DIVISION OF COMPUTER SCIENCE

AN OBJECT ORIENTED DESIGN OF A SPEECH DRIVEN USER INTERFACE MANAGEMENT SYSTEM

Ray Frank
Jill Hewitt

Technical Report No.173

March 1991

AN OBJECT ORIENTED DESIGN OF A SPEECH DRIVEN USER INTERFACE MANAGEMENT SYSTEM

Ray FRANK and Jill HEWITT, School of Information Sciences, Hatfield Polytechnic, College Lane, HATFIELD, Herts. AL10 9 AB ENGLAND. Tel: 07072 79359 e-mail: comqrjf@hatfield.uk.ac

This paper describes an object oriented design of a User Interface Management System (UIMS) for an intelligent speech driven interface to computer applications, which is based on a generic task model approach. An object-oriented method is used to represent the class of models required in the system and a particular example, a simple programming environment for the language Modula-2 is described. The paper concludes with a discussion of the extension of the method to other more complex applications.

1. INTRODUCTION

As part of the Intelligent Speech Driven Interfaces Project (ISDIP) at Hatfield Polytechnic we are investigating the usability of human-computer interfaces using speech input (Hewitt & Furner (1988),Hewitt and Zajicek (1990)). It is important that such interfaces provide transparent access to an underlying application, but because of the imperfect nature of speech recognition systems, they should also provide some means of error recovery and, for maximum effectiveness, some degree of tailoring to the individual user and to the application. The design of an effective speech input interface is a complex one, involving consideration of the recognition hardware, the user's speech patterns, the syntax of the application and the user's task model. The UIMS should therefore be structured so that due consideration can be given to these various factors by researchers into particular areas and so that the different aspects of the system can be worked on independently.

We propose an object-oriented architecture as providing the most flexible approach to the UIMS design and the best way of integrating the various components of the system.

The advantages of such an approach are well documented (Hewitt & Frank (1989)). In particular, we identify the concepts of inheritance, data integrity and the independent development of objects as important for our system design.

2. THE METHOD

The phases in the development of a system are shown in Figure 1. The system objects are identified once the functional requirements of the system have been established.

The steps of the Object Oriented Design method have been described in a number of other papers, particularly (Booch 1983), however here each object is represented as a subsystem containing three component objects, an abstraction component, a control component and a display component (Frank & Hewitt, 1990) as shown in Figure 2. The steps outlined in this paper also include the allocation of objects to MASCOT subsytems (Simpson and Jackson 1979, Simpson 1986) to allow the design to be implemented in a concurrent procedural language, and the specification of the dialogue to be used between the user and the computer system.

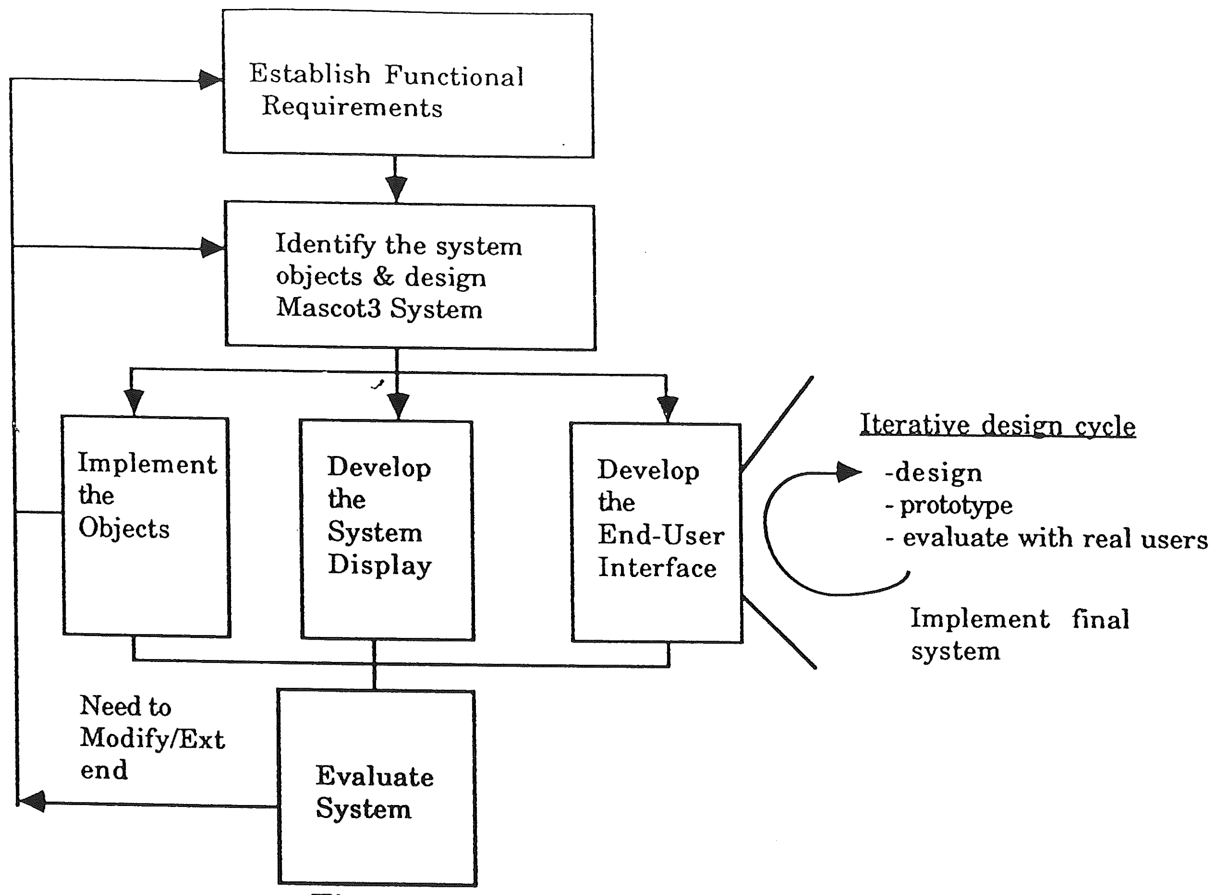

Establish Functional Requirements
Identify the system objects & design Mascot3 System
Implement the Objects
Develop the System Display
Develop the End-User Interface
Iterative design cycle
-design
- prototype
- evaluate with real users
Implement final system
Need to Modify/Extend
Evaluate System

Figure 1 The Object Oriented Method

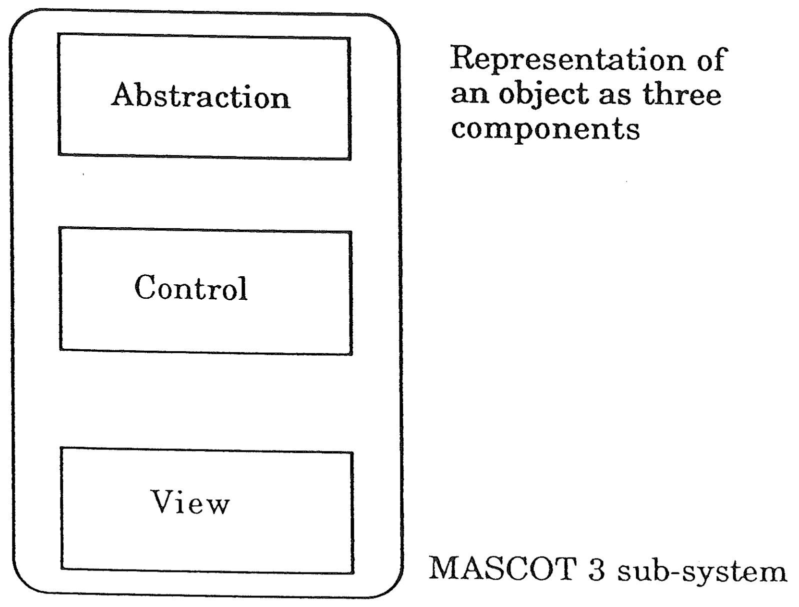


Figure 2 The Three Components of an Object

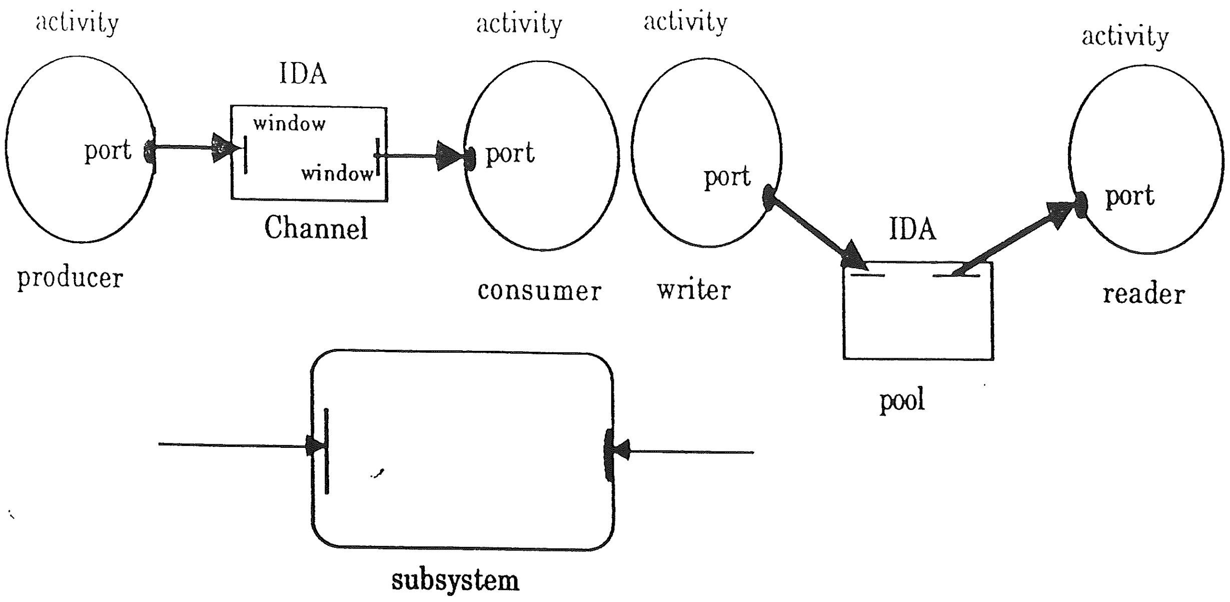


Figure 3 MASCOT 3 Element Representations

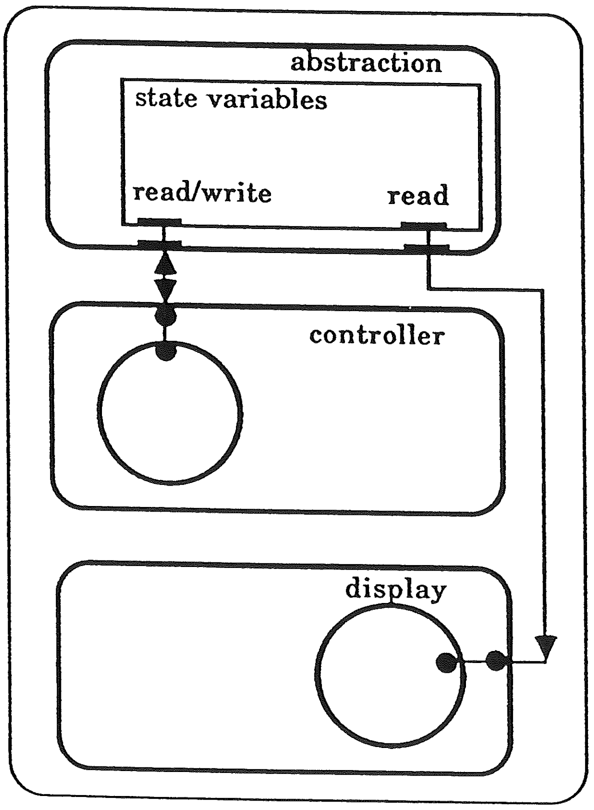


Diagram to show the communication paths between the object components.

Figure 4 Example Objects MASCOT3 Subsystems

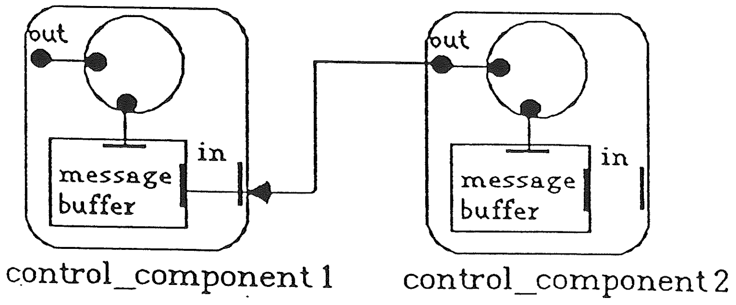


Figure 5 The Communication between Object Components

2.1 Object Oriented Design in MASCOT 3

The next three phases, the development of the abstraction objects, view and the end-user interface components may then proceed in parallel, each following an iterative design cycle. As these phases near completion, they may be combined to give more realistic prototypes, eventually contributing to a finished system; for example, the view components may initially be static screens, but once the system is running they may be linked to it to provide a dynamic representation.

It is convenient to develop separate user interfaces for the end user, the system developer and the human factors evaluator to facilitate the iterative development approach. Hence even where an object has no view representation to the end user, a view component might still be required for the viewing of diagnostic information.

3. THE MASCOT DESIGN METHOD

The MASCOT 3 design method (Simpson 1986) is based on the notion that we can represent the structure of a concurrent system in terms of a small number of system elements:

1. Activity - essentially a process representing a single thread of control
2. Intercommunication Data Area (IDA) through which the activities communicate.

There are two main types :

(a) Channel this forms a unidirectional pipe through which data flows between activities. This data is "produced" by producer activities and is " consumed" i.e. destructively read, by consumer activities.

(b) Pool this data is more permanent and may be read any number of times by reader activities before being updated by writer activities

3. Access Interface -the interface between the activity and the IDA, connecting a port in an activity to a window in an IDA. Figure 3 gives examples of the graphical representation of these elements.

3.1 Communication between object components

An object in the system can be represented by an activity and IDA's gathered together in a subsystem as shown in Figure 4. Each object subsystem contains three component objects, an abstraction component representing the real object, a control component and a display component.
Within an object subsystem, the components communicate through defined channels. The abstraction component contains the variables that represent the object's state and a set of allowed methods or operations that can be invoked to change the state of the object. These methods are invoked by the controller in response to an external stimulus to supply information or to alter the object's state. The display component may need to interrogate the abstraction component to discover its current state prior to outputting a representation of it, but it should not have access to methods that themselves change its state.

3.2 Communication between object subsystems

Communication between object subsystems is carried out through peer-group protocols between control objects and display objects.

The communication channels are shown in MASCOT 3 as access interfaces connecting ports and windows between subsystems. For maximum flexibility and to enable the distribution of processing between objects, a message passing protocol is implemented. Each object component within a subsystem contains its own message buffer and can send and receive messages and replies (see Figure 5).

3.3 The User Interface Objects

It is commonly accepted among human factors practitioners that an early focus on users is good design practice, and that

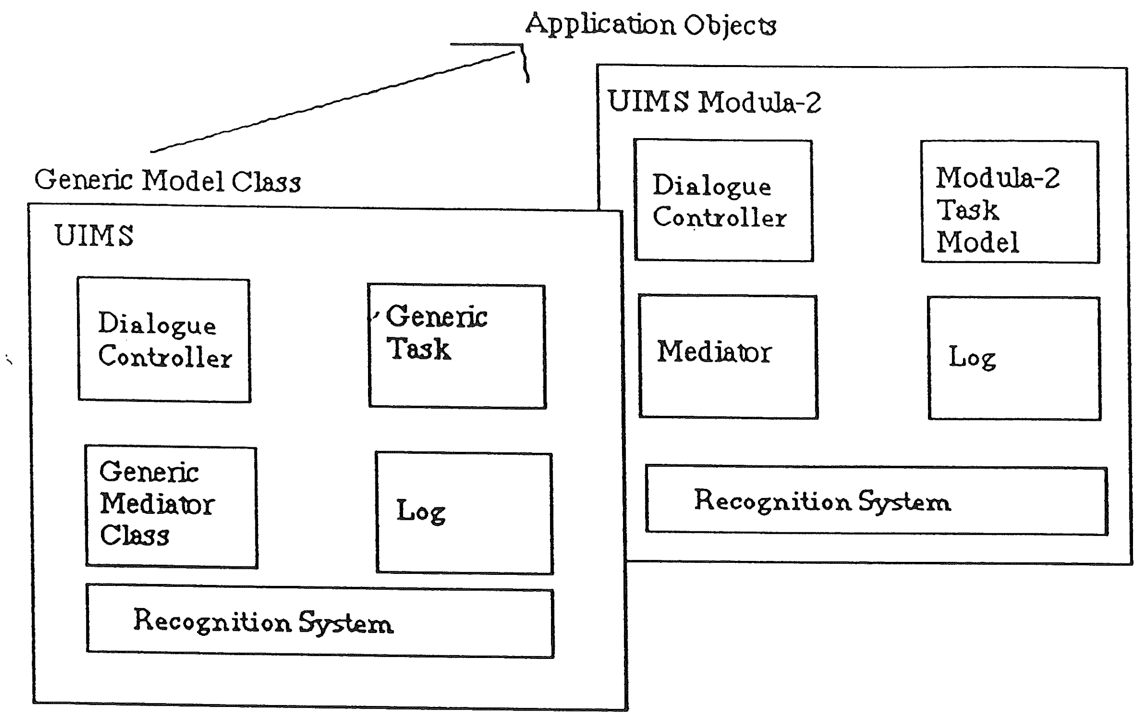


Figure 6

```
CreateProgram =
( [70] Heading ->([90] Declarations   -> ([95] Body
                    [5] Other)

                [10] Other)     -> General

  [10] Declarations ->  ([50] Heading -> ([90]Body
                                          [10] Other)
                         [50] Body    ->
                                         ([80] Heading
                                          [20] Other))                -> General

  [10] Body         ->  ([60] Declarations          -> ([80] Heading
                                                      [20] Other)

                         [40] Heading               -> ([80] Declaration
                                                      [20] Others)   )  -> General

  [8]  WriteComments   -> CreateProgram

  [2]  Other            -> General
```

Figure 7a CSP Description of a Modula 2 Programming Task

```
Start: Scroll(25)
                [] Search(25)
                [] Up(20)
                [] Down(20)
                [] Page(5)
                [] Other(5)

Scroll:     scroll up (50) -> stop -> Start
            [] scroll down(50) -> stop -> Start

Search: search ->(string(60)  [] line number(40)) -> Start

Up:         up ->{ number(.6) -> {  up (30) -> Moreup
                                    [] down (30) -> Moredown
                                    []right (15) -> Right
                                    [] left (15) -> Left
                                    []other (10) -> Start }
              []up(35) -> Moreup
              []other(5) -> Start}

Moreup:     up(30) -> Moreup
            down(30) -> Moredown
            right(15) -> Right
            left(15) -> Left
            other(10) -> Start
```

Figure 7b. Extract from a navigation task model

```
MODULE FirstProgram; (* Header Section *)
FROM InOut IMPORT WriteLn,WriteString,WriteCard, ReadCard;

VAR(* declaration section *)
 first,second : CARDINAL;

BEGIN(* main body *)

 WriteString ("Input two numbers ");
 ReadCard(first);ReadCard(second);
 WriteString(" The sum is = ");
 WriteCard(first + second);WriteLn;

END FirstProgram.
```

Figure 8

```
Begin words
CONST
TYPE
VAR


the Set of Standard types
=
:
the set of identifiers generated so far
the basic editing functions provided by the underlying editor

End words
;
BEGIN
```

Figure 9

an iterative design approach will be necessary to develop a good user interface (Gould & Lewis 1985). An advantage of the object-oriented approach is that the user interface can be developed as a separate object subsystem which communicates with the other objects through protocols as defined above. We would envisage a separate interface for the systems developer which provides access to the internal state of all the objects via their view windows and one for subsequent evaluation which allows access to diagnostic information.

4. OVERVIEW OF THE DESIGN OF THE UIMS

In view of the rapid progress being made in the design of speech recognition hardware, the UIMS must be capable of adapting to maximise the usability of ever more sophisticated recognition systems. Our initial work has been carried out with a Votan VPC 2000 isolated word speech recognition unit which utilises a working set of only 64 words, the challenge for the UIMS being to provide the unit with the most likely 64 words against which to match an utterance.

Whereas more sophisticated systems can cope with a working vocabulary of 1000 or 5000 words, and the newer breed of phoneme recognisers provide a potentially unlimited vocabulary, we maintain that the recognition rate in any system will be improved if we can accurately predict a limited vocabulary to be considered at a particular time.

Since the current vocabulary of the user is restricted by the current task in hand the problem of prediction becomes one of accurately identifying the current task, and the probable strategy within that task which is being employed. The performance of the underlying speech recognition system will affect the overall recognition rate, but does not change this premise; however the granularity of task analysis will need to be finer to cope with a system that deals with sets of 64 words rather than one that can cope with 1000 words at a time. In the case of phoneme recognisers there is no artificial limit imposed on the number of words considered at any time, it can be the exact number that are relevant in the particular task domain.

5. A GENERIC TASK MODEL

A generic UIMS will be developed from which application specific systems will be generated, these will be based initially on a generic user model, but will tailor themselves to the individual user.

The main components of such a system, shown in Figure 6, are: the dialogue controller, the generic task model, the generic mediator, the use log and the recognition system. Each of these is represented by a high level object, and communication between the objects is achieved by message passing. An instantiation of the UIMS for a particular application will inherit some of the characteristics of the generic model, but will build a task model specific to the application, instances of which will adapt to individual users.

5.1 The Object Class Generic Task

The generic task model encapsulates a structure for representing the task syntax of an application, and the strategies for achieving various goals and sub-goals within it. It will generate instances, or Application Task Models, which capture the task syntax for the particular application.

5.2 The Object Class Recognition System

This incorporates the interface to the speech recognition unit and a generic speech model. An instance of the speech model encapsulates the physical characteristics of the user's voice. Initially based on a generic user model it is capable of being automatically adjusted to provide the best possible performance for a given user. Parameters such as the recognition threshold and the input gain would be contained within an instance of this object.

5.3 The Object Class Dialogue Controller

This provides the mechanism by which the UIMS communicates with the user. It is totally separate from the underlying application, typically being presented as pop-up windows offering recognition error recovery information or user input to the recognition system in the form of new words or changed parameters. In our example we assume only a single instance of this object, although of course it is possible to envisage different instances for different applications or different users, for example a changed presentation to cope with a particular screen layout for an underlying application, or a changed dialogue to enable users to access the system via a telephone link.

5.4 The Object Class Mediator

This provides a mediator or control function by presenting information to the other objects and deciding on the basis of their responses whether to accept a particular user input or whether to invoke an error recovery routine.

Typically, an error recovery routine would be invoked if the mediator model could not determine (with a particular level of confidence) what the user had said. They might be asked to repeat the word, to select it from a list of alternatives, or even to spell it so that it could be added to the existing vocabulary. Factors contributing to the level of confidence would be not only the recognition threshold but also the probability that a particular word is likely to occur in the current task or sub-task being undertaken by the user.

5.5 The Object Class Log

The use log object contains a record of all user system dialogue. It may be interrogated to provide a recent history of use, and it may itself record statistics on error rates and recency and frequency of use of various words.

6. AN EXAMPLE APPLICATION : A Modula-2 Programming Environment

The starting point for an interface for a particular application is the instantiation of the Generic Task Model which provides a view of the typical types of tasks carried out in the application. This application task model should contain high level task information as well as more specific information about recency and frequency of task and word usage. This may be initially obtained by observation for input to a static model, but where possible usage should be directly logged by a diagnostic program which is invisible to the user. The user task model can then be updated dynamically in use.

A Modula-2 Programming environment consisting of an editor, compiler and linker provides an excellent vehicle to investigate the effectiveness of the approach since the tasks of editing, compiling, and linking are sufficiently complex to allow the method to illustrate its predictive features. However, the Modula-2 language and the development of a program in the language provide a set of words which are sufficiently restricted to allow full experimental speech driven systems to be built and tested.

6.1 Initial Subtask Configurations

The UIMS applies the rules from the Generic Task Model to create an application task model using an initial set of predictions. These initial predictions are developed from observations of a number of users carrying out the task; in this case the development of a Modula-2 program. This would typically involve the synthesis of data obtained from a high level task analysis of a number of representative users, using a method such as TKS (Johnson et. al. 1988), coupled with a low-level keystroke analysis. For example these observations may lead to a high-level task plan as shown in Figure 7a. The notation used is a form of CSP used by Alexander (1987) with the addition of percentage probabilities that each event will occur; a statement of the form:

```
x =    ([40] y
         [60] z) -> p
```

should be read as "From state x, there is a 40% probability of y occurring and a 60% probability of z occurring; after one or the other has taken place, the only event that can occur is p" The absence of a number thus indicates 100% probability.

The initial tasks shown for CreateProgram represent the different ways that people have been observed to begin the task of creating a new program. In each case the user may follow a well-defined strategy such as "create heading, write the declaration section, write the program body" or they may deviate from this as indicated by the event Other. It is not possible to map all possible strategies, but only the most likely ones. Once a deviation from plan is detected the mediator model assumes a General state from which it will try to match the beginning of a new sub-task from the application. For Modula-2 such a General state will include events for all Modula-2 statement formats as well as all the events from CreateProgram that have not yet taken place. In addition to the Modula-2 Task Model, a model of the editing tasks is required, as an example an extract of a navigation task model developed for a speech-driven editor is shown in Figure 7b.

The nature of Modula-2 statements are such that predictions are reasonably easy to make e.g. an IF statement is followed by an expression, and techniques normally used in compilers can be applied. This means that an IF statement sub-task object may create another sub-task to handle the expression and then return to the IF statement subtask to complete it.

As soon as the beginning of a sub-task is identified, the user will be assumed to be following that sub-task's plan until a deviation is detected, in which case it is back to the General state.

The tasks involved in creating the simple program in Figure 8 are CreateHeading, CreateDeclaration, CreateBody and WriteComments. The CreateBody task contains subtasks such as CreateWriteStatement, and CreateReadStatement.

Each task is associated with a set of words which represents the words most likely to occur during the carrying out of that particular task, including those words that would normally be expected to begin and end the task. Figure 9 gives a set corresponding to the program declaration section.

Dependent on the capability of the recogniser this task may, if necessary, be broken down further into sub-tasks with more restricted word sets. For each word in the task set we can associate a probability that that word will occur and a corresponding sub-task set to be created when it is recognised.

6.2 Dynamic Subtask Configurations

As the user develops more and more programs the prediction model updates the probabilities to suit the new user. Unused paths in the parent model will be discarded and new paths developed; in this way a task model specific to the user is developed gradually and the performance of the system should thereby improve.

An example MASCOT 3 design for this environment and a description of the communication between objects is presented in Appendix 1.

7. CONCLUSION

In this paper we have described a generic task model approach to the design of the user interface management system for an intelligent speech driven interface to a computer application. The use of an object oriented approach allows the separate development of the various models in the system and facilitates the creation of user-dependent instances where applicable.

The design of a simple application with a well known structure, i.e. a Modula-2 programming environment, has been considered in the first instance to provide a learning experience.

At the moment we have implemented a version of the UIMS in C running on a PC, and have developed a task model for Modula-2. We are currently investigating implementations in Ada and C++ which will allow us to more fully represent the object oriented design.

This approach can be extended to a consideration of other applications which require less restrictive word sets and more complex task strategies. We are also working on a task model for word processing legal business letters, based on an analysis of Wordperfect users (Hewitt et al. 1990).

REFERENCES

Alexander H. (1987) "Formally-Based Tools and Techniques for Human-Computer Dialogues", Ellis Horwood.

Booch, G. (1983) "Software Engineering with Ada", (Benjamin-Cummings 1983).

Cox B.J. "Object Oriented Programming An Evolutionary Approach", (Addison Wesley Publishing Company. 1987).

Frank R.J. & Hewitt (1990). "An Object-Oriented Design Method for a Real-Time Control System", in Simulation and the User Interface, Taylor and Francis Ltd. Chapter 5.

Hewitt J.A. & Furner S. (1988). "Text Processing by Speech: Dialogue Design and Usability Issues in the provision of a System for Disabled Users", in People & Computers IV, Jones D.M. & Winder R. (eds.)

Hewitt J.A. and Frank R.J. (1989)"Software Engineering in Modula-2 - an Object- Oriented Approach", Macmillan.

Hewitt J.A. and Zajicek M. (1990) "An investigation into the use of Error Recovery Dialogues in a User Interface Management System for Speech Recognition", Interact '90 Cambridge August.

Hewitt J.A. Halford P.G.R. & Zajicek M. (1990) "Improving the Usability of systems with non-standard input (speech and switch mode) by utilising a task model. presented at "Interacting with Computers: Preparing for the Nineties", Noordwikerhout, Dutch Ergonomics Society.

Johnson P., Johnson H, Waddington R, & Shouls A, (1988) "Task Related Knowledge Structures : Analysis, Modelling and Application", in People & Computers IV, Jones D, & Winder R (eds) Cambridge University Press.

Gould J.D. & Lewis C. (1985) "Designing for Usability: Key Principles and What Designers Think" Human Aspects of Computing. Henry Ledgard (Ed) ACM.

Simpson H.R.,Jackson K.L. "Process Synchronisation in MASCOT ",The Computer Journal,Vol 22,No 4 1979.

Simpson H.R., "The MASCOT Method" ,IEE Software Engineering Journal,Vol 1,No 3, May 1986

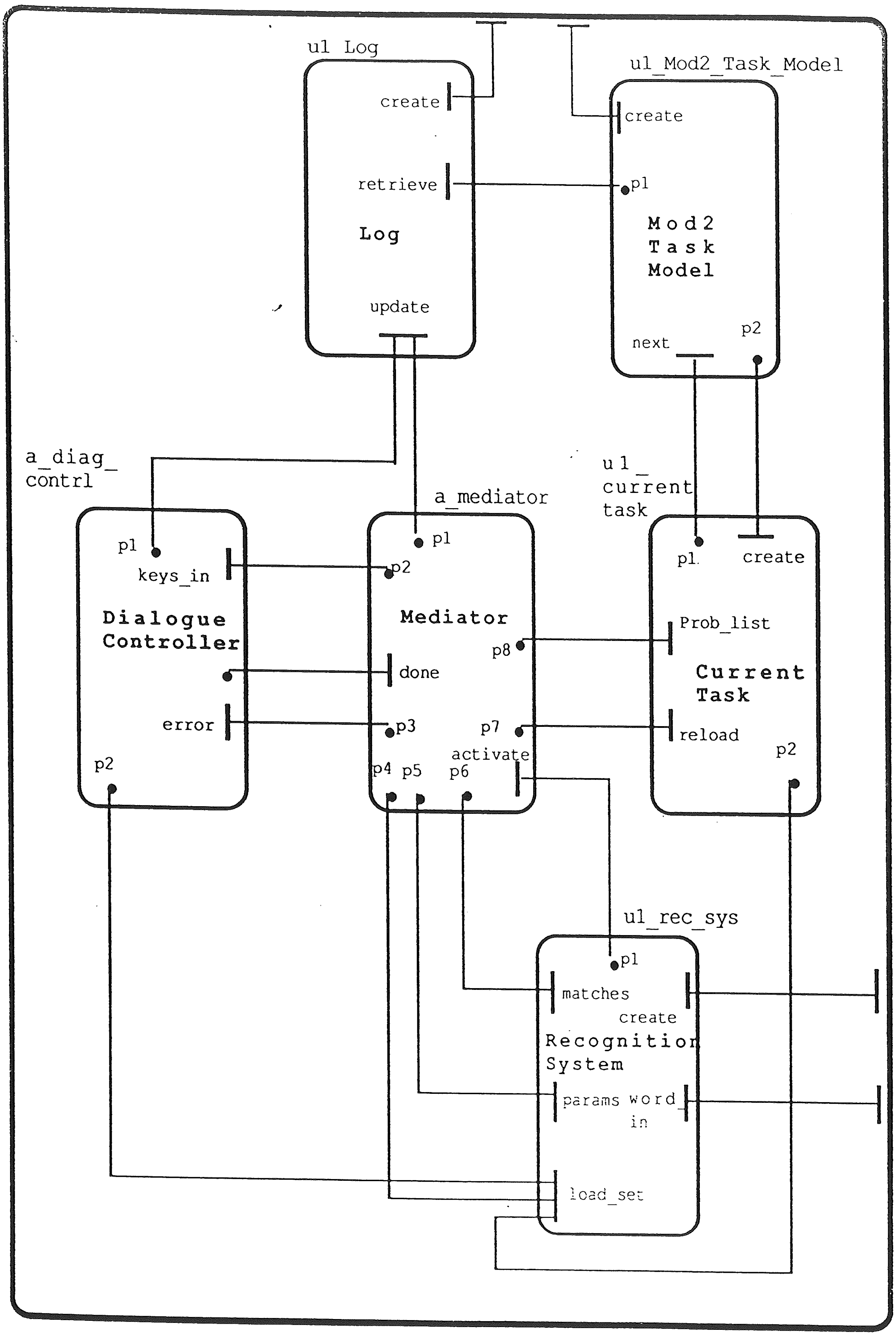
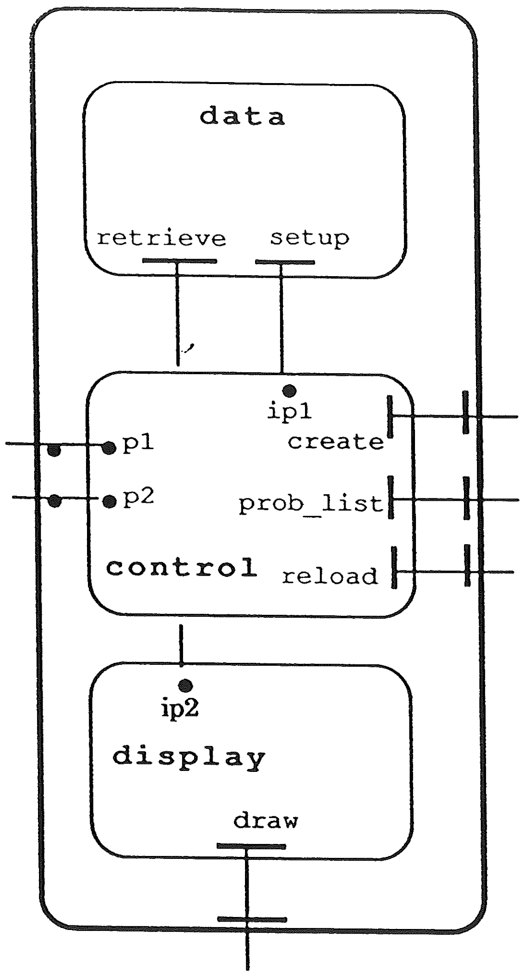


Figure 10 The Top Level UIMS Design

An example of one of the object classes of the above design is shown in Figure 11.



Windows

retrieve	->	returns current state of the object
setup	->	sets up current task information
create	->	creates a new current task object
prob_list	->	returns list of most likely words
reload	->	signal to reload the current word set
draw	->	outputs the current state of the object

Ports

p1	gets next set from parent task model
p2	loads words for current task set into recognition system
ip1	sets up the data for this object
ip2	retrieves the current state of the object

Figure 11 The Current Task Object Class

Example of the messages passed between objects when setting up and beginning a Modula-2 program creation activity for a user (u1)

```
Message Passed
     from           ->to                    Method

1.   System         -> Log                  create (u1_Log)
2.   System         -> Mod2 Task Model      create (u1_Mod2_TM)
3.   System         -> Recognition System   create (u1_rec_sys)
4.   u1_Mod2_TM                             -> Current Task
     create (u1_current_task,task_set)
5.   u1_current_task                        -> u1_rec_sys
     load_set (start_task_words)

6.   REPEAT
6.1  u1             -> u1_rec_sys           word_in (spoken_word)
6.2  u1_rec_sys     -> a_mediator           activate
6.3  a_mediator     -> u1_rec_sys           matches
6.4  a_mediator     -> u1_rec_sys           params(any_warning_signs)
6.5  a_mediator     -> u1_current_task      prob_list
(most_likely_words)

6.6  IF a good match is found by a_mediator
     THEN

6.6.1               a_mediator              -> a_diag_contrl
     keys_in(keystrokes)

6.7  ELSE  invoke ERROR_RECOVERY_ROUTINE

     WAIT for
     a_diag_contrl -> a_mediator            done
     END

     UNTIL u1 is finished


7.   ERROR RECOVERY ROUTINE

7.1  a_mediator     -> a_diag_contrl        error(details)
7.2  a_diag_contrl                          -> u1_rec_sys
     load_set(error_handling_words)

7.3  REPEAT
     u1             -> u1_rec_sys           word_in(spoken_word)
     u1_rec_sys     -> a_mediator           activate
     a_mediator     -> u1_rec_sys           matches
(list_of_matched_words)

     IF a good match is found by a_mediator
     THEN
     a_mediator     -> a_diag_contrl        keys_in(keystrokes)
     ELSE invoke ERROR_RECOVERY_ROUTINE
     END
     UNTIL done
```

8. PARALLEL ACTIVITIES

8.1. Getting the next task set

```
IF current task is finished
THEN
u1_current_task                      -> u1_Mod2_TM
next(task set)
u1_Mod2_TM                           -> u1_current_task
create(u1_current_task,task_set))
END
```

8.2. Updating the Log

```
a_mediator   -> u1_log               update (list of updates)
a_diag_contrl                        -> u1_log
update (list of updates)
```

8.3. Updating the Application Task Model

```
u1_Mod2_TM                           -> u1_log
retrieve (latest updates)
```

Description

The controlling System sets up a run-time UIMS (steps 1-3) with the Log, Mod2 Task Model and Recognition System specifically tailored for this user (the Mediator and Dialogue Controller are assumed constant)

A set of words for the expected starting task is generated (step 4)
The words corresponding to the starting task are loaded into the recognition system (step 5)

The system is now ready for use and is activated when the user speaks a word into the microphone (step 6.1). The recognition system activates the mediator (step 6.2) which retrieves a list of likely spoken words from the recogniser (step 6.3) along with an update of the speech parameters (step 6.4) in case any warnings to the user are needed (e.g. "speak louder please"). The mediator asks the current task object which are the most likely words in the current context (step 6.5).

If the mediator considers a good match has been found for a spoken word, it passes the keystrokes corresponding to it to the dialogue controller (step 6.6.1) which passes them on to the application, no further action is needed until another word is spoken.

If the mediator does not find a good match for the spoken word, it will pass a message to the dialogue controller telling it to invoke one of its error recovery routines (step 6.7). In this case, the dialogue controller will load the set of words needed for error recovery (often just "yes" and "no") into the recognition system (7.2)

Some activities can be carried out in parallel:- the retrieval of the next task from the application task model (8.1), the continuous updating of the use log (8.2) and the subsequent updates of the application task model (8.3).