The Integration of Task and Dialogue Modelling in the Early Stages of User Interface Design

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

Christopher John Scogings

2003

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

CERTIFICATE OF REGULATORY COMPLIANCE

This is to certify that the research carried out in the Doctoral Thesis entitled *The Integration of Task and Dialogue Modelling in the Early Stages of User Interface Design* in the Institute of Information Sciences and Technology, Massey University, Palmerston North, New Zealand:

(a) is the original work of the candidate, except as indicated by appropriate attribution in the text and/or in the acknowledgements;

(b) that the text, excluding appendices does not exceed 100,000 words;

(c) all the ethical requirements applicable to this study have been complied with as required by Massey University, other organisations and/or committees which had a particular association with this study, and relevant legislation.

Candidate's Name: CHRIS SCOGINGS
Signature:
Date: 17.6.03

Supervisor's Name: CHRIS PHILLIPS
Signature:
Date: 17.6.03

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

SUPERVISOR'S DECLARATION

This is to certify that the research carried out for the Doctoral Thesis entitled *The Integration of Task and Dialogue Modelling in the Early Stages of User Interface Design* was done by Christopher John Scogings in the Institute of Information Sciences and Technology, Massey University, Palmerston North, New Zealand. The thesis material has not been used in part or in whole for any other qualification, and I confirm that the candidate has pursued the course of study in accordance with the requirements of the Massey University regulations.

Supervisor's Name: CHRIS PHILLIPS

Signature:

Date: 17.6.03

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

CANDIDATE'S DECLARATION

This is to certify that the research carried out for my Doctoral Thesis entitled *The Integration of Task and Dialogue Modelling in the Early Stages of User Interface Design* in the Institute of Information Sciences and Technology, Massey University, Palmerston North, New Zealand is my own work and that the thesis material has not been used in part or in whole for any other qualification.

Candidate's Name: CHRIS SCOGINGS

Signature:

Date: 17.6.03

Abstract

In the early stages of the design of graphical user interfaces, models and notations are required for describing user tasks and for describing the structure of the human-computer dialogue to support these tasks. These descriptions should ideally be linked, but in practice *task modelling* and *dialogue modelling* are conducted in isolation with differing notations. The research reported in this thesis reviews both task and dialogue modelling and describes how the divide between the two can be bridged via the Lean Cuisine+ notation.

Lean Cuisine+ was developed as a semi-formal graphical notation for describing the underlying behaviour of direct manipulation user interfaces (Phillips, 1995). The notation has a significant advantage over other dialogue modelling notations in that it can represent task sequences within the context of the dialogue structure and hence bring dialogue modelling and task modelling together. The research describes how Lean Cuisine+ has been analysed and modified in order to simplify it and to make it more suitable for use with a supporting software tool.

There exists a significant gap between user interface design and the design of entire software applications. This is true even for the latest software engineering notation, UML, which has become the *de facto* industry standard. The thesis reviews UML and reveals its shortcomings regarding support for user interface design. The research suggests a solution for the problem by proposing a method for the early stages of user interface design that uses Lean Cuisine+, in order to combine both task and dialogue modelling, and is constructed to form an integral part of the overall software design process advocated by the authors of UML. The method is applied to two case studies.

A prototype Software Environment for Lean Cuisine+ with UML (SELCU) has been developed to enable the construction, storage, editing and printing of Lean Cuisine+ specifications, and to partially support the method. The thesis describes the design, implementation and use of this software support environment.

Finally, preliminary work by the author on the automatic generation of Lean Cuisine+ specifications is reported. This shows promise, and has been further developed recently in associated masters-level research in which extensions to SELCU have been implemented.

Acknowledgements

The completion of this thesis is due, in no small measure, to the unwavering support, guidance and patience of my supervisor Associate Professor Chris Phillips. During the lean years of high workloads and frustration he remained a source of inspiration and dedication to a task that was often pushed into the background. I wish to express my deepest thanks to him.

Thanks also to my second supervisor Dr Peter Kay for proofreading and suggesting corrections to the thesis.

I am grateful also to Professor Ken Milne and members of the Massey University Doctoral Research Committee for recognising that work was progressing, albeit slowly, and for twice adjusting the thesis submission deadline.

Finally, I wish to acknowledge the role played by Associate Professor Gavin Finnie and Mr Rob Dempster who, many years ago, suggested that a PhD thesis was something worth completing.

Contents

List of Figures

Chapter 1. Introduction

1.1 Motivation

The motivation for this research arose from what may appear to be three separate strands:

- the recognised gap between task modelling and dialogue modelling in the early stages of user interface design.

- the development and use of Lean Cuisine+ - a dialogue modelling notation, also capable of task modelling, developed at Massey University.

- the emergence of the Unified Modelling Language (UML) as the *de facto* industry standard leading to the widespread use of UML for the design of large software applications.

The unique aspect of this research is that it combines these three areas. Each area is introduced in sections 1.2, 1.3 and 1.4 and the objectives of the research are outlined in section 1.5. Section 1.6 lists the approach adopted for the compilation of the thesis, section 1.7 lists the research publications already produced and section 1.8 presents the structure of the thesis.

1.2 Linking task and dialogue modelling

The primary purpose of task models is to define the activities of the user in relation to the system, as a means of uncovering the functional requirements to be supported by the system.

However, ongoing experimental research (Terwilliger & Polson, 1997) has demonstrated that users do not often identify with tasks that are modelled using traditional notations and that there are difficulties with the elicitation of task models in model-based user interface design (Tam, Maulsby & Puerta, 1998). Various attempts have been made in order to address these problems and several task description methods and notations have been developed, often involving graphical techniques.

The traditional task modelling methods focus primarily on task decomposition and are top-down whereas later approaches, such as use cases, are concerned primarily with task flow and are bottom-up. Many of them describe the activities of both the user and the system.

A CHI'98 workshop was convened to focus on the production of a method and notational framework to support task analysis and modelling (Artim, van Harmelen, Butler, Gulliksen, Henderson, Kovacevic, Lu, Overmeyer, Reaux, Roberts, Tarby & Van der Linden, 1998). It noted the heavy use of task modelling among software designers via both scenarios and use cases. Task models form an important input to user interface design as the frequency of tasks and the objects manipulated by tasks can have a direct bearing on both the structure and appearance of the interface. The use of task models in interface design is currently limited however by a lack of tools to support their definition and a lack of any linkage to dialogue (Bomsdorf & Szwillus, 1999).

Dialogue models have been developed as a means of capturing information about the behaviour of the user interface at a level above the "look and feel". At the early stage of user interface design, the designer requires models and notations which can assist with analysis of dialogue structure, and which are unconstrained by implementation issues. An analysis of several pieces of software that were supposed to assist their users showed that the user interface can, in fact, obstruct and frustrate the user in the use of the system (Jacquot & Quesnot, 1997) and hence it is necessary to focus on the *structure* of the dialogue. Chrusch (2000) points out that many people misinterpret the visual design of an interface as the interface itself, but this ignores the interaction sequence needed to complete a task. Research suggests that designers in the early stages of interface design should focus on dialogue structure and behaviour and not on the arrangement of buttons and widgets (Landay & Myers, 1995; Jaaksi, Aalto, Aalto & Vatto, 1999). Dialogue descriptions can also be used to analyse the interface for efficiency and completeness and they have formed the basis for various attempts at the automatic generation of user interfaces.

A criticism of existing dialogue notations is that the linkage to tasks is often obscure (Brooks, 1991). They define the structure and behaviour of the dialogue but they do not always readily reveal the interconnections between dialogue components during the execution of higher-level tasks. That is, they do not represent tasks within the *context* of the dialogue structure. Ideally, task models and dialogue models should be linked.

1.3 Lean Cuisine+

Lean Cuisine (Apperley & 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 actions, objects, states and other attributes. Lean Cuisine offers a clear, concise and compact graphical notation for describing the *structure* of menu-based dialogues.

Lean Cuisine has been expanded into the Lean Cuisine+ notation (Phillips, 1995) which is an executable semi-formal graphical notation for describing the underlying behaviour of event-based direct manipulation user interfaces. Lean Cuisine+ is a multi-layered notation. The base layer – the tree diagram retained from Lean Cuisine – captures part of the behaviour of a direct manipulation interface in terms of constraints and dependencies between selectable primitives (menemes). Further constraints and dependencies associated with the dialogue are captured through overlays, where events are explicitly shown. The task layer, in particular, defines *task sequences* and provides a task-oriented view of the interface. Menemes are linked together into a task sequence that indicates the order in which they must be selected.

The Lean Cuisine+ notation has several basic strengths in that it has been specifically developed as a user interface design tool, it incorporates the structure of the interface dialogue and it fits easily into an object-oriented approach. It is currently the only notation in which task sequences can be represented in the context of the structure of the interface dialogue. Lean Cuisine+ thus combines both dialogue modelling and task modelling. This allows rapid analysis of whether the dialogue supports all the actions required for a task, and whether it can be efficiently executed. A further advantage of the Lean Cuisine+ approach of defining tasks as a sequence of events is that the sequence is made concrete and can be checked for accuracy.

1.4 The Unified Modelling Language (UML)

UML is now emerging as the software industry's dominant modelling language (Kobryn, 1999) and has been described in numerous books and publications (e.g. Muller, 1997; Alhir, 1998; Eriksson & Penker, 1998; Odell, 1998). UML is not a proprietary notation – it is freely available to all. The creators of UML take great care to point out that the notation is a *language* and not a *method*. Thus existing methods can be retained in which the UML notation can replace the original notation in order to achieve standardisation of symbols. UML is designed to be readable on a large variety of media including paper, whiteboards and computer displays. UML is not a rigid notation and extensions and tailoring are encouraged to suit particular design needs. Associated with UML is the *Unified Process* (Jacobson, Booch & Rumbaugh, 1999) that consists of a set of concepts and activities needed to transform requirements into a software system. The Unified Process is not a detailed, step-by-step method, but rather a framework that can be adapted to suit local conditions.

Prior to the introduction of UML, it had already been established that there was a lack of integration between software engineering and HCI methods (Jacquot & Quesnot, 1997). Janssen, Weisbecker & Ziegler (1993) make the point that user interface tools cannot make use

of the models developed with general software engineering methods and tools. McDaniel, Olson & Olson (1994) state that software design and user interface design must become forged into one methodology. Kemp & Phillips (1998) show that support for user interface design is weak within object-oriented software engineering methods.

As UML is primarily a collection of previously defined notations, no significant change has occurred in regard to user interface design. This is noticeable in many of the books on how to use UML. The approach taken by (Eriksson & Penker, 1998) is typical in that they state that the design of the user interface should be carried out "separately but in parallel" to other design work. They then opt out completely by stating that user interface design is beyond the scope of a book about designing software using UML. Other books on UML, for instance (Fowler & Scott, 2000; Muller, 1997), do not even have an entry for "user interface" in the index. The creators of UML themselves make the point that UML is a general purpose modelling language, and specialised domains, such as user interface design, may require a more specialised tool with its own specialised language (Rumbaugh, Jacobson & Booch, 1999). Constantine & Lockwood (2001) state that notations originally developed to support general software design are not necessarily well suited for modelling the user interface or the interactions between users and the user interface.

1.5 Objectives of the research

The principal objective of this research is to establish a method that can be used in the early stages of user interface design. The method must incorporate both task and dialogue modelling using a common notation. As user interface design is one part of a much larger process – the development of a software application – it is important that the method be situated within the framework of a widely used software design process, such as UML.

A number of secondary objectives have also been identified. These are:

- an evaluation of the level of support provided by UML and the Unified Process for user interface design, and the development of extensions to increase this;
- the development and implementation of a software support environment for the Lean Cuisine+ notation;
- the exploration of further uses for Lean Cuisine+ including the automatic generation of Lean Cuisine+ diagrams from existing user interfaces constructed with standard interface builders.

1.6 Approach adopted

The approach to achieving the objectives set out above can be summarised as follows:

- A review of task modelling is presented. This includes traditional task modelling as well as the newer scenario-based and use case-based models. This is followed by a review of dialogue modelling that includes formal methods and graphical notations, including Lean Cuisine. The nature of the gap between task and dialogue modelling is established.

- The Lean Cuisine+ notation is examined and the development of Lean Cuisine+ from Lean Cuisine is traced. Changes to the notation are suggested and justified, and a revised version of Lean Cuisine+ is presented. The usefulness of Lean Cuisine+ as a task modelling notation is established. A case study is presented and analysed in order to demonstrate that Lean Cuisine+ can be used to model both dialogue and tasks.

- An overview of the UML notation is presented, followed by an introduction to the *Unified Process*, which is a set of concepts and activities needed to complete the design of large software applications. It is established that UML provides insignificant support for user interface design and a way forward is proposed.

- A method is presented which uses Lean Cuisine+ to bridge the gap between task and dialogue modelling at the early stages of user interface design. This use of Lean Cuisine+ is positioned within the general software design framework enabled by UML and the Unified Process. The method is presented via two contrasting case studies.

- The software support environment for Lean Cuisine+ (SELCU) is presented. Functional requirements are established and design issues are discussed. A description of the SELCU package is provided and a step-by-step case study presented, illustrating how the system can be used to construct Lean Cuisine+ diagrams.

- Further applications of Lean Cuisine+ are investigated. These include the automatic generation of Lean Cuisine+ diagrams from existing user interfaces and the execution of Lean Cuisine+ specifications.

1.7 Research publications

This research has already resulted in the publications listed below.

Phillips C.H.E. & Scogings C.J. (1995) Task and Object Modelling in high level GUI Design: An integrated approach. *Proceedings OZCHI'95.* Wollongong, Australia. 24-29.

Phillips C.H.E. & Scogings C.J. (1997) Modelling the mock-up: towards the automatic specification of the behaviour of early prototypes. *Proceedings INTERACT '97.* Sydney, Australia. 591-592.

Phillips C.H.E. & Scogings C.J. (1998) Towards the automatic specification of the behaviour of early prototypes using Lean Cuisine+. *Proceedings Software Engineering & Practice.* Dunedin, New Zealand. 238-244.

Scogings C.J. & Phillips C.H.E. (1998) Beyond the interface: modelling the interaction in a visual development environment. *Proceedings HCI '98.* Sheffield, United Kingdom. 108-109.

Scogings C.J. (2000) The Lean Cuisine+ notation revised. *Research Letters in the Information & Mathematical Sciences* 2000 (1). Massey University. Auckland, New Zealand. 17-23.

Phillips C.H.E. & Scogings C.J. (2000) Task and Dialogue Modelling: Bridging the Divide with Lean Cuisine+. *Proceedings AUIC 2000.* Canberra, Australia. 81-87.

Scogings C.J. & Phillips C.H.E. (2001a) A method for the early stages of interactive systems design using UML and Lean Cuisine+. *Proceedings of AUIC 2001.* Gold Coast, Australia. 69-76.

Scogings C.J. & Phillips C.H.E. (2001b) Linking tasks, dialogue and GUI design: A method involving UML and Lean Cuisine+. *Interacting with Computers* 14 (1). 69-86.

Phillips C.H.E., Kemp E.A. & Scogings C.J. (2001) Extending UML to support user interface design. *Proceedings of ACM symposium on CHI.* Palmerston North, New Zealand. 55-60.

Scogings C.J. & Phillips C.H.E. (2003) Linking task and dialogue modelling: Toward an integrated software engineering method. In Diaper D. & Stanton N. (Eds.) *Handbook of Task Analysis for Human-Computer Interaction*. TechBooks. Fairfax, Virginia. [In print].

1.8 Structure of the thesis

The thesis is arranged in eight chapters. Chapter 1 is this introductory chapter. Chapter 2 reviews task and dialogue modelling and includes the major portion of the literature survey. Chapter 3 traces the development of Lean Cuisine+ and the later revisions made to the notation. Chapter 4 provides an overview of UML and an introduction to the Unified Process. It also presents an analysis of the failure of UML to support user interface design.

Chapter 5 is the heart of the thesis and presents the method to integrate task and dialogue modelling at the early stages of user interface design. This method uses Lean Cuisine+ to combine task and dialogue modelling and it is placed within UML as a useful and desirable addition to the Unified Process. Chapter 6 describes SELCU, the software environment developed to support the Lean Cuisine+ notation. It includes a detailed description of how a typical Lean Cuisine+ diagram is constructed.

Chapter 7 describes two other uses of Lean Cuisine+. The first of these demonstrates how it is possible to automatically generate Lean Cuisine+ diagrams from existing user interfaces developed with commercial interface builders. The second describes extensions made to the SELCU package to enable the execution of Lean Cuisine+ specifications, thus creating a "working model" of the user interface. The execution of specifications was not part of the project culminating in this thesis, but was work carried out as a separate project in partial fulfilment of a Masters degree at Massey University – see (Li, 2003). Chapter 8 examines the contribution of the research and identifies further work.

Chapter 2. A Review of Task and Dialogue Modelling

"This discussion on user interface models is not an academic exercise in the theoretical design of user interfaces. It is very real and describes the failure of many products that may have contained incredible technological and functional elements but were otherwise lacking in their attention to the user interface. These models should be understood by all those who participate in the development and use of software products."

Mandel, 1997

2.1 Introduction

Over the past two decades considerable effort has been devoted to the development of models and tools to support the analysis and design of software systems. This has resulted in the development of computer-aided software engineering (CASE) tools and a large number of analysis and design methods, although the leading object-oriented methodologies have now been brought together through the Unified Modelling Language (UML) (Quatrani, 2000). The primary focus of such methods has been on support for the design and implementation of the software comprising the underlying system, referred to in (Collins, 1995) as the *internal system*. Very little support is provided for the design of the *external system* or user interface.

The design of the user interface has assumed increasing importance as Graphical User Interfaces (GUIs) have become dominant, and as more and more attention has been devoted to usability aspects of interactive systems. User interface design requires good descriptive systems and the development of tools to support them (Artim *et al*, 1998). In particular, models and notations are required for describing user tasks, and for describing the structure of the human-computer dialogue to support these tasks.

Task models focus either on task decomposition or task flow. Their primary purpose is to define the activities of the user in relation to the system, as a means of uncovering the functional requirements to be supported by the system. Dialogue models have been developed as a means of capturing information about the behaviour of the user interface at a level above the "look and feel". These models should ideally be linked. A criticism of existing dialogue notations is that the linkage to tasks is often obscure (Brooks, 1991). They define the structure and behaviour of the dialogue but they do not always readily reveal the interconnections between dialogue components during the execution of higher-level tasks. That is, they do not represent tasks within the context of the dialogue structure.

This chapter reviews existing task modelling and dialogue modelling notations and techniques. Section 2.2 reviews task modelling in which use cases and scenarios are emphasized. Section 2.3 reviews dialogue modelling and the automatic generation of user interfaces. Section 2.4 covers dialogue modelling with Lean Cuisine and section 2.5 introduces Lean Cuisine+.

2.2 Task modelling

It is clear from ongoing experimental research (Terwilliger & Polson, 1997) that designers have difficulty in modelling tasks in a form that users feel comfortable with. Tam, Maulsby & Puerta (1998) note that the eliciting of task models is a "thorny problem" in model-based interface design. In order to address these problems, a variety of task description methods have been developed, often involving graphical techniques. The traditional methods focus primarily on task decomposition and are top-down whereas later approaches, such as use cases, are concerned primarily with task flow and are bottom-up. Many of them describe the activities of both the user and the system.

Traditional task modelling

The traditional approach to task modelling is through hierarchical decomposition where the task is described using some form of tree diagram or structured text. Hierarchical task analysis (HTA) is typical of this approach and is reviewed in (Hackos & Redish, 1998). The outputs of HTA are a hierarchy of tasks, sub-tasks and plans describing in what sequence and under what conditions sub-tasks are performed. Functional decomposition, however, can produce elegant models that have little to do with the realities of how work is accomplished or how users think about what they are doing (Constantine & Lockwood, 1999).

In the GOMS model (the acronym stands for Goals, Operators, Methods, Selection rules) the tasks are initially described as a set of *goals* and sub-goals, in a manner similar to HTA. At the required level each sub-goal is described in terms of the operations required to achieve it. An operation (or *operator*) is an elementary action (possibly cognitive) such as a button press or a menu selection. *Methods* are well-learned sequences of sub-goals and operators that can accomplish a goal and *selection rules* are the personal rules that users follow in deciding what method to use in a particular circumstance.

The GOMS approach has been criticised for not providing detailed information on how to use the notation and for appearing clumsy to use. Furthermore, the notation has only a weak

connection to the underlying theory and concentrates on expert users and error-free performance (Shneiderman, 1998). Various types of what are called "GOMS family techniques" are listed in (John & Kieras, 1996) and the designer needs to select the right model to match the type of design work at hand.

Apart from HTA and GOMS, there are several graphical notations applied to task description. These include event trees, decision-action diagrams and Petri nets. A comprehensive review is provided in (Kirwan & Ainsworth, 1992).

Scenarios

Constantine & Lockwood (1999) describe a scenario as a narrative description of an activity or activities, taking the form of a story, a vignette, or an episode bound in time and taking place within a given context. A good overview of task modelling with scenarios is provided in (Carroll, 1995) which shows that scenarios, in various forms, are widely used in software design and that there are two distinct roles for scenarios. One is in supporting the generation of design ideas and the other is in evaluating a proposed design.

Filippidou (1998) provides a useful review of the use of scenarios in practice and shows that scenarios are extensively used to understand and explain task behaviour. Four scenario perspectives are defined:

- *process perspective*: a sequence of actions or events
- *simulation perspective*: a focus on a situation or an episode
- *choice perspective*: the description of several options
- *use perspective*: the description of how users will use a proposed system

These perspectives are not mutually exclusive and are often combined.

Filippidou (1998) also provides a comprehensive review of several scenario-based approaches to task analysis. In conclusion, a list of requirements for the improved use of scenarios is outlined. This includes the need for general agreement on the definition of scenarios, a concrete vocabulary to enable discussion of task processes, and visualisation techniques to allow different views of scenarios.

Rolland, Ben Achour, Cauvet, Ralyte, Sutcliffe, Maiden, Jarke, Haumer, Pohl, Dubois & Heymans (1998) point out that a wide range of examples, scenes, narrative descriptions, mock-ups and prototypes can all be called *scenario-based approaches.* A scenario classification framework is proposed, in which scenarios are classified according to description, presentation, content, area of coverage, level of abstraction, context, role (similar to Filippidou's perspectives) and lifespan. Several scenario-based approaches are listed and then classified according to the framework. The conclusion mentions the difficulties related to the absence of formal definitions. It should also be noted that the level of detail and complexity in the proposed classification framework would probably make it unpopular with practitioners.

Scenarios have the great advantage that designers and users alike understand them and thus they form a valuable link between software producers and their clients (Sommerville & Sawyer, 1997). In particular, clients are comfortable with scenarios, as their specification and analysis is immediately recognised as a concrete and familiar design task (Rosson & Carroll, 1995a). Although scenarios typically take the form of continuous narrative, they can also be produced as visual images, for example in the form of storyboards. Rosson & Carroll (2001) have developed a general framework for scenario-based design in which scenarios are used as the only method of task modelling.

A drawback of scenarios is that they remain ill defined. No exact definition exists and may vary from scenario to scenario within the same design problem. There is also little consensus about the appropriate level at which to couch scenario descriptions (Carroll, 1995). The scenario-based design of a complex system rapidly leads to the problem of how to fruitfully deal with a large collection of weakly structured texts. One novel approach is to automatically structure document collections into two-dimensional maps to indicate relationships among scenarios (Becks & Koller, 1999).

Scenario-based development methods are still evolving, and the relationship between task scenarios and object-oriented software development has been explored elsewhere (Rosson & Carroll, 1995b).

Conventional scenarios have some serious limitations when it comes to user interface design. Their emphasis on realism and rich detail can obscure broad issues and general organization. Because scenarios focus on plausible combinations of individual tasks or activities, they can make it more difficult to present an overall view of the interface (Constantine & Lockwood, 1999).

Use cases

Use cases, which are abstract task scenarios, were developed in connection with object-oriented software engineering as a way of uncovering the functionality of a system (Jacobsen, Ericsson & Jacobson, 1995). They model user actions and system responses, and involve the manipulation of objects and the sequencing of interactions. Use cases have been so successful that they have been integrated into virtually every major approach to object-oriented analysis and design (Constantine & Lockwood, 1999). Each use case should describe an interaction that is complete, well defined and meaningful to the user.

However, in the context of object-oriented design, Stevens & Pooley (2000) sound a note of caution when modelling with use cases because:

- there is a danger of building a system that is not object-oriented. Use cases explore the functionality of the system and designers must ensure that they do not lapse back into developing function-oriented software.

- there is a danger of mistaking design for requirements. Users are likely to describe a use case as a very concrete sequence of interactions which is one way, but not the only way, of achieving the underlying goal.

Another problem is that conventional use cases often contain too many built-in assumptions about the form of the user interface that is yet to be designed. As models, they can tend too closely towards implementation. This has led to the introduction of the concept of an *essential use case* (Constantine, 1995) that is demonstrated here by example.

Figure 2.1 shows the use case for getting cash from an automatic teller machine (ATM). This use case assumes that bank cards with magnetic strips are being used, that there is a visual display and that the user will be using a keyboard. Figure 2.2 shows the corresponding essential use case that is based on the *intentions* of the user rather than on the concrete steps that might be carried out. The essential use case leaves open many more possibilities for the design and implementation of the user interface. Biddle, Noble & Tempero (2001) show that essential use cases can be used to derive class models without the need for any conventional use cases, thus saving design time.

Use Case: Getting cash

USER ACTION	SYSTEM RESPONSE
insert card	
	read magnetic strip
	request PIN
enter PIN	
	verify PIN
	display transaction option menu
press key	
	display account menu
press key	
	prompt for amount
enter amount	
	display amount
press key	
	return card
take card	
	dispense cash
take cash	

Figure 2.1: An example use case (Constantine & Lockwood, 1999)

The progression from very specific scenarios to abstract use cases has led, in a circular path, to the redefinition of the term scenario. "A use case is a collection of scenarios, bound together by a common goal. Each scenario is a trace of actions and messages from trigger to goal completion or abandonment, non-branching but possibly with parallelism." (Cockburn & Fowler, 1998). This definition would probably lead to a debate over the semantics but there is general convergence in this direction.

Essential Use Case: Getting cash

USER INTENTION	SYSTEM RESPONSIBILITY
identify self	
	verify identity
	offer choices
choose	
	dispense cash
take cash	

Figure 2.2: The corresponding essential use case (Constantine & Lockwood, 1999)

Use cases, like scenarios, are not well defined. The first problem is that there is no consensus on their scope or size. The view of Cheesman & Daniels (2001) is that a use case should be smaller than a business process but larger than a single operation on a single software component. Secondly, there is no agreed method of presenting use cases and work in progress is attempting to formalise this. In their original form, use cases usually consist of textual

descriptions (Richter, 1999; Fowler & Scott, 2000). However a large project is liable to generate many use cases and it is useful to use a standard template. One such template (Jaaksi, Aalto, Aalto & Vatto, 1999) is illustrated here in Figure 2.3.

Use Case:	Request Book
Actor:	Library User
Goal:	Permits a library user to request an unavailable book
Preconditions:	Book details are being displayed and book status is "unavailable"
Description:	At any time the library user may request a printout of book details or cancel the book request. The library user selects a copy of the book and submits the request. The system then identifies the user as an authorised library user. If this is successful, the system confirms the request and the use case ends.
Exceptions:	If the identification of the user fails, the use case ends.
Postconditions:	None.

Figure 2.3: A typical template for description of a use case based on (Jaaksi *et al*, 1999)

A tabular use case representation is proposed in (Phillips, Kemp & Kek, 2001). This has some advantages over the textual description, particularly for user interface design, because it visually separates and clearly distinguishes between user and system actions and shows any interaction sequence. An example is provided in Figure 2.4.

There is no easy way to establish use cases. A useful process appears in (Biddle, Noble & Tempero, 2000):

- identify *actors* – the people and other systems that will use the system.
- list *candidate use cases* for each actor – the required functions that are immediately obvious to each actor.
- identify *focal use cases* – candidate use cases are often small and numerous. Focal use cases are large use cases that cover the main responsibilities of the system.
- create *use case diagrams* – to show how actors and use cases are related.
- *check* the use case diagrams for missing actors and missing use cases.

A similar, though less succinct, process is discussed in (Eriksson & Penker, 1998) where it is noted that the process is highly iterative and each step has the potential for providing input to a previous step as well as to the succeeding steps.

Use Case:	Request Book
Actor:	Library User
Goal:	Permits a library user to request an unavailable book
Preconditions:	Book details are being displayed and book status is "unavailable"
Postconditions:	None

Library User	**System**
Main Flow At any time, may: - request a printout of book details (S1) - cancel the book request (S2) Select a copy of the book and submit the request.	 Identify Library User (E1) Confirm the request. The use case ends.
Subflows (S1) Request a printout of book details	 Print book details. The use case continues.
(S2) Cancel the book request	 The use case ends.
Exception Flows (E1) User enters an invalid ID	 Display a message. The use case ends.

Figure 2.4: A tabular representation of the use case described in Figure 2.3

A CHI'98 workshop was convened to focus on the production of a method and notation framework to support task analysis and modelling (Artim *et al*, 1998). It noted the heavy use of task modelling, among software designers, via both scenarios and use cases. Task models form an important input to user interface design as the frequency of tasks and the objects manipulated by tasks can have a direct bearing on both the structure and appearance of the interface. The use of task models in interface design is currently limited however by a lack of tools to support their definition and a lack of any linkage to dialogue (Bomsdorf & Szwillus, 1999).

2.3 Dialogue modelling

At the early stage of user interface design, the designer requires models and notations that can assist with the analysis of dialogue structure and that are unconstrained by implementation issues. Research indicates that designers in the early stages of interface design should focus on dialogue structure and behaviour and not on the arrangement of buttons and widgets (Landay & Myers, 1995; Jaaksi *et al*, 1999). Jacquot & Quesnot (1997) have found that it is necessary to focus on the structure of the dialogue because the analysis of several pieces of software that were supposed to assist their users showed that "interfaces actually get badly in the way." Chrusch (2000) points out that many people misinterpret the visual design of an interface as the interface itself, but this ignores the interaction sequence needed to complete a task. Dialogue descriptions can also be used to analyse the interface for efficiency and completeness and they form the basis for various attempts at the automatic generation of user interfaces.

A variety of models at various levels of abstraction have been employed in the early stages of interface design, involving both graphical and textual notations. As part of the development of Lean Cuisine+, described in Chapter 3, a review of graphical dialogue notations was carried out (Phillips, 1994) and shortcomings were uncovered in the notations reviewed in relation to describing direct manipulation GUIs. In particular, these notations had difficulties either in the capturing of event sequences, or in relating event sequences to tasks.

Graphical notations

State Transition Diagrams

State Transition Diagrams (STDs) are based on a set of nodes with links between them. The nodes represent all possible states of the system and the links represent all possible transitions between the states. Some transitions may be conditional. STDs have been widely used for some time and are now showing their age. As a formal method, a STD should show every possible transition from every possible state and this means that for large, complex interfaces the diagram rapidly becomes unwieldy and hard to follow. This problem has been exacerbated by the trend in interfaces to be "modeless" or "mode free" in order to maximise the options available to a user at any one time because this increases the number of possible transitions from a particular state. In addition, they do not handle unexpected user actions well although they can be useful in describing simple individual task sequences. STDs have been extensively reviewed as dialogue modelling tools and found wanting (Phillips, 1991; Janssen, Weisbecker & Ziegler, 1993; Myers, 1995; Shneiderman, 1998).

Figure 2.5 illustrates the STD for the Style menu from a simple word processor. The Style menu at top left shows how the menu actually appears in the user interface. According to Apperley & Spence (1989), highly interactive dialogues, such as the Style menu, have little or no sequence - they are said to be *asynchronous*. They have no obvious start or end point as the user can select any option, or combination of options, at any time. The STD completely and accurately describes the behaviour of the Style menu but it has become complex and unwieldy, and its derivation from the known behaviour of the menu is neither obvious nor intuitive.

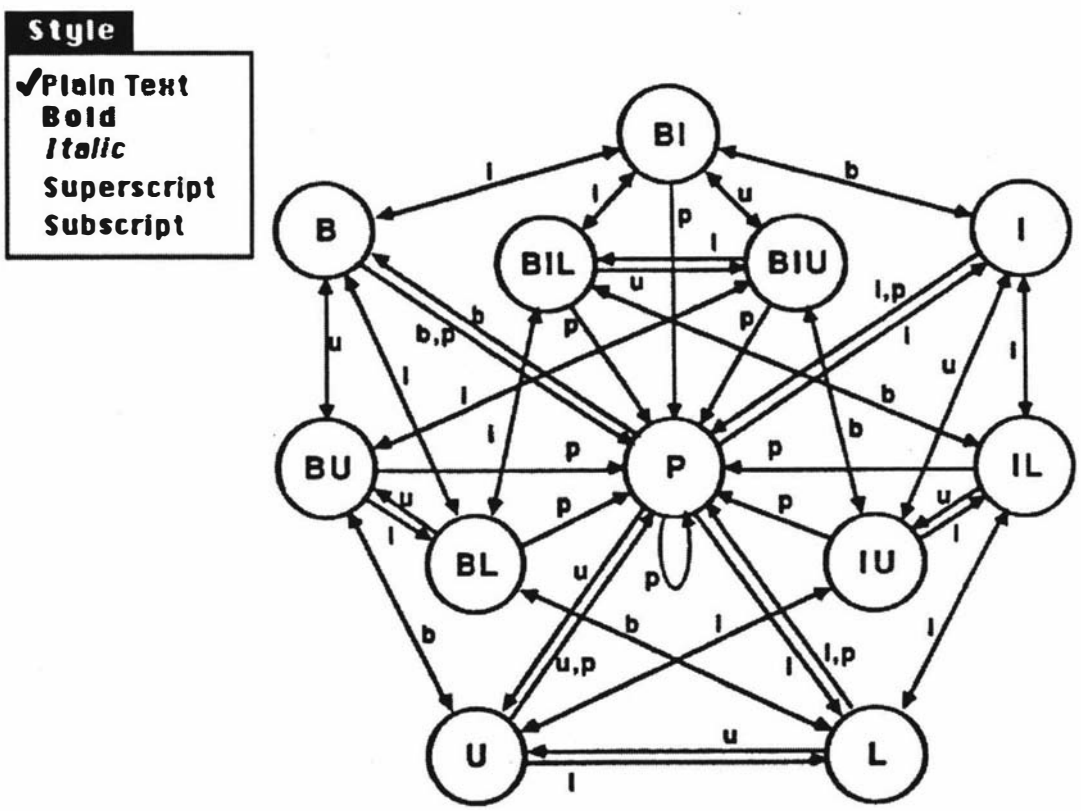


Figure 2.5: STD representing the Style menu (Apperley & Spence, 1989)

Petri nets

Petri nets are an alternative to STDs and are widely used to model concurrent systems and processes (Reisig & Rozenberg, 1998). A Petri net contains a set of *places* and a set of *transitions*. A Petri net *graph* is the graphical representation of the Petri net and follows the rules of graph theory (Peterson, 1981). It is possible to produce a graph that graphically consists of separate fragments – a *disconnected* (or unconnected) graph. When modelling with Petri nets, each node in the graph (a place in Petri net theory) is regarded as a *condition* and these are linked by the transitions (or events) that change the state of the system.

One of the strengths of the Petri net as a modelling tool is that it can be *executed* to analyse whether the system is functioning correctly and efficiently. Before execution can commence, tokens have to be positioned in the places that represent the "at rest" state of the system. The presence of a token indicates that that particular condition is true. The state of the system is defined as the combination of currently true conditions. A Petri net executes by *firing* transitions. When a transition fires, tokens are removed from its input places and new tokens are distributed to its output places.

Figure 2.6 illustrates the Petri net representation for the Style menu and should be compared with the STD representation of the same menu in Figure 2.5. A major improvement over STDs is that several conditions can exist independently, which is particularly useful for modelling concurrent processes. However, this also introduces further complexity as several additional conditions are required, for example "no B" to indicate that Bold is not selected. The system "at rest" state also needs to be defined. In the case of Figure 2.6, the "at rest" state is defined to be when only the conditions "no B", "no I" and "no UL" are true. This corresponds to the state "Plain Text" which does not explicitly appear in the Petri net graph.

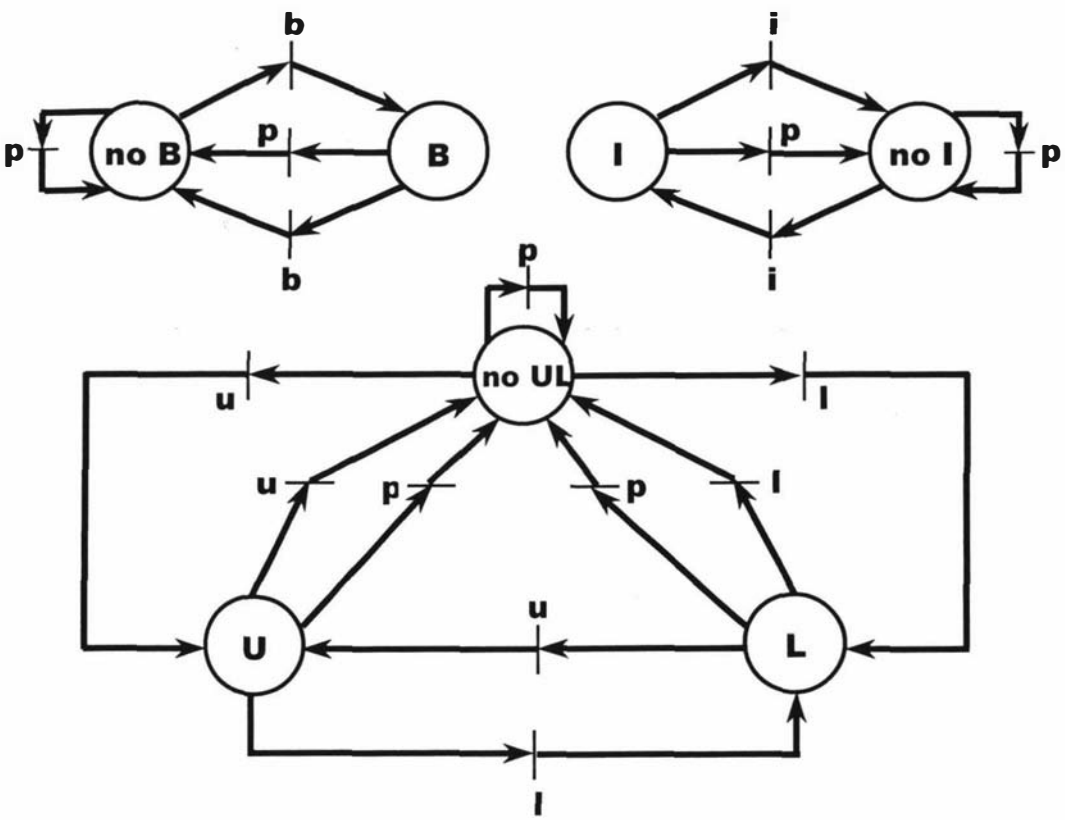


Figure 2.6: Petri net representing the Style menu

Although the Petri net representation of the Style menu is clearer than the STD representation, the need to show all transitions explicitly clutters and detracts from the diagrams and there is still no clear relation to the actual interface. Modelling with Petri nets has also been criticised for being too complicated since their design often requires a professional expert (De Michelis, 1999).

Software tools exist to support Petri nets and an overview of two of them is presented in (Li, 2003). *PetriSim* is designed to support discrete simulation and Petri nets and is based on the Pascal language. *HPSim* is a high-level Petri net simulator and editor using Java and it treats Petri nets as Java objects.

Statecharts

Statecharts were developed as a way of visually specifying complex reactive systems that are event driven and thus are continuously reacting to internal and external stimuli. They are an attempt to overcome the deficiencies of STDs in describing such systems, viz: STDs are flat (lack depth); STDs are uneconomical with respect to states and transitions; and STDs are inherently sequential. The extensions that statecharts provide over STDs: depth, concurrency and broadcast communication, make them useful for representing parallel processes and reduce the size of the final diagram for large systems (Dix, Finlay, Abowd & Beale, 1998). However, the need to show transitions explicitly remains, and is compounded to an extent by the requirement for null states to be shown, as is the case with Petri nets. This continues to pose problems despite the improvement over conventional STDs provided by the additional constructs.

Figure 2.7 shows the statechart representation of the Style menu and should be compared to the STD representation in Figure 2.5. The number of transitions has been reduced through the use of both state nesting (grouping) and state orthogonality (parallelism). The statechart description is much easier to relate to the menu it represents than both the STD and Petri net representations, showing more clearly the permissible user actions at each stage and their effect on the menu state.

A number of software tools exist to support statecharts and a review is presented in (Joy, 1999). These include BetterState from Integrated Systems Inc. (ISI) that is targeted towards embedded systems design. Statemate MAGNUM from I-Logix, developed in conjunction with David Harel, the inventor of statecharts, also targets embedded systems development. Stateflow from Mathworks is part of Matlab, but is a much simpler tool than the first two.

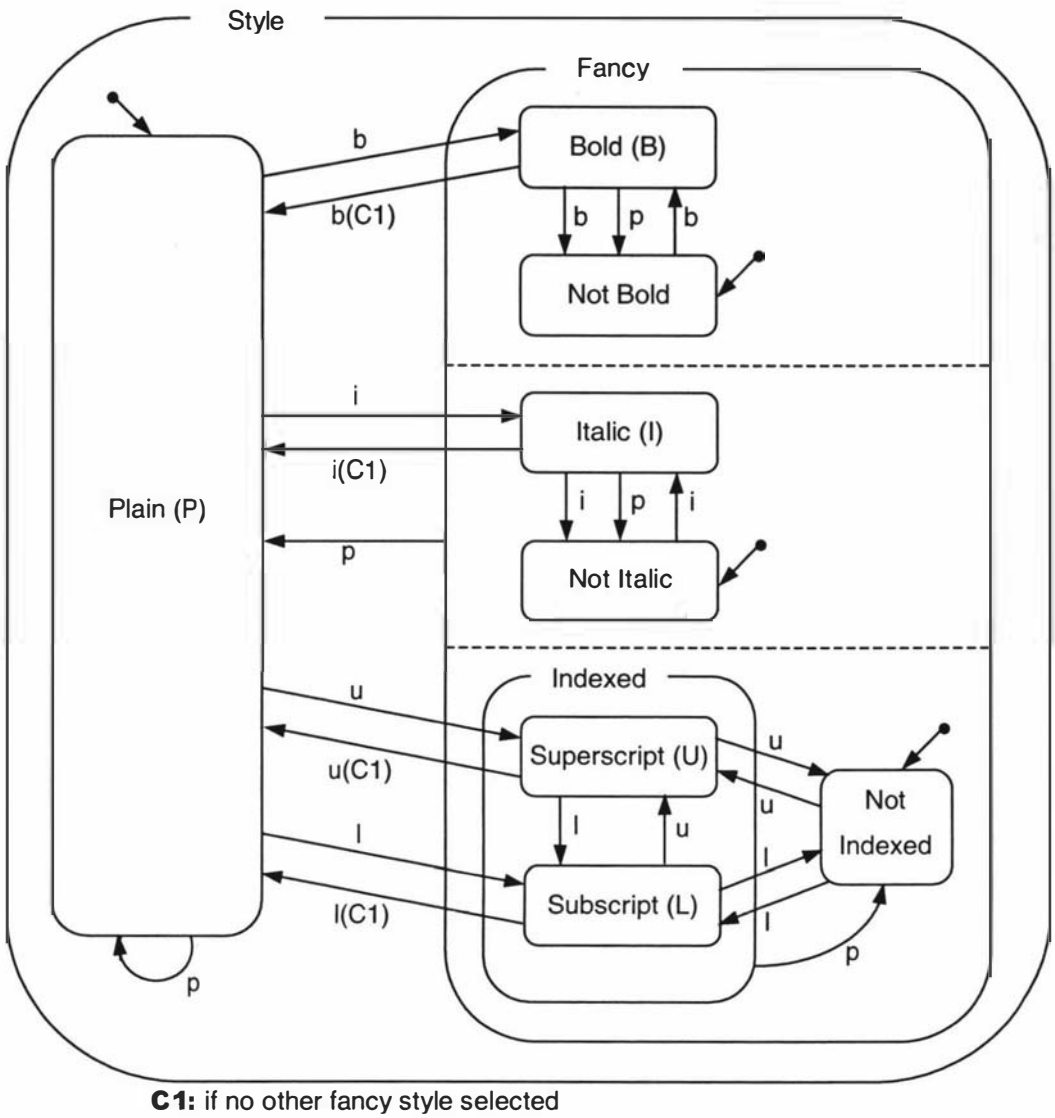


Figure 2.7: Statechart representing the Style menu

Formal languages

Dialogue modelling based on formal languages has failed (Myers, Hudson & Pausch, 2000). Preece (1993) states that the specification of dialogue using formal methods is complicated and liable to error. Rumbaugh (1995) notes that formal specification languages are too restricted to model common things without becoming tortuous. A major problem is the need for designers to learn the formal language before being able to use it. Designers are also assumed to be familiar with formal concepts such as grammars (Jaaksi *et al*, 1999). A further complicating factor is that the formal methods described above were not originally intended for dialogue modelling.

Automatic generation of user interfaces

Automatic generation of user interfaces is the "holy grail" for designers but attempts to achieve it have been dogged by the consistent failure to provide a formal notation to specify dialogue in such a way that translation to code is possible and designer choice is not restricted. Eisenstein & Puerta (2000) note that researchers have developed systems that are effective for narrowly focused domains including automatic generation of forms and automatic generation of dialog boxes for database access, but that no technique has been shown to be applicable at a general level. Dialogue modelling is usually viewed as the first step towards the automatic generation of interfaces.

The User Action Notation (UAN) (Hartson, Siochi & Hix, 1990) is a notation for describing direct manipulation user interfaces. It is used at an extremely low (less abstract) level, for example to specify mouse clicks, during interface prototyping. Since it is used at the "buttons and widgets" level of the interface, it does not fall into the area of dialogue modelling discussed here. It does have uses in checking for interface efficiency. The Dialogue Activation Language (DAL) (Anderson, 1995) uses UAN to describe particular interactions and then constructs an interface in which the specified interactions can be executed.

The Language for the Specification of Interfaces (LSI) (Jacquot & Quesnot, 1997) includes a dialogue model called a *workplan* that is a hierarchical tree structure. The LSI also provides formal syntactic rules for specifying the presentation of the interface and it can then generate an executable prototype.

Dialogue Nets (Janssen, Weisbecker & Ziegler, 1993) are based on Petri nets and are used to supply dialogue control information to GENIUS (Generator for user interfaces using software ergonomic rules) which is a tool for the automatic generation of interfaces. Also included is a list of various attempts to automatically generate interfaces from a set of specifications.

Automatic techniques for generating interfaces suffer from unpredictability (Myers, Hudson & Pausch, 2000). Such methods often rely on heuristics and the connection between specification and final result can be quite difficult to understand and control. Automatic generation of code for any type of software project is a difficult task and automatic generation systems always place significant limitations on the kinds of user interfaces they can produce.

2.4 Dialogue modelling in Lean Cuisine

Lean Cuisine (Apperley & 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 actions, objects, states and other attributes. A meneme is defined as having just two possible states, 'selected' and 'not selected'. The state of a meneme may be changed either by direct selection or by indirect modification (as the result of the selection of another meneme). Lean Cuisine offers a clear, concise and compact graphical notation for describing the *structure* of menu-based dialogues.

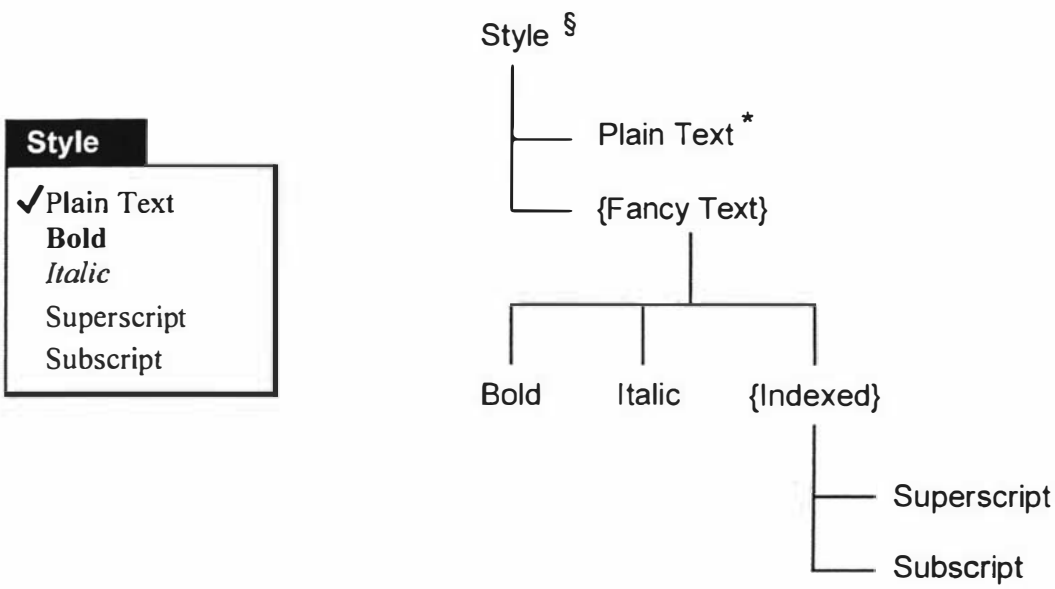


Figure 2.8: Lean Cuisine in action, showing (left) the Style menu as it appears in the user interface and (right) its Lean Cuisine description (Apperley & Spence, 1989)

Figure 2.8 illustrates the Lean Cuisine representation of the Style menu. {Fancy Text} heads a *mutually compatible* (*m* from *n*) subdialogue group with the menemes arranged horizontally whereas the subdialogue group headed by {Indexed} is *mutually exclusive* (1 from *n*) with the menemes arranged vertically. Style is designated as heading a *required choice* (§) subdialogue in which Plain Text is designated the *default choice* (*). Menemes may be *real* or *virtual*. Real menemes, such as Plain Text, represent specific actions or selectable objects in the final interface. Virtual menemes, such as {Fancy Text}, are headers of logically connected groups called *subdialogues*. Virtual menemes cannot be directly selected but they can be indirectly selected as a result of a valid selection having been made in the subdialogue that they represent. They may also be deselected as a result of the selection of a mutually exclusive meneme in which case any selected menemes in the subdialogue become deselected.

The Lean Cuisine representation for the Style menu should be compared to the STD, Petri net and statechart representations for the same menu in Figures 2.5, 2.6 and 2.7 respectively. The lack of clutter in the Lean Cuisine representation is immediately obvious and the structure of the dialogue is clearly shown. Lean Cuisine is also naturally more inclined towards object-oriented design than STDs, statecharts or Petri nets. When tasks are modelled in conjunction with dialogue, Lean Cuisine presents the objects (menemes) available for selection during task execution whereas the other representations focus on the transitions between states in the system and thus emphasise task flow. The state of a Lean Cuisine dialogue is represented by the sum of the individual meneme states. Events are not directly represented but are implicitly associated with menemes. The sequence of events is not explicitly captured, although the hierarchical structure (nesting of subdialogues) is an indicator of the order in which menemes will be selected.

A simple support environment for Lean Cuisine was developed as part of a Masters project in 1995 (Gu, 1995). This enabled the construction and browsing of Lean Cuisine diagrams.

2.5 Towards Lean Cuisine+

Lean Cuisine was developed and initially applied in the context of menu systems. A menu is essentially a static object, although it is possible to think of examples that exhibit some dynamic properties e.g. a menu of document names. In contrast, most aspects of direct manipulation interfaces are dynamic, e.g. the manipulation of windows and icons, and involve multiple states, conditions and object interactions. A study of user tasks when using a direct manipulation interface (Phillips & Apperley, 1991) indicated that Lean Cuisine could be extended to handle such interactions, even though it was designed in the context of menu systems. A case study (Phillips, 1991) used Lean Cuisine to model the dialogue of a direct manipulation interface and compared it to other dialogue approaches and further research led to the extended notation, Lean Cuisine+.

Lean Cuisine+ (Phillips, 1995) is a graphical notation for modelling the behaviour of event-based direct manipulation interfaces. The Lean Cuisine+ notation has several basic strengths in that it has been specifically developed as an interface design tool, it incorporates the structure of the interface dialogue and it fits easily into an object-oriented approach. A major advantage of the notation is that task sequences can be represented in the context of the structure of the interface dialogue and in this way, Lean Cuisine+ combines both dialogue modelling and task modelling. Lean Cuisine+ is described in Chapter 3.

Chapter 3. The Lean Cuisine+ Notation

This chapter traces the development of the Lean Cuisine+ notation for task and dialogue modelling. Section 3.1 provides a description of the first version of Lean Cuisine+ that was developed from Lean Cuisine. Section 3.2 outlines a number of modifications that have been made to the original Lean Cuisine+ notation and the reasons surrounding these changes. Section 3.3 provides in-depth coverage of how task modelling is performed with Lean Cuisine+. This is important as it enables the notation to assume its unique position as the bridge between task and dialogue modelling, whereas the original Lean Cuisine was developed as a dialogue modelling notation only. Section 3.4 provides an illustration of the latest version of Lean Cuisine+ in action via a case study and Section 3.5 is a brief review of the chapter.

3.1 Early Lean Cuisine+

The Lean Cuisine+ notation (Phillips, 1995) is an executable semi-formal graphical notation for describing the underlying behaviour of event-based direct manipulation interfaces. The base tree diagram of Lean Cuisine is retained, with menemes now annotated to differentiate objects and actions. A number of layers are then superimposed on the base diagram, to capture further constraints. These layers are:

- the *selection trigger* layer that captures any system responses to user actions;
- the *option precondition* layer that specifies any conditions relating to the availability of menemes for selection;
- the *existence dependency* layer which captures the fact that the existence of an object may depend on the existence of another object or subdialogue;
- the task layer that provides a description of higher level user tasks and thereby creates a close link between the tasks and the dialogue.

The early Lean Cuisine+ notation is illustrated here through a description of aspects of the Apple Macintosh Finder document folder system c1990. Figures 3.1 to 3.6 are taken from (Phillips, 1993) where the case study first appeared. The complete document folder life cycle is described: a document folder instance can be created; the icon representing it can be moved and renamed; the folder can be opened to make contents available via a window, which can be moved and resized and its contents scrolled; the window can be closed; and the folder can be deleted.

The base diagram

Figure 3.1 illustrates the Lean Cuisine+ base diagram for the Finder case study. The basic mutually exclusive and mutually compatible constructs of Lean Cuisine are retained. Menemes representing objects are boxed, with a diagonal in the top left corner specifying a *syntactic* (or system) object, e.g. TrashFolder. NewFolder, EmptyTrash and PrintDirectory form a mutually exclusive subdialogue group that is mutually compatible with the FolderIcon and Window subdialogues, both of which provide access to further options. The virtual meneme {Controls} has been used to further group options within the Window subdialogue.

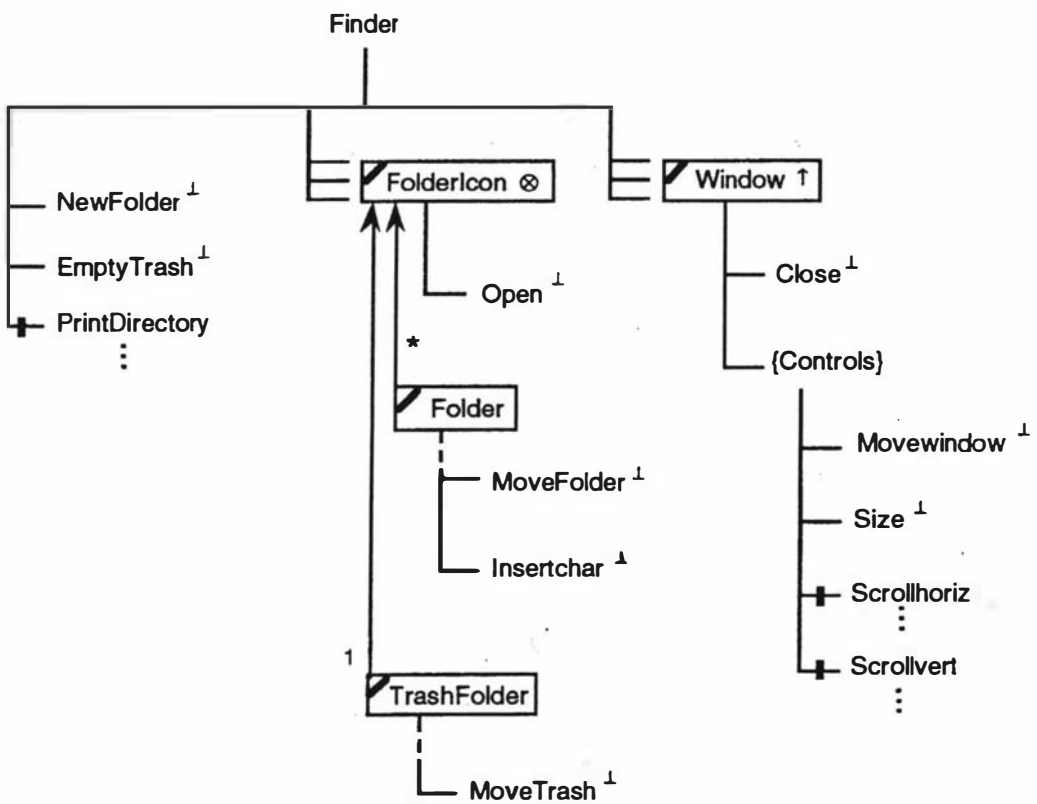


Figure 3.1: The base diagram for the Finder case study

A *modal* subdialogue is one which when selected excludes all other parts of the dialogue. There are two ways of representing modal subdialogues in Lean Cuisine+. Simple modal subdialogues such as NewFolder, involving a primitive operation, are represented by *monostable* menemes, shown thus (⊥). Monostable menemes remain selected only for the duration of the operation they represent and revert to an unselected state on its completion, through system action. More complex modal subdialogues are designated by a bar across the link to the subdialogue header, as in the case of PrintDirectory.

The fork symbol indicates an *homogenous group* – in this example many FolderIcons or Windows can appear in the interface at any time although only one instance of either group can be selected at any time (mutually exclusive groups). Folder and TrashFolder are shown with an arrow pointing back to FolderIcon and a dashed link to their own subdialogues – this identifies them as *subtypes* of FolderIcon. The subtype TrashFolder is shown as having a *cardinality* of 1 (the default is a cardinality of many represented by the * - see for example the subtype Folder). There are two different uses of the * in early Lean Cuisine+ - the other is as a meneme designator (see below). Selection of the subtype causes selection of the parent and access to its subdialogue options, e.g. selection of Folder causes FolderIcon to be selected and gains access to the Open meneme as well as its own subdialogue of MoveFolder and Insertchar. A vertical ellipsis below a meneme indicates that further development of that subdialogue takes place in another diagram, e.g. Scrollhoriz.

The following *meneme designators* are used to indicate restrictions placed on the selection, or deselection, of certain menemes:

{ } *Virtual meneme*: a non-selectable meneme used as a header of a subdialogue.

◮ *Monostable meneme*: a meneme that, following user selection, reverts to an unselected state on completion of the simple modal subdialogue it represents.

↑ *Select-only meneme*: a meneme that cannot be directly deselected by the user, although it may become deselected through selection of another mutually exclusive option.

↓ *Deselect-only meneme*: a meneme that cannot be directly selected by the user, although it may become selected as an indirect result of some other action.

⊗ *Passive meneme*: a meneme of this type is unavailable for selection or deselection by the user and can only be selected and deselected by the system.

* *Default choice*: indicates a meneme that is to be in the selected state at the commencement of a dialogue or subdialogue.

§ *Required choice*: is associated with a subdialogue header and indicates a meneme group from which a valid choice is always required.

• Dynamic default: may be associated with a meneme (assigned) or a subdialogue (unassigned) and indicates a default choice that takes on the value of the last user selection from that subdialogue.

The selection trigger layer

The selection trigger layer is shown over a "greyed" base layer tree in Figure 3.2. Selection triggers indicate system responses to the selection of particular menemes by the user. They capture the fact that the selection or deselection of a meneme may trigger the selection or deselection of other meneme(s). For example, the selection of NewFolder triggers the creation and selection of a Folder. The vertical bar across this trigger indicates that it is *constrained*, in which case the behaviour of the selected meneme is modified, inhibiting further triggers and preventing side effects such as cycles. Some triggers are *conditional* – for example, selection of a Folder triggers selection of the parent Window, if the icon is within the window (condition C1). *Trigger designators*, such as Select or Deselect, show whether the trigger is caused by (or causes) the selection or deselection of a meneme – see the legend provided in Figure 3.2.

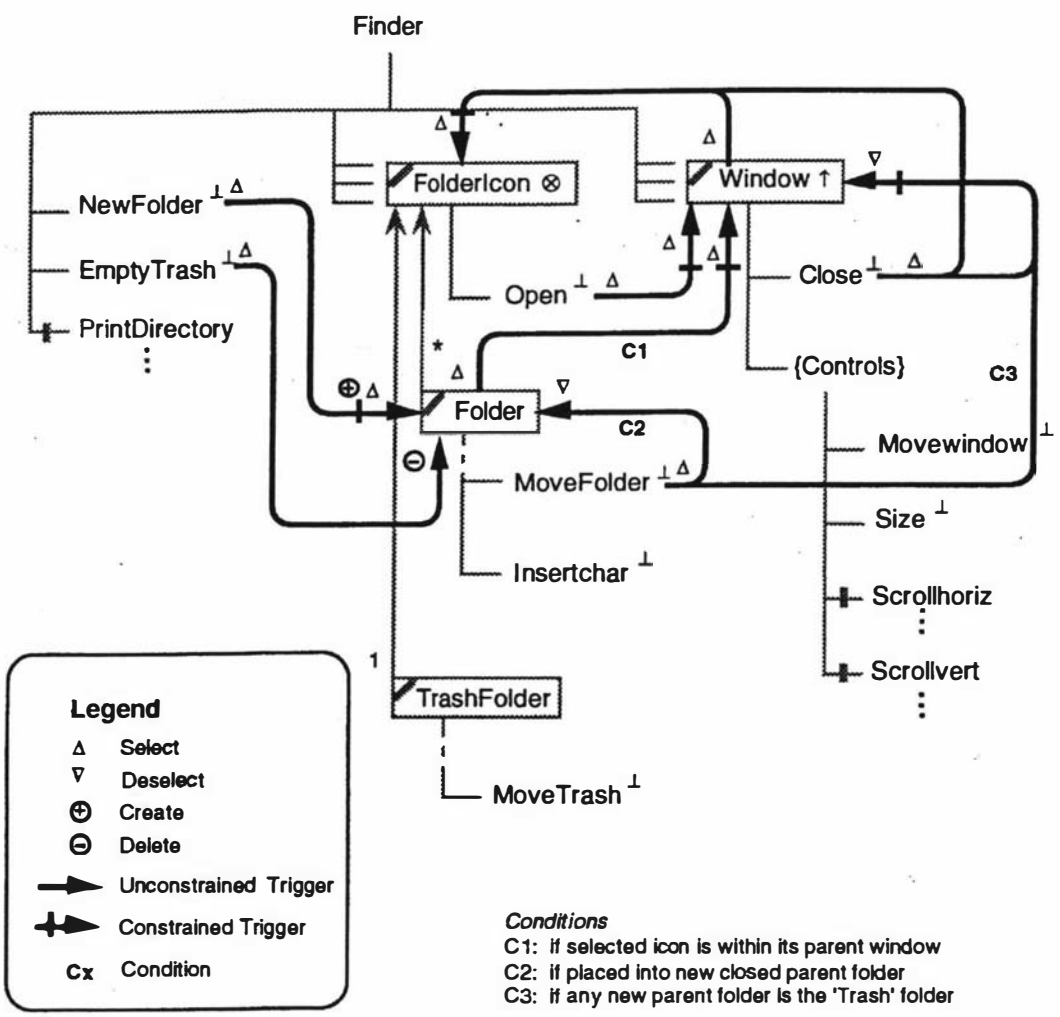


Figure 3.2: The selection trigger layer for the Finder case study

The option precondition layer

Figure 3.3 illustrates the option precondition layer shown over a "greyed" base layer tree. An option precondition is a condition for the availability for selection of a meneme, which may involve the state of a meneme in another part of the dialogue. An option precondition is thus indirectly an event precondition.

For example, in Figure 3.3, the meneme NewFolder is available for selection only if an active window exists, i.e. a window instance is currently selected (note the "Selected" designator at the start of the precondition link). A further constraint (P1) also applies, in that NewFolder is not available if the selected window is the Trash folder window. Other preconditions are not shown as links, but are attached to a specific meneme, for example EmptyTrash is only available if the Trash folder is not empty – see precondition P2.

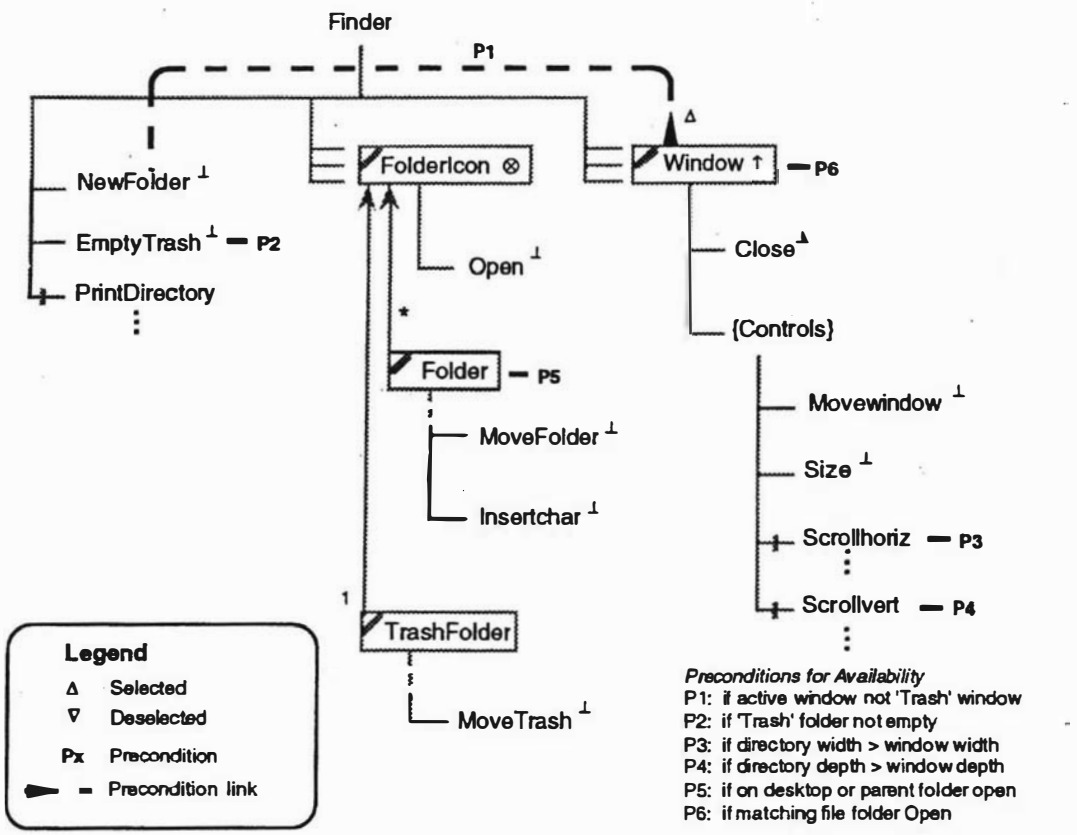


Figure 3.3: The option precondition layer for the Finder case study

The existence dependency layer

The existence of a meneme may depend upon the existence of another meneme or subdialogue. Figure 3.4 illustrates the existence dependency layer which, in this example, consists of the single mutual existence dependency between the FolderIcon and Window menemes. A Window instance cannot exist without the corresponding FolderIcon instance and vice versa. The base layer diagram is shown "greyed".

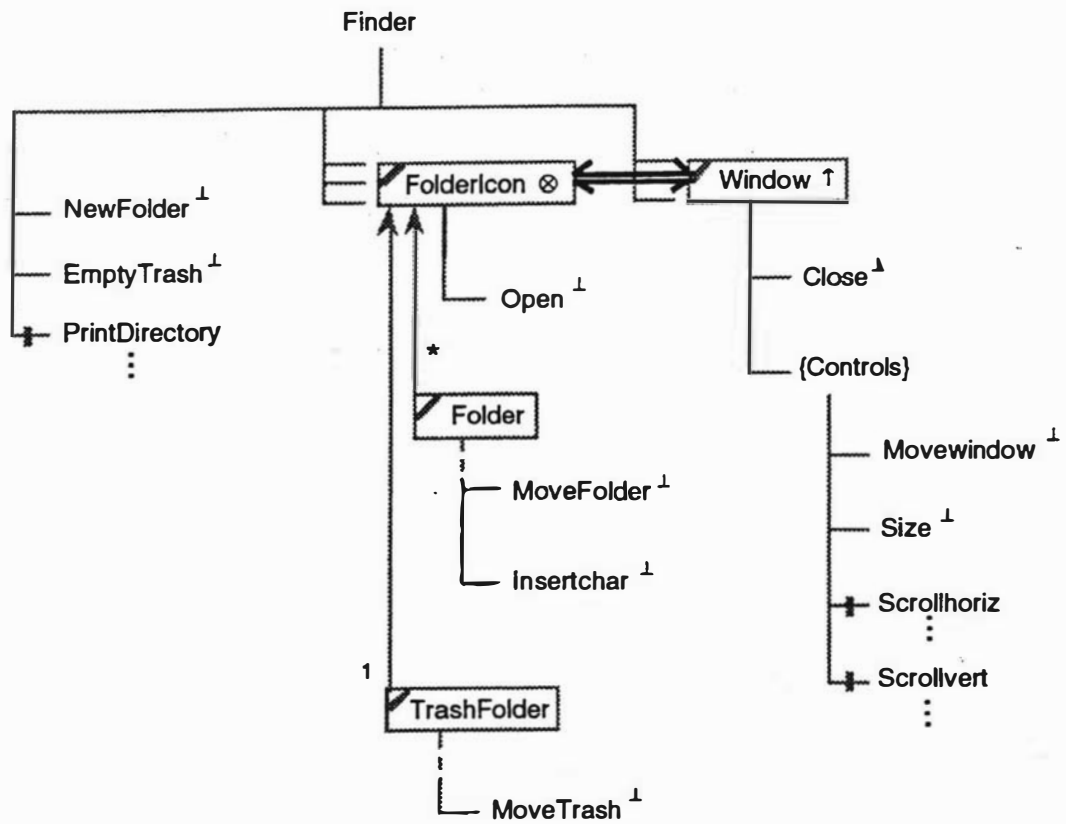


Figure 3.4: The existence dependency layer for the Finder case study

The task layer

The task layer provides a bridge between task and dialogue modelling, and is arguably the most significant Lean Cuisine+ layer in the context of the subsequent development of the notation. Each task (or sequence of related sub-tasks) can be represented in Lean Cuisine+ as an overlay of linked menemes superimposed on the "greyed" base diagram. Menemes are linked together into a task sequence that indicates the order in which they must be selected.

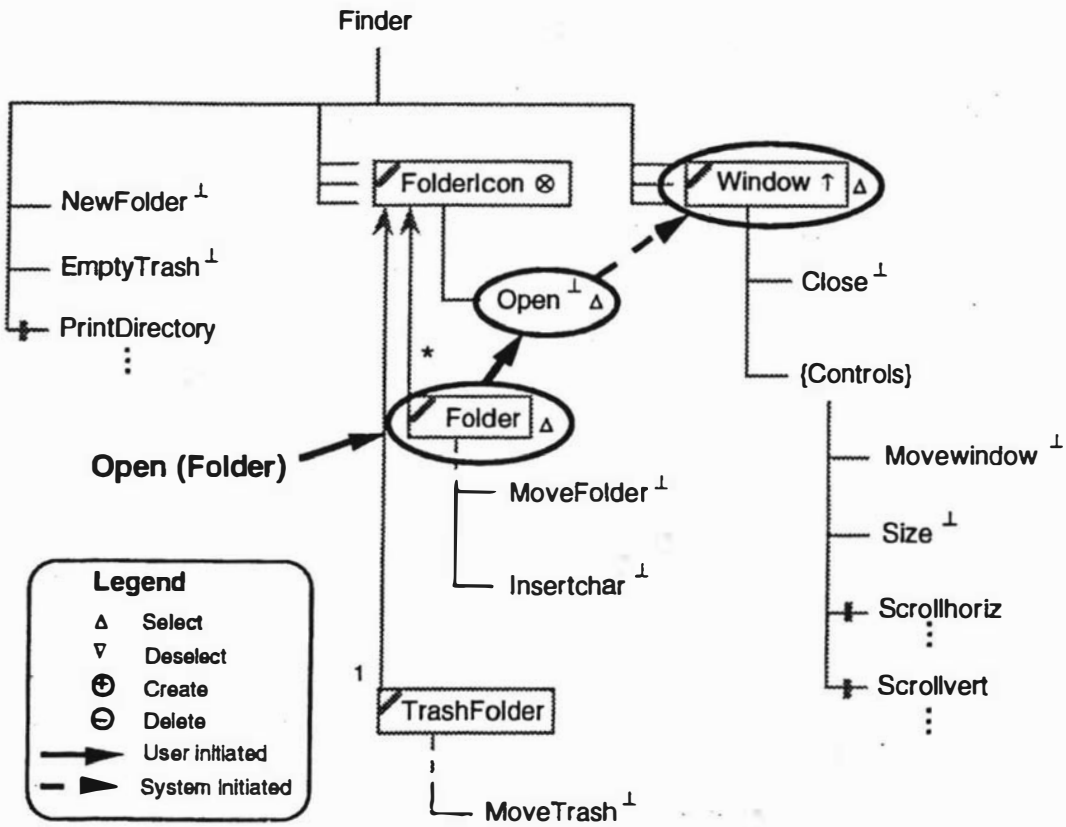


Figure 3.5: One of the task sequences for the Finder case study

In Figure 3.5, the task Open (Folder) is represented as a sequence of three meneme selections. In this example, the user selects a Folder instance and then selects Open and the system responds by selecting the corresponding Window instance. Note that the system action is designated by the dashed arrow to differentiate it from the preceding user actions.

Review of the original Lean Cuisine+ notation

Lean Cuisine+ is a multi-layered notation. The base layer is a tree diagram that captures part of the behaviour of a direct manipulation interface in terms of constraints and dependencies

between selectable primitives (menemes). Associated with each meneme is an implicit event. User manipulation of a meneme can be restricted through the use of meneme modifiers. Further constraints and dependencies associated with the dialogue are captured through overlays, where events are explicitly shown. The task layer, in particular, defines task sequences and provides a task-oriented view of the interface. Lean Cuisine+ is thus able to combine both task and dialogue modelling.

3.2 Lean Cuisine+ revised

The first version of Lean Cuisine+ (described in Section 3.1) was not supported by any tailored software package. Diagrams were produced using a standard drawing package. The notation was analysed prior to the design and development of a supporting software environment (discussed in Chapter 6) and several modifications were implemented. This section provides examples of the original and modified notations, and gives reasons for the modifications.

The object model

In the original Lean Cuisine+ version, the object model was embedded in the dialogue tree. This had the advantage that an object could be identified with its actions, but could lead to confusion as to what constituted an object in the first place. An example is illustrated in Figure 3.6 - which is a copy of Figure 5.2 in (Phillips, 1993).

This figure shows the further refinement of the PrintDirectory subdialogue that appears as a modal subdialogue header in Figure 3.1 with a downward ellipsis. There are several default choices (*) such as Cassette in the {Papersource} subdirectory. In this diagram, it is not clear why Copies, From and To are identified as objects. The specifications for printing a document would presumably place these three menemes in the same category as All pages or Color in that they all modify the behaviour of the printer during the printing process. However, All pages and Color are represented as actions in the diagram, and not objects. It is equally unclear why the overall Print Directory is represented as an action and not as an object.

The problem arises from the difference between high-level objects and low-level objects. For example, in a word processing package, a Document would certainly be regarded as an object (because it is a high-level object). However, the OK Button to confirm printing would be regarded by the developer as a low-level object whereas the user would not regard it as an object at all.

For these reasons, the object model has been removed from Lean Cuisine+ and this has led to a number of improvements in the clarity of diagrams using the notation, viz:

- the distinction between syntactic and semantic objects is no longer required. This was often difficult to determine, particularly in the automatic generation of Lean Cuisine+ diagrams for existing interfaces. See Chapter 7.

- boxes around the menemes of objects are no longer required which reduces visual clutter and makes the diagram easier to read. Compare the notations for the same diagram in Figures 3.1 and 3.11.

- subtypes disappear which make the behaviour of certain menemes clearer. For example, see the Trash folder in Figures 3.1 and 3.11. It also means that the * is used only as a meneme designator (see Figure 3.6) and no longer has two distinct meanings.

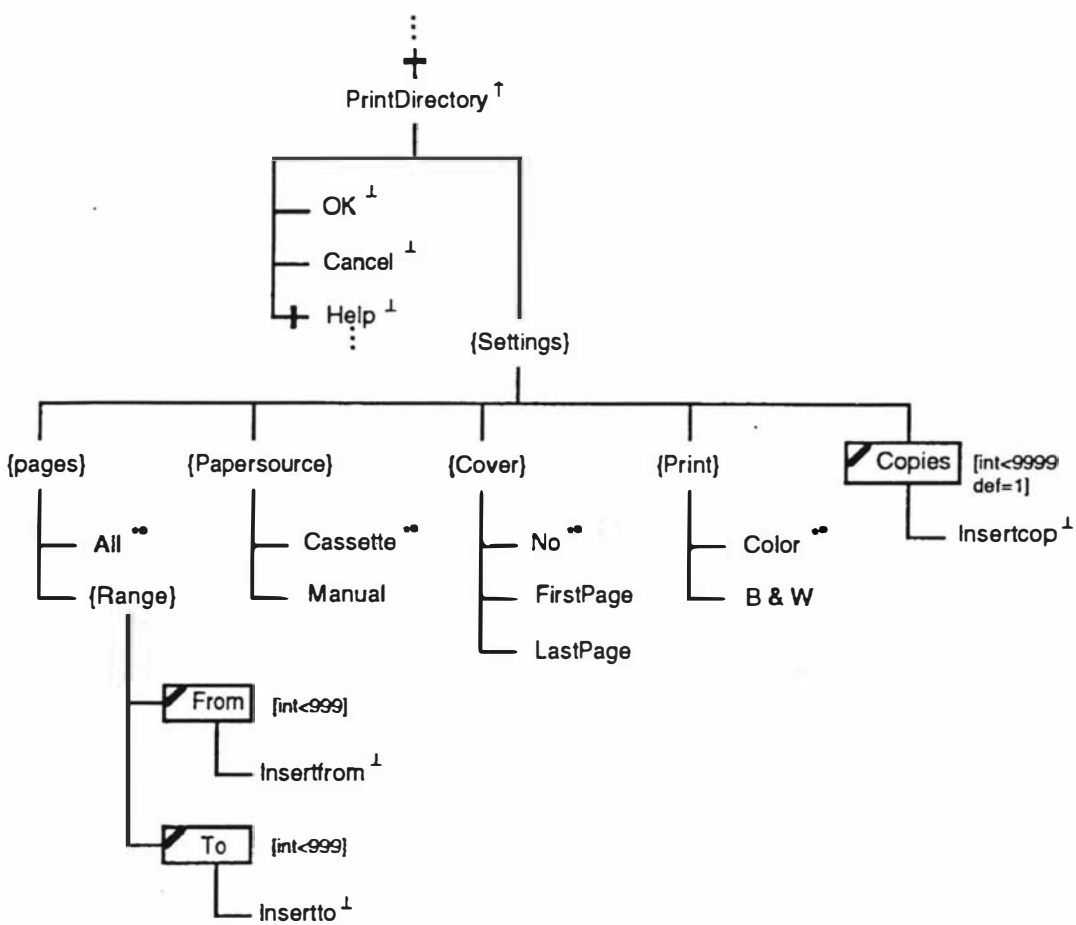


Figure 3.6: Objects and actions in a subdialogue tree

Since the object structure of the interface has been removed from the Lean Cuisine+ dialogue tree, it needs to be described elsewhere. In the method presented in Chapter 5, object structure is described using UML class diagrams.

The layers

Lean Cuisine+ was developed as a paper-based notation that consisted of a base diagram and four overlays: the option precondition layer, the existence dependency layer, the selection trigger layer and the task layer. As the software environment for the notation was constructed, it became possible to display or hide different sections or attributes of the diagram and a more fluid system evolved, leading to the removal of some of the layers. The term "base diagram" became less appropriate and has been replaced by the term "dialogue tree".

The option precondition layer has been removed. Instead, each meneme has been allocated a new attribute called a *condition*. The condition states any restrictions on the availability for selection of the meneme and is displayed adjacent to the meneme (or can be hidden if required). Figure 3.7 demonstrates how the new conditions can replace option preconditions and option precondition links. This figure should be compared with Figure 3.3.

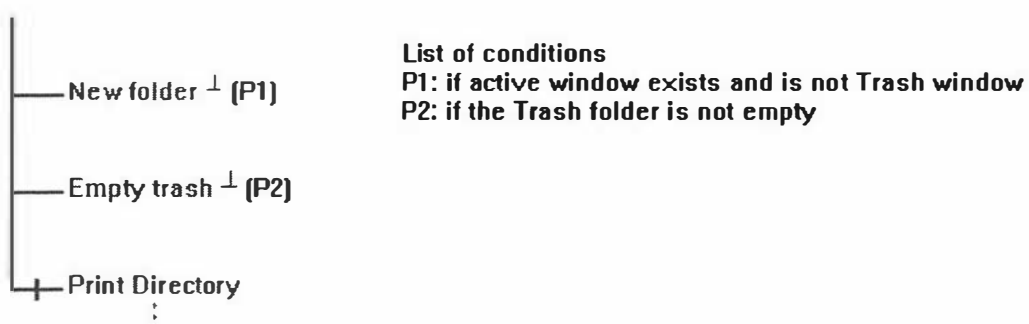


Figure 3.7: Conditions have replaced option preconditions

An existence dependency is an object relationship and thus the existence dependency layer has been removed along with the object model. If existence dependencies need to be shown, this can be done as part of a separate object model using, for example, UML class diagrams as described in Chapter 5.

The selection trigger layer has been retained and appears visually as before. With the advent of the software environment for the notation, it is possible to display or hide a trigger and thus all, some or none of the triggers can be viewed, or printed, at any time. Triggers can also be

displayed with conditions. Figure 3.8 illustrates that conditions can be displayed in one of two ways: the condition can be displayed alongside the meneme or trigger, or placed in the list of conditions with only the identifying label appearing alongside the meneme or trigger.

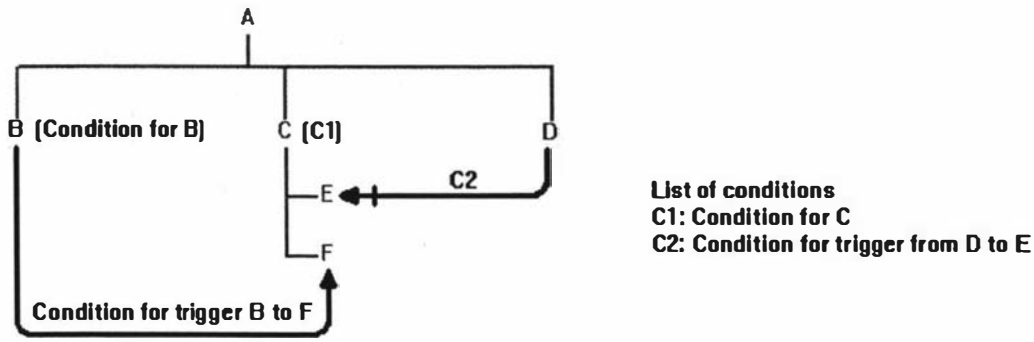


Figure 3.8: Different ways of displaying conditions

The task layer has been retained since this layer forms an important link between the description of the dialogue structure (how functionality is grouped) and interaction sequences (how functionality is accessed). It is possible to view all tasks or no tasks or a single task by selecting that task from the list provided. The notation for representing tasks has been expanded and the various additions to it are described in section 3.3.

Subdialogues

The ellipsis is used in Lean Cuisine+ to indicate the presence of a subdialogue; i.e. that the diagram should continue from the current meneme but has been truncated at the point where the ellipsis has been inserted. This supports *stepwise refinement* where it is convenient to develop the detail of the subtree in a separate diagram and thus the original diagram is spared extra size and clutter.

The paper-based Lean Cuisine+ notation presented subdialogues as separate diagrams. This idea has been extended in the prototype software environment where subdialogues have become a dynamic concept since sections of the diagram can be viewed or hidden as desired. A meneme is selected and the subdialogue of that meneme can be hidden. Alternatively, the rest of the diagram can be hidden, leaving only the selected meneme as the header of a subdialogue. A following and/or leading ellipsis is displayed when any section of the diagram is hidden. Thus the diagram can have any sections displayed or hidden in order to focus on a particular part.

Figure 3.9 illustrates this concept by showing three views of the same dialogue. View (a) shows all menemes in the dialogue; in view (b) the subdialogue headed by C has been hidden; and in view (c) all menemes *except* the subdialogue headed by C have been hidden.

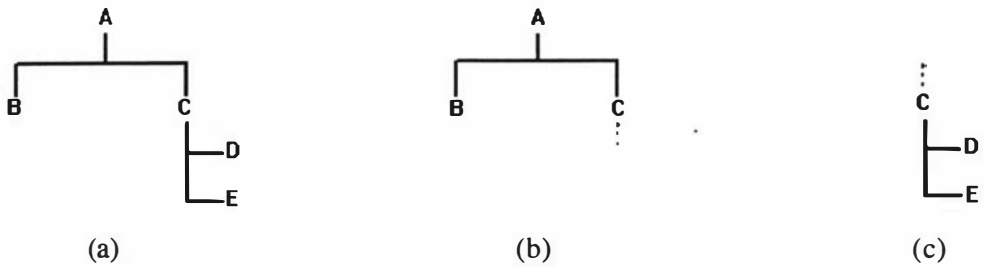


Figure 3.9: Three views of the same dialogue

The concept of separate subdialogues is still supported and is, in fact, necessary to avoid duplication of the same subdialogue. For example, the typical word processing package shown in Figure 3.10 has three separate places in the dialogue that lead to the print directory. However, this is always the same print directory and would be displayed in a single separate subdialogue. A revised version of the Lean Cuisine+ notation has been published (Scogings, 2000) but several further refinements have occurred since then and the current notation is included as Appendix 1.

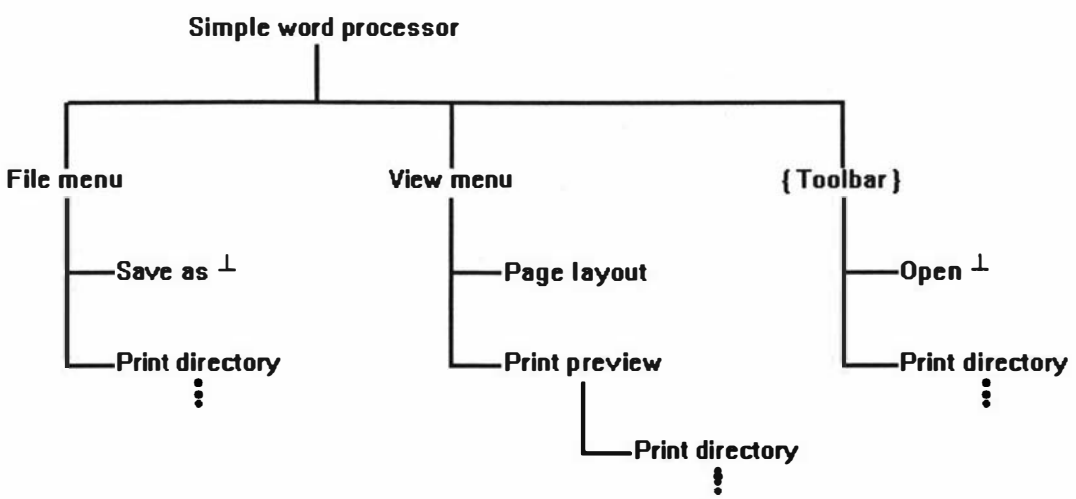


Figure 3.10: Lean Cuisine+ diagram for some commands in a simple word processor.

Minor changes

Unlike Lean Cuisine, or the original version of Lean Cuisine+, the revised notation is supported by a software environment, and for this reason, certain minor changes have appeared in the

notation to make it more amenable to computer generation. A rule has been introduced that all subdialogues require a header meneme. Thus, for example, the subdialogue containing NewFolder, EmptyTrash and PrintDirectory has no header meneme in Figure 3.1 but is provided with the header {Commands} in Figure 3.11. This rule does not affect the dialogue in any way but ensures that subdialogues can be treated in a consistent manner within the software support environment.

Figure 3.11 (see page 38) shows the revised Lean Cuisine+ notation representing the same Apple Macintosh Finder system as Figure 3.1 - which is repeated on page 38 with Figure 3.11 for easy comparison.

Lean Cuisine+ does not enforce any particular conventions for naming menemes and astute readers will note a subtle change from Figure 3.1 to Figure 3.11 in the way menemes are named. For example, TrashFolder in Figure 3.1 has become Trash folder in Figure 3.11. This change simply reflects different personal preferences for naming conventions since different people constructed the two figures at different times.

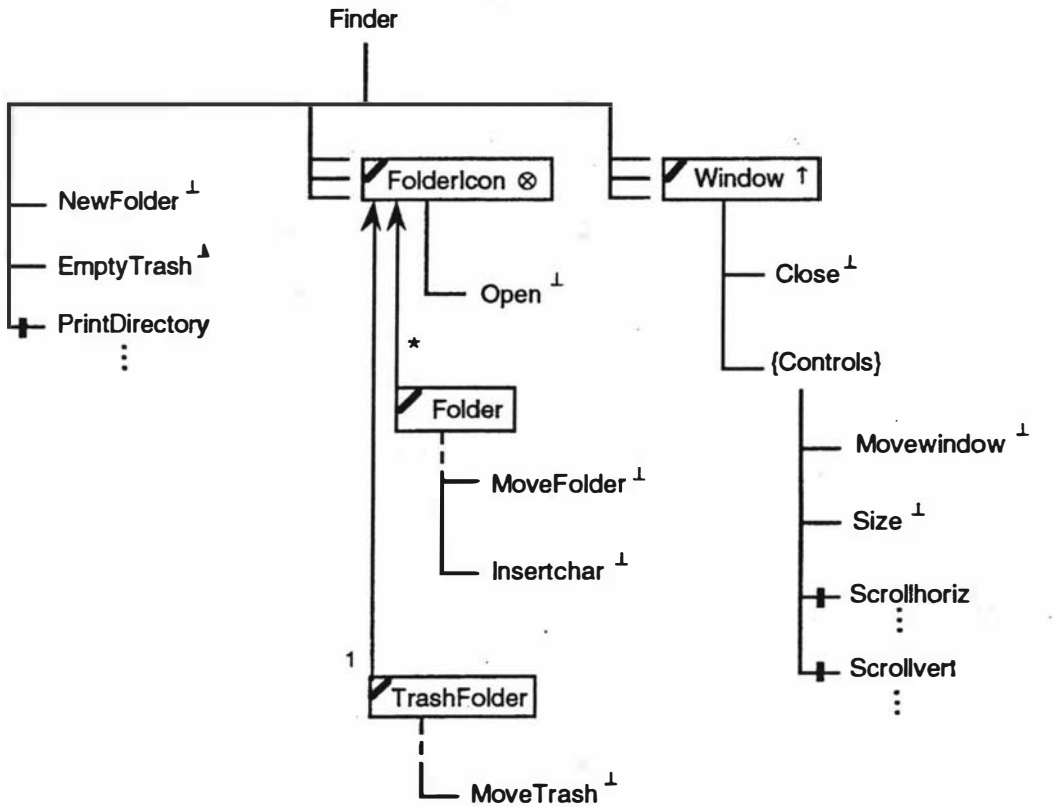


Figure 3.1 (repeated): The base diagram for the Finder case study (early notation)

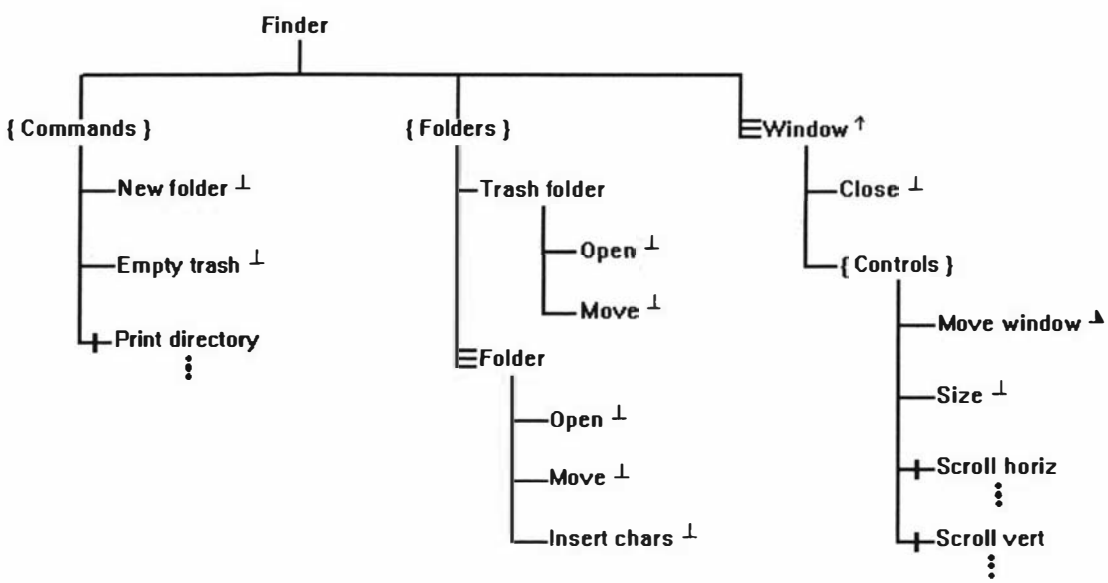


Figure 3.11: The same dialogue tree as Figure 3.1 using modified Lean Cuisine+

3.3 Task modelling with Lean Cuisine+

The Lean Cuisine+ notation has several basic strengths in that it has been specifically developed as an interface design tool, it incorporates the structure of the interface dialogue and it fits easily into an object-oriented approach. A major advantage of the notation is that task sequences can be represented in the context of the structure of the interface dialogue. Lean Cuisine+ thus combines both dialogue modelling and task modelling. The interface is represented primarily as a dialogue tree and each task (sequence of related sub-tasks) is represented as a series of actions (meneme selections or deselections) superimposed on the dialogue tree diagram. This allows rapid analysis of whether the dialogue supports all the actions required for a task, and whether they can be efficiently executed.

Two graphical representations have been added to the original Lean Cuisine+ task sequences:

- *The selection of a virtual meneme.* This does not indicate the selection of the virtual meneme during the execution of the task, but rather the selection of one or more of the menemes belonging to the subdialogue headed by that virtual meneme. In the case of a *mutually exclusive* subdialogue, any one of the menemes comprising the subdialogue may be selected. In the case of a *mutually compatible* subdialogue, any combination of the menemes comprising the subdialogue may be selected.

- *The double-headed arrow.* This indicates that the user may select any number of menemes at this step in the task sequence. This may be used in conjunction with the selection of a virtual meneme as described above.

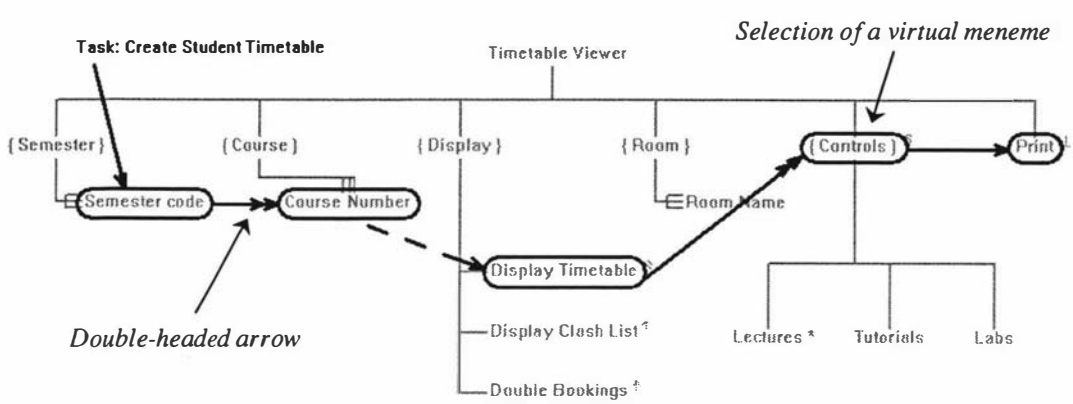


Figure 3.12: An example showing changes to the task sequence notation

Figure 3.12 shows the Lean Cuisine+ diagram representing the user interface for a University Timetable Viewer system. Students can view their timetables for a particular semester by selecting several courses and displaying, and then printing, the corresponding timetable. The level of detail in the timetable can be controlled by choosing to show only lectures, or lectures and lab sessions, etc.

Figure 3.12 demonstrates how the selection of the virtual meneme {Controls} using a double-headed arrow means that any combination of the menemes in the mutually compatible sub-dialogue may be selected without affecting the main flow of the task sequence. The double-headed arrow leading to Course Number indicates that any number of course numbers may be selected at that point. Figure 3.12 is part of the university timetable case study that appears in Chapter 5.

The Lean Cuisine+ task sequence provides the following advantages:

- the task is represented at a level appropriate for interface design;
- the task is shown within the context of the interface;
- both system responses and user actions can be represented;
- the task sequence can be used in interface analysis.

The task sequence forms an important link between the description of the dialogue structure and valid interaction sequences as reported in (Phillips & Scogings, 2000). No other task description method represents tasks within the context of the dialogue and Lean Cuisine+ therefore uniquely allows the dialogue to be analysed in a variety of ways, such as whether superfluous menemes exist in the dialogue, whether all required tasks are supported by the dialogue and whether the required tasks can be efficiently executed (by minimizing meneme selections).

A further advantage of the Lean Cuisine+ approach of defining tasks as a sequence of events is that the sequence is made concrete and can be checked for accuracy. One of the difficulties of text-based task modelling is the fact that goals are not well defined and exist on many levels, and this can often be confusing (Cockburn, 1997).

3.4 The revised Lean Cuisine+ notation in action

This section shows how the revised Lean Cuisine+ notation can be used to model the dialogue for a direct manipulation user interface. The interface described is that of a typical automatic washing machine - a touch sensitive interface incorporating coloured LEDs and warning sounds. When the machine is switched on, it offers a number of options all set to the default settings. The user can change any of the settings or can select a particular wash cycle – such as Wool. The selection of certain wash cycles leads to the automatic selection of certain wash settings. The examples in this section produce a sample of the Lean Cuisine+ diagrams needed to fully represent the user interface of the washing machine and there is no intention to provide a complete description of the behaviour of the device.

Figure 3.13 shows the basic tree diagram for the Lean Cuisine+ representation of the washing machine interface. Menemes are divided into four logical groups, each headed by a virtual meneme. Most subdialogues are mutually exclusive except for {Wash options} which presents two mutually compatible options. A number of subdialogues are designated as required choice(§) and in each case a default choice(*) is indicated. The menemes Start and Advance are both designated as monostable(⊥) as they revert to an unselected state through system action.

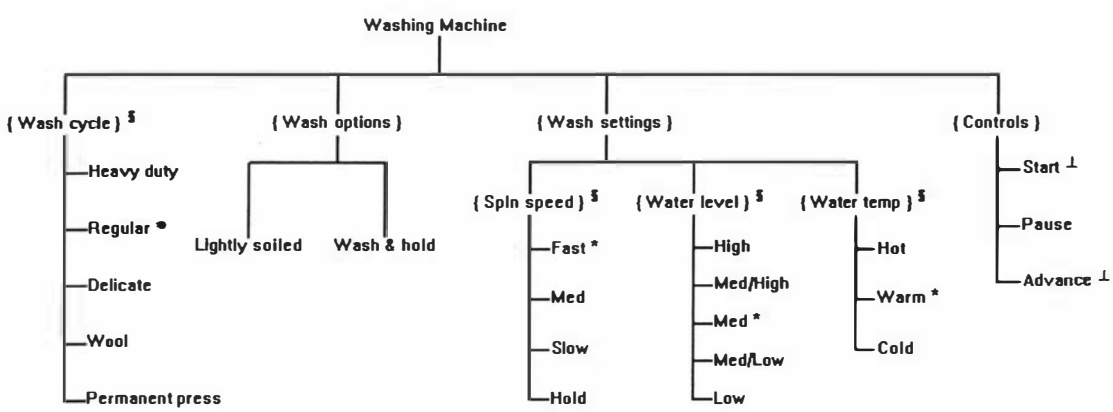


Figure 3.13: The Lean Cuisine+ tree diagram for the washing machine

In Figure 3.14, conditions have been placed on the selection of certain menemes. For example, a High water level can only be selected if the wash cycle is Regular or Heavy duty, and a Fast or Medium spin speed can only be selected if the wash cycle is not Delicate or Permanent press.

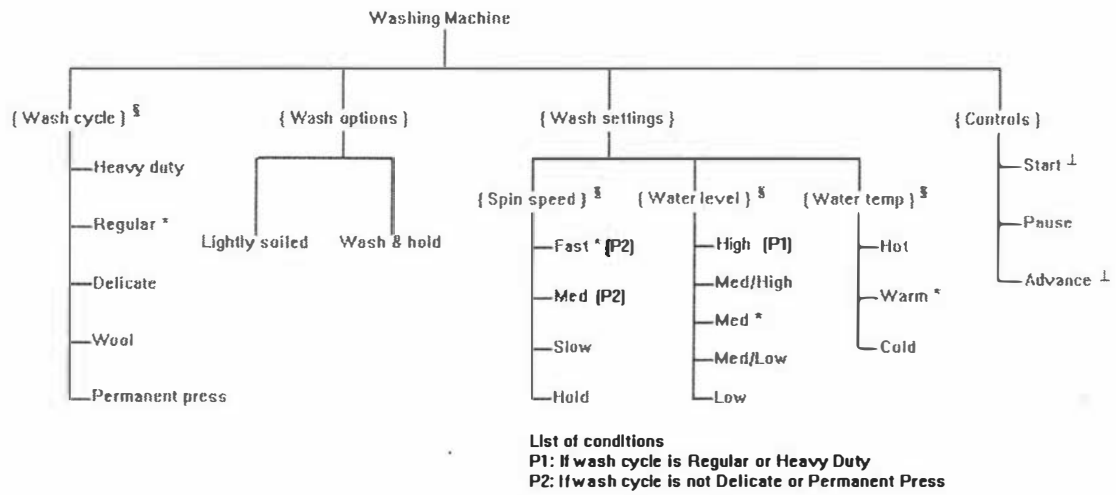


Figure 3.14: Example conditions in the tree diagram for the washing machine

Example selection triggers are illustrated in Figure 3.15. The triggers indicate that the user selection of Wool in the {Wash cycle} subdialogue will trigger the system selection of Medium (spin speed), Medium (water level) and Warm (water temperature). Likewise, the user selection of Permanent press in the {Wash cycle} subdialogue will trigger the system selection of Slow (spin speed), Medium (water level) and Warm (water temperature). The triggers are annotated with the select symbol (Δ) to indicate that the *selection* of menemes, not their deselection, triggers the described actions.

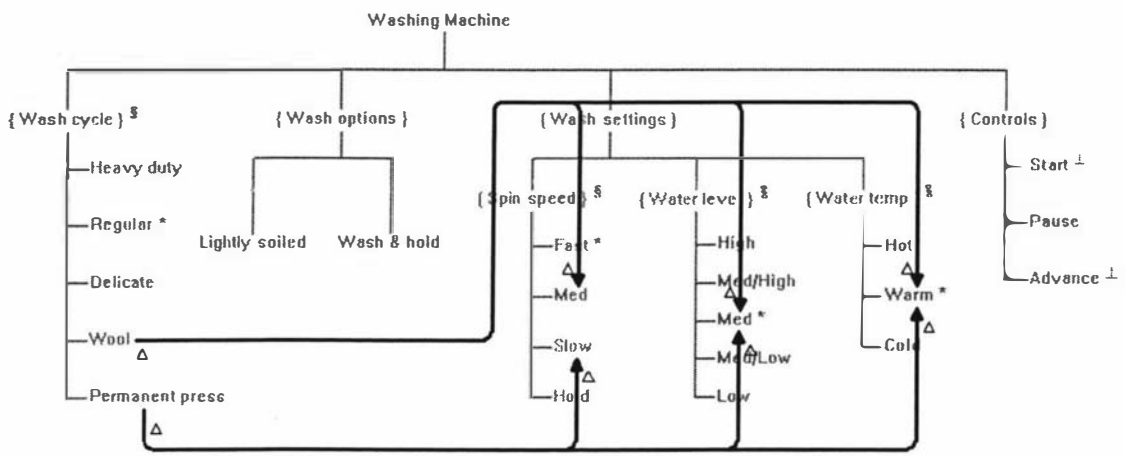


Figure 3.15: Example selection triggers for the washing machine

An example task sequence is provided in Figure 3.16. In order to perform the task Wash woollens, the user must select Wool in the {Wash cycle} subdialogue. This triggers the system selection, indicated by the dashed links, of Medium (spin speed), Medium (water level) and Warm (water temperature). Finally, the user selects Start.

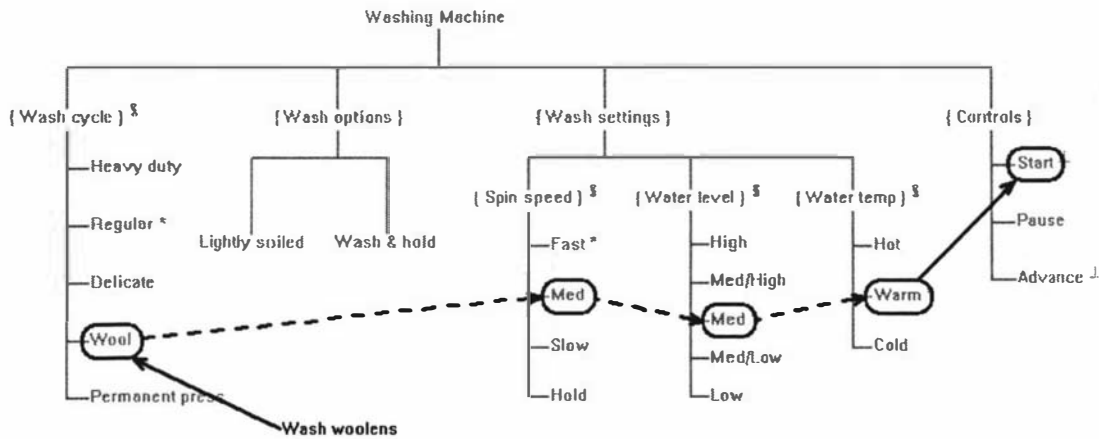


Figure 3.16: An example task sequence for the washing machine

3.5 Review

Lean Cuisine was originally designed as a dialogue modelling tool for simple menu-based user interfaces. Lean Cuisine+ was then developed as a powerful dialogue modelling tool for graphical user interfaces, whilst retaining the advantage of a simple structure that is clear and easy to use. The main features introduced in Lean Cuisine+ are:

- selection triggers – that reflect system responses to user actions;
- meneme conditions – that specify any conditions relating to the availability of menemes for selection.
- task sequences – that enable the modelling of tasks within the context of the dialogue structure.

It is important to place Lean Cuisine+ in the wider context of software design and development (discussed at the beginning of Chapter 2). The use of modelling techniques and notations is enhanced if they can fit in with existing methods of software engineering. UML is a widely used software development language and is described in Chapter 4. A method that incorporates Lean Cuisine+ modelling as part of the UML design process is introduced in Chapter 5.

A Lean Cuisine+ support tool has been developed and is described in detail in Chapter 6. This prototype environment enables the designer to:

- generate Lean Cuisine+ dialogue trees
- add selection triggers
- create task overlays
- save, load and print diagrams (including triggers and task overlays)

Chapter 4. The Unified Modelling Language (UML) and the Unified Process

"(It is important to realise) that user interface design is not software design. Models originally developed to support the design of software components and their interactions are not automatically and necessarily well-suited for organising user interface components and the interaction between users and these components."

Constantine & Lockwood, 2001

4.1 Introduction

At the wider level, this research is concerned with the positioning of user interface design within the software engineering process. Previous chapters have addressed the role of dialogue and task modelling in user interface design and the ability of the Lean Cuisine+ notation to form a bridge between those two areas. This chapter covers the UML notation, the Unified Process and the role of user interface design within the overall software design process. Section 4.2 outlines the UML notation and Section 4.3 introduces the Unified Process (the software design process associated with UML). Section 4.4 reviews support for user interface design in software engineering methods, including UML and the Unified Process. Section 4.5 investigates ways of integrating user interface design with UML.

Faced with new object-oriented programming languages and increasingly complex applications, alternative software analysis and design methodologies began to multiply during the late 1980s. The number of object-oriented methods increased from less than 10 to over 50 between 1989 and 1994, fuelling the so-called "method wars" (Booch, 1999). As designers and methodologists learned from experience, several of these methods began to converge. Booch and Rumbaugh released a version of the Unified Method in 1995. They were later joined by Jacobson and in September 1997 the Object Management Group (OMG) voted to accept UML as its object modelling standard. UML is now emerging as the software industry's dominant modelling language (Kobryn, 1999) and has been described in numerous books and publications (e.g. Muller, 1997; Alhir, 1998; Eriksson & Penker, 1998; Odell, 1998). UML is not a proprietary notation – it is freely available to all.

4.2 The UML notation

The creators of UML take great care to point out that the notation is a *language* and not a *method*. Thus existing methods can be retained in which the UML notation can replace the original notation in order to achieve standardisation of symbols. UML is designed to be readable on a large variety of media including paper, whiteboards and computer displays. The creators of UML have sought simplicity above all. During the development of UML, the following guidelines were adhered to as much as possible: a clear mapping of concepts to symbols, no overloading of symbols, easy to draw by hand, looks good when printed, can fax and copy well, is consistent with past practice, users can remember it (Rumbaugh, 1996). Some of these principles contradict others at times and some trade-offs had to be made. UML is not a rigid notation and extensions and tailoring are encouraged to suit particular design needs.

Models are browsed and manipulated by means of graphical representations. Many different perspectives can be constructed; each can show all or part of the model and each has one or more corresponding diagrams. UML defines nine different types of diagram: Use case diagrams; Class diagrams; Object diagrams; Collaboration diagrams; Sequence diagrams; Activity diagrams; Statechart diagrams; Component diagrams and Deployment diagrams. A brief description of these diagrams is provided below with emphasis on those pertinent to this research. A full description of the UML notation appears in the UML user guide (Booch, Rumbaugh & Jacobson, 1999). A 400-page encyclopaedia of every UML term is provided in the UML reference manual (Rumbaugh, Jacobson & Booch, 1999).

The various UML diagrams are presented as part of the design for a system to support the operations of a mail-order company. Customers send in orders (in which each line refers to a different item, or set of items). The orders are processed, items are paid for and delivered to the customer and new items are ordered to replenish stock, when required. Operations have been simplified to avoid unnecessarily large and complex diagrams.

Use case diagrams

Use cases have been discussed in detail in Chapter 2. The use case diagram provides an overview of all the use cases in the system and the relationships between use cases and actors. Note that the use case diagram does not include details of each use case and UML does not include a notation for describing this. The details must be provided separately in one of a

number of possible forms, as discussed in Chapter 2. The use case diagram is a "context" diagram, and on its own is not a very powerful formalism (it contains hardly any information about the functionality of the system). It is mainly useful as a structuring aid for other diagrams that describe the system and its actions in more detail (Bergner, Rausch & Sihling, 1998). It also scopes the system and shows its boundaries.

Figure 4.1 shows the use case diagram for the mail-order operation. In this example, there are four *actors* (someone or something that interacts with the system) and three use cases that actors can initiate: - Order Supplies, Process Orders and Update Accounts. The <<extends>> relationship means that the source use case extends the behaviour of the target use case. An extended use case can handle exceptions that are not easily described in the general use case. The <<uses>> relationship means that the source use case includes the behaviour described by the target use case. When a use case uses another, the entire use case must be used.

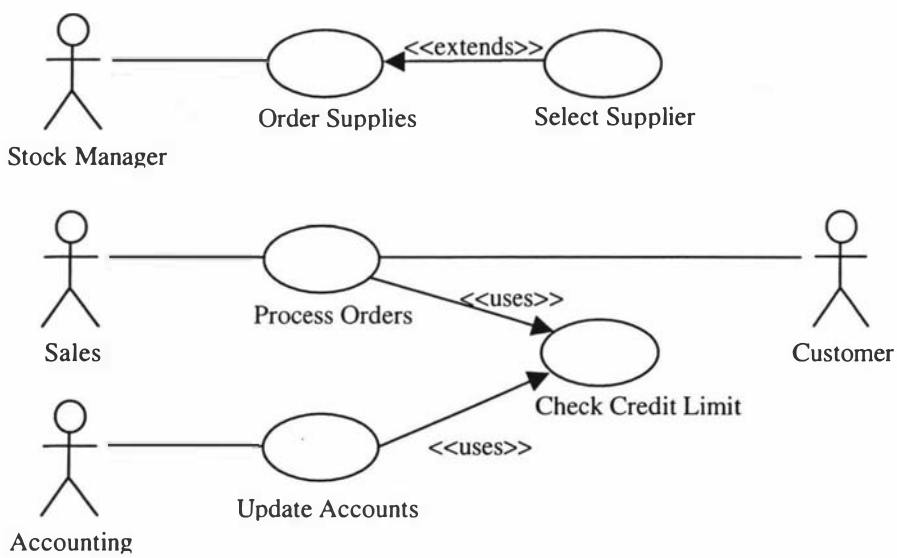


Figure 4.1: An example UML use case diagram

Class diagrams

The definition of classes is at the heart of any object-oriented method. Each class defines the attributes and operations pertaining to any object belonging to that class. Class diagrams express the structure of each class as well as the relationships between classes. Each class has a heading, then a list of *attributes* and finally a list of operations, or *methods*. Associations represent structural relationships between classes and are represented by a line linking the classes. Each association has a *multiplicity value* that indicates how many objects of the given

class may be linked to an object of the other class.

Figure 4.2 presents a class diagram for the mail-order company. Orders are received from customers and each order consists of several order lines. There are two types of customer: corporate customers and individuals. In this example, each Order may be associated with any number (*) of Order Lines but each Order Line may only be associated with one (1) Order. The large arrow indicates a *generalization* and specifies a classification relationship between a general element and a more specific element. In this case, Customer is the general class (or *superclass*) and Corporate and Individual are the specialized classes (or *subclasses*). The attributes, operations and relationships of the superclass are fully *inherited* by the subclasses. For example, the attributes of Individual are Name, Address and Credit card number. Paech (1998) proposes that class diagrams be developed incrementally along with the development of use cases.

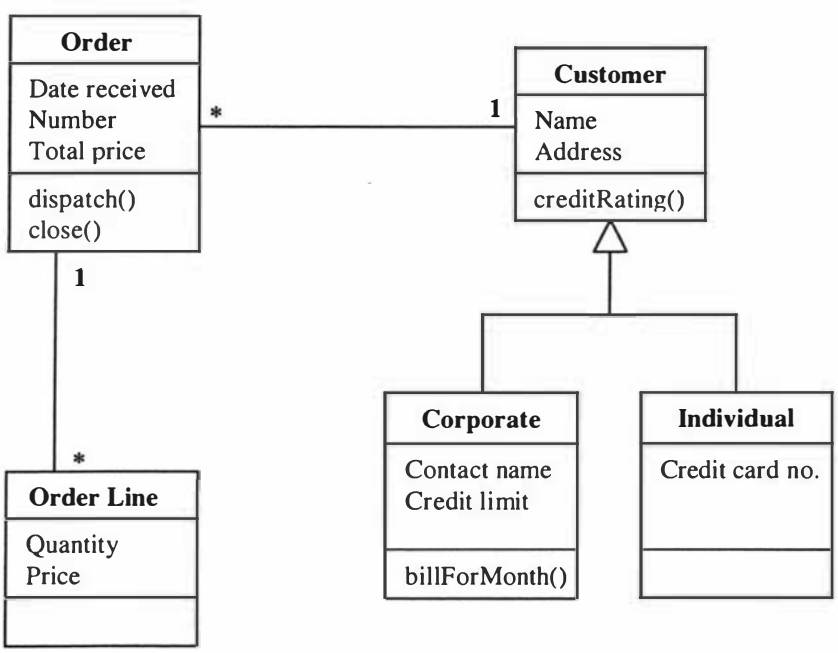


Figure 4.2: An example UML class diagram

Establishing the classes required for a particular project is a difficult task. It is essential that classes be correctly identified as they, along with use cases, form the foundation for the design process. There are various techniques for assisting in the correct identification of classes and they should all be used to some extent in any project. Lee and Tepfenhart (1997) identify the important techniques as: use of domain (expert) knowledge, use of noun phrases and use of CRC cards.

Using noun phrases is a technique of looking through textual descriptions of the use cases and identifying nouns, or objects, in the text. These objects form the basis of classes and also the attributes within classes. Likewise, the identification of verbs, or actions, in the text assists in the identification of the operations required for the class. Work is in progress attempting to automate the creation of classes based on the identification of noun phrases (Overmeyer, 2000).

CRC cards are not part of UML. However, they are widely recommended as useful tools to use along with UML class diagrams as they assist in focusing on the essence of the class (Cantor, 1998; Oestereich, 1999; Stevens & Pooley, 2000; Fowler & Scott, 2000; Richter, 1999). Written on each CRC card is the name of a class, the responsibilities of the class and the collaborators of the class. The responsibilities of the class describe at a high level the purpose of the class's existence, for example, "maintain information about a customer". A class should have only one or two purposes.

CRC cards can be used to walk through use cases, working out how the class model provides the required functionality and where items are missing. They are also used in role playing where different team members hold different sets of cards, each representing a major aspect of the system's responsibilities.

Object diagrams

Object diagrams, or *instance diagrams*, illustrate objects and links. Thus they are similar to class diagrams but at a less abstract level and show a snapshot of the runtime objects and their interconnections in a certain situation. The notation for object diagrams is derived from that of class diagrams and elements that are *instances* are underlined. Each object is represented by a rectangle that contains either the name of the object, the name and the class of the object (separated by a colon), or only the object's class preceded by a colon (in which case the object is said to be *anonymous*).

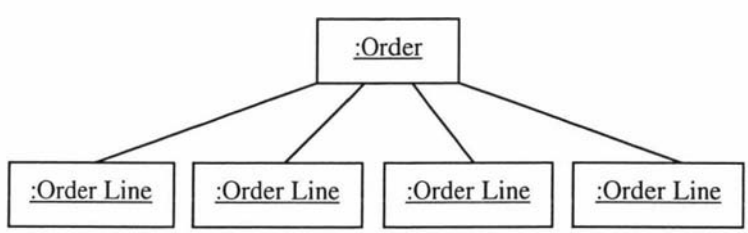


Figure 4.3: An example UML object diagram

Figure 4.3 is an object diagram showing the relationships between an Order and its Order Lines. Note that, although there is only one *class* Order Line, four Order Lines are shown, as these are (anonymous) runtime *instances* of the class.

Object diagrams are primarily used to show a context, for example, before or after an interaction. They are not suitable for the description of large object configurations as they do not scale well and get too large and complicated even for small examples (Bergner, Rausch & Sihling, 1998).

Collaboration diagrams

Collaboration diagrams illustrate interactions between objects and are an extension of object diagrams. They express both the context of a group of objects and the interaction between these objects by representing message broadcasts. An interaction is implemented by a group of objects that collaborate by exchanging messages, represented by arrows. Time is not represented (as it is in a sequence diagram) and the messages can be numbered to indicate the sending order. Note the use of the multiplicity symbol (*) to indicate iteration.

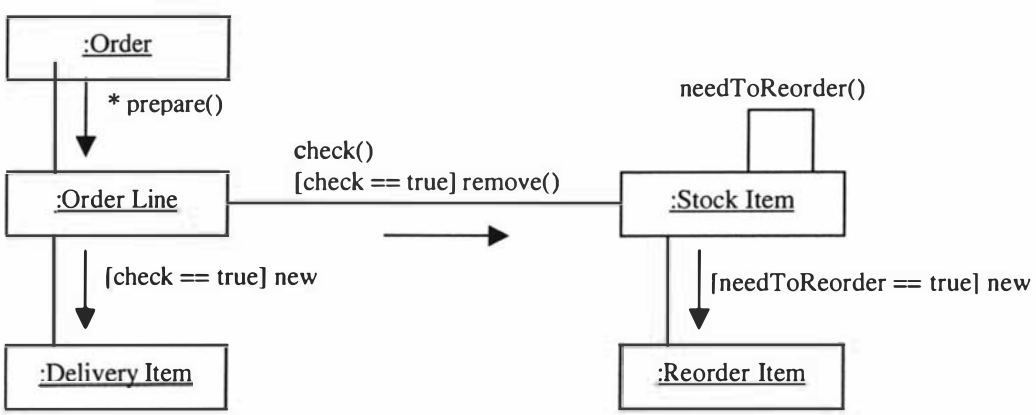


Figure 4.4: An example UML collaboration diagram

Figure 4.4 shows the collaboration between the objects comprising the mail-order system. An Order uses the *method* prepare() to prepare several (*) Order Lines. Each Order Line uses check() to check if the item of stock (that appears in the Order Line) exists. If check is successful then (a) a message commands Stock Item to remove one such item, and (b) a new Delivery Item is created. Meanwhile, Stock Item performs the needToReorder() method (which affects only itself) and, if necessary, creates a new Reorder Item.

Sequence diagrams

Sequence diagrams display interactions between objects with the focus on the sequence of the interactions. Arrows represent messages and the sending order is indicated by the position of the message arrow on the vertical axis. An object that appears lower down than the top of the diagram (see a Reorder Item in Figure 4.5) is created at that point in the sequence. An object may also be deleted which is indicated by ending the dashed "timeline". An object may send itself a message (see needToReorder() in Figure 4.5). Unlike collaboration diagrams, the context of the objects is not shown. Collaboration and sequence diagrams are grouped together under the heading of *interaction diagrams*.

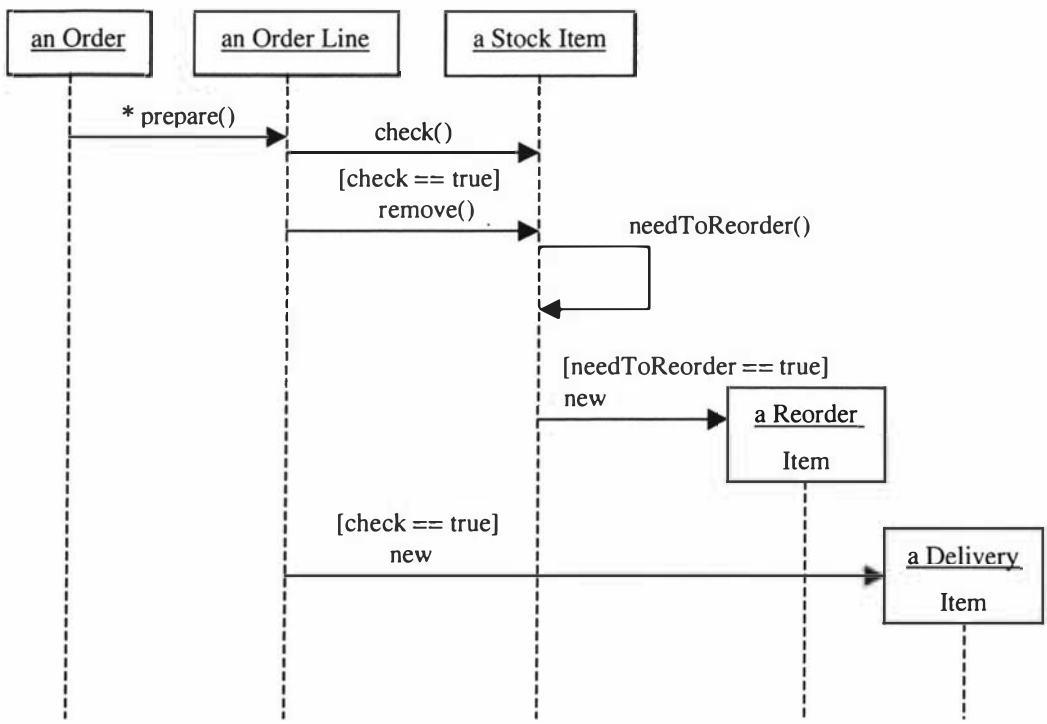


Figure 4.5: An example UML sequence diagram

Figure 4.5 shows the corresponding sequence diagram for the collaboration diagram of Figure 4.4. According to (Bergner, Rausch & Sihling, 1998), the sequence diagram (along with the class diagram) is more useful than the collaboration diagram. Note that the collaboration diagram refers to *objects* whereas the sequence diagram refers to *instances* (e.g. an Order). One problem is that the sequence diagram does not lend itself to concurrent operations. For example, in Figure 4.5, the message from a Stock Item to a Reorder Item could be happening before, after or simultaneously with the message from an Order Line to a Delivery Item.

Activity diagrams

An activity diagram describes a workflow view of the dynamics of a process, a scenario or a use case. It is essentially an upgrade of the traditional *flow chart*. (Paech, 1998) claims that activity diagrams are useful as a bridge between use case diagrams and class diagrams.

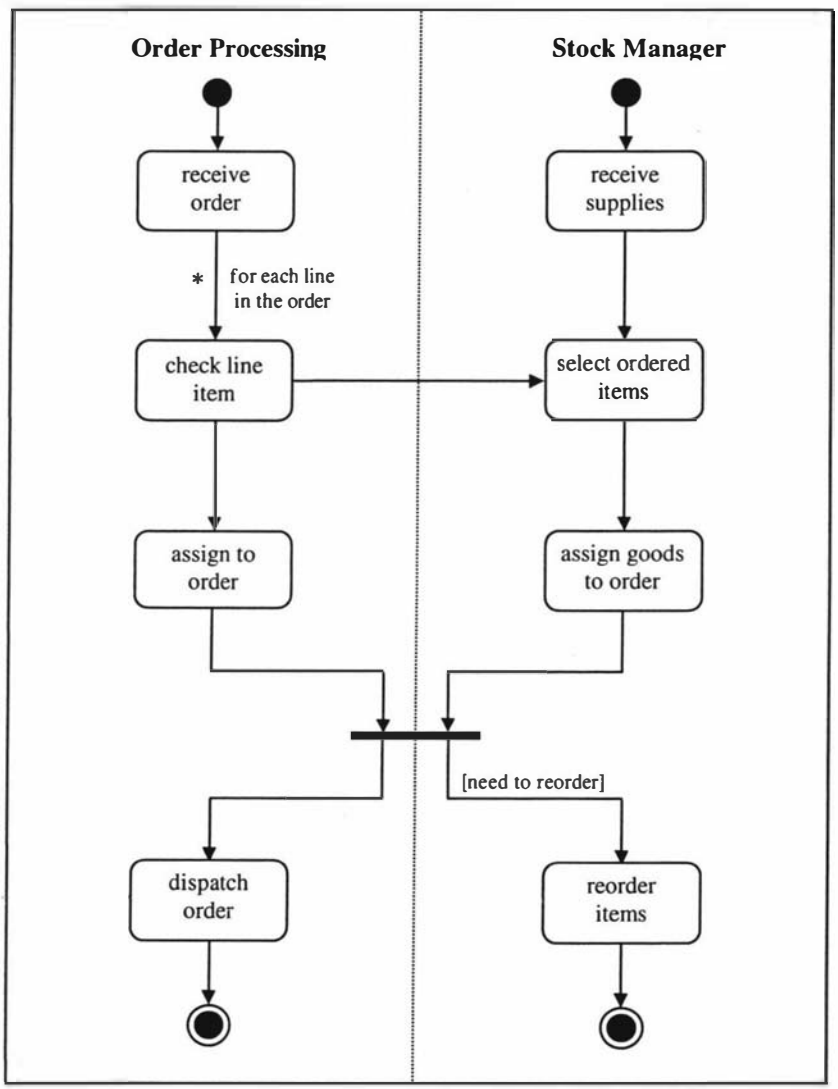


Figure 4.6: An example UML activity diagram

Activity diagrams are particularly useful for representing parallel operations. Synchronization between control flows is achieved by using a *synchronization bar*, thus making it possible to open and close parallel branches within the flow of execution. A synchronization bar may only be crossed when all activities with inputs to the bar have ceased. In order to show the various responsibilities within a system, activity diagrams may optionally be split into *swimlanes* with

the responsible object(s) listed at the top of the lane and each activity allocated to a given lane. The relative position of the lanes is not important and transitions may freely cross lanes. Figure 4.6 provides an activity diagram for the (simplified) order processing segment of the mail-order system that excludes checks if the correct item is in stock, etc. Note the explicit representation of possible concurrency, for example dispatch order and reorder items may happen concurrently (or in any order).

Statechart diagrams

Also known as *state diagrams*, the statechart diagram shows the sequence of states that a scenario, use case, class or system goes through in response to events. State Transition Diagrams (STDs) are a well-known formalism for specifying system states and transitions between them. However, questions about the methodical use in the context of object-oriented modelling are left open in the UML specification (Bergner, Rausch & Sihling, 1998). Guidelines for the use and transition between activity diagrams, state diagrams and interaction diagrams are needed. STDs and statecharts have been discussed in Chapter 2 as possible dialogue modelling notations. It should be noted that statecharts in UML are not envisioned as a dialogue modelling notation, but rather as a notation for representing the various states and transitions of the underlying software.

Component diagrams

The word *component* has many meanings. In this context, a component is "a distributable piece of implementation of a system, including software code but also including business documents, etc. in a human system." (Stevens & Pooley, 2000). Component diagrams specify units such as files of source code, class libraries and executable applications. They indicate choices made at implementation time. In a detailed article (Kobryn, 2000) it is argued that UML does not support component modelling adequately, and that the semantics for components are dated and typically occur near the end of the modelling process. These semantics should be updated to encourage modellers to specify components earlier in the software life cycle, for example during design.

Deployment diagrams

Deployment diagrams show the physical layout of the various hardware components, called *nodes*, as well as the distribution of executable programs on this hardware. Component

diagrams can be overlayed on deployment diagrams to indicate which software components run on which hardware node. Component diagrams and deployment diagrams are grouped together under the heading of *implementation diagrams.*

4.3 The Unified Process

The UML notations, described in Section 4.2 above, are just that – notations. Hence the name: the Unified Modelling *Language.* A language provides structure and syntax but does not prescribe how the language should be used. UML is not a *methodology* and does not prescribe a method or process for analysis and design. Associated with UML is the *Unified Process* (Jacobson, Booch & Rumbaugh, 1999) that consists of a set of concepts and activities needed to transform requirements into a software system. The Unified Process is not a detailed, step-by-step method, but rather a framework that can be adapted to suit local conditions. One of the authors of UML and the Unified Process (Rumbaugh, 1995), stresses that a process can only be a guide to producing a model and it is necessary to include thought and creative endeavour in each design project.

Object-oriented design requires a very different process style compared to the traditional *waterfall* approach. Although conventional techniques can be used, they do not fully utilise the power of objects. A number of object-oriented techniques existed prior to the appearance of UML and a review of some of these appears in (Eriksson & Penker, 1998). Fowler (1997) describes how the new object-oriented approaches should use the technique of *evolutionary delivery* in which software is delivered in a series of steps. Each step delivers a subset of the functionality of the system and the final step produces production code that has been fully documented and tested.

The Unified Process recommends that a process should be iterative and incremental, i.e. one should work continually with an overview of the whole system and gradually refine sections of it. It also recommends that the process should be *use case driven.* Use cases should not only initiate the development process but should also bind it together. Use cases should establish specifications, they should feed into the design and at the end, use cases should be the source from which testers construct test cases.

Four stages are defined for the Unified Process. These stages are not strictly sequential but may overlap, depending on the project at hand. They should not be assumed to be of equal duration and some may consume more or less time than others. These stages are defined as:

- *inception* – get ideas, make a plan, specify requirements
- *elaboration* – create and detail all use cases, prototype user interfaces
- *construction* – develop the software
- *transition* – install and test the software, act on feedback, adapt

These definitions are deliberately vague in order to accommodate a wide variety of software projects. Unfortunately this also means that they are too vague for general use. Rational Software have produced a product called the Rational Unified Process which provides a number of software tools and printed material to enable developers to make more efficient use of the Unified Process. More attention has also been devoted to user interface design (see Section 4.4 below). A description of the Rational Unified Process can be found in (Kruchten, 1999).

A number of books have been published in which the Unified Process is defined at a more detailed level, usually from the point of view of constructing commercial software. (Paech, 1998; Eriksson & Penker, 1998; Oestereich, 1999; Quatrani, 2000; Stevens & Pooley, 2000; Fowler & Scott, 2000) all present case studies using the Unified Process and suggest guidelines to follow when designing software. They all stress that the process should be use case driven and iterative.

The UML diagrams are used in slightly different ways in the various processes. They all begin with the definition of use cases, some form of use case description (usually textual) and one or more use case diagrams. Then classes are developed, using class diagrams and other techniques such as CRC cards. After that the methods tend to diverge. Sequence and collaboration diagrams are sometimes used to ensure that the system is capable of providing the functionality required by each use case. Activity diagrams and statecharts can be used to model task flow. Component and deployment diagrams are used to create an overview of complex systems, particularly where different hardware platforms are required to communicate.

A review of processes is presented in (Stevens & Pooley, 2000). These include the UML-based processes, Fusion and Extreme Programming. Hewlett-Packard developed the Fusion method in 1994, prior to the release of UML. It was influenced by a number of previous ideas and techniques such as Booch's method, OMT, CRC and formal methods. Fusion uses the CRC design process but adds visibility graphs, from Booch's method, to indicate where an object knows of another object's existence (Coleman, Arnold, Bodoff, Dollin, Gilchrist, Hayes & Jeremaes, 1994).

Fowler (1997) also reviews the *elaborational* approach, in which a high level model is gradually elaborated into a working system. In this approach the design model is a detailed executable model and code generation tools take the model and compile it into the final system. This approach would require a level of formalism in the model that has not yet been attained.

Extreme Programming is not a process for software *design* but rather a process for software *development* (Beck, 2000). It includes topics such as producing clear code, how to create efficient test cases, and so on. Although it covers important aspects of software production, the focus is not the same as that of the UML-based methods or the Fusion method. Extreme Programming does also stress the importance of an iterative process releasing final code in small increments.

4.4 UML, the Rational Unified Process (RUP), and user interface design

Prior to the introduction of UML, it had already been established that there was a lack of integration between the software engineering and HCI communities (Jacquot & Quesnot, 1997). Janssen, Weisbecker & Ziegler (1993) make the point that user interface tools cannot make use of the models developed with general software engineering methods and tools. McDaniel, Olson & Olson (1994) state that software design and user interface design must become forged into one methodology and Kemp & Phillips (1998) show that support for user interface design is weak within object-oriented software engineering methods.

As UML is primarily a collection of previously defined notations, no significant change has occurred in regard to user interface design. This is noticeable in many of the books on using UML. The approach taken by (Eriksson & Penker, 1998) is typical in that they state that the design of the user interface should be carried out "separately but in parallel" to other design work. They then opt out completely by stating that user interface design is beyond the scope of a book about designing software using UML. (Cheesman & Daniels, 2001) opt out in a similar manner. Quatrani (2000) mentions that prototyping the user interface is useful and adds that UML can be used to model "for example, a window but not for modelling each of its dialogue boxes or buttons". Stevens & Pooley (2000) state that a separation of user interface and underlying system is useful as it makes modification or replacement of the user interface more feasible. Other books on UML, for instance (Fowler & Scott, 2000; Muller, 1997), do not even have an entry for "user interface" in the index. The creators of UML themselves make the point that the UML is a general purpose modelling language and specialised domains, such as user interface design, may require a more specialised tool with its own specialised language

(Rumbaugh, Jacobson & Booch, 1999).

UML has a tentative approach towards user interface design in that some of the objects in the system are designated as containing operations that involve communicating with users. This is based on the work in (Jacobson, Ericsson & Jacobson, 1995). In the UML notation, an interface is represented as a little named circle next to the appropriate class – known as a *lollipop*. This notation is illustrated in Figure 4.7.

Figure 4.7: UML notation for a class with an interface – a lollipop

This notation may be useful to the system designer in that it indicates which classes require a user interface but it is of little help to the user interface designer as it indicates nothing whatsoever about the structure of the user interface or the interactions required (Bergner, Rausch & Sihling, 1998).

Some of the UML notations are already being used in user interface design:

- State Diagrams - a traditional method for representing the behaviour of event driven systems such as graphical user interfaces. See (Jacob, 1983) and (Phillips, 1994).

- Use Cases - use cases or scenarios have been used extensively in user interface design for some time as a means of capturing requirements and specifying the interaction between the user and the system in the performance of tasks. See (Carroll, 1995) and (Constantine & Lockwood, 1999).

- Activity Diagrams - these are similar to a number of task analysis and task decomposition techniques. See (Dix, Finlay, Abowd & Beale, 1998) and (Shneiderman, 1998). The use of activity diagrams in user interface design is evaluated in (Phillips & Kemp, 2002) which concludes that, although activity diagrams show task flow clearly, they do not identify the actors involved, nor do they differentiate the actions of the actor from those of the system.

Kovacevic (1998) states that UML activity diagrams and interaction diagrams (collaboration and sequence diagrams) are not enough to perform the task modelling required for user interface design. This is supported by (Rourke, 2002) which states that UML diagrams are not

sufficient for task modelling and need to be linked to separate task models. UML interaction diagrams are also described as a "poor fit" with the needs of user interface design (Constantine & Lockwood, 2001).

User interface design within the RUP

User interface design within the RUP involves two stages: user interface modelling and user interface prototyping. The RUP is *use case driven*, so the main input to these activities is the use case model that describes how the system is used. At the user interface modelling stage, each use case is described via a *use case storyboard* which is a high level description of how it is to be supported by the user interface. It includes the following (Krutchen, Ahlqvist & Bylund, 2001):

- *Flow of events storyboard*: a high-level textual description of the use case.
- *Interaction diagrams*: provide a detailed graphical description of how objects interact in the realisation of the use case.
- *Class diagrams*: provide a description of boundary classes and their relationships.
- *Usability requirements*: a textual description of requirements relating to quality and ease of use of the system.
- *References to the prototype*: references to the elements comprising the user interface.

The first three activities are not unique to user interface design as they are also required for the development of the underlying system. Interaction diagrams and class diagrams are well defined in the UML notation. The references to textual descriptions are not particularly useful and lack detail. Ongoing research has provided some alternatives to textual descriptions (Phillips & Kemp, 2002). The linkage to the prototype is also rather vague.

It must be noted that the UML notations were never intended for user interface design, but were designed to represent various aspects of the underlying system. This is made clear in (Fowler, 1997) and (Quatrani, 2000) and is supported by the work of (Kemp & Phillips, 1998). Thus UML in fact provides no explicit support for user interface design, while the RUP provides some support albeit rather lacking in detail.

4.5 Uniting UML with user interface design

The software design process should incorporate user interface design as an important component. The existing UML notations are not sufficient for the task and the RUP is too vague to be helpful. Researchers are following several possible avenues that are reviewed in this section.

Object View and Interaction Design (OVID)

OVID is a method developed by IBM for using existing UML notations in order to perform graphical user interface design. A good overview of the method is presented in (Roberts, Berry & Isensee, 1997). OVID consists of the following steps:

- *requirements analysis* – task analysis leading to use cases
- *modelling* – establishing class diagrams and creating sequence diagrams to model tasks and state diagrams to model dialogue
- *screen design* – creating screens and widgets and choosing interaction techniques
- *prototyping* – screen mockups, storyboards, HyperCard functional prototypes
- *evaluation* – of the design and the models and checks for inconsistencies
- *implementation* – producing the code for the final user interface

OVID appears to suffer from two flaws. First, by using UML diagrams, it has perpetuated the problem of using notations that are not intended for user interface design. Secondly, it uses state diagrams to model dialogue when they are known to be inadequate. A case study using OVID is described in (Corlett, 2000) where it is noted that it is difficult to move from the model to screen design and that a method is needed which bridges the gap between tasks and dialogue.

Extending UML to support user interface design

Work is in progress to add notation and method to UML in order to assist with the design of graphical user interfaces. Phillips, Kemp & Scogings (2001) examine use case modelling from the viewpoint of its suitability as input to the early stages of the *visual* design of GUIs. It is concerned with the link between the task and screen models, and tackles the problem of the lack of guidelines for transforming the results of task analysis into prototype GUI designs. Three use case representations are compared and a two-column tabular representation is seen

as preferable from the viewpoint of supporting user interface design. An example of the tabular representation of a use case appears in Figure 2.4. An *extended* (three-column) tabular use case representation has since been developed (Phillips & Kemp, 2002). The third column identifies the user interface elements required to support the interactions. Work is proceeding on support for the grouping of these elements, within the framework of the RUP, as a first step in prototyping the visible user interface.

Using Lean Cuisine+ in conjunction with UML

It has been shown above that there is a definite gap in UML as regards support for user interface design and that this shortcoming could be remedied with the addition of an appropriate dialogue modelling notation. Unfortunately, as demonstrated in Chapter 2, most such notations model only part of the behaviour of the user interface with a focus on either dialogue structure or task sequence, but not both. The exception is Lean Cuisine+ which is capable of modelling tasks within the context of the interface dialogue.

Lean Cuisine+ has further advantages in that it was specifically developed as a user interface design tool, unlike the UML notations, and it fits easily into an object-oriented approach, unlike earlier notations such as STDs or Petri nets. As described in Chapter 3, Lean Cuisine+ task sequences have several benefits such as representing tasks at a level useful for user interface design, clearly distinguishing system actions from user actions, and providing possibilities for use as an analytical tool – for example, key strokes in a particular sequence could be analysed and minimised.

Lean Cuisine+ is thus a powerful tool for linking dialogue and task modelling. As an example, a Lean Cuisine+ task sequence in Figure 4.9 is compared with a UML activity diagram representing the same task in Figure 4.8. This illustration clearly demonstrates the difference between the isolated task in the activity diagram and the task shown in the context of the dialogue in the Lean Cuisine+ representation. In Chapter 5, a method is developed for combining UML and Lean Cuisine+ during the early stages of user interface design.

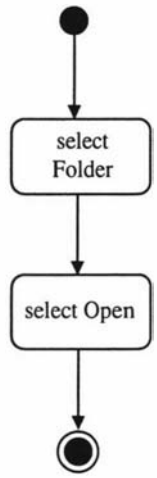


Figure 4.8: UML activity diagram for opening a folder

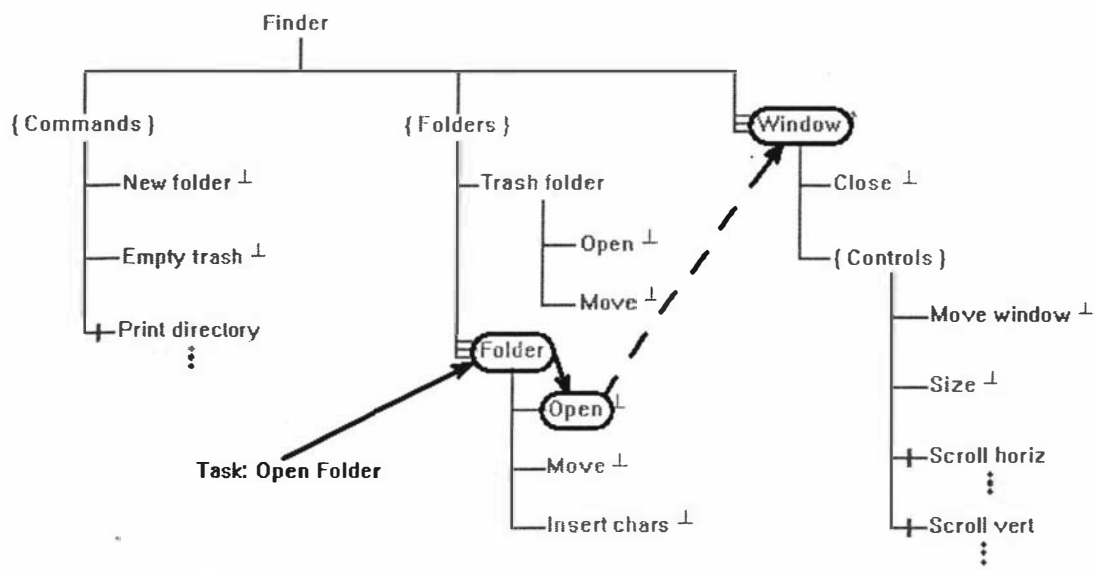


Figure 4.9: Lean Cuisine+ task sequence for opening a folder

Chapter 5. A Method for the Early Stages of User Interface Design

5.1 Introduction

The design of the user interface is one aspect of developing an interactive software system. The suggested advantages of modelling with Lean Cuisine+ must be considered in relation to the wider system development life cycle, to other research, and to emerging tools and methods. The interactive system development life cycle is a highly iterative process that involves five key activities:

- requirements gathering and analysis
- design of the user interface
- design of the underlying software
- system implementation
- system maintenance

It is the first two activities which are of interest here - what might be jointly labelled 'user interface analysis and design'. These activities are now explored further from the viewpoint of system modelling.

Both object modelling and task modelling are components of the 'requirements gathering and analysis' activity. Collectively these two models define the static and dynamic requirements of the system. The 'design of the user interface' activity involves both dialogue modelling and the development of a screen model. The dialogue and screen models each contribute to the design of the 'look and feel' of the system, the former being a behavioural model, and the latter a visual model. The first two activities in the life cycle thus involve the construction of four models: task, object, dialogue and screen.

Task and object modelling are a significant aspect of all current object-oriented analysis and design methods (Artim *et al*, 1998) and have been discussed in Chapters 2 and 3. UML supports both task and object modelling but does not directly support dialogue modelling. More importantly, there is no direct means of relating tasks, captured as use cases and scenarios, to dialogue structure.

The relationship between dialogue and screen models, including the order in which they are developed remains an open issue. Puerta, Cheng, Ou & Min (1999), through their MOBILE system, explore the relationship between a screen model (as produced by an interface builder), a

simple dialogue model, and a task model. The dialogue and task models are both represented in the form of hierarchical decomposition diagrams. This research is an effort to bridge the gap between interface builders and user-centred design. The task and dialogue models are defined in separate tree diagrams, and the MOBILE system supports the allocation of tasks to windows. Phillips & Kemp (2002) explore the relationship between task and screen models through extended use cases and user interface cluster diagrams, within UML and the RUP.

The above research has as a common aim the integration of some of the models involved in interface analysis and design. One of the difficulties in this area is the lack of guidelines for transforming the results of task analysis to produce prototype GUI designs. Some progress has been made, for example in connection with task and object models. What is missing in current development methods, including UML, and in the research discussed in the previous chapters, is a means of representing tasks within the context of the structure of the interface dialogue, i.e. of explicitly showing the transformation of tasks to dialogue. The Lean Cuisine+ notation, in combining task and dialogue models, offers a possible way forward in this regard.

5.2 User interface design as part of the Unified Process

UML is rapidly becoming the *de facto* modelling standard for software design and it would therefore be advantageous to develop a method for user interface design that is compatible with UML notation and can be placed in the context of the Unified Process. The aim is to create a seamless environment wherein the design of the user interface takes place as part of the natural progression of events during the design of the software package as a whole. McInerney & Sobiesiak (2000) state that user interface activities need to be interleaved with general software engineering activities. Brown & Marshall (1998) report that there is strong support for a co-evolutionary design style.

In the arena of user interface design, which is quite separate from UML and the Unified Process, it is common practice to begin the design process with task modelling using use cases and scenarios (Dix, Finlay, Abowd & Beale, 1998; Galitz, 1997; Weinschenk, Jamar & Yeo, 1997; Shneiderman, 1998). The Unified Process demands that the design process should be use case driven. The numerous books on using UML from a practioner's point of view support this. Thus a UML-based user interface design method needs to begin with the establishment of use cases. This will ensure that the proposed user interface design method will fit neatly in with current practice.

The definition of, as well as the creation of, use cases has elicited vigorous debate and has been touched on in preceding chapters. Fowler & Scott (2000) note that use cases can represent different levels of task detail and may subsume other use cases and determining the level of detail remains a challenge for system developers. Several sets of useful guidelines have been established for creating new use cases when commencing a project. UML provides no standard for the presentation of use cases and various techniques exist. Textual use case descriptions are used in Step 1 of the method proposed in the next section.

Object modelling is a core function of any object-oriented approach and is accomplished in UML by constructing class diagrams. The Unified Process suggests that use cases be established and then objects (classes) be derived from these. Object modelling is also a key component of user interface design (Galitz, 1997); for example Weinschenk, Jamar & Yeo (1997) describe the necessity of identifying objects and listing attributes and operations that affect the user. A user interface design method based on class diagrams would thus make the connection to the Unified Process even more secure.

Dialogue modelling is not directly supported in UML, but might be attempted using state diagrams. More significantly, there is no direct means of relating tasks, captured as use cases, to dialogue structure. A CHI'98 workshop (Artim *et al*, 1998) called for a separate user task model to be defined and also that a model of traceability be provided between tasks and their corresponding use cases.

Lean Cuisine+ is a dialogue modelling notation that represents tasks as sequences of actions. It is relatively simple to construct a Lean Cuisine+ task sequence from a given use case. Lean Cuisine+ therefore presents a unique opportunity to place both task and dialogue modelling within the Unified Process.

5.3 The method via a case study

In this section, a method is proposed for the early stages of user interface design. It involves the creation of use cases and classes, the definition of a dialogue structure and the mapping of tasks to dialogue. This method builds on the strengths of UML and Lean Cuisine+ and is presented as a series of steps. However, like the Unified Process, it presents guidelines, not precise rules, and is use case driven and highly iterative. Thus in practice, some steps may be repeated several times or a selection of several steps may be repeated as the process of checking the functionality of each use case takes place.

The method is presented via a case study that concerns the development of part of an on-line library catalogue system. Library users can search for books, request that books be reserved and/or display their lending record. The system requires user identification for the reservation of books or the display of a lending record. The Library Catalogue case study is a particularly good one to use to illustrate the combination of UML and interface design as the use cases and class diagram were constructed as an earlier, separate project (Phillips, Kemp & Kek, 2001). This emphasizes the point that Lean Cuisine+ dialogue can be extracted from independently produced UML use case specifications.

Step 1: Identify use cases

Identifying use cases is the usual starting point for any UML-based design process – thus system design and user interface design commence together. Use cases are identified following the technique described in (Biddle, Noble & Tempero, 2000):

- Identify the actor(s)
- Determine what the system needs to do to support the actor(s)
- Identify the focal use cases and ignore, or delay, peripheral use cases

It is useful to include a use case diagram for an overview of how the use cases and the actors interact.

Four use cases are defined for the library catalogue system and they are illustrated in the UML use case diagram in Figure 5.1. The functionality of the Identify Library User use case is common to both Request Book and View User Lending Record. This is represented in the use case diagram by the UML <<uses>> relationship.

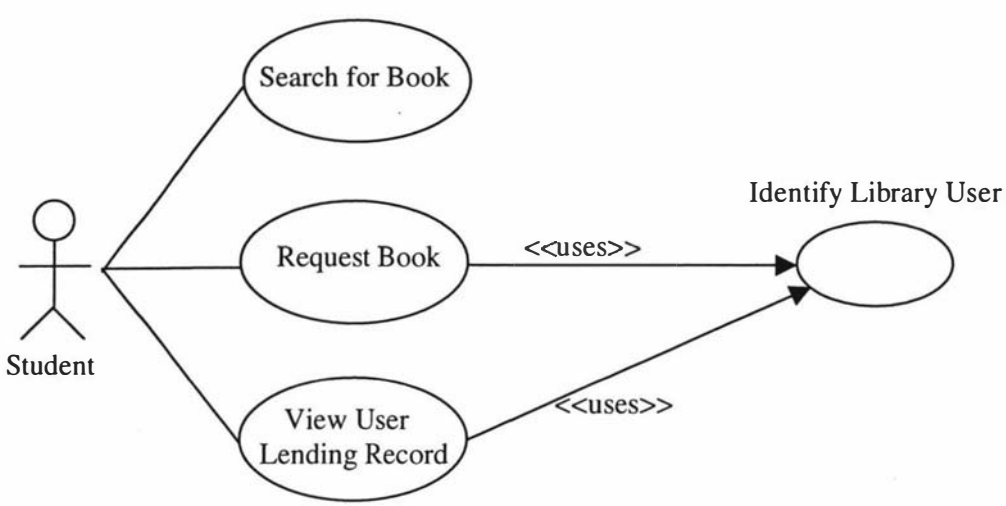


Figure 5.1: UML use case diagram for the Library Catalogue

In order to keep the case study manageable, the use of the library catalogue system has been greatly simplified. A "real" library system would probably require further use cases.

The details of each use case are presented in Figures 5.2 to 5.5.

Use Case:	*Search for Book*
Actor:	Library User (Student)
Goal:	The library user can search for a book.
Preconditions:	None.
Main flow:	The library user activates the use case and then selects the search criterion, which can be one of title, author, ISBN/ISSN, call number, keywords, subject or date of publication. The system searches the catalogue using the search criterion and then displays a list of books that meet the search criterion. This list could be empty. The library user selects a book from the list. The system displays all the information about the selected book. The use case ends.
Exceptions:	None.
Postconditions:	The details of a particular book are visible.

Figure 5.2: Textual description for the *Search for Book* use case.

Use Case:	*Request Book*
Actor:	Library User (Student)
Goal:	The library user can request a book.
Preconditions:	The details of a particular book are visible.
Main flow:	The library user selects a particular copy of the book. (**E1**) *Identify Library User* is activated. (**E2**) The system places the request in the queue for processing. The use case ends.
Exceptions:	**E1** If the selection is invalid, the system prompts the user to re-select. The use case continues. **E2** If *Identify Library User* fails, the use case ends.
Postconditions:	None.

Figure 5.3: Textual description for the *Request Book* use case.

Use Case:	*View User Lending Record*
Actor:	Library User (Student)
Goal:	The library user can check details of checked-out books, requested books or unpaid library fines.
Preconditions:	None.
Main flow:	The library user activates the use case. *Identify Library User* is activated. (**E1**) The library user selects one or more of: **1.** Display checked-out books :- the system displays the list of books. **2.** Display requested books :- the system displays the list of books. **3.** Display fine information :- the system displays the fine information. The use case ends.
Exceptions:	**E1** If *Identify Library User* fails, the use case ends.
Postconditions:	None.

Figure 5.4: Textual description for the *View User Lending Record* use case.

Use Case:	*Identify Library User*
Actor:	Library User (Student)
Goal:	The library user must be identified as a registered user of the system
Preconditions:	None
Main flow:	The library user enters identifying information. The system verifies that this information is correct. (**E1**) The use case ends and reports successful identification. (**E2**)
Exceptions:	**E1** If the information is invalid, the system may request that it be re-entered. The use case continues. **E2** If the information is persistently invalid or the user abandons the system, the use case ends and reports a failure.
Postconditions:	The identification check reports success or failure.

Figure 5.5: Textual description for the *Identify Library User* use case.

Textual descriptions of use cases are recommended by the Unified Process and used by most UML-based methods. The descriptions for this case study are based on a template from

(Oestereich, 1999, pp 108 - 111) and include the name of the use case, the actor who requires the use case, the goal or reason why the use case is necessary or desirable, a description of the usual process and a description of any significant variations to the process.

Step 2: Construct a class diagram

Having established the use cases, the design method proceeds to establish the classes for the main objects in the system. At this point, it is only necessary to establish classes for the *domain objects* (data relating to the user domain), as opposed to *system objects* (such as window and menu) that may be added later. These domain objects are represented as a UML class diagram – see Figure 5.6. Since the method under development is concerned with the design of the user interface, the entire class diagram is not required. Only those attributes and operations that relate to user-system interaction are required as described in (Weinschenk, Jamar & Yeo, 1997).

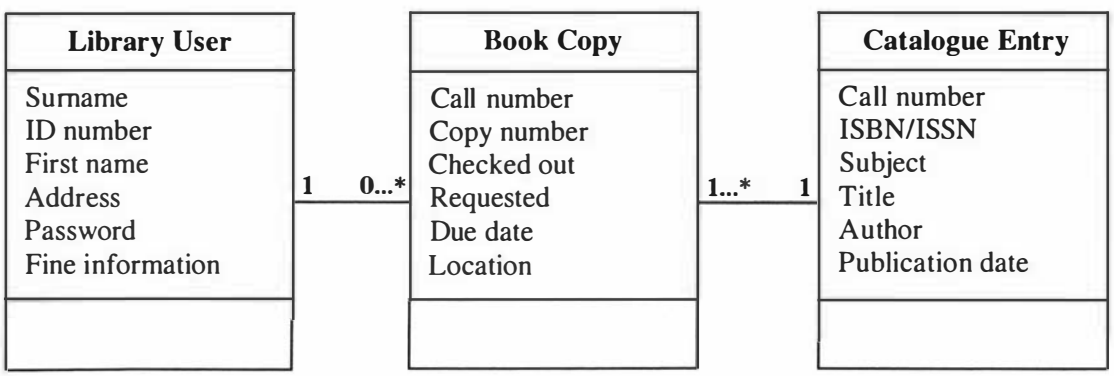


Figure 5.6: The domain classes for the Library Catalogue - *this is not a complete class diagram.*

In Figure 5.6, the relationship between Library User and Book Copy is annotated to indicate that *each* (**1**) Library User is associated with zero or more (**0...***) Book Copies. This means that each student can have several books (possibly none) checked out of the library. Likewise, each (**1**) Catalogue Entry is associated with one or more (**1...***) Book Copies to indicate that any book that appears once in the catalogue may have several physical copies on the shelves. However, relationships between classes are not usually required for the early stages of interface design. Note that any attributes and operations required for user selection *must* be included at this stage.

As the design of the underlying system progresses, the full class diagrams, including all relationships between classes, would need to be completed as part of the system. Thus work on the class diagrams would continue after the separation of user interface design and system design. Paech (1998) notes that class diagrams are developed incrementally as the system is

designed so it is acceptable to be dealing with only parts of a class diagram at different stages of the design.

Step 3: Construct the initial Lean Cuisine+ diagram

The basic Lean Cuisine+ diagram consists of a collection of selectable menemes. These menemes have some correspondence with the domain object classes of the system under construction. Indeed, the whole purpose of a user interface is to provide the user with access to objects in the domain of interest and this is particularly so in the case of *direct manipulation* GUIs (Dix, Finlay, Abowd & Beale, 1998). Thus it is appropriate to base the Lean Cuisine+ menemes and their relationships on the domain object classes of the system.

The basic Lean Cuisine+ tree structure is constructed as follows:

- A subdialogue header meneme corresponding to each domain class name is created. These menemes are created as virtual menemes – subgroup headers that will not be available for selection in the final interface. A virtual meneme is indicated by placing the meneme name within braces {....}.

- A meneme for each *selectable* class attribute in the appropriate subdialogue is created. A selectable attribute is one that may be selected by a user. For example, library users in the case study are never selected by first name – thus no meneme is created to correspond to the first name attribute in the class diagram. The knowledge of which attributes can be selected is obtained through domain analysis. If there is uncertainty, an attribute should be included, as all unused menemes are removed later on.

Note that this process will mean that the menemes in the Lean Cuisine+ tree will tend to have "system oriented" names as opposed to "user oriented" names. However, the tree at this point is merely a framework for the iterative part of the process that is yet to come. If classes obviously have nothing to do with the user interface, they can be omitted from the basic tree. For example, a class may deal with low-level communication protocols. Again, if uncertainty exists, the class should be included at this stage.

Figure 5.7 shows the basic dialogue tree structure for the library case study. The meneme names are a subset of those appearing in the class diagram in Figure 5.6. *Non-selectable* attributes have not been included as menemes. For example, books are never selected by location, so Location is omitted as a meneme in the {Book copy} subdialogue. It must be

emphasized that this is the starting point for an iterative process and the diagram is not intended to be complete at this point.

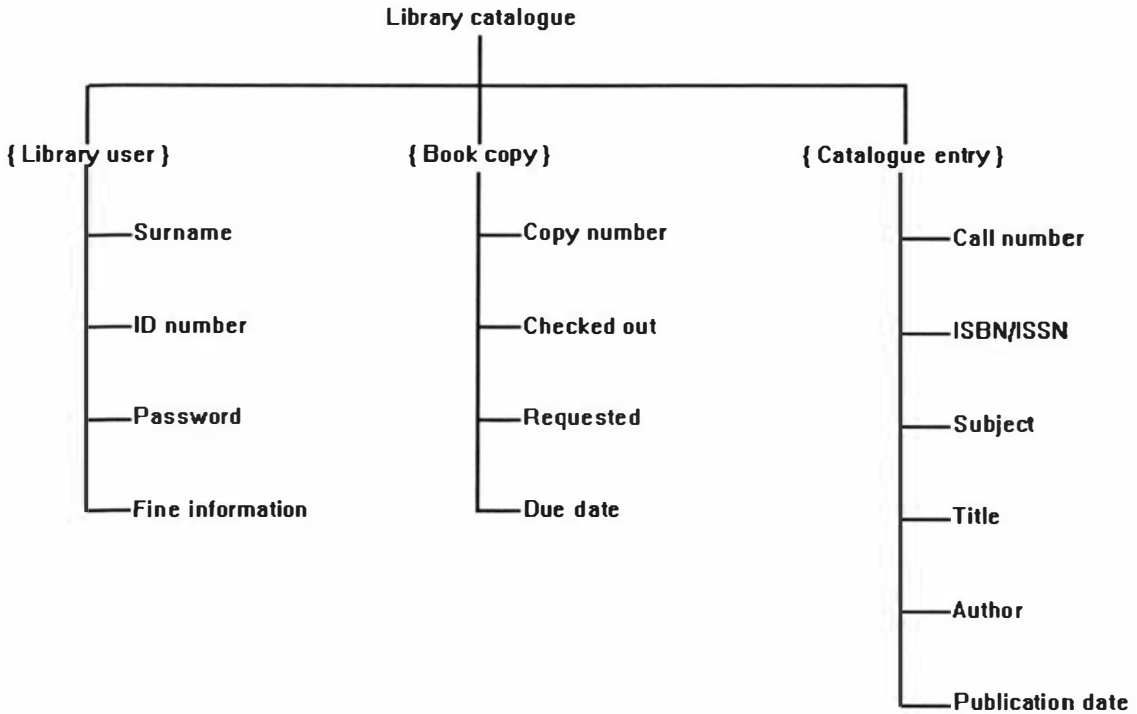


Figure 5.7: First draft of the basic Lean Cuisine+ diagram for the Library Catalogue

The way menemes are arranged within a Lean Cuisine+ subdialogue indicates whether that group of menemes is mutually exclusive (1 from n) or mutually compatible (m from n). At this early stage of development of the diagram it may be difficult to decide which type of structure is required for each subdialogue and the initial structure may require revision at a later stage. It is assumed that meneme groupings within subdialogues are mutually exclusive until known to be otherwise. Sometimes the correct structure can be identified early on, in which case the mutually compatible representation can immediately be specified.

Step 4: Construct initial Lean Cuisine+ task sequences

Lean Cuisine+ task sequences are now overlayed on the initial base diagram. These task sequences are directly derived from the use cases. A task sequence may correspond to an entire use case but it is more likely to correspond to a scenario (one path through a use case). Figure 5.8 shows the initial task sequence for the *Search for Book* use case superimposed on the initial tree of Figure 5.7.

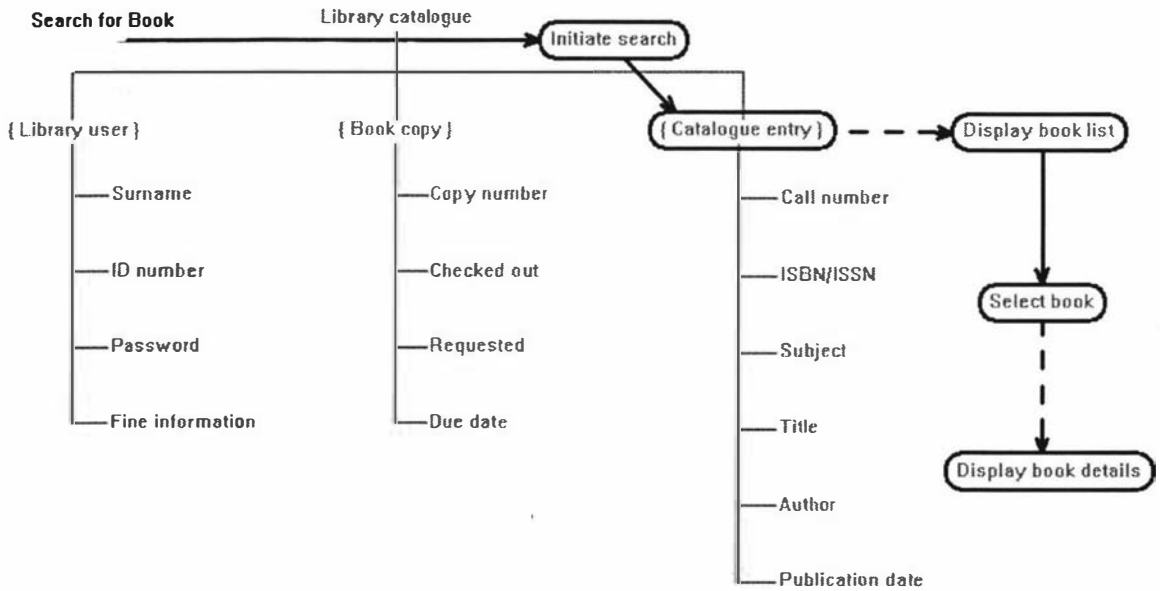


Figure 5.8: First draft of the task sequence for *Search for Book*.

During the construction of the task sequence, it is common to find that some of the required actions do not have a corresponding meneme. In each such case, a new meneme is created and thus the construction of the task sequence can continue. At this stage, these newly created menemes are not connected to the Lean Cuisine+ tree diagram and so they are known as *floating* menemes. The creation of floating menemes highlights the iterative nature of the method. There is an inherent checking process whereby the construction of the dialogue is constantly guided by the tasks that are required. At this stage, all actions in the task sequences are assumed to be user actions. If it is known that certain actions are system actions, those actions can immediately be represented as such.

In Figure 5.8, four floating menemes, Initiate search, Display book list, Select book and Display book details, have been added to the diagram. During the construction of the task sequence it was discovered that these menemes were required but had not been provided. They will be included in the next iteration of the tree structure. The dashed link to Display book list indicates that this is a system action. This is derived from domain knowledge - in this case the system (not the user) causes the list of books to be displayed. It has also become obvious that {Catalogue entry} is not a good name for the subdialogue that allows the user to select the search criteria. Several names will be changed during step 5 as part of the transition from class diagram to interface design. Virtual menemes cannot be selected and the "selection" of the virtual meneme {Catalogue entry} indicates that any one (but only one) of the menemes within the mutually exclusive subdialogue may be selected at that point.

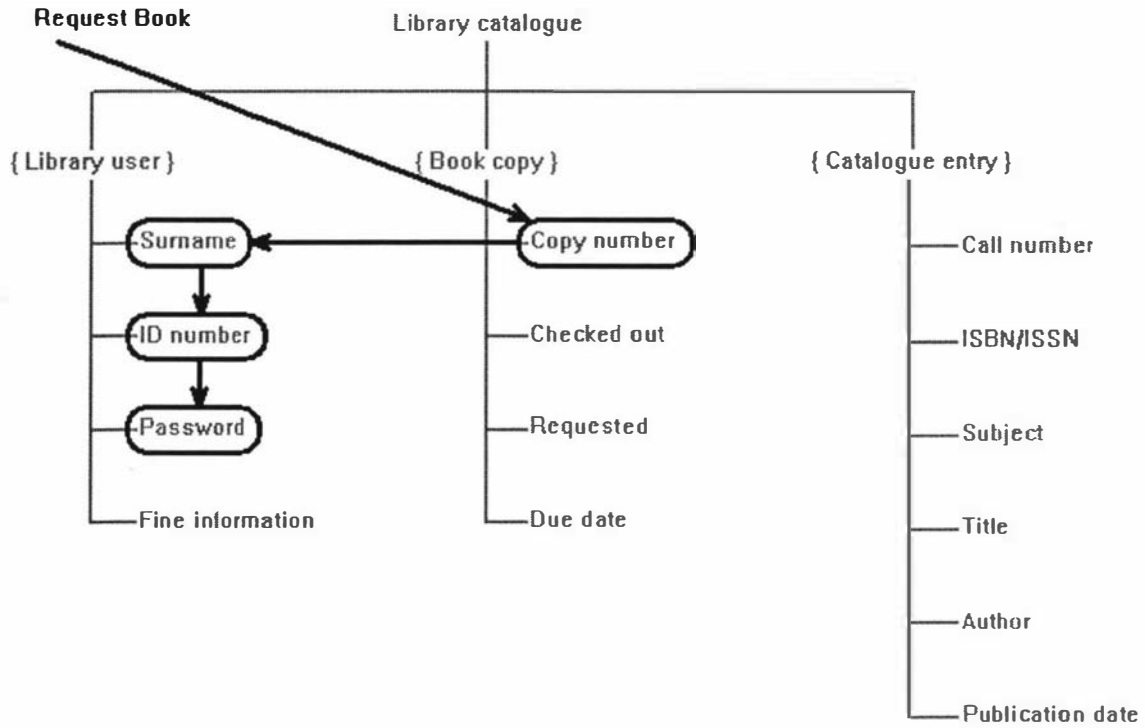


Figure 5.9: First draft of the task sequence for *Request Book*

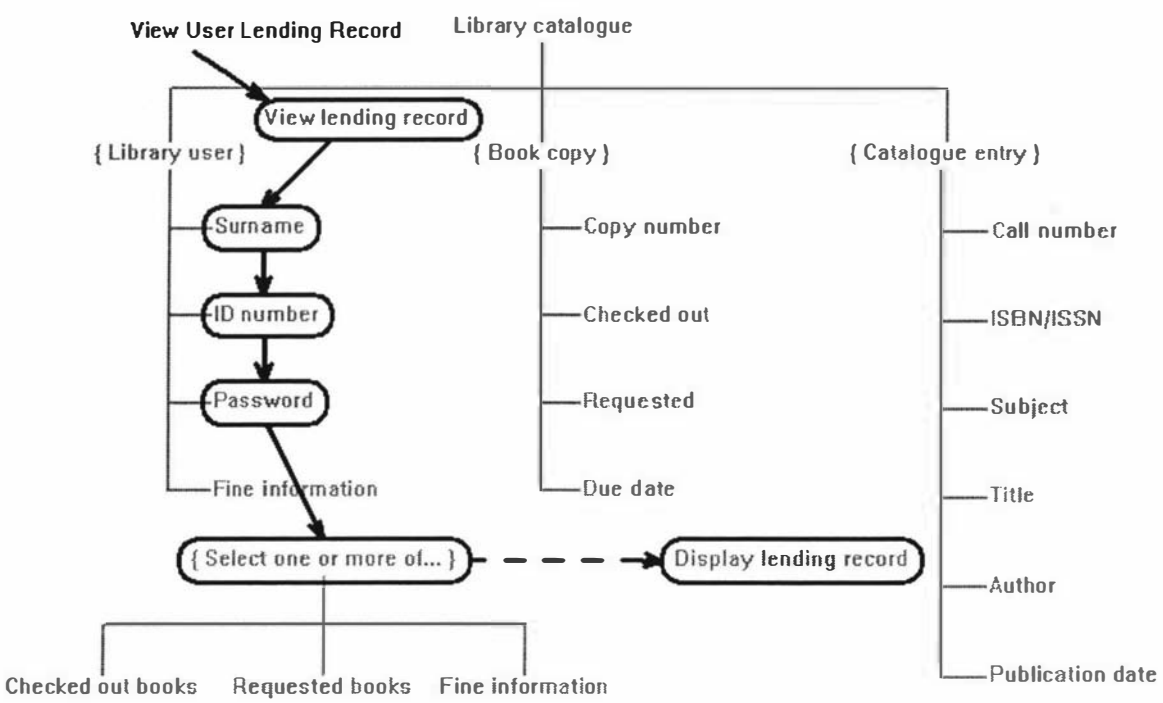


Figure 5.10: First draft of the task sequence for *View User Lending Record*

One (or more) task sequence(s) is constructed for each use case. Figures 5.9 and 5.10 illustrate the task sequences constructed for the *Request Book* and *View User Lending Record* use cases respectively. Each task sequence introduces new floating menemes that will need to be included in the next iteration of the Lean Cuisine+ dialogue tree.

A Lean Cuisine+ task sequence has not been constructed for the use case *Identify Library User* since this use case is used by *Request Book* and *View User Lending Record* – see Figure 5.1 – and has been included in the task sequences for those two use cases. It could easily be represented as a separate task sequence if that were deemed to be useful.

Step 5: Refine the Lean Cuisine+ dialogue tree

The initial task sequences are reviewed to determine the following:

- menemes that are required but are not yet present in the diagram (called floating menemes);
- menemes that are present but are never used by any task sequence (redundant menemes).

The presence of redundant menemes indicates one of two possibilities: either the meneme should not exist and was erroneously created by including a non-selectable attribute from the class diagram, in which case the meneme should be deleted; or there is a missing task and an additional task sequence should be constructed that includes the redundant meneme(s).

The Lean Cuisine+ tree diagram is now revised to exclude any redundant menemes and also to include any floating menemes created during step 4, which must be attached to appropriate subdialogues. Menemes can also have their names changed and be regrouped in order to provide a more logical and efficient dialogue structure. New subdialogues may be formed and existing subdialogues may be disbanded. Decisions are required on which meneme groupings should be mutually exclusive or mutually compatible, which menemes should represent homogeneous groups and thus require *forks* in the diagram and which subdialogue headers should no longer be represented as virtual menemes.

Figure 5.11 illustrates refinements made to the Lean Cuisine+ tree structure for the Library case study. This refined dialogue tree should be compared with the initial tree structure in Figure 5.7.

The meneme Due date has been removed as it was not used in any task sequence. Floating menemes were identified as being required by the task sequences but were not present in the first draft of the tree diagram and thus the following menemes are included in the refined tree

structure: Initiate search (renamed Book search), Display book list, Select book (renamed Book), Display book details, View lending record (renamed View record) and Display lending record.

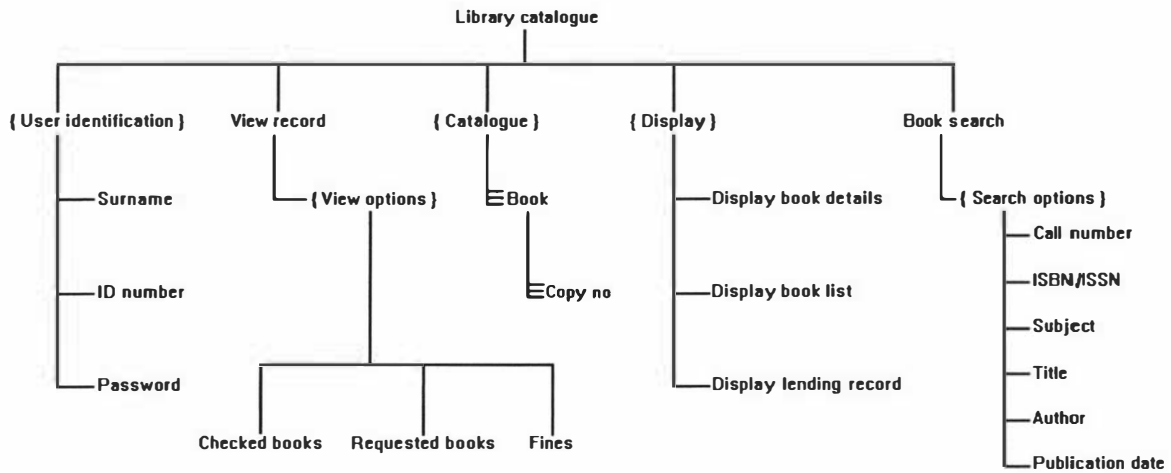


Figure 5.11: The refined tree structure for the Library Catalogue.

Subdialogues were named after domain classes and the name is not always appropriate from a user interface perspective. For example, the domain class Library user contains the data that describes a library user but this information appears in the user interface to *identify* the library user. Thus the subdialogue {Library user} has been renamed {User identification}. Likewise, the information comprising a copy of a book makes up the catalogue in the user interface and hence {Book copy} has been renamed {Catalogue}. Similarly the data attributes of the Catalogue Entry class are used in the user interface to search for an appropriate book and thus the subdialogue {Catalogue entry} has been renamed {Search options}.

The {Display} subdialogue was created to group the "display" actions – all previously floating menemes. The View record subdialogue was created to group together all menemes relating to information describing the user's own library record. These menemes were created by selecting and renaming menemes from the former {Book copy} subdialogue and this process once again highlights the change from data attributes of the Book Copy *class* to selectable options in the user interface: thus Checked out has been changed to Checked books, Requested has been changed to Requested books and Fine information (from the former {Library user} subdialogue) has been changed to Fines.

Significantly, this is the point where the design process changes from a *system-centred* approach to a *user-centred* approach and the designer now embarks on user interface design and moves away from the system design, which would continue in parallel. Thus menemes that were

constructed from class attributes (internal to the system) now change (in name and logical structure) to reflect the user's view of the system.

In Figure 5.11, it is known that the library user needs to be able to simultaneously select one, two or three menemes in the {View options} subdialogue so the menemes in this subdialogue are arranged to indicate a mutually compatible group (the menemes in the group are displayed horizontally instead of vertically). The *fork* symbol has been added to two menemes to represent homogeneous groups: Book is a mutually exclusive group as many books appear in the catalogue but the library user may only view one at a time in the user interface, and Copy no is also a mutually exclusive group as each book may have several copies in the library but the user may only select one copy at a time.

This phase of the design demonstrates the unique power of Lean Cuisine+ to present the task model within the context of the dialogue model. The task sequences provide immediate input into the construction of the dialogue model and ensure that the final interface will be efficient without unnecessary and redundant options. Conversely the presence of redundant menemes in the dialogue can lead to the discovery of a missing task sequence – a potential source of great irritation to a client in a commercial environment – that can then immediately be included.

Step 6: Enrich the Lean Cuisine+ diagram

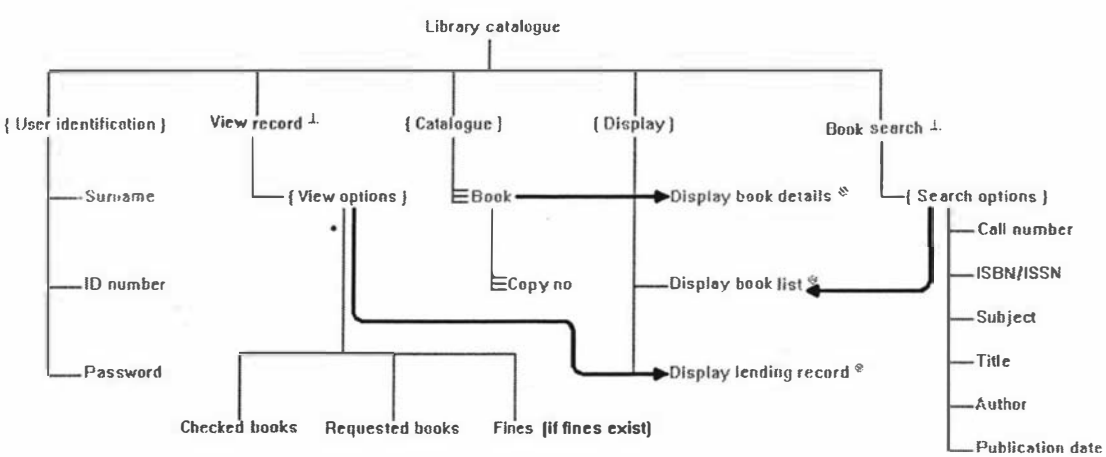


Figure 5.12: Selection triggers and meneme designators for the Library Catalogue

Selection triggers are added to the diagram. These correspond to the system actions in the task sequences. In Figure 5.12 the trigger from Book to Display book details indicates that the selection of Book by the user will trigger the selection of Display book details by the system. The trigger from {View options} to Display lending record indicates that the selection of one or more of the menemes in the {View options} subdialogue will trigger the selection of Display lending record by

the system. Note that {View options} itself can never be selected as it is a virtual meneme. Similarly, the third trigger indicates that the selection of one of the menemes in the {Search options} subdialogue will trigger the selection of Display book list.

Conditions can be added to the diagram. For example, in Figure 5.12, a condition indicates that the meneme Fines may only be selected if the user has actually received one or more fines. Meneme designators are also added to the diagram at this stage. Most menemes can be selected or deselected by the user and the designators indicate the exceptions to this rule. A *monostable* meneme (⊥) can be selected by the user, but then reverts to an unselected state on completion of the operation it represents. A *passive* meneme (⊗) cannot be selected or deselected by the user but can be selected by the system. A dot next to a subdialogue (see {View options}) indicates an *unassigned default* choice that takes on the value of the last user selection from the subdialogue.

Step 7: Remap the task sequences

The Lean Cuisine+ task sequences (first constructed in step 4) are now remapped on to the refined dialogue tree. Each task sequence should still proceed through the same actions, but the underlying meneme for each action may have a new name and/or be in a new position in the tree. It is possible that further floating menemes may still have to be created. The revised task sequence for *Search for Book* is illustrated in Figure 5.13 and is derived from the initial task sequence presented in Figure 5.8.

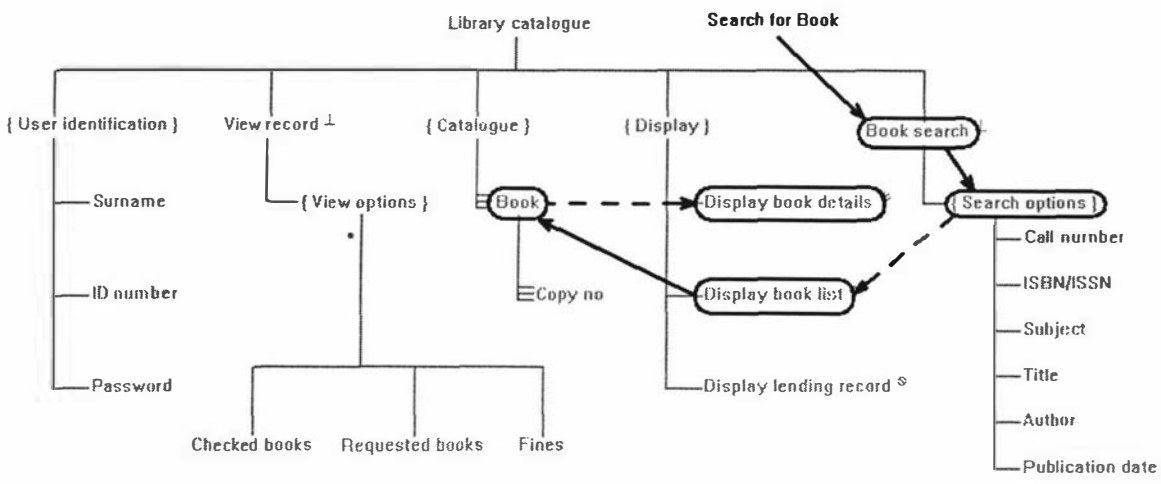


Figure 5.13: Revised task sequence for *Search for Book*.

Likewise, the revised task sequences in Figures 5.14 and 5.15 are derived from the initial task sequences in Figures 5.9 and 5.10 respectively.

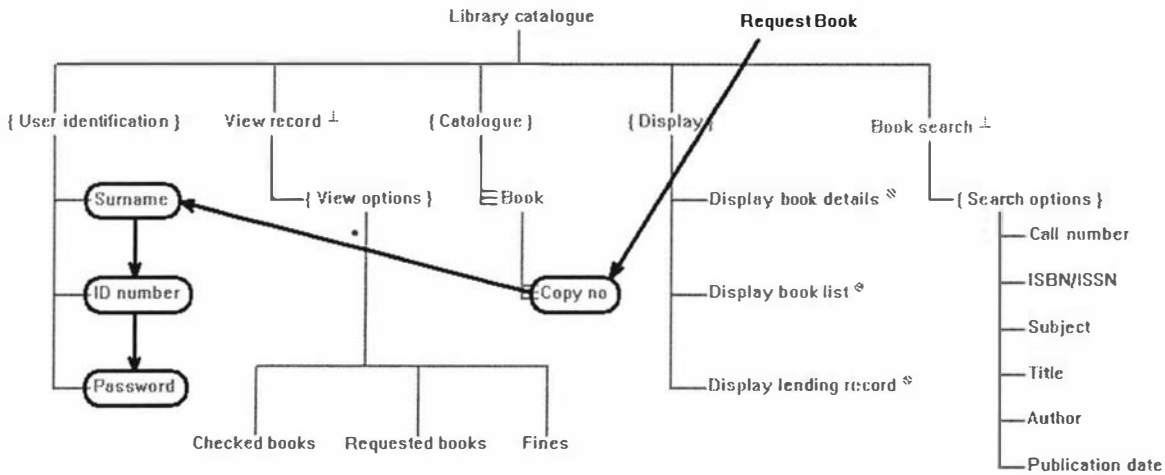


Figure 5.14: Revised task sequence for *Request Book.*

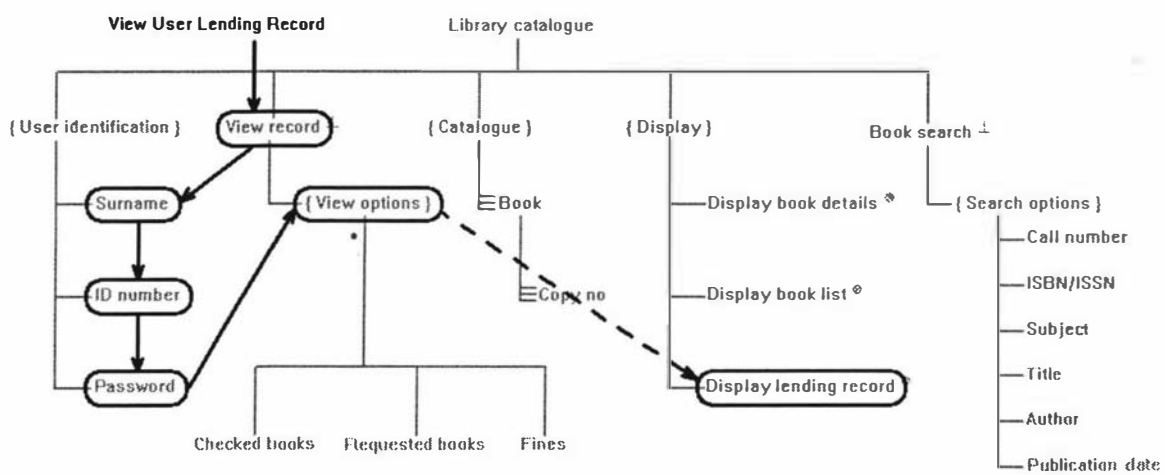


Figure 5.15: Revised task sequence for *View User Lending Record.*

The process is highly iterative and steps 5, 6 and 7 should be repeated until all task sequences are represented and no redundant menemes are present.

Analysis and optimisation

Lean Cuisine+ is unique among modelling notations in that the task sequences are represented within the structure of the dialogue. The previous phases of the method (steps 1 to 7) have been used to construct the task sequences from the UML use cases and class diagrams and also to verify that the task sequences match the use cases. This provides a good platform from which to perform analysis of the user tasks.

The task sequences should be analysed in order to make the dialogue structure more efficient and user-friendly. It is difficult to provide a set formula for this process but one obvious ideal is to reduce the number of selections required to execute a task. For example, in Figure 5.13 the consecutive selection of the menemes Book search and {Search options} could be combined into one selection. Figure 5.16 illustrates how the shortened task sequence (and the suitably modified Lean Cuisine+ tree structure would then appear.

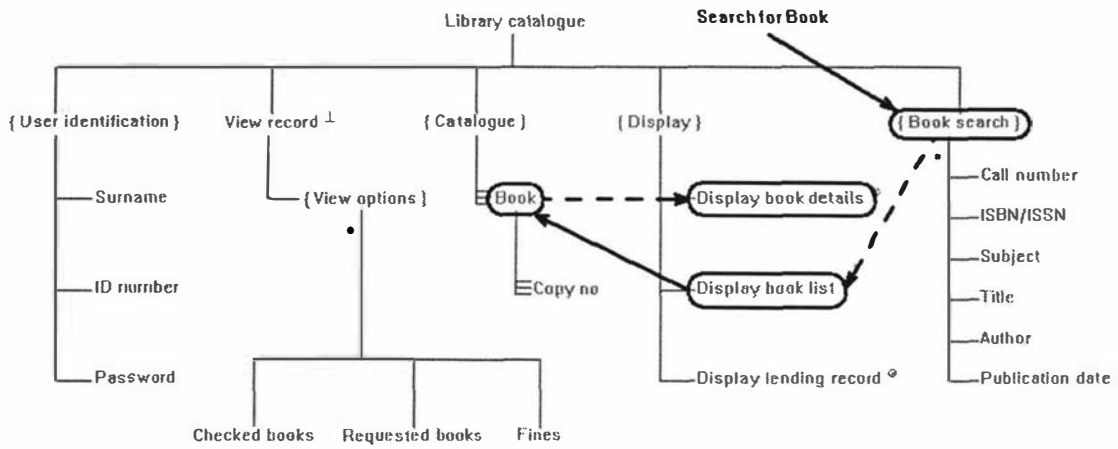


Figure 5.16: A shortened form of the task sequence for *Search for Book*.

It may also be possible to identify task sequences that often jump from one subdialogue to another and which may subsequently cause unnecessary opening and closing of various views or windows in the final user interface.

5.4 A summary of the method

The method described above can be summarised as follows:

1. Define a set of detailed UML use case descriptions.

2. Identify the important domain classes and construct a UML class diagram that does not have to be complete, but must include any attributes that may be selected by a user.

3. Construct an initial Lean Cuisine+ dialogue tree structure by using the domain class names as subdialogue headers and creating other menemes corresponding to the selectable class attributes. Meneme groupings are assumed to be mutually exclusive unless known to be mutually compatible. Each subdialogue header is represented as a virtual meneme.

4. Construct initial Lean Cuisine+ task sequences from the use cases. There must be at least one task sequence for each use case, but there may be more than one if the use case contains significantly different scenarios. These task sequences are superimposed on the initial tree structure and may include new *floating* menemes that are not yet allocated to a subdialogue.

5. Review and refine the Lean Cuisine+ tree diagram. Include the floating menemes identified in **(4)** above and remove unused menemes. Possibly construct new task sequences and form subdialogues of menemes according to functionality. Modify the subdialogue structure where appropriate. Rename menemes to reflect the user point of view.

6. Enrich the Lean Cuisine+ model. Add selection triggers to correspond to system actions in the task sequences. Add conditions and meneme designators as required.

7. Remap the task sequences on to the modified tree structure.

Steps **5**, **6** and **7** may need to be repeated as part of an iterative design process. At each stage, the task sequences need to be analysed to ensure that:

- all use cases are covered in the model;
- no required menemes are missing;
- no menemes are superfluous;
- the flow of each task sequence is logical and efficient.

5.5 A further case study

The method is applied to a second case study that concerns the development of a Timetable Viewer software package to be used at universities to provide reports (visual and printed) on the academic timetable and on venue allocation. The various users – students, the facilities manager and the timetable manager – require different views of the information.

Step 1: Identify use cases

Four use cases are defined for the Timetable Viewer system and appear in the UML use case diagram in Figure 5.17. The use case diagram shows the four use cases: Create Student

Timetable, Create Room Timetable, Check for Clashes and Check for Double Bookings. The diagram also indicates which actor (or user) will be using which use case.

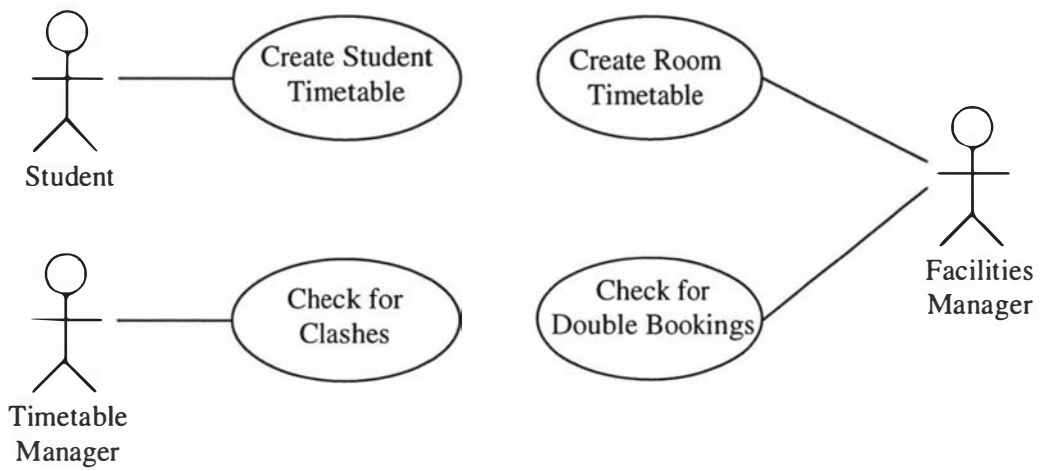


Figure 5.17: UML use case diagram for the Timetable Viewer.

Textual descriptions are provided for each use case using the same format as that used for the initial case study in section 5.3. The details of each use case are presented in Figures 5.18 to 5.21.

Use Case:	*Create Student Timetable*
Actor:	Student
Goal:	The student wants to print out her own timetable.
Preconditions:	None.
Main flow:	The student selects a semester and then selects a number of courses. As each course is selected, the system includes the times for that course 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 and the use case terminates.
Exceptions:	The student selects the wrong course. To correct the problem, the student may deselect the course. The system displays the timetable without the deselected course.
Postconditions:	None.

Figure 5.18: Textual description for the *Create Student Timetable* use case

Use Case:	*Check for Clashes*
Actor:	Timetable Manager
Goal:	The Timetable Manager wants to check whether a specific course is taught at the same time as any other courses. i.e. produce a *clash list* for the selected course.
Preconditions:	None.
Main flow:	The Timetable Manager selects a semester and one course. The system displays a list of courses which clash with the selected course. The Timetable Manager can select to display only lectures, only tutorials or only laboratories; or any combination of these. The clash list may be amended due to these changes. The final clash list can then be printed and the use case terminates.
Exceptions:	None.
Postconditions:	None.

Figure 5.19: Textual description for the *Check for Clashes* use case

Use Case:	*Create Room Timetable*
Actor:	Facilities Manager
Goal:	A timetable is required which shows the activities in a particular room.
Preconditions:	None.
Main flow:	The Facilities Manager selects a semester and a room. A timetable is displayed showing all activities in that room. The timetable can then be printed and the use case terminates.
Exceptions:	None.
Postconditions:	None.

Figure 5.20: Textual description for the *Create Room Timetable* use case

Use Case:	*Check for Double Bookings*
Actor:	Facilities Manager
Goal:	A list of double bookings is required.
Preconditions:	None.
Main flow:	The Facilities Manager selects a semester. The system checks all activities in all rooms and displays a list of double bookings, i.e. two exclusive events booked into a room at the same time. The list can then be printed and the use case terminates.
Exceptions:	None.
Postconditions:	None.

Figure 5.21: Textual description for the *Check for Double Bookings* use case

Step 2: Construct a class diagram

The classes are established for the Timetable Viewer system. Only those attributes and operations that relate to user-system interaction are required.

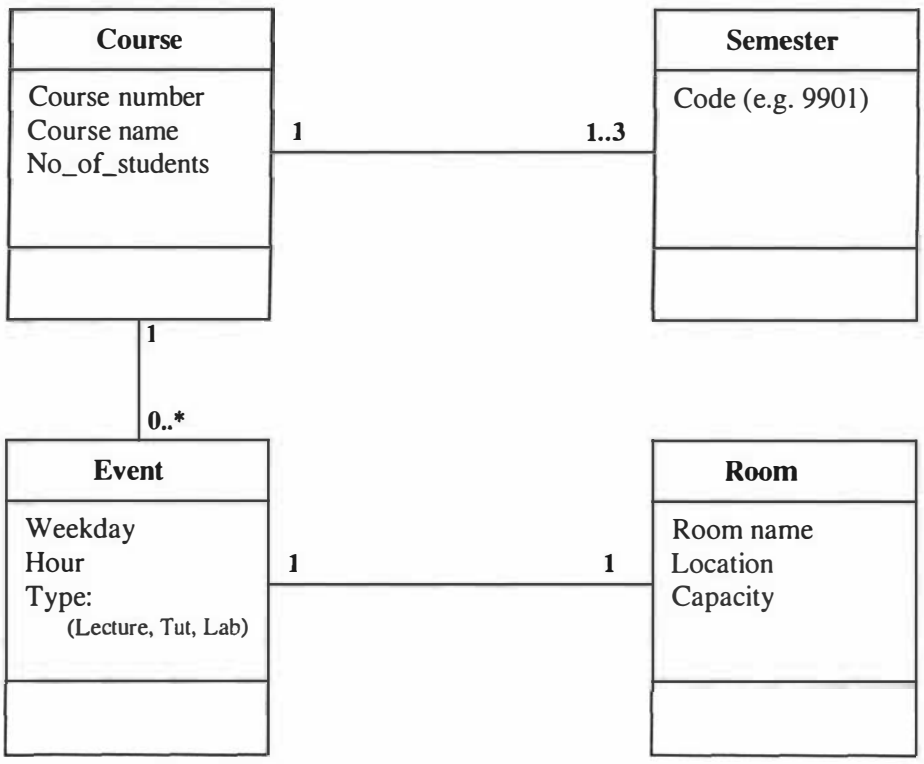


Figure 5.22: Domain classes for the Timetable Viewer - *this is not a complete class diagram.*

In Figure 5.22, the relationship between Course and Semester is annotated to indicate that each (1) course is associated with one, two or three (1..3) semesters during the academic year. Likewise, each course is also associated with zero or more (0..*) events and each event is associated with one (1) room. This class diagram is not complete. The full class diagram would be completed as part of the system design, but not part of the user interface design.

Step 3: Construct the initial Lean Cuisine+ diagram

The basic Lean Cuisine+ tree structure is constructed as before. That is, a virtual subdialogue header meneme is created corresponding to each domain class name, and a meneme is created for every selectable attribute. Figure 5.23 shows the basic dialogue tree structure constructed for the Timetable Viewer system.

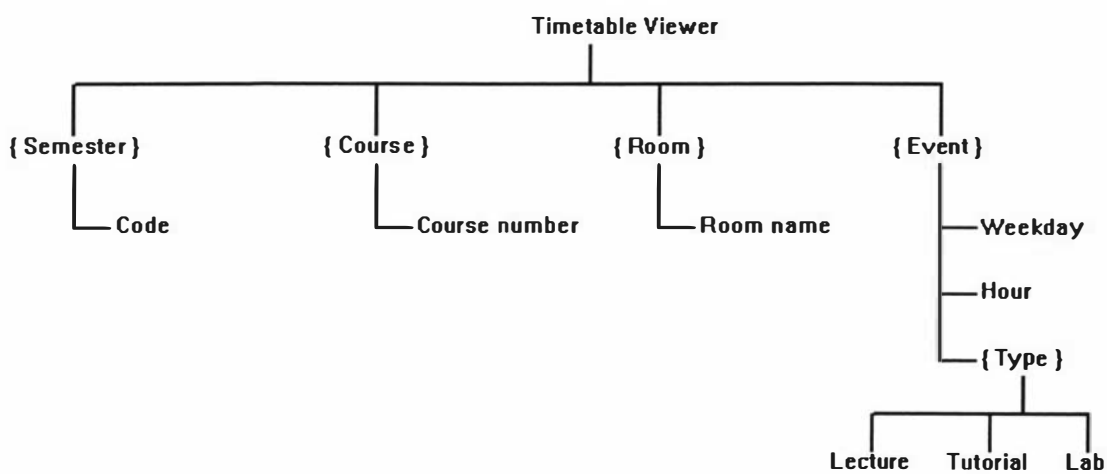


Figure 5.23: First draft of the basic Lean Cuisine+ diagram for the Timetable Viewer

At this stage, it is known that courses are never selected by course name or number of students, so menemes are not created for these attributes. In the same way, menemes are not created for location or capacity in the {Room} subdialogue. It is also known that event type (lecture, tutorial or lab) can be viewed in any combination. Hence the {Type} subdialogue is represented as mutually compatible whereas all other subdialogues are assumed to be mutually exclusive. Some of them may be changed in later drafts of the diagram.

Step 4: Construct initial Lean Cuisine+ task sequences

The task sequences are derived from the use cases and appear in Figures 5.24 to 5.27.

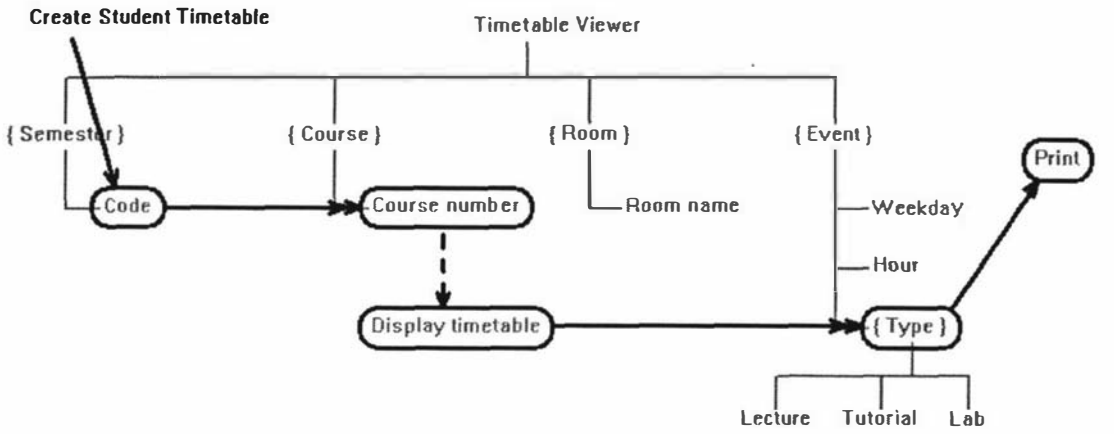


Figure 5.24: First draft of the task sequence for *Create Student Timetable*

In Figure 5.24, two floating menemes, Display timetable and Print, have been added to the diagram in order to complete the task. The dashed link from Course number to Display timetable indicates that it is a system action. This is derived from domain knowledge – in this case the system (not the user) causes the timetable to be displayed. The double-headed arrow leading to Course number means that any number of Course numbers can be selected. Similarly, the double-headed arrow leading to {Type} means that any type combination may be selected.

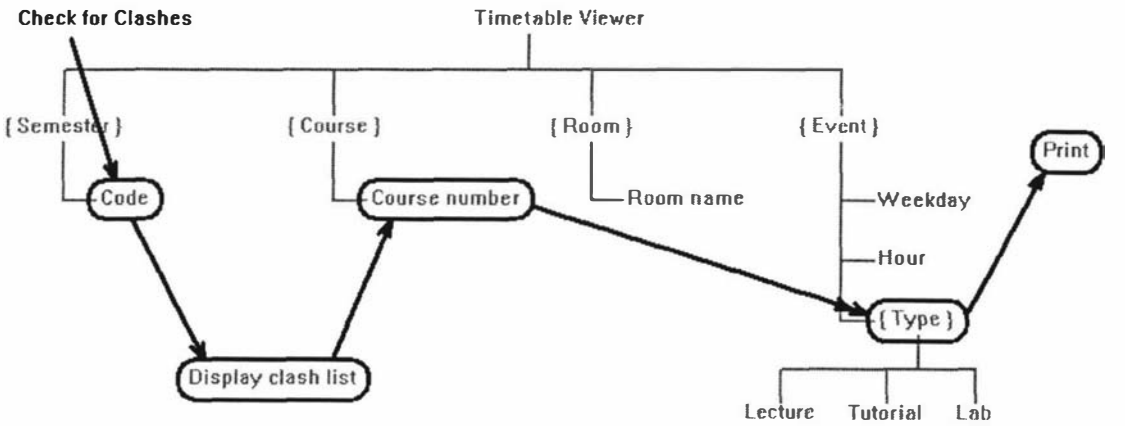


Figure 5.25: First draft of the task sequence for *Check for Clashes*

A new floating meneme, Display clash list has been added to the diagram in Figure 5.25 and the floating meneme Print, from Figure 5.24, is reused. {Type} again has the double-headed arrow leading to it.

In Figure 5.26 the floating menemes from Figure 5.24, Display timetable and Print, are both used again. Display timetable is again used via a system action and not a user action.

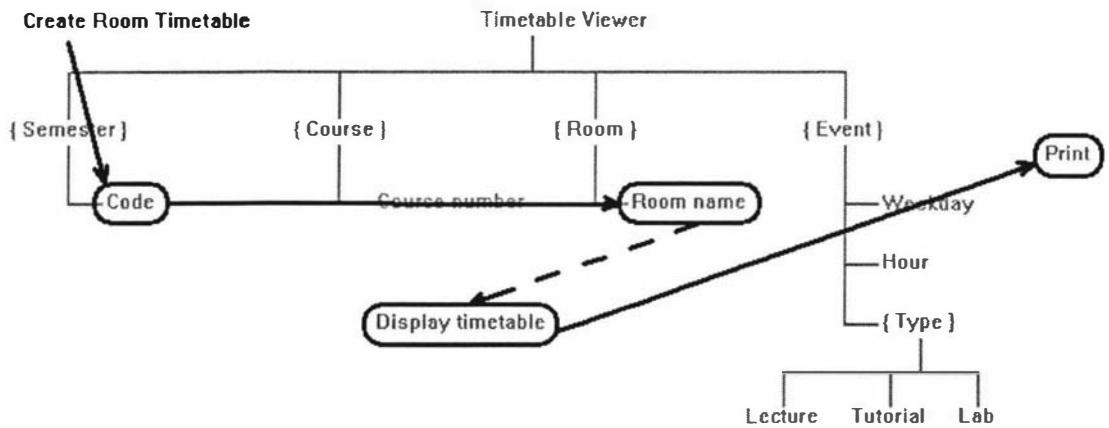


Figure 5.26: First draft of the task sequence for *Create Room Timetable*

Figure 5.27 uses Print once again and adds a new floating meneme, Double bookings.

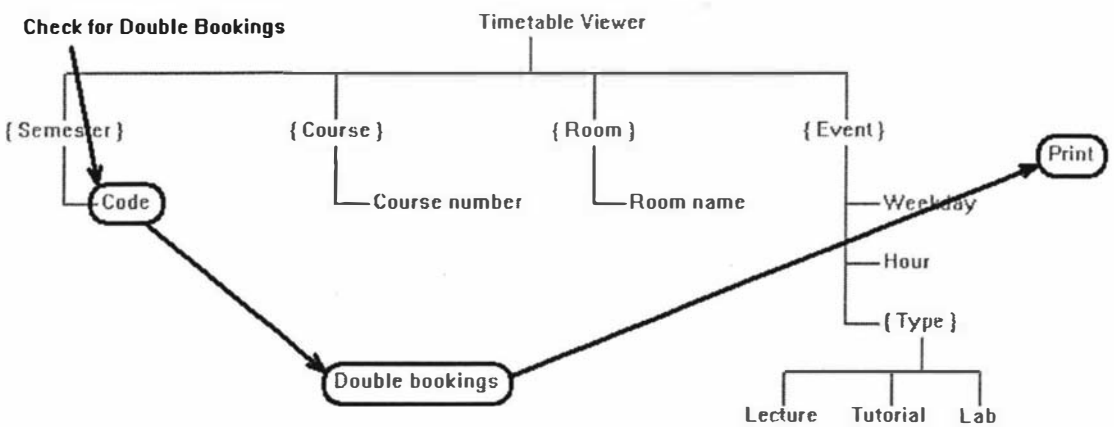


Figure 5.27: First draft of the task sequence for *Check for Double Bookings*

Step 5: Refine the Lean Cuisine+ dialogue tree

The initial task sequences are reviewed. Two menemes in the {Event} subdialogue, Weekday and Hour, are never used and must be removed from the dialogue tree. The floating menemes, identified during the construction of the task sequences, must be added to the tree. These are Display timetable, Display clash list, Double bookings and Print. A new {Display} subdialogue is created to group the first three. The {Event} subdialogue heading is changed to {Controls} as this more accurately describes the role of the menemes in this subdialogue. Room name and Code are represented in the refined tree as mutually exclusive groups using the fork symbol because, in both cases, a number of items appear but only one is selected. Course number appears as a

mutually compatible group as a number of courses are listed and any number of them may be selected. The refined dialogue tree appears in Figure 5.28.

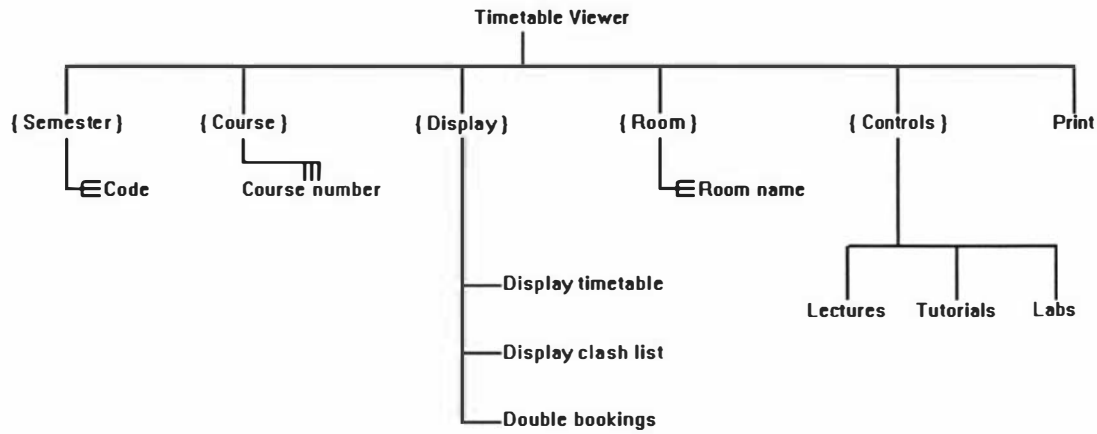


Figure 5.28: The refined tree structure for the Timetable Viewer.

Step 6: Enrich the Lean Cuisine+ diagram

Selection triggers are added to the diagram, corresponding to system actions in the task sequences. Figure 5.29, the trigger from Course number to Display timetable shows that the selection of Course number by the user will trigger the selection of Display timetable by the system, provided the condition is satisfied. The other trigger indicates that the selection of Room name will also trigger the selection of Display timetable by the system.

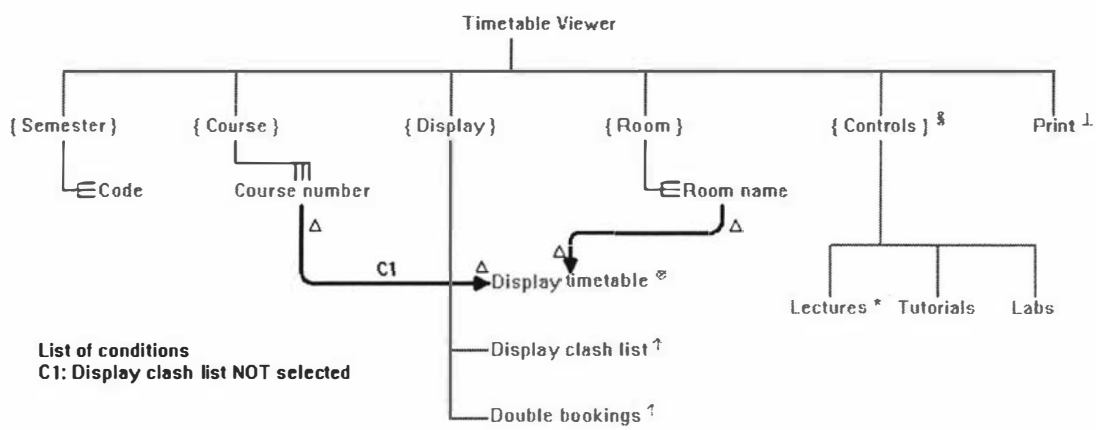


Figure 5.29: Selection triggers and meneme designators for the Timetable Viewer

Meneme designators are also added to the diagram at this stage. A *passive* meneme (⊗) cannot be selected or deselected by the user but can be selected by the system. In Figure 5.9, Display timetable is a passive meneme. A *monostable* meneme (◭) can be selected by the user, but then reverts to an unselected state on completion of the operation it represents. Print is a monostable

meneme as it may be selected but does not require deselection. A *required choice* (§) subdialogue is one from which a valid choice is always required and the *default choice* (*) indicates a meneme that is in the selected state at the commencement of the subdialogue. In Figure 5.9, {Controls} is a required choice subdialogue and the default choice is Lectures.

Step 7: Remap the task sequences

The task sequences constructed in step 4 are modified to fit the refined underlying tree structure.

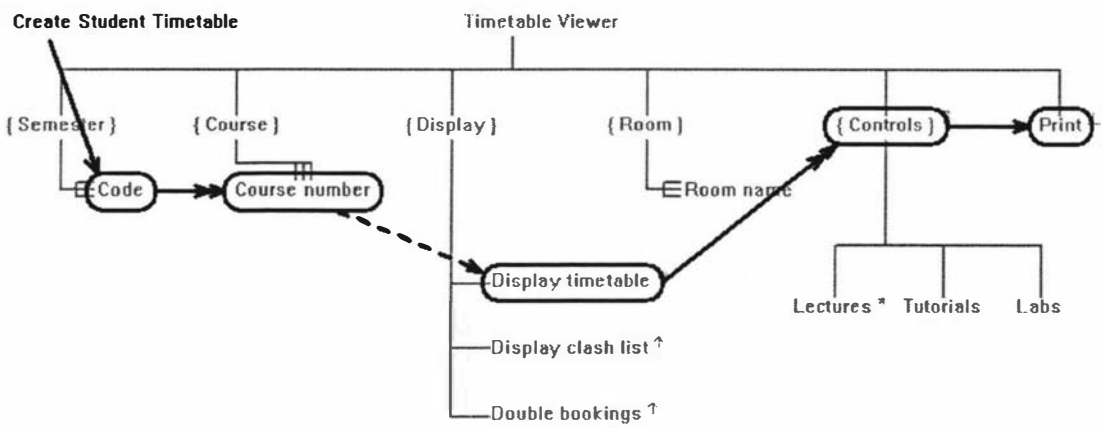


Figure 5.30: Revised task sequence for *Create Student Timetable*

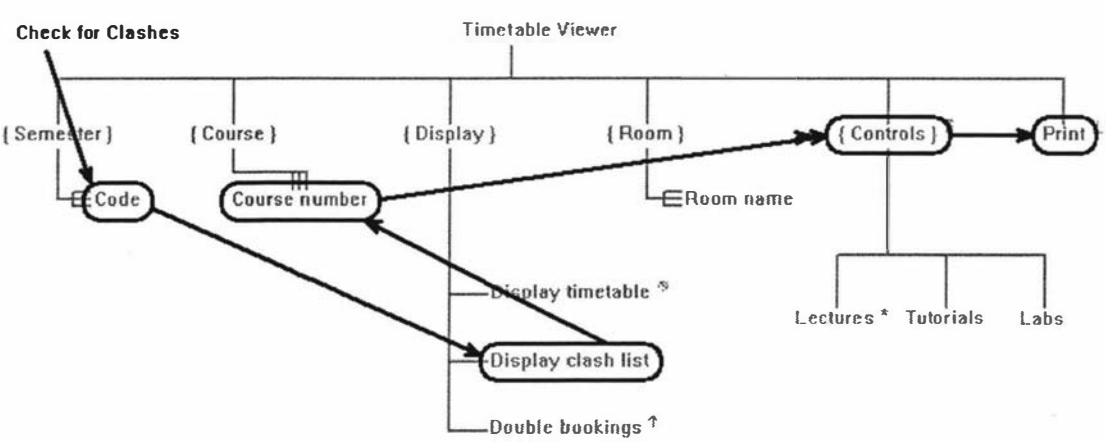


Figure 5.31: Revised task sequence for *Check for Clashes*

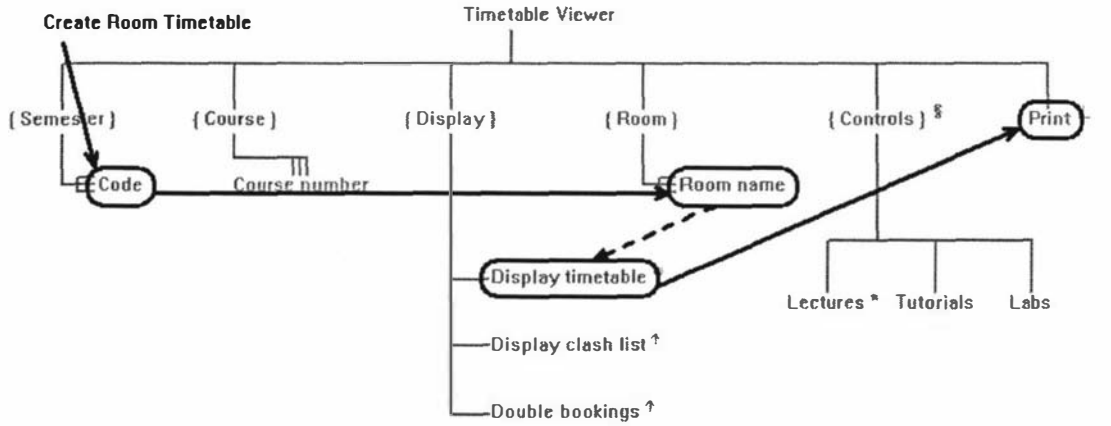


Figure 5.32: Revised task sequence for *Create Room Timetable*

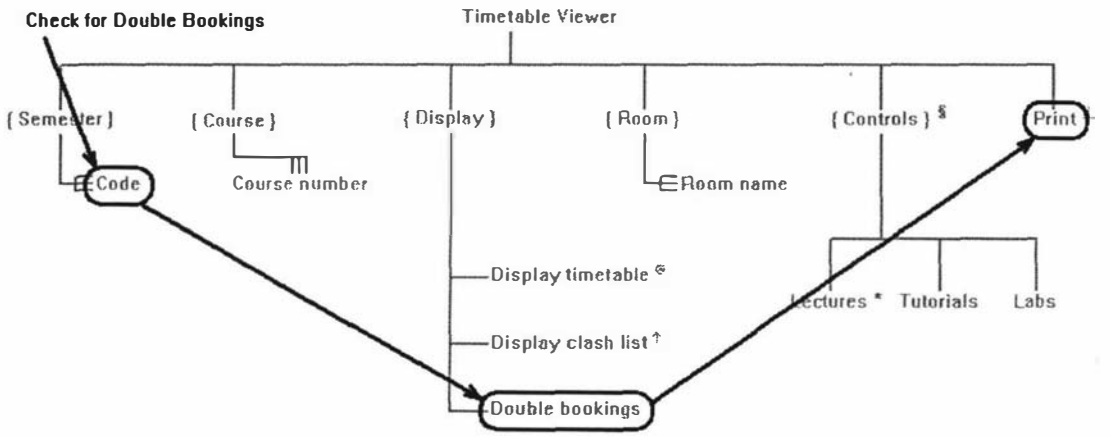


Figure 5.33: Revised task sequence for *Check for Double Bookings*

5.6 Review

This chapter has described a method for the early stages of interactive systems design that incorporates both UML and Lean Cuisine+. This provides a means of representing tasks in the context of the structure of the user interface, i.e. of explicitly showing the transformation of tasks to dialogue. The method involves development of UML use cases and class diagrams, the definition of an initial Lean Cuisine+ dialogue tree, and the subsequent refinement and modification of this structure in order to better support task sequences. This refinement process is repeated for each task sequence, and any conflicts resolved. The final dialogue structure is thus arrived at via an iterative process driven by an analysis of task sequences.

The refined Lean Cuisine+ model captures the behaviour of the interaction at a level above the "look and feel" and has established an overarching dialogue structure. The task sequences can now be analysed independently of the original use cases, in order to find inefficiencies in the execution of the tasks, and this process may lead to further refinements to the model. The next stage would be to translate the model into a visible interface. This requires decisions to be made on the interaction style to be adopted, any metaphors to be employed, and ultimately the spatial grouping of objects but this is outside the scope of this research.

This method is an important step towards integrating user interface design and systems design and development. In addition, the ability of Lean Cuisine+ to represent tasks within the structure of the user interface, presents a unique opportunity to perform efficiency analysis during the design process.

Chapter 6. The SELCU Software Support Environment

6.1 Introduction

This chapter introduces the Software Environment for Lean Cuisine+ and UML (SELCU) and discusses the design issues, implementation and use of the system.

The functional requirements for the system are listed in Section 6.2 and the main design issues appear in Section 6.3. Section 6.4 provides an overview of the SELCU CASE tool. Section 6.5 works through the format of the SELCU data file which has been developed to allow external editing (and automatic generation) of Lean Cuisine+ diagrams. Section 6.6 explains how various revisions were made after initial informal use of the system. Section 6.7 outlines the development history of SELCU. Section 6.8 covers a small case study and Section 6.9 concludes the chapter.

The full SELCU User Manual is provided in Appendix 2 and a limited set of Lean Cuisine+ diagrams representing the SELCU user interface is found in Appendix 3.

6.2 Functional requirements of the system

SELCU is designed as a support environment for the production of Lean Cuisine+ diagrams. To this end, any user of the system needs to be able to:

- *Construct a basic Lean Cuisine+ tree diagram.* It must be possible to create new menemes and arrange them in a hierarchical tree structure with appropriate subdialogues.
- *Edit the basic Lean Cuisine+ tree diagram.* It must be possible to directly select menemes either singly or in groups, and move them, delete them, rename them, and add or remove attributes such as conditions and designators. It must be possible to also directly select links between menemes and move them, delete them or change them to a fork representation.
- *Add triggers to the diagram.* It must be possible to construct a trigger in the diagram from one meneme to another and with specified turning points in between.
- *Edit existing triggers in the diagram.* It must be possible to directly select a trigger and move it, delete it, and add or remove attributes such as conditions and designators.

- *Add task sequences to the diagram.* It must be possible to create a task sequence in the diagram as a series of connected menemes.
- *Edit existing task sequences in the diagram.* It must be possible to directly select a task sequence and delete it, include other menemes in it, exclude menemes from it, and add or remove attributes such as system actions and designators.
- *Reinstate deleted objects.* It must be possible to reinstate accidentally deleted objects via an undo option.
- *Load, save and print diagrams.*

Further, non-functional requirements include:

- making the system as intuitive and as easy to use as possible;
- allowing triggers, tasks and other objects to be displayed in a choice of colours;
- allowing text to be presented in a choice of font and point size;

Ideally the system should also provide an on-line help system, but this has not yet been implemented due to time constraints.

6.3 Design issues

CASE tools are not widely used and many developers find them too restrictive and difficult to learn (Jarzabek & Huang, 1998; Lending & Chervany, 1998). One way of reducing this problem is to keep the user interface of the CASE tool as simple and as intuitive as possible. This guiding principle led to the design issues discussed below.

Direct manipulation of objects

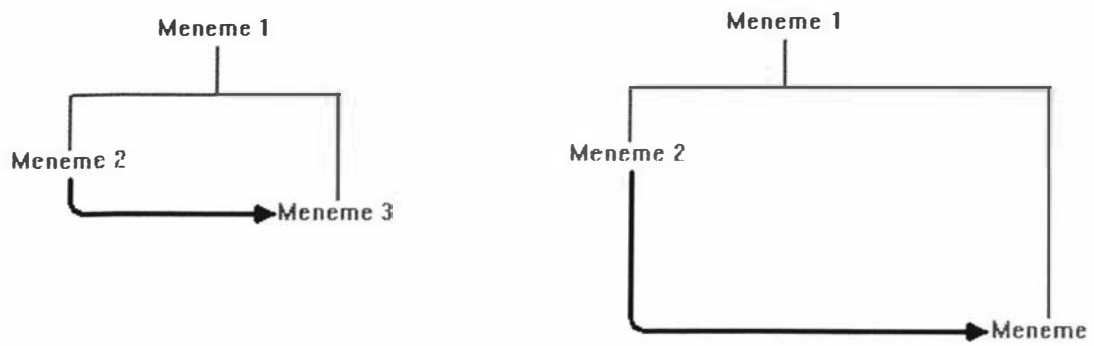
A good way of making a CASE tool easy to use is to provide for the direct manipulation of objects (Shneiderman, 1998). The initial placement of graphical objects in the drawing space needs to be simple and direct (Adams, 1998) and objects need to be able to be selected directly and moved to new positions, if required. A direct manipulation interface should provide high visibility of the objects of interest and incremental actions with immediate feedback (Dix, Finlay, Abowd & Beale, 1998; Beaudouin-Lafon, 2000).

These principles have been adhered to in the design of the SELCU system and have resulted in a system that could initially be used without a user manual because the placement and movement of objects was intuitive.

Initial use of the system has demonstrated that direct manipulation is not ideal in all circumstances. For example, the "ring" forming the action around a meneme that is part of a task sequence can be resized by dragging it (direct manipulation). However, if all actions in a particular task sequence are resized in this manner, it is extremely difficult to drag them all to the same distance from the meneme. For this reason, SELCU provides for the editing of an action and the entering of the distance (in pixels) that the ring should be displayed from the meneme.

Diagram integrity when objects are moved

The Lean Cuisine+ notation comprises several different kinds of objects, viz: menemes, links between menemes, triggers and task sequences. Menemes form the basis of any Lean Cuisine+ diagram and for this reason menemes can be selected and dragged to any point in the drawing area. The positions of the other objects, however, are all dependent on the position of the menemes and so the coordinates of a link, trigger or task are stored relative to the top-left corner of the meneme to which they are connected. This enables SELCU to immediately adjust the position of a link, trigger or task when a meneme is moved. An example is provided in Figure 6.1 where Meneme 3 has been dragged down and over to the right showing how the link (back to Meneme 1) and the trigger from Meneme 2 both extend to retain the connection with the new position of Meneme 3.



(a) (b)

Figure 6.1: Before (a) and after (b) moving Meneme 3

The other Lean Cuisine+ objects can be moved independently of the menemes, but only within a restricted area of the diagram. For example, the trigger that begins at Meneme 2 in Figure 6.1 can be moved to start near the left (or right) edge of Meneme 2 but cannot be moved away from the meneme all together.

Modes of operation

A CASE tool should generally be a *modeless* environment (Mandel, 1997). This means that the user can perform any action at any time without penalty. The "modeless" description is, however, somewhat misleading as it actually means that any mode can be entered at any time but there is often a period when the user prefers to remain within the selected mode. For instance, it is convenient in SELCU to create several new menemes at a time and thus the mode "Insert meneme" is continued until the desired number of new menemes has been created.

A problem with modeless interfaces is that certain actions (e.g. a mouse click) mean different things in different modes. For example in SELCU, a mouse click within an existing meneme can mean "start a trigger from this meneme" but can also mean "include this meneme in the current task sequence". At times like these it is critical to know which mode is in operation.

The most severe disadvantage of having numerous modes of operation is that the system can become locked into a particular mode while the user believes that it is in some other mode. This can lead to a situation where an increasingly frustrated user attempts a series of desperate mouse and keyboard actions that cause considerable mayhem in the diagram under construction. To prevent such a scenario, SELCU provides an option (the Escape key) to exit any mode and return to the "neutral mode" from which any other mode selection can proceed.

Portability of graphical data

It was felt that it might be useful for SELCU to receive graphical data from other applications. In this way, Lean Cuisine+ could be used as a documentation notation for user interfaces developed using other prototyping tools. For this reason, SELCU data files have been organised as simple text files that can be easily generated by other applications. Two examples of the use of this facility are discussed in Chapter 7.

6.4 An overview of the current version of SELCU

SELCU is a CASE tool developed to provide a platform for the efficient production of Lean Cuisine+ tree diagrams and provides for the creation, editing, storage and printing of diagrams. The package does not pretend to be a commercially viable product. It is a prototype with no online help system and is subject to constant revision and upgrading as new Lean Cuisine+ features are evolved. SELCU is a Windows-based package that will run on Windows®2000 or similar. It was developed in Microsoft Visual C++®6.0 using Microsoft Foundation Classes.

SELCU has been designed to support the production of Lean Cuisine+ diagrams and was used to produce all the Lean Cuisine+ diagrams in this thesis. The system is designed to be as intuitive as possible and any part of a diagram can be directly selected. Single menemes can be placed at any point in the work area and new diagrams can be rapidly constructed and expanded by the addition of subdialogues where a mouse-click places each meneme in the subdialogue and links are automatically constructed back to the header meneme. Individual menemes can be edited by selecting the meneme and then changing the name, adding meneme designators, etc.

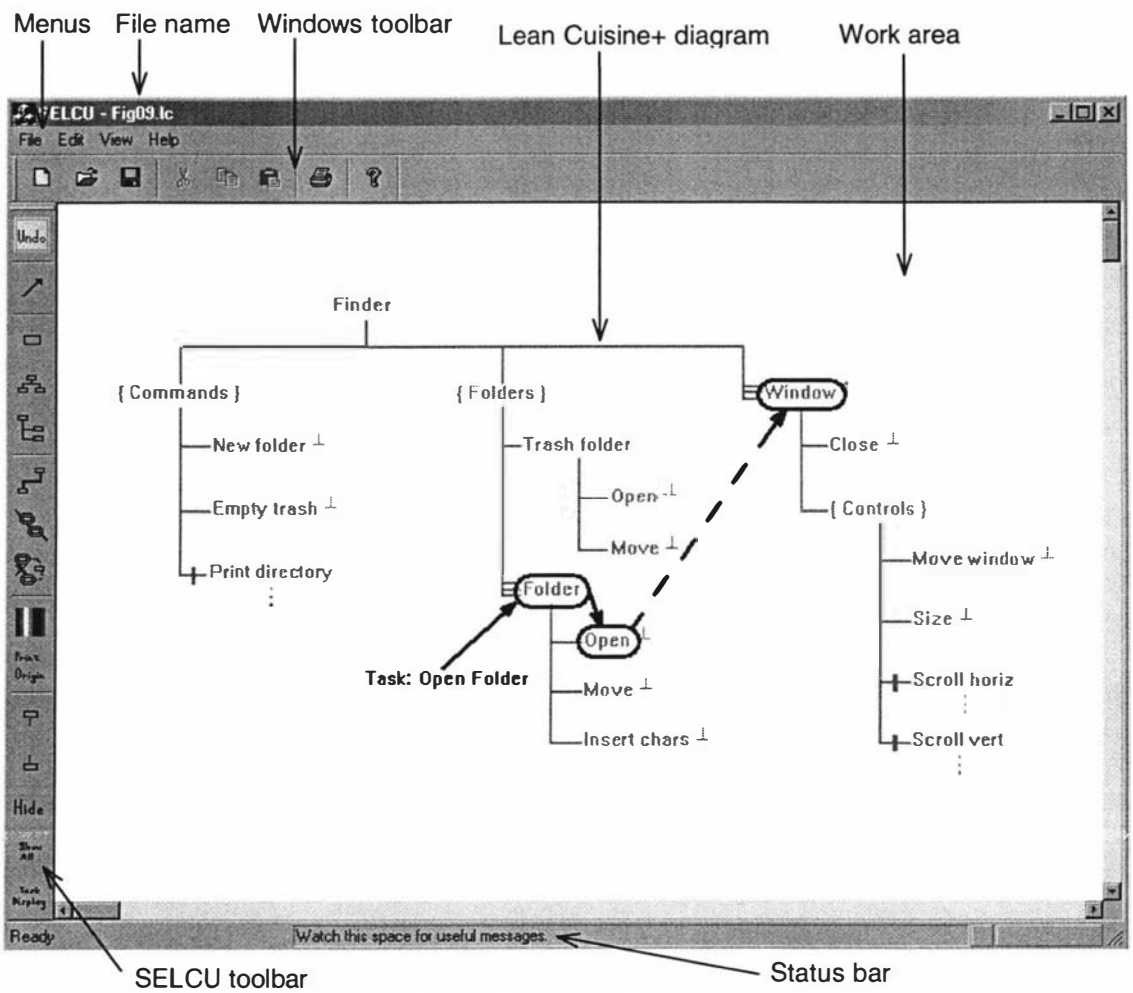


Figure 6.2: SELCU in action

Figure 6.2 illustrates the SELCU interface in use with a typical Lean Cuisine+ diagram under construction in the work area, surrounded by toolbars and status information. At the top of the interface is the title line that displays the file name of the diagram currently on display. This is followed by the menu line, containing four pull-down menus, and then the standard Windows toolbar. To the left of the work area is the SELCU toolbar. Below the work area is the status bar that provides useful messages indicating what action should be taken next.

SELCU provides direct manipulation of diagram objects wherever possible. Thus a meneme can be moved by selecting it with the mouse and "dragging" it to another location – the so-called "drag and drop" method. Most other objects can be moved in a similar manner. Any object can be edited (name changed, etc) by double-clicking on it and then interacting directly with the dialogue box that appears. Selecting an object and then pressing the **Delete** key will delete the object.

Menus and toolbars

Four pull-down menus appear on the top line of the interface: File, Edit, View and Help. The File menu provides the standard Windows options for loading, saving and printing diagrams. Several of these options are repeated on the standard Windows toolbar. The Edit menu provides the standard Windows options such as cut, copy and paste but none of these have been implemented in the current prototype. ***Note: the Undo button provided on the SELCU toolbar is used to "un-delete" Lean Cuisine+ objects and is not related to the standard Windows Undo function.*** The Help menu only provides the single option About SELCU. No on-line help is available in this prototype.

Figure 6.3: The SELCU View menu

The View menu, shown in Figure 6.3, provides the following options:

Menemes only	displays only the basic dialogue tree of linked menemes, excluding any that have been individually hidden
Unhide menemes	displays all menemes including any that have been previously hidden
Conditions	displays all meneme conditions and the list of conditions
Triggers	toggles display of all triggers excluding any that have been hidden
Unhide triggers	displays all triggers including any that have been previously hidden
Tasks	presents a list of tasks from which one (or none) can be selected for display
ALL tasks	displays all task sequences
Toolbar	toggles display of the standard Windows toolbar
Status Bar	toggles display of the standard Windows status bar

Some of these options are mutually exclusive, while others are mutually compatible. These relationships are nicely illustrated in Figure 6.4 which shows the Lean Cuisine+ diagram for the View menu.

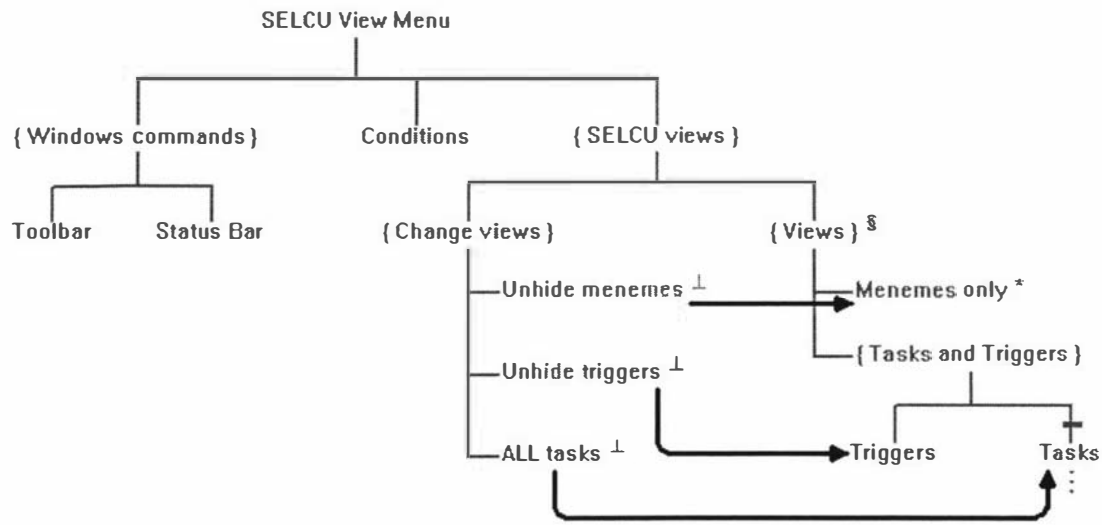


Figure 6.4: Lean Cuisine+ diagram for the SELCU View menu

Toolbar, Status Bar and Conditions are mutually compatible with all other options. Triggers and Tasks are mutually compatible options but are both mutually exclusive to the display of Menemes only. The selection of Tasks produces a modal subdialogue, indicated by the bar across the link to Tasks. Unhide menemes and Unhide triggers are both designated as monostable options as they revert to an unselected state once all previously hidden menemes (or triggers) have been displayed. ALL tasks is similarly designated monostable as it reverts to an unselected state once all tasks have been displayed.

The triggers displayed indicate that the selection of Unhide menemes, Unhide triggers or ALL tasks selects the corresponding display of menemes, triggers or tasks.

To the left of the work area in Figure 6.2 is the SELCU toolbar that provides options relating directly to the construction and editing of Lean Cuisine+ diagrams in the work area. These options include creating a new meneme, constructing a mutually compatible (or mutually exclusive) subdialogue, constructing a new trigger, constructing a new task sequence, modifying an existing task sequence, hide or display selected objects.

Selecting a meneme, trigger or task and pressing the **Delete** key will delete the selected object. The Undo button on the toolbar can be used to Undo the most recent deletion.

Modes

The selection of several of the toolbar options causes the application to change into a different *mode.* It is useful for the user to be aware of the current mode. For example, if *Construct a new meneme* is selected, any click of the mouse inside the work area will cause a new meneme to appear. An indication of the current mode appears on the status bar at the bottom of the screen. Pressing the **Escape** key or selecting the *Neutral mode* option will cancel any and all special effects of the current mode.

6.5 Format of the data file

A Lean Cuisine+ data file is stored as a text file for two reasons:

- 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

A small Lean Cuisine+ example is presented. The Lean Cuisine+ diagram is shown in Figure 6.5 with conditions and triggers visible but task sequences are hidden. The same Lean Cuisine+ diagram is shown in Figure 6.6 with a visible task sequence but conditions and triggers hidden.

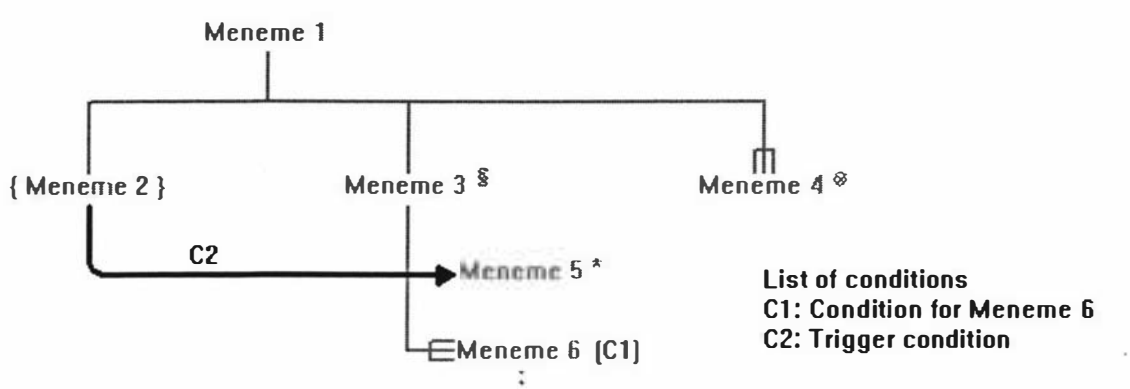


Figure 6.5: A small Lean Cuisine+ example showing conditions and triggers

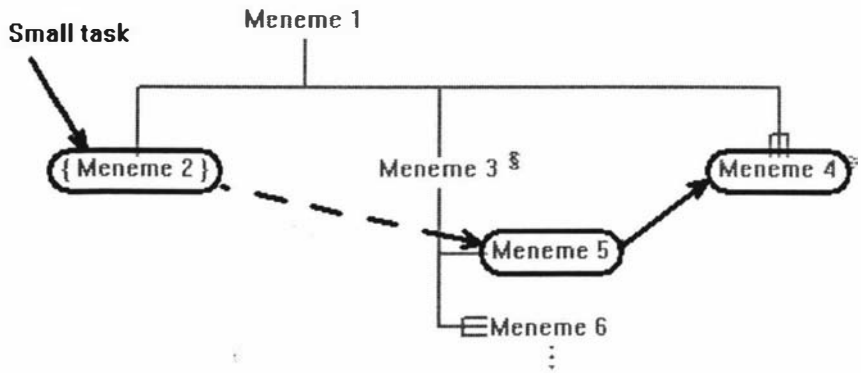


Figure 6.6: The example from Figure 6.5 showing a task sequence.

The same Lean Cuisine+ diagram is then converted to text data, listed below, in order to save it to a text file.

System, 16, 7, 0, 0, 700, 0, 0, 0, 0, 1, 2, 2, 34, 10,
1, [402:-132,494:-154]Vs(0)<>Nf; ; ;CL[0,0:0,0][46,-22:46,-55][-74,-55:378,-55] ;Meneme 1
2,{}[271:-234,385:-256]Vs(1)<>Nf; ; ;nL[57,47:57,0][57,-22:57,-124][0,0:0,0] ;Meneme 2
3, [497:-234,589:-256]Vs(1)<r>Nf; ; ;EL[46,47:46,0][46,-22:46,-118][0,0:0,0] ;Meneme 3
4, [734:-234,826:-256]Vs(1)<p>Cf; ; ;nL[46,47:46,15][0,0:0,0][0,0:0,0] ;Meneme 4
5, [575:-290,667:-312]Vs(3)<e>Nf; ; ;nL[-32,-11:0,-11][0,0:0,0][0,0:0,0] ;Meneme 5
6, [574:-343,666:-365]Vs(3)<>Ef; ;D;EL[-31,-9:-15,-9][46,-22:46,-68][0,0:0,0]*;Meneme 6
2:C1;Condition for Meneme 6
7, [654:-400,746:-422]Dt(6)<>Nf; ; ;nL[-34,-11:0,-11][0,0:0,0][0,0:0,0] ;Meneme 7
A. Triggers ###
1[2,B,57<>][5,L,10<>] (395,-279:416,-299)2:C2;Trigger condition
(57,-68)
B. Tasks ###
1[-141:234,-167:324]B20;Small task
[2(T,25)(R,25)U;4;20;][5(L,8)(R,9)S;4;20;][4(L,21)(B,10)U;4;20;]#

Explanation of the text file format

Delimiters

Delimiters (or separators) are very important in a text file of this nature. Different delimiters have been used in order to make the data file more readable if it needs to be edited. The correct delimiters are shown when each data field is discussed below. For example, [left:top,right:bottom] means "open square bracket" (delimiter) followed by an integer value representing the pixel value of the left of the meneme border followed by colon (delimiter)

followed by an integer value representing the pixel value of the top of the meneme border followed by comma (delimiter) followed by an integer value representing the pixel value of the right of the meneme border followed by colon (delimiter) followed by an integer value representing the pixel value of the bottom of the meneme border followed by "close square bracket" (delimiter).

The first line of the file

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

The first line contains the font information for the file as follows:

Font name, Height, Width, Escapement, Orientation, Weight, Italic, Underline,

Strikeout, Char set, Out precision, Clip precision, Quality, Pitch & family, Font size,

The first line in the above example shows the default "System" font.

The meneme format

2,{}[271:-234,385:-256]Vs(1)<>Nf; ; ;nL[57,47:57,0][57,-22:57,-124][0,0:0,0] ;Meneme 2

AB C D E F G H I J K L M N

Each meneme starts on a new line with format as follows:

A The meneme ID number,

B { } - for a virtual meneme (see Meneme 2) or a space for a meneme which is not virtual

C [left:top,right:bottom] - the pixel coordinates of the meneme border

D Vs - for a Visible meneme – could be Hd for Hidden or Dt for Dots (see Meneme 7). Note that Meneme 7 is not visible in Figures 6.3 and 6.4 as it is in the subdialogue headed by Meneme 6 which is currently hidden with a downwards ellipsis.

E (ID number of header) - zero if there is no header

F <designators> s = Select, d = Deselect, p = Passive, e = Default, m = Monostable, r = Required, l = Modal link, t = Unassigned default for subgroup

G Nf; - No Fork – can also be Cf (compatible fork) or Ef (exclusive fork)

H ; ; - no ellipsis present – U; ; would be "Up" and ;D; would be "Down" (see Meneme 6)

I nL – no Link – can also be CL (Compatible Link) or EL (Exclusive Link)

J [x1,y1:x2,y2] – pixel coordinates of 2 points forming the line "into" the meneme

K [x3,y3:x4,y4] – pixel coordinates of 2 points forming the line "out of" the meneme

L [x5,y5:x6,y6] – 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.*

M ; - A meneme without a condition – can also be *; (see Meneme 6) in which case the next line carries the abbreviation and condition for the meneme – see example below.

N The meneme name

Example of a meneme with two lines in the text file:

6, [574:-343,666:-365]Vs(3)<>Ef; ;D;EL[-31,-9:-15,-9][46,-22:46,-68][0,0:0,0]*;Meneme 6

2:C1;Condition for Meneme 6

A B C

The first line follows the meneme format described above. Note the *; before the name which indicates that a second line is present. The format for the second line is as follows:

A The number of characters in the abbreviation:

B The abbreviation;

C The condition

In the case of a meneme with a condition but no abbreviation, the number of characters in the abbreviation would be zero and the second line of the meneme format would appear as:

0:;A condition with no abbreviation

The trigger format

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. An example of the first line is:

1[2,B,57<>][5,L,10<>] (395,-279:416,-299)2:C2;Trigger condition

A BC D E F G H I J K L M N

The format for the first line is as follows:

A The Trigger ID number

B [ID number of the first meneme

C ,first edge, - can be T (top), L (left), R (right) or B (bottom)

D Distance in pixels from top-left corner of first meneme to the point where the trigger starts

E <trigger designators for the first meneme>] - s = Select, d = Deselect, C = Create, D = Delete

F [ID number of the last meneme

G ,last edge, - can be T (top), L (left), R (right) or B (bottom)

H Distance in pixels from top-left corner of last meneme to the point where the trigger ends

I <trigger designators for the last meneme>] - s = Select, d = Deselect, C = Create, D = Delete

J A space - if + appears after the last meneme, then the trigger is *constrained*

K (left,top:right,bottom) - pixel coordinates of the position of the condition

L The number of characters in the abbreviation:

M The abbreviation;

N The condition

A trigger is a connection from one meneme (the first meneme referred to above) to another (the last meneme referred to above). Graphically, the trigger connects to a specific edge, or side, of each meneme defined as the top, left, right or bottom edge – see **C** and **G** above. The actual connection point is measured in pixels from the top of the meneme (in the case of right or left edge connections) or from the left of the meneme (in the case of top or bottom edge connections). This distance in pixels is stored as **D** and **H** above.

In between the first and last menemes, a trigger may have several *turning points*. These are stored on the second line of the trigger entry in the data file. An example of the second line is: (57,-68). Each point is stored as x and y pixel coordinates ***relative*** to the top-left corner of the ***first*** meneme. It is important that the turning points are stored relative to the menemes so that the trigger can "follow" the menemes if either meneme is moved.

The trigger in the above example has only one turning point – see Figure 6.3. Many triggers would have more than one turning point – for example, a trigger with three points would have a second line entry like this: (71,-110)(368,-110)(368,-351) It is also possible for a trigger to have no turning points – this is a trigger that runs straight from the first meneme to the last. In that case, the trigger would have a second line entry like this:
no points, i.e. a straight trigger

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. An example of the first line is:
1[-141:234,-167:324]B20;Small task
A B C D E

The format for the first line is as follows:

A The task sequence ID number

B [left:top,right:bottom] - the pixel coordinates of the "namebox" containing the task name

C edge - T = first arrow leaves the name "namebox" from the Top edge
L = first arrow leaves the name "namebox" from the Left edge
R = first arrow leaves the name "namebox" from the Right edge
B = first arrow leaves the name "namebox" from the Bottom edge

D Distance in pixels from top-left corner of "namebox" to the point where the arrow starts

E The task name

The sequence of menemes making up the task begins on the second line, as follows:
[2(T,25)(R,25)U;4;20;][5(L,8)(R,9)S;4;20;][4(L,21)(B,10)U;4;20;]#

This example contains three menemes. The information for each meneme is presented using the format [id(edge1,d1)(edge2,d2)type;ring-offset;dash-length;designators] as explained below:

A **B** **C** **D** **E** **F** **G**

A id - The meneme ID number
B (edge1,d1) – edge and distance from top-left of where an arrow ***arrives*** at the meneme
C (edge2,d2) – edge and distance from top-left of where an arrow ***leaves*** the meneme
D type; – the arrow ***into*** the meneme can be a user action (U) or system action (S) or double-headed (D)
E ring-offset; – distance in pixels from the meneme border to the "ring" which is drawn around the meneme
F dash-length; – length in pixels of dashes used to draw a system action (only used if type is S)
G designators – s = Select, d = Deselect, C = Create, D = Delete

There are usually up to four meneme data sets on each line. If the line ends with + then the task sequence continues on the next line. There can be many lines for one task. The task sequence finally ends with #.

6.6 Revisions made after initial use

Staff and students have used SELCU for some time and this has led to various requests for improvements and amendments to the system. Such amendments include the following:

Minor graphical corrections

Minor graphical problems were identified and corrected. These included forks and ellipses not being drawn correctly, links between menemes appearing rather faint and other similar problems.

The Undo option

Users found that Lean Cuisine+ objects – menemes, links, triggers and task sequences – were sometimes deleted by accident and the Undo option was added to the SELCU toolbar in order to

restore deleted objects. One beneficial side effect of this was to eliminate the irritating *confirm boxes* that previously appeared during the deletion of an object as there is no need to confirm a deletion that can immediately be reversed if required.

The Escape key

Users tended to become locked into an unwanted mode or became unsure as to which mode was currently in operation. Pressing the Escape key was incorporated as a quick and consistent method to exit any mode and revert to the neutral state.

The Delete key

Pressing the Delete key was incorporated as a consistent method for deleting any Lean Cuisine+ object – meneme, link, trigger or task. It can also be used to delete a *group* of menemes.

The Ctrl (control) key

Holding down the Ctrl key at the same time as selecting a meneme with a mouse click was incorporated to add (or remove) the selected meneme to (or from) the currently selected group of menemes. This is a standard Windows technique for adding (or removing) an item to (or from) a group of items.

Improving visual feedback

This relates again to the modes of operation. For example, in order to construct a new subdialogue the user has to first select the meneme to become the header of the required subdialogue. Users became uncertain as to whether they had selected the header or not. This was resolved by highlighting the header as soon as it was selected.

Entering meneme names

One of the major tasks in constructing a Lean Cuisine+ diagram is the entry of a name for every meneme. It was therefore decided that this task should minimise the number of selections required. When a meneme is edited in SELCU (by double-clicking on the meneme), the Edit Meneme dialog box will appear and the cursor will be waiting in the name edit box. This means that the new name can be entered immediately without needing any further selections.

Existing menemes or new menemes?

The first version of the system provided an option to construct a subdialogue from existing menemes and another option to construct a subdialogue from new menemes. It was found that

these options could be combined. Thus the current option to build a subdialogue allows the new subdialogue to incorporate existing menemes (by selecting those menemes) and/or to build new menemes within the subdialogue (by selecting a point in a blank part of the drawing area). This applies to both the "construct a mutually compatible subdialogue" and the "construct a mutually exclusive subdialogue".

In Figure 6.7, Meneme 1 has been made the header of a new subdialogue consisting of the existing Meneme 2 and new menemes, Meneme 3 and Meneme 4. The two new menemes were created during the "construct subdialogue" operation by "selecting" a blank point in the work area for each new meneme.

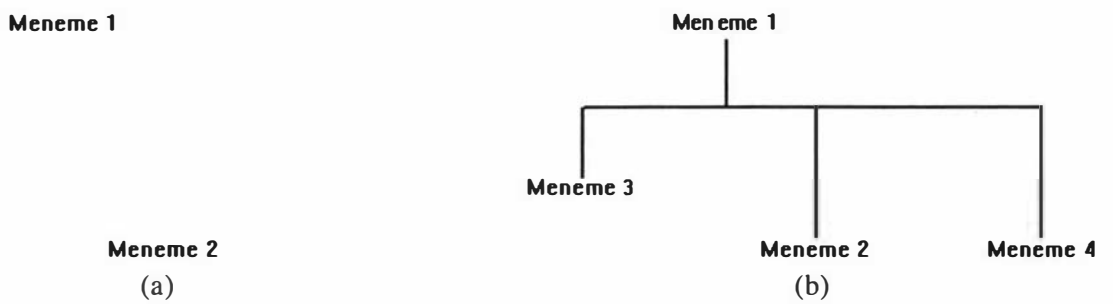


Figure 6.7: Before (a) and after (b) constructing a new subdialogue

Similarly, when constructing a task sequence (during the application of the method described in Chapter 5) it is sometimes useful to include a meneme that does not exist at the time that the sequence is being constructed – a so-called *floating* meneme. Thus the current version of SELCU enables a task sequence to include existing menemes (by selecting the menemes) or to include new floating menemes (by selecting a point in a blank part of the drawing area).

In Figure 6.8, Meneme 4 has been created as a new "floating" meneme during the construction of the task sequence by "selecting" a blank point in the work area.

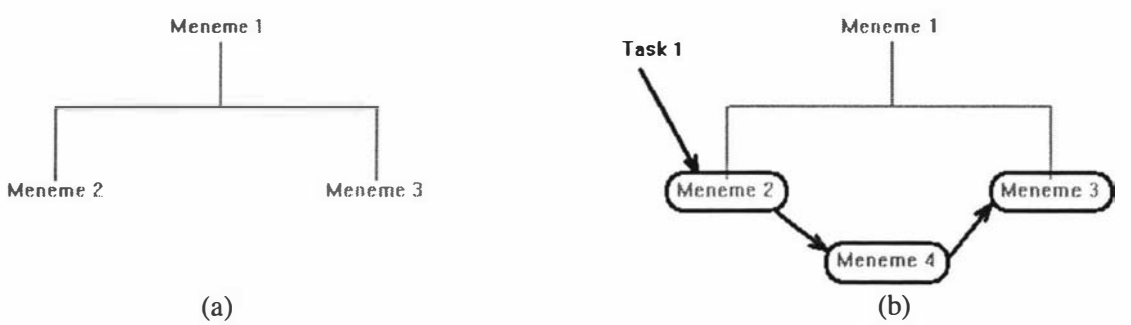


Figure 6.8: Before (a) and after (b) constructing a new task sequence

6.7 The development history of the system

The system was developed over several years and has seen extensive modification and, at one stage, was completely rewritten. The first version was written in Object Pascal on a Macintosh platform. Pascal was not designed as an object-oriented programming language but Object Pascal includes most object-oriented principles (Conner, Niguidula & van Dam, 1995; Rink, Wisenbaker & Vance, 1995).

Extensive use was made of the Macintosh Toolbox Libraries that are also written in Pascal (see references by Apple Computer, Inc, 1992 - 1994). These libraries provide extensive software facilities for managing Macintosh resources, user interfaces, text and graphics. The Macintosh Toolbox Libraries are comprehensive and daunting but there are several useful books on how to get started including case studies and examples (May & Whittle, 1991; Sydow, 1995)

This version of the system was used for the early work on automatic generation of Lean Cuisine+ diagrams from HyperCard and Delphi, described in Chapter 7. Over time, two major drawbacks became apparent with the Macintosh-based system. These were:

- **The Macintosh guidelines for user interface design**
  These so called "guidelines" are actually extremely rigid rules that limit the way in which user interfaces can be constructed for the Macintosh. This provides the advantage that all Macintosh applications have an almost identical user interface but place severe constraints on experimentation with user interface design (see Knaster & Rollin, 1992).

- **The decline of the Macintosh**
  Macintosh use declined appreciably during the late 1990's and this became relevant to the SELCU project in two ways. Firstly, Massey University began to withdraw support for the platform (particularly on the Albany campus) and this made it more difficult to work on a Macintosh platform. Secondly, and more importantly, as outside interest in SELCU grew, other users were frustrated in their attempts to use the system, as they did not have access to a Macintosh platform.

For these reasons, it was decided to recreate the system on a PC platform. At the same time it was decided to change the language from Object Pascal to Visual C++ which is a more powerful object-oriented language and provides a seamless integration with the ubiquitous Windows environment. Visual C++ also enables the programmer to take full control of the

details of interface implementation and it is fully integrated with the Standard Template Library (STL) and the Microsoft Foundation Classes (MFC) – see (Sphar, 1999).

The STL facilitates the consistent use of important data structures, in particular: lists. The STL can only be used successfully at this level by analysing pertinent examples. (Budd, 1998) is very useful, particularly the case study on pp 196-203 and the clever use of iterators in the example on pg 200.

The MFC provides the graphical subroutines for building a Windows-based application on a PC. These routines are similar to those provided on the Macintosh by the Macintosh Toolbox. However, the MFC routines are more extensive, providing far greater programmer control of the implementation of the user interface and also providing consistency within the Windows environment. For example, an MFC-based user interface will behave in the same way when used on a different monitor with a smaller, or larger, screen area and whether the window has been resized or not.

It is essential to have recourse to a number of books on the subject when programming with MFC. Although all the books cover most areas of MFC, it was found that each book was better than the others in at least one particular area, as noted below:

(Blaszczak, 1996) for text entry and editing and using and changing fonts.

(Brain & Lovette, 1999) for dialogue boxes and particularly radio buttons.

(Gregory, 1998) for printing, use of scroll bars, different coordinate systems.

(Horton, 1997) for the technique of recalculating the mouse coordinates after moving the viewport. This was the only book adequately covering this topic and was also generally the best reference on using MFC.

(Prosise, 1996) for dynamic menu elements, changing cursors and graphical "rubber-banding".

(Schildt, 1998) for menus, toolbars, icons and bitmaps.

Rather than translating the Object Pascal version to Visual C++ - which would also have involved the non-trivial task of substituting MFC graphical library routines for their Macintosh Toolbox counterparts – the system was redesigned and then implemented in new Visual C++ code. This process proved invaluable as the experience gained with the use of the early prototype led to a new, more efficient and user-friendly system.

Students are using the SELCU environment in Human-Computer Interaction courses at Massey University and at University College, London and have indicated that they have found the system to be useful and intuitive (Dowell, 2001).

6.8 A case study

Chapter 5 introduced a method for the early stages of user interface design. The discussion of the method used as a case study, the design of a user interface for part of an on-line library catalogue system. This section will illustrate how SELCU was used to construct the Lean Cuisine+ diagrams required for the second part of the case study, viz: the refined tree structure which appears in Figures 5.11 to 5.15.

The process starts with the creation of the first "level" of menemes. Experience with the package has shown that it is useful to complete the first level of meneme names, designators, etc before proceeding to the next level. The first level of menemes is shown in Figure 6.9.

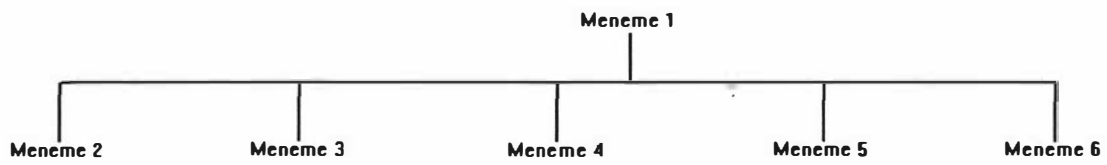


Figure 6.9: The first level of new menemes is created

The menemes are then named and edited as shown in Figure 6.10.

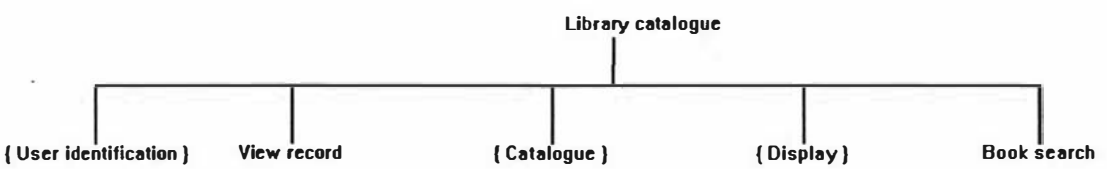


Figure 6.10: Menemes are named and virtual menemes are identified

The subdialogues are then added to the diagram. In each case the subdialogue is created and then the menemes in the subdialogue are named. Figure 6.11 shows the first subdialogue.

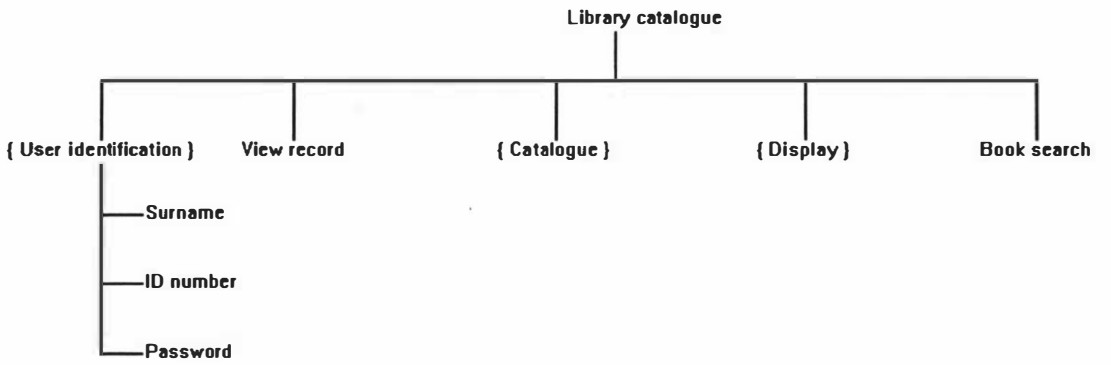


Figure 6.11: The first subdialogue is constructed

All the subdialogues are constructed in turn until the diagram looks like that in Figure 6.12.

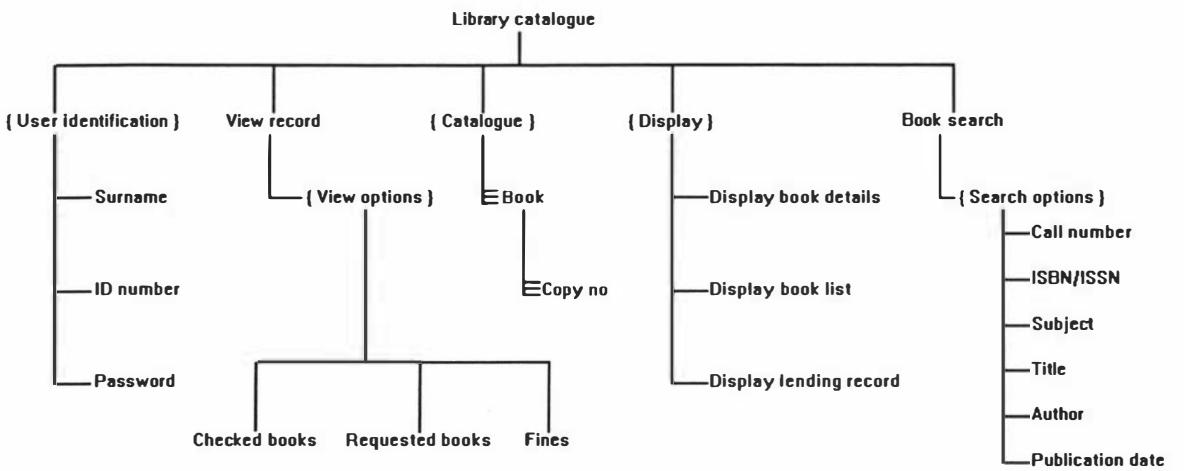


Figure 6.12: The completed tree structure

Triggers, conditions and meneme designators are then added to the basic tree structure resulting in Figure 6.13.

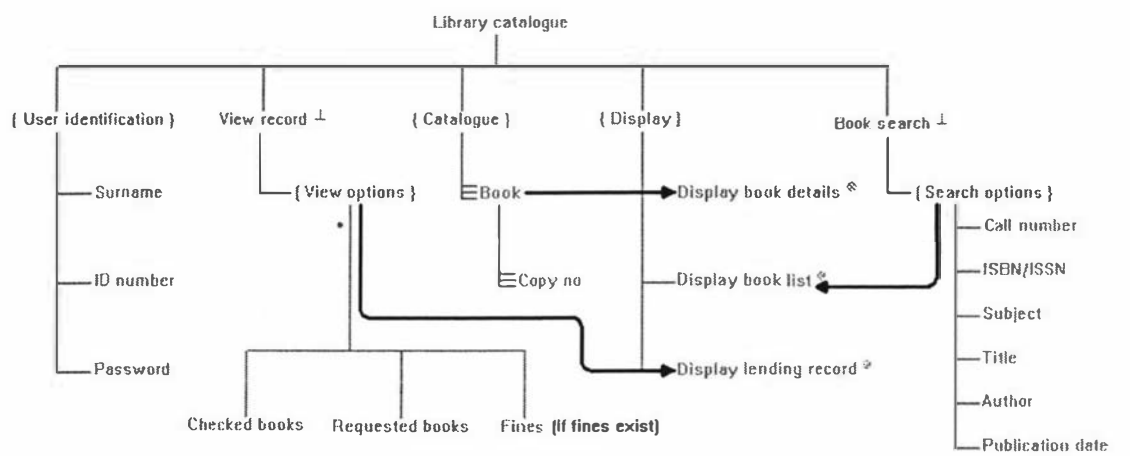


Figure 6.13: Triggers, conditions and meneme designators

Task sequences are then constructed. It is useful to hide triggers when constructing or editing task sequences. Figure 6.14 shows the sequence for the task *View User Lending Record.*

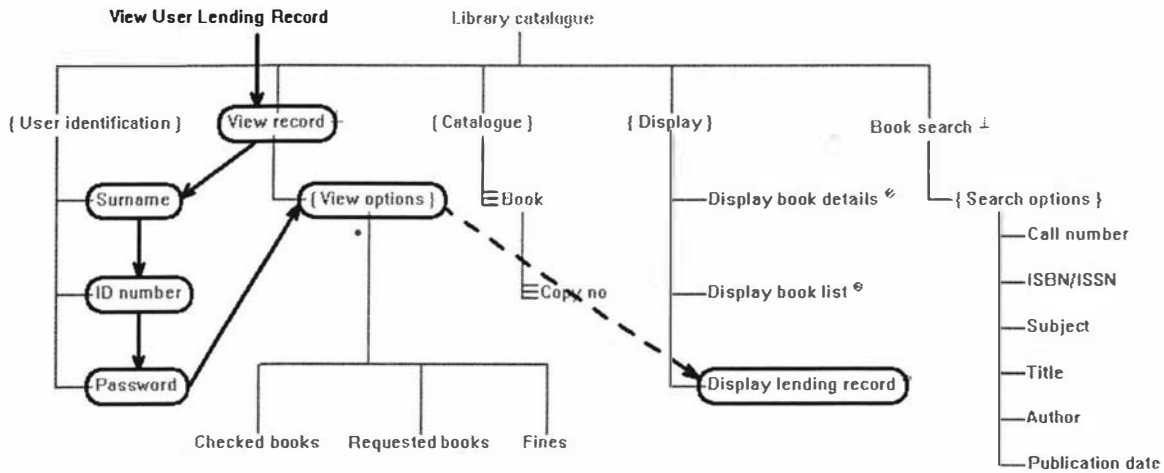


Figure 6.14: Revised task sequence for *View User Lending Record.*

6.9 Conclusions

A software environment has been developed to support the browsing and construction of Lean Cuisine+ diagrams. The data format has been designed to enable the external generation or modification of diagrams. The system has been developed over several years and has been through various modifications. Use in academic environments has led to several revisions and indicates that the system is useful and reasonably intuitive to use.

Chapter 7. The Automatic Generation and Execution of Lean Cuisine+ Specifications

This chapter describes preliminary work by the author on the automatic generation of Lean Cuisine+ specifications and also reports on associated work (Sections 7.1 to 7.3). Section 7.4 outlines work (not part of this research project) on extensions to the SELCU environment to enable the execution of the notation. Section 7.5 provides a review of the chapter.

7.1 Prototyping tools

Prototyping involves constructing a smaller scale version or mock-up of a proposed system. This may be undertaken primarily in order to learn more about the system requirements, but in any case is an information gathering process. Prototyping consumes resources, and the costs must be set against the benefits. A variety of prototyping tools and techniques are available to support the iterative development of interactive systems, from pencil and paper mock-ups to full-scale integrated development environments (Szekely, 1994; Galitz, 1997; Constantine & Lockwood, 1999; Sommerville, 2001).

Prototyping tools provide for the construction of system prototypes which are independent of functionality, and which permit the exploration of alternatives without the overhead of modifying the code associated with functionality during each iteration. Their focus is on how the user interface looks and behaves, which determines what users can see and do at each stage of interaction with the system.

(Galitz, 1997) defines two groups of prototyping tools. Those in the first group, such as HyperCard (HyperCard, 1992), can be used to create a programmed façade. This enables the designer to view certain dialogues and screens as they would appear in the finished user interface. The façade approach retains many of the benefits of pencil and paper, while adding the ability to execute the prototypes and hence provide the illusion of interaction with the application. This is valuable in acquiring knowledge of the system, and in obtaining early feedback from users.

The second group includes integrated development environments (IDEs) such as Delphi (Cantu, 1995) and Visual Basic (Bradley & Millspaugh, 2002). IDEs 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 support

visual programming that provides designers with immediate feedback on the look and feel of the interface.

Although current prototyping tools permit the designer to construct a mock-up of the look and feel of the user interface, they provide no model of the interaction. The mock-up exists and can be exercised, but no specification or description of the interaction exists outside the code produced. The focus is rather on "screens", i.e. on the *interface* rather than the interaction. A model of the interaction would provide additional information, would be useful as documentation, and could be analysed for structural shortcomings.

Prototyping tools could be extended to produce a model of the interaction as a by-product of the construction of the prototype and this chapter describes two such extensions where the interaction model is specified in Lean Cuisine+. Two software applications have been developed to automatically generate Lean Cuisine+ diagrams from existing prototypes. Neither is comprehensive and they are intended to work only with relatively simple interfaces.

The first application loads the script information produced by a HyperCard description of an interface and generates the corresponding Lean Cuisine+ tree diagram, including selection triggers. This work was first described in (Phillips & Scogings, 1997) and later published in (Phillips & Scogings, 1998). The second application reads the data files of a simple Delphi application and automatically generates the corresponding Lean Cuisine+ tree diagram, including selection triggers. This was described in (Scogings & Phillips, 1998).

It should be noted that these applications generated Lean Cuisine+ data compatible with the early version of SELCU, running on a Macintosh, at the time. The applications are not compatible with the latest version of SELCU.

7.2 A HyperCard prototype

Despite its age, HyperCard remains an intuitive and accessible early prototyping tool, and a suitable vehicle for exploring the ideas being proposed here. It should be noted that HyperCard is limited in its ability to simulate more dynamic interactions with a graphics display, and is most suited to simulating event driven behaviour involving basic interaction techniques.

HyperCard is based on two primary abstractions: cards and stacks (of cards). Each card corresponds to one screen, and may contain buttons and text fields in addition to artwork. Backgrounds may be defined to contain features common to several cards. Each HyperCard

object (button, field, stack etc) has an associated HyperTalk script that describes what it must do in response to messages (e.g. mouse clicks). Messages are passed up the HyperCard object hierarchy until they are consumed at the appropriate level. Using HyperCard, the designer may define the appearance of screens, and also simulate the behaviour of menus, buttons etc in linking screens. Thus both the look and feel of the user interface can be created.

A HyperCard prototype has been developed for a real estate application in which potential house buyers can select likely properties from a database of the properties currently for sale in a given town. Users of the application can select properties either by pointing to them on a map of the town displayed on the screen, or by entering selection criteria (suburb, price, number of bedrooms). In the latter case the system responds by allowing the user to step through the database and display details of each matching property in turn.

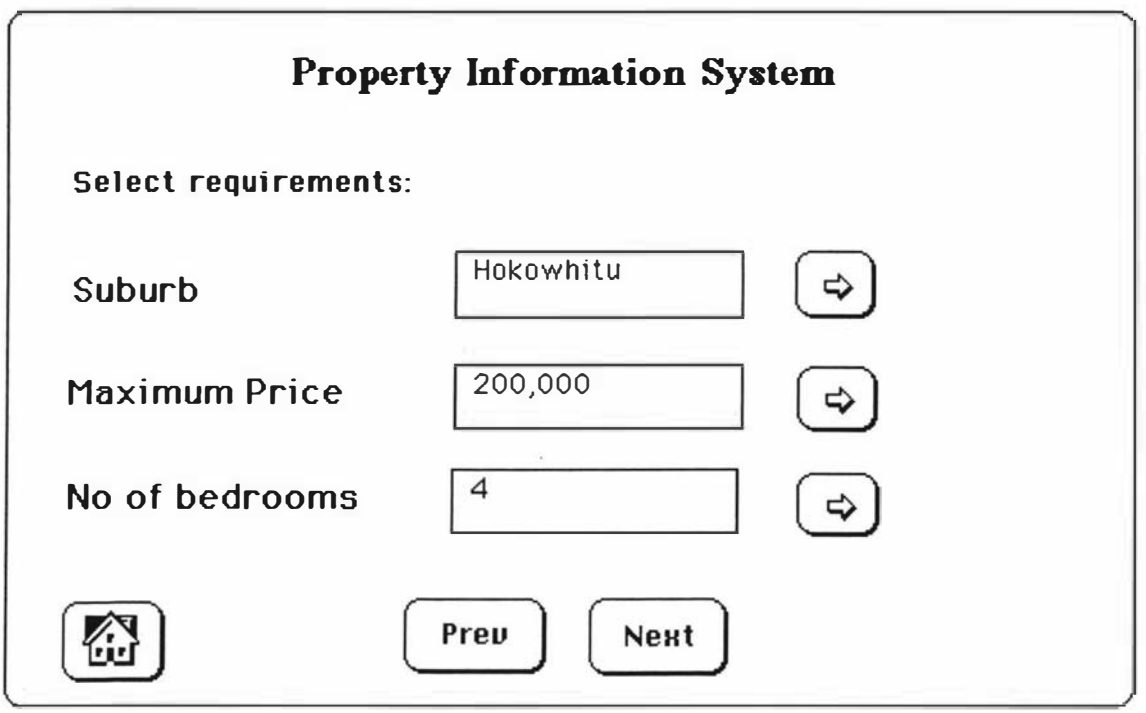


Figure 7.1: The Search Card in the Property Stack

The HyperCard Stack consists of the following cards, each representing a screen in the prototype:

- A Town Map card – this shows a map of the town with the location of properties marked. Properties can be selected by direct manipulation.
- A Search card – this provides fields for the entering of selection criteria, and buttons for initiating the selection of properties. This card is shown in Figure 7.1.
- House cards – each house card displays details of one property, and provides buttons for continuing the search.

The selection and/or data entry options relating to each card are shown in Figure 7.2, together with system responses.

USER ACTION	SYSTEM RESPONSE
Town Map Card	
Select a property on the map	Display the House Card for that property
Select Next	Display the Search Card
Select Home	Display the Home Card
Search Card	
Enter desired suburb	Display the text of the suburb entered
Enter maximum price	Display the price entered
Enter minimum no of bedrooms	Display the number entered
Find suburb	Display first House Card in that suburb
Find price	Display first House Card <= price
Find rooms	Display first House Card >= bedrooms
Select Next	Display the first House Card
Select Previous	Display the Town Map Card
Select Home	Display the Home Card
House Card	
Select Next in this suburb	Display the next House Card in this suburb
Select Next at this price	Display the next House Card <= this price
Select Next this no of bedrooms	Display the next House Card >= no of bedrooms
Select Next	Display the next card
Select Previous	Display the previous card
Select Town Map Card	Display the Town Map card
Select Home	Display the Home card

Figure 7.2: User options and system responses for each card type in the Property Stack

A HyperCard stack is driven by textual information contained in a *script file*. Each card in the stack has a corresponding numbered entry (or *page*) in the script file. The entry for a card contains definitions of the fields and buttons appearing on that card and each button is followed by instructions to control the actions of the HyperCard stack when that button is selected. Part of the script information relating to the Property Stack example appears in Figure 7.3. The script information shown is for the Search Card in the Property Stack.

A software application was developed that parses the text in the script file and produces a corresponding Lean Cuisine+ data file (also a text file). Each card is represented as a meneme in a mutually exclusive dialogue. Each such meneme becomes the header for a subdialogue consisting of menemes representing the fields and buttons listed in the script information for the card.

For example, the Property Stack contains a card called the Search Card (Figure 7.3) and so the final Lean Cuisine+ diagram (Figure 7.4) contains the header meneme Search Card. The menemes in this subdialogue represent the three fields and three buttons listed in the script information in Figure 7.3. The position of selection triggers in the Lean Cuisine+ is derived from the coded instructions appearing under each button, in particular the instruction "go to next card". The next card can be calculated using the card sequence number, which appears on the first line of the script information (see Figure 7.3).

```
Card: 4229;  Number 2 of 6;  "Search Card"
Background: 4073
Script:
On Opencard
  put " " into card field "Q-location"
  put " " into card field "Q-price"
  put " " into card field "Q-bedroom"
End Opencard
3 Card Fields
Field: 1  "Q-location"
Field: 2  "Q-price"
Field: 3  "Q-bedroom"
3 Card Buttons
Button: 1  "First this suburb"
  global holder
  put card field "Q-location" into holder
  go to next card
  repeat until field "P-location" = holder
    go to next card
    if name of card is "Search Card" then
        exit repeat
    end if
  end repeat
Button: 2  "First this price"
{... similar to Button 1 ...}
end repeat
Button: 3  "First this no of rooms"
{... similar to Button 2 ...}
end repeat
```

Figure 7.3: Part of the script information for Search Card in the Property Stack

Once the corresponding Lean Cuisine+ data file has been constructed, the Lean Cuisine+ software environment can be used to display and, if required, edit the diagram. The application of this conversion process to the Property stack script file produces the Lean Cuisine+ diagram shown in Figure 7.4.

The Property Stack is shown as consisting of a Town Map Card, a Search Card and one or more House Cards. These objects are shown as being mutually exclusive (vertical subdialogue) and

thus only one can be selected at a time. The symbol alongside the stack (§) indicates a required choice group, which implies that one card within the stack must always be selected. The initial default (*) is shown as the Town Map Card.

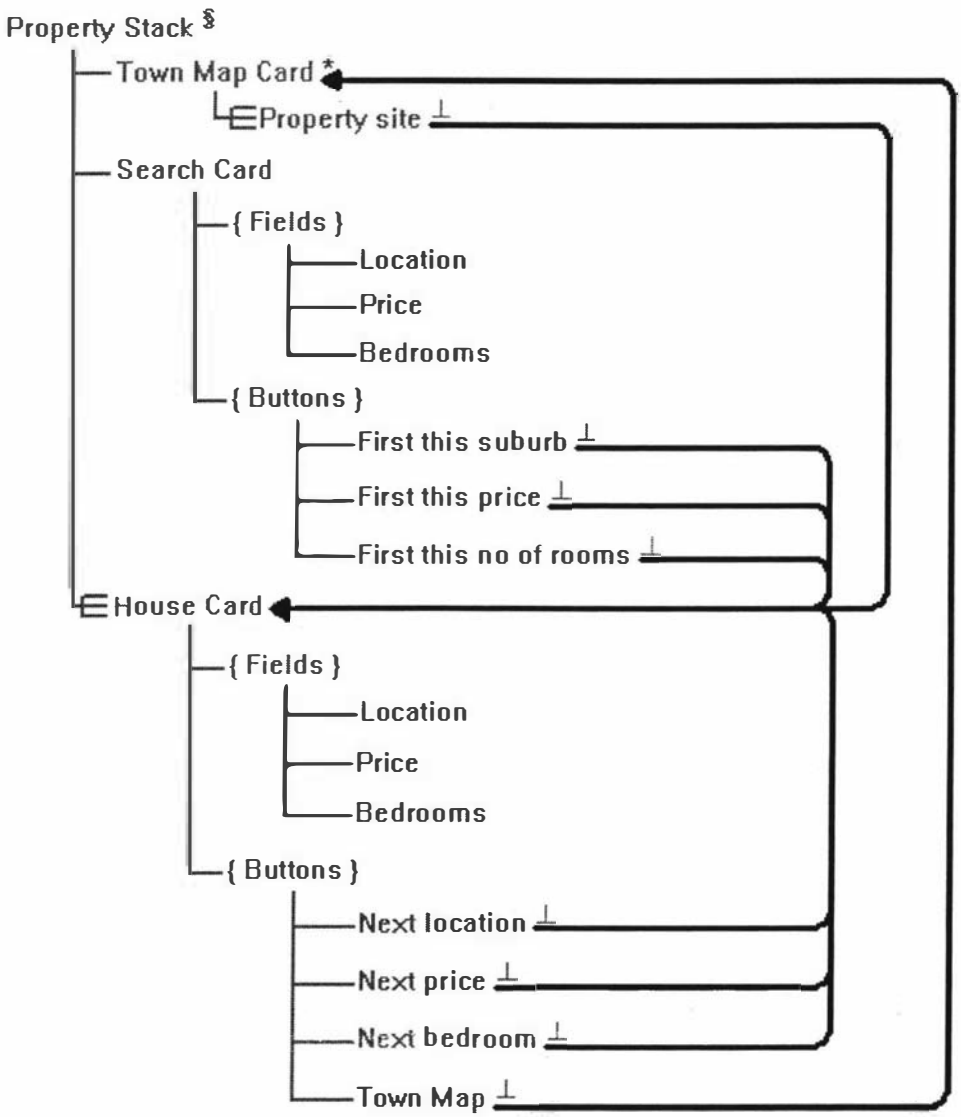


Figure 7.4: Lean Cuisine+ diagram for the Property Stack

The Town Map Card is shown as containing a group of mutually exclusive property site selector buttons that are designated as monostable menemes (⊥), which remain selected only for the duration of the operation and then revert to an unselected state.

The Search Card is shown as containing three mutually exclusive fields into which values can be entered for location (suburb), price and bedrooms, and three mutually exclusive search initiating buttons, which can lead to the selection of House Cards. Virtual menemes, e.g. {Fields}, are used as a grouping device throughout.

Each House Card is shown as consisting of three fields which display values, three mutually exclusive buttons for continuing the search (on either location, price or number of bedrooms), plus a button for directly selecting the Town Map card.

7.3 A Delphi prototype

Delphi (Cantu, 1995) supports a 'drag-and-drop' approach to creating interfaces and is a visual development tool that can be used to create Windows applications. Using Delphi, the designer can define the appearance of screens, and also the behaviour of menus, buttons etc in linking screens. Thus both the look and feel of the user interface can be created.

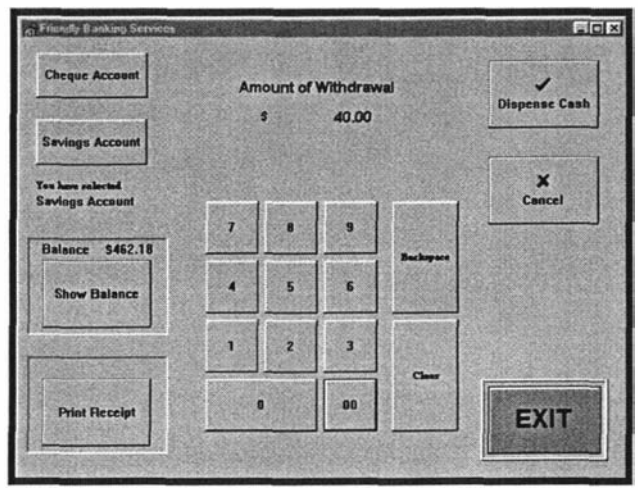


Figure 7.5: The Delphi ATM Withdrawal screen

A Delphi prototype has been developed for an Automatic Teller Machine (ATM) based on a touch screen. Numeric data, e.g.: the PIN number, is entered using touch keys. Six Delphi screens (Forms) have been defined, including the Withdrawal screen shown in Figure 7.5. This screen displays the account and balance, the user entering the withdrawal amount using the numeric, Backspace and Clear keys. The user can then either request that cash be dispensed, or select 'Cancel' (to return to the Main Menu screen) or 'Exit' (to return to the Card Entry screen).

A Delphi application saves data in a Delphi Project File (DPR) and a number of Delphi Unit Files (DFM's). Each DFM contains a *form* (or window) containing buttons, panels and labels. Each button or panel may have associated instructions that are executed when that button is selected.

A software application was developed that could extract data from the DPR and DFM files and use this information to build menemes, subdialogues and selection triggers in a Lean Cuisine+

(textual) data file. This process was applied to the data files of the ATM example and produced the Lean Cuisine+ diagram shown in Figure 7.6.

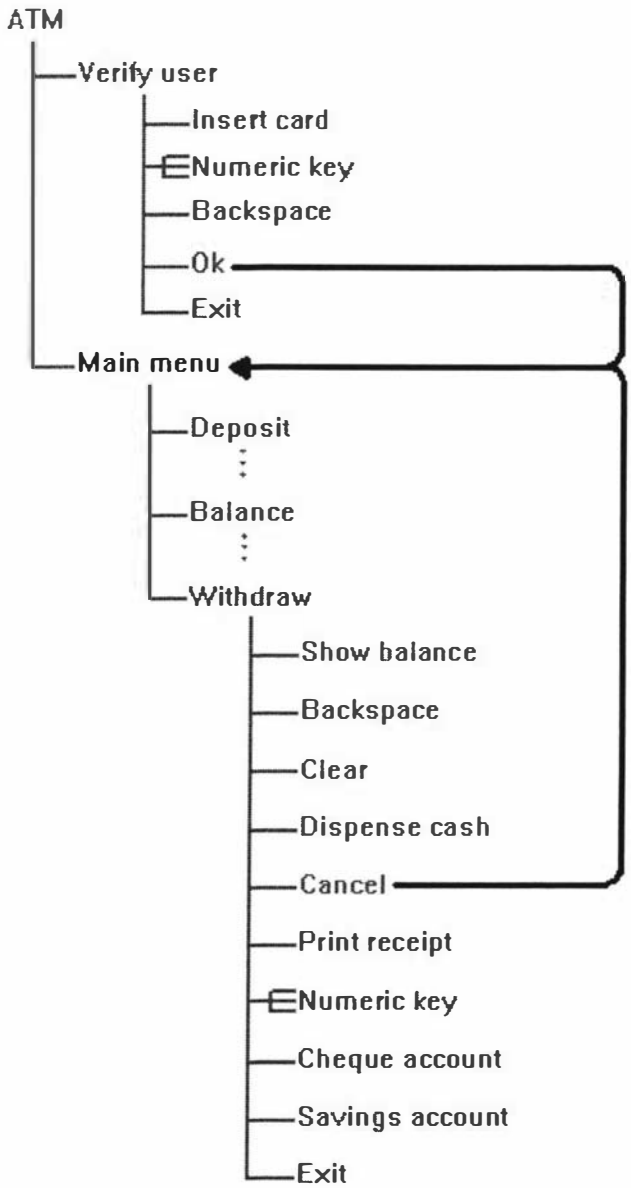


Figure 7.6: Lean Cuisine + diagram for the ATM

The basic Lean Cuisine+ diagram (produced automatically) can be post-edited within the SELCU environment. This can include rearrangement of the diagram to improve clarity (for example changing the order of selectable options), the renaming of options, and the addition of further visual information.

The work described here, including the ATM example, was published as (Scogings & Phillips, 1998). This system made use of the early Apple Macintosh version of the Lean Cuisine+

support environment and could only manage a subset of Delphi forms and instructions.

Later work (Li, 2003) developed a new system to automatically generate Lean Cuisine+ specifications from Delphi user interfaces. This system feeds into the latest version of the software support environment (SELCU) and also accommodates a greater range of Delphi forms and instructions. The general approach used in the new system is unchanged from that described above.

7.4 Execution of Lean Cuisine+ specifications

As part of an associated Masters research project (Li, 2003) SELCU has been extended to enable the execution of Lean Cuisine+ specifications. During execution, the state of each meneme in the Lean Cuisine+ diagram is shown by using colours and line styles: a selected meneme is red, a non-selected meneme is black, an available meneme has a solid-line link leading to it and a non-available meneme has a dashed link leading to it.

The designer can then select menemes and SELCU will reflect the state of the user interface following that selection. For example, selection of a meneme in a mutually exclusive subdialogue will mean that all other menemes in the subdialogue become non-selected and non-available. It is also possible to preview a meneme selection. If a meneme is selected for preview, the system indicates which menemes will change state if the previewed meneme is selected during execution.

A task sequence can be selected for execution. When a task is displayed in execution mode, two buttons, *next* and *previous*, are provided on the system toolbar. Each time the *next* button is selected, the next action in the task sequence is executed and the current state of all menemes is displayed. An example is shown in Figure 7.7 which originally appeared in (Li, 2003) as Figure 6.24.

This figure shows the state of the dialogue after the *next* button has been selected twice. The task would have been initially displayed with the initial meneme, Library Catalogue System selected (and displayed in red). Then *next* would have been selected and the second action would have been displayed, showing the menemes {Select Group} and Select in red. Then next would have been selected for the second time, leading to the state displayed in Figure 7.7 where EditBox has been selected and all four selected menemes are shown in red.

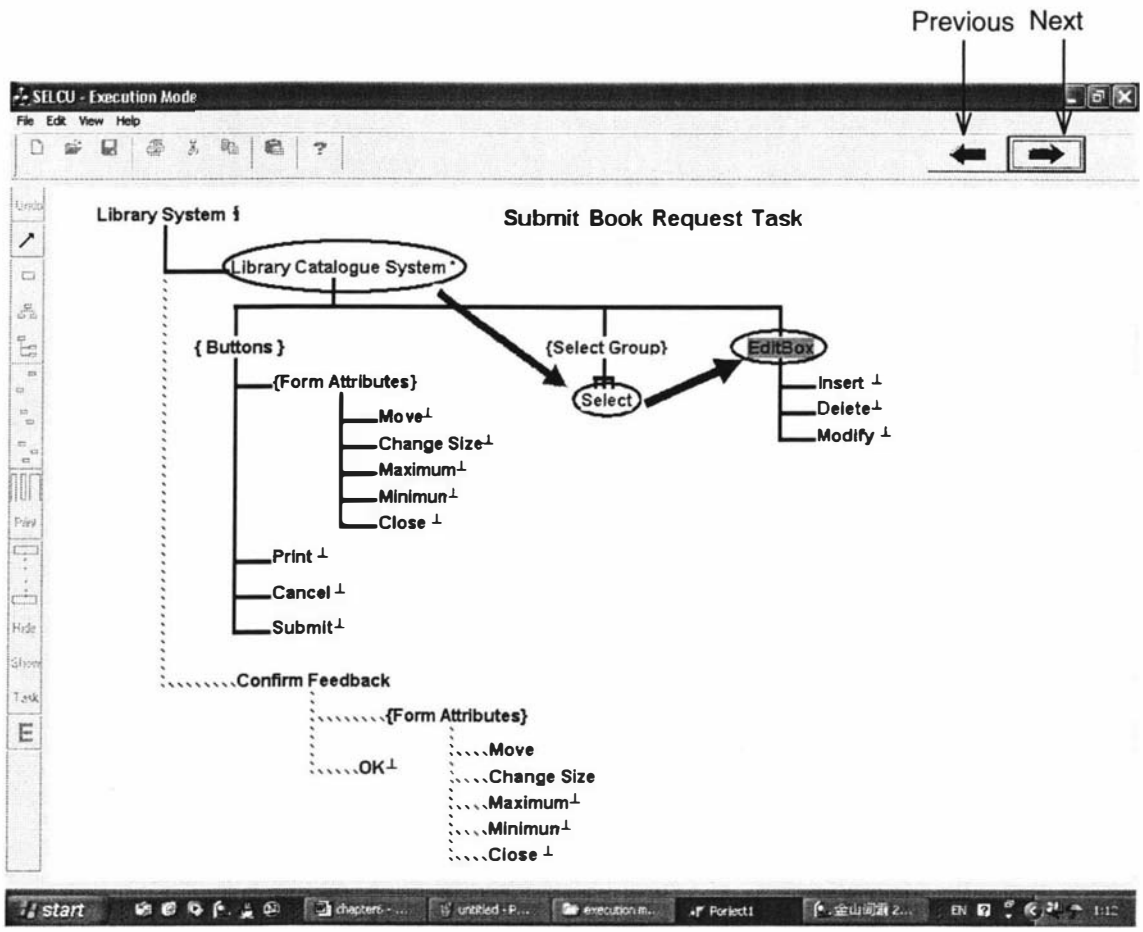


Figure 7.7: Executing Lean Cuisine+ specifications

The meneme Confirm Feedback (and its subdialogue) is not available for selection at this stage so the link leading to it is shown as a dotted line. The system will display an error message if an illegal action is performed - such as selecting a meneme not currently available for selection.

Execution of the model opens a range of possibilities not previously available. It provides the designer with a "working model" of the user interface and can be used to determine the completeness of the interface and the efficiency of task sequences. It enables a user interface under design to be rapidly checked for design flaws. For example, a task sequence may indicate that a particular meneme should be selected when that meneme is not available.

7.5 Review

The research covered in this chapter relates to the wider use of the Lean Cuisine+ notation, in two specific areas.

First, Lean Cuisine+ specifications have been automatically generated from existing user interface prototypes developed with industry interface builders – HyperCard and Delphi. Each model provides information on the existing user interface that could otherwise be uncovered only by using the prototype and committing the behaviour to memory, or by studying the scripts attached to various objects. In particular, concise information in graphical form is presented on the structure of the interaction of objects within, and across, screens. The model could also have task sequences included and then be analysed for structural shortcomings and inconsistency within the dialogue. This could include excessive navigation between screens in performing tasks, inconsistencies in selecting options on different screens, and missing options.

Secondly, through associated work, the Lean Cuisine+ software support environment (SELCU) has been extended to enable the execution of Lean Cuisine+ specifications. This provides the designer with the opportunity of testing the behaviour of a proposed user interface before the interface is constructed – a working model. The model could then be used to test the interface under design for efficiency and completeness by executing each of the task sequences.

Each of the topics covered above is the starting point for further work in the area and the exploration of what is possible with a semi-formal graphical notation, supported by a sophisticated software environment.

Chapter 8. Conclusions and Further Work

In this chapter the research reported in this thesis is reviewed and further research is identified. The research seeks to build bridges (a) across the gap between task and dialogue modelling within user interface design and (b) across the gulf between user interface design and general software engineering notation and methodology.

8.1 The Lean Cuisine+ notation

Lean Cuisine+ was developed as a semi-formal graphical notation for describing the underlying behaviour of direct manipulation user interfaces. The notation has the following advantages:

- it has been specifically designed as a user interface design tool – as opposed to other notations, such as UML, which have been developed for general software design;
- the task models (or *task sequences*) can be displayed within the context of the dialogue structure.

Part of the research reported in this thesis (Chapter 3) has been to analyse Lean Cuisine+ and modify the notation in order to simplify it and to make it more suitable for use with a supporting software tool. In particular, it has become a dynamic notation with certain aspects able to be displayed or hidden at will during the user interface design process. The object model has been removed from the notation resulting in a number of improvements in the clarity of diagrams. For example, boxes around the menemes of objects are no longer required and this makes the diagram easier to read. It is possible to represent the information previously contained within the object model by using UML class diagrams.

Lean Cuisine+ was originally developed as a paper-based notation that consisted of a base diagram and four overlays: the existence dependency layer, the option precondition layer, the selection trigger layer and the task layer. The existence dependency layer reflects relationships between objects and has thus been removed along with the object model. Existence dependencies can be represented within the appropriate UML class diagram. The option precondition layer has been simplified and is now represented by meneme conditions.

Selection triggers and the task layer have been retained. The task layer is an extremely important part of Lean Cuisine+ as it forms the link between the dialogue structure and task representation. The task sequence notation has been expanded to provide for the selection of a

virtual meneme (to indicate the selection of any allowable combination of the menemes belonging to its subdialogue) and to provide a means of indicating the selection of any number of menemes at one step in the task sequence – often used in conjunction with an *homogenous group* of menemes. These enhancements have enabled the representation of more complex task sequences.

8.2 The method for the early stages of user interface design

Support for user interface design has been identified as sadly lacking in software design notations and methodologies. User interface design has either been omitted entirely or has made use of methods and notations originally developed for mainstream software engineering. Such tools have been found to be generally unsuitable for the specialised task of user interface design. As part of this research a method has been developed which is to be applied at the early stages of user interface design (Chapter 5). It has been presented with reference to a number of case studies. The method has the following two advantages:

- It uses Lean Cuisine+ to link task and dialogue modelling

Task and dialogue modelling have long been part of user interface design but have been conducted in isolation with differing notations. Calls have been made to bring task and dialogue modelling closer together (Brooks, 1991; Bomsdorf & Szwillus, 1999). The research reported in this thesis has reviewed task and dialogue modelling (Chapter 2) and described how the divide between the two can be bridged via the Lean Cuisine+ notation, which provides for the representation of tasks within the context of the dialogue.

- It fits in well with UML and the Unified Process

The otherwise comprehensive UML notation provides almost no support for user interface design. Solutions have been proposed in which UML diagrams are used to model tasks and dialogue but these perpetuate the problem of using general tools for a specialised task. The method developed in this research has been placed within the framework of UML and the Unified Process. One of the most important aspects of the Unified Process – ensuring that the process is "use case driven" – has been retained. The same use cases are used to drive both the software design according to the Unified Process (outside this research) as well as the development of the Lean Cuisine+ dialogue and task model. The development of classes is acknowledged as an important part of the Unified Process and the class diagrams are also used to represent the object model removed from earlier versions of Lean Cuisine+ (see Section 8.1). This means that the proposed user interface design method can be easily incorporated into the wider software design process.

8.3 The SELCU environment

The thesis describes the design and implementation of the supporting software environment for Lean Cuisine+ (SELCU) (Chapter 6). This environment assists user interface designers in the production of Lean Cuisine+ diagrams. Diagrams can be constructed, edited, printed and saved in data files.

Cognisance has been taken of the fact that designers often find similar tools restrictive and difficult to learn, and SELCU has been designed to be as simple and intuitive as possible. The system has a direct manipulation interface and any component of a Lean Cuisine+ diagram can be directly selected and moved, changed or deleted.

The system was originally constructed on an Apple Macintosh platform and this initial version was used for the automatic generation of Lean Cuisine+ diagrams – see Section 8.4. This version was later abandoned and a new version of SELCU was developed using Visual C++ and a PC-based Windows environment. This process proved invaluable as the experience gained with the use of the early prototype led to a new, more efficient and user-friendly system.

8.4 Automatic generation and execution of Lean Cuisine+ specifications

The thesis reports on preliminary work in the area of automatic generation of Lean Cuisine+ diagrams from existing user interfaces (Chapter 7). Existing user interfaces need to be documented prior to the production of reference material and also with a view to possible modification and updating. Lean Cuisine+ diagrams have been automatically generated for two existing user interfaces, both constructed with standard interface builder applications. As well as the documentation aspects, the construction of the Lean Cuisine+ diagrams also enables the user interfaces to be analysed for efficiency and completeness.

It should be noted that this initial work has spawned further independent research in the form of a masterate (Li, 2003) that includes both an updated process for the automatic generation of Lean Cuisine+ specifications as well as new work on the execution of the specifications.

8.5 The wider contribution of this thesis

The major contribution of the research reported in this thesis is that it has:

- established Lean Cuisine+ as a dialogue modelling notation capable of representing tasks within the dialogue structure;
- developed a software environment to assist in the construction of Lean Cuisine+ diagrams;
- proposed a method for the use of Lean Cuisine+ in the early stages of user interface design.

However, user interface design is only one aspect of the design of a complete software application. At the wider level, what should be one seamless process (i.e. the construction of new software applications) has been fragmented into two almost separate academic disciplines and areas of research. The two areas have become so polarised that it is rare to find a contribution within one field that acknowledges the other. From a practical point of view, the standard software engineering notations and techniques – culminating in UML – ignore user interface design almost completely.

This research draws attention to the fact that a significant gulf exists between user interface design and software engineering and provides one step along the road to bringing the two areas closer together.

8.6 Further work

A number of areas for further related research have been identified:

- The establishment of more precise relationships between use cases, scenarios and Lean Cuisine+ task sequences and an analysis of the level at which task sequences are most useful. For example, should a task sequence represent a use case or one scenario within a use case? Should a task sequence include alternative paths or optional actions?

- The establishment of a method to move from Lean Cuisine+ to screen design, where elements must be grouped spatially. For example, would it be useful to include "screen layout" designators within the Lean Cuisine+ framework?

- The extension of the software environment to at least partially automate the production of initial Lean Cuisine+ diagrams and task sequences from UML use cases.

- The exploration of the use of Lean Cuisine+ as an analytical tool to check that the system supports all tasks uncovered at the requirements stage and that they can be carried out in an efficient manner (in regard to key strokes and selections). It may also be possible to partially automate this.

- Further work in the area of the execution of Lean Cuisine+ specifications – in particular the execution of task sequences. This would include an exploration of the feedback that the execution of specifications could provide to the designer and may lead to the development of guidelines for the use of such a "working model" of the user interface.

References

Adams S.K. (1998) *Development of a Client Interface for a Methodology Independent Object-Oriented CASE Tool.* MSc thesis. Massey University. New Zealand.

Alhir S.S. (1998) *UML in a nutshell.* O'Reilly & Associates Inc. California.

Anderson P. (1995) *Dialogue Activation: An approach to user-centred constructional modelling of direct manipulation interfaces.* PhD thesis. Massey University. New Zealand.

Apperley M.D. & Spence R. (1989) Lean Cuisine: A Low-Fat Notation for Menus. *Interacting with Computers* 1 (1). 43-68.

Apple Computer, Inc (1992) *Inside Macintosh: Overview.* Addison-Wesley. Reading, Mass.

Apple Computer, Inc (1992) *Inside Macintosh: Macintosh toolbox essentials.* Addison-Wesley. Reading, Mass.

Apple Computer, Inc (1992) *Inside Macintosh: Files.* Addison-Wesley. Reading, Mass.

Apple Computer, Inc (1994) *Inside Macintosh: Printing extensions and drivers.* Addison-Wesley. Reading, Mass.

Apple Computer, Inc (1994) *Inside Macintosh: Imaging with QuickDraw.* Addison-Wesley. Reading, Mass.

Artim J., van Harmelen M., Butler K., Gulliksen J., Henderson A., Kovacevic S., Lu S., Overmeyer S., Reaux R., Roberts D., Tarby J-C. & Vander Linden K. (1998) Incorporating work, process and task analysis into Commercial and Industrial object-oriented systems development. *SIGCHI Bulletin* 30 (4).

Beaudouin-Lafon M. (2000) Instrumental Interaction: An interaction model for designing post-WIMP user interfaces. *Proceedings of CHI'2000.* The Hague, Netherlands. April 2000. 446-453.

Beck K. (2000) *Extreme Programming Explained.* Addison-Wesley. Reading, Mass.

Becks A. & Koller J. (1999) Automatically structuring Textual requirement scenarios. *Proceedings of ASE'99.* IEEE. Los Alamitos, California. 271-274.

Bergner K., Rausch A. & Sihling M. (1998) A critical look upon UML 1.0. In Schader M. & Korthaus A. (Eds) *The Unified Modelling Language*. Physica-Verlag. Germany. 79-92.

Biddle R., Noble J. & Tempero E. (2000) Patterns for Essential Use Cases. *Technical Report CS-TR-01/02 April 2000*. School of Mathematical & Computing Sciences. Victoria University. Wellington, New Zealand.

Biddle R., Noble J. & Tempero E. (2001) Essential Use Cases and Responsibility in Object-Oriented Development. *Technical Report CS-TR-01/01 March 2001*. School of Mathematical & Computing Sciences. Victoria University. Wellington, New Zealand.

Blaszczak M. (1996) *The Revolutionary Guide to MFC 4 Programming with Visual C++*. Wrox Press Ltd. Canada.

Bomsdorf B. & Szwillus G. (1999) Tool Support for Task-based User Interface Design. *SIGCHI Bulletin* 31 (4). 27-29.

Booch G. (1999) UML in Action. *Communications of the ACM* 42 (10). 26-28.

Booch G., Rumbaugh J. & Jacobson I. (1999) *The Unified Modelling Language User Guide*. Addison-Wesley. Reading, Mass.

Bradley J.C. & Millspaugh A.C. (2002) *Programming in Visual Basic 6.0*. Update Edition. McGraw-Hill. New York.

Brain M. & Lovette L. (1999) *Developing Professional Applications for Windows® 98 and NT® using MFC* (3rd Edition). Prentice-Hall. Upper Saddle River, N.J.

Brooks R. (1991) Comparative Task Analysis: An Alternative Direction for Human-Computer Interaction Science. In Carrol J.M. (Ed). *Designing Interaction: Psychology at the Human-Computer Interface*. Cambridge University Press. 50-59.

Brown J. & Marshall S. (1998) Sharing Human-Computer Interaction and Software Engineering Design Artefacts. *Proceedings of OZCHI'98*. Adelaide, Australia. 53-60.

Budd T. (1998) *Data structures in C++ using the Standard Template Library*. Addison-Wesley. Reading, Mass.

Cantor M. (1998) *Object-oriented project management with UML*. John Wiley & Sons. New York.

Cantu M. (1995) *Mastering Delphi*. Sybex. San Francisco, California.

Carroll J. (Ed) (1995) *Scenario-based Design*. John Wiley & Sons. New York.

Cheesman J. & Daniels J. (2001) *UML Components*. Addison-Wesley. Boston, Mass.

Chrusch M. (2000) Seven great myths of usability. *Interactions* 7 (5). 13-16.

Cockburn A. (1997) Structuring Use Cases with Goals. Draft paper.

Cockburn A. & Fowler M. (1998) Question Time! about use cases. *Proceedings of Conference on OOP Systems, Language and Applications*. Vancouver, Canada. 226-243.

Coleman D., Arnold P., Bodoff S., Dollin C., Gilchrist H., Hayes F. & Jeremaes P. (1994) *Object-Oriented Development: The Fusion Method*. Prentice-Hall. Englewood Cliffs, N.J.

Collins D. (1995) *Designing object-oriented user interfaces*. Benjamin/Cummings. Redwood City, California.

Conner D.B., Niguidula D. & van Dam A. (1995) *Object-Oriented Programming in Pascal*. Addison-Wesley. Reading, Mass.

Constantine L.L. (1995) Essential Modelling: Use Cases for User Interfaces. *ACM Interactions* 2 (2). 34-46.

Constantine L.L. & Lockwood L.A.D. (1999) *Software for Use*. Addison-Wesley. Reading, Mass.

Constantine L.L. & Lockwood L.A.D. (2001) Structure and style in use cases for user interface design. In van Harmelen M. (Ed) *Object Modeling and User Interface design: Designing interactive systems*. Addison-Wesley. Boston, Mass. 245-279.

Corlett D. (2000) Innovating with OVID. *Interactions* 7 (4). 19-26.

De Michelis G. (1999) Net Theory and Workflow Models. In Donatelli S. & Kleijn J. (Eds) *Application and Theory of Petri Nets 1999*. Lecture Notes in Computer Science 1639. Springer. Berlin, Germany.

Dix A., Finlay J., Abowd G. & Beale R. (1998) *Human-Computer Interaction* (2nd Edition). Prentice Hall Europe. London, UK.

Dowell J. (2001) e-mail communication from the Department of Computer Science, University College London. London, UK. 9th November 2001.

Eisenstein J. & Puerta A. (2000) Adaptation in automated user-interface design. *Proceedings of IUI 2000.* New Orleans, Louisiana. 74-81.

Eriksson H-E. & Penker M. (1998) *UML Toolkit.* John Wiley & Sons. New York.

Filippidou D. (1998) Designing with scenarios: A critical review of current research and practice. *Requirements Engineering 1998* (3). Springer-Verlag. 1-22.

Fowler M. (1997) A survey of object-oriented analysis and design methods. *Proceedings of ICSE 97.* Boston, Mass. 653-654.

Fowler M. & Scott K. (2000) *UML distilled* (2nd Edition). Addison-Wesley. Reading, Mass.

Galitz W.O. (1997) *The essential guide to user interface design.* John Wiley & Sons. N.Y.

Gregory K. (1998) *Special edition using Visual C++™ 6.* Que Corporation.

Gu J. (1995) *A menu interface development environment based on Lean Cuisine.* MSc thesis. Massey University. New Zealand.

Hackos J.T. & Redish J.C. (1998) *User and Task Analysis for Interface Design.* Wiley & Sons. New York.

Hartson H.R., Siochi A.C. & Hix D. (1990) The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Trans. on Information Systems* 8 (3). 181-203.

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

HyperCard (1992) Claris Corporation.

Jaaksi A., Aalto J-M., Aalto A. & Vatto K. (1999) *Tried and true object development.* Cambridge University Press.

Jacob R.J.K. (1983) Using formal specifications in the design of a human-computer interface. *Communications of the ACM* 26 (4). 259-264.

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

Jacobson I., Ericsson M. & Jacobson A. (1995) *The Object Advantage*. Addison-Wesley. Wokingham, UK.

Jacquot J-P. & Quesnot D. (1997) Early specification of user interfaces: towards a formal approach. *Proceedings of ICSE 97*. Boston, Mass. 150-160.

Janssen C., Weisbecker A. & Ziegler J. (1993) Generating user interfaces from data models and dialogue net specifications. *Proceedings of INTERCHI'93*. ACM. 418-423.

Jarzabek S. & Huang R. (1998) The Case for User-Centered CASE Tools. *Communications of the ACM* 41 (8). 93-99.

John B.E. & Kieras D.E. (1996) Using GOMS for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction* 3 (4). 287-319.

Joy B. (1999) *Statechart Implementation.* Honours Project Report. Institute of Information Sciences & Technology. Massey University. New Zealand.

Kemp E.A. & Phillips C.H.E. (1998) Extending Support for User Interface Design in Object-Oriented Software Engineering Methods. *Proceedings of 13th Conference on HCI.* Sheffield, UK. September 1998. British Computer Society. 96-97.

Kirwan B. & Ainsworth L.K. (1992) *A Guide to Task Analysis.* Taylor & Francis. London, UK.

Kobryn C. (1999) UML 2001: A standardization odyssey. *Communications of the ACM* 42 (10). 29-37.

Kobryn C. (2000) Modelling components and frameworks with UML. *Communications of the ACM* 43 (10). 31-38.

Knaster S. & Rollin K. (1992) *Macintosh Programming Secrets.* Addison-Wesley. Reading, Mass.

Kovacevic S. (1998) UML and user interface modelling. In Bezivin J. & Muller P-A. (Eds) *Lecture Notes in Computer Science 1618*. Springer-Verlag. New York.

Kruchten P. (1999) *The Rational Unified Process*. Addison-Wesley. Reading, Mass.

Kruchten P., Ahlqvist S. & Bylund S. (2001) User Interface Design in the Rational Unified Process in van Harmelen M. (Ed) *Object Modelling and User Interface Design.* Addison-Wesley. Boston, Mass. 161-196.

Landay J.A. & Myers B.A. (1995) Interactive sketching for the early stages of user interface design. *Proceedings of CHI'95*. Denver, Colorado. 43-50.

Lee R.C. & Tepfenhart W.M. (1997) *UML and C++*. Prentice-Hall. Upper Saddle River, N.J.

Lending D. & Chervany N.L. (1998) The use of CASE tools. *Proceedings of the conference on computer personnel research*. Boston, Mass. March 1998. 49-58.

Li L. (2003) *The automatic generation and execution of Lean Cuisine+ specifications*. MSc thesis. Massey University, New Zealand.

Mandel T. (1997) *The elements of user interface design*. John Wiley & Sons. New York.

May J.C. & Whittle J.B. (1991) *Extending the Macintosh® Toolbox: Programming Menus, Windows and More*. Addison-Wesley. Reading, Mass.

McDaniel S.E., Olson G.M. & Olson J.S. (1994) Methods in search of methodology – combining HCI and object-orientation. *Proceedings of Human Factors in Computing Systems*. Boston, Mass. April 1994. 145-151.

McInerney P. & Sobiesiak R. (2000) The UI design process. *SIGCHI Bulletin* 32 (1). 17-21.

Muller P-A. (1997) *Instant UML*. Wrox Press Ltd. Birmingham, UK.

Myers B. (1995) User interface software tools. *ACM Transactions on Computer-Human Interaction* 2 (1). 64-103.

Myers B., Hudson S.E. & Pausch R. (2000) Past, present and future of user interface software tools, *ACM Transactions on Computer-Human Interaction* 7 (1). 3 –28.

Odell J.J. (1998) *Advanced object-oriented analysis and design using UML*. Cambridge University Press.

Oestereich B (1999) *Developing software with UML*. Addison-Wesley. Harlow, UK.

Overmeyer S. (2000) *Conceptual modelling through linguistic analysis using LIDA*. Seminar at Massey University. Auckland, New Zealand. 8 May 2000.

Paech B. (1998) On the role of Activity diagrams in UML – A user task centred development process for UML. In Bezivin J. & Muller P-A. (Eds) *Lecture Notes in Computer Science 1618*. Springer-Verlag. New York.

Peterson J.L. (1981) *Petri net theory and the modelling of systems*. Prentice-Hall. Upper Saddle River, N.J.

Phillips C.H.E. (1991) Towards a notation for describing the behaviour of direct manipulation interfaces. *New Zealand Journal of Computing* 3 (1). 11-25.

Phillips C.H.E. (1993) *The Development of an Executable Graphical Notation for Describing Direct Manipulation Interfaces*. PhD thesis. Massey University, New Zealand.

Phillips C.H.E. (1994) Review of Graphical Notations for Specifying Direct Manipulation Interfaces. *Interacting with Computers* 6 (4). 411-431.

Phillips C.H.E. (1995) Lean Cuisine+: An Executable Graphical Notation for Describing Direct Manipulation Interfaces. *Interacting with Computers* 7 (1). 49-71.

Phillips C.H.E. & Apperley M.D. (1991) Direct manipulation tasks: a Macintosh-based analysis. *Interacting with Computers* 3 (1). 9-26.

Phillips C.H.E. & Kemp E.A. (2002) In Support of User Interface Design in the Rational Unified Process. *Proceedings of AUIC 2002*. Melbourne, Australia. January 2002. 21-27.

Phillips C.H.E., Kemp E.A. & Kek S-M. (2001) Extending UML use case modelling to support graphical user interface design. *Proceedings of ASWEC 2001*. IEEE Press. Canberra, Australia. 48-57.

Phillips C.H.E., Kemp E.A. & Scogings C.J. (2001) Extending UML to support user interface design. *Proceedings of ACM symposium on CHI*. Palmerston North, New Zealand. 55-60.

Phillips C.H.E. & Scogings C.J. (1995) Task and Object Modelling in high level GUI Design: An integrated approach. *Proceedings OZCHI'95*. Wollongong, Australia. 24-29.

Phillips C.H.E. & Scogings C.J. (1997) Modelling the mock-up: towards the automatic specification of the behaviour of early prototypes. *Proceedings INTERACT'97*. Sydney, Australia. July 1997. IFIP, Chapman & Hall, London. 591-592.

Phillips C.H.E. & Scogings C.J. (1998) Towards the automatic specification of the behaviour of early prototypes using Lean Cuisine+. *Proceedings Software Engineering & Practice*. Dunedin, New Zealand. 238-244.

Phillips C.H.E. & Scogings C.J. (2000) Task and Dialogue Modelling: Bridging the Divide with Lean Cuisine+. *Australian Computer Science Communications* 22 (5). 81-87.

Preece J. (Ed) (1993) *A guide to usability: Human factors in computing.* Addison-Wesley. Wokingham, UK.

Prosise J. (1996) *Programming Windows®95 with MFC.* Microsoft Press. Redmond, Washington.

Puerta A., Cheng E., Ou T. & Min J (1999) MOBILE: User-Centered Interface Building. *Proceedings of CHI'99.* Pittsburgh, Pennsylvania. 426-433.

Quatrani T. (2000) *Visual Modelling with Rational Rose 2000 and UML.* Addison-Wesley. Reading, Mass.

Reisig W. & Rozenberg G. (Eds) (1998) *Lectures on Petri Nets I: Basic Models.* Lecture Notes in Computer Science 1491. Springer-Verlag. New York.

Richter C. (1999) *Designing flexible object-oriented systems with UML.* Macmillan Technical. Indianapolis, Indiana.

Rink R.A., Wisenbaker V.B. & Vance R.G. (1995) *Programming in Macintosh® and Think Pascal™* (2nd Edition). Prentice-Hall. Englewood Cliffs, New Jersey.

Roberts D., Berry D. & Isensee S. (1997) OVID: Object View and Interaction Design. *Proceedings of IFIP TC13 International Conference.* Sydney, Australia. July 1997. Chapman & Hall. London, England. 663-664.

Rolland C., Ben Achour C., Cauvet C., Ralyte J., Sutcliffe A., Maiden N., Jarke M., Haumer P., Pohl K., Dubois E. & Heymans P. (1998) A proposal for a scenario classification framework. *Requirements Engineering* 3. Springer-Verlag. 23-47.

Rosson M.B. & Carroll J.M. (1995a) Introduction to object-oriented design: A minimalist approach. *Proceedings of CHI'95.* Denver, Colorado. 361-362.

Rosson M.B. & Carroll J.M. (1995b) Integrating Task and Software Development for Object-oriented Applications. *Proceedings of CHI'95.* Denver, Colorado. 377-384.

Rosson M.B. & Carroll J.M. (2001) Scenarios, objects and points of view in user interface design. In van Harmelen (Ed) *Object Modelling and User Interface Design.* Addison-Wesley. Boston, Mass. 39-69.

Rourke C. (2002) Making UML the lingua franca of usable system design. *Interfaces* 50. British HCI Group. Swindon, UK.

Rumbaugh J. (1995) What is a method? *JOOP* 8 (6) October 1995. 10-16.

Rumbaugh J. (1996) Notation notes: Principles for choosing a notation. *JOOP* May 1996. 11-14.

Rumbaugh J., Jacobson I. & Booch G. (1999) *The Unified Modelling Language Reference Manual.* Addison-Wesley. Reading, Mass.

Schildt H. (1998) *MFC Programming from the ground up.* Osborne/McGraw-Hill. Berkeley, California.

Scogings C.J. (2000) The Lean Cuisine+ notation revised. *Research Letters in the Information & Mathematical Sciences* 2000 (1). Massey University. Auckland, New Zealand. 17-23.

Scogings C.J. & Phillips C.H.E. (1998) Beyond the interface: modelling the interaction in a visual development environment. *Proceedings HCI'98.* Sheffield, UK. 108-109.

Scogings C.J. & Phillips C.H.E. (2001a) A method for the early stages of interactive systems design using UML and Lean Cuisine+. *Australian Computer Science Communications* 23 (5). 69-76.

Scogings C.J. & Phillips C.H.E. (2001b) Linking tasks, dialogue and GUI design: A method involving UML and Lean Cuisine+. *Interacting with Computers* 14 (1). 69-86.

Scogings C.J. & Phillips C.H.E. (2003) Linking task and dialogue modelling: Toward an integrated software engineering method. In Diaper D. & Stanton N. (Eds.) *Handbook of Task Analysis for Human-Computer Interaction*. TechBooks. Fairfax, Virginia. [In print]

Shneiderman B. (1998) *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (3rd edition). Addison-Wesley. Reading, Mass.

Sommerville I. & Sawyer P. (1997) *Requirements Engineering: A good practice guide.* John Wiley & Sons. New York.

Sommerville I. (2001) *Software Engineering* (6th Edition). Addison-Wesley. Reading, Mass.

Sphar C. (1999) *Learn Microsoft® Visual C++® 6.0 now.* Microsoft Press. Redmond, Washington.

Stevens P. & Pooley R. (2000) *Using UML: Software Engineering with Objects and Components* (Updated Edition). Addison-Wesley. New York.

Sydow D.P. (1995) More Mac Programming Techniques. M&T Books. New York.

Szekely P. (1994) User interface prototyping: Tools and techniques. *Proceedings of ICSE'94* – workshop on SE-HCI. Sorrento, Italy. 76-92.

Tam R.C-M., Maulsby D. & Puerta A.R. (1998) U-TEL: A tool for eliciting user task models from domain experts. *Proceedings of IUI'98*. San Francisco, California. 77-80.

Terwilliger R.B. & Polson P.G. (1997) Relationships between users' and interfaces' task representations. *Proceedings Conference on Human Factors and Computing Systems*. Atlanta, Georgia. 99-106.

Weinschenk S., Jamar P. & Yeo S.C. (1997) *GUI Design Essentials*. John Wiley & Sons. New York.

Appendix 1. The Lean Cuisine+ Notation

The Lean Cuisine+ notation was developed in (Phillips, 1995) as an executable semi-formal graphical notation for describing the underlying behaviour of event-based direct manipulation interfaces. Lean Cuisine+ builds on the original Lean Cuisine notation introduced in (Apperley & Spence, 1989). During the construction of a CASE tool for the notation, as well as further research into the use of the Lean Cuisine+ notation, various changes were made and the revised notation is presented here. The format and much of the content of this document follows that of Appendix C of (Phillips, 1993) in order to easily distinguish between the earlier and later versions of the notation.

The object model

The object model has been removed from the Lean Cuisine+ notation in order to improve the clarity of the diagrams since boxes around menemes (as well as subtypes) are no longer required. The distinction between actions, syntactic objects and semantic objects was often difficult to determine, particularly in the automatic generation of Lean Cuisine+ diagrams for existing interfaces. The work on the automatic generation of Lean Cuisine+ diagrams has been published in (Phillips & Scogings, 1997) and (Scogings & Phillips, 1998). Since the object structure of the interface has been removed from the Lean Cuisine+ dialogue tree, it needs to be described elsewhere. UML case diagrams can be used for this purpose as described in (Scogings & Phillips, 2001b).

Dialogue

A *dialogue* is described by a set of selectable primitives, *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*. The top level or *root dialogue*, when made available through selection of its *dialogue header* (the root node), provides access to the *subdialogue(s)* and/or *state variable values* immediately below it, subject to any constraints applying.

Menemes

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

Real menemes represent specific selectable options, and may be terminal or non-terminal. Non-terminal real menemes are headers to further subdialogues. *Virtual* menemes are always non-

terminal and are used to partition a dialogue or subdialogue 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 sequence but this indicates that one or more of the real menemes within the subdialogue may be selected at that point in the task. See *Task Sequences* below.

In each dialogue state, a meneme is either *available* (for selection) or *unavailable*, 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. Menemes may be *bistable* (default), *select-only*, *deselect-only*, *monostable* or *passive*. The default type may be explicitly overridden by specifying one of the other types through the use of *meneme designators* (which are summarised at the end of this section).

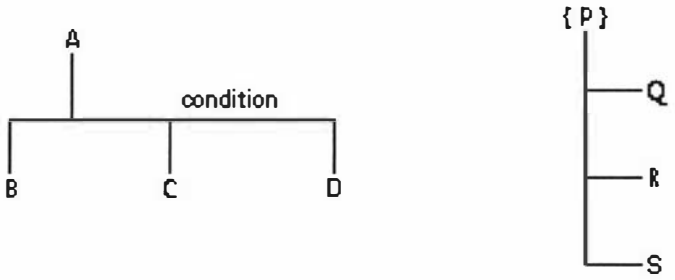
Each meneme can have attributes attaching to it. These can be either *meneme state variables* (MSVs) or *selection preconditions*. MSVs can also exist as parameters to options that may modify them as dialogue progresses. The exact form of presentation of MSVs on a Lean Cuisine+ tree diagram is implementation dependent.

A *selection precondition* is a precondition for the availability (for selection) of a meneme in the Lean Cuisine+ notation, which may involve the state of a meneme in another part of a dialogue, or the existence of some other state or condition, or both. The exact form of presentation is implementation dependent.

Subdialogues

Menemes may be grouped into *subdialogues* to any level of nesting. A subdialogue is a logical grouping of menemes within the body of a Lean Cuisine+ dialogue. The subdialogue corresponding to a non-terminal meneme in a dialogue tree consists of all the menemes with which the *subdialogue header* is connected by downward directed branches. When made available through selection of its header, a subdialogue provides access to the options represented by these menemes, subject to any constraints applying. A subdialogue is available for selection only if its parent dialogue is active.

The grouping of menemes within a subdialogue places constraints on their behaviour. The grouping may be either *mutually exclusive* (1 from *n*) or *mutually compatible* (*m* from *n*). In the latter case, selection may be governed by a condition. In the Lean Cuisine+ notation these structures are represented graphically as follows:



(a) (b)

(a) A mutually compatible meneme group within the subdialogue headed by the real meneme A.

(b) A mutually exclusive meneme group within the subdialogue headed by the virtual meneme P.

Where a group of options constitute a homogeneous set, they may be represented graphically using a *fork* symbol, as shown below:

(c) (d)

(c) The tree diagram for a mutually compatible homogeneous group.

(d) The tree diagram for a mutually exclusive homogeneous group.

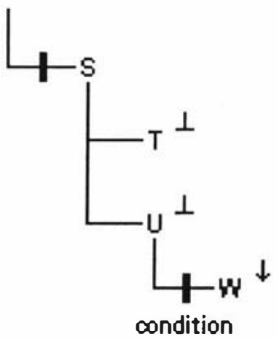
Modal subdialogues

A subdialogue may be *modal*. This means that the selection of the subdialogue causes all options in all other parts of the dialogue to become unavailable. A modal subdialogue that involves a subtree is designated by a bar across the arc leading to the meneme that heads the subdialogue. An example (headed by meneme S) appears in Figure (e).

A modal subdialogue may be *simple*, in which case it is represented by a *monostable* meneme, shown thus (⊥). An example (meneme T) appears in Figure (e). Following selection, a meneme of this type reverts to an unselected state on completion of the operation it represents, through system action.

Simple subdialogues are normally terminal nodes. Alternatively they may have explicit modal subdialogues associated with them which are triggered under certain conditions. In such cases the operation represented by the monostable meneme becomes suspended pending user

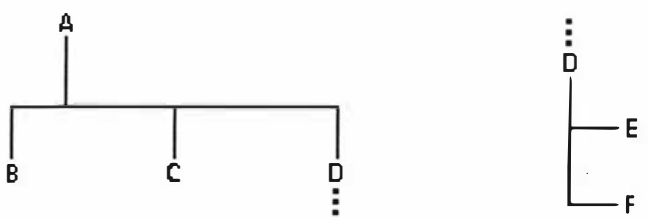
termination of the further modal subdialogue, when it completes in the usual way. An example (meneme U) appears in Figure (e).



(e) S represents a modal subdialogue, and T a simple modal subdialogue.
U represents a simple modal subdialogue which under certain conditions triggers a further modal subdialogue W requiring user action.

Stepwise refinement

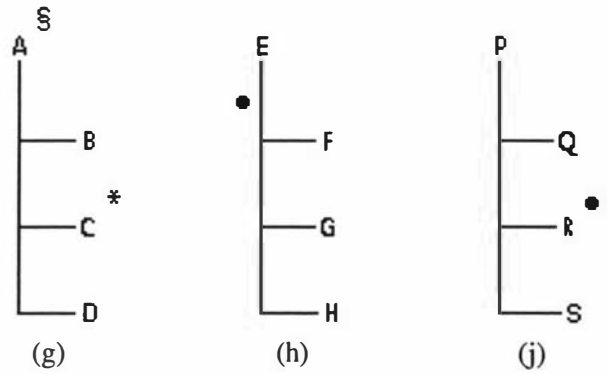
To support *stepwise refinement*, a dialogue or subdialogue tree may be represented visually by an ellipsis to indicate that the detail is developed in a separate diagram. In the case of a subdialogue, the lower level diagram is headed by an ellipsis in order to visually complete the association:



(f) Meneme D heads a subdialogue that is developed separately.

Further subdialogue constraints

A subdialogue header may be tagged to indicate 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*, which indicates a meneme that is to be initially in the selected state, must be shown (*) under these circumstances (see Figure (g) below). A *dynamic default* is a default that takes on the value of the last user selection from that group (•). It may have an initial assignment, or it may be initially unassigned, in which case the first user choice from that group becomes its first value (see Figures (h) and (j) below).



(g) (h) (j)

(g) A is a required choice group, and C is the initial default.

(h) An unassigned dynamic default applying to the subgroup FGH.

(j) A dynamic default applying to the group QRS, initially assigned to R.

Meneme designators

The following meneme designators are used in the Lean Cuisine+ notation:

{} *Virtual meneme:* a non-selectable meneme that is used to partition a dialogue or subdialogue. The meneme name appears between the braces.

◭ *Monostable meneme:* a meneme that, following user selection, reverts to an unselected state on completion of the simple modal subdialogue it represents, through system action.

↑ *Select-only meneme:* a meneme of this type cannot be directly deselected by the user, but may be deselected through selection of another mutually exclusive option.

↓ *Deselect-only meneme:* a meneme of this type cannot be directly selected by the user, but may become selected as an indirect result of some other action.

⊗ *Passive meneme:* a meneme of this type is unavailable for selection or deselection by the user and can only be selected and/or deselected by the system.

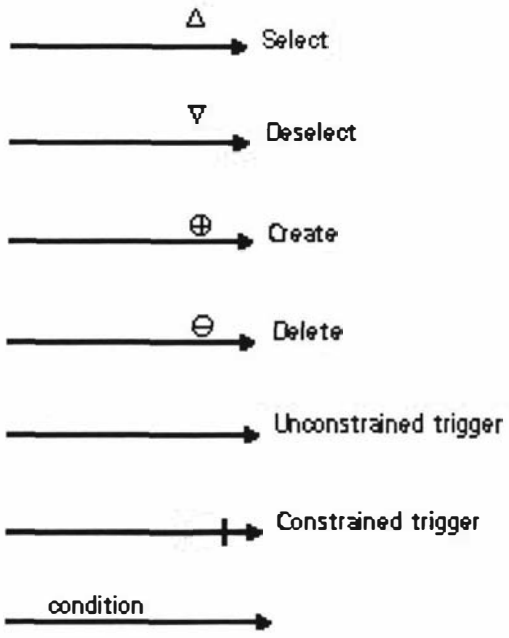
* *Default choice:* indicates a meneme that is to be in the selected state at the commencement of a dialogue or subdialogue.

§ *Required choice:* is associated with a subdialogue header and indicates a meneme group from which a valid choice is always required.

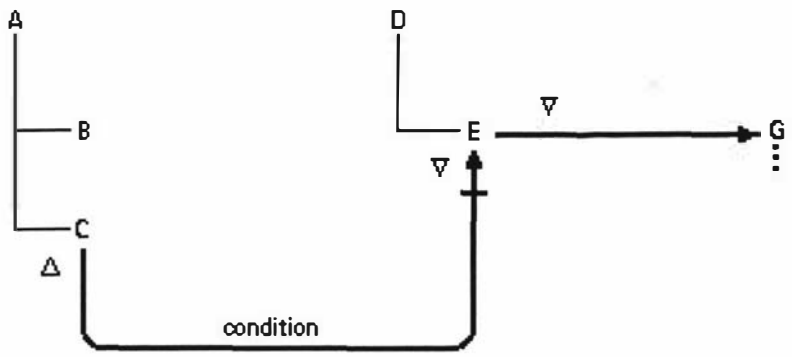
• *Dynamic default:* may be associated with a meneme (assigned) or a meneme group (unassigned) and indicates a default choice that takes on the value of the last user selection from that group.

Selection Triggers

A *selection trigger* is a directed link between menemes such that the selection or deselection of a meneme triggers (possibly conditionally) the selection or deselection of one or more other menemes. Selection triggers represent a system response to another action and are annotated as shown below.



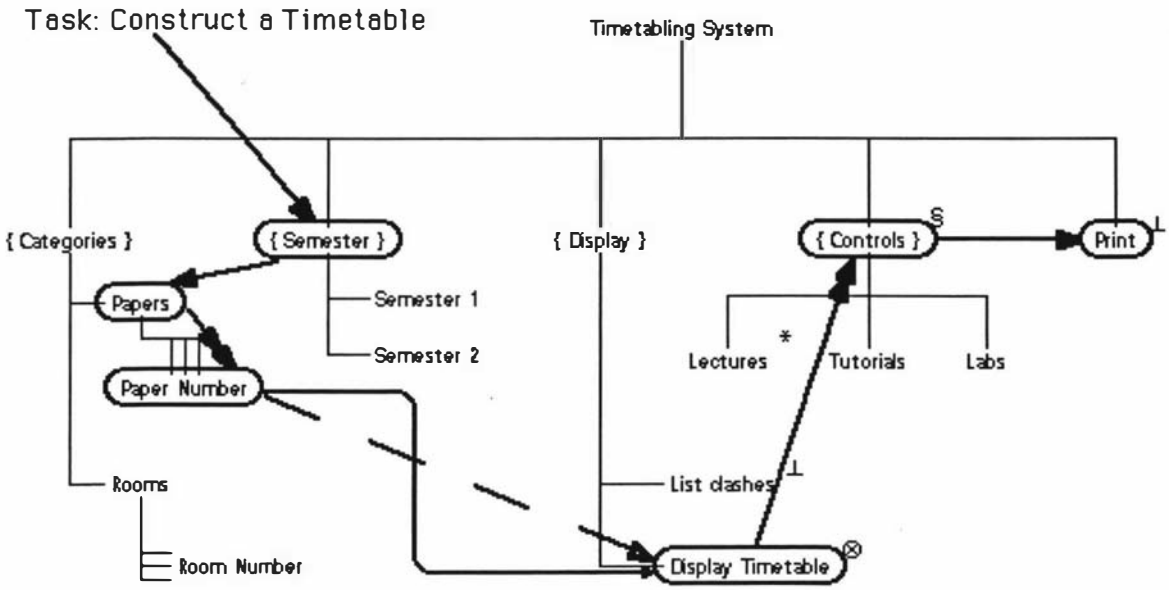
Triggers are either *unconstrained*, in which case further "knock on" effects are possible, or *constrained*, in which case the behaviour of the selected or deselected meneme is modified, inhibiting further links, and preventing side effects such as cycles. An example appears in Figure (k).



(k) Selection (Δ) of C causes deselection (∇) of E provided the condition is true.
The selection of G is not triggered because the first trigger is constrained.

Task Sequences

A *task sequence* describes a higher level user task in terms of primitive actions (and associated events), capturing any temporal relationships between them. The task sequence is overlayed on the dialogue tree diagram and is orthogonal to the constraint-based behavioural interface model.



(I) The task sequence for the task *Construct a Timetable*. The selection of the virtual meneme {Controls} actually indicates the selection of at least one of the menemes in the subdialogue. The double-headed arrow indicates that any combination of the menemes in the subdialogue may be selected since it is a mutually compatible subdialogue. The dashed arrow indicates the system selection of Display Timetable in response to the selection trigger.

The notation distinguishes user initiated and system initiated events by means of the links connecting selections. A solid link between two selections denotes that the new selection is the result of a user action, whereas a dashed link indicates that the new selection is a system response triggered by some other action. Selection and deselection actions are indicated as defined above for selection triggers.

A virtual meneme can be shown as being selected in a task sequence. This implies the selection of any allowable combination of menemes in the subdialogue below it. The double-headed arrow indicates that an action may be repeated. This may be used in conjunction with the selection of a virtual meneme.

The example provided in Figure (l) above, as well as further discussion on the usefulness of the new Lean Cuisine+ task sequence notation, has been published in (Phillips & Scogings, 2000).

Execution of a Lean Cuisine+ specification

A Lean Cuisine+ dialogue 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 - that is of simulating dynamically the behaviour of the dialogue as selections are made. This process takes cognisance of all specified constraints, such as selection triggers.

The exact form of presentation of the "execution" process within the context of a Lean Cuisine+ tree diagram is implementation dependent.

Appendix 2. The SELCU User Manual

Contents

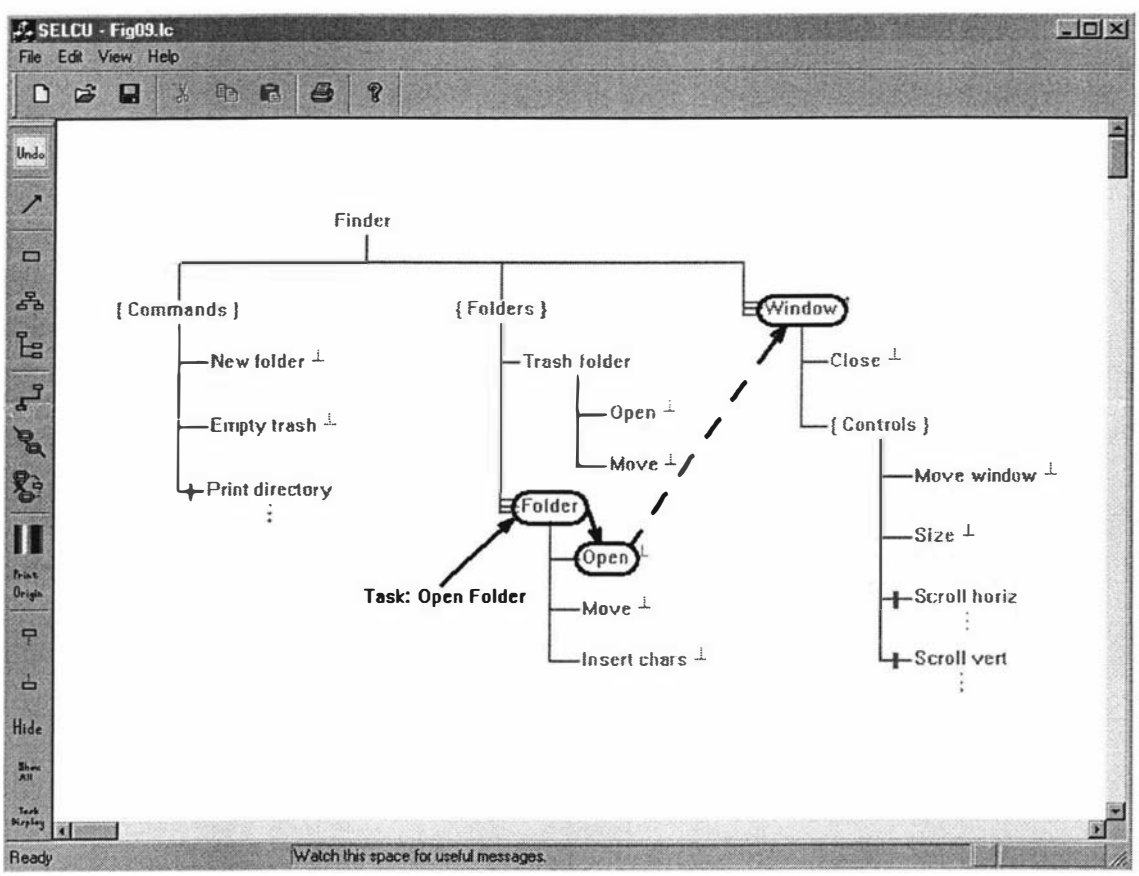


Figure 1: The SELCU screen layout

1. Introduction

This manual covers the use of the software environment supporting the Lean Cuisine+ notation (SELCU). SELCU is a CASE tool developed to provide a platform for the efficient production of Lean Cuisine+ tree diagrams and provides for the creation, editing, storage and printing of diagrams. The package does not pretend to be a commercially viable product. It is a prototype with no online help system and is subject to constant revision and upgrading as new Lean Cuisine+ features are evolved. SELCU is a Windows-based package that will run on Windows®2000 or similar. It was developed in Microsoft Visual C++®6.0 using Microsoft Foundation Classes.

SELCU has been designed to support the production of Lean Cuisine+ diagrams and was used to produce all the Lean Cuisine+ diagrams in this document. The system is designed to be as intuitive as possible and any part of a diagram can be directly selected. Single menemes can be placed at any point in the work area and new diagrams can be rapidly constructed and expanded by the addition of subdialogues where a mouse-click places each meneme in the subdialogue and links are automatically constructed back to the header meneme. Individual menemes can be edited by selecting the meneme and then changing the name, adding meneme designators, etc.

A large work area, surrounded by toolbars and status information, dominates the SELCU layout. The Lean Cuisine+ diagram is constructed in the work area. At the top of the interface is the title line that displays the file name of the diagram currently on display. This is followed by the menu line, containing four pull-down menus, and then the standard Windows toolbar. To the left of the work area is the SELCU toolbar. Below the work area is the status bar that provides useful messages indicating what action should be taken next. Figure 1 illustrates the SELCU interface in use with a typical Lean Cuisine+ diagram displayed in the work area.

SELCU provides direct manipulation of diagram objects wherever possible. Thus a meneme can be moved by selecting it with the mouse and "dragging" it to another location – the so-called "drag and drop" method. Most other objects can be moved in a similar manner. Any object can be edited (name changed, etc) by double-clicking on it and then interacting directly with the dialogue box that appears. Selecting an object and then pressing the **Delete** key will delete the object. The process of constructing a comprehensive Lean Cuisine+ diagram is outlined in the following sections.

2. The menus

Four pull-down menus appear on the top line of the interface: File, Edit, View and Help. The File menu provides the standard Windows options: New, Open, Save, Save As, Print, Print Preview, Print Setup, Exit and a list of most recently used diagrams. Several of these options are repeated on the standard Windows toolbar. The Edit menu provides the standard Windows options: Undo, Cut, Copy and Paste but none of these have been implemented in the current prototype. ***Note:*** *the Undo button provided on the SELCU toolbar is used to "un-delete" Lean Cuisine+ objects and is not related to the standard Windows Undo function.* The Help menu only provides the single option About SELCU. No on-line help is available in this prototype.

Figure 2: The SELCU View menu

The View menu, shown in Figure 2, provides the following options:

Menemes only	displays only the basic dialogue tree of linked menemes, excluding any that have been individually hidden
Unhide menemes	displays all menemes including any that have been previously hidden
Conditions	displays all meneme conditions and the list of conditions
Triggers	toggles display of all triggers excluding any that have been hidden
Unhide triggers	displays all triggers including any that have been previously hidden
Tasks	presents a list of tasks from which one (or none) can be selected for display
ALL tasks	displays all task sequences
Toolbar	toggles display of the standard Windows toolbar
Status Bar	toggles display of the standard Windows status bar

Some of these options are mutually exclusive, while others are mutually compatible. These relationships are nicely illustrated in Figure 3 which shows the Lean Cuisine+ diagram for the View menu.

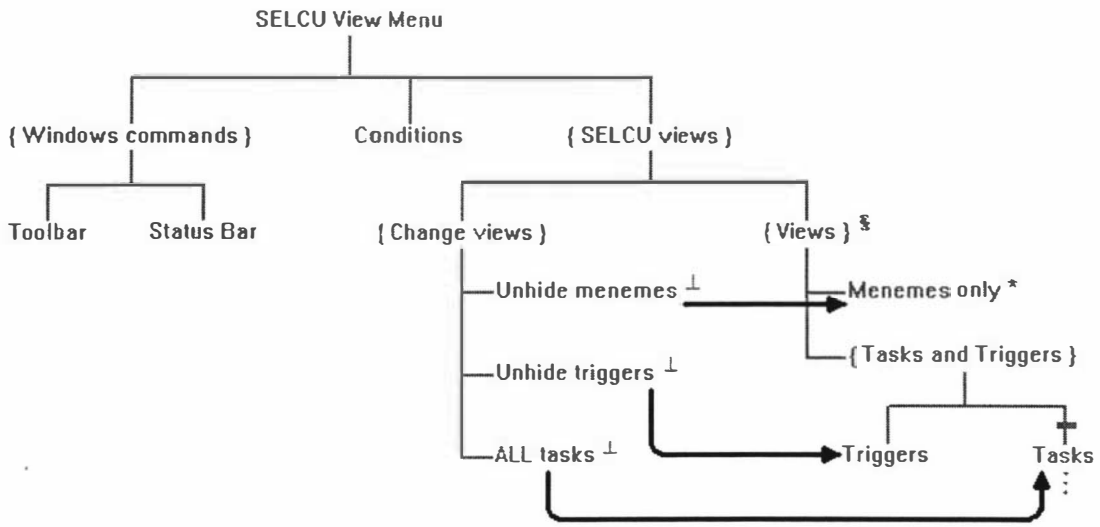


Figure 3: Lean Cuisine+ diagram for the SELCU View menu

Toolbar, Status Bar and Conditions are mutually compatible with all other options. Triggers and Tasks are mutually compatible options but are both mutually exclusive to the display of Menemes only. The selection of Tasks produces a modal subdialogue, indicated by the bar across the link to Tasks. Unhide menemes and Unhide triggers are both designated as monostable options as they revert to an unselected state once all previously hidden menemes (or triggers) have been displayed. ALL tasks is similarly designated monostable as it reverts to an unselected state once all tasks have been displayed.

The triggers displayed indicate that the selection of Unhide menemes, Unhide triggers or ALL tasks selects the corresponding display of menemes, triggers or tasks.

3. The toolbars

3.1 The standard toolbar

The standard Windows toolbar appears above the work area with the following options:

Start a new diagram

Open an existing new diagram

Save the current diagram

Cut – *not available in this prototype*

Copy – *not available in this prototype*

Paste – *not available in this prototype*

Print part (or all) of the current diagram

On-line help – *not available in this prototype*

3.2 The SELCU toolbar

To the left of the work area is the SELCU toolbar. This provides options relating directly to the construction and editing of Lean Cuisine+ diagrams in the work area. The following buttons are provided:

Undo a previous deletion

Neutral mode – *cancels the current mode*

Construct a new meneme – *click in the work area and a meneme will appear*

Construct a mutually compatible subdialogue – *click on the parent and then on each child.*

Construct a mutually exclusive subdialogue – *click on the parent and then on each child.*

Construct a new trigger – *select first meneme, turning points, and last meneme*

Construct a new task – *select the menemes to form the sequence*

Modify an existing task – *change order of sequence, add or remove menemes*

Change colours and font

Set the print origin – *the page is printed with this point as the top-left corner*

Place downwards ellipsis – *all succeeding (lower) menemes are replaced by ellipsis*

Place upwards ellipsis – *all preceding (higher) menemes are replaced by ellipsis*

Hide the currently selected object – *hide a meneme, trigger, task or list of conditions*

Display all menemes, triggers, tasks and list of conditions even if previously hidden

Select a task to display from the list of tasks (or choose to display no or all tasks).

The ellipsis buttons are toggles – the first button press replaces other menemes with the ellipsis and the next button press replaces the ellipsis with the other menemes. Selecting a meneme, trigger or task and pressing the **Delete** key will delete the selected object. The Undo button on the toolbar can be used to Undo the most recent deletion. The use of all these buttons will be explained in the following sections.

4. Modes

The selection of several of the toolbar options causes the application to change into a different *mode*. It is useful for the user to be aware of the current mode. For example, if *Construct a new meneme* is selected, any click of the mouse inside the work area will cause a new meneme to appear. An indication of the current mode appears on the status bar at the bottom of the screen. Pressing the **Escape** key or selecting the *Neutral mode* option will cancel any and all special effects of the current mode.

5. Constructing a Lean Cuisine+ tree diagram

The sequence that follows describes the construction of the Finder example shown in Figure 1.

5.1 Start with a mutually compatible subdialogue

To construct a new Lean Cuisine+ tree diagram, proceed as follows:

1. Select *Construct a new meneme* from the SELCU toolbar.
2. Click anywhere in the work area and a new meneme will appear.

3. Select *Construct a mutually compatible subdialogue* from the SELCU toolbar.
4. Select the existing meneme to be the parent of the new subdialogue – the meneme will be highlighted.
5. Click at three points in the work area. Each click will create a new meneme with a link back to the existing (parent) meneme. Press the **Escape** key to stop. The diagram should now look like that in Figure 4.

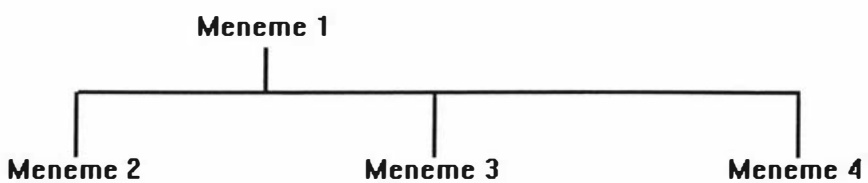


Figure 4: Construction of a mutually compatible subdialogue

5.2 Add a mutually exclusive subdialogue

6. Select *Construct a mutually exclusive subdialogue* from the SELCU toolbar. 
7. Select Meneme 2 to be the parent of the new subdialogue – it will be highlighted.
8. Click at three points in the work area. Each click will create a new meneme with a link back to the parent – Meneme 2. Press the **Escape** key to stop. The diagram should now look like that in Figure 5.

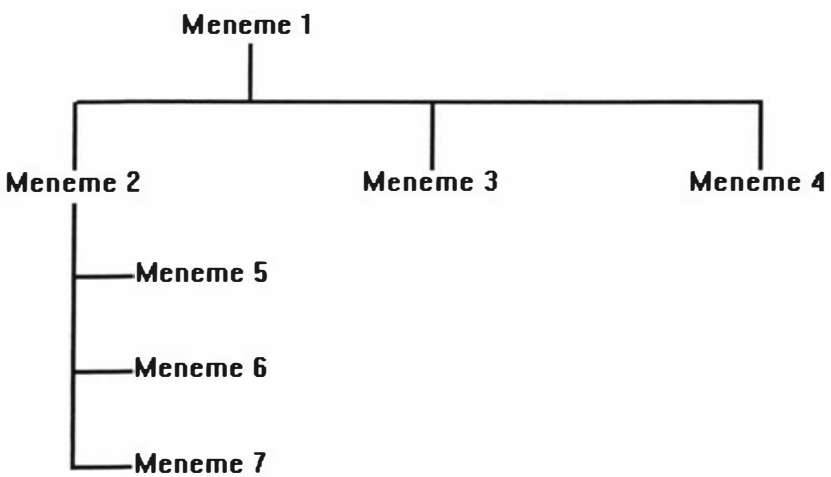


Figure 5: Addition of a mutually exclusive subdialogue headed by Meneme 2

5.3 Add meneme names and designators

9. Double-click on Meneme 5. This will open the Edit Meneme dialog box for Meneme 5 as illustrated in Figure 6. Enter a new name (New folder), then tick Monostable and then select OK.

10. Similarly change Meneme 1 to Finder, Meneme 4 to Window, Meneme 6 to Empty trash and Meneme 7 to Print Directory. Also designate Empty trash as monostable.

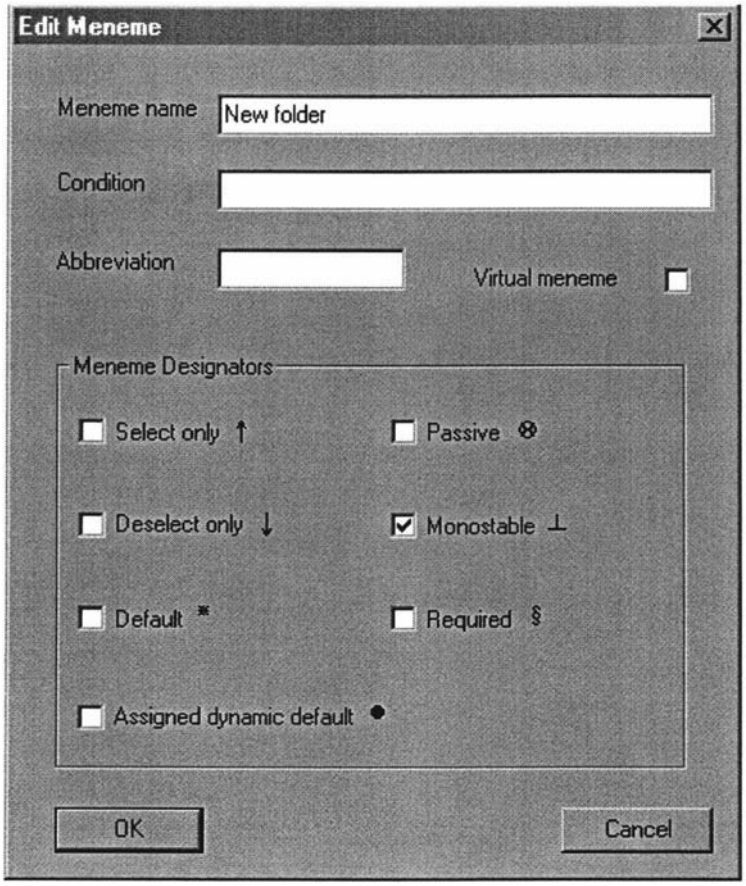


Figure 6: The Edit Meneme dialog box with Monostable selected.

5.4 Create and name a virtual meneme

11. Double-click on Meneme 2. In the Edit Meneme dialog box, enter the name Commands and tick Virtual Meneme. Then select OK.
12. Double-click on Meneme 3. In the Edit Meneme dialog box, enter the name Folders and tick Virtual Meneme. Then select OK. The diagram should now look like Figure 7.

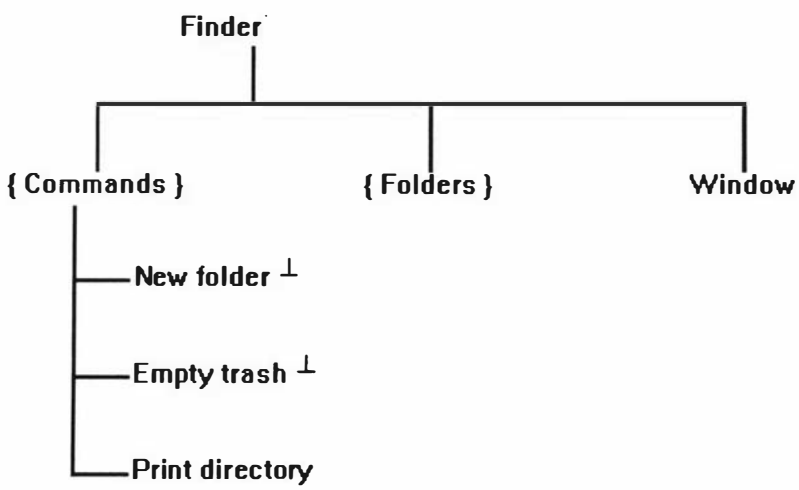


Figure 7: New names, virtual menemes and designators

5.5 Modify link designators

13. Double-click on the link (line) leading to Window. This will open the Edit Link dialog box as illustrated in Figure 8. Select Exclusive Fork followed by OK.

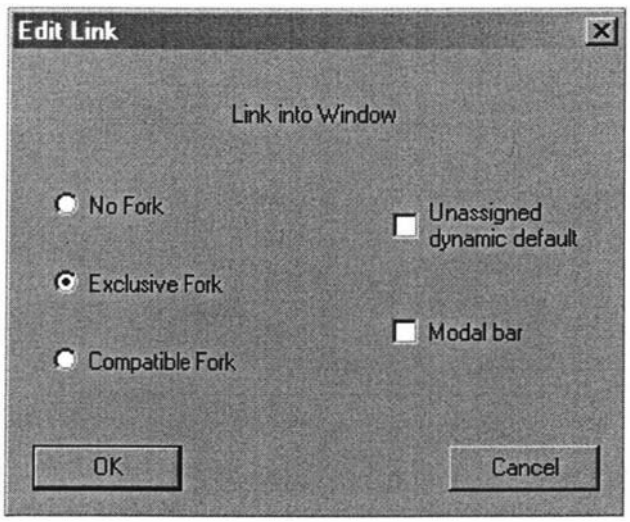


Figure 8: The Edit Link dialog box for Window with Exclusive Fork selected.

14. Double-click anywhere on the link into Print directory. In the Edit Link dialog box, select Modal bar and then OK. The diagram should now look like that in Figure 9.

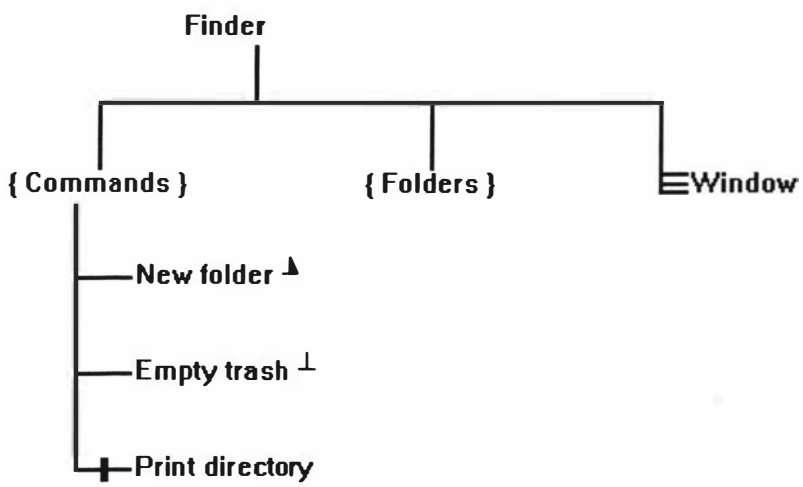


Figure 9: Exclusive fork on Window and modal bar on link into Print directory.

15. Add other menemes and rename and/or designate them as required to complete the diagram shown in Figure 1.

5.6 Optional – create links between existing menemes

The above process describes how to construct a tree structure by creating new menemes. It is also possible to create new links between existing menemes in a diagram. For example, if Meneme 1, Meneme 2 and Meneme 3 all existed but were not linked together, they could be combined into a mutually compatible subdialogue as described below:

1. Select *Construct a mutually compatible subdialogue* from the SELCU toolbar. 
2. Select Meneme 1 to be the parent of the new subdialogue – Meneme 1 will be highlighted.
3. Select Meneme 2 and then Meneme 3. As each meneme is selected, the links back to Meneme 1 will be created.
4. Press the **Escape** key to stop or select the *Neutral mode* option from the SELCU toolbar.

These two techniques can be combined to enable the creation of subdialogues with a combination of existing and newly created menemes.

6. Meneme conditions

6.1 Creating a condition

A condition (for availability for selection) can be attached to a meneme by double-clicking on the meneme to reveal the Edit Meneme dialog box and entering a condition. The condition will be displayed alongside the meneme. If the condition is too long to be displayed within the diagram, an abbreviation can also be entered in the Edit Meneme dialog box. In this case, the abbreviation is displayed alongside the meneme and the condition is displayed in a separate list of conditions.

6.2 Hiding a condition

Deselecting (or selecting) the Conditions option from the View Menu will hide (or display) all conditions including the list of conditions. The list of conditions can be moved by "drag and drop" and can also be hidden by selecting it and then selecting Hide from the SELCU toolbar. If hidden in this manner, it can be redisplayed by selecting Show All.

In Figure 10, a condition has been attached to Meneme 3. This condition is rather lengthy so an abbreviation, C1, has been provided and the condition appears in the list of conditions. This list can be moved to any suitable position in the work area. Meneme 4 has been provided with a shorter condition and no abbreviation is necessary.

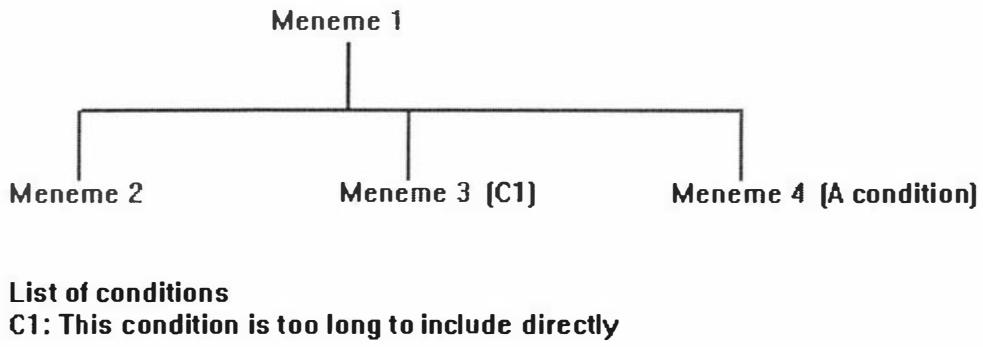


Figure 10: A condition and a condition with abbreviation.

7. Deleting menemes

A meneme can be deleted at any time be selecting the meneme (which will highlight it) and pressing the **Delete** key. The meneme will immediately be deleted and disappear – there is no confirmation process. If necessary, the deletion can be reversed by selecting the Undo option from the SELCU toolbar. When Undo is selected, the most recently deleted meneme reappears. Any number of deleted menemes can be made to re-appear in reverse order to that in which they were deleted.

8. Hiding and displaying parts of the tree

A large tree diagram can easily become cluttered and it is useful to be able to hide certain parts of the tree to enable the focus to be directed on the remaining sections.

8.1 The upwards ellipsis

To focus on one particular subdialogue (and nothing else), select the meneme heading the subdialogue and then select Place Upwards Ellipsis from the SELCU toolbar. An upwards ellipsis will appear and the rest of the tree will be hidden. See Figure 11.

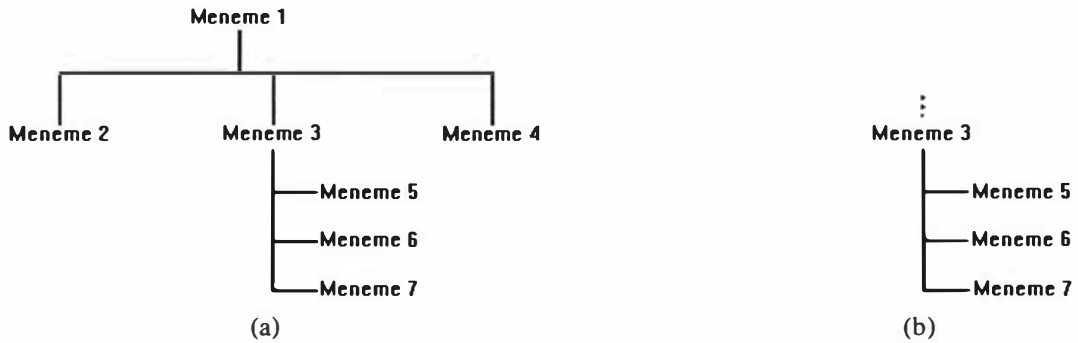


Figure 11: (a) before and (b) after selecting Place Upwards Ellipsis with Meneme 3

8.2 The downwards ellipsis

To focus on other sections of the tree, and not on a particular subdialogue, select the meneme heading the subdialogue and then select Place Downwards Ellipsis from the SELCU toolbar. A downwards ellipsis will appear and the subdialogue will be hidden.

See Figure 12.

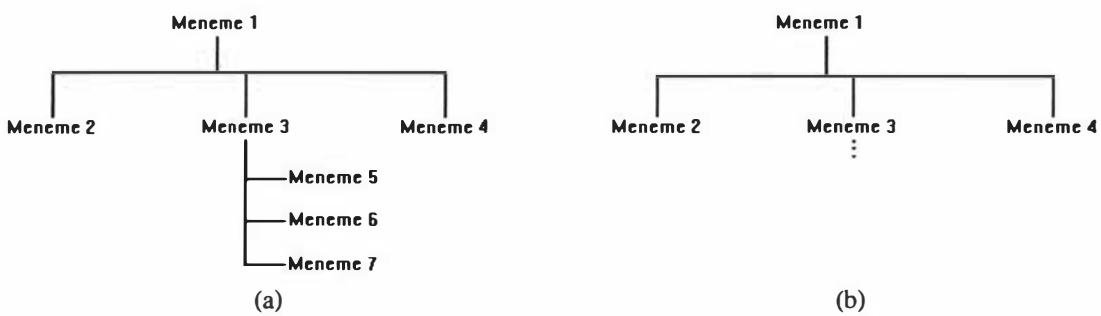


Figure 12: (a) before and (b) after selecting Place Downwards Ellipsis with Meneme 3

The options Place Upwards (or Downwards) Ellipsis *toggle* the appearance of the ellipsis. Thus if a meneme is selected that already has an upwards (or downwards) ellipsis, and the corresponding option is selected, the ellipsis will disappear and the original tree structure will be displayed.

Selecting the meneme and then selecting the Hide option from the SELCU toolbar will also hide individual menemes. Any number of menemes can be hidden in this way. All hidden menemes can be displayed by selecting Unhide menemes in the View menu. All hidden menemes will also be displayed if the Show All option is selected. Note that the selection of Show All does not affect menemes hidden by using an ellipsis. These can only be displayed by toggling the ellipsis option.

Figure 13 illustrates what happens when a meneme is hidden. Meneme 3 is hidden and is removed from the diagram, along with all links into and out of that meneme. Selecting the Unhide menemes option in the View menu will restore Meneme 3 along with the links.

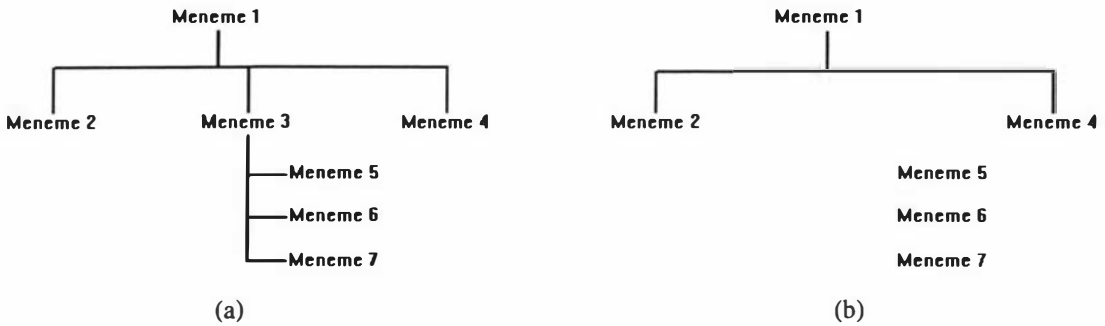


Figure 13: (a) before and (b) after hiding Meneme 3

9. Moving menemes and links

Menemes and links can be moved by the "drag and drop" method. If a meneme is moved, links to and from that meneme remain in the same position relative to the meneme. For example, Figure 14 shows what happens when a meneme is moved.

Figure 14: (a) before moving Meneme 1 and (b) after moving Meneme 1 to the right

Meneme 1 is moved to the right and the downward link from the meneme moves with it, but the position of the downward link relative to the meneme, remains unchanged.

If the link is moved, it changes its position relative to the meneme as illustrated in Figure 15. The downward link from Meneme 1 is moved to the right. The meneme does not move. The link moves and changes its position relative to Meneme 1 (and all other menemes).

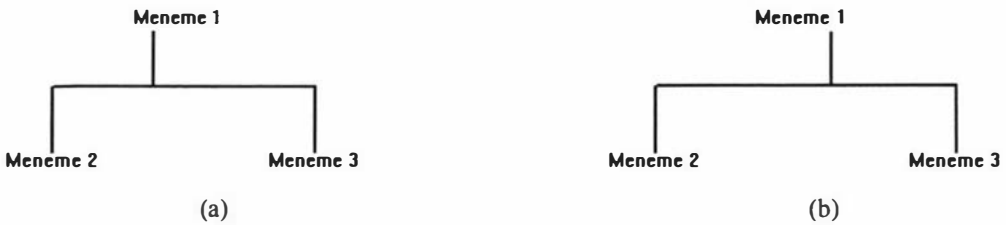


Figure 15: (a) before moving the link and (b) after moving the link to the right

10. Working with groups of menemes

10.1 Creating a group

Menemes can be formed into groups. This is purely a convenience for moving or editing the diagram and these groups need not correspond in any way to the subdialogues of the dialogue.

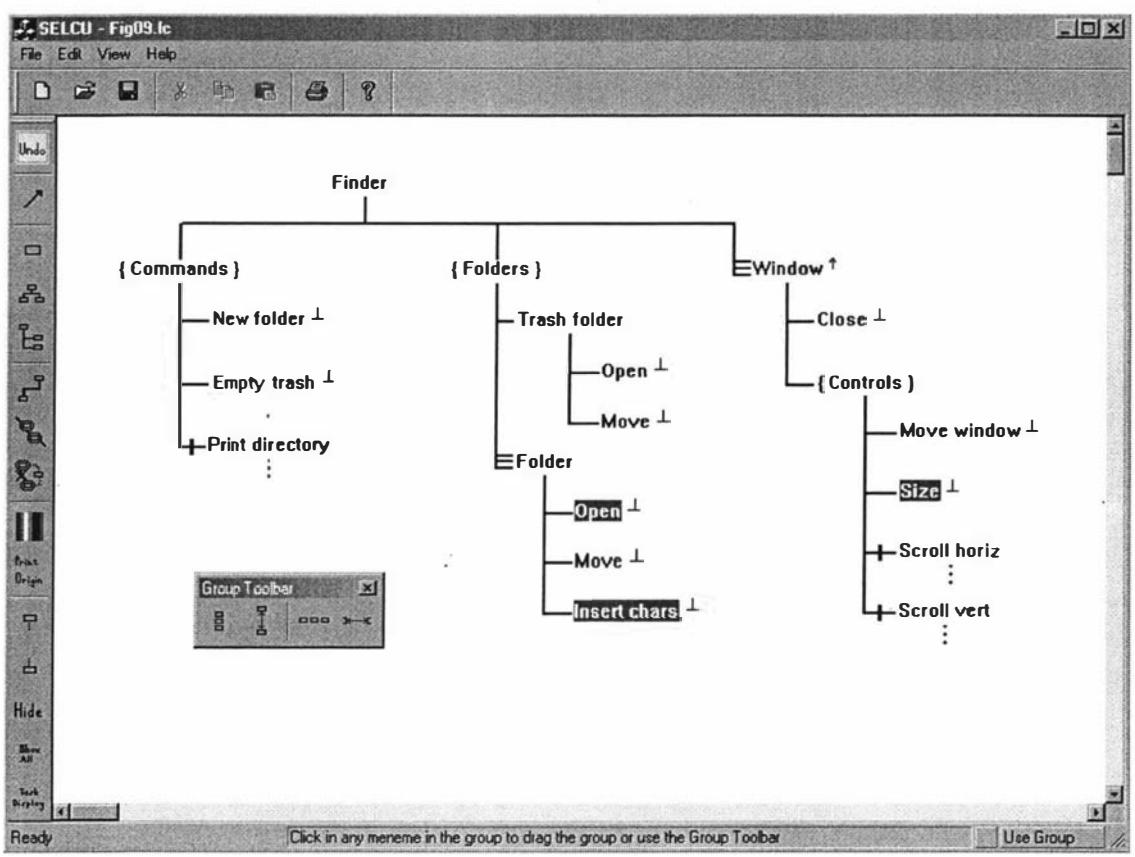


Figure 16: SELCU showing a group and the group toolbar

A group is usually formed by performing a "click and drag" away from any diagram objects. This action will cause a faint, dotted "group outline" to appear in the work area. Move the mouse to stretch, or shrink, the group outline around several menemes. These menemes will all

be included in the group. Figure 16 shows a selected group of menemes.

Important: *A meneme will be included in the group if the group outline touches any part of the meneme. It is not necessary to totally include the meneme within the group outline.*

Once a group has been formed, the group outline will disappear and all menemes in the group will be highlighted in the group colour. At this stage, it is possible to include or exclude individual menemes by selecting the meneme with "Ctrl + mouse click". The formation of a group causes the *group toolbar* to appear in the work area. The group toolbar can be moved to any position in the work area by using "drag and drop".

10.2 Adjusting meneme spacing within a group

Selecting Vertical Spacing will cause the uppermost and lowermost menemes in the group to retain their positions. All other menemes in the group will be shifted vertically so that the gap between all menemes in the group is equidistant. Selecting Horizontal Spacing will cause the leftmost and rightmost menemes in the group to retain their positions. All other menemes in the group will be shifted horizontally so that the gap between all menemes in the group is equidistant.

Vertical line up

Vertical spacing

Horizontal line up

Horizontal spacing

10.3 Lining up menemes within a group

The Vertical or Horizontal Line Up options can only be used if a *group leader* has been chosen for the group. Simply click once inside the meneme selected to be group leader and this meneme will be highlighted in a different colour from the rest of the group. Selecting Vertical Line Up will cause the left edge of all menemes in the group to line up with the left edge the group leader. Selecting Horizontal Line Up will cause the lower edge of all menemes in the group to line up with the lower edge of the group leader. Figure 17 shows the effect of selecting Horizontal Line Up and Horizontal Spacing.

(a) (b)

Figure 17: (a) before and (b) after Horizontal Line Up *and* Horizontal Spacing.

Groups can also be manipulated without using the Group Toolbar at all. The entire group can be moved by "dragging and dropping" any one of the group menemes. During this process, the meneme being used to move the whole group will automatically become the group leader and be highlighted as such. ***Warning:*** *if the header of a subdialogue is part of a moving group but the rest of the subdialogue is not, the links from header to other menemes may be compromised.*

10.4 Hiding a group

Selecting the Hide option from the SELCU toolbar (after the group has been formed) will hide a group. All menemes in the group will disappear. These menemes can be displayed at a later date by selecting the Show All option from the SELCU toolbar.

10.5 Deleting a group

A group can also be deleted by pressing the **Delete** key after forming the group. This will delete every meneme in the group. Selecting the Undo option on the SELCU toolbar will reverse the deletion, but only one meneme at a time. The group does not reappear as a group.

11. Selection triggers

11.1 Constructing a trigger

To construct a new trigger in the diagram, proceed as follows. Note that the process of constructing a new trigger can be abandoned at any time by pressing the **Escape** key or by selecting the *Neutral Mode* option from the SELCU toolbar.

1. Select *Construct a new trigger* from the SELCU toolbar. 
2. Select any existing meneme as the *source meneme* for the trigger.
3. Click at various points in the work area that are not inside other menemes – these points will form the turning points for the trigger.
4. To end the trigger, select another meneme as the *target meneme* for the trigger.
5. The trigger will be then be displayed between the two menemes.
6. Double-click somewhere on the new trigger. This will open the Edit Trigger dialog box as illustrated in Figure 18.

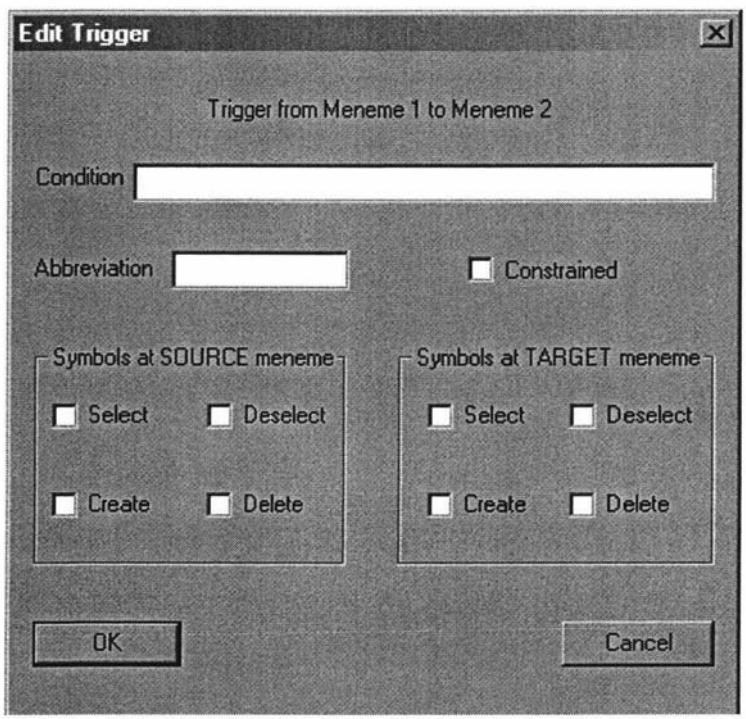


Figure 18: The Edit Trigger dialog box

11.2 Adding a trigger condition

The trigger can be supplied with a condition that will be displayed near the trigger. If an abbreviation is also supplied, the abbreviation is displayed near the trigger and the condition is placed in the list of conditions. The trigger can also be designated as *constrained* and symbols can be placed at the beginning (source meneme) or the end (target meneme) of the trigger.

Figure 19 shows (a) a trigger that has just been constructed and (b) the same trigger after editing to add a condition, to designate it as constrained, to add a *select symbol* to the source meneme (Meneme 1) and to add a *deselect symbol* to the target meneme (Meneme 2).

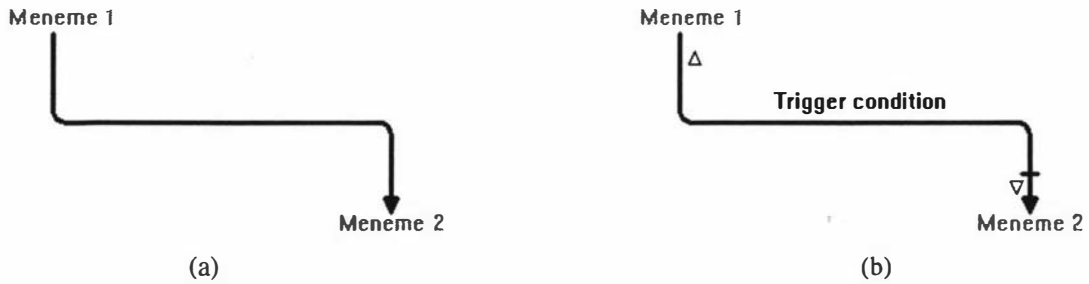


Figure 19: (a) before and (b) after editing a trigger.

Each straight segment of a trigger can be individually moved by "drag and drop" and a trigger will also be automatically adjusted if the first or last meneme is moved.

11.3 Displaying triggers

Toggling the Triggers option in the View menu controls the display of triggers in the diagram. However any hidden triggers remain hidden even if other triggers are currently being displayed. Selecting a trigger and then selecting the Hide option from the SELCU toolbar will hide the trigger. Any, and all, hidden triggers can be redisplayed by selecting Unhide triggers in the View menu. Hidden triggers can also be displayed by selecting the Show All option from the SELCU toolbar. ***Note:*** *the selection of Show All will display all hidden menemes, conditions, triggers and tasks.*

11.4 Deleting a trigger

Selecting a trigger and then pressing the **Delete** key will delete the trigger. If necessary, the deleted trigger can be retrieved by selecting the Undo option from the SELCU toolbar.

Note: *triggers cannot be extensively modified. If the source and target menemes are moved in such a way that the trigger no longer correctly joins them, then the trigger should be deleted and a new trigger constructed to replace it.*

12. Task sequences

12.1 Creating a task sequence

To construct a new task sequence in the diagram, proceed as follows.

1. Select *Construct a new task* from the SELCU toolbar.
2. Select several existing menemes to form the task sequence.

It is possible to include non-existing menemes in the task sequence as it is being created. A click on an empty area will create a new meneme in that area as part of the task sequence. Such menemes can be edited later to give them the desired name, designators, etc.

3. Press the **Escape** key to end the construction of the task sequence.
4. "Drag and drop" the task name to the desired final position.
5. Double-click on the task name, enter the desired name for the task, and select OK.

Note that pressing the **Escape** key (or selecting the *Neutral Mode* option from the SELCU toolbar) will end the construction process and will not abort it. However, an unwanted task sequence can be deleted later.

12.2 Editing a task sequence

The actions comprising the task sequence can now be edited. A task sequence is portrayed as a series of rings (actions) around selected menemes linked by arrows indicating the flow of the task sequence – see Figure 21.

"Drag and drop" can be used to move an arrow. If the arrow is selected by clicking near the beginning of the arrow, the starting point of the arrow will be moved. If the arrow is selected by clicking near the end of the arrow, the ending point of the arrow will be moved. ***Note:*** *clicking near the centre of an arrow could move either end with possibly unexpected results.*

An action cannot be edited by double-clicking on the meneme, as this will edit the meneme and not the action. Thus an action is edited by double-clicking on the arrow *leading to* the action. This will open the Edit Task Arrow dialog box for the action as illustrated in Figure 20.

The action type can be designated as one of: user action (solid arrow and the default), system action (dashed arrow) or double arrow (double-headed arrow indicating a user action in which a number of choices may be made before moving on). Various symbols can be placed next to the end of the arrow. The Dash length can be adjusted for the dashed arrow and the size of the ring around the meneme can be adjusted. The default size is 6 pixels out from the meneme name. The ring can also be adjusted by "drag and drop".

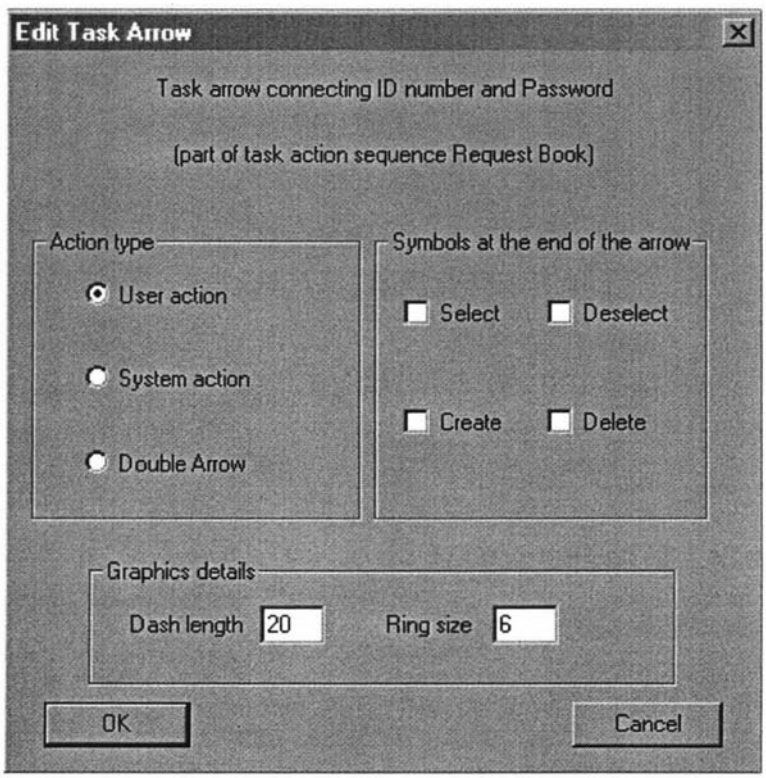


Figure 20: The Edit Task Arrow dialog box

Figure 21 shows (a) a task sequence that has just been constructed and (b) the same task sequence after the name has been changed, the action based on Meneme 3 has been edited to make it a system action (dashed arrow) with the dash length set to 5, the ring around Meneme 3 has been set to 12, and the arrow leading to Meneme 4 has been made double-headed.

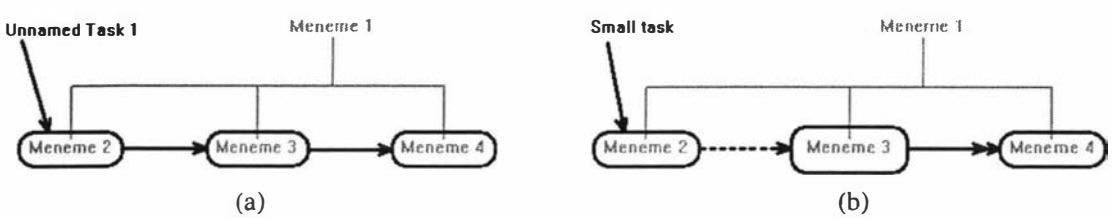


Figure 21: (a) before and (b) after editing actions in a task sequence.

12.3 Displaying task sequences

Task sequences are not usually all displayed simultaneously. Selecting the Task option in the View menu will provide a list of existing task sequences and one of them can be selected for display. The list also provides options to hide all task sequences or to display all task sequences. This list can also be reached by selecting the Task Display option from the SELCU toolbar. Selecting the ALL Tasks option in the View menu will also display all the existing task sequences.

Several, but not all, task sequences can be displayed by displaying all tasks and then hiding unwanted tasks. Selecting a task sequence and then selecting the Hide option from the SELCU toolbar will hide the task. Any, and all, hidden tasks can be redisplayed by selecting the Show All option from the SELCU toolbar. ***Note:*** *the selection of Show All will display all hidden menemes, conditions, triggers and tasks.*

12.4 Modifying a task sequence

An initial task sequence often requires modification and it is common to find that a meneme that was previously not part of the sequence needs to be included or vice versa. This sort of modification is illustrated in Figure 22 by including Meneme 3 into a task sequence

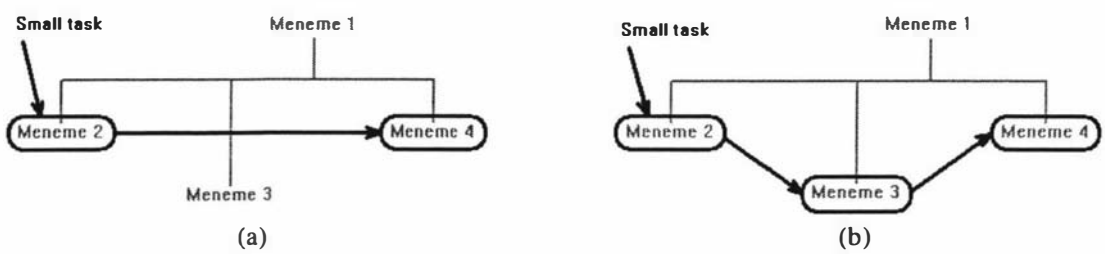


Figure 22: (a) before and (b) after adding another meneme to a task sequence.

The following process was used to change from Figure 22(a) to 22(b):

1. Select the existing task sequence (the task is then highlighted).
2. Select *Modify an existing task* from the SELCU toolbar.
3. Select a meneme that is already in the sequence (Meneme 2).
4. Select the meneme that will become the next in the sequence (Meneme 3).
5. Select another meneme that is already in the sequence (Meneme 4).

The task sequence is thus modified so that it proceeds as before to the meneme selected in step 3, then proceeds to the new meneme, and then continues on to the meneme selected in step 5.

A similar process is used to remove a meneme from a task sequence. For example, to modify the task sequence in Figure 22(b) to that of Figure 22(a), the following steps would be required:

1. Select the existing task sequence (the task is then highlighted).
2. Select *Modify an existing task* from the SELCU toolbar.
3. Select a meneme that is already in the sequence (Meneme 2).
4. Select another meneme that is already in the sequence (Meneme 4).

Meneme 3 is thus deliberately omitted from the task sequence.

It is also possible to select the task name instead of a meneme in step (3). This enables a new meneme to be included as the first meneme in the sequence or alternatively enables the existing first meneme to be excluded from the sequence.

If extensive modification is required it may be preferable to delete the task sequence and then construct a new one. Selecting a task sequence and then pressing the **Delete** key will delete the sequence. If necessary, the deleted task sequence can be retrieved by selecting the Undo option from the SELCU toolbar.

13. Printing Lean Cuisine+ diagrams

To print a Lean Cuisine+ tree diagram, proceed as follows:

1. Select *Print Setup* in the File menu.
2. Set options as appropriate in the Print Setup dialog box. In particular, it may be useful to change the orientation from Portrait to Landscape. Select OK when the Print Setup box is no longer required.
3. Select *Print Origin* from the SELCU toolbar.

4. Use the mouse to place the top-left corner of the printed page (the print origin) at an appropriate point in the work area.
5. Select *Print Preview* in the File menu.
6. A preview of the printed page will be displayed. The preview should be examined to ensure that all desired Lean Cuisine+ elements will be included.

Steps 3 to 6 should be repeated until the user is satisfied that the previewed page is showing the desired display. When this has been achieved, selecting Print in the File menu (or selecting the Print option from the standard toolbar) will print the page.

Important: *SELCU does not automatically print all pages of a large diagram. The system is designed to print only one page at a time and, by arranging the print origin, each page can show any part of the diagram. This means that pages can overlap, if this would be useful, or parts of the diagram can be deliberately omitted.*

14. Colours and fonts in Lean Cuisine+ diagrams

Colour is used extensively in online Lean Cuisine+ diagrams although it is often lost when the diagrams are printed. SELCU uses the following colour defaults:

The currently selected meneme is highlighted with *green.* The currently selected link between menemes is *orange.* Meneme conditions are displayed in *black.* Triggers appear in *red* and the currently selected trigger is *mauve*. Task sequences are in *blue* and the currently selected task is *orange.* When a group is formed, every meneme in the group is highlighted in *dark gray* except the group leader which is highlighted in *red.* The meneme names in the group appear in white. The font for meneme names is initially set to Microsoft system, 10 points.

Colours and fonts can be changed by selecting the *Colours and fonts* option from the SELCU toolbar. A new colour can be selected from the colour palette that appears after selecting the type of object that requires a colour change. Likewise, the font, the font style and the font size can be changed, if desired.

Note: *when printing in black-and-white, changing the colour of triggers, tasks, etc to black produces a crisper contrast. This technique has been used throughout this document.*

Appendix 3. Modelling the SELCU User Interface with Lean Cuisine+

This Appendix shows how the revised Lean Cuisine+ notation can be used to model the dialogue for a more complex graphical user interface. The user interface modelled here is that of the software environment for Lean Cuisine+ (SELCU), which is described in detail in Chapter 6. It should be noted that this is not an extensive case study but rather a large example of a Lean Cuisine+ model that highlights the use of the notation in a typical graphical environment.

The tree diagram

Figure 1 shows the Lean Cuisine+ tree diagram for SELCU and the leading subdialogues are described below:

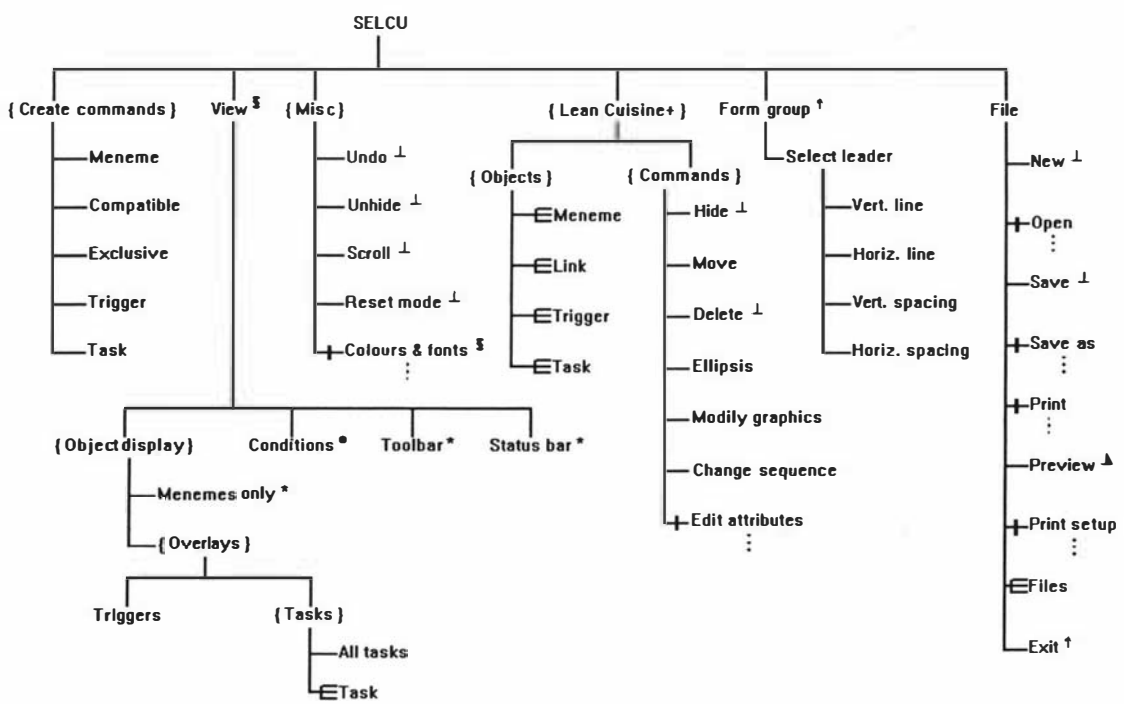


Figure 1: The dialogue tree for the SELCU interface

Create commands: these commands are used to create new Lean Cuisine+ objects. For example, Meneme is selected to create one or more new menemes. Compatible is selected to create a mutually compatible subdialogue, either from existing menemes or by placing new menemes under an existing header meneme. Likewise, Exclusive is selected to create a mutually exclusive subdialogue.

View: these commands are used to change the display. This subdialogue is designated as required choice (§) and when the system starts, the mutually compatible options Menemes only, Conditions, Toolbar and Status bar are all selected. To view triggers, the Triggers option would need to be selected, etc. Note that the display of conditions is mutually compatible with the display of any Lean Cuisine+ objects, but that the display of menemes only is mutually exclusive with the display of triggers or tasks.

Misc: this subdialogue contains the following commands:
Undo – to undo one or more previous deletions
Unhide – to display all currently hidden objects
Scroll – to move the viewing window to see more of a large diagram
Reset mode – to cancel the current mode, for example, to stop inserting menemes
Colours & fonts – to change the colours of objects and change the font of meneme names
(Colours & fonts is a modal subdialogue that is not displayed in this view of the dialogue.)

Lean Cuisine+: this contains two mutually compatible subdialogues – {Objects} and {Commands}. This enables any of the commands to be executed while a specific Lean Cuisine+ object is selected. For example, selecting a Meneme (object) and then selecting Delete (command) would delete that meneme. The menemes in the {Objects} subdialogue are all designated as homogenous groups as any number of those objects may be present.

Form Group: dragging a "frame" around several menemes will form them into a group and a "leader" of the group can then be selected. The group can cross subdialogue boundaries so menemes from several subdialogues can be part of a group. The formation of a group enables several group commands such as Horizontal line - when selected all menemes in the group line up (horizontally) with the group leader.

File: this provides the typical options found in most *File* menus.

Conditions

Certain menemes have conditions restricting their availability for selection. Figure 2 shows some of the conditions displayed on the "greyed" tree diagram of Figure 1. Note that the subdialogues *View* and *File* have been hidden in order to make the diagram less cluttered and to emphasise the display of the conditions. Some menemes have the same conditions, e.g. both Move and Delete require an object or a group to have been selected.

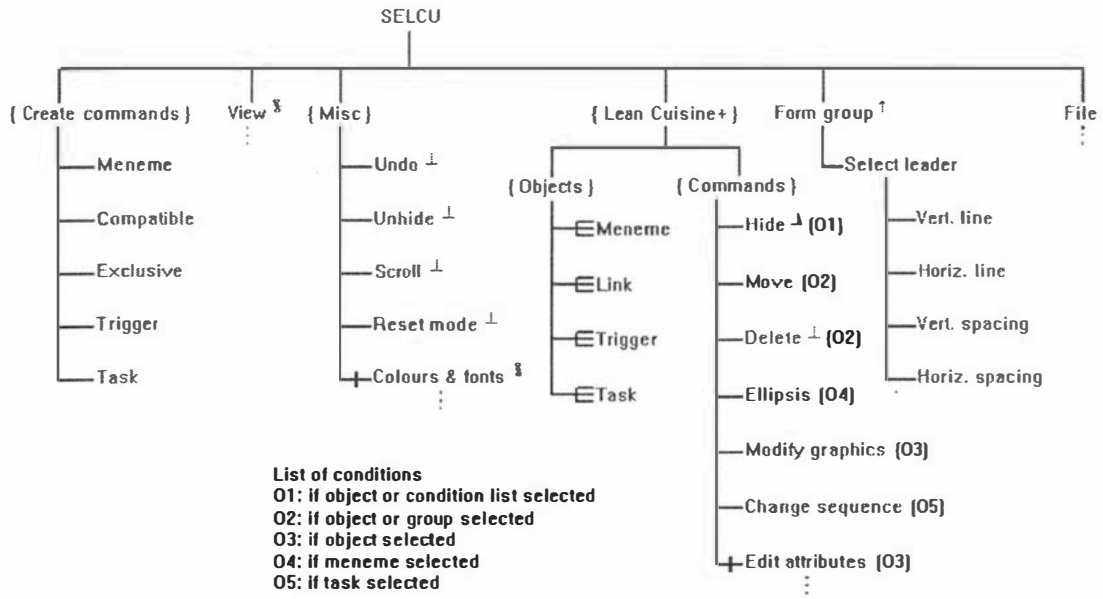


Figure 2: The dialogue tree for the SELCU interface showing conditions

Task sequences

Figure 3 shows two task sequences overlayed on the SELCU tree diagram.

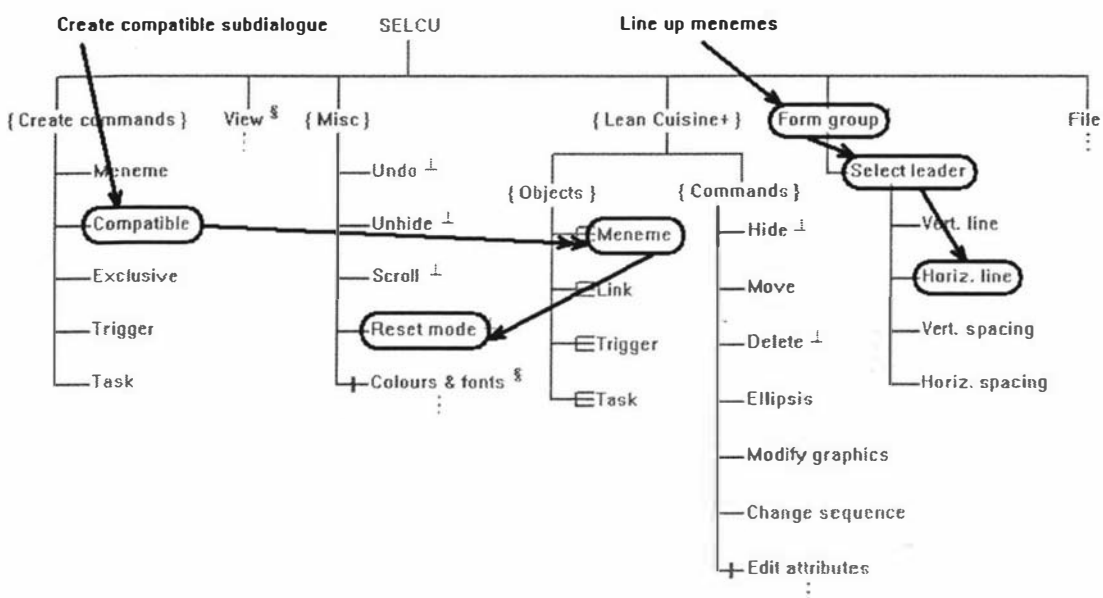


Figure 3: The dialogue tree for SELCU showing two task sequences

The *View* and *File* subdialogues remain hidden in this view. The *Create compatible subdialogue* task commences with the selection of the Compatible option in the *Create*

commands subdialogue to put the dialogue into the "create compatible subdialogue" mode. Then a number of Menemes must be selected – the double-headed arrow indicates the selection of several objects from the homogeneous group. The fork symbol indicates a mutually exclusive homogenous group so the menemes are selected sequentially. The first selected meneme forms the header and the following menemes make up the body of the new subdialogue. On completion, the user selects Reset mode.

The *Line up menemes* task provides for the arrangement of selected menemes into a straight (horizontal) line. In order to do this, the user must first form these menemes into a group. Then select one meneme as the leader of the group and then select the Horizontal line option. This process was used in Figure 3 to form the menemes from {Create commands} to File into a straight line.