Lean Cuisine: a low-fat notation for menus

M.D. Apperley and R. Spence*

The specification, design, implementation and control of highly interactive direct manipulation dialogues is of increasing interest. However, existing techniques fall well short of the goal of isolating the design of the dialogue from the detail of its implementation. This paper closely analyses the structural characteristics of menu systems, a major component of such dialogues, and arising from this analysis proposes a new diagrammatic approach to their description. This approach is shown to be able to completely specify the details and behaviour of a system of menus from an external point of view. The parallels between this notation and the recently defined class of automata, Event-Response Systems, are discussed, demonstrating the potential for a direct implementation of an interface from this description. Further, it is suggested that the notation could be extended to cover all aspects of direct manipulation interaction.

Keywords: dialogue design, design notation, menu dialogues, menu syntax, dialogue control

A menu is essentially a list from which choices can be made. The term originated in the context of restaurants and eating establishments, where it has two distinct but closely related meanings; a list (explicit or implied) of dishes available, from which the diner may make a choice, or a list of the dishes that constitute the meal to be presented, without necessarily any choice.

In the context of human–computer dialogues, the term 'menu' has been adopted to describe a system in which the user is presented with a number of options, from which a choice can be made. With the proliferation of high resolution graphical workstations and the sophisticated software they support, menus are becoming an increasingly common form of dialogue. In fact, it could be argued that the menu is fundamental to achieving the direct manipulation interfaces (Hutchins *et al.*, 1986) which these systems are moving towards. Over the past ten years, the use of menus in the computer context has been the subject of a number of studies and expositions: the best number of choices per

Department of Computer Science, Massey University, Palmerston North, New Zealand. Tel: 64 63 69 099

* Department of Electrical Engineering, Imperial College of Science, Technology and Medicine, Exhibition Road, London SW7 2BT, UK. Tel: 01-589 5111

0953-5438/89/010045-24 $03.00

menu (Miller, 1981); the layout of these choices (Perlman, 1984); the organisation of hierarchies of menus (Norman and Chin, 1988), and the navigation through such structures (Apperley and Spence, 1983); the classification of information in hierarchical systems (Giroux and Belleau, 1986); and the use of icons rather than words (Hemenway, 1982), to mention just some of this work.

The more general problems of dialogue specification, implementation and control have received considerable attention in the past few years. At one extreme there are comprehensive user interface management systems (Kilgour, 1987; MacLean, 1987; Palmer, 1987; Pfaff, 1985) which attempt to solve all of these in a single package and, at the other, there are notations and methodologies to simplify and improve the task of implementation (Borufka *et al.*, 1982; Cardelli and Pike, 1985; Lieberman, 1985). A number of specification techniques for the dialogue designer have also been proposed (Brown, 1982; Browne *et al.*, 1986; Hill, 1987; Jacob, 1985; Kasik, 1976; Kieras and Polson, 1983; Wasserman, 1985). Although many of these schemes provide extensive tool-kits and primitives specifically to support the direct manipulation style of interaction, these highly interactive interfaces remain by no means easy to implement, or even to specify. It is not that the supplied libraries are inadequate in themselves, but that there is an enormous gap between these primitives and the concepts that are fundamental to the direct manipulation paradigm.

It has been suggested (Lieberman, 1985) that '. . . construction of these interfaces is still a black art . . . (and) with today's software often requires much effort, is error-prone, and much work must be repeated for each application'. In fact, in the majority of cases it is considerably easier to produce a poor interface than it is to produce a good one. These facts, coupled with the additional problems that the technical complexity of high-performance bit-mapped display systems can divert the implementors' attention from the dialogue itself (MacLean, 1987), and that in general, the user interface receives too little attention in the design process (Wasserman, 1985), or is treated as being of secondary importance to efficiency (Borufka *et al.*, 1982), make the effective design and implementation of good direct manipulation interfaces an awesome task at the present time.

Fundamental to this problem is the lack of progress towards an appropriate dialogue specification technique, one which caters for the complexity of the direct manipulation environment, which truly isolates the dialogue from the detail of its implementation, and which is a tool for the dialogue designer rather than the programmer. Jacob (1985), in advocating a diagrammatic technique for dialogue specification, has neatly summarised the requirements:

> 'A visual representation chosen for this purpose needs to describe the external (user-visible) behaviour of the user interface of a system precisely, leaving no doubt as to the behaviour of the system for each possible input. It should separate function from implementation, describing the behaviour of the user interface completely, and precisely, without unduly constraining the way it will be implemented. The visual representation should be easier to understand, and take less effort to produce than the more conventional symbolic software. Ideally, the overall structure of the visual program should represent the cognitive structure of the user interface. The program should describe the constructs a user will keep in mind

when learning about the system — the basic outline around which the user's mental model of the system will be built. Finally, the visual representation must be directly executable in a visual programming environment.'

In the next section of this paper, the structure of computer menus is closely examined and, from this analysis, the requirements of a menu specification notation are established. Existing descriptive techniques are reviewed in the third section, and it is shown that these are inadequate as design and implementation tools for menu systems. From the analysis of the structural characteristics of menus, a pictorial notation for their representation is then developed in the fourth section. This notation is intended specifically for the dialogue designer, who may have little or no programming knowledge. However, because it is based on a familiar tree form, it is shown in section five that the step to direct execution of the notation is not great. Finally it is further suggested that the methodology may not just be useful for the specification of menu dialogues, but could be extended to cover other aspects of the direct manipulation environment.

Structural characteristics of menus

Computer display menus appear in many forms; pull-down, pop-up, static, dynamic, peel-and-stick, ordered, dispersed, horizontal, vertical, textual, iconic, explicit, implicit, to name but a few. In this paper, it is not the presentation of menus that is of immediate concern, but their structure, in terms of the dialogue facilities they provide. Menus are sometimes used to initiate actions, sometimes to change the state of a system, sometimes to choose between alternative parameters, and sometimes to select other menus from which further choices can be made. It is interesting to note that in the computer-based context, the notion of a menu is used in both of the gastronomical senses; a menu is used to convey the range of choice, and also to convey what it is that will be provided. To cover this diversity of presentation and function, and in the anticipation of providing extensibility to other facets of human–machine dialogue, the following definition of a menu is proposed:

'A menu is a set of selectable representations of actions, parameters, objects (which may be other menus), states, and other attributes.'

The term selectable representations conveys the concept of choice in the definition, but without introducing any constraints as to presentation (of the menu) or selection mechanisms, or any assumptions about the use of icons or words. Thus the definition could describe such forms as car dashboards and hi-fi control panels, as well as the familiar direct manipulation and keyboard-controlled computer menu systems.

Although it is true that menus are often just a simple list from which a single choice is to be made, in many cases more than one choice can or must be made, and there may be complex interrelationships between these choices. Consider the (gastronomical) menus shown in Figure 1. The 'Greasy Spoon' (Figure 1(a))

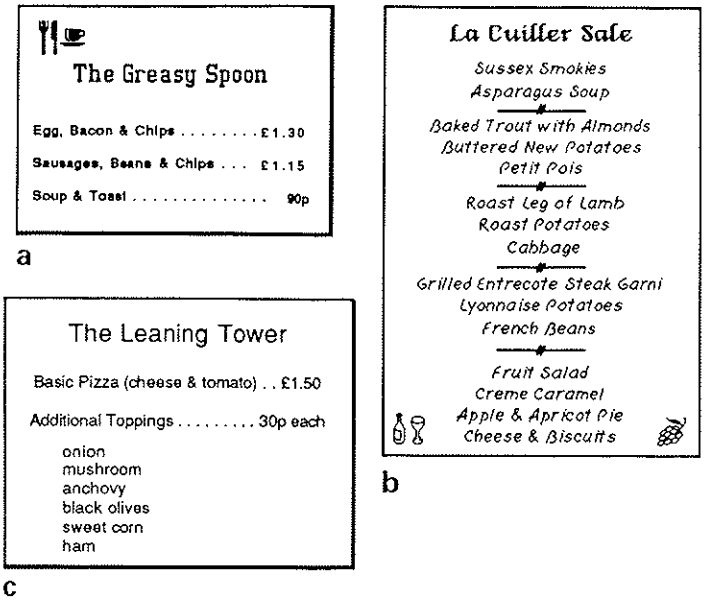

*Figure 1. Example restaurant menus showing (a) and (b) mutually exclusive groups or 1-from-*N *choices, and (c) a mutually compatible group or* M*-from-*N *choice*

offers a simple menu from which the diner is required to make just a single selection (which in fact defines the entire meal). On the other hand, 'La Cuiller Sale' (Figure 1(b)) has a menu which is considerably more complex. The discerning diner will be expected to make a selection of one from the two starters, one from the three main dishes, and another from the four desserts. It could be said that for the Greasy Spoon's menu, all items are mutually exclusive, requiring a 1-from-*N* choice, while for La Cuiller Sale, the menu comprises three mutually exclusive groups, each requiring a choice to define a complete meal (but of course some diners may skip the starter or the dessert). It should also be noted that the layout (presentation) of these menus conveys very little about their structural similarities or differences.

A further variation in the internal structure of a restaurant menu can be seen in Figure 1(c). Here the diner can order a basic pizza, with as many additional toppings as are required. In other words, any number (from 0 to *N*) of the *N* items in the topping section can be chosen. This can be described as a mutually compatible group, or an *M*-from-*N* choice. (These two basic menu structures, 1-from-*N* and *M*-from-*N*, were not the only ones identified by Brown (1982). However, certainly in the direct manipulation context, all menus can be described in terms of these two structures.)

There are other subtle constraints in the menus shown. It would seem likely that one could not choose a starter at La Cuiller Sale without also selecting a main course. Similarly, in the menu of Figure 1(c), choosing any number of toppings without choosing a basic pizza would not make sense. These constraints, together with the classifications of mutual exclusivity and mutual compatibility, can be described as the grammar or syntax of the menus. It might

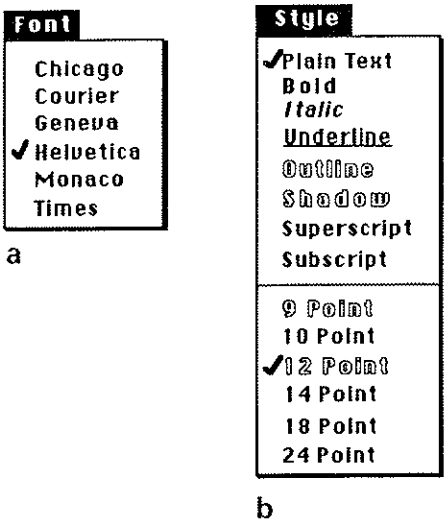

Figure 2. Two example menus taken from Macintosh application MacWrite

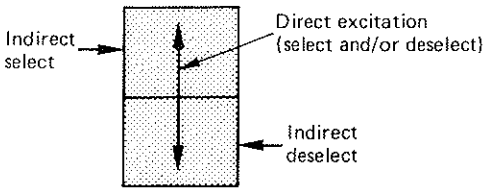

Figure 3. Meneme model based on behaviour of a flip-flop

be expected that the diner would be aware (perhaps subconsciously) of this grammar, but it is also likely that the waiter would ensure that the meal chosen was syntactically (if not gastronomically) correct.

That such constraints are also present in computer-based menus can be seen in the two examples of Figure 2. These are two menus taken from the Macintosh application MacWrite.

In Figure 2(a), the user is required to make just a single choice from the Font menu. This menu clearly constitutes a mutually exclusive group. In Figure 2(b), the Style and Size menu is more complex. In the upper Style section, the Plain Text item is mutually exclusive with respect to each of the other items, but Bold, Italic, Underline, Outline and Shadow form a mutually compatible set; any number can be chosen from this group. The remaining two items in this section, Superscript and Subscript, form another mutually exclusive group; only one of these two can be chosen, but that choice is compatible with the Bold group. The Size section (the lower part of the menu) is a simple mutually exclusive group from which just a single item can be chosen.

The structure of these menus is further complicated by the requirement that a valid choice always be present in the Font menu, and in each of the two sections of the Style and Size menu. Together, these constraints do result in some less than obvious behaviour. For example, although items in the Bold group can be selected and deselected at will, Plain Text cannot be deselected directly, as there is no unique alternative to satisfy the requirement for a choice of Style.

In other words, what appears to be a relatively simple menu (the Style and Size menu) can have a quite complex structure of interrelationships and constraints. Users of MacWrite, however, are shielded from these complexities. The grammatical rules have been built into the menus themselves (in this case they have been programmed as a part of the application); valid selections are always

present initially by default, and it is impossible to make an invalid or grammatically incorrect modification. In this sense, the concept of 'syntax' has little meaning from the user's point of view. Selecting a second font in Figure 2(a), for example, will automatically cancel the selection of the first. Selecting Bold from the Style menu (Figure 2(b)) will automatically deselect Plain Text, but subsequently selecting Italic will leave both Bold and Italic selected, as these two are mutually compatible. If at some later stage Plain Text is reselected, then both Bold and Italic will automatically be deselected. On the other hand, if only Bold is currently selected, and it is deselected by selecting it a second time*, then Plain Text will be automatically selected as the default alternative. Just as the waiter would ensure that a chosen meal was syntactically correct, so in this case the menu handler ensures that the options chosen from the menu conform to the prescribed rules. Thus the menu syntax is a complexity for the dialogue designer, and possibly the implementor, to deal with and to specify, rather than a problem for the user.

The earlier definition of a menu can now be expanded to cover these interrelationships and constraints, and to adequately describe the menus of Figures 1 and 2:

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

The components of a menu, the selectable representations of the preceding definition, have commonly been referred to as items (Shneiderman, 1987), options or labels (Field, 1988). These representations can take many forms. In traditional (gastronomic) menus they may be single words or short phrases, but in some cases may include a paragraph describing the dish. In the computer context, in addition to similar textual variations, increasingly menus may comprise a collection of graphical symbols or icons. In either case there is the further possibility of several 'options' being incorporated within a single 'item'. For example, the Greasy Spoon menu (Figure 1) might contain the entry, 'Pie with beans or chips'. This is not just a single option (selectable representation), but should be considered as either two (Pie with beans, Pie with chips) or three (pie, beans, chips), with appropriate constraints. To address the inadequacy of the word item, and to avoid the connotations of the words option and label, a new term meneme is introduced:

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

In this way, the single item 'Pie with beans or chips' can be regarded as comprising two or three menemes.

It follows from the above definitions, that for each meneme there are just two

* This bistable behaviour in menu selection is discussed later.

possible states, selected and not selected*, and that menemes can be described as bistable (Apperley and Spence, 1983). In other words, the state of a meneme can be considered to be a binary variable. Without considering the detail of the selection mechanism, it is possible to draw from the earlier discussion that there are two distinct mechanisms by which a meneme may change its state; direct excitation involving specific reference to the meneme itself† ('I'll have the trout' or 'I've changed my mind, I won't have any soup'), or indirect modification, where because of a constraint or interrelationship, direct excitation of one meneme causes modification of another. (For example, selecting a second Font in the menu of Figure 2(a) automatically cancels the first choice.)

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

In fact, a meneme behaves in a manner rather similar to that of a generalised (JK) flip-flop (Tocci, 1980), and a model based on this device (see Figure 3) does provide for the description of complex interrelationships in menus, such as those just discussed.

Of the four possible modifications that may take place (direct select, direct deselect, indirect select, and indirect deselect), not all are necessarily present for any given meneme. For example, the Plain Text meneme of Figure 2(b) can be directly selected, but cannot be directly deselected. It will be indirectly selected if all other menemes in the Style section have been deselected, and it will be indirectly deselected by the selection of any of the other menemes in that section. Thus a collection of meneme models such as those shown in Figure 3, with appropriately labelled indirect excitations, could be used to provide a specification of the interrelationships, and thus a full description of the behaviour, of a complex menu. Figure 4 shows an example of such a description for a reduced version of the Style menu of Figure 2(b), containing just the five menemes Plain Text, Bold, Italic, Superscript and Subscript, but exhibiting the same general characteristics as the full menu earlier described.

A characteristic of the menu systems discussed, for example the Style menu of Figure 2(b), is that they generally allow the user to select a number of different menemes from a single presentation or menu. The selected menemes might be considered to constitute the information being passed from the user to the system. However, there is no specific sequence associated with the selections; it is not the order in which menemes have been selected that conveys the information, but rather the total state represented by the eventual selections. In other words, the dialogue could be described as asynchronous, occurring in no particular order or sequence. The term asynchronous is used here in preference to multi-threaded or concurrent (Hill, 1987). It is maintained that in a direct

---

* Two other apparent states may commonly be observed; disabled, and thus not available for selection (which is not really a meneme at all), and temporarily highlighted to show that the meneme will be selected if the appropriate action is taken. In both cases these 'states' are part of the presentation detail, and of little consequence to the application.

† Often the same direct excitation (for example, a mouse click) which causes a meneme to become selected, will also cause it to be deselected if it is already in the selected state.

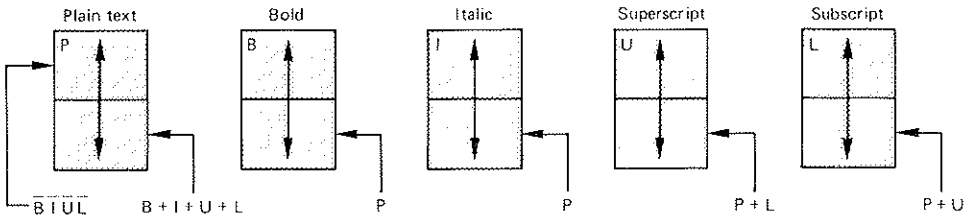

Figure 4. Meneme-based description of behaviour of simplified Style menu, using flip-flop meneme model

manipulation environment, where the basic dialogue unit (sentence) is generally a single action, this term more correctly describes the absence of sequence. In fact, this asynchronous style is a general characteristic of all aspects of direct manipulation dialogues.

In the restaurant examples, the diner may specify the main meal first, followed by the starter. Prompts from the waiter may then elicit the type of salad dressing required, the choice of vegetables, and how the steak is to be done. Dessert and coffee may be ordered at a subsequent interaction. Orders for different parts of the meal from different diners at the same table may be interspersed. Clearly, no significance can be attached to the meneme sequence. The final total state, defined by the individual meneme states and inscribed on the waiter's pad, defines the meal. Perhaps a more graphic example of the asynchronous nature of a menu dialogue can be seen in many fast-food restaurants. Here there is an active menu in the form of a concept keyboard operated by the waiter. This device has buttons (menemes) representing individual components of the meal. These can be selected or deselected by the waiter in any sequence until the final displayed state is to the satisfaction of the diner. During the interaction, the state may not only be incomplete, but it may also be incorrect or at least misleading, as a choice yet to be made may cancel or modify an earlier one.

In the computer context, a variety of sequences that achieve the same end result is typical of a direct manipulation environment. A MacDraw user might define a line having initially specified the line style. Alternatively, the line style may be specified during the definition of the line (perhaps the start point has been fixed, but not yet the end point), or after the line itself has been completely defined (both start and end points fixed). It is even possible to subsequently change the line style in an entirely separate interaction.

It has been said of direct manipulation interfaces,

> 'There are no hidden operations, no syntax or command names to learn. What you see is what you get. Some classes of syntax errors are eliminated. For example, you can't point to a nonexistent object. The system requires expertise in the task domain, but only minimal knowledge of the computer or computing' (Hutchins *et al.*, 1986).

It is now contended that, in fact, there is nothing that one can do with a well-defined menu interface that could be construed as syntactically incorrect. In

other words, all classes of syntax errors are eliminated. If a file has not been named before it is saved, and as a result another is inadvertently overwritten, then this is not a syntax error but a semantic one, brought about by a mismatch between the user's model of the interface, and the designer's model. Further, the consequence of such semantic errors can be reduced, if not completely eliminated, by more careful design of the system. It is suggested that if there is a syntax of menus, then it is the structural constraints that have been discussed in this section, and that these constraints will generally be built into the menus themselves.

Existing notations for interactive dialogue description

Dialogue description techniques have been the subject of two recent comprehensive surveys (Cockton, 1985; Green, 1986). Green (1986) makes a distinction between design notations (informal) and implementation notations (formal), and concentrates on the latter. Existing techniques are then grouped into three classes; transition networks, formal grammars, and event models. Following a formal analysis of these three models, Green (1986) shows that the event model has the greatest descriptive power. However, this paper is concerned with techniques for the description and implementation of menus, and in this section, the three classes identified above are closely examined in this specific regard.

Transition networks

Transition networks in various guises (finite state machines, finite state automata, or state transition diagrams) have been extensively used as a dialogue description technique, by both designers and implementors, and are probably the most popular form of dialogue description currently in use. Although the majority of detailed examples and implementations are based on keyboard interaction (Jacob, 1983; Jacob, 1985; Kieras and Polson, 1983; Wasserman, 1985), transition diagrams also have a long history in the description of more interactive graphical dialogues (Foley and Wallace, 1974; Koivunen and Mantyla, 1988; Newman, 1968), and their use as a specification language for direct manipulation interfaces has been proposed (Jacob, 1986).

Transition network diagrams, which essentially describe the behaviour of finite state automata, are based on a set of nodes (states), and the arcs or links between them (state transitions). In their basic language processing form, each arc is associated with a token in the input language. As such, a transition diagram defines the legal or permissible sequences of input tokens. In describing interactive dialogues, several enhancements have been made to this basic scheme

- to allow program actions to be attached to arcs and states
- to provide hierarchical representation of more complex systems using subdiagrams
- to provide for recursive calls to these subdiagrams
- to encode some potential states within internal registers (Green, 1986)

These enhancements, in various combinations, have enabled transition diagrams to be used not just as an informal design tool, but as a means of directly implementing the user interface.

Although it is true that 'One of the principal virtues of the state diagram notation is that . . . they make more explicit what the user can do at each point in the dialogue, and what the effect will be' (Jacob, 1985), this notation has obvious shortcomings when attempting to describe a menu containing a number of interrelated subgroups, such as the Style menu of Figures 2 and 4. Because of the absence of sequence in the use of such a menu, there is no obvious start point or end point in the dialogue. In Figure 5, the behaviour of the reduced five meneme Style menu is described using a transition network. Each of the possible menu states (there are twelve in all) is represented by a state in the diagram. Each of the possible input actions in each state (there are five for each of the twelve states, corresponding to the direct excitation of each meneme) is represented by an arc. States have been labelled using upper-case letters representing the selected menemes in each state, with U representing superscript, and L representing subscript. The corresponding lower-case letters are used to represent the direct excitation of each of these menemes. The diagram has been simplified by using bidirectional arcs where possible.

No explicit system responses have been shown in this diagram, because the nature of this particular menu is such that the change of state is the only response that is required. Although the diagram does completely and accurately describe the behaviour of this particular menu, it is neither simple to produce nor to understand, and its derivation from the known external behaviour of the menu is not obvious.

An alternative description is possible, based on the Generalised Transition Network of Kieras and Polson (1983). In this description (see Figure 6), the menu state information is not expressed in the diagram at all. Rather, this information has been included as a set of global variables, one corresponding to each meneme, and the only true state in the diagram is that of 'modify the Style selection'. There is an arc corresponding to each of the input tokens, in this case the direct excitation of each of the menemes, and a system response is listed for each of these. Input tokens and transition conditions are written above the arcs, and system actions are written below. Arcs emanating from the current 'state' are processed in a clockwise sequence from the top, until a successful transition is made. An arc with no token or condition will always be traversed. Some actions are represented by subdiagrams, in which case the name of the subdiagram appears in parentheses as a response on the relevant arc. Upper and lower-case letters are used to represent selected menemes and their direct excitation respectively, as for Figure 5, and S is the set of currently selected menemes.

This is a more satisfactory transition network description of the Style menu in that it clearly spells out the permissible user actions for the menu as a whole, and the effect that each of these actions will have on the 'state variable set' S. However, the subdiagram states are artificial from the user's point of view, and the description is based more on events rather than states. Consequently, this description should be considered to be a graphical representation of an event

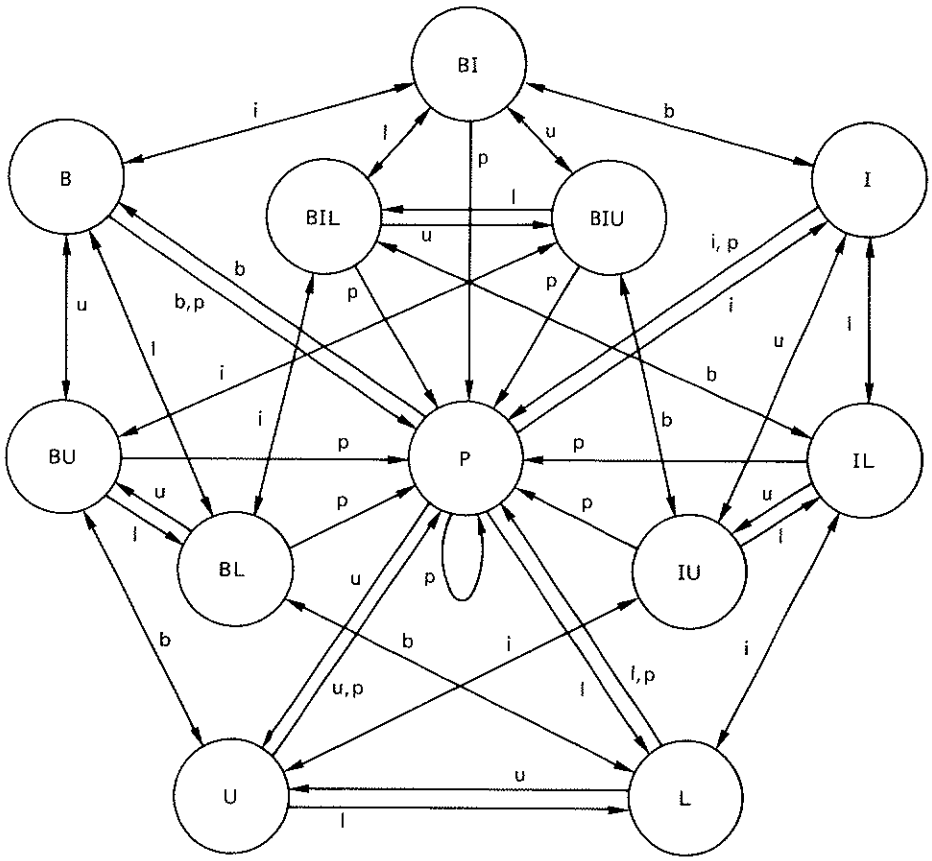

Figure 5. Transition network description of Style menu

model (see the 'event models' section) rather than a true transition network.

Formal Grammars

Formal grammars are a technique developed to precisely describe natural languages, and for some time have been a standard for describing the syntax of programming languages (to a compiler). They have also been used to describe interactive dialogues (Jacob, 1983; Reisner, 1981; Reisner, 1982; Shneiderman, 1982), and form the basis for dialogue description in some UIMSs (Palmer, 1987). A fundamental problem that is encountered in using this technique to describe human–computer interaction is that generally two distinct languages are involved; one language is used to enter commands to the program, and the computer uses another to generate its responses (Green, 1986). Essentially, as with transition networks, the basic technique describes only the syntax of the language, and various enhancements are needed to define the responses generated by the computer.

The form most familiar to computer scientists is the Backus–Naur form (BNF) which is based on production rules. A language is described as a set of rules

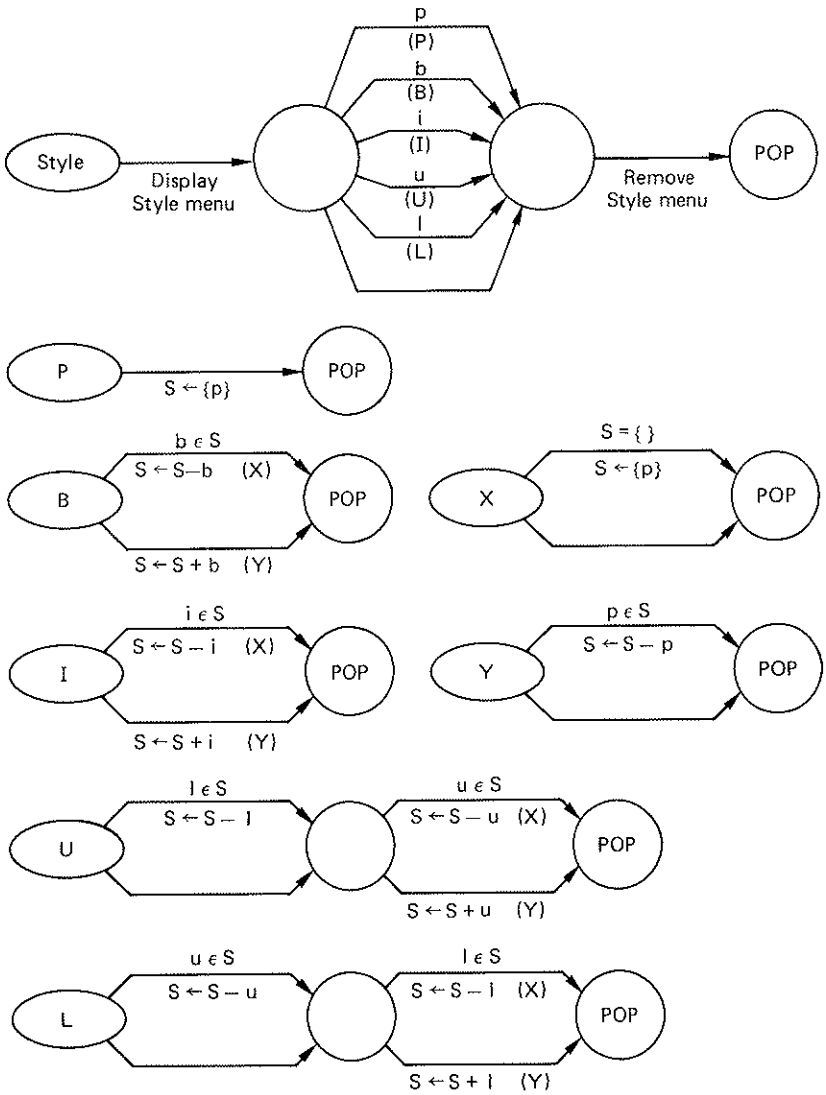

Figure 6. Description of Style menu using Generalised Transition Network

that define the syntactically correct sequences or sentences in the language. The description is hierarchical, with rules defining sequences of terminal symbols (the words of the language) produced by nonterminals (invented higher-level constructs such as phrases), and sequences of nonterminals produced by higher-level nonterminals (for example, sentences). In the case of interactive graphical dialogues, the terminal symbols are the basic actions the user has to learn and remember, for example the movement of a mouse or the double click of a mouse button; the nonterminals correspond to sets of actions that can be grouped together and described as a single operation, for example the

Macintosh sequence to open a file, which involves the concatenation of the two terminals mentioned.

Because of the emphasis on permissible sequences, this technique does not adapt well to direct manipulation dialogues, in particular to menus containing interrelated subgroups such as those described in Figures 2 and 4. At one extreme, there can be a plethora of rules, listing every possible input sequence. This case corresponds closely to the transition network of Figure 5. History, or 'current state' is conveyed by nonterminals, leading to some 60 rules for the reduced five meneme Style menu. At the other extreme, the technique can be used to indicate just the permissible user actions associated with the Style menu, requiring only a single rule. This would correspond to the top-level transition network of Figure 6, and would convey no information at all about the behaviour of the menu. Consequently, comprehensive program code would need to be attached to this rule to encode this behaviour, to a large extent negating any benefit to be derived from using the notation.

Although other production-rule descriptions of the Style menu are certainly possible, in general they will all suffer from the intrinsically sequential bias of the description technique. Either rules for all possible input sequences must be listed, an anathema in a direct manipulation environment, and with the inherent problem of including meaningless interactions (Kieras and Polson, 1983), or alternative or conventional programming notations need to be introduced to provide much of the description.

Event Models

Event models, a more recent development, are a description technique centred on events rather than states. In many graphics packages, interaction is based on queued input events, events generated by physical or logical input devices. Event models take this notion further, with the dialogue control program and the application also being possible sources of events, and the programmer able to define new event types, as appropriate (Green, 1986). Such models have been used for the dialogue specification and control in a number of UIMSs (Flecchia and Bergeron, 1987; Green, 1985; Hill, 1986).

Hill (1987) has developed a formal event-response system model, and from this, a dialogue specification language. An event-response system (ERS) is a new class of automata comprising a set of flags F, a set of events Σ, and a set of rules. A rule comprises an event, a condition, and an action, such as

$$\sigma: F_1 \rightarrow F_2$$

which specifies that if a certain event σ occurs when a certain subset F_1 of the flags are all set, then the result is that the flags F_1 are cleared, and the flags in the subset F_2 will be set. In addition to these regular rules, there can be ε-rules in which the event component is omitted. The execution of an ERS involves a cycle which first processes the ε-rules until there are no longer any true conditions, then waits for an event, and simultaneously processes all of the regular rules for which this is the event and for which the condition components are true. This cycle is repeated continuously.

The ERS model adapts well to the representation of the behaviour of a menu. The direct excitations of the menemes become the events, and the states of the menemes become the flags. In Figure 7, three menus are described in this way, with lower-case letters denoting events, and upper-case denoting flags. Figure 7(a) is the ERS description of a simple three-meneme mutually exclusive group, with the left-hand group of rules showing how the selection of one meneme by direct excitation causes the deselection of the others. Figure 7(b) is the ERS description of a simple mutually compatible group, showing no interaction between menemes. Finally, in Figure 7(c), the ERS description of the more complex Style menu can be seen to bear a very close resemblance to the meneme-based description of Figure 4. It should also be noted that this example uses an ε-rule, corresponding to the indirect selection of the Plain Text meneme.

Although it lacks the visual impact of a diagrammatic technique, the ERS notation has obvious advantages over transition diagrams in conveying the external behaviour of a menu. With its event bias, it clearly shows the effect of each possible user action with a minimum of context; by showing the incremental effect on the individual flags, it avoids the enumeration of all possible states of the menu.

Lean Cuisine: a notation for describing menu syntax

In the preceding section several existing dialogue specification techniques were examined in light of the requirements of menu systems. Although the event model was found to be more straightforward in this respect, it could be said of all of these techniques that they are more suited to the implementor than the designer; they are better for describing how to achieve a particular behaviour rather than describing what that behaviour actually is.

Following the analysis of the characteristics of menu systems, and this critique of current interface design tools, a diagrammatic technique for specifying and describing menu dialogues is now proposed. This technique, referred to as 'Lean Cuisine', is consistent with the notions that:

- a design methodology is required for the dialogue designer
- the methodology should in itself provide a good interface to the designer (MacLean, 1987)
- it should reflect the structure of the dialogue rather than the underlying program
- it be amenable to the direct manipulation environment.

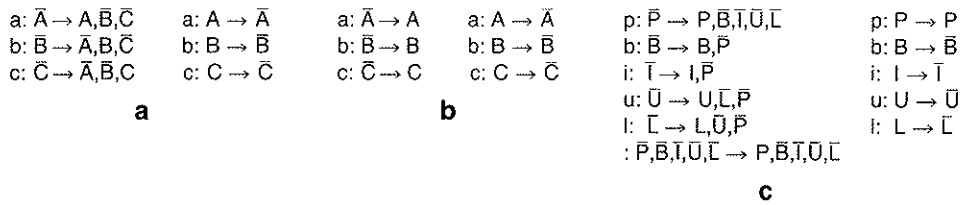

Figure 7. ERS descriptions of (a) mutually exclusive meneme group, (b) mutually compatible group, and (c) five-meneme Style menu

The LC notation is based on a tree diagram, and shows the relationships between menemes, which are the nodes of the tree. For example, Figure 8(a) describes a simple menu called A, which offers a choice between B, C and D. A possible presentation form for this menu is shown in Figure 8(c). The two basic menu structures identified in Section 2 are represented by two different forms of branch. The vertical fork shown in Figure 8(a) is used to describe a mutually compatible group, or an *M*-from-*N* choice. In other words, the menu A offers a choice of 0, 1, 2 or 3 menemes from B, C and D. A mutually exclusive group, or 1-from-*N* choice, is represented by a horizontal fork, as shown in Figure 8(b). This diagram also describes a menu called A, but one which offers a mutually exclusive choice between the three menemes B, C and D. The presentation shown in Figure 8(c) could represent either of these two menus. Based on these two fundamental structures, more complex menus can then be defined.

In the two examples given, A is the menu header or name, but it is treated as if a meneme. In general, menemes may be terminal (leaf) menemes, with no emanating branches (such as B, C, and D, in Figure 8), or they may be nonterminal menemes (such as A), in which case they are themselves headers to other menus. In this way, hierarchies of menus can be described. For example, Figure 9(a) represents a system of three menus. The first, A, offers a mutually exclusive choice between two submenus, B and C. Of these, B offers a mutually exclusive choice of three items, and C a mutually compatible choice of four. Selecting B from A would give access to the submenu B. Figure 9(b) shows a possible presentation of the menu A as a horizontal menu bar, and Figures 9(c) and 9(d) show the menus B and C presented as pull-down menus from this menu bar.

From Figure 9, it can be seen that the menu corresponding to a nonterminal meneme consists of all of its immediate subordinate menemes in the LC diagram. If there is an order associated with the menemes (for example, as they are to be presented in a list), it is as scanned from left-to-right and from top-to-bottom in this diagram.

The LC technique so far described does not provide for more complex menus that involve composite structures of both branch types. The structure of Figure 9(a) could be represented only through the use of submenus. However, more complex structures within a single menu can be accommodated by introducing

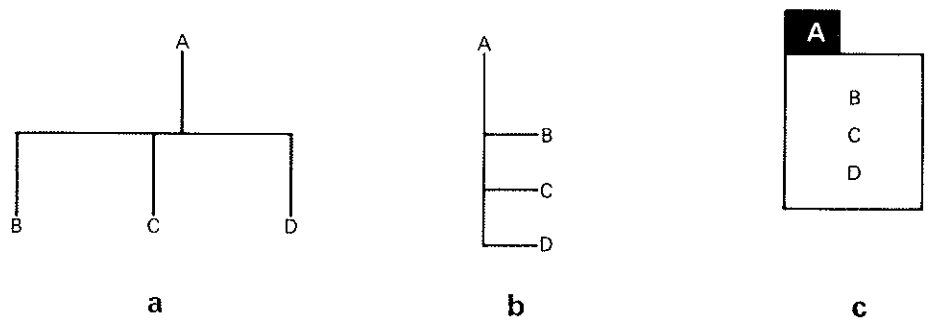

Figure 8. (a) LC diagram for simple M-*from*-N *menu, (b) that for a 1-from*-N *menu, and (c) possible presentation for either menu*

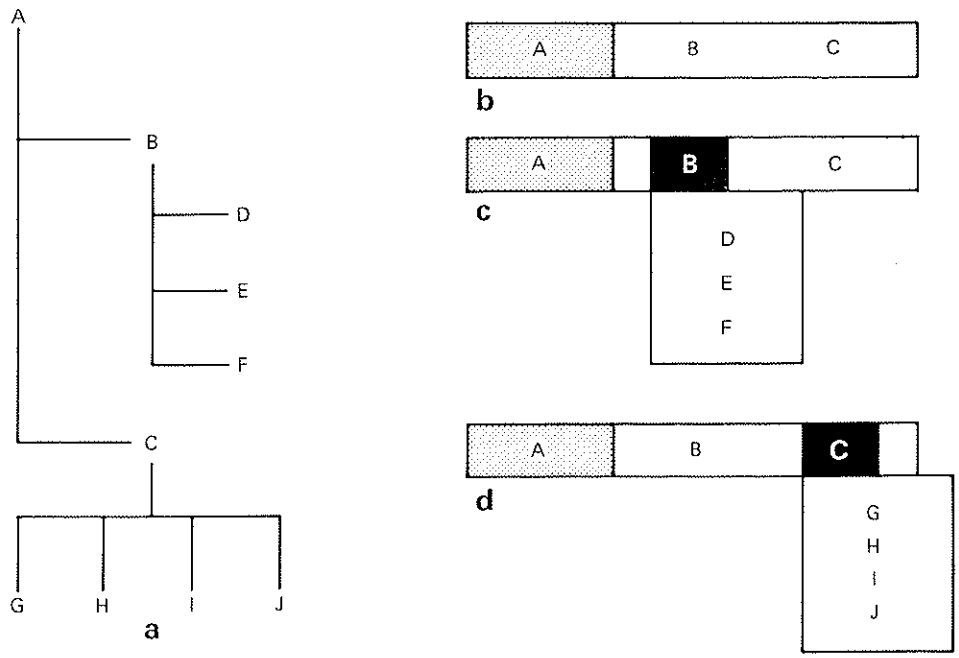

Figure 9. (a) System of three menus, and (b), (c) and (d) their possible presentation

the concept of a virtual nonterminal meneme, which is present as a node in the LC diagram, but does not appear as the header of a menu. Whereas real nonterminal menemes refer to and invoke further menus, virtual menemes partition subgroups of basic structures within a menu. The menu (or submenu) corresponding to a nonterminal meneme can now be defined as all of the immediately subordinate real menemes, together with any submenus defined by immediately subordinate virtual menemes.

In Figure 10, the Style section of the menu pictured in Figure 2(b) is described using this notation. Because of the three distinct groups present in this menu, two virtual menemes have been introduced. In the first branch, Style offers a mutually exclusive choice between Plain Text and the virtual meneme {Fancy Text}. (To distinguish between real and virtual menemes, the latter are enclosed in braces.) This virtual meneme in turn represents the mutually compatible choice of Bold, Italic, Underline, Outline, Shadow and {Indexed}, the latter another virtual meneme offering a mutually exclusive choice between Superscript and Subscript.

The notation also provides for the specification of initial default values and the requirement for a valid selection always to be present in certain subgroups. Default choices are indicated by an asterisk (*) alongside any terminal menemes which are initially to be in the selected state. In assigning these default selections, any constraints that they may be subject to must be recognised, so that only valid combinations are specified. Similarly, the symbol § alongside a nonterminal meneme (real or virtual) indicates that a valid choice must always be present in the menu or submenu that this meneme represents; a default

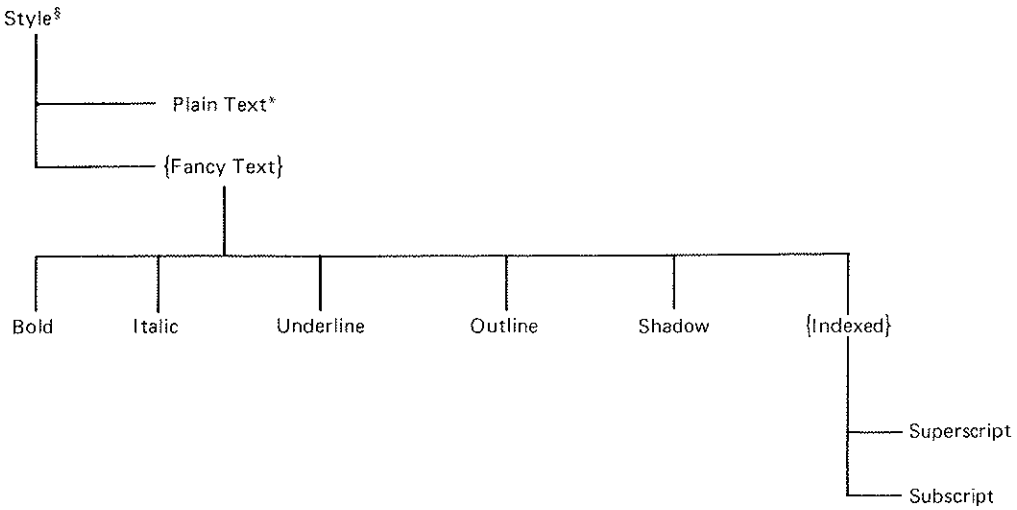

Figure 10. Description of Style portion of menu shown in Figure 2(b), using virtual menemes to allow basic structures to be combined. Default selections and 'required-choice' constraints are also shown

choice must be shown under these circumstances. In this context, a virtual meneme can be considered to be in the selected state if there is a valid selection present in the menu that it represents.

From Figure 10 it can be clearly seen that the initial (default) selection is to be Plain Text. Closer examination of the diagram reveals the complete specification of the behaviour of the Style menu, including the behaviour of the individual menemes as suggested in Figure 4. The Style item is marked with a 'required-choice' symbol, implying that there must always be a valid choice present in this menu. Taken together with the fact that the mutually exclusive alternative to Plain Text is a virtual meneme {Fancy Text}, this proscribes the deselection of Plain Text by direct excitation, because there is no single alternative that can be automatically selected in its stead. It further specifies that (in order to satisfy the required-choice constraint) when {Fancy Text} is deselected (i.e. no valid selection is present in that subgroup) then Plain Text will be automatically selected by indirect modification. However, because the virtual meneme {Indexed} is not tagged with a required-choice, then Superscript and Subscript can both be directly selected and deselected, and when one is deselected, it does not automatically select the other.

From this analysis, it can be seen that the complete behaviour of the individual menemes of the Style menu can be derived from the LC description shown in Figure 10. For example, that direct excitation is able only to select Plain Text and not to deselect it, is implied by the LC diagram without the fact being explicitly stated. It is a consequence of the desired menu structure rather than a basic design assumption. In a similar way Bold is seen to be bistable under direct excitation, another consequence of the desired structure. This suggests that there are different meneme types, according to their individual behaviour, a fact alluded to earlier in the discussion of the meneme model of Figure 3.

Although the basic meneme is bistable under direct excitation, syntactic constraints will often restrict this behaviour to select-only, as in the case of the Plain Text meneme above, and there are situations in which deselect-only, monostable, or even passive behaviour of a meneme may be required under direct excitation (Apperley, 1988).

With the LC notation, the type of an individual meneme is determined by its context. By default, menemes will be bistable, unless they are members of a required-choice mutually exclusive set, in which case they will generally be select-only. Presentation detail is a part of the context (see the next section), and headers to pull-down menus may be monostable by implication (that is, they revert to their deselected state automatically when the associated activity is completed). It is possible, and indeed sometimes desirable, to override these implied types, and facilities for this have been provided in the notation (Apperley, 1988). However, it must be remembered that much of the power of Lean Cuisine resides in the fact that it takes care of such details, generally making them not of immediate concern to the user/designer.

Towards the direct execution of Lean Cuisine

The LC notation that has just been detailed can be used to describe a menu dialogue at the highest level, that of the dialogue designer. Although Lean Cuisine makes no claim to be more than a design aid at the present time, it is worth investigating the possible direct implementation of menus using this technique, and in particular, the possible role of Lean Cuisine as the dialogue specification component of a UIMS (user interface management system). The diagram form, as shown in Figure 10, contains no presentational or semantic information, nor does it make any statement about underlying control models. As Green (1986) has suggested, it is desirable that the dialogue control component of a UIMS should closely follow the designer's model of the user interface. In the next two sections, a possible control model is discussed, and a textual representation of the LC tree is proposed which is amenable to the inclusion of presentational and semantic information.

Event-response system model

As it forms a complete description of the manner in which the menu or system of menus is to behave, an LC diagram could obviously be transformed into one of the more familiar dialogue descriptions, such as a transition network, and then implemented using available tools. However, the LC notation implicitly concerns flags associated with each meneme, real or virtual, and the events or excitations that modify these flags, rather than any concept of individual states associated with the permissible combinations of menemes. For this reason, the event-response system model (Hill, 1987) discussed in the earlier section on event models provides an ideal mechanism to bring Lean Cuisine closer to implementation.

As was shown earlier, the ERS model for a simple menu bears a close relationship to the corresponding Lean Cuisine tree diagram. In general there will be an ERS flag corresponding to each meneme in the diagram, although for

consistency with Hill's (1987) notation, it is necessary to use both the normal flag and its complement (i.e. strictly speaking, there are actually two flags corresponding to each meneme). There will also be an event associated with each terminal meneme, corresponding to the direct excitation of that meneme.

Although for the more complex structure of the Style menu, the relationship between the LC diagram and the ERS model shown in Figure 7(c) is more obscure, now that the LC model, with its virtual menemes, has been developed, it is possible to produce a more derivative ERS model that reflects this structure. Figure 11(b) shows the response rules for such an ERS model of the five meneme reduced version of the Style menu, the LC diagram for which is shown in Figure 11(a). To accommodate virtual menemes, some modifications have been made to Hill's (1987) model:

- When the flag corresponding to a virtual meneme appears as either a condition or an action in a rule, it is to be substituted by an appropriate Boolean expression, the 'or' combination of the menemes of its immediate subtree. The appropriate substitutions for the menu in question are shown in Figure 11(c).
- When a flag corresponding to a virtual meneme appears as a condition in a rule, then that flag is not to be cleared when the rule is invoked.

As can be seen from this example, there will generally be two regular rules corresponding to each real meneme, representing the selection and deselection of that meneme by direct excitation. There will also be two ε-rules associated with each virtual meneme that is a member of a mutually exclusive set, but no rules are generated by mutually compatible virtual menemes. Each virtual meneme also gives rise to two substitution rules.

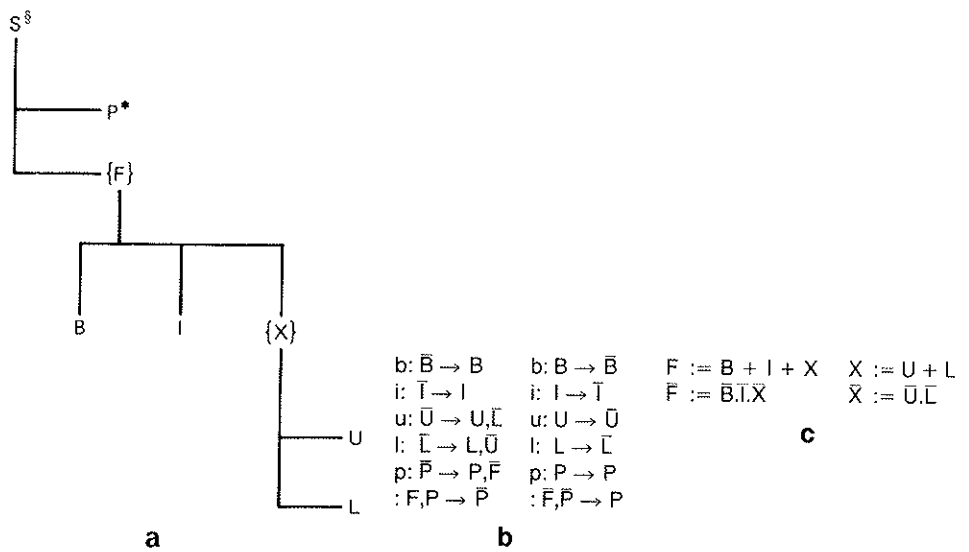

Figure 11. (a) Reduced form of Style menu, involving only five real menemes, (b) its description as an ERS, and (c) substitutions required for virtual meneme flags

In comparing Figure 11 with the simple menu groups of Figure 7, it can be seen that the first two lines of Figure 11(b), those corresponding to the direct excitations of B and I, follow a standard pattern for a mutually compatible pair, as seen in Figure 7(b). Similarly, the third and fourth lines of Figure 11(b), which correspond to the direct excitations of U and L, follow a standard pattern for a mutually exclusive pair, as seen in Figure 7(a). The last two lines of Figure 11(b), which correspond to the direct excitation of P, and to the virtual meneme {F}, differ from the standard pattern for a mutually exclusive pair because of the required-choice constraint, and because one of the pair is a virtual meneme.

The ERS model for this five (real) meneme menu comprises a total of 12 response rules. To complete the definition, there is a set of 14 flags (two for each meneme), a set of 5 events (one for each terminal meneme), and four substitution rules (two for each nonterminal meneme), together with the set S={P} of flags initially on. By contrast, as shown in the section on transition networks, a transition network representation of this menu comprises 12 individual states, and a total of 60 transition arcs interconnecting them. In fact, in the worst case of a menu comprising a mutually compatible set of N menemes, the transition network requires 2^N states and $N \times 2^N$ arcs, while the equivalent ERS model requires just $2N$ rules. However, while the effect of increasing the complexity of the menu syntax (by introducing mutually exclusivity, or further subgroups of both types) is to reduce the number of states in the transition network representation because not all of the potential states can occur, it increases the number of rules in the ERS model because of the virtual menemes introduced. Nevertheless, even in the case of the relatively complex Style menu shown in Figure 11(a), the syntax complexity reduces the number of states by a factor of only 2.7, and increases the number of rules by only 1.2; the ERS notation remains relatively economical.

The relationship between the ERS model and the LC notation has been dwelt upon here for two reasons. Most importantly, both are based firmly on the notion of an asynchronous set of events and the effect of these on a set of flags which define a total state, rather than on a set of states and the effect that each possible action might have on each of these states. In addition, because it is more formal, the ERS model shows a clear path to the implementation of the LC notation. Hill (1987) describes an implementation of the event-response system model as an event-response language (ERL). This language encompasses the formalism of ERS, but also includes parameters attached to flags and events, and provides for outgoing events and assignments within the action portion of the rules. These extensions provide the all-important semantic link between the interaction devices and the application. However, as it stands at present, ERL provides a means of implementing asynchronous dialogues within an application. The flags and their associated parameters are a part of the application data structure; ERL specifies the syntax of the dialogue and provides the link with the interaction devices. ERL does not necessarily achieve the functional separation of application and interface aspired to by the propounders of UIMSs.

Textual representation of Lean Cuisine

Tree structures, such as those used in LC notation, can be readily expressed in

textual form. Figure 12(a) shows a possible list representation of the reduced Style menu of Figure 11(a). Two menu constructors have been introduced, MC and ME, to specify respectively the two basic subgroup types, mutually compatible and mutually exclusive. Each menu subgroup comprises a list of three components; the header, the constructor, and the list of component menemes. Any one of the component menemes may itself be a menu subgroup.

Presentation information can be relatively easily added to such a notation; there may be style information for menemes (icon, text, etc) and for menus (pull-down, pop-up, etc). Although in the interests of functional separation of the application and the interface, there are arguments for specifying this presentation separately from the menu structure, the two could be combined as suggested in Figure 12(b). This approach has the advantage, from the dialogue-designer's point-of-view, that all relevant information about a menu or a meneme is contained in a single description. A system of menus, such as that shown in Figure 9, can also be represented in this way. A possible representation including menu styles (but not meneme styles) is shown in Figure 12(c). In all three examples shown in Figure 12, one is reminded of Hoare's (1978) comment that 'A design for a programming language must necessarily involve a number of decisions which seem to be fairly arbitrary.'

Two forms of semantic link can be provided to the application; both will in general be necessary. The first of these is the state variable record, the collection of all of the meneme state flags and any associated parameters as suggested by Hill (1987). This ensures that the application (or the set of application functions) can be constantly aware of the detail of the current menu states. The second form of semantic link is provided by allowing the association, with any meneme flag, of an invocation of an application function. It should be noted that in many cases it will not be necessary to invoke an application function

```
(S§,ME,(P*,({F},MC,(B,I,({X},ME,(U,L))))))
```

a

```
(S§:"Style",
    ME,(P*:"Plain Text",
        ({F},
            MC,(B:"Bold",
                I:"Italic",
                ({X},
                    ME,(U:"Superscript",
                        L:"Subscript")))))):pull-down
```

b

```
(A,
  ME,((B,ME,
          (D,E,F)):pull-down,
      (C,MC,
          (G,H,I,J)):pull-down
    ):menu-bar
```

c

Figure 12. (a) List representation of LC tree diagram of Figure 11(a), (b) the same list with presentational information included, and (c) system of menus as described in Figure 9

when a meneme changes state; the change of state may be the only action required. Terminal menemes will usually have some direct application consequence, causing a command to be executed or a parameter to be modified. However, nonterminal menemes may merely serve to provide access to lower level menus, so that their semantic function may already be established within the menu syntax description, and require no application action. Application links can be added to menemes in the list notation of Figure 12, in a manner analogous to that of the presentation information (Apperley, 1988).

This discussion assumes that there would be some form of UIMS in overall control of the system, to which the application functions were subservient. The relative merits of this model for a UIMS have been discussed extensively elsewhere (Gallop, 1987; Pfaff, 1985), and are essentially still unresolved. It is sufficient to say, however, that in a UIMS, the Lean Cuisine notation could be used to specify the control structure, the presentation detail (which it is anticipated would be derived from a library) and the semantic links to the application.

Conclusions

This paper has analysed the internal structure of computer display menus, and has developed a syntax or grammar of menus. Following from this, the general inadequacy of existing dialogue description techniques for the design of menu-based systems has been discussed, leading to a proposal for a diagrammatic notation: Lean Cuisine. This notation not only clearly specifies the structural information, but can also be extended to encompass the semantic and presentational details, showing the way to a possible direct execution. A more extensive example of the Lean Cuisine notation in use, one which includes a supporting design methodology, has also been produced (Apperley, 1988).

Fundamental to the Lean Cuisine technique is the notion that, at least in the early stages of the design, a major goal is to separate the dialogue description itself from the detail of its implementation. The design study mentioned (Apperley, 1988) demonstrates the way in which the notation provides a graceful evolution from generic design to specific implementation. Lean Cuisine is a tool for the designer, and throughout the design cycle it concentrates on the desired interface behaviour rather than the required underlying processes, and unnecessary early intrusion of implementation details are avoided.

While it is acknowledged that a design notation for menu dialogues addresses only a part of the total user interface management issue, the analysis leading to the notation provides an important understanding of the problems of menu specification and the more general description of direct manipulation interfaces. In the effort to produce UIMSs that ease the development burden of highly interactive direct manipulation systems, developers must be wary of temptation to progress too far too soon. It is very important that before attempting to construct tools, the tasks that the tools are to assist with are fully understood. For example, if an ill-equipped plumber was observed attempting, unsuccessfully, to disconnect a pipe using a hammer, it might be suggested that a heavier hammer be provided. What is required, of course, is a better under-

standing of the problem, which might reveal that a more appropriate tool was a spanner.

Further, it is suggested that not only are menu systems fundamental to the direct manipulation model, but that the approach taken in Lean Cuisine could be extended to cover the other interactions of a direct manipulation system. In developing the event-response language, Hill (1987) has proposed that parameters be associated with flags. In a similar way, parameters could be associated with menemes in Lean Cuisine, so that an interaction object, such as a scroll bar, could be treated as a single parametrised meneme. When the meneme was selected, its value would be able to be adjusted (by direct manipulation), and this value would be both passed back to the application and reflected in the current appearance of the meneme. The basic action of selection maps directly into the meneme model, and other more general direct manipulation actions, such as positioning and dragging, can also be regarded as 'selection + parameter modification' actions. Thus, with further development, LC notation could become a basic design tool for direct manipulation interfaces.

References

Apperley, M.D. (1988) 'Tap: a menu interface design study using the Lean Cuisine notation' *Information Engineering Report #88/2* Department of Electrical Engineering, Imperial College, London, UK

Apperley, M.D. and Spence, R. (1983) 'Hierarchical dialogue structures in interactive computer systems' *Softw. Pract. Exper.* **13**, 777–790

Borufka, H.G., Kuhlmann, H.W. and ten Hagen, P.J.W. (1982) 'Dialogue cells: a method for defining interactions' *IEEE Comput. Graph. Appl.* **2**, 5, 25–33

Brown, J.W. (1982) 'Controlling the complexity of menu networks' *Commun. ACM* **25**, 7, 412–418

Browne, D.P., Sharrat, B.D. and Norman, M.A. (1986) 'The formal specification of adaptive user interfaces using command language grammar' *CHI '86 Conf. Proc.* 256–260

Cardelli, L. and Pike, R. (1985) 'Squeak: a language for communicating with mice' *Comput. Graphics* **19**, 3, 199–204

Cockton, G. (1985) 'Three transition network dialogue management systems' in **Johnson, P. and Cooke, S. (eds)** *People and computers: designing the interface* Cambridge University Press, UK, 138–147

Field, G.E. (1988) 'Navigation of menu-accessed information space: psychological experimentation in human–computer interaction' *PhD Thesis* University of London, UK

Flecchia, M.A. and Bergeron, R.D. (1987) 'Specifying complex dialogs in ALGAE' *CHI + GI '87 Conf. Proc.* 229–234

Foley, J.D. and Wallace, V.L. (1974) 'The art of natural graphic man–machine conversation' *Proc. IEEE* **62**, 462–471

Gallop, J.R. (1987) 'User interface management and graphics standards' *Inf. Softw. Technol.* **29**, 4, 202–206

Giroux, L. and Belleau, R. (1986) 'What's on the menu? The influence of menu content on the selection process' *Behav. Inf. Tech.* **5**, 2, 169–172

Green, M. (1985) 'The University of Alberta user interface management system' *Comput. Graphics* **19**, 3, 205–213

Green, M. (1986) 'A survey of three dialogue models' *ACM Trans. Graphics*, **5**, 3, 244–275

Hemenway, K. (1982) 'Psychological issues in the use of icons in command menus' *Proc. CHI '82 Conf. Human factors in Computer Systems* 20–23

Hill, R.D. (1986) 'Supporting concurrency, communications and synchronization in human–computer interaction — the Sassafras UIMS' *ACM Trans. Graph.* 5, 3, 179–210
Hill, R.D. (1987) 'Event-response systems — a technique for specifying multi-threaded dialogues' *CHI + GI '87 Conf. Proc.* 241–248
Hoare, C.A.R. (1978) 'Communicating sequential processes' *Comm. ACM* **21**, 8, 666–677
Hutchins, E.L., Hollan, J.D. and Norman, D.A. (1986) 'Direct manipulation interfaces' in **Norman, D.A. and Draper, S.W. (eds)** *User centered system design* Lawrence Erlbaum Associates Inc, 87–124
Jacob, R.J.K. (1983) 'Using formal specifications in the design of a human–computer interface' *Commun. ACM* **26**, 4, 259–264
Jacob, R.J.K. (1985) 'A state transition diagram language for visual programming' *IEEE Computer*, **18**, 51–59
Jacob, R.J.K. (1986) 'A specification language for direct manipulation interfaces' *ACM Trans. Grap.* **5**, 4, 283–317
Kasik, D.J. (1976) 'Controlling user interaction' *Comput. Graphics* **10**, 2, 109–115
Kieras, D. and Polson, P.G. (1983) 'A generalised transition network representation for interactive systems' *Proc. CHI '83: Human Factors in Computer Systems* (oston, MA, USA) 103–106
Kilgour, A. (1987) 'Theory and practice in user interface management systems' *Inf. Softw. Technol.* **29**, 171–175
Koivunen, M-R. and Mantyla, M. (1988) 'Hut Window: an improved architecture for a user interface management system' *IEEE Comput. Graph. Appl.* 8, 1, 43–52
Lieberman, H. (1985) 'There's more to menu systems than meets the screen' *Comput. graphics* **19**, 3, 181–189
MacLean, A. (1987) 'Human factors and the design of user interface management systems: EASIE as a case study' *Inf. Softw. Technol.* **29**, 4, 192–201
Miller, D.P. (1981) 'The depth/breadth tradeoff in hierarchical computer menus' *Proc. 25th Annual Meeting of the Human Factors Society* 296–300
Newman, W.M. (1968) 'A system for interactive graphical programming' *Proc SJCC, AFIPS Conf.* **32**, 47–54
Norman, K.L. and Chin, J.P. (1988) 'The effect of tree structure on search in a hierarchical menu selection system' *Behav. Inf. Tech.* **7**, 51–66
Palmer, T.R. (1987) 'GRAPE programming environment' *Inf. Softw. Technol.* **29**, 219–225
Perlman, G. (1984) 'Making the Right Choice with Menus' *Proc. INTERACT '84: 1st IFIP Conf. on Human–Computer Interaction* orth-Holland, Amsterdam, Holland, 291–295
Pfaff, G.E. (ed) (1985) *User interface management systems.* Springer-Verlag, Berlin, FRG
Reisner, P. (1981) 'Formal grammar and human factors design of an interactive graphics system' *IEEE Trans. Softw. Eng.* **SE-7**, 2, 229–240
Reisner, P. (1982) 'Further developments toward using formal grammar as a design tool' *Proc. CHI '82 Conf. human factors in Computer Systems* 304–308
Shneiderman, B. (1982) 'Multiparty grammars and related features for defining interactive systems' *IEEE Trans. Syst. Man and Cybern.* **SMC-12**, 2, 148–154
Shneiderman, B. (1987) *Designing the user interface: strategies for effective human-computer interaction* Addison-Wesley, Reading, MA, USA
Tocci, R.J. (1980) *Digital systems principles and applications* Prentice-Hall, Englewood Cliffs, NJ, USA
Wasserman, A.I. (1985) 'Extending state transition diagrams for the specification of human–computer interaction' *IEEE Trans. Softw. Eng.* **SE-11**, 8, 699–713