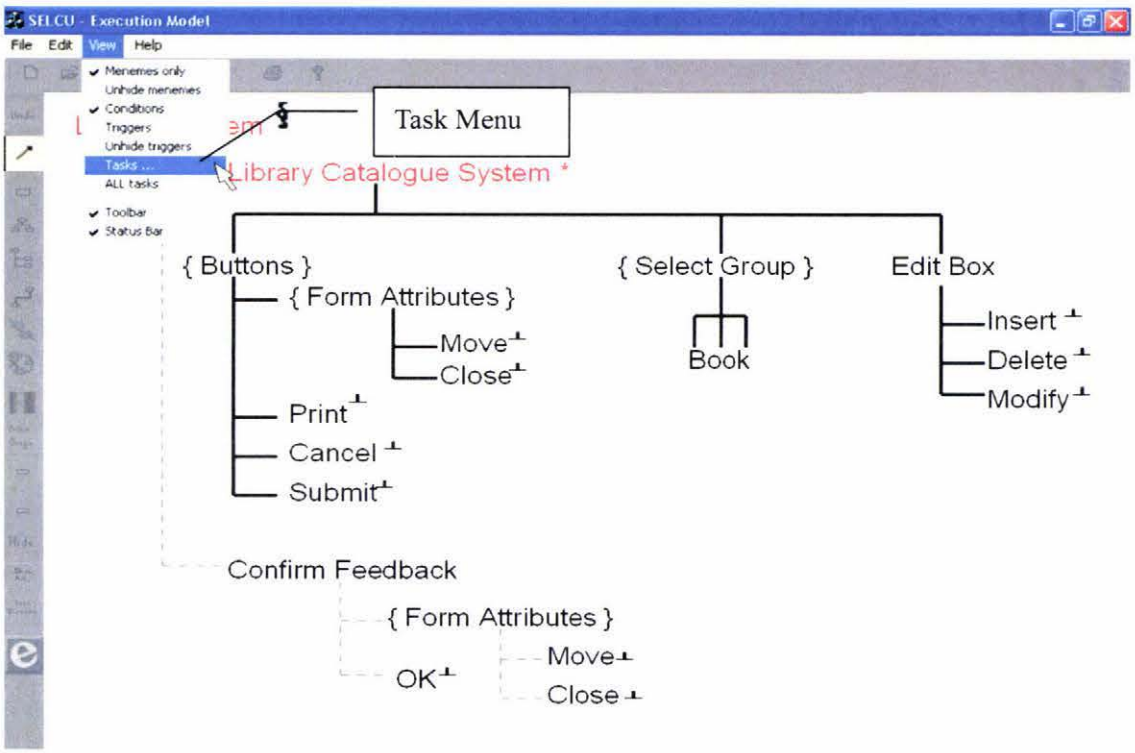


Figure 4-5 How to Access the Task List in Execution Mode

After the user clicks the "Task... " menu in Figure 4-5, a Task Display sub-window will pop up. It lists all the tasks for this Lean Cuisine+ diagram. The user can select one from the list and "OK" button to display that task. Figure 4-6 presents this interface.

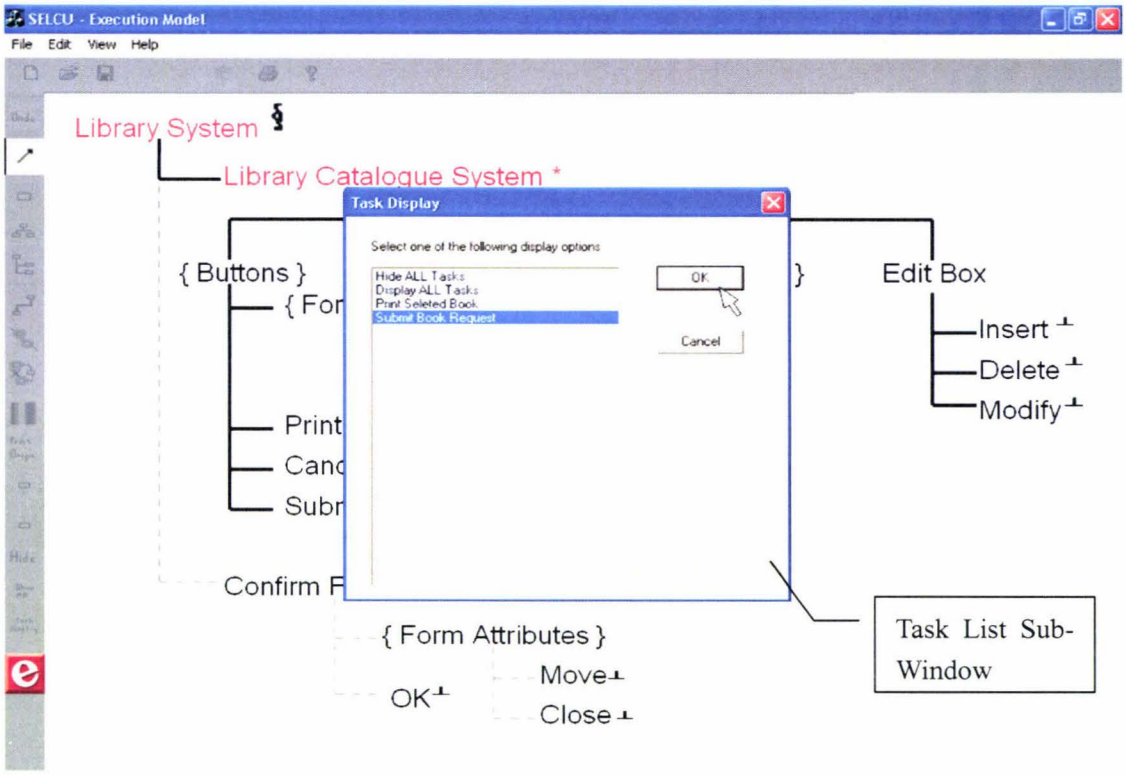


Figure 4-6 Task List Display Sub-Window

After the user selects the task from figure 4-6, they have a "Task View" of the Lean Cuisine+ diagram. In order to let the user to see the task more clearly and control the task step more easily, there are four new indicators in the task interface:

(1) A blue ellipse and arrowhead line are used to show the task sequence

(2) The meneme name color is changed to show whether it is selected or not selected

(3) Flash motion is used to present the task

(4) A button group is provided to "single step control" the task. There are three buttons inside the group: "Previous Button" (go back to last task step); "Next Button" (Move to next task step); and "Auto Button" (which lets the system present all task steps automatically).

Figure 4-7 shows an interface using the "single step control" task view.

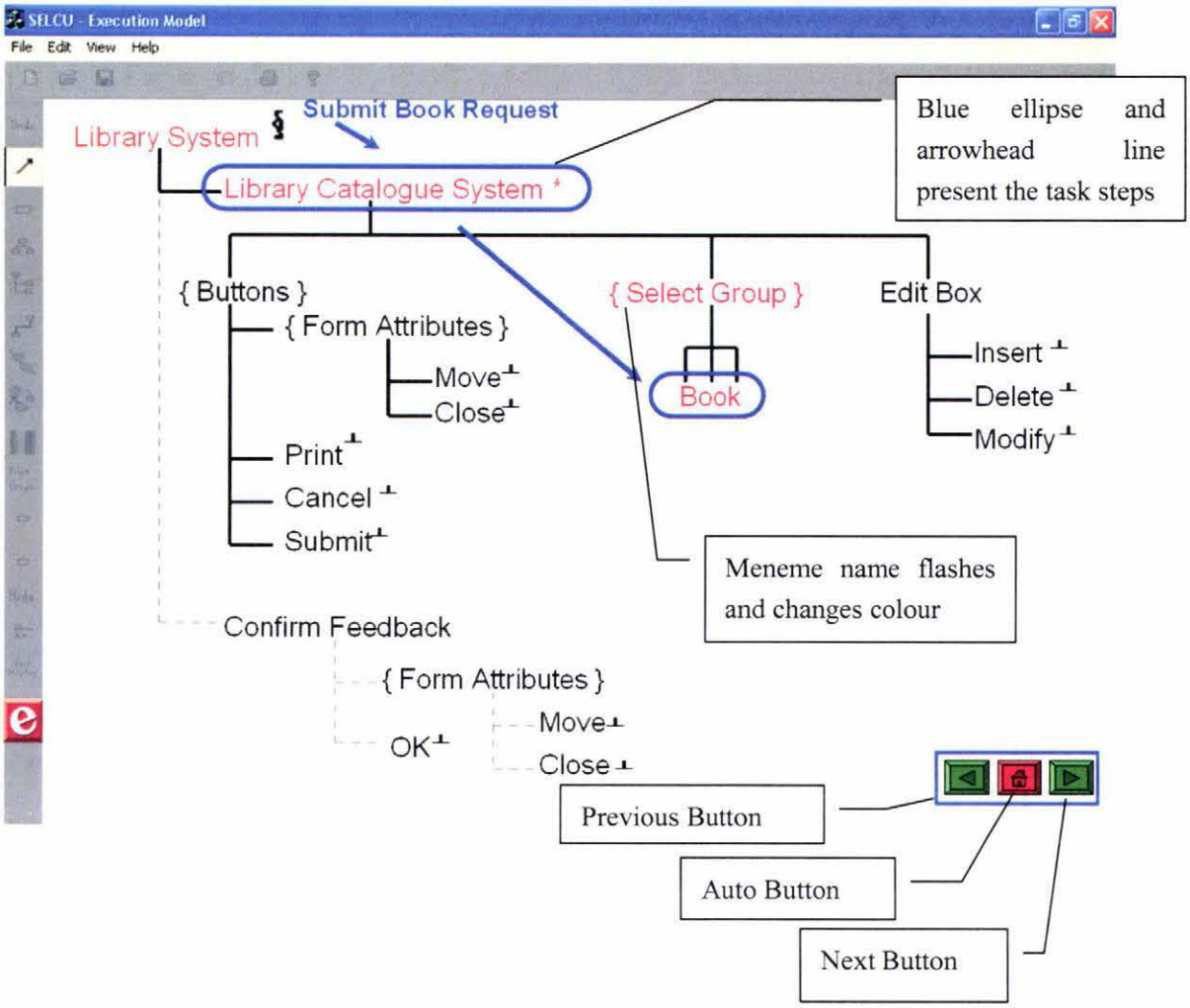


Figure 4-7 Single Step Control Task View

If the user clicks the "Auto Button" inside the task view interface, the system will run all the task steps automatically. At this time, the user cannot single step control the task. They can click the "Single Step" button to go back to the single step control view model (like Figure 4-7). If the user wants to finish the task view function, they can simply click the "Execution Mode" button to jump to the execution mode view, or they can click the "SELCU View" button to finish the execution and go back to the SELCU edit mode.

Figure 4-8 shows the situation when the user clicks the "Auto Button" in the task view interface.

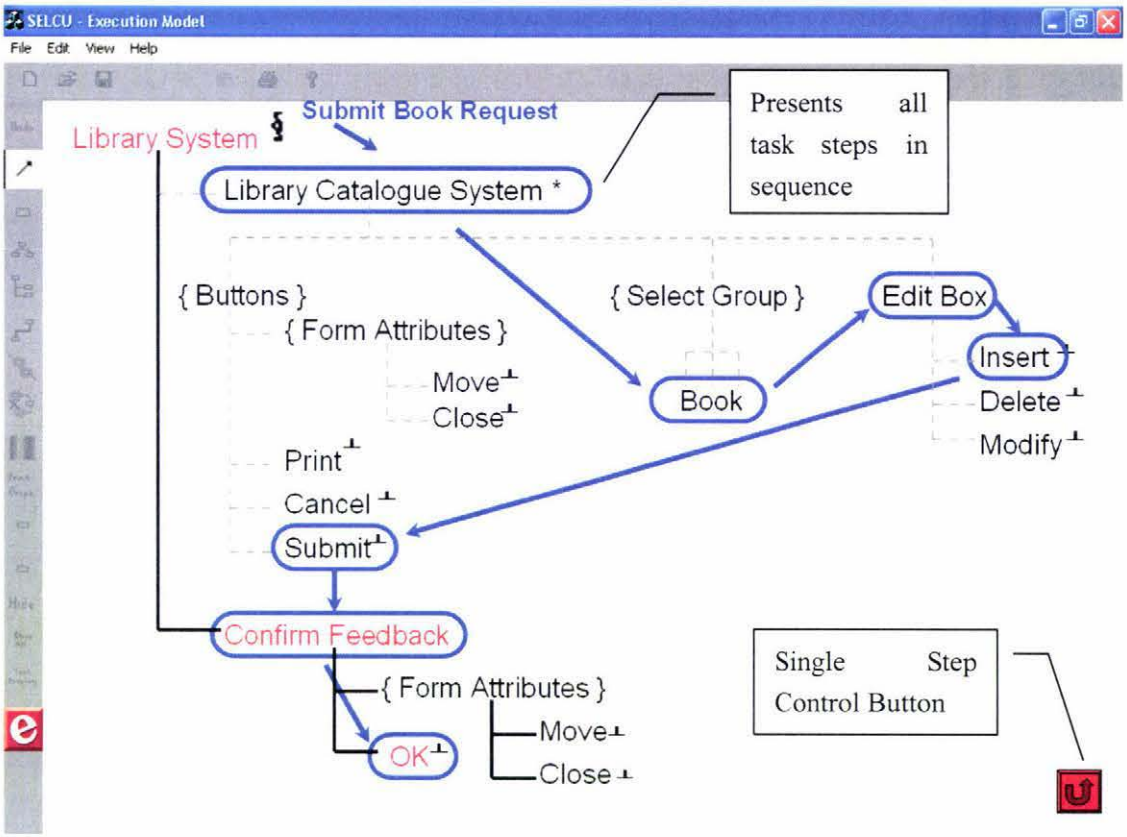


Figure 4-8 System Presents All Task Steps in Sequence

4.3.6 Error Feedback

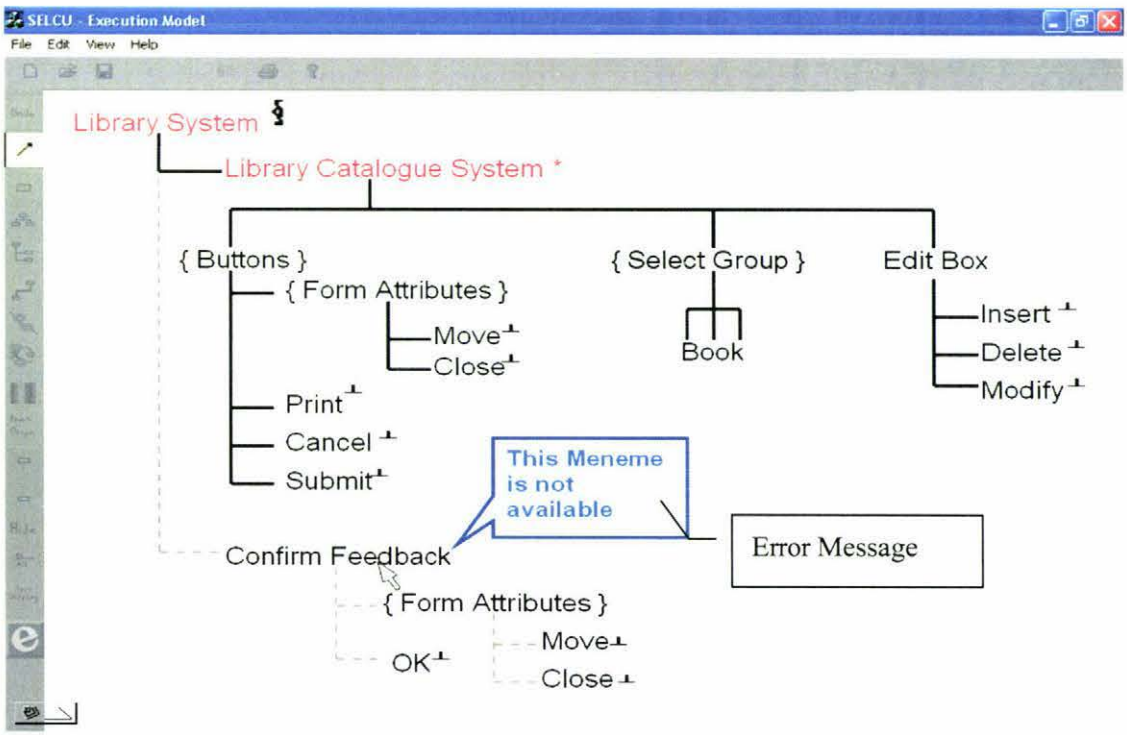


Figure 4-9 Error Message Interface

If at anytime, an unavailable meneme is "selected", the system will popup an error message (as in Figure 4-9) and provide a voice-over.

4.3.7 Trigger View in Execution Mode

If the user wants to see the triggers in this Lean Cuisine+ diagram, they can select the "View" menu on the menu bar, then select "Triggers" from the drop down list (as in Figure 4-10).

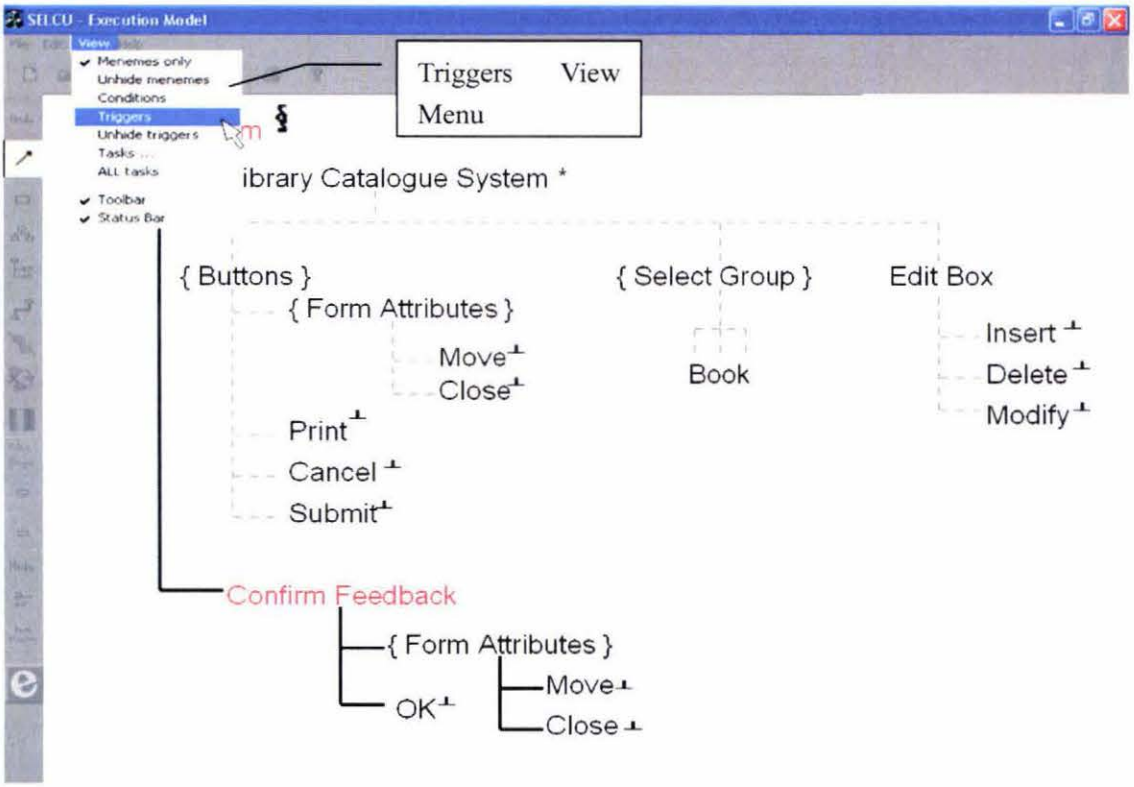


Figure 4-10 Select a Trigger View

The trigger view interface presents all the triggers in the Lean Cuisine+ diagram. The user can click the "Execution Mode" button to end the trigger view and go back to execution mode, or alternatively, they can click "SELCU View" button to go back to SELCU edit mode (as in Figure 4-11).

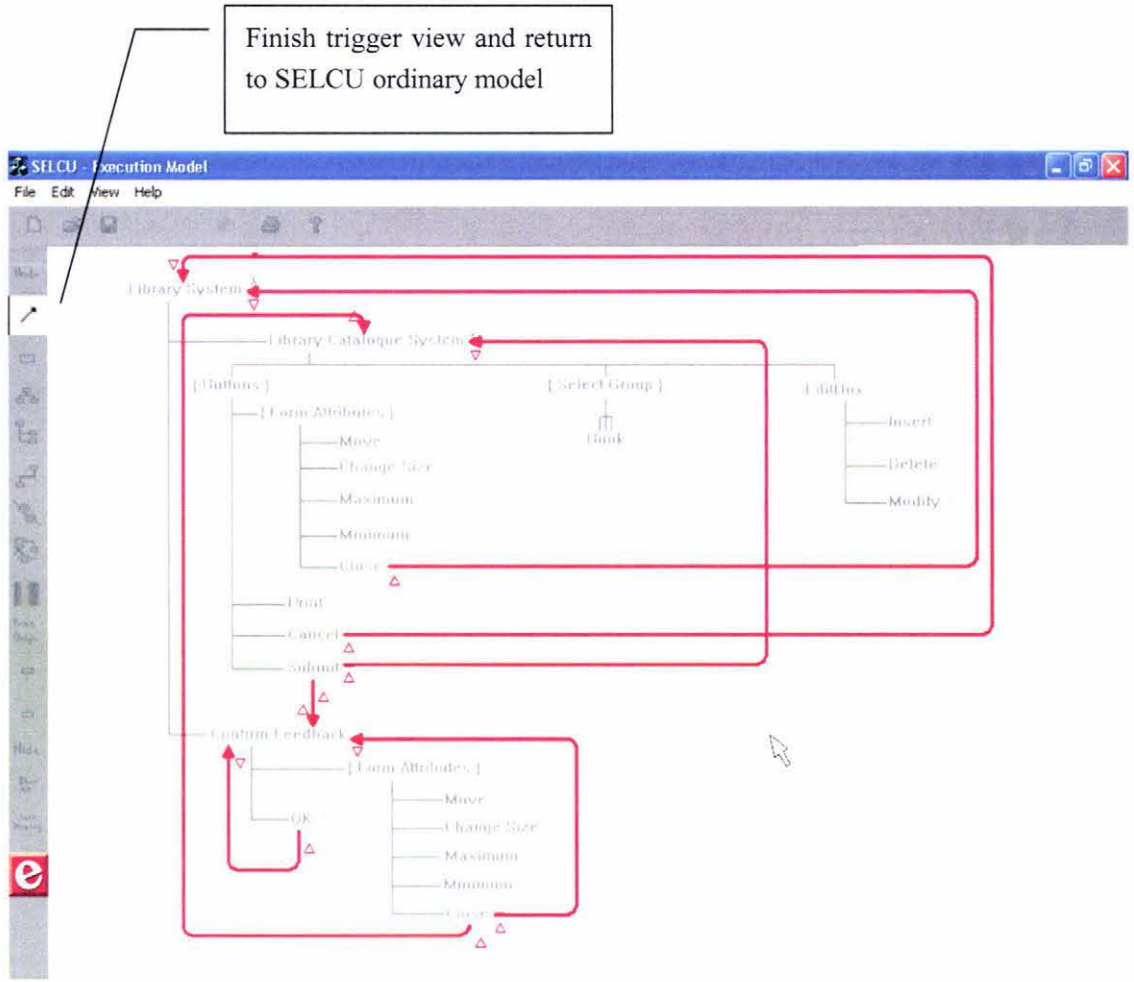


Figure 4-11 Triggers View in Execution Mode

4.4 Summary

A set of design decisions have been made in this chapter. Design issues have been discussed in relation to functional requirements. The interface components supporting the auto-generation software have been described. Power Point has been used to design a lo-fi prototype of the execution environment. It includes the main execution mode interface, a preview mode, a task view, a trigger view and error feedback.

Chapter 5

IMPLEMENTATION OF THE SELCU EXTENSIONS

This chapter describes the implementation of the SELCU extensions, for the Auto Generation System and the Execution Environment. Section 5.1 provides an overview of the extensions. Section 5.2 works through the language choice processes for the software. Development issues associated with the Auto Generation System are presented in Section 5.3 and main implementation issues of the Execution Environment appear in Section 5.4. Section 5.5 is a summary of this chapter. The SELCU User Manual in regard to the extensions is provided in Appendix A.

5.1 Overview of the SELCU Extensions

The SELCU extensions support the Lean Cuisine+ dialogue notation. They have been produced as Windows-based software that will run under Windows® XP with Microsoft Dot Net Framework or similar. They have been separately developed in Delphi® and C++®.

The Auto-Generation Software has been developed to support the translation from a Delphi interface to a Lean Cuisine+ description. The final source file is SELCU compatible. The user can use SELCU to present a graphical view, and can directly transfer to the SELCU environment from the Auto-Generation software.

The Execution Environment has been designed to support the execution of the Lean Cuisine+ notation within the SELCU system. A Lean Cuisine+ diagram can be "marked" to show its state at a given point in the dialogue in terms of meneme states. Menemes can be shown as being either selected or unselected, and either available for selection or unavailable. This provides for the "execution" of a Lean Cuisine+ specification, and simulates dynamically the behaviour of the dialogue as selections are made (Scogings, 2000). This process takes cognisance of all specified constraints, such as tasks and triggers. The environment is embedded in SELCU, and developed in C++.

5.2 Language Choice

The selection of development languages was based on the following criteria.

- **The language to be used for developing SELCU**

  SELCU has been developed in Microsoft Visual C++ 6.0. The execution environment is to be imbedded in SELCU, and therefore the language must be similar or compatible with Visual C++.

- **The language to be used for the Auto-Generation Software**

  The Auto-Generation Software takes a Delphi interface as input. Its implementation language needs to be capable of easily recognizing and analyzing Object Pascal (Delphi).

- **Object-Orientation**

  Delphi and C++ are object-oriented programming languages. The object-oriented nature of Delphi and C++ fits well into the underlying object model of the Lean Cuisine+ system, and is ideal for the event-driven nature of window based graphical user interfaces.

- **Ease of GUI Design**

  A large portion of the SELCU extensions are concerned with interface design, displaying Lean Cuisine+ diagrams, and allowing the user to manipulate these diagrams on the screen in an intuitive manner. An effective, easy to use IDE is therefore essential.

- **Portability**

  Although portability is not essential for the construction of the software, being able to run this project software on multiple platforms is a significant advantage.

- **Past Experience**

  The programmer's past experience is an important point for the language selection. If the programmer is familiar with the language, higher productivity can be achieved, and code will probably be more efficient. However, if a new language fits the problem better, then it is likely that the overhead of leaning this new language will be minimal compared to the productivity gains from using the better tool.

5.2.1 Decision

Two separate decisions have been made to suit this project:

- **For the Auto-Generation Software**

  The SELCU compatible file is generated from a Delphi user interface, so Delphi will be used to design the Auto-Generation Software. Delphi is an object oriented programming language, with good support for GUI design.

- **For the Execution Environment**

  The execution environment is an extension of the SELCU system, and is embedded in SELCU. SELCU has been developed in C++. In order to optimize the source code and retain compatibility with the original system, C++ has been selected to implement this environment. C++ is an object-oriented programming language, with good portability.

5.2.2 Brief Introduction to the Selected Languages

- ***Object Pascal (Delphi)***

  Delphi is a Rapid Application Development (RAD) environment with an Integrated Development Environment (IDE) and a Visual Components Library. A Delphi program is actually written in Object Pascal, a textual Object-Orient programming language. Delphi is a powerful and easy to use tool for generating stand-alone graphical user interface (GUI) programs.

- ***Visual C++***

  C++ is a programming language, which retains all of C's strengths such as power and flexibility in dealing with the hardware/software interface, plus its low level system programming. Visual C++ is a good programming language for developing graphical programs that run on platforms such as Macintosh and Windows. Visual C++ extends the C++ language by allowing users to include Microsoft Foundation Classes that can be used as building blocks for creating Windows applications. One of the C++ GUI tools is Microsoft Visual C++. It is a Windows-based, visual development environment for creating C++ applications.

5.3 Implementation Issues of the Auto Generation Software

The Auto-Generation System has been developed to generate a Lean Cuisine+ diagram from a Delphi produced user interface. All the Delphi interface components that can be recognized by the Auto-Generation System have been discussed in section 4.2. The system includes two main functions to analyze the Delphi source files. By using these two functions, a Lean Cuisine+ diagram can be generated from Delphi source code.

5.3.1 Basic Tree Structure Function

The *basic tree structure function* is the first main part of the Auto-Generation System. It opens the *.dfm file, extracts meneme names from the code, and builds the basic tree structure for the Lean Cuisine+ diagram. Figure 5-1 shows a simple interface developed using the Delphi IDE.

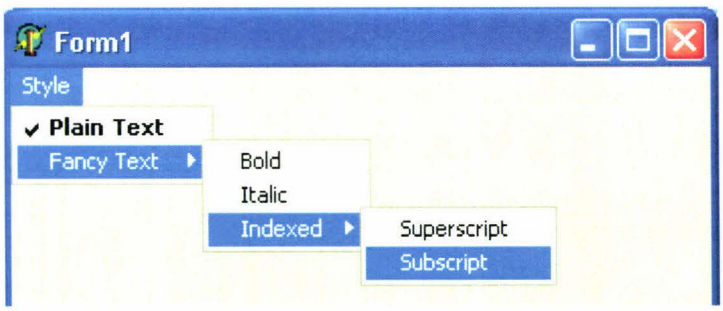


Figure 5-1: Delphi Interface

The "dfm" file of the above example contains the main tree structure of the interface. When the software opens the "dfm" file, the code is presented as in figure 5-2. Only the core part of the code is used in the analysis of the interface.

```
object Form1: TForm1
  Left = 192
  Top = 114
  Width = 698
  Height = 496
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Menu = MainMenu1
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object MainMenu1: TMainMenu
    Left = 40
    Top = 72
    object Style1: TMenuItem
      Caption = 'Style'
      object PlainText1: TMenuItem
        Caption = 'Plain Text'
        Checked = True
        Default = True
        OnClick = PlainText1Click
      end
      object FancyText1: TMenuItem
        Caption = 'Fancy Text'
        object Bold1: TMenuItem
          Caption = 'Bold'
          OnClick = Bold1Click
        end
        object Italic1: TMenuItem
          Caption = 'Italic'
          OnClick = Italic1Click
        end
        object Indexed1: TMenuItem
          Caption = 'Indexed'
          object Superscript1: TMenuItem
            Caption = 'Superscript'
            OnClick = Superscript1Click
          end
          object Subscript1: TMenuItem
            Caption = 'Subscript'
            OnClick = Subscript1Click
          end
        end
      end
    end
  end
```

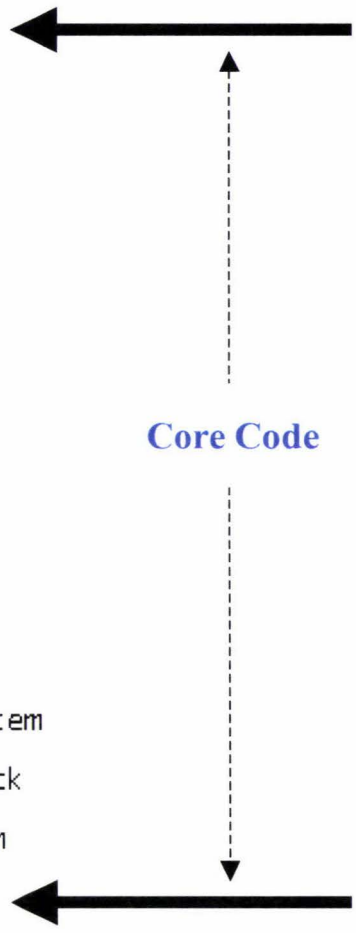


Figure 5-2: Main Part of Example code in "dfm" File

The auto-generation software opens the "dfm" file, and read information from the first line. The software searches for key words (such as *Object, MainMenu,*

and so on) from the core code. The core code (in figure 5-2) includes the following useful information:

- *object MainMenu1: TMainMenu*

  The code tells the software there is a menu item in the interface.

- *Caption = 'Style'*

  The Pascal code "Caption" translates into a meneme name.

- *Default = True*

  The Pascal code "Default" means that this meneme is a default one.

- *object PlainText1: TMenuItem*

  The "object" in the Pascal code signifies the start of a new meneme description.

- *End*

  The Pascal code "End" indicates the end of a meneme description. Every "End" corresponds to the nearest "object"

- Objects may be embedded in other objects, which signifies a parent-child relationship.

The extracted tree structure extracted from the core code is shown in Figure 5-3. Please note that:

(1) This content is saved in a temp file and the user does not see the tree structure.

(2) The basic tree structure includes only meneme names and father/son relationships.

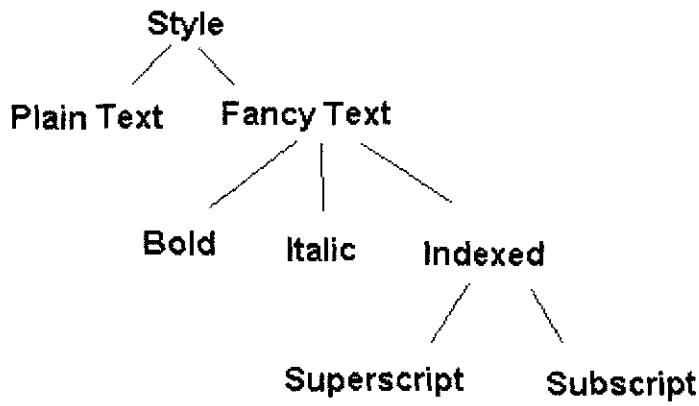


Figure 5-3: Basic Tree Structure for Delphi Interface

5.3.2 The Lean Cuisine+ Attributes Function

When the basic tree structure has been generated (figure 5-3), the auto-generation software starts to analyze the "pas" file to determine the logical relationship between the nodes. The software opens the corresponding "pas" file, and reads the information from the first line. By examining the key words (for example "procedure" or "TForm1.PlainText1Click"), the software locates the core code, describing the logical relationship between meneme "bold" and other nodes. Figure 5-4 presents a piece of core code. The code shows that:

- **In code part A,** if "bold1" becomes "true", then "plainText1" changes to "false". This means that "bold" and "Plain Text" are mutually exclusive.

- **In code part B,** if "bold1" is "true", "italic1" is "true". This means "bold" and "italic" are mutually compatible.

- **In code part C**, if "bold1" is true, "superscript1" still remains "true". This means "bold" and "superscript" are mutually compatible.

- **In code part D**, if "bold1" is "true", "subscript" still remains "true". This means "bold" and "subscript" are mutually compatible.

- **Code part E** is same meaning as **code part A**. This means that "bold" and "Plain Text" as mutually exclusive.

```
procedure TForm1.Bold1Click(Sender: TObject);
begin
      if bold1.checked =false then                                    } A
        begin
          PlainText1.Checked:=false;
          Bold1.Checked:=true;
        end
      else if italic1.Checked =true then                              } B
           begin
             Plaintext1.Checked:=false;
             bold1.Checked:=false;
           end
           else if superscript1.Checked =true then                    } C
                begin
                  plaintext1.Checked:=false;
                  bold1.Checked:=false;
                end
                else if subscript1.Checked=true then
                     begin                                           } D
                       plaintext1.Checked:=false;
                       bold1.Checked:=false;
                     end
                     else
                       begin                                         } E
                          plaintext1.Checked:=true;
                          bold1.Checked:=false;
                       end;
end;
```

Figure 5-4: A Piece of Example Core Code in "pas" File

The software determines real and virtual menemes as follows:

- If an object (meneme) in the "dfm" file has corresponding core code in the "pas" file, this meneme is a real one.

- If an object (meneme) in "dfm" file dose not has corresponding core code in the "pas" file, this meneme is a virtual one or this meneme is the header of this Lean Cuisine+ diagram.

The "Lean Cuisine+ Attributes" function analyzes the *.pas file, and adds more Lean Cuisine+ attributes into a SELCU compatible source file. As a result of this function, all the menemes are distinguished real and virtual. Their internal relationships (such as mutually exclusive or compatible) are presented on the diagram. Figure 5-5 shows the main structure of the Auto-Generation Software.

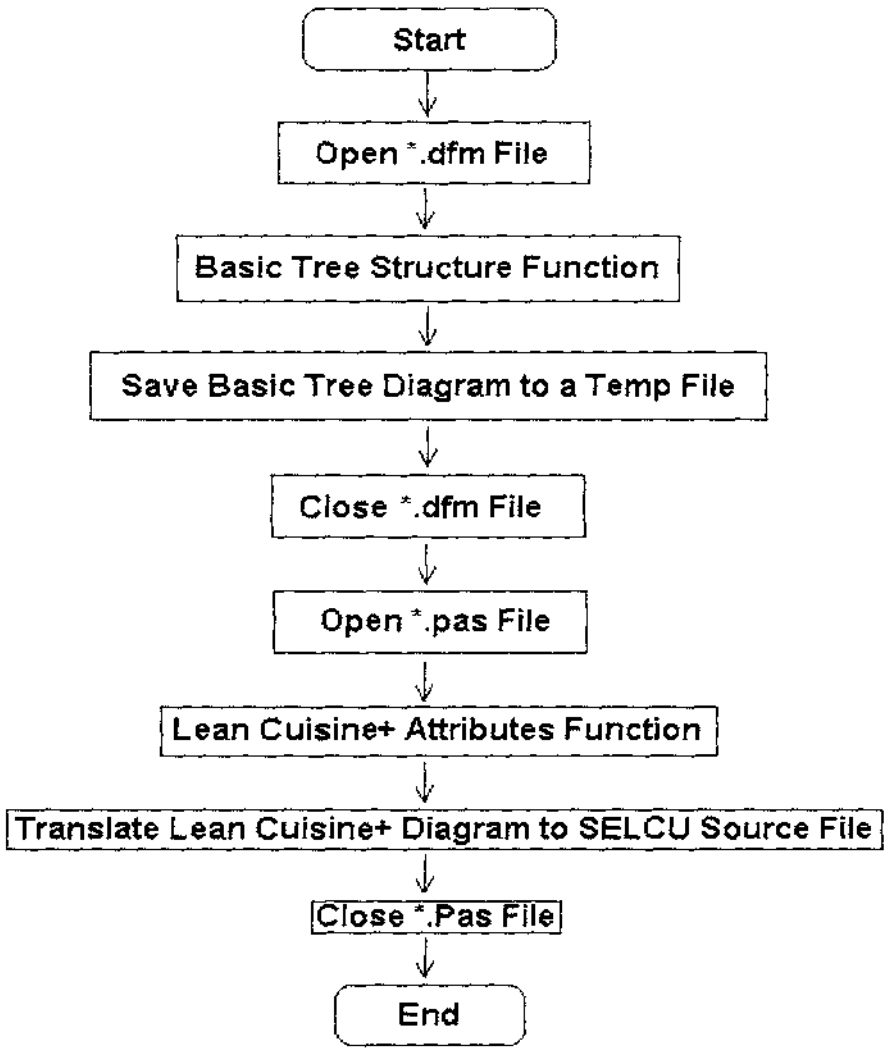


Figure 5-5: Main Auto-Generation Software Process Structure

5.4 Implementation Issues for the Execution Environment

The Execution Environment allows users to execute the Lean Cuisine+ diagram within the SELCU system. The execution involves the selection of menemes and tasks, and takes accout of any triggers. There are four main implementation issues included in this environment, which are discussed below.

5.4.1 Meneme Selection

The Meneme selection component lets user select real menemes and provides feedback to the user. More specifically, the implementation of meneme selection includes the following five sub-programs:

- If a real meneme is not monostable, the sub-program changes its text color to red (means this meneme is selected, as shown in figure 5-6).

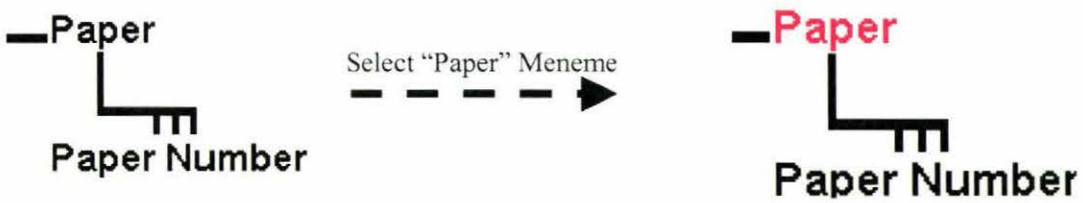


Figure 5-6 Selecting a Meneme

- If a real meneme is monostable, the sub-program changes its text color to red for five seconds. Then it returns to a not selected state. This sequence is shown in figure 5-7.

Figure 5-7: Selecting a Monostable Meneme

- If other menemes are mutually compatible with the selected meneme, the sub-program retains their original state. This process is shown in figure 5-8.

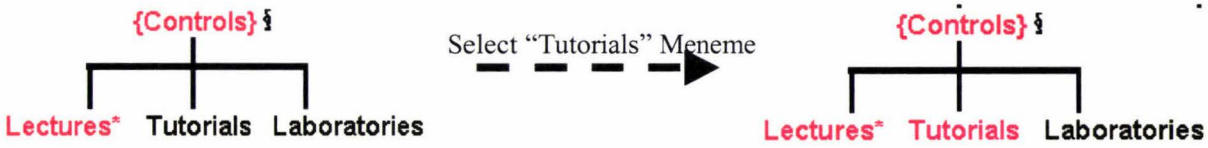


Figure 5-8: Mutually Compatible Group

- If the other menemes are mutually exclusive with the selected meneme, the sub-program deselects them (shown in figure 5-9).

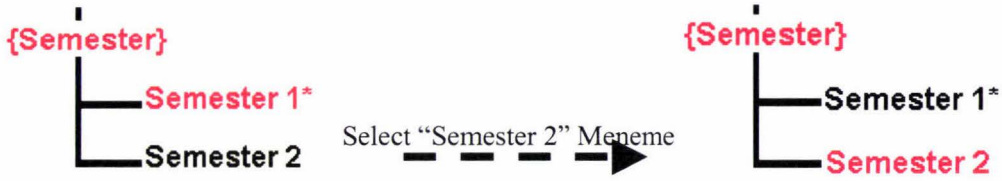


Figure 5-9: Mutually Exclusive Group

- If the user tries to select a virtual or not available meneme, the sub-program pops up error feedback. Figure 5-10 shows this action.

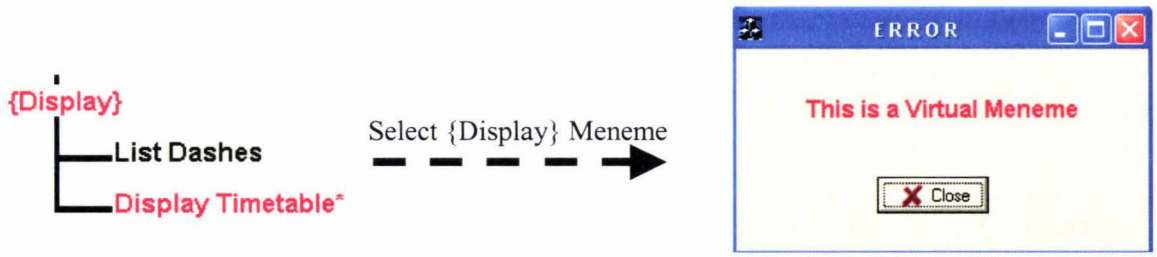
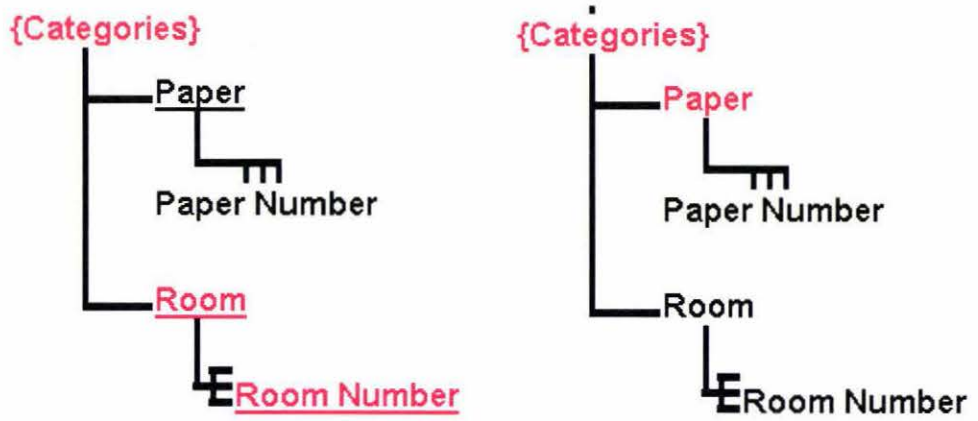


Figure 5-10: Error Feedback

5.4.2 Meneme Preview Mode

The implementation of the Meneme preview part of the program is used to let the user have a preview of the effect of meneme selection. This program detects the right click as selecting action preview mode. Two sub-programs are implemented here:

- Sub-program one is used to detect the right pressing mouse button to start the preview function. If the user releases the right button, the meneme preview mode terminates, and all the menemes go back to their original states. The underlines are developed on the names to present the changed menemes. Figure 5-11 compares preview and original states of a sub-dialogue.



Preview "Room Number" Meneme Selection **Original States**

Figure 5-11: Preview and Original States

- Sub-program two in the preview mode is used to take triggers into account, if an object meneme includes triggers to other menemes. This sub-program locates the triggers and presents them in preview. After the user releases the right button, the diagram returns to its original state.

  Figure 5-12 and figure 5-13 present the original state and preview state where triggers are involved.

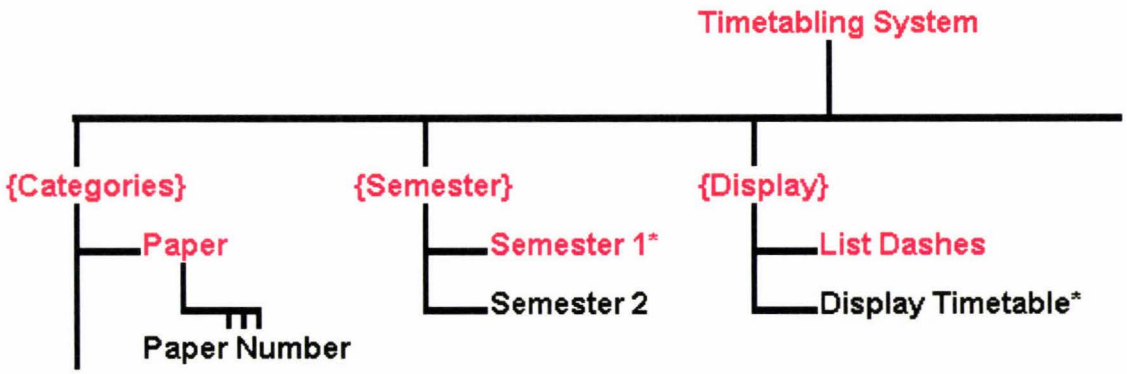


Figure 5-12: Original States before Preview

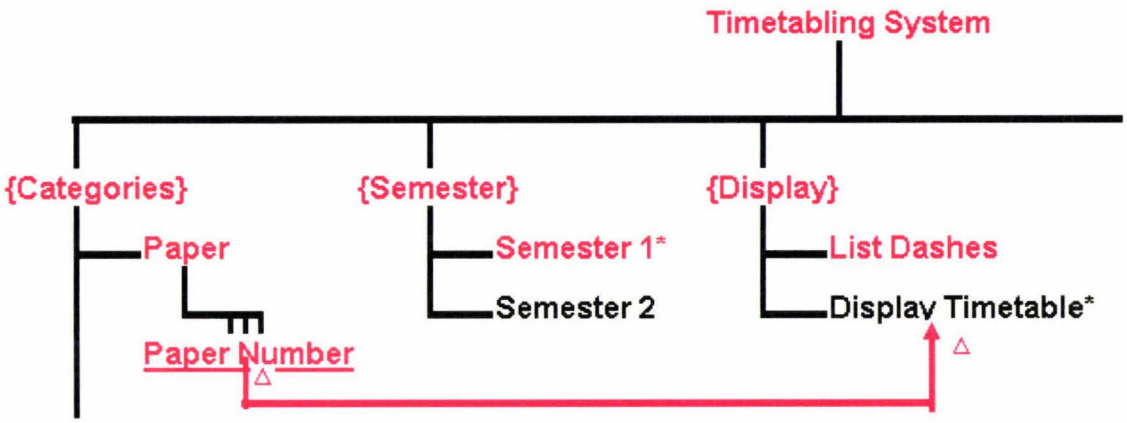


Figure 5-13: Preview Showing Trigger

5.4.3 Task View Mode

The implementation of this sub-program involved two steps:

- Step one, the sub-program gets task information from the SELCU source file (section 3.2.2), and saves this in the computer memory.

- Step two, the task view sub-program displays the task information on the diagram. Because the execution task requires single step control, a global public variable is used to remember the task steps. Two structure arrays save the menemes state parameters. The numbers of

task menemes depends on the Lean Cuisine+ diagram, so these two arrays are designed as dynamic link structures.

The implementation of task view also includes Meneme selection code. As the software performs each task step, it causes selection or de-selection of menemes.

5.4.4 Triggers View Mode

The implementation of the triggers view is used to present all the triggers in the system. The triggers information is taken from the SELCU source file (section 3.2.2.). This sub-program reads them from the file, and then presents all the triggers in a read only style.

A global trigger action sub-function is designed to allow access to other sub-programs, should the meneme selection, meneme preview or task view programs include any triggers.

The main structure of the Execution Environment is shown in figure 5-14.

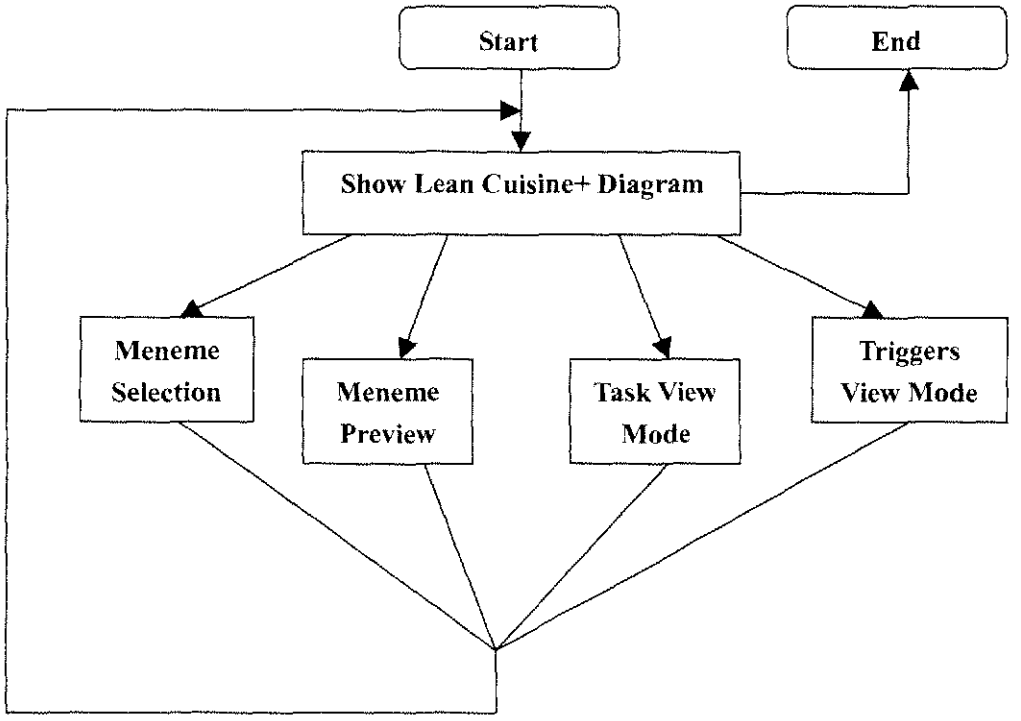


Figure 5-14: Main Structure of the Execution Environment

5.5 Summary

Two effective and reliable tools, Auto-Generation System and Execution Environment, have been developed successfully, and added to the SELCU environment. The programming languages have been selected to suit this project. Several key implementation issues have been discussed. Informal use in academic environments has led to several revisions and indicates that the system is useful and reasonably intuitive to use.

Chapter 6

CASE STUDIES

Two software tools have been developed to support the generation and execution of Lean Cuisine+ specifications. This chapter includes two practical case studies to illustrate the main functions in the generation and execution software.

Section 6.1 introduces the graphical user interface of case study one. The interface components and tasks are discussed here. It also describes how to use the auto-generation and execution software to generate a Lean Cuisine+ diagram and execute the notation. Another more complicated example is showed in section 6.2.

6.1 Case Study One: Timetabling System

The timetabling system is a simple interactive application (from Phillips and Scogings, 1998). The main goal of this system is to help students to design and print their timetable. In this system, the user can choose to show any combination of lectures times, tutorial times and laboratory times. The final timetable is then printed. For example, the student selects a semester and then selects a number of papers. As each paper is selected, the system includes the times for that paper in the timetable display. The student can select to display only lectures, only tutorials or only laboratories, or any combination of these. When the display is satisfactory, the student can print the timetable.

The graphical user interface of this example is shown in figure 6-1. It was developed using Delphi 6.0 IDE.

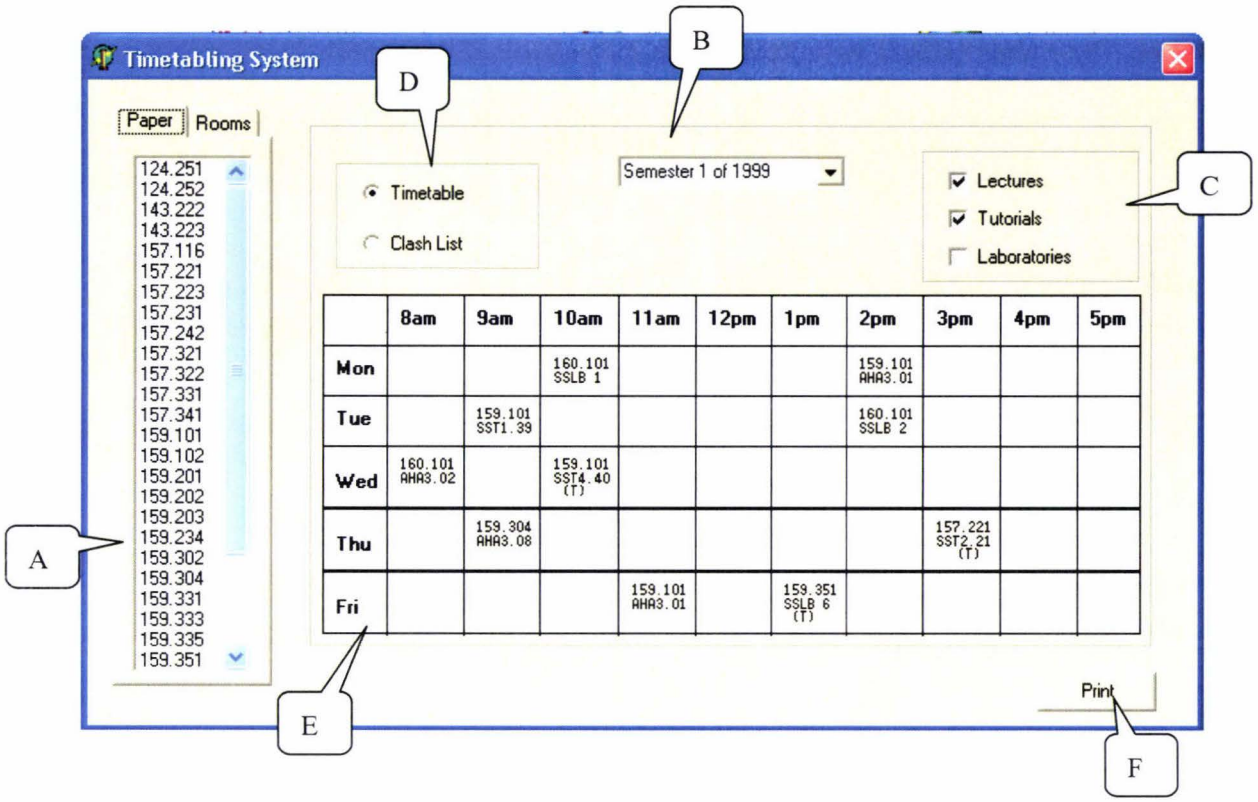


Figure 6-1: GUI of Timetabling System

6.1.1 Main Functions of the Timetabling System

This system allows users to select category options which include paper number and room number (A of Figure 6-1), semester information (B of Figure 6-1) and paper controls detail (C of Figure 6-1). More specifically:

- Only one of the category options can be selected at any time
- Only one of the room numbers can be selected at any time
- Several paper numbers can be selected at same time
- Only one of the semester information options can be selected at any time
- All the paper controls options can be selected at the same time. This is a required option, and the lecture is the default selection in paper control.

When the user selects a paper number, the system will automatically display the selected paper's information in the timetable (E of Figure 6-1). At the same time, users can switch between two display options (***List Clashes*** and ***Display Timetable***) from part D of Figure 6-1. The ***Display Timetable*** option is the default selection in display options. The users also can print the screen information at any time by clicking the ***Print*** button (F of Figure 6-1).

6.1.2 Main Interface Components in Timetabling System GUI

- Form Component

  This is the basic interface component for the whole system. All the other components are placed on this one. It can be selected and deselected.

- Page Control Component (A of figure 6-1)

  This component is used to present the categories information. It includes *Paper* and *Rooms* options. All the paper numbers and room numbers are selectable and de-selectable.

- Radio Group Component (D figure 6-1)

  Only one display option can be selected. The system uses a radio group component to present "List clashes" and "Display Timetable" options. All the options in this component can be selected and deselected.

- Combo Box Component (B figure 6-1)

  The combo box is used to present the semester information. In this component, only one option can be selected at anytime, and all the options are selectable and de-selectable.

- Check Box Component (C figure 6-1)

  The paper controls information is implemented using check boxes. Three options (Lectures, Tutorials and Laboratories) can be selected in any combination. They are all selectable and de-selectable.

- Image Component (E figure 6-1)

  This component is used to display the timetable. It is used to present the information, but it cannot be selected or deselected.

- Button Component (F figure 6-1)

  The system uses a button to trigger the print function. When the user clicks the button, it becomes selected. It will automatically go back to a deselected state on completion of the opereration. This component can be selected or deselected.

6.1.3 Using the Auto-Generation Software

The Auto-Generation Software is used to generate an initial Lean Cuisine+ diagram from a Delphi designed interface. The generated file will be SELCU compatible. The auto generation process includes two main steps. In step one, the software reads select *.pas and *.dfm Delphi source files, and analyzes them in the memory. In step two, the software will pick the analysis results from memory and integrate them into a SELCU compatible source file. The system lets users read the final generation file immediately, and it also allows users to directly call the SELCU application to translate the file into a graphical Lean Cuisine+ diagram.

Generation Progress

The generation process is described in detail in Appendix A1. With reference to figure 6-2, the process involves user manipulation of the first three buttons on the left-hand side of the Auto Generation window, from top to bottom. The first two buttons are used to locate the .pas and .dfm source files, and the generated SELCU compatible file appears in the main window.

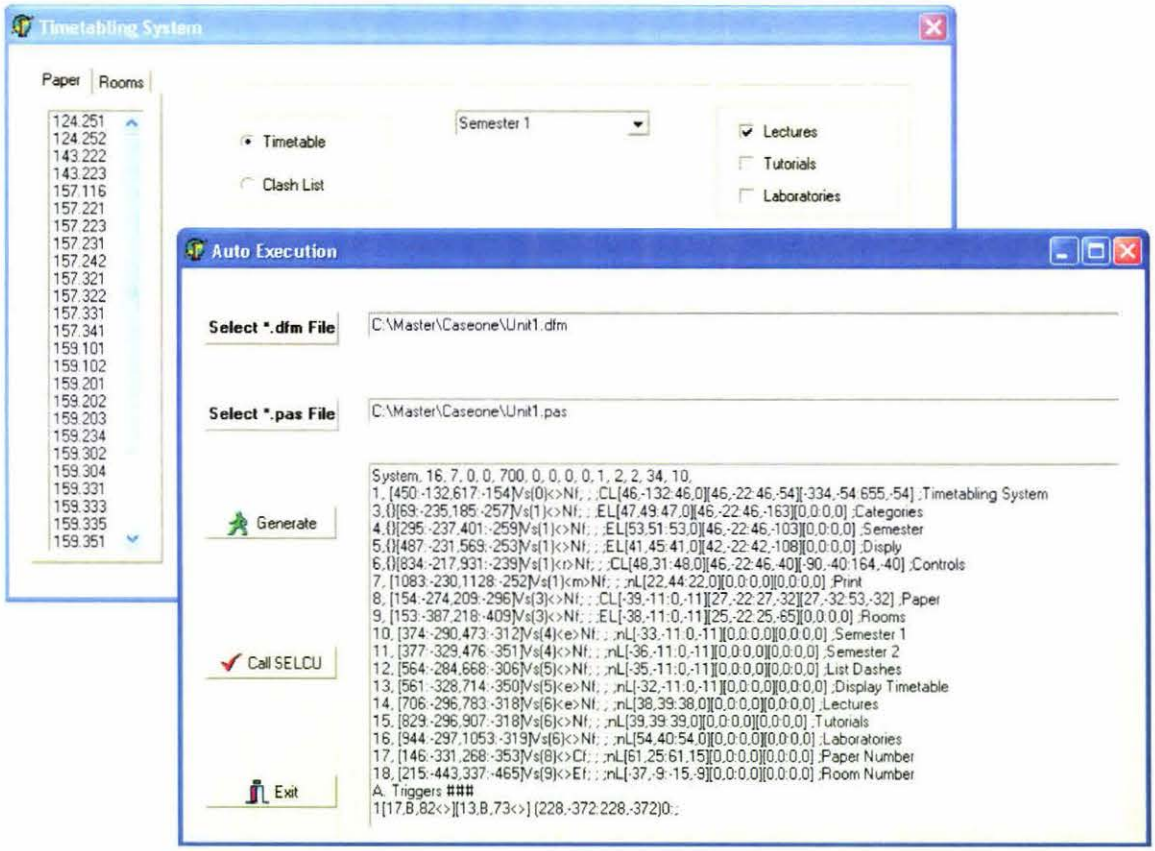


Figure 6-2: Generation of the Lean Cuisine+ Source File

Discussion of Auto-Generation

After the Auto-Generation software has produced a new Lean Cuisine+ file, the user can simple click the ***Call SELCU*** button in figure 6-5 to switch to the SELCU application, which will display the Lean Cuisine+ diagram (as shown in figure 6-3).

In figure 6-3, the ***Timetabling System*** has five mutually compatible sub-dialogues ({Categories}, {Semester}, {Display}, {Controls} and {Print}). Only the meneme "Print" is directly selectable by the user. The other four sub-dialogues are virtual menemes, which are mechanisms for grouping other user selectable options.

- {Categories} includes two mutually exclusive menemes, which are ***Paper*** and ***Room***. The vertical fork from ***Paper*** to ***Paper Number*** means that one or more instances of ***Paper Number*** can be selected at any time. The horizontal fork from ***Rooms*** to ***Room Number*** presents a set of mutually exclusive ***Room Number*** options.

- {Semester} is used to head two selectable and mutually exclusive menemes ***Semester 1*** and ***Semester 2***.

- {Display} is designed to group two mutually exclusive menemes ***List Clashes*** and ***Display Timetable***.

- {Controls} provides access to three mutually compatible presentation options (which are ***Lectures, Tutorials*** and ***Laboratories***).

- ***Print*** is a real meneme, which is mutually compatible with the other four virtual menemes.

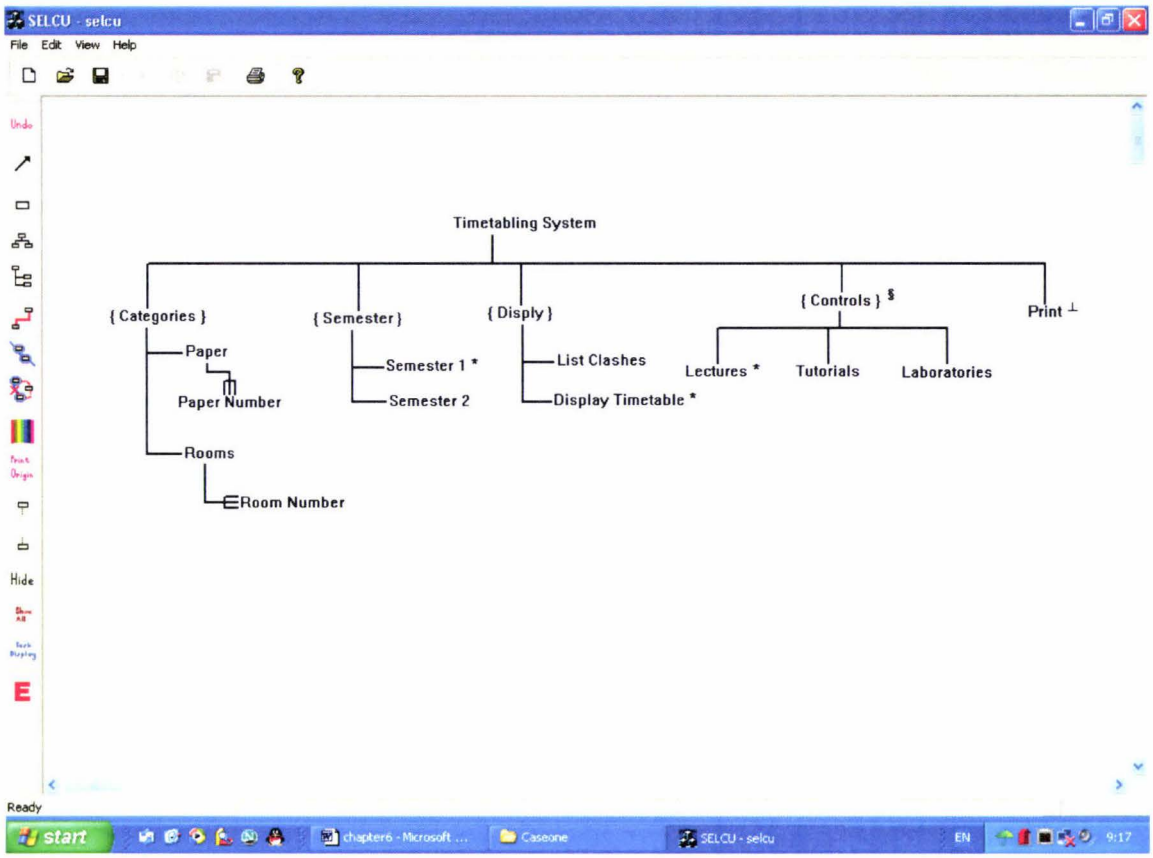


Figure 6-3: The Generated Lean Cuisine+ Diagram for Case Study One

Three ***meneme modifier symbols*** appear in figure 6-3:

- ***{Controls}*** is shown as a required choice group (§).

- ***Semester 1, Display Timetable*** and ***Lectures*** are the default selected options (*) in {Semester} group, {Display} group and {Controls} group respectively.

- ***Print*** is shown as a monostable meneme (⊥). Following selection, it will return to an unselected state on completion of the print operation.

If the user selects any paper number, the system will automatically display this paper's timetable information. This means there is a trigger between the ***Paper Number*** meneme and ***Display Timetable*** menemes. When the user selects ***Paper***

Number, the ***Display Timetable*** meneme will be automatically selected as well.

During the Auto-Generation process, this trigger will also be taken into account. The final generated Lean Cuisine+ diagram includes triggers. The user can see the Lean Cuisine+ notation with triggers when they switch to the ***triggers view*** mode in SELCU (click ***Construct Trigger*** button). The system interface appears as in figure 6-4.

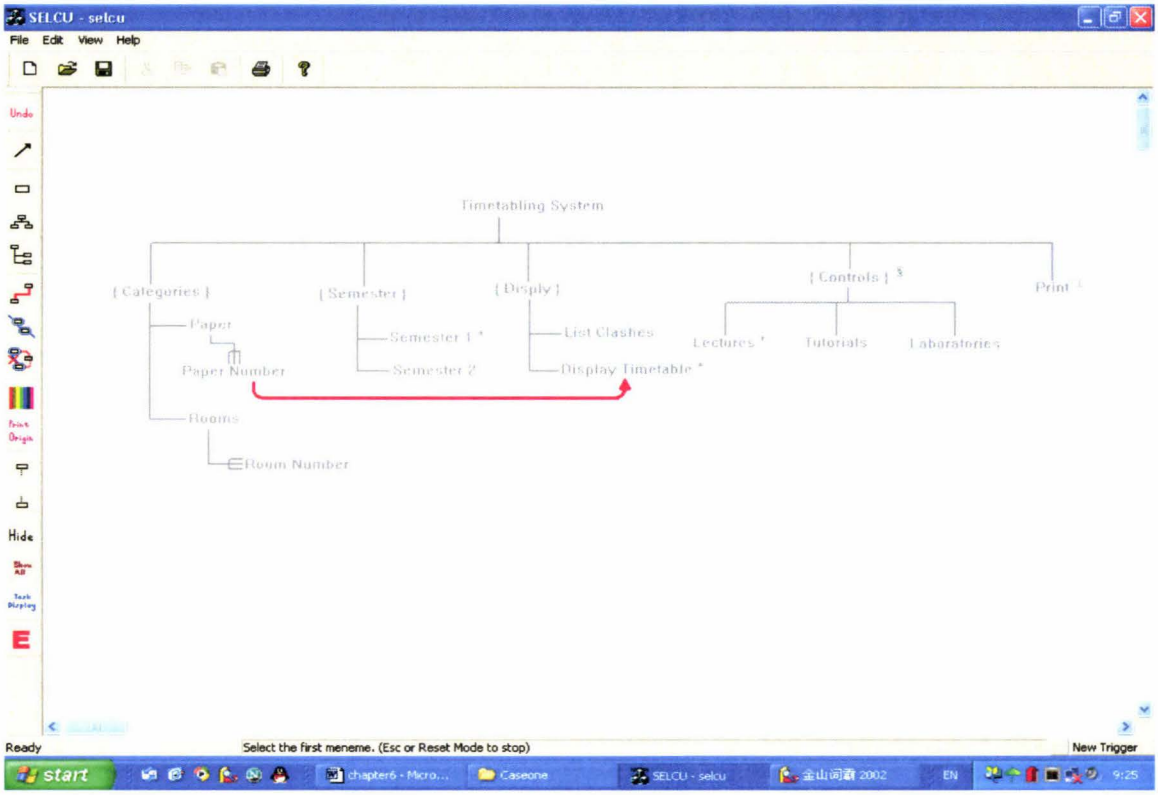


Figure 6-4: Lean Cuisine+ Diagram with Trigger Displayed

Part E of the Timetabling system GUI (which is an image interface component) in figure 6-1 is not represented in the Lean Cuisine+ notation. Although this component appears in the system interface, it cannot be selected or deselected. The main function of this component is to display the timetable or clashes. The Auto-Generation software ignores this interface component.

6.1.4 Using the Execution Environment

The execution environment is an application embedded inside SELCU. It allows the user to select or deselect any available menemes in Lean Cuisine+ diagram; supports the execution of tasks represented in notation and it also takes trigger information into account to provide useful feedback for the designer at an early stage of system development. Using this environment, the user can execute tasks event by event, while observing the results on meneme states.

Switching to Execution Mode

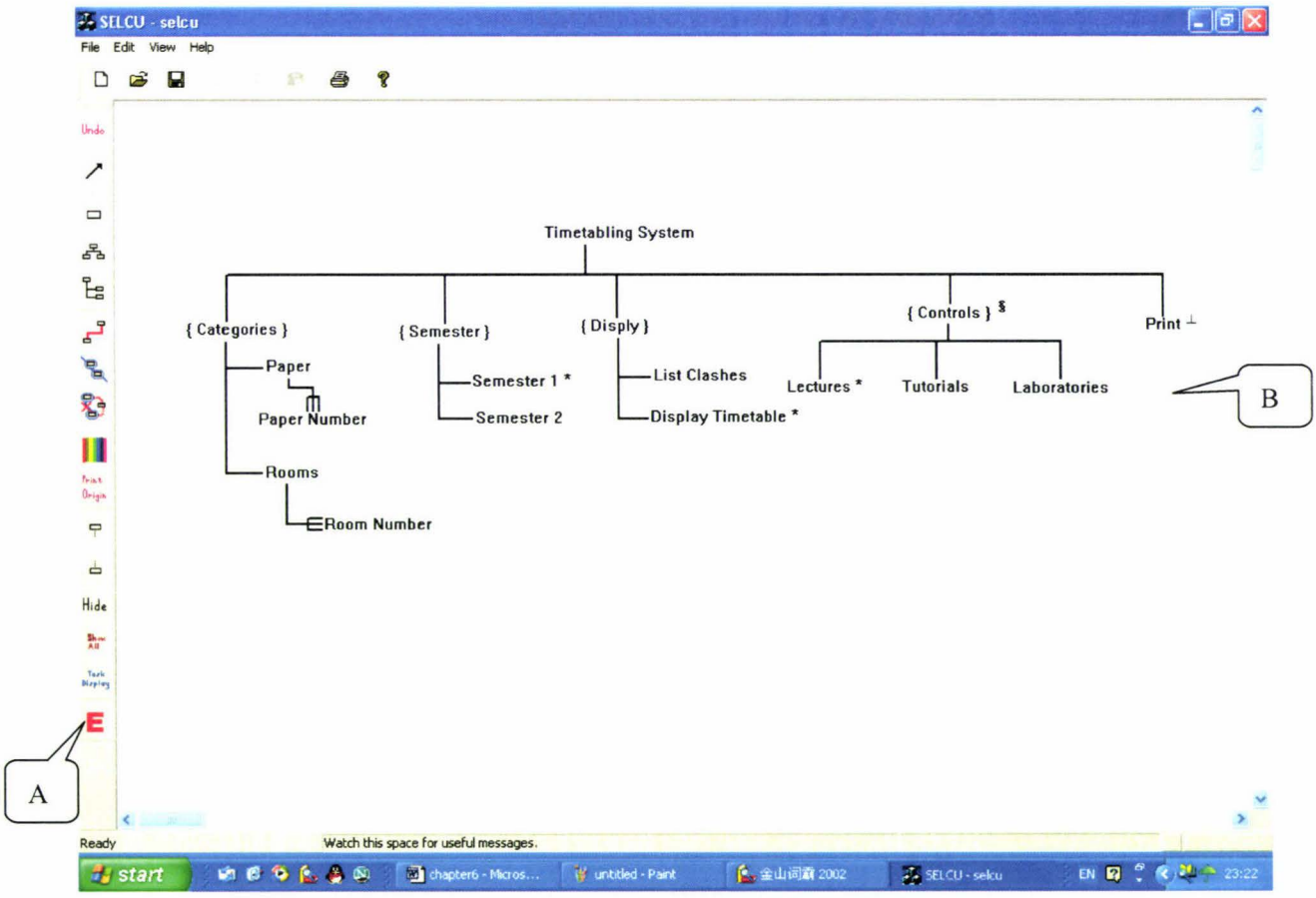


Figure 6-5: A Lean Cuisine+ Diagram in the SELCU System

A detailed description of SELCU is provided in section 3.2. In figure 6-5, the Lean Cuisine+ diagram is displayed at B, and the button that allows the user to

switch to execution mode appears at A. At this stage:

- The user can edit the diagram using standard SELCU operations, or
- The user can click button "E" in part A to change to execution mode

When the system changes to the execution mode, most of the function buttons on the task bar will become unavailable. Only the ***Select Object*** button is still available, the user can click this button to return to the SELCU edit mode. After clicking the "E" button, the system layout appears as shown in figure 6-4.

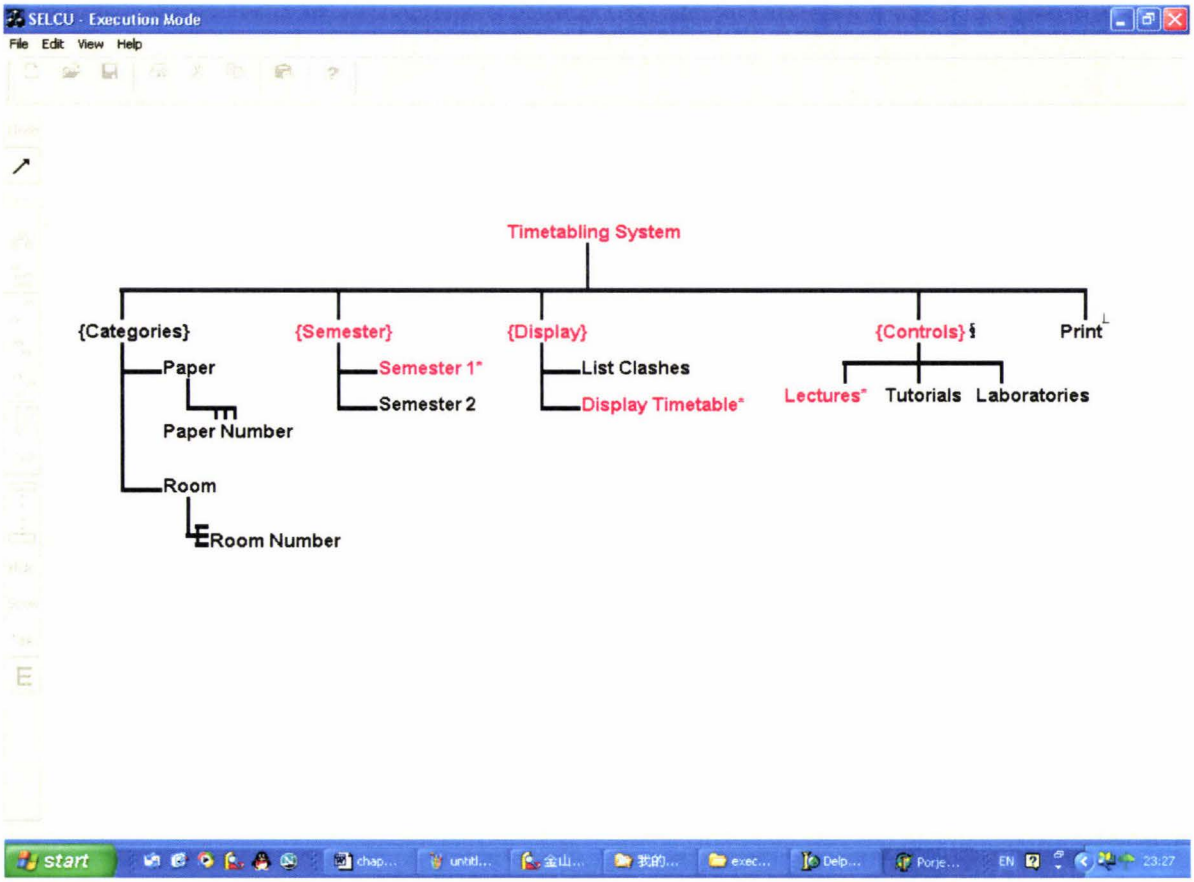


Figure 6-6: Execution Environment for Case Study One

Preview Meneme Selection

Before selecting any meneme, the user can right press the mouse to preview the effect of selecting it. For example, if the user wants to preview the selection of the ***Semester 2*** meneme in figure 6-6, they just need to move mouse to the ***Semester 2*** meneme and right press the button. Following this, the Lean Cuisine+ diagram looks like figure 6-7.

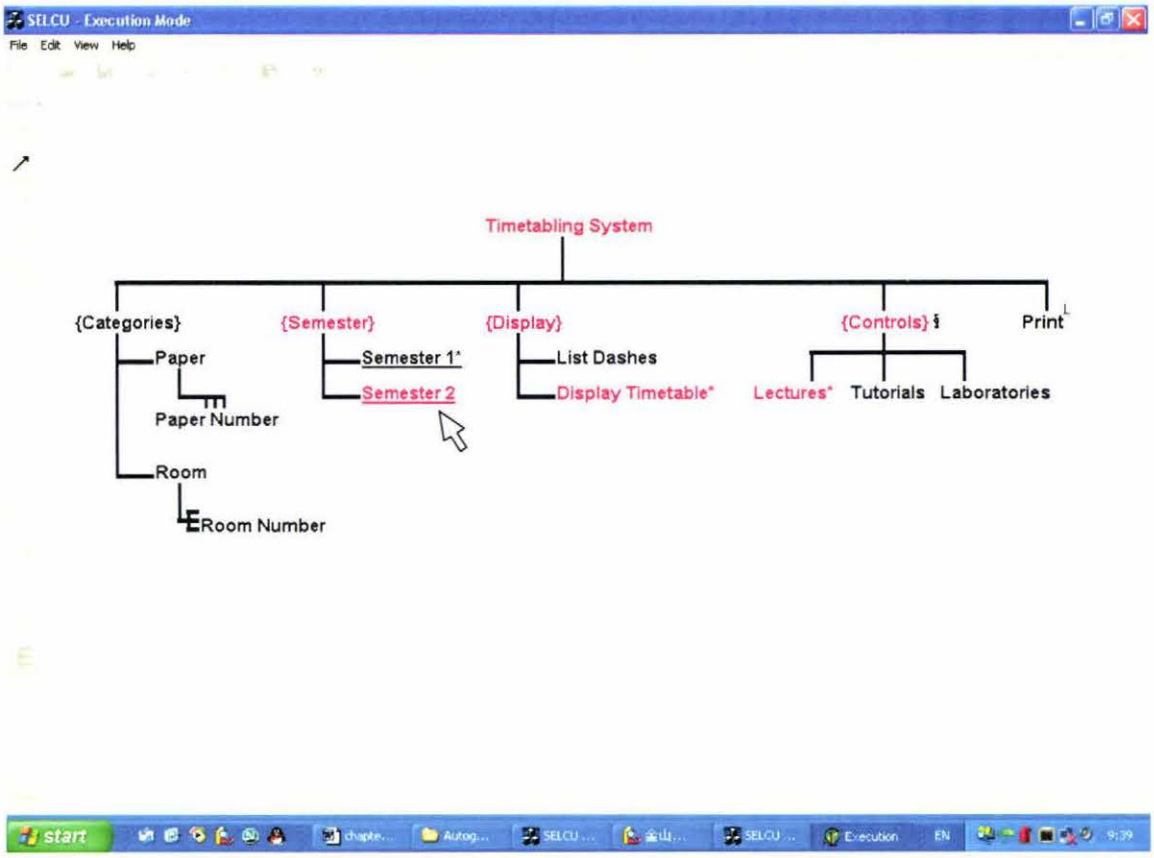


Figure 6-7: Preview for Selection of "Semester 2" Meneme

In figure 6-7, the underlined meneme names would be changed by the selection of ***Semester 2***. Specifically:

- The ***Semester 2*** meneme name changes from black to red with underline
- The ***Semester 1*** meneme name changes from red to black with underline

The above analysis implies that if the user selects ***Semester 2***, the meneme ***Semester 2*** will switch to a selected state, the meneme ***Semester 1*** will switch to an unselected state and all the other menemes will be unchanged.

The preview is a temporary state. If the user releases the right button, the system will revert to the original state without any change. For example, if the user releases the right button in figure 6-7, the Lean Cuisine+ diagram will return to the state shown in figure 6-8 (which is the same as figure 6-6).

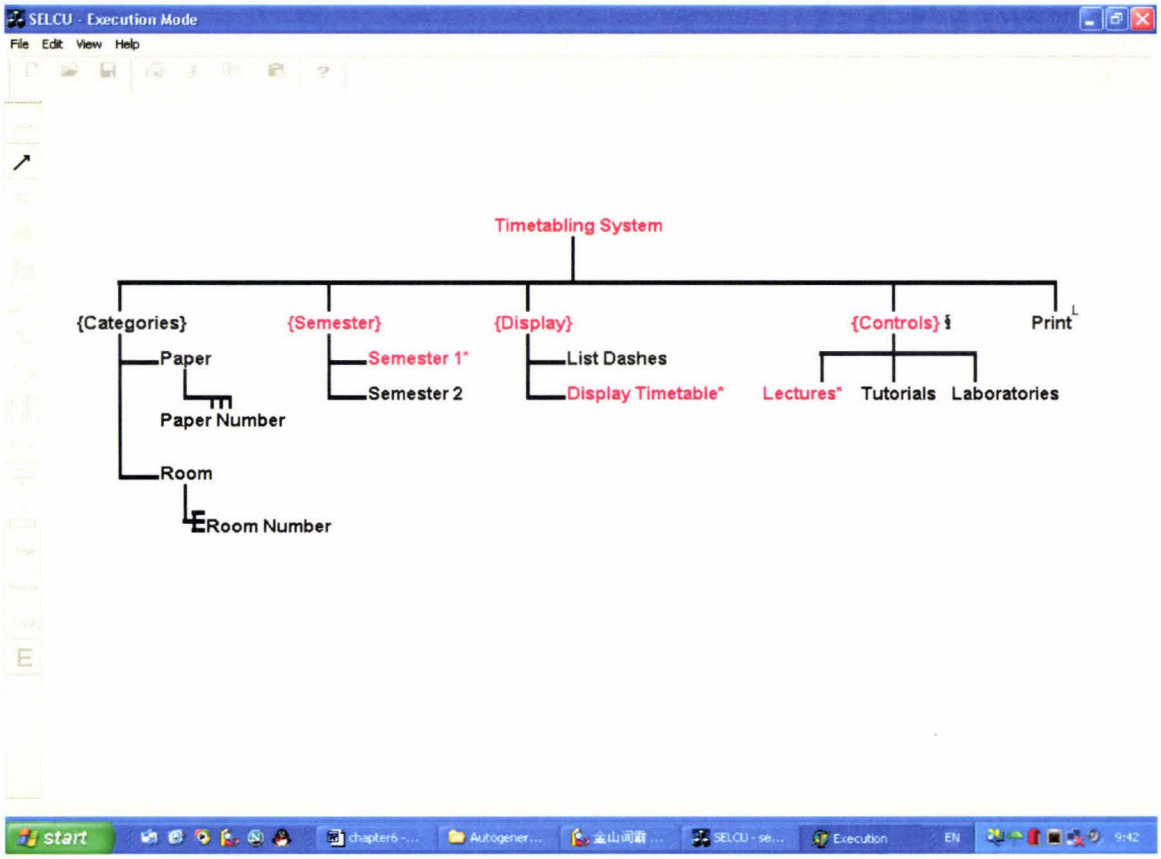


Figure 6-8: The Lean Cuisine+ Diagram after Release of the Right Button

Select Meneme

The user can select a meneme by left clicking the mouse. In this case, the state of the diagram changes to reflect the selection. Figure 6-9 shows the meneme states after left clicking ***List Clashes***.

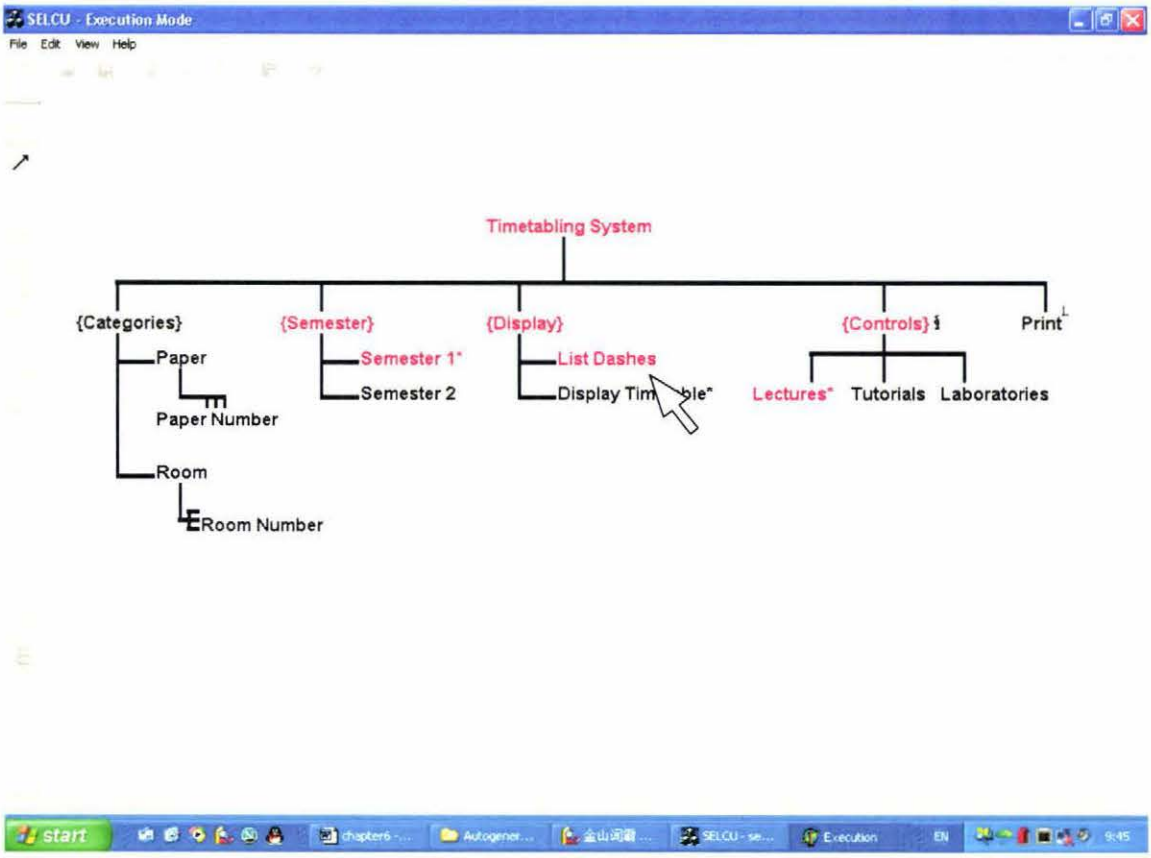


Figure 6-9: Diagram After Left Clicking (Selecting) "List Clashes" Meneme

Comparing figure 6-8 (before selecting) and figure 6-9 (after selecting), the main changes are:

- The ***List Clashes*** name changes from black to red. It means this meneme changes from an unselected state to a selected state.
- The ***Display Timetable*** meneme changes from red (selected state) to black (unselected state).
- All the other menemes stay in original states.

In this case study, a simple example has been used to present auto-generation and some execution functions (which includes switching to execution mode, previewing meneme selection and selecting a meneme). More complicated execution functions will be described in case study two.

6.2 Case Study Two: Library Catalogue System

The library catalogue system is a more complex example from (Phillips and Kemp, 2002). The ***request book*** process from this system is used to demonstrate the generation and execution functions in action. The four possible meneme states ("available & selected", "available & not selected", "not available & selected" and "not available & not selected") all appear in this case study. More complex execution functions such as the execution of tasks, the selection of menemes with special attributes and execution functions which take account of triggers all appear in this case study.

6.2.1 Library Catalogue System GUI Description

The ***Request Book*** process of the ***Library Catalogue System*** has been developed in Delphi. This system is used to permit a library user to request a book. In this system, book details are displayed (F of figure 6-10). At any time, the library user can request a printout of book details (E of figure 6-10), or cancel any request (D of figure 6-10). If the user wants to request a book, they need to select a copy, enter a last date (C of figure 6-10) and submit the request (G of figure 6-10). Then, the system will confirm the request and inform the library user to check their lending record twice weekly to see whether the book is available (figure 6-10). The main graphical user interface is shown in figure 6-10. It shows the book search result screen for the users.

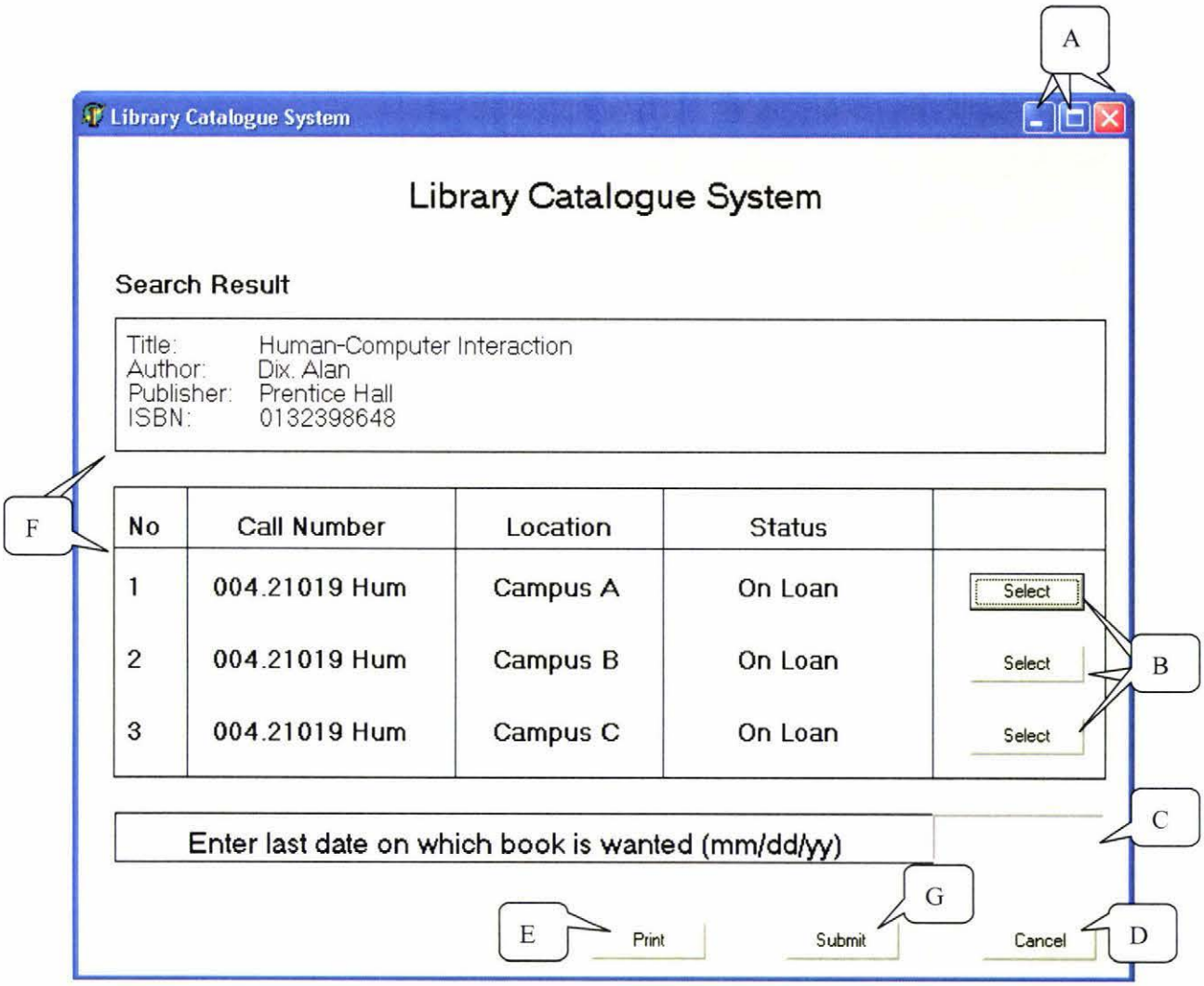


Figure 6-10: GUI of Library Catalogue System (Before Submit Request)

In figure 6-10:

- A is the window attributes button group. It includes the minimum, maximum and close window functions.
- B is a group of "select" buttons. Every book has one "Select" button. So the number of these buttons depends on the search result of books.
- C is an input area. The last date is typed here.
- D is a cancel function button. The user clicks this button to cancel the book request requirement and close the library catalogue system.
- E is a print button. It is used to print the screen information.
- F is a view area. All the book search results appear in this area

- G is a submit button. The user selects this button to submit the book request. A feedback window will pop up when the user submits the request information. Following this, the system interface appears as in figure 6-11.

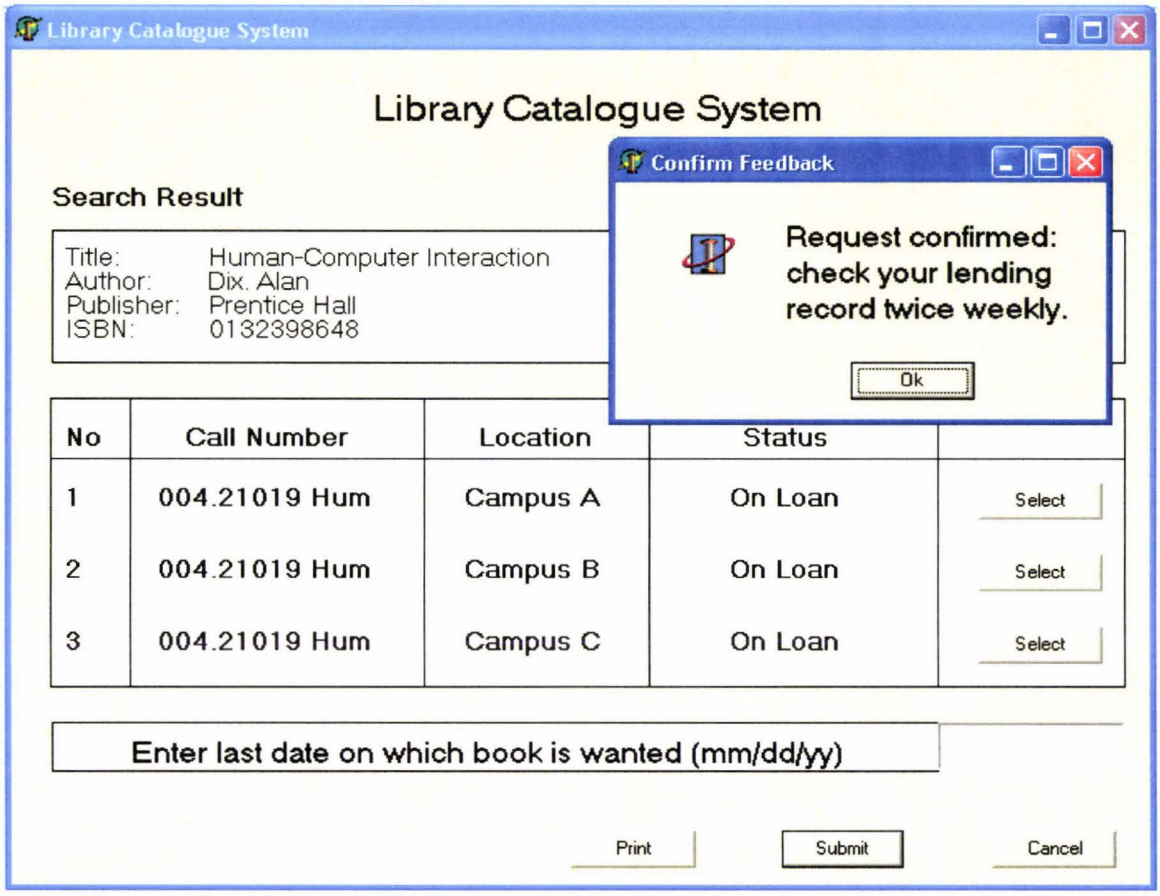


Figure 6-11: Library Catalogue System GUI (After Submit Request)

In figure 6-11:

- The user still can see the ***Library Catalogue System*** window, but it is not active until the ***Confirm Feedback*** window is closed.
- The ***Confirm Feedback*** window does not appear until the user clicks the ***submit*** button in the ***Library Catalogue System*** window.
- The ***OK*** button is used to close the ***Confirm Feedback*** window.
- The ***Confirm Feedback*** window also has attribute buttons (minimum, maximum and close functions).

6.2.2 Using the Auto-Generation Software

The GUI and main functions of the Auto-Generation Software are described in detail in Appendix A1. After the "Auto-Generation Software" has produced the final file, a graphical Lean Cuisine+ diagram can be displayed in SELCU. This diagram is shown in figure 6-12. In this diagram:

- The ***Library System*** meneme is provided by the software. It is used to head two mutually exclusive window menemes (***Library Catalogue System*** and ***Confirm Feedback***). It is a required choice group (§).

- The ***Library Catalogue System*** is a real and default meneme in the diagram. It includes three mutually compatible sub-dialogues {Buttons}, {Select Group} and EditBox.

- The {Buttons} virtual meneme is used to group three mutually exclusive function buttons (Print, Cancel and Submit), plus a group of mutually exclusive {form attributes}.

- Although the ***Move*** and ***Change Size*** attributes do not appear explicitly on the system interface, they are required as sub-dialogues in the {Form Attributes} group to represent available actions. They are mutually exclusive with other form attributes menemes.

- The {Select Group} allows the user to access numbers of mutually compatible options (via ***Select*** menemes). The number of ***Select*** menemes depends on the number of search results. A vertical fork from {Select Group} to ***Select*** meneme implies one or more instances of the ***Select*** button can be selected at any time.

- The ***EditBox*** meneme includes three mutually exclusive input menemes (insert, delete and modify).

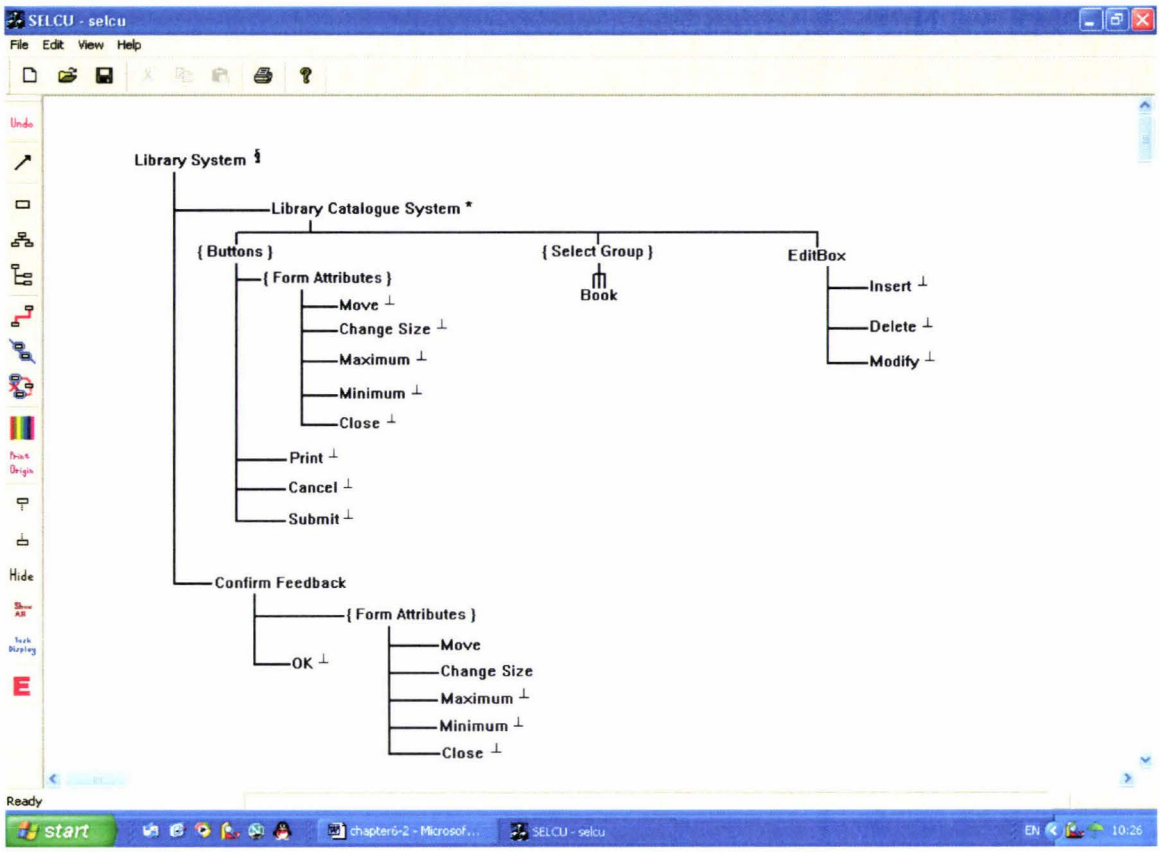


Figure 6-12: Lean Cuisine+ Diagram of Library Catalogue System

In the above figure, all the input menemes and button menemes are monostable, which means they will automatically revert to an unselected state on completion of the operation they represent.

Some of the library catalogue system functions are translated into triggers in Lean Cuisine+. The triggers view diagram is shown in figure 6-13.

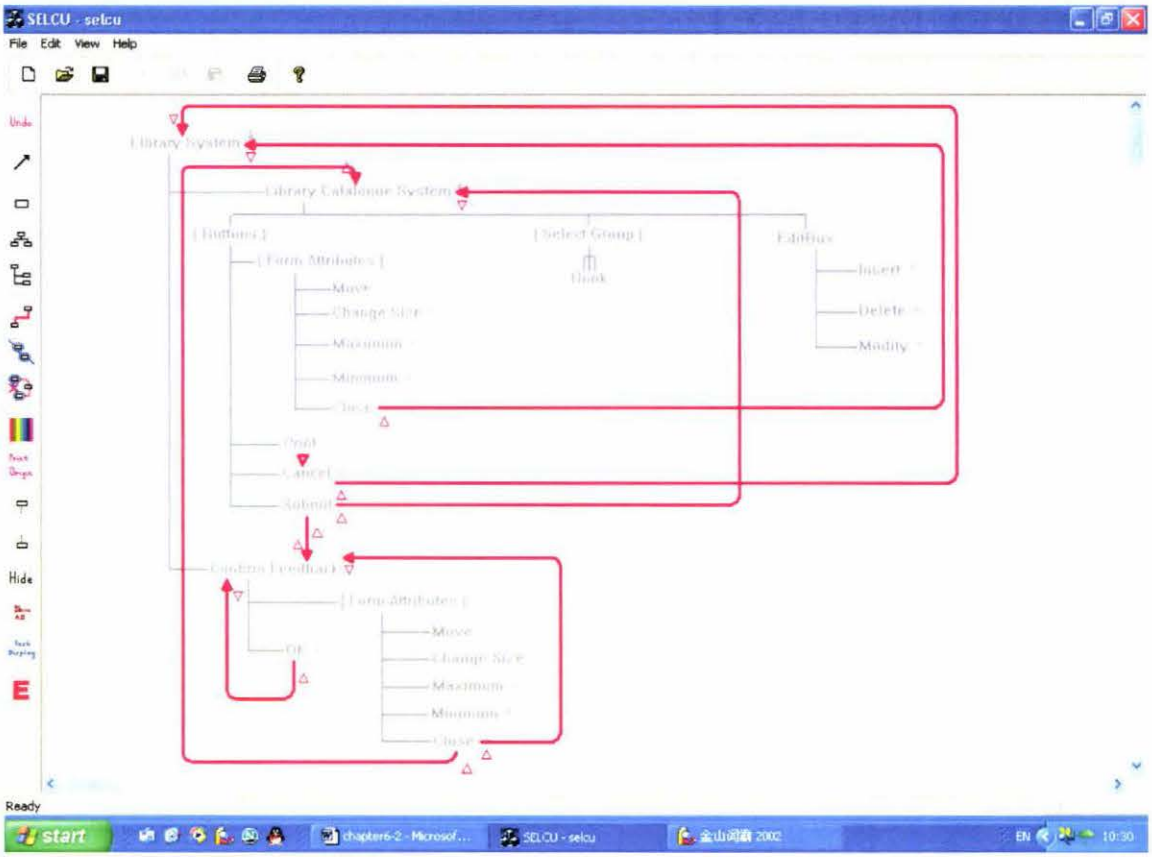


Figure 6-13: Lean Cuisine+ Diagram with Triggers for Library System

Figure 6-13 presents several system actions are represented using triggers:

- If the ***Close*** meneme in ***Library Catalogue System*** window is selected, the trigger will automatically switch off the whole library system.
- If the ***Cancel*** meneme is selected, the trigger will automatically switch off the whole library system.
- If the ***Submit*** meneme is selected, the trigger will switch off the ***Library Catalogue System***, and switch on the ***Confirm Feedback***.
- If the ***OK*** meneme is selected, the trigger will switch off the ***Confirm Feedback*** meneme, and switch on the ***Library Catalogue System*** meneme.
- If the ***Close*** meneme in the ***Confirm Feedback*** window is selected, the trigger will switch off the ***Confirm Feedback*** meneme, and switch on the ***Library Catalogue System*** meneme.

The system also supports the user tasks defined in this Lean Cuisine+ diagram. Figure 6-14 shows an example task. The objective of this task is submitting a book request.

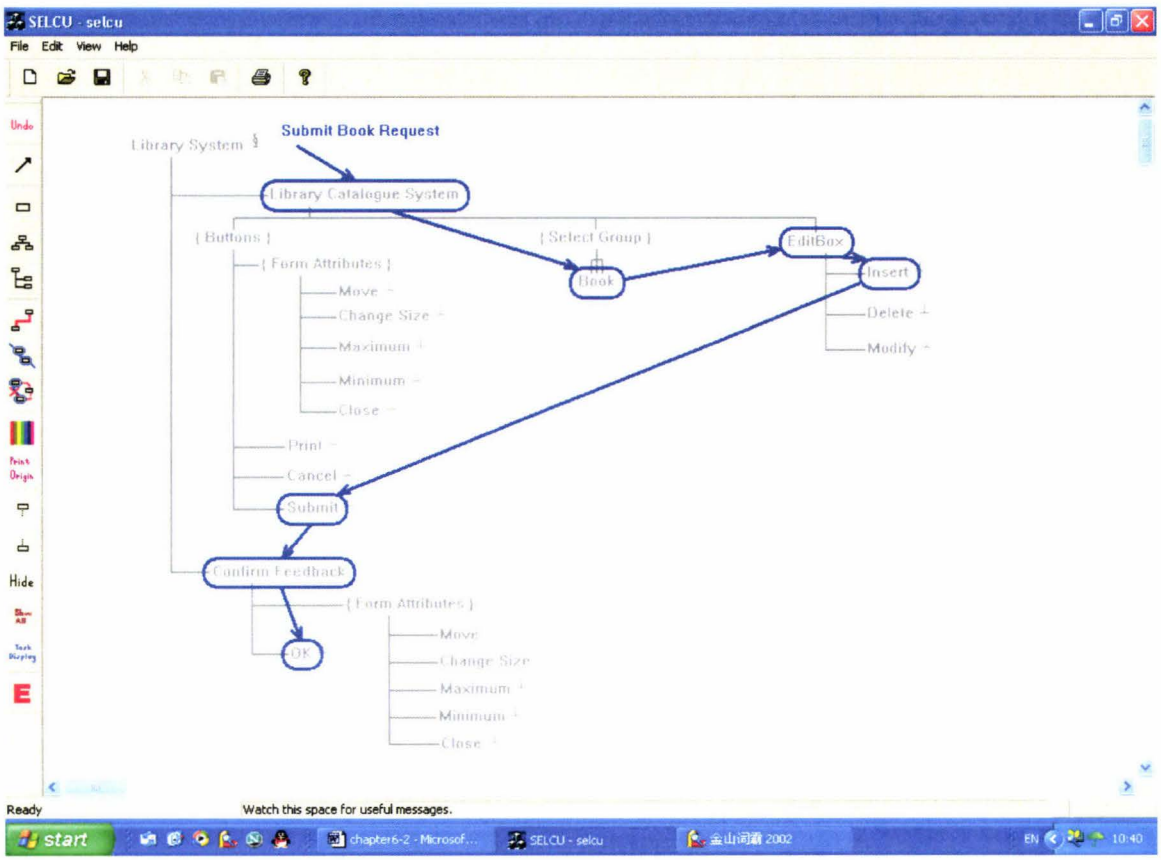


Figure 6-14: Lean Cuisine+ Diagram with Submit Book Request Task

In figure 6-14, the ***submit book request*** task process involves the following steps:

(1) Select ***Library Catalogue System*** window

(2) Select one or more books

(3) Move mouse to edit box

(4) Insert the date

(5) Click the ***submit*** button

(6) Pop up the ***Confirm Feedback*** window

(7) Click ***OK*** button

6.2.3 Using the Execution Environment

The basic execution environment was described in section 6.3. This part of the report will present some new and more complicated execution sequences from case study two.

After the user changes to the execution mode, the system layout of the Lean Cuisine+ diagram appears as in figure 6-15.

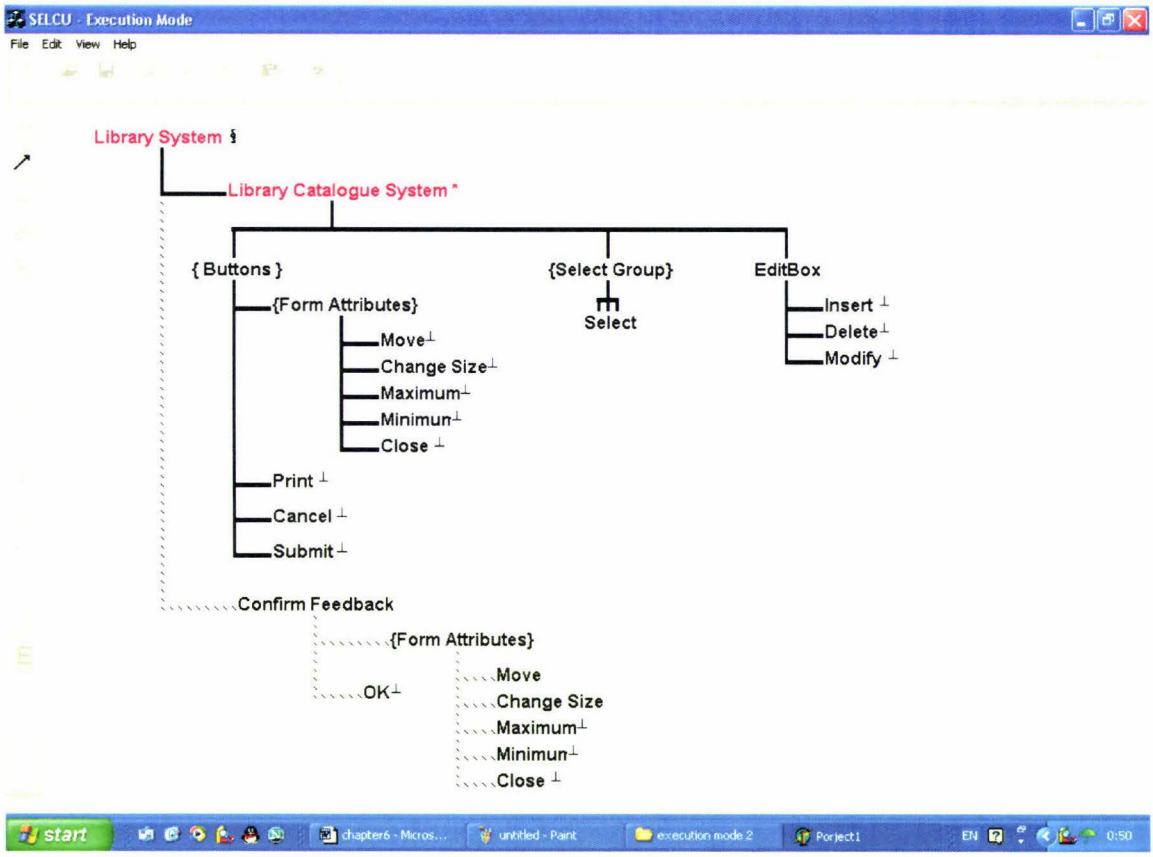


Figure 6-15: Lean Cuisine+ Diagram in Execution Mode

In this mode:

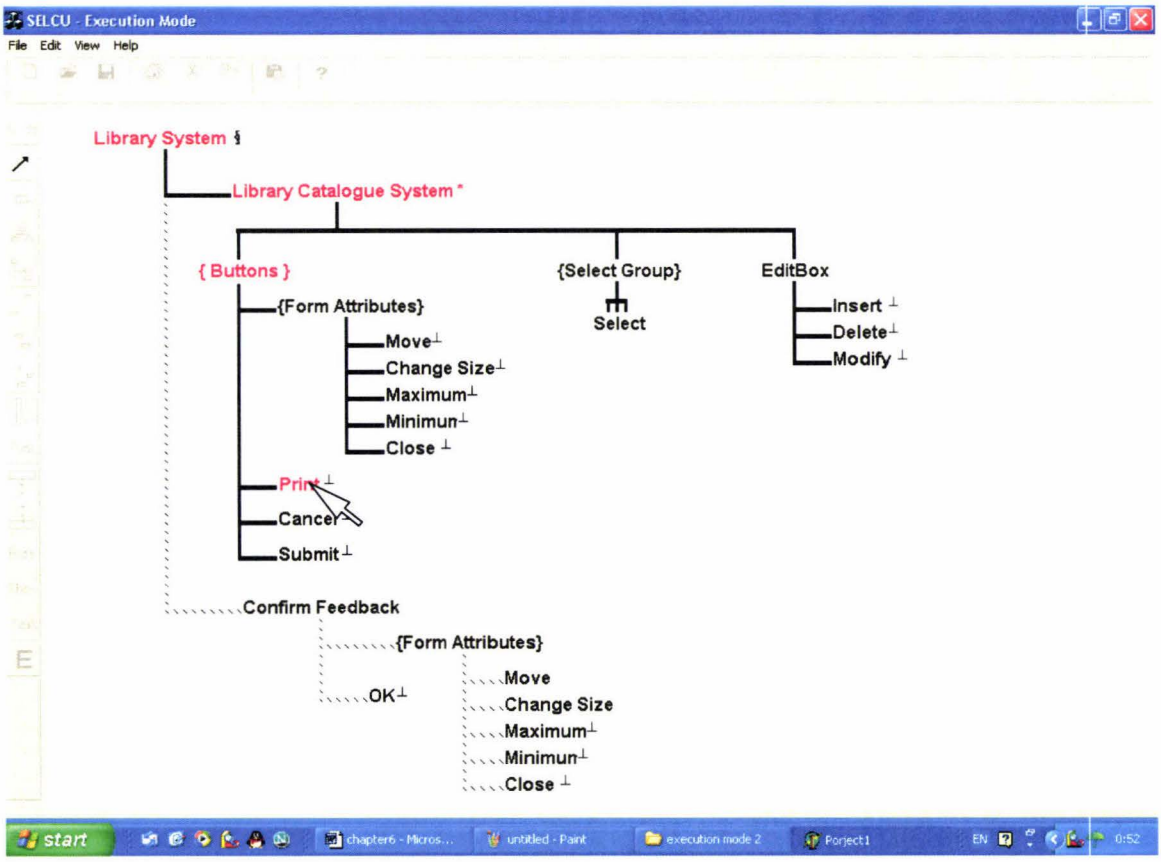
- Some meneme names are black. These menemes are not selected (e.g. ***move*** and ***change size***).
- Some meneme names are red. These menemes are selected (e.g.

library system and ***library catalogue system***).

- Some lines are solid. The menemes attached to these lines are available (e.g. ***print*** and ***cancel***).
- Some lines are dashed. The menemes attached to these lines are not available (e.g. ***confirm feedback*** and ***OK***).
- Menemes cannot be modified, deleted or added in execution mode.

Selecting Monostable Menemes

In execution mode, a selected monostable meneme will remain in a selected state for three seconds (red colour name), and then three seconds later, will revert to an unselected state. Figure 6-16 presents the diagram interface when the user clicks the monostable meneme ***Print***. After three seconds, the system layout changes back to figure 6-15.



Finger 6-16: Selecting a Monostable Meneme (First Three Seconds)

Using Triggers in the Execution Environment

The execution environment also includes trigger related functions. There are three situations which are significant:

- **Previewing a Meneme with Triggers**

  When the user previews a meneme that includes triggers, the system will present the triggers on screen. For example, if a user wants to preview ***cancel*** in the ***Library Catalogue System*** window, the software interface will appear as in figure 6-17.

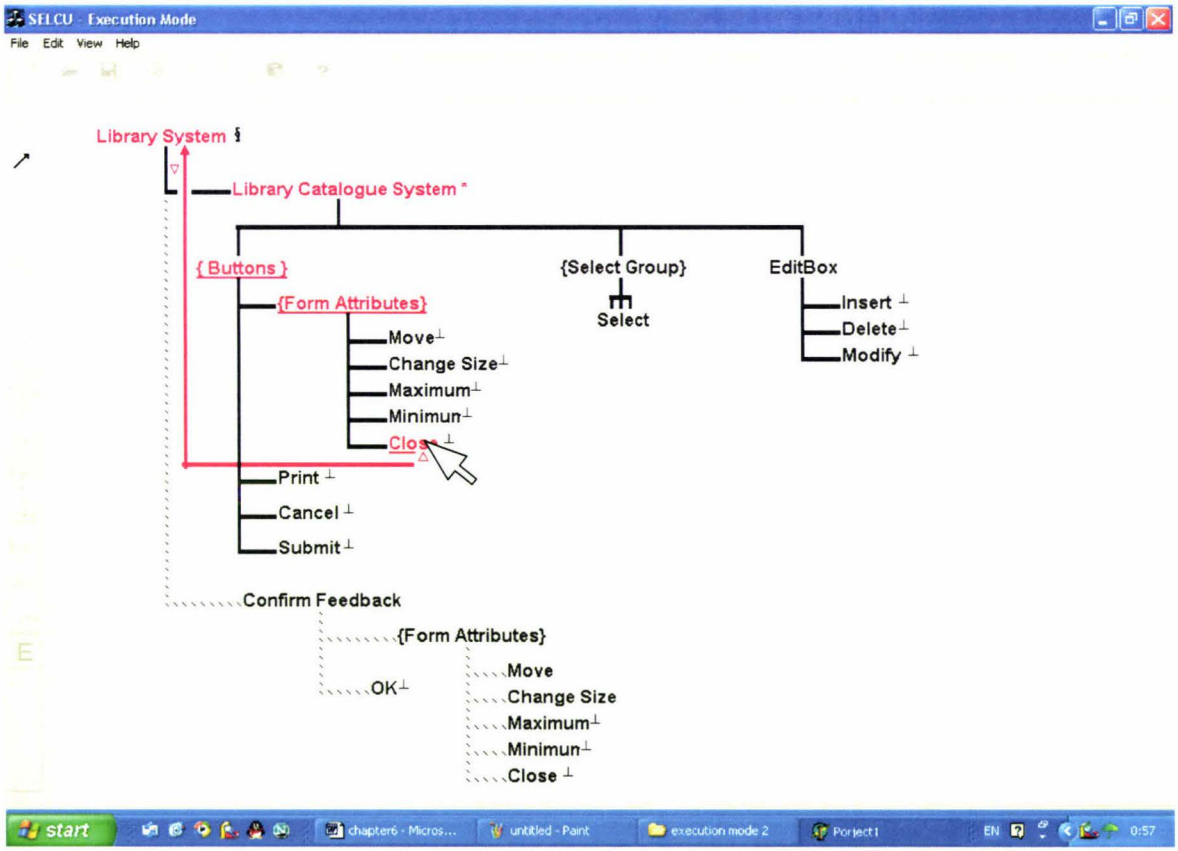


Figure 6-17: Meneme Preview Showing Trigger

In figure 6-17, when the user presses the right mouse button:

- ***close, {form attributes}*** and ***{buttons}*** change from black to red.
- ***close, {form attributes}*** and ***{buttons}*** all have underlines

- A trigger appears from ***close*** to the ***Library System*** meneme

If the user releases the right button, the trigger will disappear, and the system layout will go back to figure 6-15.

- **Selecting a Meneme with Triggers**

Continuing the above example, if the user wants to the select ***close*** meneme, they move the mouse to the ***close*** meneme and left click the mouse. The system will deal first with selection propagation (***close, {form attributes}*** and ***{buttons}*** meneme become selected), and then triggers will be activated (trigger between ***close*** and ***Library System*** meneme). Figure 6-18 shows the selections, and figure 6-19 shows the interface after the trigger has been activated.

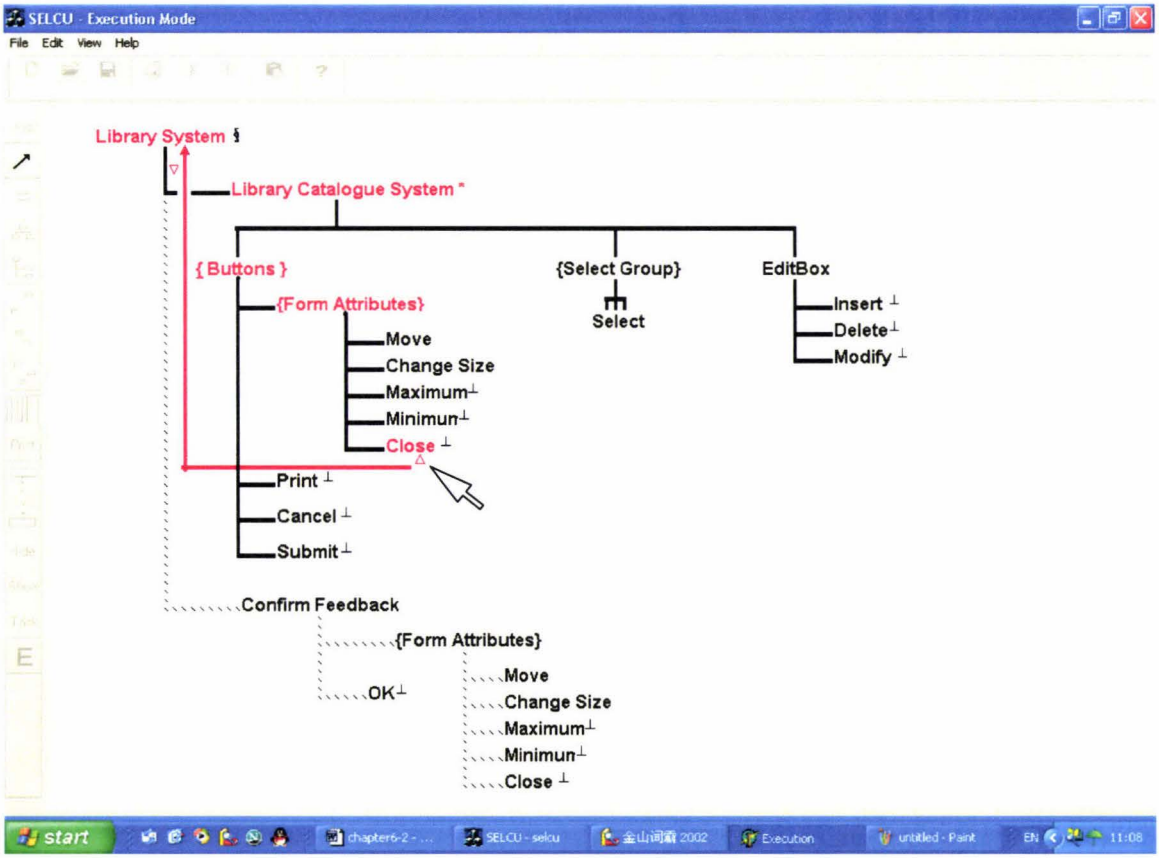


Figure 6-18: Selection Propagation

In figure 6-18,

- ***Close*** is a monostable meneme, so when it is selected, its name only stays in red for three seconds.
- Three seconds later, the system will process triggers. (The screen layout will change to figure 6-20).
- The trigger will switch off the ***Library System*** meneme. As the ***Library system*** meneme is the head of the whole Lean Cuisine+ diagram, the whole system will close after the trigger has been processed.

In figure 6-19, all the linking lines become dashed, this means that all the menemes in the diagram are now unavailable.

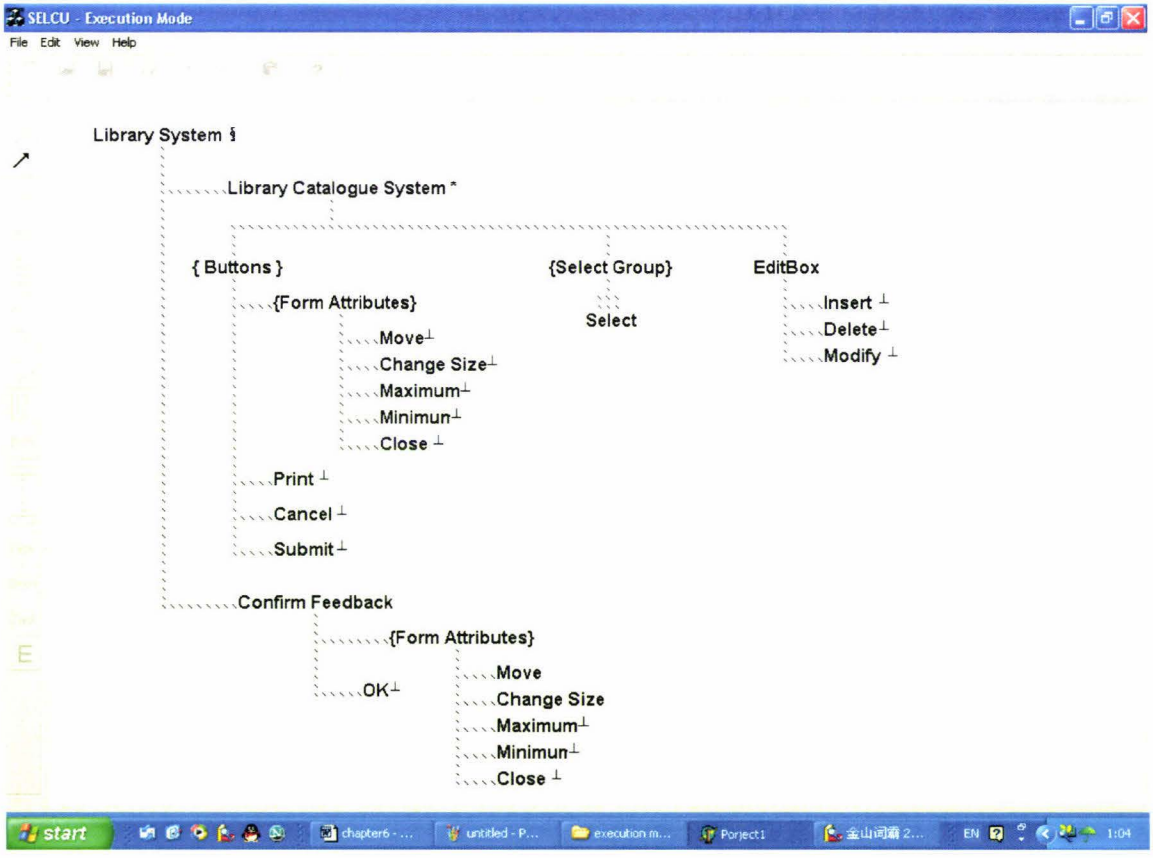


Figure 6-19: System Interface (After Clicking the "Close" Meneme)

- **Trigger View of Lean Cuisine+ Diagram in Execution Environment**

The execution environment also supports a triggers view of the Lean Cuisine+ diagram. The user needs to call this function from the menu at the top of the interface. Figure 6-20 shows how to call the triggers view function. After the user selects the triggers view function from the system menu, the software will present all the triggers on the screen as shown in figure 6-21.

Figure 6-21 (in execution mode) is very similar to figure 6-13 (in the SELCU edit mode). However, in execution mode the user cannot modify the triggers or the Lean Cuisine+ diagram,

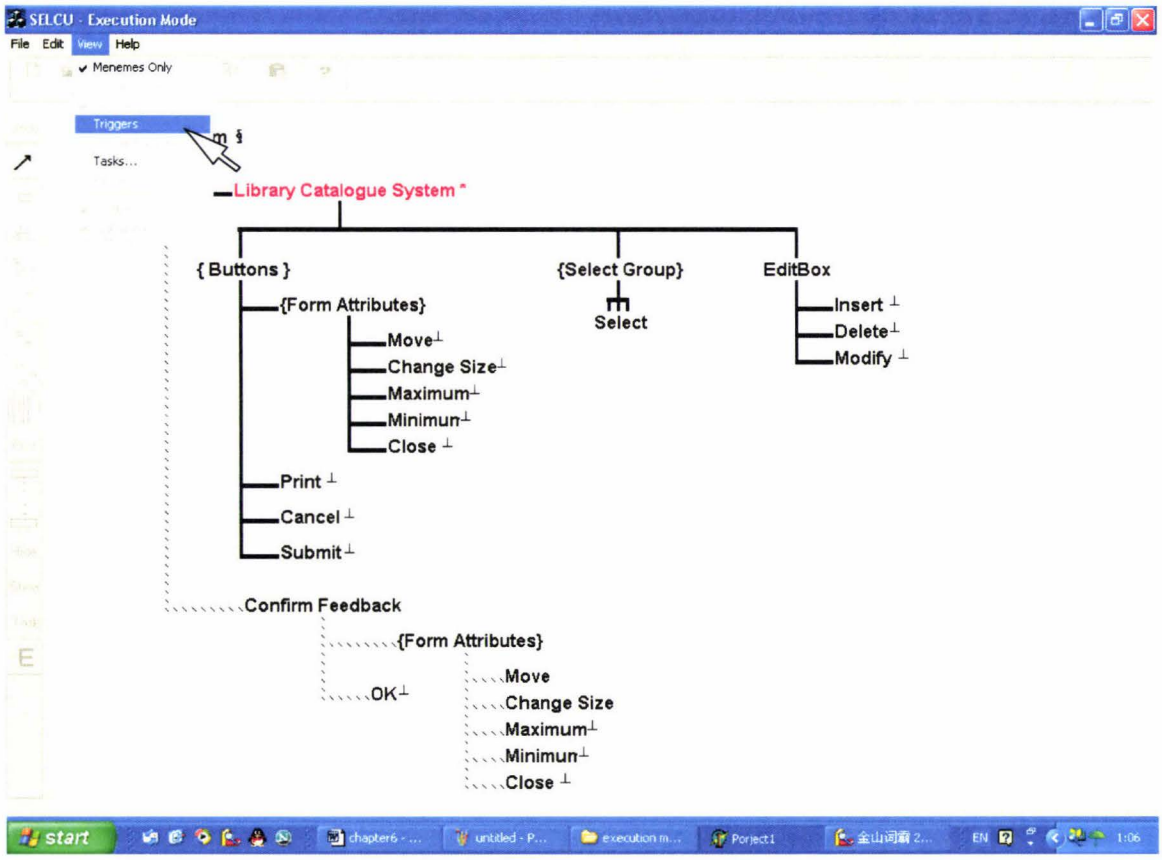


Figure 6-20: Selecting Triggers Functions from Menu

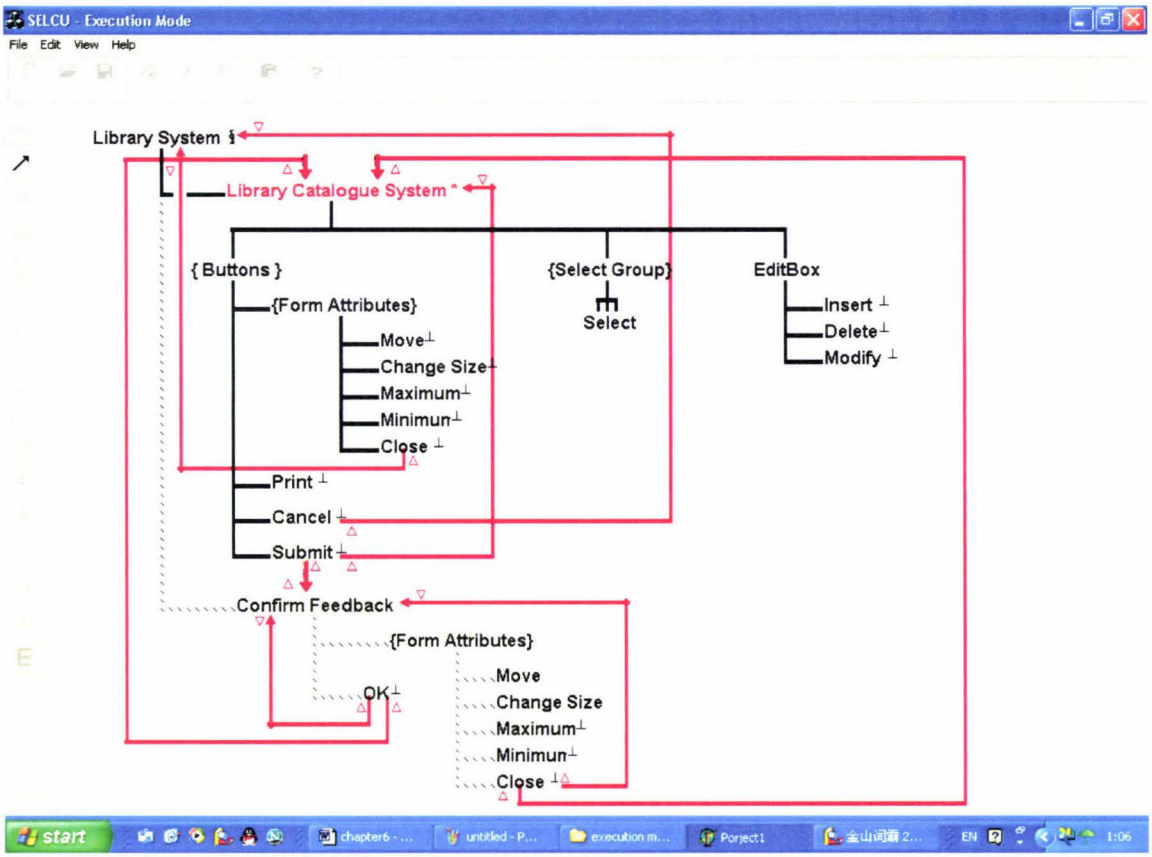


Figure 6-21: Triggers View in Execution Mode

Execution of Tasks

Tasks in Lean Cuisine+ diagrams can be single step controlled in the execution environment. In this section, the ***submit book request*** task will be demonstrated in execution mode.

At first, the user needs to call the task function from the system menu. A task display window will appear on the screen. All the tasks will be displayed in this list (see section 4.3.5). The user can select the ***Submit Book Request*** option, and then click ***OK*** button go to next step.

Following this, the software moves to the first step of the ***Submit Book Request*** task interface (which is shown in figure 6-22).

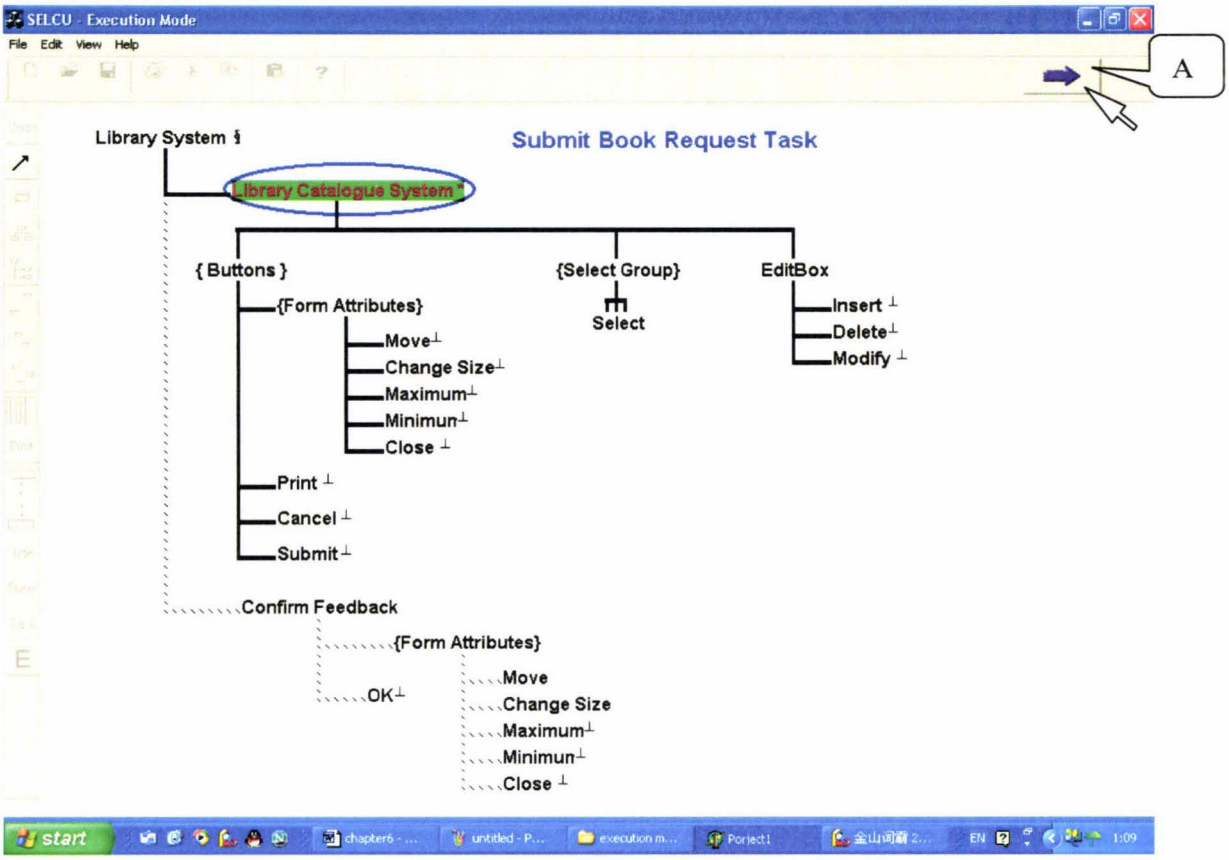


Figure 6-22: First Step of "Submit Book Request" Task

In figure 6-22,

- The user clicks the ***Next*** button (A) to move to the next step
- This is the first step of the task, so the ***Previous*** button is not available
- The blue circle on the name means this meneme is one step of the task
- The green background on the name highlights this as the current step of the task

When the user clicks the ***Next*** button, the task goes to the second step. At this time, the system layout appears as in figure 6-23. In figure 6-23, the user can click the ***Next*** button (A) to move to next step of this task, alternatively, they also can click the ***Previous*** button (B) to move to the previous step. If the user keeps clicking the ***Next*** button, the system will move to the third step (shown in figure 6-24), fourth step (shown in figure 6-25) and fifth step (shown in figure 6-26) of the task.

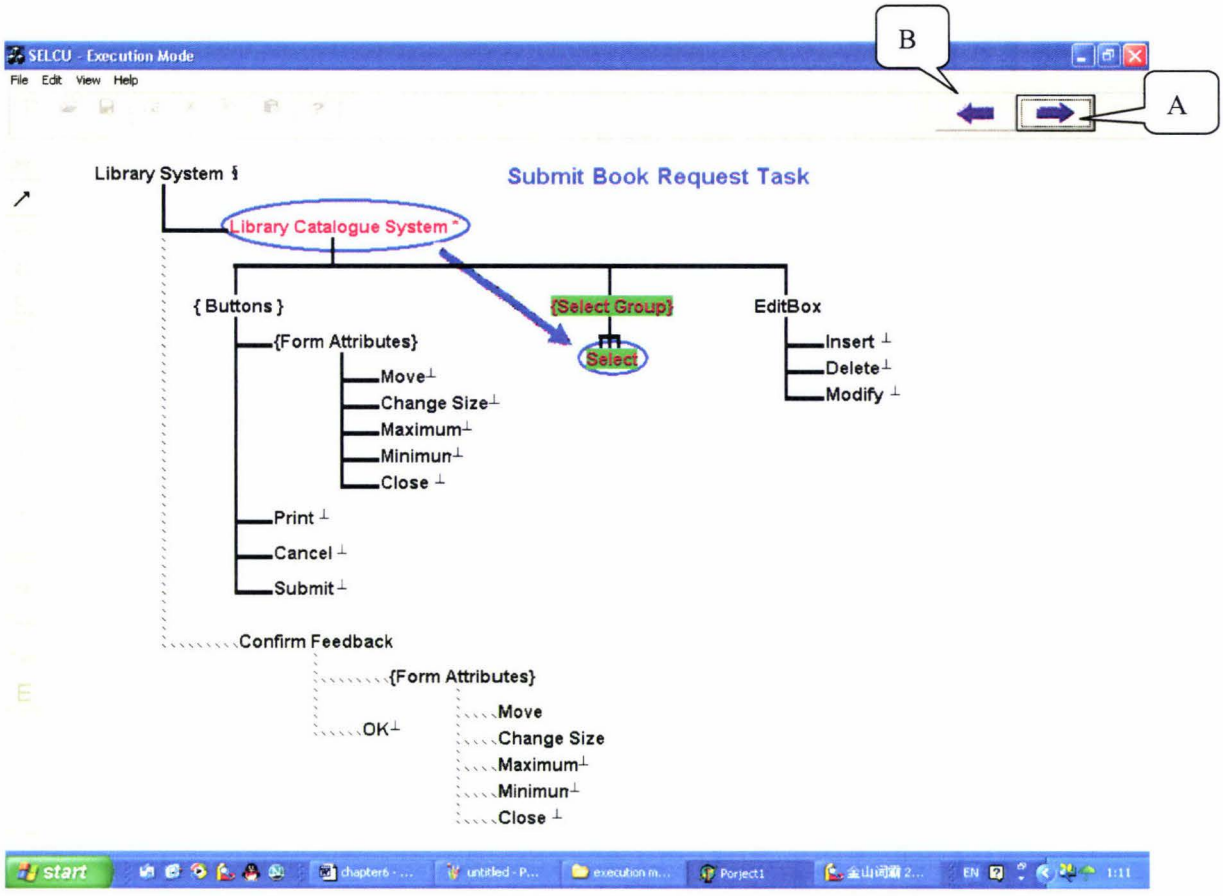


Figure 6-23: Second Step of "Submit Book Request" Task

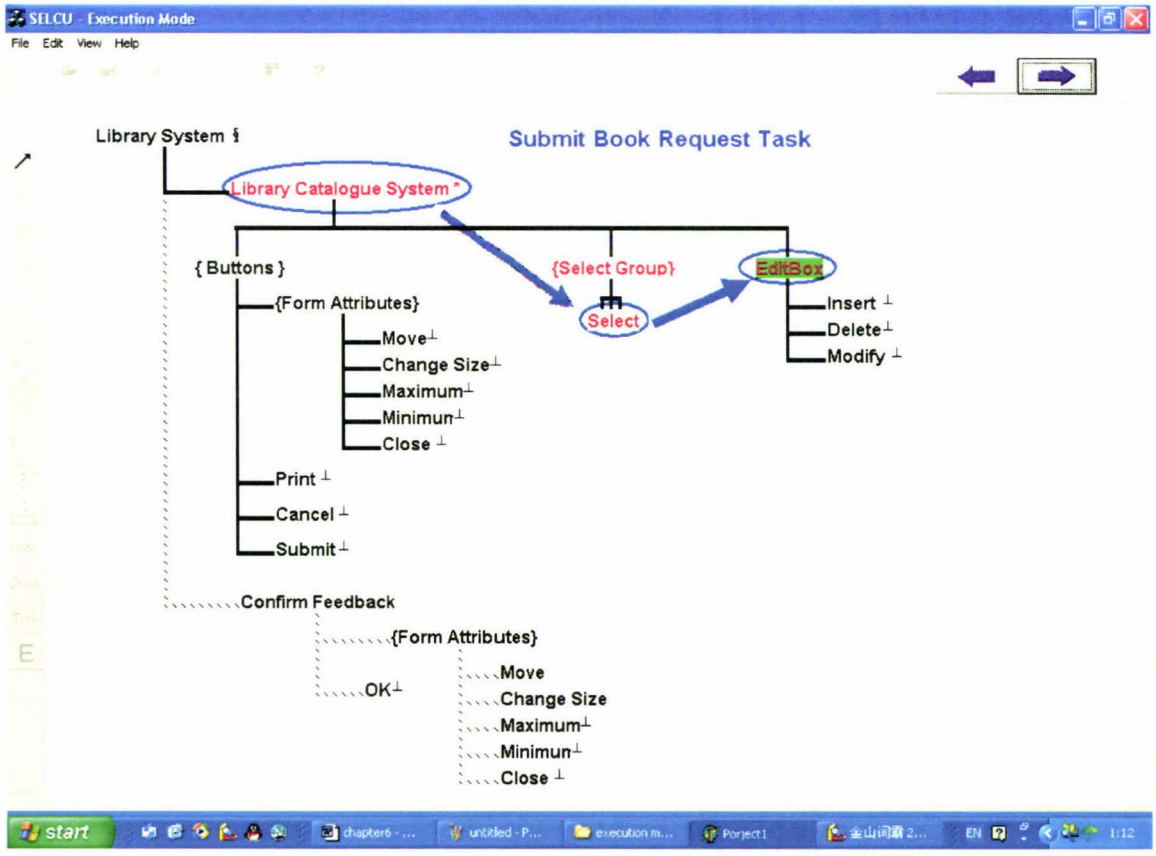


Figure 6-24: Third Step of "Submit Book Request" Task

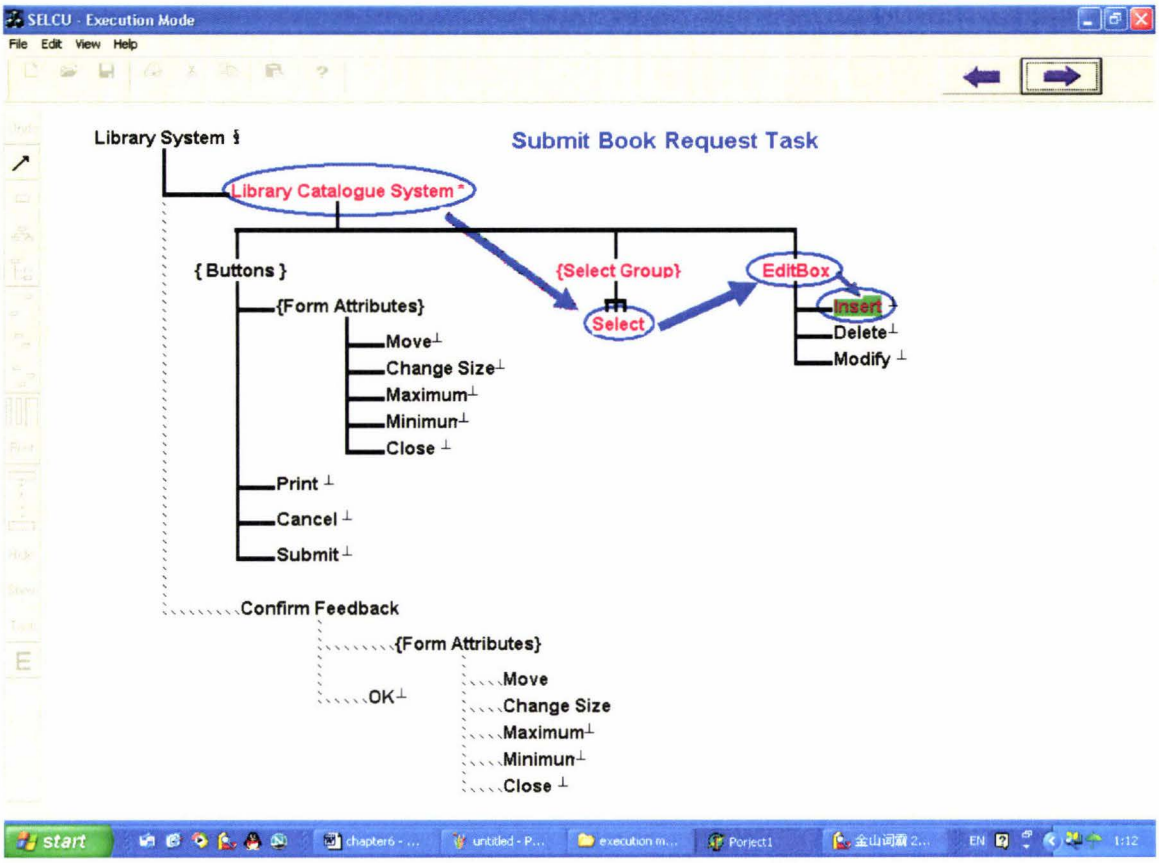


Figure 6-25: Fourth Step of "Submit Book Request" Task

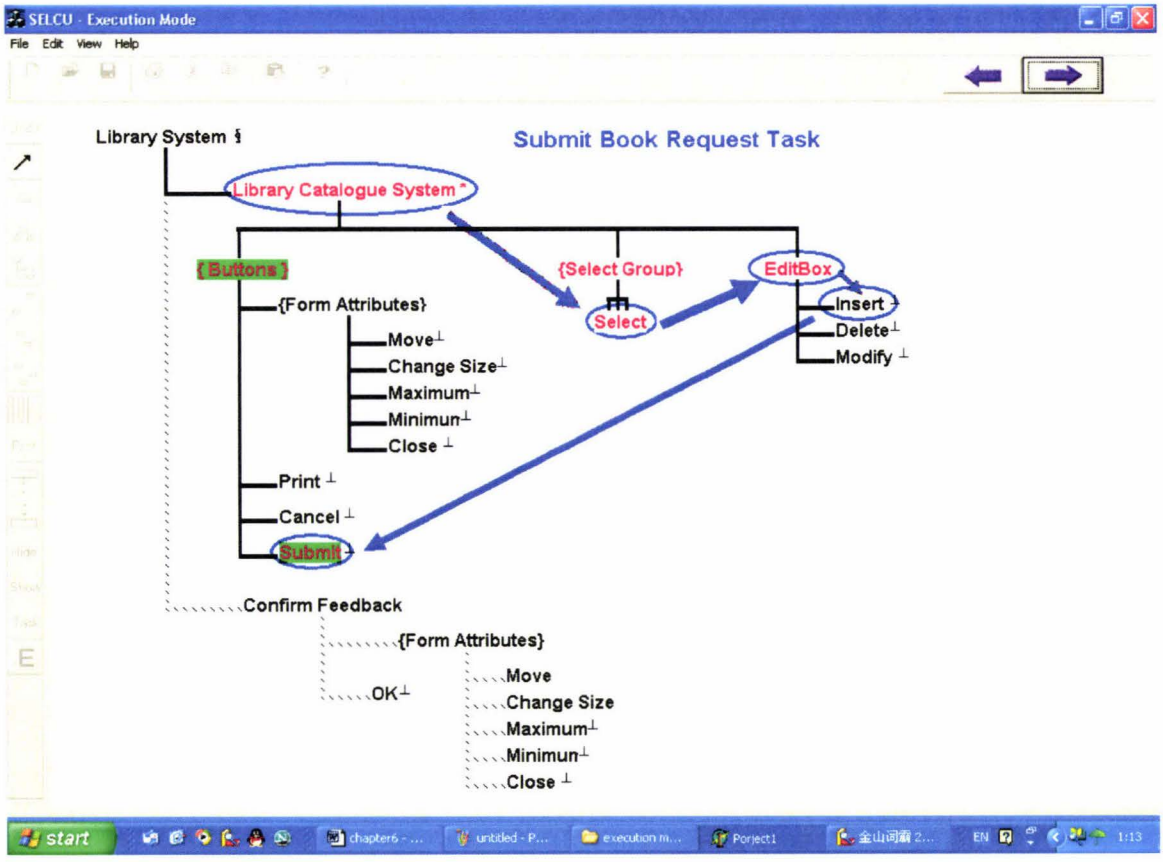


Figure 6-26: Fifth Step of "Submit Book Request" Task

Insert is a monostable meneme, so it reverts to an unselected state when the task moves to the next step.

When the task executes the sixth step (shows in figure 6-27), the ***Library Catalogue System*** group of menemes will become unavailable. At the same time, the ***Confirm Feedback*** group of menemes will become available.

When the task move to the seventh and final step (shown in figure 6-28), the ***Next*** button will disappear.

If the user wants to change back to ***trigger view*** or ***meneme view***, they can call the corresponding functions from the system menu. Otherwise, the user can click the ***Select Object*** button and move back to the SELCU editing environment.

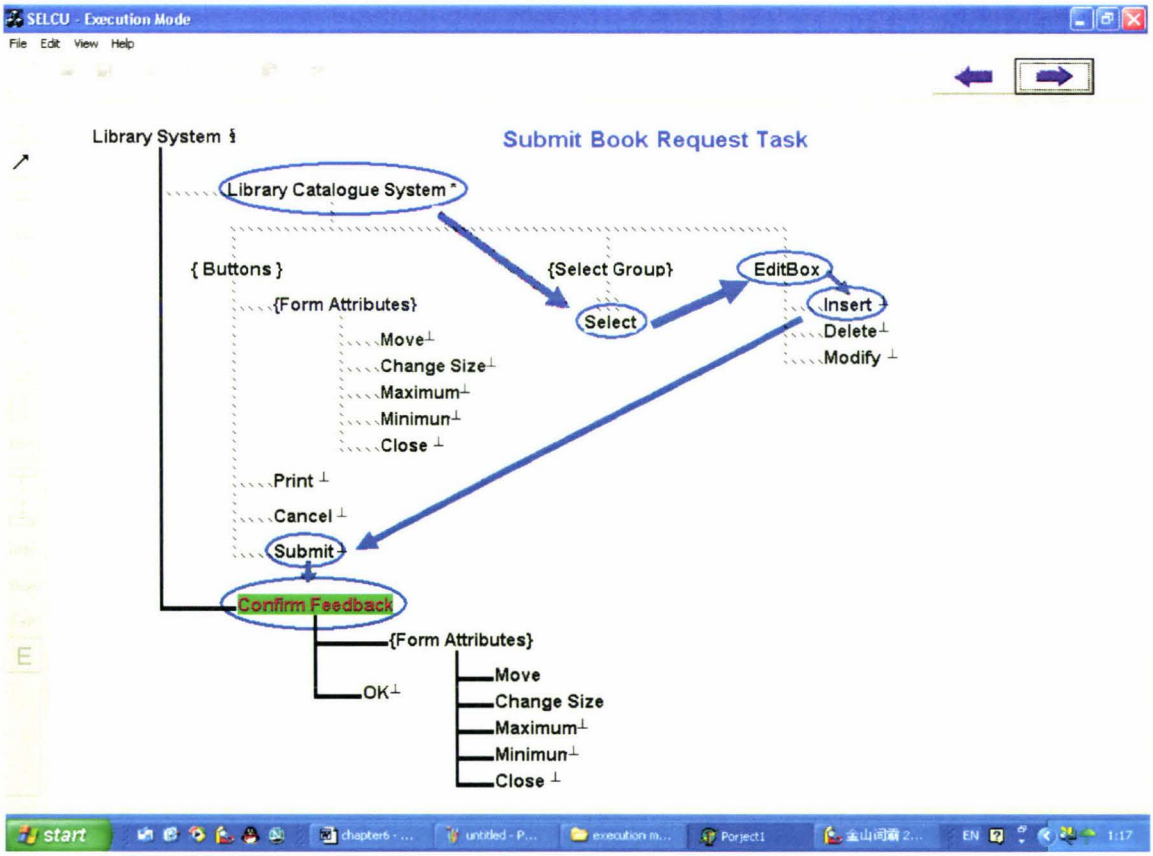


Figure 6-27: Sixth Step of "Submit Book Request" Task

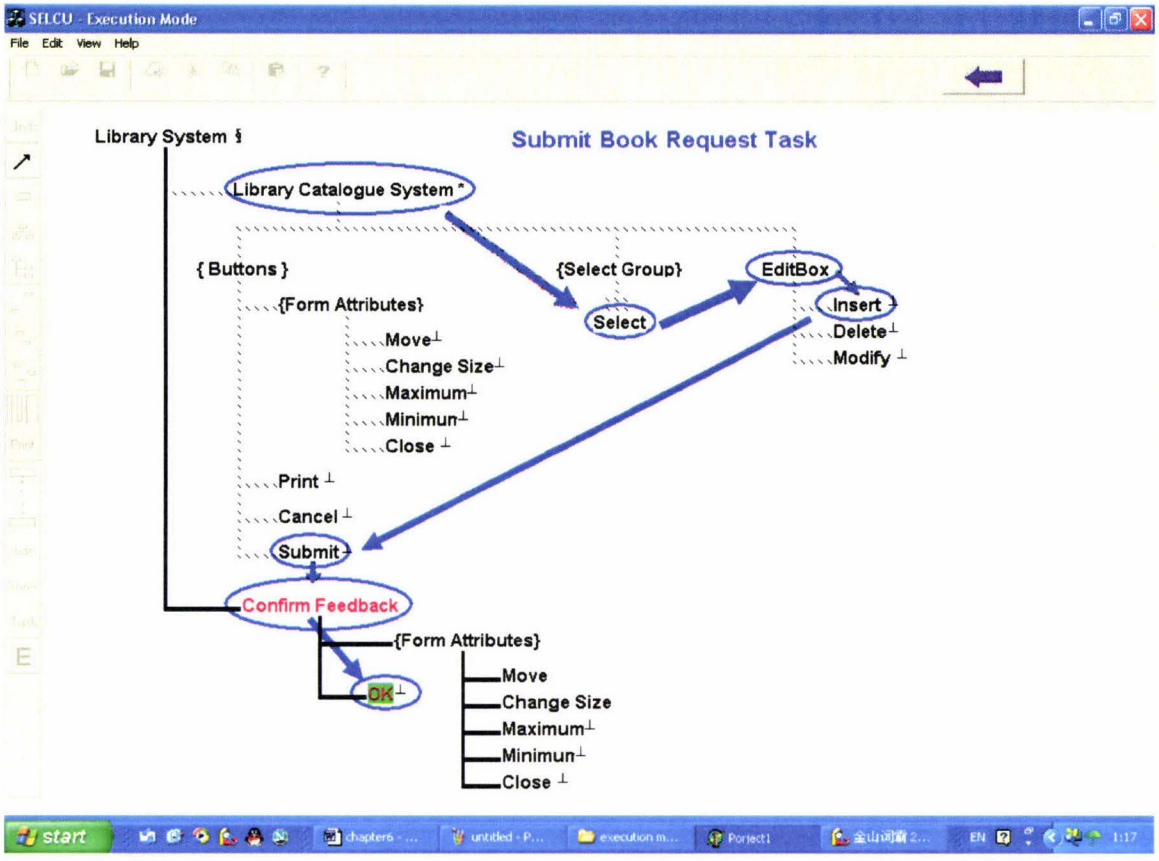


Figure 6-28: Last Step of "Submit Book Request" Task

Error Feedback

Very limited screen feedbacks have been designed in this prototype environment. When the user clicks a not available meneme (as in figure 6-29), or a virtual meneme (as in figure 6-30), an error massage is displayed

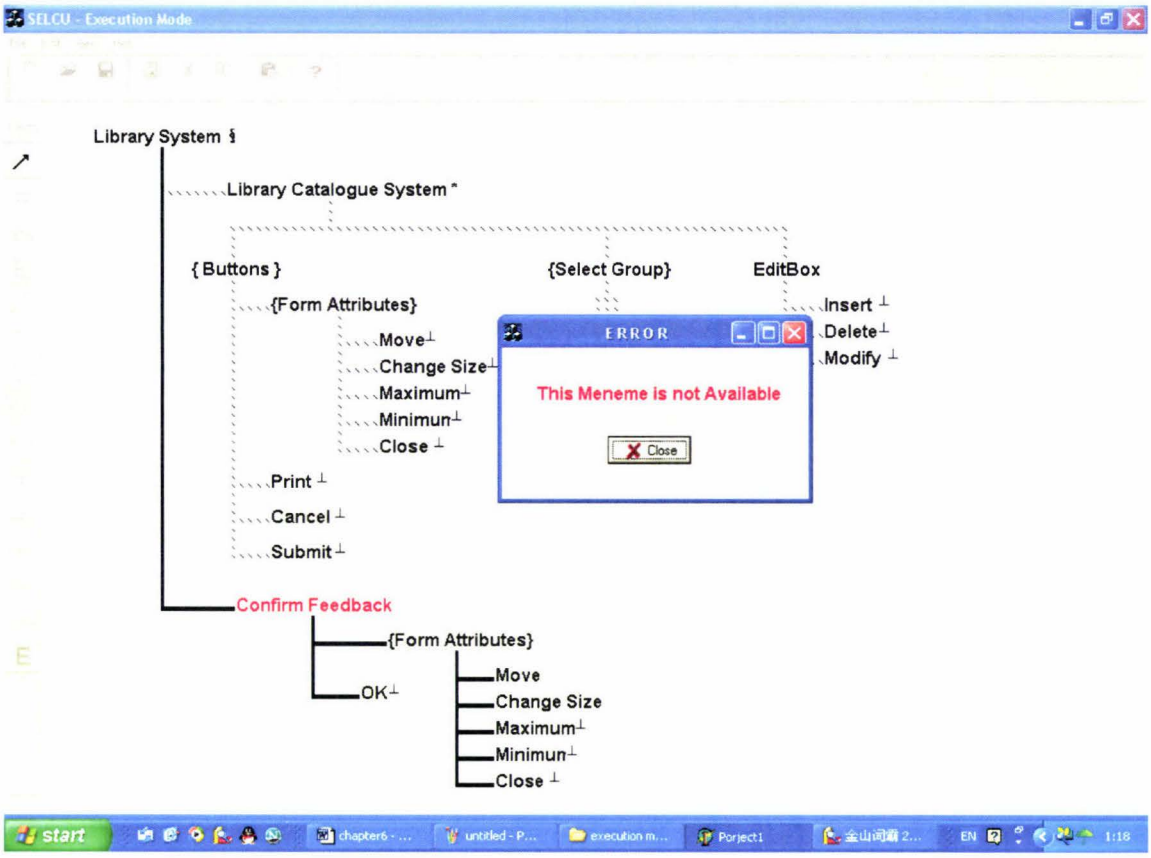


Figure 6-29: Error Feedback when the User Clicks an Unavailable Meneme

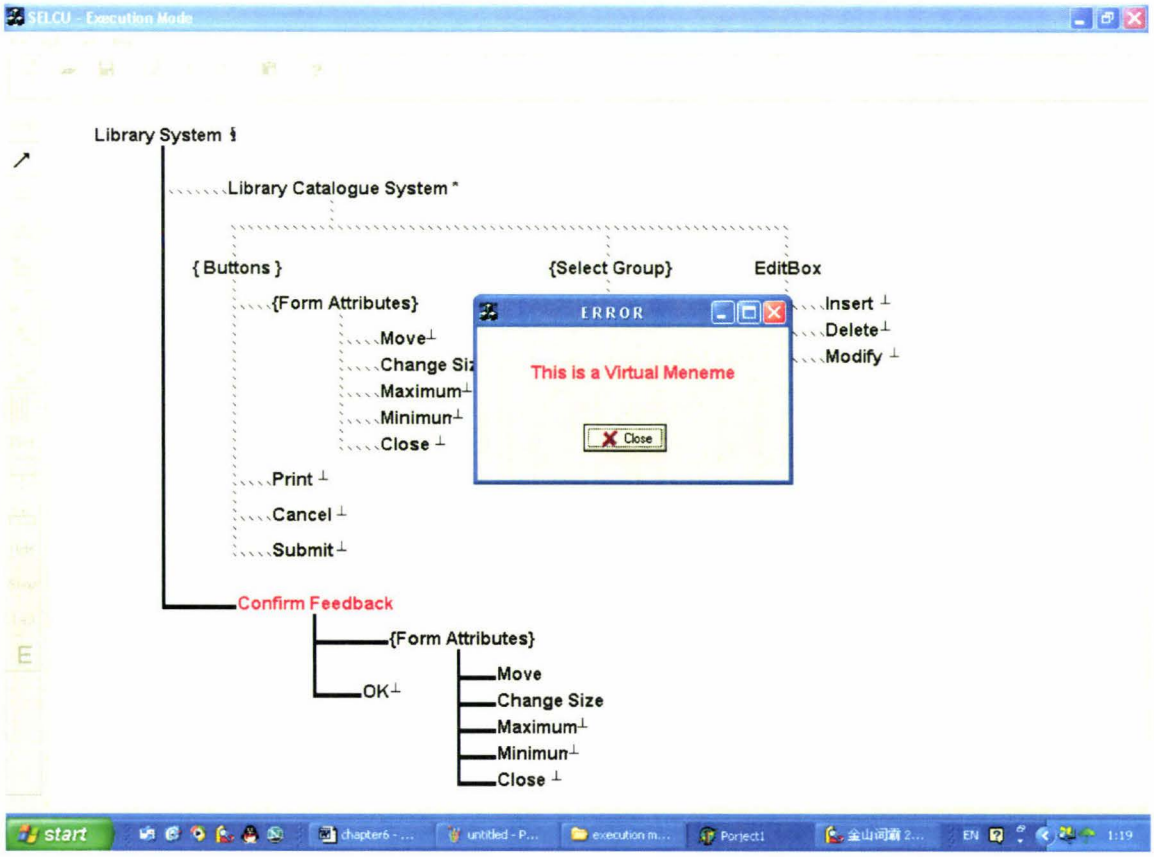


Figure 6-30: Error Feedback when the User Clicks a Virtual Meneme

6.3 Summary

Two case studies have been presented in this chapter to illustrate the Auto-Generation Software and Execution Environment functions. Two Delphi example GUIs have been successfully translated into a SELCU compatible source file, which can be opened in text format (by using the Auto-Generation Software or a text editor) or as a Lean Cuisine+ diagram (by using SELCU).

The execution environment supports the selection of menemes, and the execution of tasks. Triggers are taken into account. Most of the system functions have been demonstrated in this chapter though the case studies.

Chapter 7

CONCLUSIONS

The overall aim of the research presented in this thesis was the design and implementation of Auto-Generation Software and an Execution Environment for the Lean Cuisine+ notation. The Auto-Generation Software is used to extract the SELCU compatible source file from a Delphi interface. The Execution Environment supports meneme selection and task execution, and also takes account of triggers.

Graphical notations and their support environments, and a series of early prototyping tools and IDEs have been reviewed. SELCU, a software environment which is used to construct Lean Cuisine+ diagrams, has been described, and the requirements of the SELCU extensions have been defined. A range of design decisions corresponding to the functional requirements have been made. Lo-Fi prototyping of the Execution Environment has been carried out and the Auto-Generation Software analyzable interface components have been described. Two case studies have been used to demonstrate the SELCU extensions.

7.1 Review of the Auto-Generation Software

The Auto-Generation Software is used to generate a Lean Cuisine+ diagram from a Delphi interface. The software has been developed using the Delphi IDE. By analysis of the *.dfm and *.pas files, the Auto-Generation Software extracts a Lean Cuisine+ tree diagram, which is saved in memory using a dynamic stack

structure. When the final Lean Cuisine+ tree has been built, the software writes the diagram information into a SELCU compatible file. The user can use SELCU or any text editing software to modify the source file.

The software supports a Delphi interface components library, which includes most of the popular user interface elements. Triggers are used to implement functional events. A set of adequate error messages has been developed to give feedback. The user can directly transfer to SELCU with a generated Lean Cuisine+ diagram.

Several improvements and extensions could be made to the Auto-Generation Software to increase its flexibility and functionality. These include:

- **Extending the Recognizable Delphi Interface Components**

  The current software can recognize and analyze most of the popular Delphi interface components menu, director list box, radio group, check box and so on. An obvious extension would be to introduce support for more Delphi interface elements such as frames, memos, action lists, value lists and other such features that can improve the completeness of the generation.

- **Supporting Other Programming Languages**

  Several other software IDEs like Visual C++ IDE, Visual Basic IDE, Java IDE are used to design applications. Most of the popular programming languages have their own comprehensive integrated development environments. The software could be extended to support other programming languages.

- **Supporting a Wider Range of Programming Styles**

  The current version of the Auto-Generation Software can only analyze certain programming styles (those based on a nested if...then... else... structure). An extension would be to increase the range of programming styles capable of being analyzed.

7.2 Review of the Execution Environment

The Execution Environment is embedded in SELCU, and allows the user to easily switch from SELCU edit mode to execute mode. In the Execution Environment, all the edit mode functions become unavailable, and modification of the Lean Cuisine+ diagram is not permitted. The Execution Environment lets the user select menemes and tasks, and takes account of triggers. The system uses different meneme name colors (black and red) and line types (solid and dotted) to represent the four meneme states.

The Execution Environment includes execute mode, preview mode, task view mode and trigger view mode. Execute mode is the basic state of the Execution Environment, and allows the user to select the menemes on the Lean Cuisine+ diagram. The right mouse button can be used to select a preview mode, when selected meneme names are presented with underline. The user can single step control a task in task view mode. This involves step back and step forward functions. The user can select trigger view mode from the system menu, when all the triggers are presented on the diagram.

The Execution Environment has been developed in C++. The Microsoft Foundation Classes have been used in creating applications.

A significant improvement would be to extend the Execution Environment to execute the Delphi interface as well. This would require a means of mapping from Lean Cuisine+ into Delphi (object Pascal). The Execution Environment could then provide for manipulation of both the Lean Cuisine+ specification and the associated Delphi interface in tandem. Figure 7-1 shows a mock up of this extension. In this environment, the user could either make selections in the Lean Cuisine+ diagram which would be reflected in the Delphi interface, or manipulate the interface and make changes to the state of the Lean Cuisine+ diagram.

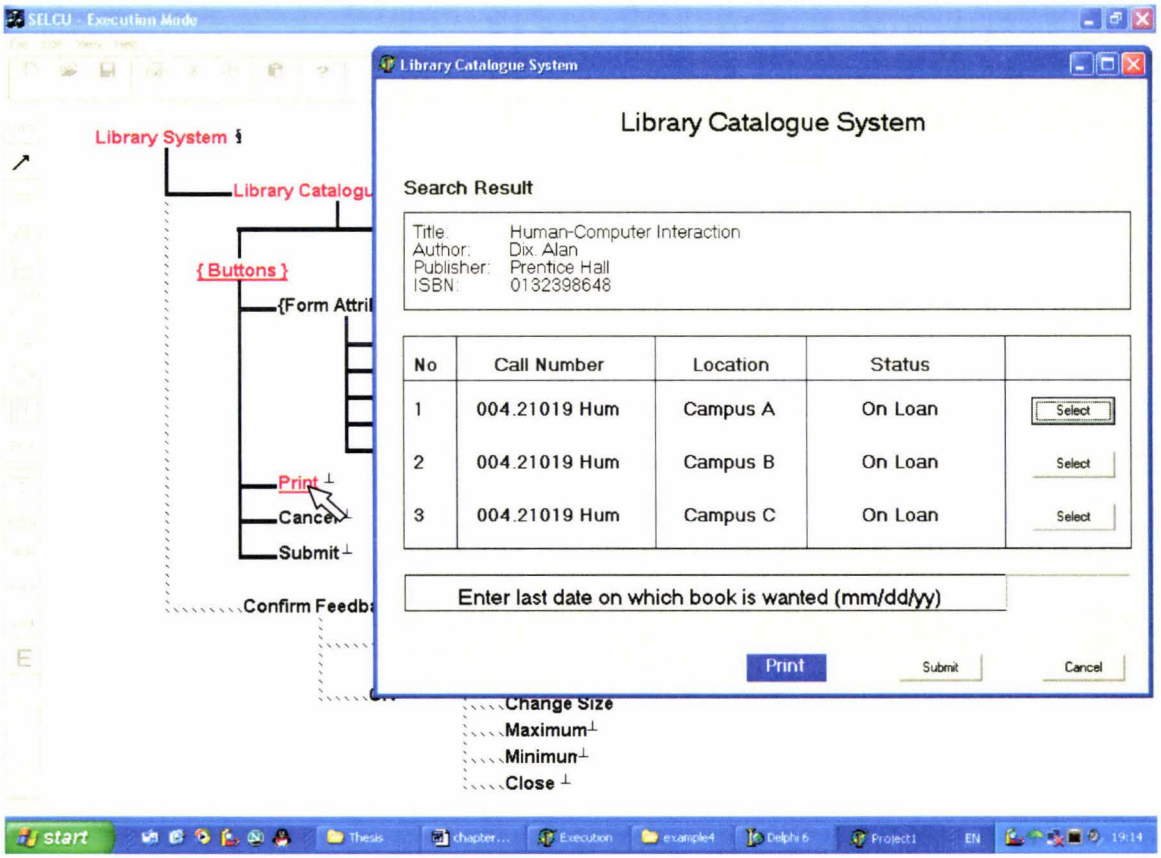


Figure 7-1:

Tandem Manipulation of the Lean Cuisine+ Diagram and the Interface

In figure 7-1,

- If the user selects the ***Print*** meneme on the Lean Cuisine+ diagram in the Execution Environment, the ***Print*** button in the Delphi interface also becomes selected.

- If the user clicks the ***Print*** button in the Delphi interface, it also causes a selection of the ***Print*** meneme in the Lean Cuisine+ diagram.

Figure 7-2 is a mock up of tandem task execution.

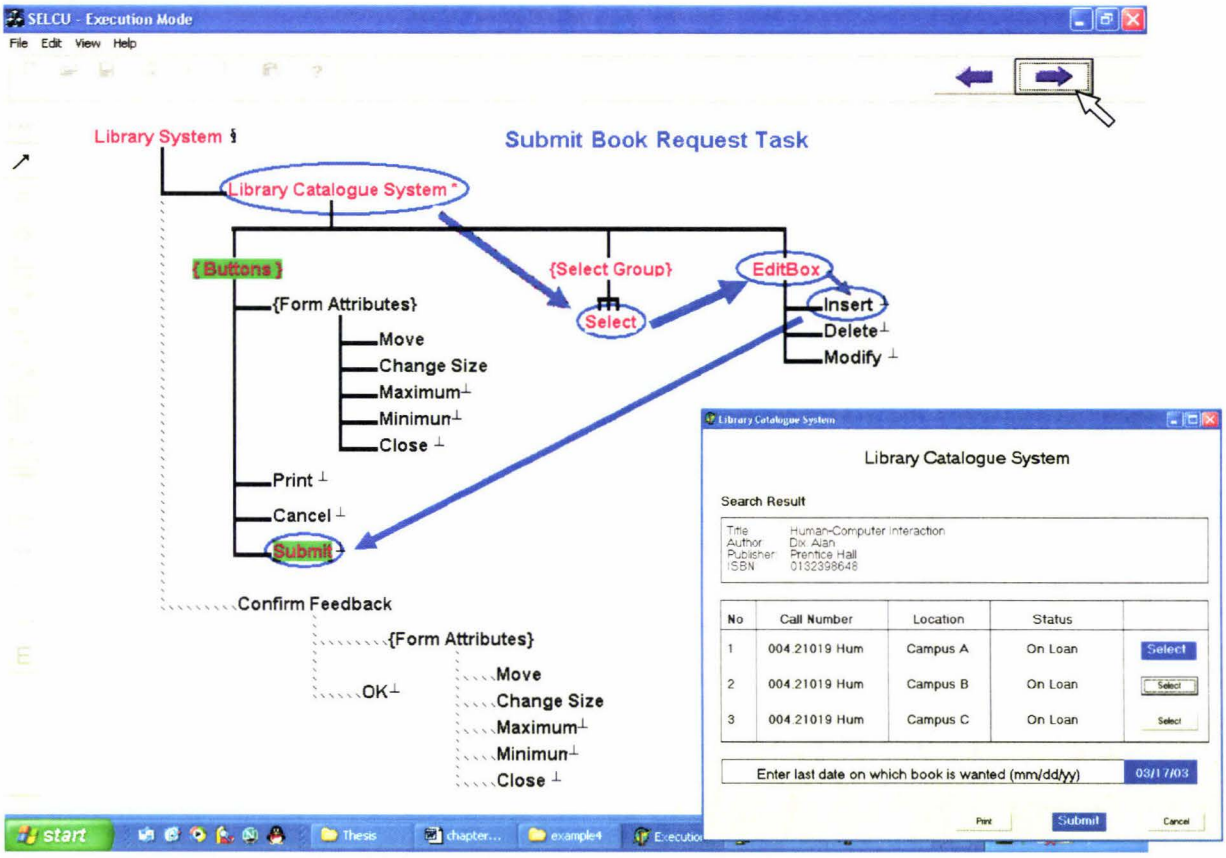


Figure 7-2:

Executing a Task both in Lean Cuisine+ and the Delphi Interface

7.3 Conclusion

The SELCU extensions to support the Lean Cuisine+ graphical notation have successfully achieve the goals presented in section 1.3, and provide software tools to support the generation (from Delphi source code) and the execution of Lean Cuisine+ diagrams.

The Auto-Generation Software successfully integrates the Lean Cuisine+ notation, the Object Pascal programming interface (Delphi) and an existing graphical notation support environment (SELCU). It offers the Delphi interface designer the option of obtaining a Lean Cuisine+ diagram describing the underlying behavior of the interface from Delphi source code. The Execution Environment extends the functionality of the SELCU system. It supports the selection of menemes, the execution of tasks, takes account of triggers, and presents a dynamic view of a Lean Cuisine+ diagram.

These SELCU extensions, coupled with the strong descriptive power of the Lean Cuisine+ notation, offer a way forward for the early design of graphical user interfaces.

REFERENCES

Abowd G.D and Beale R. (1991): Users, systems and interface: A unifying framework for interaction, *HCI'91, People and Computers VI*, Cambridge University Press, 73 ~ 87

Anderson P.S. and Apperley M.D. (1990): An interface prototyping system based on Lean Cuisine, *Interacting with Computers Journal*, VOL 2, No 2, 217~226

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

Apperley M.D. and Spence R. (1989): Lean Cuisine: a low-fat notation for menus, *Human-Computer Interaction Readings,* IIST, MASSEY University, 77 ~ 100

Baecker R. M., Grudin J. and Buxton W.A.S. (1995): *Readings In Human-Computer Interaction: Toward the Year 2000*, Morgan Kaufmann Publishers, Inc

Bennett D. (1997): *Visual C++ 5.0 Developer's Guide*, Sams Publishing

Bennett S., McRobb S. and Farmer R. (1999): *Object-Oriented Systems Analysis and Design using UML*, McGRAW-HILL

Brown J. and Johnston C. (2001): Extending the Value of Prototype with Panorama a Tool to Browse Software Artifacts, *SIGCHI_NZ'01Proceedings*, IEEE Computer Society Press, 51~20

Brown J. and Marshall S. (1998): Sharing Human-Computer Interaction and Software Engineering Design Artifacts. *OZCHI'98 Proceedings*, IEEE Computer Society Press, 53~60

Calvert C. (1999): *Charlie Calvert's Delphi 4 Unleashed*, Sams Publishing

Cantu M. (1999): *Mastering Delphi 5*, SYBEX Inc

Cantu M. (2001): *Mastering Delphi 6*, SYBEX Inc

Cockton G. (1990): Lean Cuisine: no sauces, no courses, *Interacting with Computer*, Vol 2, no 2, 205 ~ 216

Collins D. (1995): *Designing Object-Oriented User Interfaces*, Benjamin / Cummings Publishing Company, Inc.

Deitel H.M. and Deitel P.J. (1994): *C++ How to Program*, Prentice-Hall, Inc

Diaper D. (1997): Integrating HCI and Software Engineering Requirements Analysis, *SIGCHI Bulletin*, Volume 29, Number 1, 41 ~ 50

Dix, A., Finlay, J., Abowd and G., Beale, R. (1997): Dialogue Notations and Design, *Human-Computer Interaction*, Prentice Hall, 251~297

Dorfman L. (1995): *C++ By Example: Object-Oriented Analysis, Design & Programming*, McGraw-Hill, Inc.

Farooq M.U. and Dominick W.D. (1998): Survey of formal tools and model for developing user interfaces, *Int.J.Man-Mach*, 479 ~ 496

Fung, P.W., Kemp, R.H., and Kemp, E.A (1998): *Exploiting the statechart in interactive learning systems authoring*, Proceedings of the 1998 Australian Computer Human Interaction Conference OzCHI '98, 340~341

Ford W. and Topp W. (1996): *Data Structures with C++*, Prentice Hall, Englewood Cliffs, New Jersey 07632

Gosselin D. (2001): *Microsoft Visual C++ 6.0*, Course Technology, a division of Thomson Learning, Inc.

Gu J.P. (1995): A Menu Interface Development Environment Based on Lean Cuisine, *Master Thesis*, Institute of Information Science and Technology, Massey University

Harel D. (1987): Statecharts, A visual formalism for complex systems, *Science of Computer Programming*, 8, 231- 274

Harel D. (1988): On Visual Formalisms, *Communications of the ACM,* Vol 31, No 5, 514 ~ 530

Harel D., Pnueli A., Schmidt J.P. and Sherman R. (1987): On the Formal Semantics of Statecharts, *Proceedings of the 2nd Symposium on Logic in Computing Science*, 54 ~ 64

Harel D. and Politi M. (1998): *Modelling reactive systems with statecharts*, McGraw-Hill

Hix D. and Harson R. (1993): *Developing User Interfaces: Ensuring Usability Through Product and Process*, Wiley Professional Computing

Horton I. (1997): *Beginning Visual C++ 5*, Wrox Press Ltd

Jacobson I., Booch G. and Rumbaugh J. (1999) *The Unified Software Development Process*, Addison-Wesley

Jacobson J.Q. (1999): *Delphi Developer's Guide to OpenGL*, Wordware Publishing, Inc

Jacobson R.J.K (1985): A state Transition diagram Language for Visual Programming, IEEE Computer, 18, 51-59

Jensen K. (1992): *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, Volume 1, Basic Concepts Springer Verlag

Joy B. (1999): Statechart Implementation, Honours project report, IIST, Massey University

Kate G. (1998): *Special Edition Using Visual C++ 6*, Prentice Hall

Kemp E. and Phillips C.H.E (1998): Extending Support for User Interface Design in Object-Oriented Software Engineering Methods, *Proc of HCI'98*, Sheffield, England, 96 ~ 97

Kruglinski D.J., Wingo S. and Shepherd G. (1998): *Programming Microsoft Visual C++*, Microsoft Press

Li G.M., Chao L. and Fu R. (2002): *Delphi 6.0 Cheng Xu She Ji Jiao Cheng*, Ye Jing Gong Ye Chu Ban She

Lischner R. (2000): *Delphi In A Nutshell A Desktop Quick Reference*, O'Reilly & Associate, Inc

Luo P., Szekely P. and Neches R. (1993): Management of Interface Design in Humanoid, *ACM InterCHI'93*, 107 ~ 114

Murata T. (1989): Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, Vol 77, No 4, 541-580

Myers B.A. (1995): State of the Art in User interface Software Tools, *Reading in Human-Computer interaction: Toward the Year 2000,* Morgan Kaufmann Publishers Inc, 323~343

Myers B.A. (1995): User Interface Software Tools, ACM Transactions on Computer-Human Interaction. Vol. 2, no. 1, March 1995. 64 ~ 103

Myers B.A. Web site at: **http//www-2.cs.cmu.edu/~bam/**

Myers B.A., Hudson S.E. and Pausch R. (1999): Past, Present and Future of User Interface Software Tools, *HCI In the New Millennium,* ACM Press, Addison-Wesley, 213 ~ 233

Pacheco X. and Teixeira S. (1998): *Delphi 4 Developer's Guide*, Sams Publishing

Palanque P. (1995): Task Model-System Model: Towards an Unifying Formalism, *Proceedings of HCI International Conference*, 489 ~ 494

Pappas C.H. and Murray W.H. (1994): *The Visual C++ Handbook*, Osborne McGraw-Hill

Pappas C.H. and Murray W.H. (1995): *C/C++ Programmer's Guide*, Ziff-Davis Press, Emeryville, California

Pemberton S. (1997): *Human Factors In Computing Systems, CHI 97 Conference Proceedings,* ACM, Inc

Peterson. J. L. (1981): *Petri Net Theory and The Modeling of Systems*, Prentice-Hall

Petri Nets Web sit at: **http://www.daimi.au.dk/PetriNets/**

Petri Nets Bibliography Web sit at: **http://www.informatik.uni-hamburg.de/TGI/pnbib/**

Pfaffenberger B. (1999): *Linux Clearly Explained*, Academic Press

Phillips C.H.E (1992): *Extending Lean Cuisine: A Case Study in Reverse Engineering*, School of Information Science, MASSEY University

Philips C.H.E. (1993): Software Support for Lean Cuisine+, *PhD Thesis,* MASSEY University, 157~171

Phillips C.H.E. (1994): Review of Graphical Notations for Specifying Direct Manipulation Interfaces. *Interacting with Computers,* 411~431

Phillips C.H.E. (1994): Serving Lean Cuisine+: Towards a Support Environment, *Proceedings OZCHI94*, CHISIG of Ergonomics Soc. of Australia, 41~46

Phillips C.H.E (1995): Lean Cuisine+: An Executable Graphical Notation for Describing Direct Manipulation Interfaces. *Interacting with Computers*, 49~71

Phillips C.H.E and Apperley M.D (1990): *Direct Manipulation Interaction Tasks: A Macintosh-Based Analysis*, School of Information Science, MASSEY University

Phillips C.H.E and Kemp E. (2001): Extending UML use case modeling to support graphical user interface design, *Proceedings of ASWEC 2001*, IEEE, Canberra, Australia, 48 ~ 57

Phillips C.H.E and Kemp E. (2002): In Support of User Interface Design in the Rational Unified Process, *Proceedings of the Third Australasian User Interface Conference*, Australian Computer Society, 21~27

Phillips C.H.E and McKauge J. (1997): *OZCHI'96 Industry Session, Sixth Australian Conference on Human-Computer Interaction*, Department of Computer Science, University of Waikato

Phillips C.H.E and Scogings C. (1997): Modelling the mock-up: towards the automatic specification of the behaviour of early prototypes, *Interact '1997*, IFIP, Chapman and Hall, London, 591~592., Sydney, Australia

Phillips C.H.E and Scogings C. (1998): Task and Dialogue Modelling: Bridging the Divide with Lean Cuisine+, *Proceedings of the First Australasian User Interface Conference*, IEEE, 81~87

Phillips C.H.E and Scogings C. (1998): Towards the automatic specification of the behavior of early prototypes using Lean Cuisine+, Proc of Software Engineering: Education and Practice (SE:E&P'98), IEEE, Dunedin, January 1998, 238~244

Quatrani T. (1998): *Visual Modeling with Rational Rose and UML*, Addison-Wesley

Rachele W. (2001): *Learn Object Pascal with Delphi*, Wordware Publishing, Inc.

Rosson M.B and Carroll J.M. (1995): Integrating Task and Software Development for Object-Oriented Applications, *CHI95' Proc*, ACM, New York, 377 ~ 384

Roy P., Smith J. and Lyons P. (2000): *Doing Battle with the Delphi IDE*, IIST, MASSEY University

Santos L.A. (1999) *Essential Skills with Delphi5*, World Wide Express Books

Schach, S.R. (1997): *Software Engineering with Java*, Times Mirror higher Education Group, Inc

Scogings C. (1998): Modifications to the Lean Cuisine+ Notation, *Internal Report*, Institute of Information & Mathematical Sciences, Massey University

Scogings C. (2000): The Lean Cuisine+ notation revised, *Research Letters in the Information & Mathematical Sciences (2000), no. 1*, Massey University, Auckland, New Zealand, 17~23

Scogings C. (2002): The SELCU User Manual, *Appendix of PHD Thesis*, Information & Mathematical Sciences, Massey University

Scogings C. (2003): PHD thesis, Massey University, In Progress

Scogings C. and Phillips C.H.E (1998): Beyond the interface: modeling the interaction in a visual programming environment, Proc. Of HCI'98, Sheffield, England, September 1998, 1~9

Scogings C. and Phillips C.H.E (1999): Modelling tasks and dialogue with Lean Cuisine+, *Position Paper*, Workshop 7, CHI'99, Pittsburgh

Scogings C. and Phillips C.H.E (2000): A method for the Early Stages of Interactive System Design using UML and Lean Cuisine+, Proceedings of AUIC 2001, Gold Coast, Australia, 69 ~ 76

Scogings C. and Phillips C.H.E (2001): Linking tasks, dialogue and GUI design: a method involving UML and Lean Cuisine+, *Interacting with Computers Journal*, Elsevier, Vol14, No 1, 69-86

Scogings C. and Phillips C.H.E (2002): *Linking Task and Dialogue Modelling: Towards an Integrated Software Engineering Method*, Working Papers, MASSEY University, Working Paper

Shannon C. (1999) *Developer's Guide to Delphi Troubleshooting*, Wordware Publishing, Inc

Sphar C. (1999): *Learn Microsoft Visual C++ 6.0 Now*, Microsoft Press

StateChart Tools Web site at:

http://moncs.cs.mcgill.ca/people/sborla/tools/

Stevens P. and Pooley R. (2000) *Using UML: Software Engineering with Objects and Components*, Addison-Wesley

Swan T. (1998): *Delphi 4 Bible*, IDG Books Worldwide,Inc.

Wiener R. and Wiatrowski C. (1996) *Visual Object-Oriented Programming using Delphi,* SIGS Books & Multimedia

Williams S. and Walmsley S. (1999): *Discover Delphi Programming Principles Explained*, Addison Wesley Longman Limited

Wolfgang R. (1985): Petri Nets: An introduction, *EATCS Momographs on Theoretical Computer Science*, vol 4

Wright C. and Jamsa K. (2001): *1001 Microsoft Visual C++ Programming Tips*, Prima Publishing

Beck Z. (1997): *Microsoft Visual C++ Owner's Manual*, Microsoft Press

Zhou C.F. (2000): *Jing Tong Visual C++ Tu Xiang Bian Chen*, Publishing House of Electronics Industry

Zurawski R. and Zhou M.C. (1994): Petri Nets and Industrial Applications: A Tutorial, *IEEE Transactions on Industrial Electronics*,Vol. 41,No. 6, 567 ~ 583

Appendix A

USER MANUAL

This manual covers the use of the Auto-Generation Software and Execution Environment for the Lean Cuisine+ notation. The Auto-Generation Software extracts the Lean Cuisine+ diagram from Delphi source code, and translates it into a SELCU compatible file. The Execution Environment is embedded in SELCU, and is used to select the menemes, execute the tasks and take account of triggers. The software also includes adequate error feedback.

The Auto-Generation Software and Execution Environment programs run on Windows XP or similar environments which have the Microsoft .Net framework. The Auto-Generation Software was developed in Delphi 6.0, and the Execution Environment was implemented in C++ using Microsoft Foundation Classes. Both are recommended to run on a PIII 266 (or higher), 256M memory (or more) and 8M Graphical Memory (or more) hardware system.

A1. The Auto-Generation Software

Five buttons are presented in sequence in the left area of the Auto-Generation screen. The button sequence (from top to bottom) indicates the process order. Two edit bars are used to show the result of selecting (the ***Select *.dfm File*** and ***Select *.pas File*** buttons). Alternatively, if the user knows the paths of the object files (which are Delphi Dfm and Pas source files), they can directly enter these in the edit bars. The large text area in the bottom-right of the interface is used to display the SELCU source file content. If the width and length of the contents are bigger than this text area, scroll bars will appear to adjust the view. Figure A-1 shows the Auto-Generation Software interface.

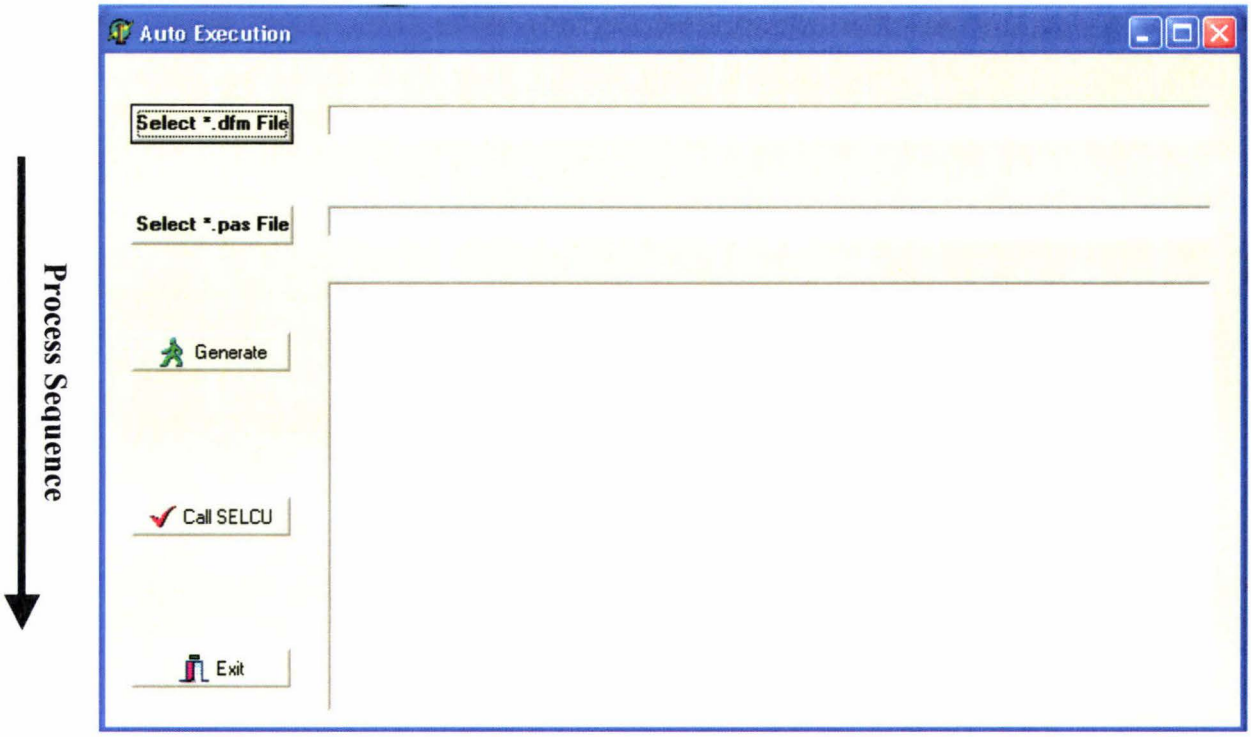


Figure A-1: The Auto-Generation Software Interface

In figure A-1, the five buttons have the following meanings:

Select *.dfm File	Find the object *.dfm Delphi source file
Select *.pas File	Find the object *.pas Delphi source file
Generate	Generate the SELCU source file
Call SELCU	Switch to SELCU with the generated source file
Exit	Quit the software

To generate a Lean Cuisine+ diagram from a Delphi interface, proceed as follows:

(1) Click the ***Select *.dfm File*** button to find the object *.dfm file.

(2) Click the ***Select *.pas File*** button to find the object *.pas file.

(3) Click the ***Generate*** button to generate the SELCU source file:

(3.1) If the object files are not recognized by the software, then go back to step 1. The error feedback is presented in figure A-2.

(3.2) If the object files are recognizable, a SELCU source file is generated and displayed. The system interface is shown in figure A-3.

(4) Click the ***Call SELCU*** button to switch to the SELCU environment with the generated SELCU source file.

(5) At anytime, click the ***Exit*** button to quite the Auto-Generation Software, and save the SELCU source file.

Figure A-2: Error Feedback

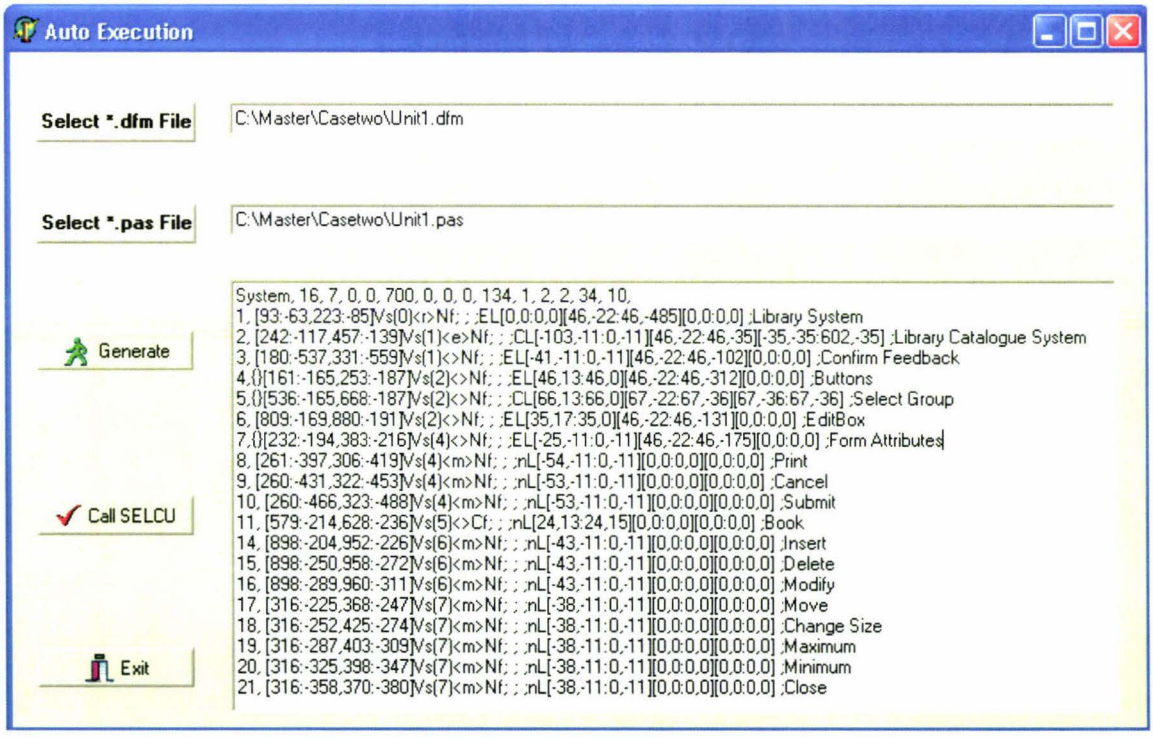


Figure A-3: Automatic Generation of SELCU Source File

A2. The Execution Environment

The user clicks the ***Execution Environment*** button (E) in SELCU edit mode to enter the execution mode. A large work area is displayed in the center of the execution environment. All the shortcuts on the system toolbar and most of the buttons on the SELCU toolbar become unavailable. Only one button (↗), which is used to switch back to the SELCU edit mode, is available on the SELCU toolbar. At the top of the interface, the title line displays the words ***Execution Mode***. Only one of the four menus in the drop-down menu bar (***view*** sub-menu) is selectable. Figure A-4 presents the interface of the execution environment with a Lean Cuisine+ diagram in the work area.

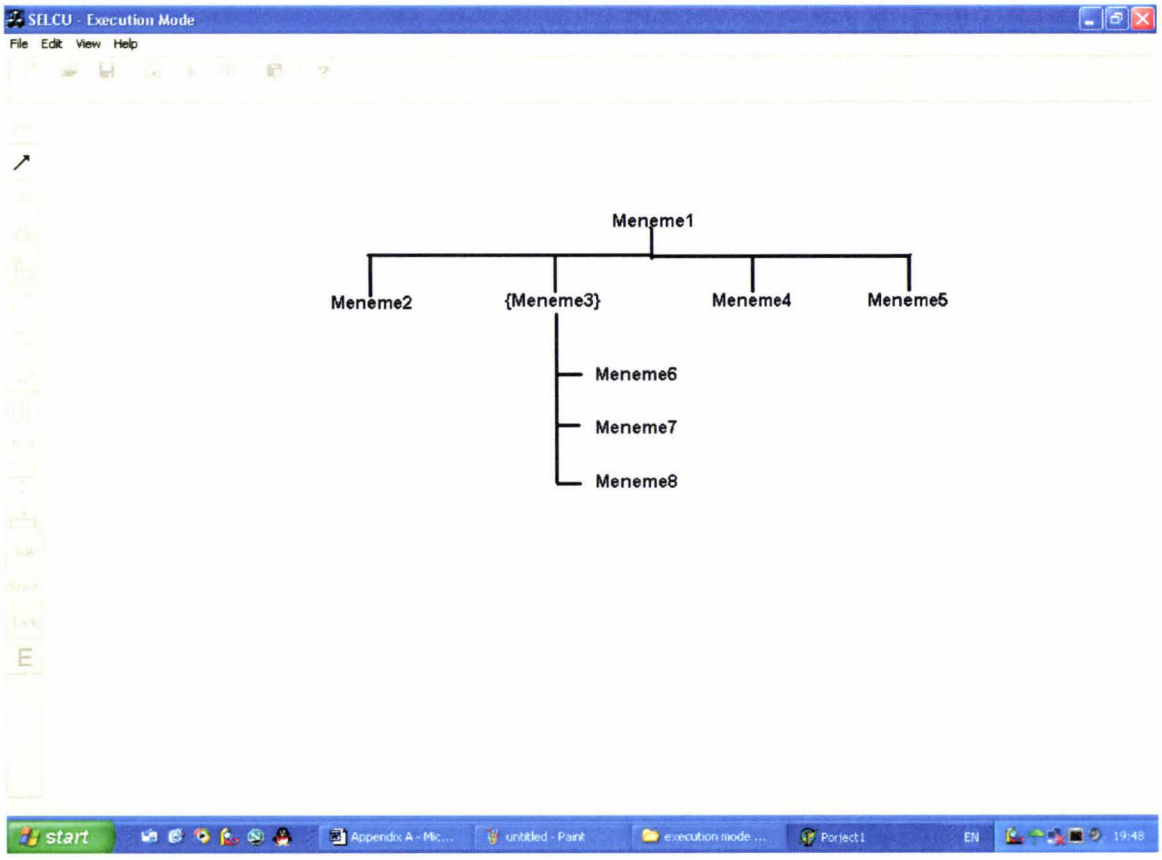


Figure A-4: Interface of the Execution Environment

In the above figure, four pull down menus appear on the top line of the interface: ***File, Edit, View*** and ***Help***. But only the ***View*** menu is selectable. It includes three sub-options that are shown in figure A-5:

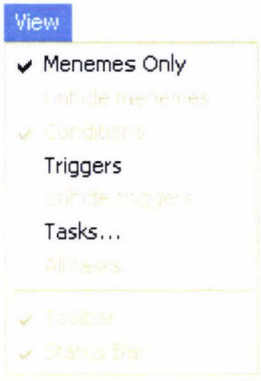


Figure A-5: "View" Menu

Three mutually exclusive options are used to switch between different modes:

Menemes Only	switch to Meneme view mode
Triggers	switch to Triggers view mode
Tasks...	switch to Task view mode

Five main functions are included in the execution environment. These are meneme preview, meneme selection, task execution, trigger view and exit execution mode.

Meneme Preview

To preview the effect of selecting a meneme, proceed as follows:

(1) If the execution environment is not in meneme view mode, select the ***Menemes Only*** option from the ***View*** pull down menu.

(2) Move the cursor to the object meneme (***Meneme 6***) and, press the right mouse button. The object meneme enters preview mode. The object meneme (***Meneme 6***) and related menemes (***Meneme 3 and Meneme 1***) change to red, and an underline appear on their names. This is shown in figure A-6.

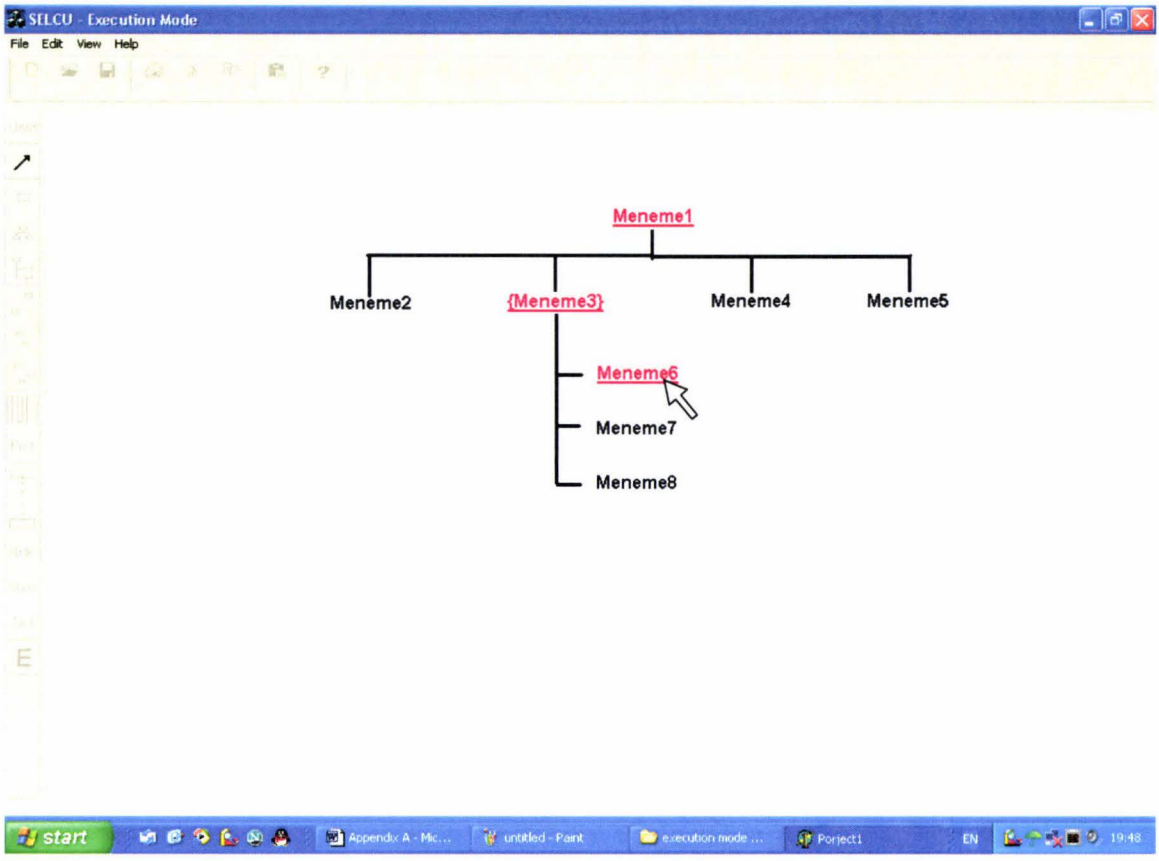


Figure A-6: Meneme Preview Interface

(3) Release the right mouse button. The system returns to Meneme view mode. The object meneme (***Meneme 6***) and related menemes (***Meneme 3 and Meneme 1***) become black, and the underlines disappear. The interface returns to the state shown in figure A-4.

Meneme Selection

A meneme can be selected as follows:

(1) If the execution environment is not in meneme view mode, select the ***Menemes Only*** submenu from the ***View*** pull down menu.

(2) Move the cursor to the object meneme (***Meneme 6***) and, single click the left mouse button. The object (***Meneme 6***) and related menemes (***Meneme 3 and Meneme 1***) change to red. Note that there is no underline on any meneme. Figure A-7 shows the meneme selection interface.

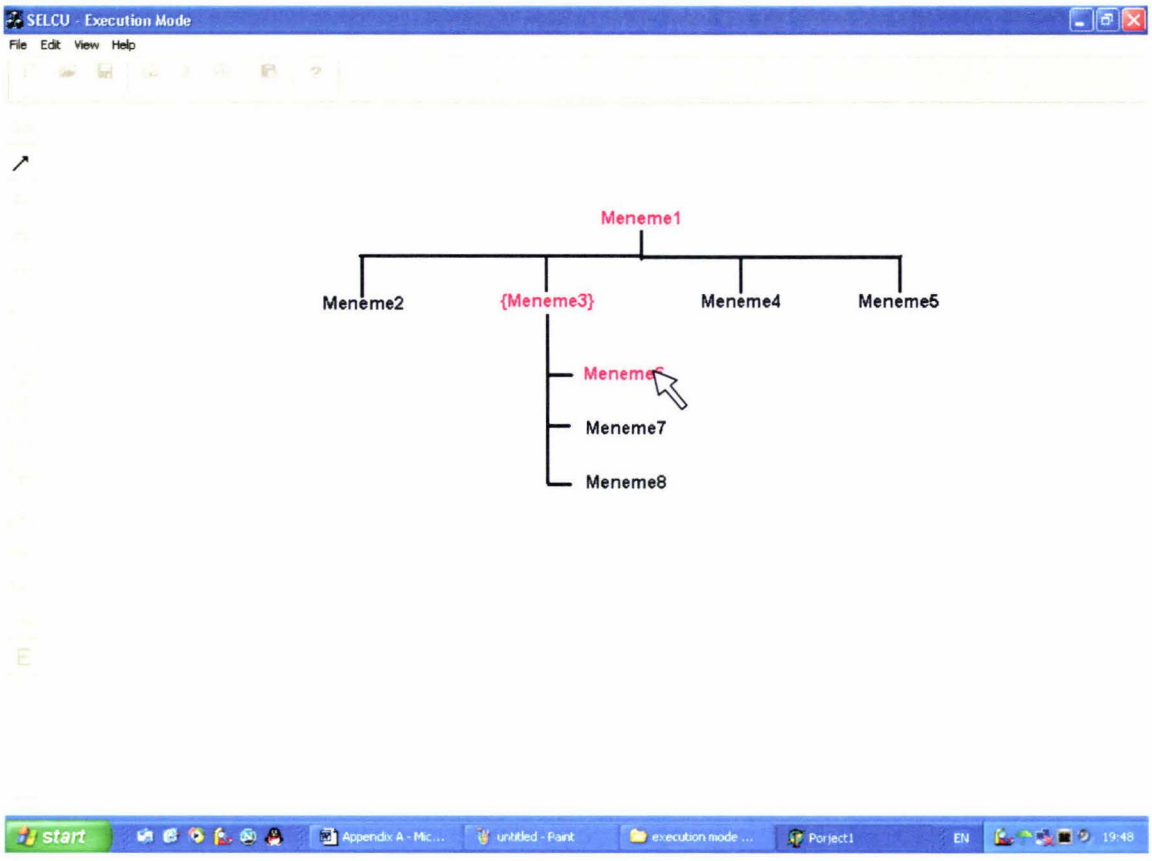


Figure A-7: Meneme Selection Interface

Task Execution

To execute a task, proceed as follows:

(1) If the execution environment is not in Task view mode, select the ***Tasks...*** submenu from the ***View*** pull down menu.

(2) A ***Task Display*** dialogue box pops up. The screen is shown in figure A-8.

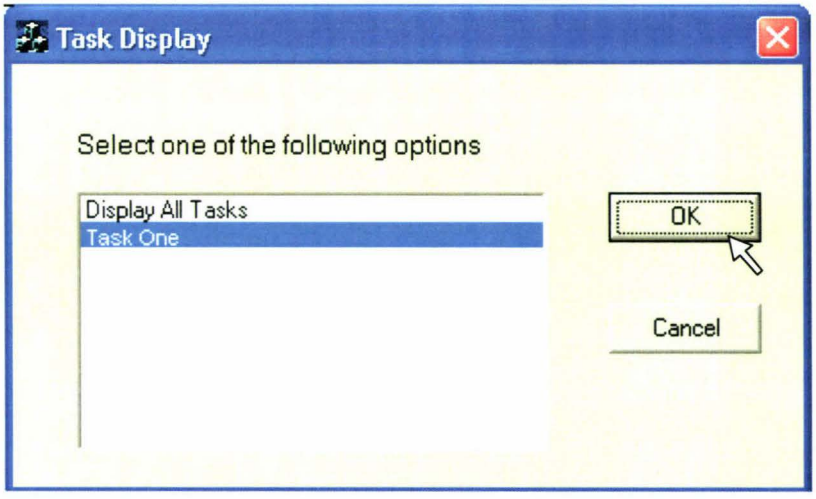


Figure A-8: Task Display Dialogue

(3) Select an item (***Task One***) from the ***Task*** list, and click the ***OK*** button.

(4) The task name (***Task One***) appears on the work area. Two single control buttons (***Next button*** and ***Back button***) appear on the system toolbar. ***Next button*** () allows the user execute the next step of the task, and ***Back button*** () lets the user return to the last step of the task. At the first step of the task, only the ***Next button*** is available; and at the last step of the task, only the ***Back button*** appears. Figure A-9 presents a task execution step screen snapshot.

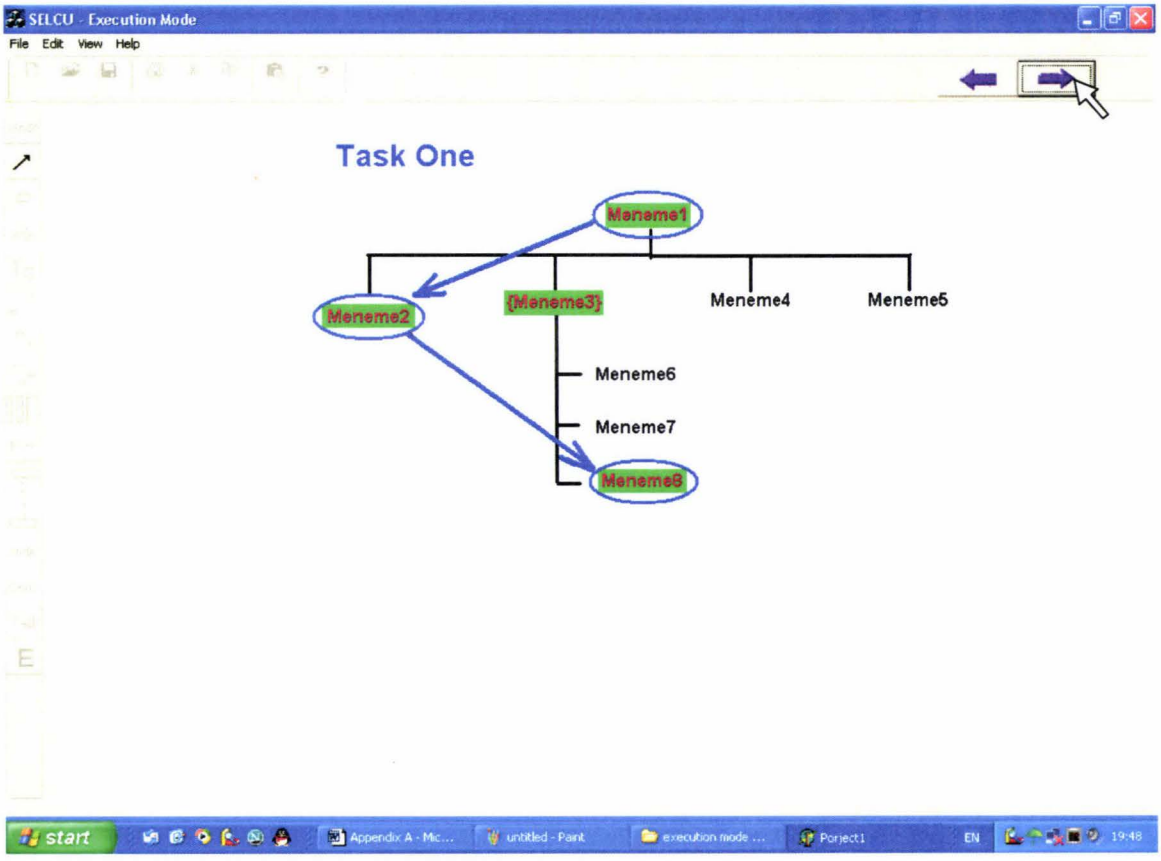


Figure A-9: Task Execution Interface

As menemes are selected, they are highlighted (red name on a green background). Any menemes selected via propagation (e.g. ***Meneme 3*** in figure A-9) are also highlighted.

Trigger View

All the triggers can be displayed by selecting the ***Triggers*** submenu from the ***View*** pull down menu. Figure A-10 shows the triggers view interface.

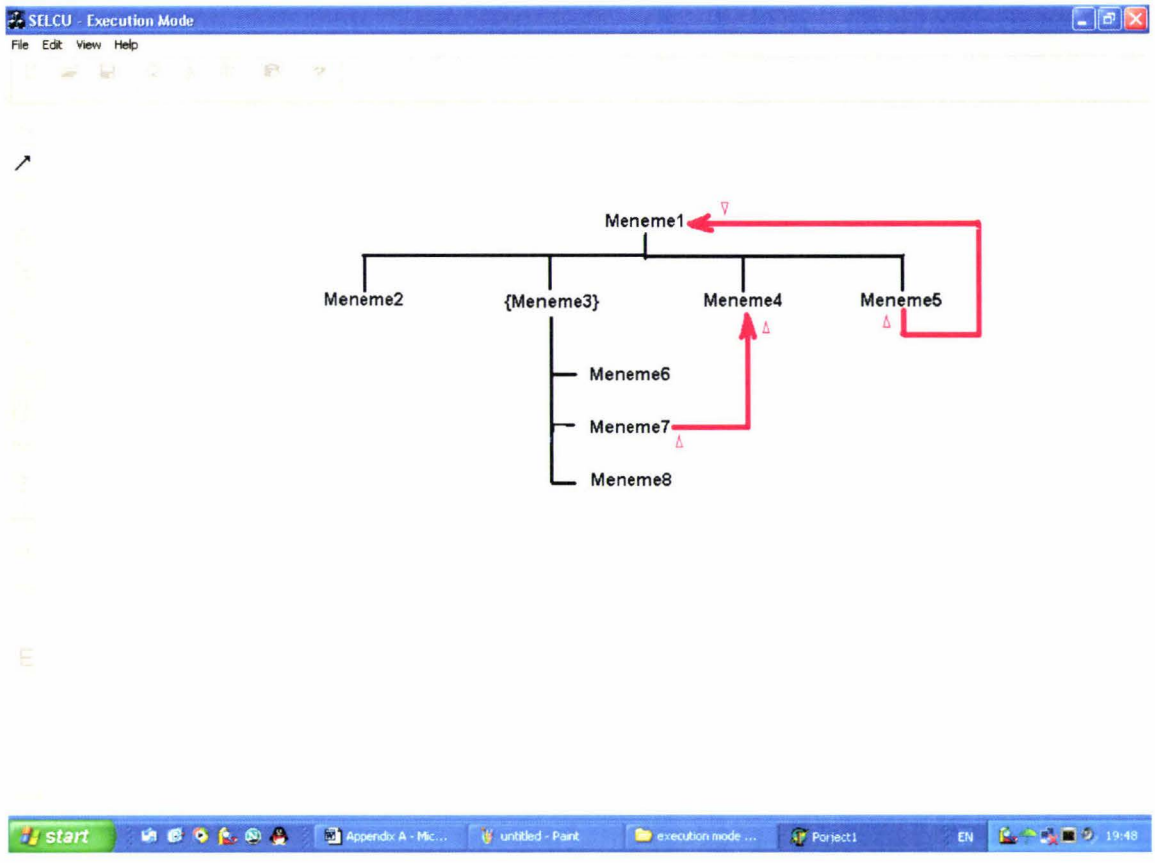


Figure A-10: Triggers View Interface

Exit Execution Mode

At anytime, the user can click the ***Quit*** button () on the SELCU toolbar to exit the execution mode, and return to the SELCU edit mode.