A Method for the Early Stages of Interactive System Design using UML and Lean Cuisine+

Chris Scogings and Chris Phillips
Massey University
New Zealand
email: {C.Scogings, C.Phillips}@massey.ac.nz

Abstract

In interactive system design, 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. This paper examines task modelling in UML and dialogue description in Lean Cuisine+, and describes a method for the early stages of interactive systems design which incorporates both notations. 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.

Keywords: *interactive systems design, dialogue modelling, task modelling, UML*

1. Introduction

The design of the external or visible system [8] has assumed increasing importance over the past decade as graphical user interfaces (GUIs) have become dominant, and as more and more attention has been devoted to usability aspects of interactive systems. As a relatively new aspect of system development, external system design requires good descriptive systems, and the development of tools to support it [11, 14]. 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 form an important input to user interface design. For example 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 by a lack of tools to support their definition, and weak linkage to dialogue [6].

Task models focus on task decomposition and/or task flow. Their primary purpose is to define the activities of the user in relation to the system, as a means of uncovering functional requirements to be supported by the system. Over the past 20 years considerable effort has been devoted to the development of models and tools to support the functional 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. Some convergence is currently occurring through initiatives such as the Unified Modelling Language (UML).

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". 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. A criticism of existing dialogue notations is that the linkage to user tasks is often obscure [4]. They define the structure and behaviour of the dialogue but they do not always 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.

In a previous paper [26] the authors proposed that the divide between task and dialogue modelling could be bridged via the Lean Cuisine+ notation. This paper develops this proposal and describes a method for the early stages of user interface design which involves Lean Cuisine+ and UML.

In Section 2 of the paper, activities and models associated with the interactive system development lifecycle are briefly reviewed. Section 3 examines task modelling as it is supported within UML. The Lean Cuisine+ notation is introduced in Section 4, and its task and dialogue modelling capabilities are outlined. In Section 5 a method for the early stages of interactive systems design involving both notations is presented via a case study. Work in progress is described in Section 6.

2. Interactive systems design

The interactive system development life-cycle is a highly iterative process which involves 5 key activities:

1. requirements analysis and specification;
2. design of the external (visible) system;
3. design of the internal (software) system;
4. system implementation;
5. system operation and 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 briefly explored from the viewpoint of system modelling.

0-7695-0969-X/01 $10.00 © 2001 IEEE

Both task modelling and object modelling are components of activity 1. Collectively these two models define the static and dynamic requirements of the system. Activity 2 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.

Connections and relationships between the artefacts of these models are the subject of a number of recent and current research projects (see [26] for a review by the authors of this paper). Task and object modelling are a significant aspect of all current object-oriented analysis and design methods and are also the subject of some recent research [5, 31]. UML supports both task and object modelling, as described in the next section.

The research referred to above has as a common aim the integration of some of the models involved in interface analysis and design. An identified difficulty in this area [37] 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 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.

3. Task modelling in UML

A variety of task description methods have been developed, often involving graphical techniques. Some of these focus primarily on task decomposition, e.g. HTA [35]. Others are concerned primarily with task flow. Use case modelling, as supported in UML [9, 13, 29], falls into the latter category.

UML appears set to become the object-oriented modelling language standard [20]. UML consists of a collection of semi-formal graphical notations, These notations, which can be used to support different development methodologies, include use case diagrams, class diagrams, interaction diagrams, activity diagrams and state diagrams.

Use cases, which are abstract task scenarios, were originally developed in connection with object-oriented software engineering [18], as a way of uncovering the functionality of a system. They model user actions and system responses, and involve the manipulation of objects and the sequencing of interactions.

It should be noted that the term scenario is used in UML to describe one thread or path through a use case. This suggests that no conditions are possible, although opinions differ on this [7, 30]. A comprehensive review of the use of scenarios in design is provided in [12]. Scenario-based development methods are still evolving, and the relationship between task scenarios and object-oriented software development is being explored [32].

Existing UML-based methods are based on the initial definition of use cases. Useful guidelines for establishing use cases are provided in [30]. It has been pointed out [13] that use cases can represent different levels of task detail and may subsume other use cases. Determining the level of detail remains a challenge for today's system developers.

Use cases can be represented at a system context level in use case diagrams, which show the boundaries of the system, the external actors, and any structural relationships which exist between use cases. The internal behaviour of a use case can be described graphically in UML using either activity diagrams, state diagrams or interaction diagrams, depending on the perspective required and the complexity of the task. Static object relationships are captured in separate class diagrams. Various techniques exist for identifying classes [23, 36], including derivation from use cases.

Many commentators [e.g. 13, 17, 30] support the need for detailed textual use case descriptions. Textual descriptions are supported in the UML reference manual [33]. A structured format involving primary and secondary flows is generally recommended [3, 29]. Pre- and post-conditions can also be stated for each use case [15]. Textual use case descriptions are used in the method described in Section 5 (see Figure 1).

Dialogue modelling is not directly supported in UML, although it might be attempted using state diagrams. More importantly, there is no direct means of relating tasks, captured as use cases, to dialogue structure. UML provides little direct support for user interface design and several researchers have suggested extensions in this regard [2, 21]. The only direct support is that objects in the system can be designated *interface objects* and defined as operations which involve communication with the environment. This is based on the work in [16].

4. The Lean Cuisine+ notation

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. Dialogue models can provide a useful starting point for interface design [15].

A variety of dialogue models at various levels of abstraction have been employed in the early stages of interface design, involving both graphical and textual notations[27]. High level user interface representation embodies information about the interface in terms of objects, actions, states, and relationships, including both pre- and post-conditions associated with the actions.

Lean Cuisine+ [28] is a semi-formal graphical notation for describing the behaviour of event-based direct manipulation GUIs, and is an extension of Lean Cuisine

[1], a graphical notation based on the use of tree diagrams to describe systems of menus.

4.1 Representing dialogue structure

In a Lean Cuisine+ specification, the interface is represented primarily as a dialogue tree. The behaviour of the interface is expressed in terms of the constraints and dependencies which exist between the nodes of the tree - selectable representations called *menemes*. A meneme can be selected or not selected, and available (for selection) or unavailable, and can thus be in one of four possible states. Events are not directly represented but are implicitly associated with menemes.

Menemes can represent actions, parameters, objects, states and other attributes. The state of the interface at any time is represented by the collection of individual meneme states. Menemes can be grouped into subsets that are either mutually exclusive (1-from-n) or mutually compatible (m-from-n). Graphically, mutually exclusive menemes are grouped vertically and mutually compatible menemes are grouped horizontally. Examples of both groupings appear in the Lean Cuisine+ tree in Figure 3.

The availability of an option for selection can be governed by a precondition attaching to the meneme which represents it. This provides for description of the unavailability of selectable options at appropriate points in a dialogue. System responses to user generated events are represented by selection triggers. They capture the fact that the selection or deselection of an option may trigger (possibly conditionally) the selection or deselection of other options. Examples appear in Figure 6.

It is possible to automatically generate a Lean Cuisine+ dialogue tree from prototype code [25, 34].

4.2 Representing tasks

Of particular interest to the research reported here, is the fact that each task (sequence of related sub-tasks) can be represented in Lean Cuisine+ as an overlay of linked nodes superimposed on the dialogue tree diagram. The task overlay forms an important link between the description of the dialogue structure (how functionality is grouped) and interaction sequences (how functionality is accessed). This allows analysis of whether all the actions required for a task are supported by the dialogue, and whether they can be efficiently executed. Examples are shown in Figures 4 and 7.

The Lean Cuisine+ task overlay provides a number of 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; and the overlay can be used in interface analysis.

It may also be possible for Lean Cuisine+ task overlays to be generated automatically from suitably defined task descriptions, and work is proceeding on this.

5. A design method

To date, system development methods have shown a marked lack of support for the design of the user interface [19, 38]. A number of UML-based system design approaches have been proposed [for example, 10, 13, 15, 17, 30]. There is considerable variation in the way the notation is applied.

In this section, a method for the early stages of interactive system design, which supports the mapping of tasks to dialogue, is presented via a case study. It builds on the strengths of UML and Lean Cuisine+. The case study concerns the development of a Timetable Viewer software package for tertiary institutions which provides 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: Define use cases

The first step in the method is the definition of use cases. Four initial use cases were defined for the Timetable Viewer system. One of these, Create Student Timetable, is shown in Figure 1. The template used here is described in [22, pp 108 – 111].

Use Case:	Create Student Timetable
Actor:	Student
Goal:	The student wants to print her own timetable.
Process:	The student selects a semester and then selects a number of papers. As each paper is selected, the system includes the times for that paper in the timetable display. The student can select to display only lectures, only tutorials or only laboratories, or any combination of these. When the display is satisfactory, the student can print the timetable and the use case terminates.
Variations:	The student selects the wrong paper. To correct the problem, the student may deselect the paper. The system displays the timetable without the deselected paper.

Figure 1: The 'Create Student Timetable' use case.

Step 2: Identify domain object classes

In common with other UML-based system development methods, the next step involves establishing the classes for the main objects in the system. In general, class diagrams are developed incrementally as a system is designed [24]. The object classes, and the relationships between them, are uncovered through analysis of the use cases defined above. At this point, it is only necessary to establish classes for the *domain objects* (such as semester

and paper), as opposed to *system objects* (such as window or menu) which may be added later. These domain objects are represented as a UML class diagram (Figure 2). Attributes and operations required for user selection *must* be included at this stage.

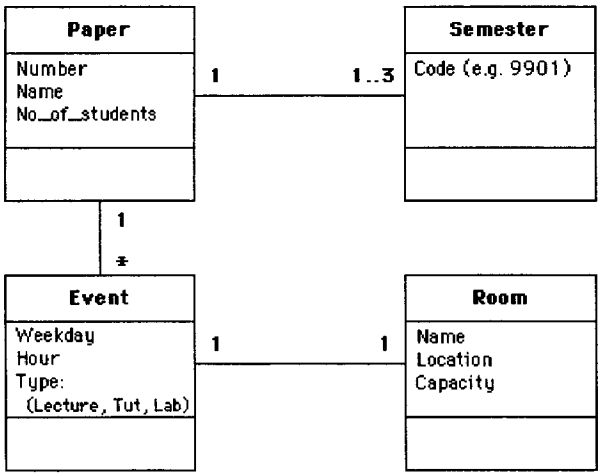


Figure 2: Initial domain classes for the Timetable Viewer, expressed as a UML class diagram.

Step 3: Construct an initial Lean Cuisine+ tree

The purpose of a user interface is to provide the user with access to objects in the domain of interest. Thus it is appropriate to base the initial dialogue structure on the domain object classes of the system. This is particularly so in the case of direct manipulation GUIs.

Using the information captured in the UML class diagram developed above, the basic Lean Cuisine+ tree is now constructed as follows:

- A subdialogue header meneme corresponding to each domain class name is created.
- A meneme for each *selectable* class attribute in the appropriate subdialogue is created. A selectable attribute is one which may be selected by an actor. For example, rooms in the Timetable Viewer system are only selected by name - they are not selected by capacity. The knowledge of which attributes are selected is determined through domain analysis. If there is uncertainty, an attribute should be represented by a meneme as all unused menemes are removed later on.

Figure 3 shows the basic tree structure for the Timetable Viewer example. The meneme names are exactly those appearing in the class diagram for the Timetable Viewer. *Non-selectable* attributes have not been included as menemes, e.g. Paper Name or Room Capacity. It must be emphasised that this is a starting point of an iterative process and the diagram is not intended to be complete at this point.

All known classes have been included in the basic tree for this example. However, if classes obviously have nothing to do with the user interface, they could be omitted from the basic tree. For example, a class may deal with low level communication protocols. If uncertainty exists, the class should be included as all unused menemes are removed at a later stage.

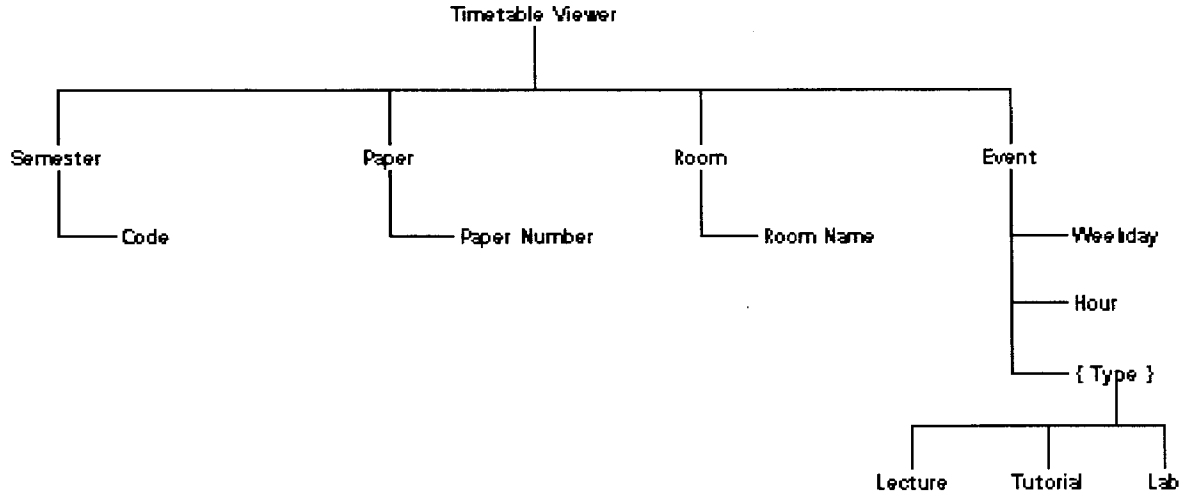


Figure 3: The first draft of the initial Lean Cuisine+ diagram for the Timetable Viewer example.

The way menemes are arranged visually within a Lean Cuisine+ subdialogue indicates whether the group 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 grouping is required for each subdialogue and the initial grouping may require revision at a later stage. It is proposed that groups should be assumed to be mutually

exclusive until shown to be otherwise. Sometimes the correct grouping can be identified early on, e.g. in Figure 3 the whole purpose of the meneme group [Lecture, Tutorial, Lab] is to provide for any combination of the three and thus they are represented as a mutually compatible group. Note that {Type} is shown as a *virtual* meneme – a grouping mechanism in Lean Cuisine+. Type would not be presented to the user.

Step 4: Construct initial Lean Cuisine+ task sequences

Lean Cuisine+ task sequences are now superimposed on the initial tree diagram. These task sequences are derived directly from the use cases. A task sequence may correspond either with an entire use case, or with a scenario (one path through a use case).

Figure 4 shows the initial task sequence for the Create Student Timetable use case of Figure 1, mapped over the Lean Cuisine+ tree of Figure 3. Two *floating* menemes, Display Timetable and Print, have been added to the diagram, as 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 Timetable indicates that this is a system action. This is derived from domain knowledge - in this example the system (not the user) causes the timetable to be displayed. The double-headed arrow indicates that several paper numbers can be selected and the selection of the virtual meneme {Type} indicates that any, or all, of the menemes within that sub-dialogue may be selected. A full description of task sequences can be found in [26].

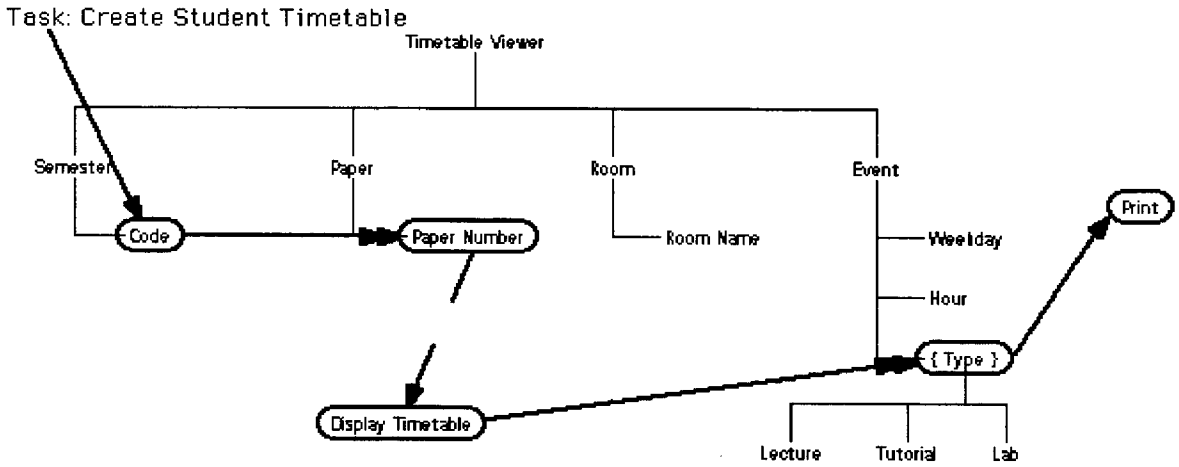


Figure 4: The first draft of the task sequence for Create Student Timetable.

Step 5: Refine the Lean Cuisine+ tree

The initial task sequences are now reviewed to determine the following:

- menemes that are required but are not yet present in the diagram;
- menemes that are present but are never used;
- structural refinements required in relation to meneme sub-groups: whether sub groups should be mutually exclusive or mutually compatible; whether any homogeneous subgroups exist; and whether sub-group headers should become virtual menemes.

Significantly, this is the point where the design process changes from a *system-centred* approach to a *user-centred* approach. Thus, menemes which were constructed from class attributes (internal to the system) may now change (in name) to reflect the user's view of the system.

Figure 5 illustrates refinements made to the Lean Cuisine+ tree structure for Create Student Timetable. Floating menemes which are used to change the display have been added and grouped together in a new Display subdialogue. Menemes which are not used, e.g. Weekday, have been removed. The name of the reduced Event subdialogue has been changed to Controls, to reflect the fact that from a user point of view, this is used to control the contents of the timetable display (some combination of the options in the sub-group). The *fork* symbol has been added to represent homogeneous groups: Paper Number is a mutually compatible group as several papers can be displayed in one timetable, whereas Semester Code is a mutually exclusive group as a timetable display only covers one semester at any one time. Most of the sub-group headers have become virtual menemes, reflecting the fact that they will not be directly selectable by the user.

Step 6: Enrich the Lean Cuisine+ model

Any selection triggers are now added to the diagram. These correspond to system responses to user generated

events in the task sequence. In Figure 6 the trigger between Paper Number and Display Timetable shows that the selection of Paper Number by the user will trigger selection of Display Timetable by the system, provided the condition is satisfied.

Meneme designators are also added to the diagram at this stage. These further constraints on the dialogue are a part of early user interface design. They provide a means of over-riding the default bistable behaviour of menemes, and provide for select-only (↑), deselect-only (↓), monostable (⊥) or passive (⊗) behaviour. Figure 6 contains some examples, and also shows an example of a required choice group (§) with an initial default (*).

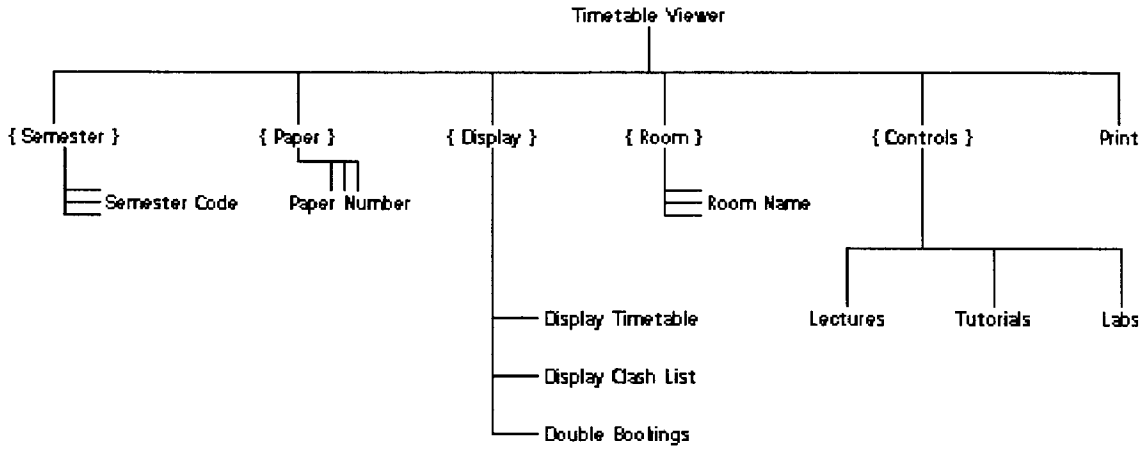


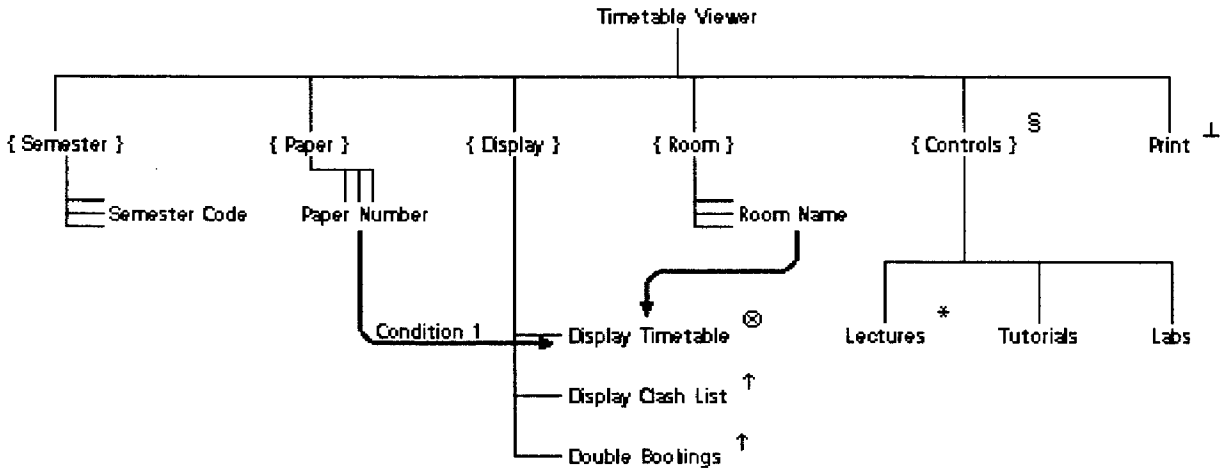
Figure 5: The refined tree structure for the Timetable Viewer example.

Step 7: Remap the Lean Cuisine+ task sequences

The Lean Cuisine+ task sequences identified at Step 4 are remapped on to the refined dialogue tree. The task sequence illustrated in Figure 7 is a development of the task sequence presented in Figure 4.

The method described above can be summarised as follows:

1. Define a set of detailed UML use case descriptions.
2. From the use cases and other domain knowledge, identify the important domain object classes. Construct a UML class diagram. Any attributes which may be selected by a user must be included.
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. Mutually exclusive meneme grouping is the default.



Condition 1: *Display Clash List is NOT currently selected*

Figure 6 : The Lean Cuisine+ diagram for the Timetable Viewer with triggers and designators added

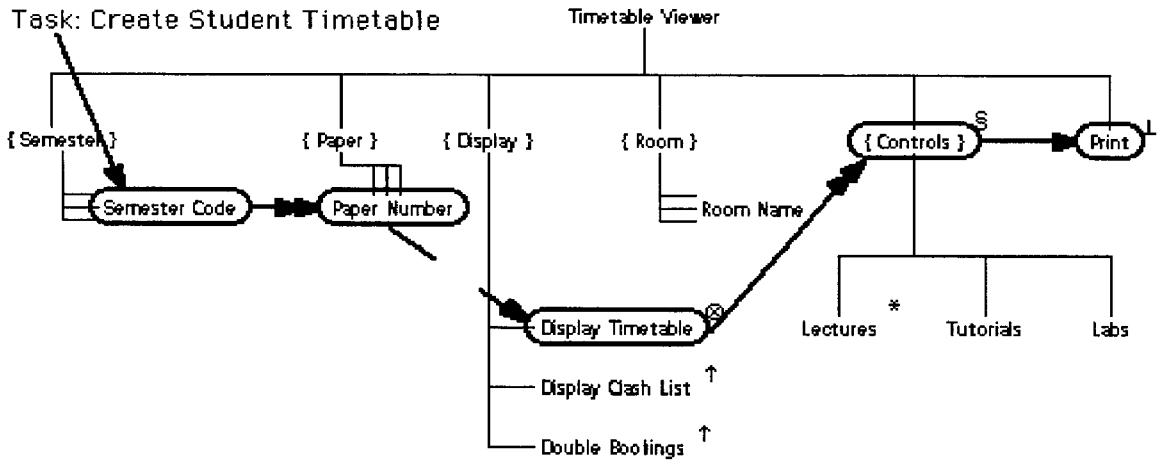


Figure 7 : The new Lean Cuisine+ task sequence for Create Student Timetable

5. Review and refine the Lean Cuisine+ tree diagram. Add floating menemes identified at Step 4, and remove unused menemes. Group new menemes according to functionality. Make structural refinements as required in regard to meneme grouping and sub-group headers. Rename menemes where appropriate to reflect a user view.
6. Enrich the Lean Cuisine+ model. Add triggers to correspond to system actions in the task sequences. Add meneme designators to represent further dialogue constraints 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.

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 next stage is to translate this 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.

6. Work in progress

This paper has examined task modelling in UML and dialogue description in Lean Cuisine+, and described a method for the early stages of interactive systems design which incorporates both notations. 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 method needs to be further developed and tested. A number of other issues remain to be resolved, including the relationship between UML, Lean Cuisine+ and screen design, where elements must be grouped spatially. A related project is exploring extensions to UML, including alternative ways of specifying use cases, to better support the spatial design of the user interface.

Work in progress on this project includes:

1. Development of a software environment to support the method. A prototype Lean Cuisine+ support environment already exists, and provides for the generation of dialogue trees and task overlays. It is hoped to at least partially automate the production of initial Lean Cuisine+ diagrams and task overlays from UML use cases and class diagrams.
2. Exploration of the use of Lean Cuisine+ as an analytical tool to check that all tasks uncovered at the requirements stage are supported by the system and that they can be carried out in an efficient manner (in regard to key strokes and selections). It may also be possible to partially automate this.

References

[1] Apperley, M.D. & Spence, R. (1989): Lean Cuisine: A Low-Fat Notation for Menus, Interact. with Comput., 1, 1, 43-68.

[2] Artim J. et al. (1998): Incorporating work, process and task analysis into commercial and industrial object-oriented systems development, SIGCHI Bulletin, vol. 30, no. 4.

[3] Booch G., Rumbaugh J. & Jacobson I. (1999): The Unified Modeling Language User Guide, Addison-Wesley, Reading, Massachusetts.

[4] Brooks, R. (1991): Comparative Task Analysis: An Alternative Direction for Human-Computer Interaction Science, in Designing Interaction: Psychology at the Human-Computer Interface, Carrol, J.M. (Ed), CUP, Cambridge, 50-59.

[5] Brown, J. & Marshall, S. (1998): Sharing Human-Computer Interaction and Software Engineering Design Artefacts, Proc. of OZCHI'98, Adelaide Australia, IEEE, 53-60.

[6] CHI (1999): Tool Support for Task-based User Interface Design, Workshop 7, CHI'99, Pittsburgh, Pen.

[7] Cockburn A. (1997): Structuring Use Cases with Goals, from website acockburn@aol.com.

[8] Collins, D. (1995), Designing Object-Oriented User Interfaces, Redwood City, Benjamin/Cummings Publishing Company.

[9] Constantine L.L. & Lockwood L.A.D. (1999): Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design, Addison-Wesley, Reading, Massachusetts.

[10] Eriksson H-E. & Penker M. (1998): UML Toolkit, John Wiley & Sons, New York.

[11] Farooq, M.U. & Dominick, W.D. (1988): Survey of formal tools and models for developing user interfaces, Int. J. Man-Mach. Stud., 29, 479-496.

[12] Filippidou D. (1998): Designing with Scenarios: A critical review of current research and practice, Requirements Engineering 1998, no 3, Springer-Verlag, London, UK, pp 1 - 22.

[13] Fowler M. & Scott K. (1997): UML Distilled: Applying the standard object modeling language, Addison-Wesley, Reading, Massachusetts.

[14] Guindon, R. (1990): Designing the Design Process: Exploiting Opportunistic Thoughts, HCI, 5, 305-344.

[15] Jaaksi A., Aalto J-M., Aalto A. & Vatto K. (1999): Tried & True Object Development, Cambridge University Press, Cambridge, UK.

[16] Jacobsen I., Ericsson M. & Jacobson A. (1995): The Object Advantage, Addison-Wesley, Wokingham, England.

[17] Jacobson I., Booch G. & Rumbaugh J. (1999): The Unified Software Development Process, Addison-Wesley, Reading, Massachusetts.

[18] Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G. (1992): Object-oriented Software Engineering, A Use Case Driven Approach, Addison-Wesley, Reading, MA.

[19] Kemp E.A. & Phillips C.H.E. (1998): Extending support for User Interface Design in Object-Oriented software engineering methods, Proc. of HCI'98, Sheffield, England, September 1998, 96-97.

[20] Kobryn, C. (1999): UML 2001: A Standardisation Odyssey, Comm. ACM, Vol 42, No 10, 29-37.

[21] Kovacevic S. (1998): UML and User Interface Modeling, Lecture Notes in Computer Science 1618, Bezivin J. & Muller P-A. (Eds), Springer-Verlag, Berlin, Germany.

[22] Oestereich B. (1999): Developing Software with UML, Addison-Wesley, Harlow, England.

[23] Overmeyer S. (2000): Conceptual modeling through Linguistic Analysis using LIDA, Seminar at Massey University, Auckland, New Zealand, 8 May 2000.

[24] Paech B. (1998): On the role of Activity Diagrams in UML, Lecture Notes in Computer Science 1618, Bezivin J. & Muller P-A. (Eds), Springer-Verlag, Berlin, Germany.

[25] Phillips C.H.E. & Scogings, C. (1997): Modelling the mock-up: towards the automatic specification of the behaviour of early prototypes, Interact'97, Sydney, Australia, July 1997, IFIP, Chapman & Hall, London, 591-92.

[26] Phillips, C.H.E. & Scogings, C. (2000): Task and Dialogue Modelling: Bridging the Divide with Lean Cuisine+, Proc AUIC'2000, IEEE, Canberra, Australia, 1-3 February 2000, 81-87.

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

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

[29] Quatrani T. (1998): Visual modeling with Rational Rose and UML, Addison-Wesley, Reading, Massachusetts.

[30] Richter, C. (1999): Designing Flexible Object-Oriented Systems with UML, Macmillan, Indianapolis, Indiana.

[31] Roberts, D., Berry, R. & Isensee, S. (1998): Object View & Interaction Design, Tutorial 6, Interact'97, Sydney Australia.

[32] Rosson, M.B. & Carroll, J.M. (1995): Integrating Task and Software Development for Object-oriented Applications, CHI'95 Proc., ACM, New York, 377-384.

[33] Rumbaugh J., Jacobson I. & Booch G. (1999): The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, Massachusetts.

[34] Scogings, C. & Phillips C.H.E. (1998): Beyond the interface: modelling the interaction in a visual progamming environment, Proc. of HCI'98, Sheffield, England, September 1998, 96-97, 108-109.

[35] Shepherd, A. (1989): Analysis and training in information technology tasks, in Task Analysis for Human-Computer Interaction, Diaper, D. (Ed), Ellis Horwood, Chichester, 15-55.

[36] Stevens P. & Pooley R. (2000): Using UML: Software Engineering with Objects and Components [Updated Edition], Addison-Wesley, Harlow, England.

[37] Wood, L. E. & Zeno, R. (1996): Transforming User-Centered Analysis into Concrete Design: A CHI'96 Workshop, SIGCHI Bulletin, Vol 28, No 4, 35-38.

[38] Yourdon, E. (1994): Object-Oriented system design: an integrated approach, Prentice-Hall, New Jersey.