Pre-publication draft of a paper which appeared in Ercoli, P. and R. Lewis (Eds.), *Artificial Intelligence Tools in Education: the IFIP Tc3 Working Conference 1987*, pages 203-219. North-Holland, Amsterdam

OBJECT-ORIENTED CONTROL FOR INTELLIGENT COMPUTER ASSISTED LEARNING SYSTEMS

Tom Richards
Computer Science, La Trobe University
Geoff Webb
Computing and Information Studies, Griffith University
Noel Craske
EDP Department, Chisholm College

Abstract

This paper investigates an approach to providing a general-purpose authoring/tutoring shell for intelligent computer assisted learning systems. The approach is to outline an object-oriented representation of task/goal hierarchies, then to consider the ways in which domain expertise, student information and teacher expertise can be made to interact with such hierarchies. The result is a skeleton in terms of which exploratory lessons are being constructed; and in terms of which further research on the domain expert system, student modelling, educational expertise modelling, and user interfacing can more concretely be developed.

Keywords

Object oriented programming; intelligent tutoring systems authoring systems; computer assisted learning; frames; student models; teaching strategy.

1 INTRODUCTION

Most current work on the construction of Intelligent Computer Assisted Learning (ICAL) systems has concentrated on one-topic systems: diagnosing faults in a particular electrical circuit - the SOPHIE system (Brown, Burton and De Kleer, 1961), solving the WUMPUS game (Goldstein, 1981), and ACE, which discusses the analysis of nuclear magnetic resonance spectra (Sleeman and Hendley, 1981). These systems and others like them were not intended as vehicles for widespread educational use; rather, they are vehicles for investigating and solving important research problems: natural-language communication, modelling the evolution of procedural knowledge in a learner, and dialogues to monitor and guide student reasoning, respectively. Those are problems that need solving first if we are to use artificial intelligence effectively in education.

In the EXCALIBUR Project, by way of contrast, the authors and others are investigating what is involved in providing a general-purpose ICAL system. The search is for a system which provides an authoring and knowledge engineering shell, analogously to a standard expert system shell, and a delivery system for students (Richards, 1986a). Such a system is intended to be general-purpose in that the architecture and knowledge representation methods used should involve no specific assumptions about the content or nature of the subject domain to be taught, nor about the modes of learning or presentation (instructional, student driven, exploratory, and problem-oriented, etc) to be used.

To use such a system, a person, whom we will call the author, will interact with the system to create what, for lack of a less pejorative word we will call a tutorial - a program which can then be run by a student with the aim of learning from his or her interaction with it. We stress that it is up to the author to decide the extent to which the student's interaction with the tutorial is to be constrained by (for instance) predetermined delivery sequence and student assessment results, in which case the author will need to specify instructional modes or rules to the system; or to which the student may treat the tutorial as an educational information kit and tool bag, in which case the author will need to specify what resources and modes of exploration are available to the student. The point is that a general-purpose ICAL system should place little constraint on the educational philosophy the author wishes to adopt.

The question is: how might one build such a general-purpose system? The main problems we have identified, for which solutions must be found that do not compromise the sought-after generality, are:

a. How to store data on the subject domain of the tutorials, and how to perform analyses on that material in ways relevant to the learning process at hand (designing the Domain System facility, which in many cases will amount to an Expert System or Knowledge Based System);

b. How to construct a knowledge base about the student, including what the student believes and has learnt in the domain of the tutorial, and why and how. This facility is called the Student Model.

c. How to incorporate rules about student knowledge and performance criteria that can be used to tailor or guide the presentation of the student session in the light of information gathered about the student so far. This is the Mentor facility (Sussex, 1987).

d. How to unify these under a System Driver which handles authoring sessions as well as student sessions.

The Domain Information System is not necessarily the same as the material which the student may learn. Often, it may be much more extensive, for the same reason as a teacher usually needs to know far more about a subject than s/he will teach, just in order to handle the teaching process well. In the case of an ICAL system, the Domain Information System may often need to contain intelligent subsystems enabling the computer to solve problems whose solution is needed for the tutorial interaction, but whose solution method is either far too advanced for the student or is a computer-oriented process that can be quite confusing to a human. Crudely put, machines and people do not think alike. Examples are:

- The parsing of a given French sentence in a French grammar tutorial, in order to generate the specific grammatical and pragmatic information the student needs for the interaction about that sentence. Machine parsing methods are usually totally different from those used in a French language tutorial class, even if the end-products (parts of speech, agreement relations, etc.) are much the same.

- The checking or production of a proof step in the derivation of a proof in a tutorial on mathematics or formal logic. A student may, for example, be learning one particular method of doing formal logic, such as natural deduction; yet the tutorial system will use a pattern-matching procedure which is not in the least educationally relevant, to check the students proof steps.

We cannot easily do without the special expertise of the Domain Information System. If we did away with the French parser, then an author could provide the parsing information needed by the student for each stored sentence; but then that information could not be generalised to cover sentences typed in by the student. In the logic example things are worse, for proof checking information cannot in principle be provided by the author: for non-trivial derivations there can be an infinity of correct "next steps". Only a special proof-checking procedure will do the job.

We turn now from considerations about the material to be learned, to questions of how to learn it. In French grammar, logic, and indeed courses on most topics there may be many modes of learning. The student might be, for example:

- Working through a set system-controlled sequence of actions;
- Freely exploring an environment containing knowledge and techniques relevant to the subject domain;
- Trying to devise his/her own techniques for reaching a goal - and testing them.

The more learning modes such as these that are made available to the student, the more demands are put onto the content and problem solving power of the underlying Domain Information System. But even more stress can be put onto the other three facilities, the Student Model, the Mentor, and the System Driver.

In the rest of this paper we will look at a method of designing and unifying these four facilities. The result is the specification of an architecture for a general-purpose ICAL system.

2 GOAL/TASK HIERARCHIES

One place to start in trying to analyse this problem is the idea of Goal/Task hierarchies (see Fig. 1.)

A tree-structured Goal/Task hierarchy has at its root the overall goal of the current tutorial, and the (immediate) dependants of any node are the sub-goals that need to be achieved in order to achieve the node's own goal. On each node is a task, which to be carried out requires that the tasks on the dependant nodes have been executed. The terminals of the tree are initial goals which have no prior goal; tasks which can be executed without performing prior tasks. Formally, the links from a node N to its dependant nodes N1, N2 . . . . , mean that the tasks on the dependants are individually necessary and jointly sufficient for the execution of the task on N; so a task hierarchy is an and-tree, and a link from N to a dependant node N1 means that achieving the goal on N1 is a necessary condition of achieving that on N. That is to say, in order to achieve the goal on N it is (logically, pedagogically, epistemically, or . . .) necessary to achieve, or to have achieved, the goals on N1, N2; and when they have been achieved it is (logically, pedagogically . . .) possible to achieve the goal on N by doing the task on N. The relation between the goal and the task on a node is simply that the goal is what has been achieved when that task has been executed, given that the sub-goals on N1, N2 . . . . , have been achieved. Any tighter definition of the linkage relation in a goal/task hierarchy is not really a matter for the system designer, but for the author of a tutorial, who will wish to exploit it as a control structure that may be adapted to his or her ideas about the flow of the tutorial.

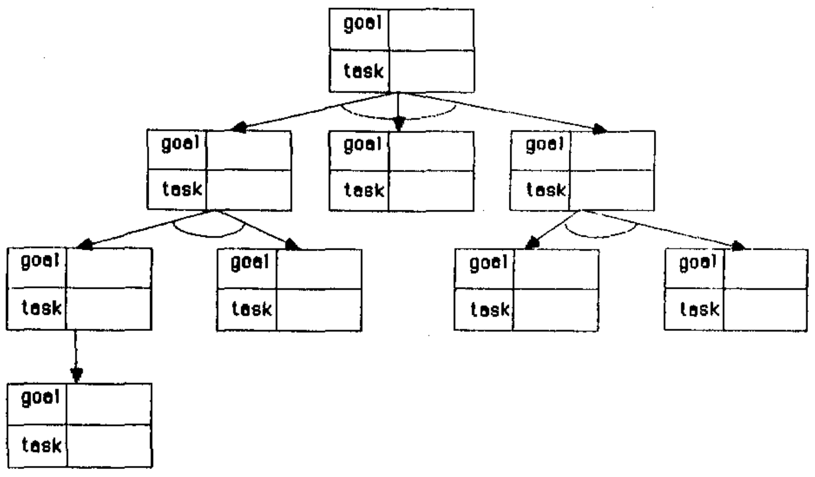

Figure 1: A tree-structured Goal/Task hierarchy

In Figure 2 we have a small example of a Goal/Task hierarchy for detecting subject-verb agreement in number, which illustrates the points just made.

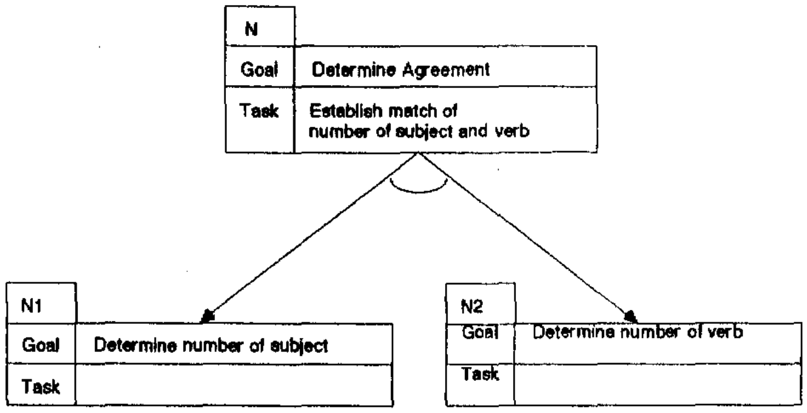
N
Goal
Determine Agreement
Task
Establish match of number of subject and verb
N1
Goal
Determine number of subject
Task
N2
Goal
Determine number of verb
Task

Figure 2: A task/goal hierarchy for subject-verb agreement

The knowledge needed to determine agreement of subject and verb (the goal on N) is the gender and number of the subject (left-hand sub-goal) and the gender and number of the verb (other sub-goal). When they are known, a comparison of them to see if they are the same (task on N) suffices to reach the goal on N.

S1		
OwnSlot	**aSentence**	
	Sentence	The girls were running
	Parse	*Parse data*
	Subject-number	Plural
	Verb-number	Plural
	Subject	The girls
	Verb	were running

N1	
Goal	Determine number of subject
StudentTest	(=(response-to "What is the number of the subject of this sentence" (subject-number CurrentSentence))
StudentRecord	(if(=StudentTest true) then (put CurrentStudent Goal passed) else (put CurrentStudent Goal failed))

Figure 3: A Record or Object S1 holding an example sentence and grammatical information about it, and a corresponding Goal/Task Object N1.

Italic text is a description of data not the data itself. Parenthesised text is Method code.

Sentence		
InstanceSlot	**aSentence**	
	Sentence	
	Parse	*Parsing function returning parse data*
	Subject-number	(subject-number-of Parse)
	Verb-number	(verb-number-of Parse)
	Subject	(Subject-of Parse)
	Verb	(Verb-of Parse)

Figure 4: A type Object for sentence data. Note that S1 of Fig. 3 is an instance of this Object

Suppose now a student is attempting to use an ICAL program based on this Goal/Task hierarchy to learn about subject-verb agreement. We imagine that the student is presented with a series of examples - French sentences that may or may not obey the rules about agreement - and is required to move through the hierarchy with each example. (The pedagogical effectiveness of this approach is not up for discussion at present: we simply need an example to motivate this description of our system.) Then on each node we need to record a method of determining if the student is correct in his/her task solution. One way to do this might be to look up the answer as entered by the tutorial author on a record containing the current sentence example (S1 of Figure 3 is a record containing the needed data). More intelligently, such a frame might be created by a parse procedure belonging to the underlying Domain Information System (Figure 4).

Figure 3 contains (sketchily) three types of control (or, if you prefer, information):

- Domain information, as in the record S1;
- Tutorial driving procedures, as in the StudentTest slot of N1;
- Student modelling services, as in the StudentRecord slot of the frame N1

Each of these three control regimes may embody a great deal of information, procedures, and the like, but as Figure 3 suggests, each regime intersects with a Goal/Task record, such as N1, in a simple way. The architectural design of the system is based on this, bringing us to the central design issues of the system

3 OBJECT HIERARCHIES AND TYPE-SYSTEMS

In Object-Oriented Programming Languages (OOPL's: see Byte, 1986; OOPSLA, 1986) the entity that does the work is not a procedure, as in procedural languages, but an Object (with a capital O to avoid ambiguity) containing local data (like Pascal records) and procedures (called Methods), which are accessed and activated by Messages (like command lines) sent by one Object to another, and returning values. Objects are similar to the sets and individuals of set theory, in that one Object can be a member (usually called an instance) of other Objects; or it can be a subset (subtype) of other Objects. For instance, an Object containing information about the behaviour of the French verb 'voir' will be an instance of another Object containing generic information about the behaviour of transitive verbs. That Object in turn will be a subtype of another Object containing generic information about the behaviour of verbs in general. Note the difference between an instance and a subtype: 'voir' is an instance of a transitive verb, and of a verb. It is not a subtype of either. And transitive verbs, as a type, are not an instance of verbs; they are a type of verb.

Subtype relations form a hierarchy (see Fig. 5), with instances as the terminal nodes of the hierarchy tree.

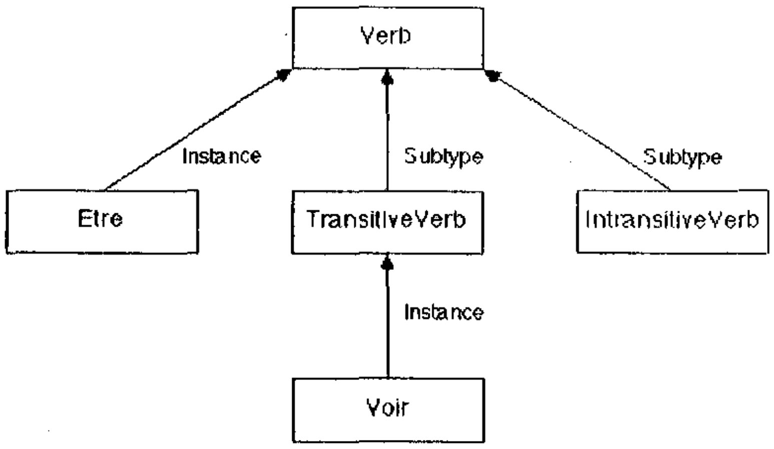

Figure 5: An inheritance hierarchy of Objects

Objects in the tree normally inherit properties belonging to any higher Object; for instance if Verb in Figure 5 recorded that all verbs have subjects, that would hold of transitive verbs and intransitive verbs, and all instances ‘etre’ and ‘voir’ as well. So if a query were addressed to this tree: “Do transitive verbs take subjects?” the TransitiveVerb Object would receive this query message and answer “Yes.” If the lesson designer wished to treat the verb ‘etre’ as an exception to the rule about subjects, s/he records on the Etre object that it does not take a subject, and the inheritance of the contradictory property from Verb is blocked. The query “Does ‘etre’ take a subject?” would go to the Etre Object, which would directly answer “No,” without consulting further up the hierarchy. In the same way procedures can be stored on the appropriate Object, and get consulted by lower objects unless overridden by a lower procedure. The procedure for finding a verb’s subject belongs on Verb, and the procedure for finding a verb’s object belongs on TransitiveVerb.

Objects that are instances typically record different sort of information from objects that are types. Instance Objects record information specific to the thing that Object represents, but a type Object will record generic information true of all instances of that type. The type Object StudentModel would record the pass mark needed for any student to succeed in the current tutorial; but an instance of that type, such as Student36, would record that student’s name and mark achieved.

In our ICAL design, the Goal/Task hierarchies are not Object hierarchies as described above, even though the nodes (see Figure 2 and N1 of Figure 3) are Objects. So from now on we will call them Goal/Task Objects. The hierarchy shown in Figure 2 is plainly not one of inheritance, for N1 does not inherit anything from N. The link, as remarked earlier, is that the lower Goal/Task Object contains a goal that is a necessary condition for attaining that in the higher Object. A goal, filling a slot in a Goal/Task Object such as N1, is however an instance of the type Object called Goal, which specifies the structure of any of its instances, which are of course individual goals. (See Figure 6)

(a)

Goal		
InstanceOf		
SubtypeOf	Universe	
InstanceSlot	theGoal	
	Key	*function returning unique keys*
	Description	*instances put English description here*
InstanceSlot	Goal/Task	*instances put a pointer to a Goal/Task Object here*
InstanceSlot	Subgoals	*instances put pointers to subgoals here*
InstanceSlot	Supergoals	*instances put a pointer to the supergoal here*
InstanceSlot	GoalData	*instances put methods or pointers here to obtain data about the goal*

(b)

Goal5		
InstanceOf	Goal	
SubtypeOf		
OwnSlot	theGoal	
	Key	Goal5
	Description	Determine number of subject
OwnSlot	Goal/Task	N1
OwnSlot	Subgoals	()
OwnSlot	Supergoals	Goal49
OwnSlot	GoalData	()

Figure 6: Objects and instances

Figure 6a shows a type Object called Goal containing a number of structured InstanceSlots, which describe generically the slots occurring on each token of this Object that may get created, i.e. on any individual goal Object. Goal5 is a token of Goal; and what the Goal slot of N1 in Figure 3 really contains is a pointer to Goal5. To create Goal5, the InstanceSlots of Goal are called upon to define the structure of Goal5, and so is any Object of which Goal is a subtype - in this case Universe, which is the unique upper bound of the inheritance lattice. This is the method used to create any new Object.

The OwnSlots contain data or Methods belonging to that Object itself; these slots are the ones defined by the corresponding generic InstanceSlot data on the higher Objects. One Object can access the data in another Object's OwnSlots, or activate Methods stored there, by sending it a Message, containing arguments for the Method's parameters if needed. For example, since the Goal slot of a Goal/Task Object points to some instance of Goal, Goal/Task Objects are designed so that they can send messages to Goal. For example, the pointers to Goal/Task nodes making up the Goal/Task hierarchy (as seen in Figure 2) are determined by a Goal/Task Object sending a message to the instance of Goal that its Goal slot points to. In this way, a Message from N1 to its Goal instance G5, to find G5's supergoal, will return N's goal.

Then a Message to that goal to report its subgoals will return G5's "sibling subgoals" (just one, in the example) which must also be achieved by the student before the goal of N can be attempted. Messages can now be sent from N1 to each of those subgoals, accessing the Goal/Task objects they belong to (just N2) and checking which of them the student has already achieved. This helps to determine the flow of tutorial control.

Where does the code for these messages reside? In OwnSlots in the type Objects Goal/Task and Goal, to be inherited by their instances. For example, the Method to find a Goal/Task Object's siblings resides on the type Object Goal/Task, and looks like this, in pseudocode:

```
GetSiblings:
        supergoal       := ask Goal for its supergoal Object;
        sib-goals       := ask supergoal for list of its subgoal Objects;
        siblings        := collect into a list:
                           for each sib in sib-goals:
                                   ask sib for its Goal/Task Object;
        return (siblings less self)
```

Now if N1 is sent the message GetSiblings it will return the list (N2).

This example of how a slot in an Object can point to another Object, which belongs to a particular hierarchy of Objects, illustrates an important aspect of the architecture of our design. We have identified a number of Object hierarchies, or clusters thereof, that are needed in our ICAL structure. Major ones are:

- Goal type-systems, discussed in detail above;
- Task type-systems, defining activities for the student;
- A type-system to describe the Student Models for individual students and associated diagnostic procedures;
- Logic type-system to provide rule frameworks for a logic- programming reasoning service where needed. Sets of these rules comprise the Domain Expert System, reasoners about the student's belief base, and the analysers for a natural language interface.

4 OBJECT NETWORKS

We remarked earlier that the hierarchy of Goal/Task Objects was not linked up by inheritance, so in present terminology it is not a type-system. Type-systems are used to define Objects as types, subtypes, and ultimately instances, and to provide through inheritance powerful built-in reasoning services. But the hierarchy of Goal/Task Objects is effectively a simple semantic network: data nodes joined by labelled links.

Overall, the system we are designing can be viewed as comprising several connected semantic networks or conceptual graphs (Sowa, 1984), which we will call Object networks. Major Object networks in a tutorial program are:

- The underlying Domain Information System;
- The Student Model facility;
- The Mentor facility;
- The system Driver;
- The Goal/Task network;
- Human/machine interfaces;

- Natural language processor;
- Reasoning systems, primarily for the Domain Information System if it is knowledge-based and for conducting rational dialogues with system users.

Concentrating attention on these Object networks as the architectural framework of an ICAL system makes the type-systems disappear into the background. On the other hand, if you concentrate on the type-systems, then you see the Object networks as largely derivative structures, being created by the pointers in slots of the Objects as created maintained, and manipulated by the type-systems. Look at how we found the siblings of a particular Goal/Task Object, which illustrates how the Goal type- system defined the Goal/Task Object network.

Figure 7 tries to depict, on two-dimensional paper, the multi dimensional structure of the resulting system.

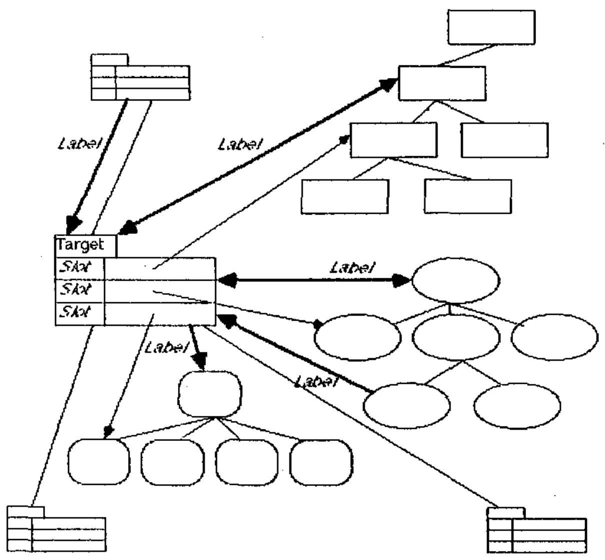

Figure 7: Type -systems and Object networks

It would be better to visualise slots (in Target of Figure 7, for instance) not as containing pointers, but containing the Objects pointed to, and to visualise each type-system as lying in a different plane or dimension. The Object networks cannot be entirely determined by the background type-systems and slot pointers; but to the extent that they are, the authoring task for a given tutorial is simplified and automated. The author is presented with ready-made prototypes for type—systems, eg. The type Objects Goal/Task and Goal, and needs to specify instances and what goes into the OwnSlots of those instances. That can be more like a data-entry task than a programming task. To the extent that Methods need to be developed by the author of a tutorial, typically in an intelligent Domain Information System, then to that extent programming or knowledge engineering is needed.

At its simplest, then, the tutorial construction procedure for an author is:

- Construct a Goal/Task network for the tutorial domain;
- Construct a Domain Information System which will provide the domain data and processing that a student (or the system) may need at each node of the Goal/Task network.

From the point of view of the system designer, who has to think about how to make the authoring process simple yet powerful, three important research goals that we are exploring are:

- To devise skeleton Goal/Task node objects that put few constraints on the type, content, or style of tutorial to be developed around those tasks, yet which allow the author to express the cognitive structure of the tutorial with a minimum of special programming or modification;
- To provide a cle an interface to the Domain Information System that allows domain information to be entered securely; and in a way that permits reasoning on that information and creates suitable output for student users;
- To develop an author/system interface that requires as little learning as possible by authors, while giving them as much power and control as possible over the system and the tutorial they are creating. To do this very high-level editors and browsers are needed, such as network displays and network editors.

To illustrate the problem of designing authoring facilities for a general-purpose ICAL, and the way in which an Object-based architecture helps, we will now look at the question of providing various tutoring strategies or methods for any given tutorial material.

5 PROVISION OF TUTORIAL STRATEGIES

The object-oriented approach allows several useful features and procedures to be developed, virtually free, as aspects of the OOPL methodology.

- The Goal/Task hierarchy can be generalised to an and/or digraph, in the manner of the Feature Networks of (Webb, 1986). This now allows us to represent any structure of necessary and sufficient conditions between nodes, which give great freedom in representing the epistemological structure of the concepts, skills, etc. being tutored.
- DoFirst and (inversely) DoNext slots on a Goal/Task Object can contain pointers to other such Goal/Task Objects, thus controlling lesson sequencing in a rigid manner. Alternatively, the Goal/Task hierarchy can be used to provide default tutorial sequencing. If the type Object for these nodes has an instruction to do first the dependants of a node (if any), then a bottom-up tutorial regime is established as the default. Or, an instruction there to do the parent first establishes a top-down regime, allowing the student to avoid doing explicitly the subtasks of a task she/he can do in one hit.
- A TutorialStrategy slot on a Goal/Task Object can contain code for an explicit tutorial strategy, to override the contents of DoFirst and DoNext slots, but accessing them if needed, typically under certain conditions laid down by the author or inferred from the student model.
- DoFirst and DoNext slots will in general create a pointer universe for the Goal/Task Objects that organises them as a directed lattice or even a general digraph, with lower bounds and upper bounds representing the first and last things to do. This can be even more complicated if these slots contain disjunctions or conjunctions of pointers. A system-provided graph search algorithm will act as a tutorial planner for the student, finding feasible paths to the root task/goal node(s)
- In general, a student will have some task to do at a Goal/Task node, even if it is only to read something. Slots on the node can specify different tasks for different tutorial strategies, or for different conditions that the student is in: slots such as Remediate, Check knowledge, TestKnowledge, Tellstudent. Access to these will typically be determined by the code on the TutorialStrategy slot. Thus, except in rather primitive lessons, the Goal/Task network need not of itself define any sort of sequencing of student tasks.
- The task(s) at a Goal/Task node can recursively be defined in terms of another lower-level Goal/Task hierarchy, effectively providing different levels of abstraction or detail at which a student might proceed with the tutorial.

- Costs can be statically or dynamically associated with nodes on the graph, reflecting difficulty of task, desirability of doing the task, etc. The graph-search algorithm mentioned above will then seek minimal-cost paths for the student.
- Any slot on a Goal/Task node can have a UserAccess? facet on it, indicating whether the user may access the information or execute the procedure in that slot. This provides a control over the openness of the tutorial material to the student, and the extent to which they can browse, select their own learning method, etc. Such a facet may of course contain code, not just Yes or No, determining the access conditions for that slot, and distinguishing between types of user, such as programmer, tutorial author, and student.
- Authoring a tutorial can be treated as one mode of use of the tutorial. This allows a student access to the authoring tools, and very open access to slots in the Goal/Task network, as well as to the Domain Information System and the Student Model system. This has the advantages of providing a uniform user interface, and not one for the author and another for the student; and of opening up a powerful learning-by-making tutorial strategy. The disadvantage is that the student could damage or destroy the tutorial by his/her modifications; so a student-as-author access class needs to be distinguished from a true-author access class.

6 STUDENT MODELLING

Here too, the OOPL approach to system design allows a lot of student information to be collected and used more or less automatically.

- Demons on a Goal/Task node can collect any type of monitoring data on student behaviour when the goal is visited. The data can then be dispatched to appropriate nodes in a student-modelling structure. Demons on nodes in that structure can then update student statistical data, revise the profile of the student, and despatch information to the blackboard that will be needed for the next stages of user interaction.
- Collected student data, either on a Goal/Task node or in the Student Model database or on the blackboard, can be accessed by message-passing to help control the tutorial or to determine how to respond to the user.
- Overlay models of student beliefs (Goldstein, 1981) are effectively generated automatically by storing student beliefs about a Goal/Task node on that node, and similarly by storing student beliefs about some item or concept in the Domain Information System database on the Object encoding that concept. This provides the raw data for analysing bugs in student beliefs (Stevens, Collins and Goldin, 1981), which are important in directing further tutorial interaction.

7 COMPARISON WITH OTHER SYSTEMS

The idea of Goal/Task nodes is found in a simplified form in the ECCLES system (Richards and Webb, 1985); although there they are arranged strictly in an or-tree whereas here any and-or graph is constructible. The point of the or-tree was that a path through the tree represented an appropriate series of tasks for the student when faced with a particular problem in the domain. Analysis of the grammar of one type of French sentence, for example, would proceed down one path, determined by the grammatical features of that type, and analysis of sentences of another type would proceed down a second path, taking alternatives appropriate to the different features of the second type. Each sentence to be analysed is flagged by a tip node in the decision tree, thus determining the path of tasks to take (See Figure 8).

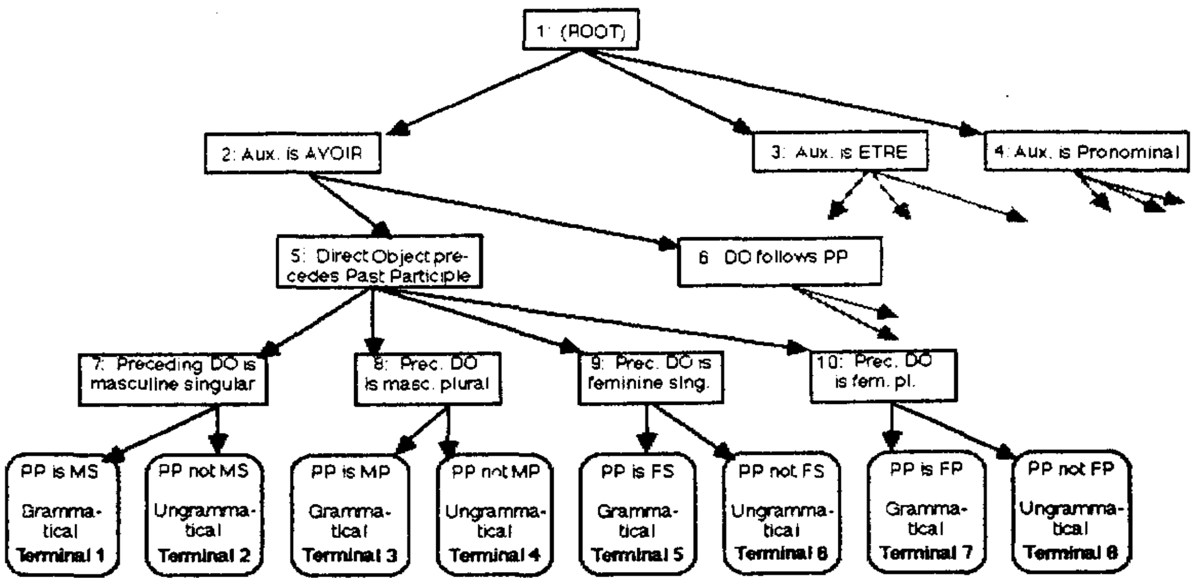
1: (ROOT)
2: Aux. is AVOIR
3: Aux. is ETRE
4: Aux. is Pronominal
5: Direct Object precedes Past Participle
6 DO follows PP
7: Preceding DO is masculine singular
8: Prec. DO is masc. plural
9: Prec. DO is feminine sing.
10: Prec. DO is fem. pl.
PP is MS Grammatical Terminal 1
PP not MS Ungrammatical Terminal 2
PP is MP Grammatical Terminal 3
PP not MP Ungrammatical Terminal 4
PP is FS Grammatical Terminal 5
PP not FS Ungrammatical Terminal 6
PP is FP Grammatical Terminal 7
PP not FP Ungrammatical Terminal 8

Figure 8: Part of an ECCLES decision tree for analysing the French 'passe compose' construction. Nodes show information gained to reach them; terminal nodes (rounded boxes) also show the path number to that node.

Also, in ECCLES, the actual tutorial interactions are determined by question-response material stored on each node. On the present model, many tutorial modes are possible, and a student is not constrained to any path through the Goal/Task network (though can be). Moreover, in the present system, if a student is "at" a particular Goal/Task node, the interaction there can be of many sorts, and be determined by processes in the Student Model, Mentor, and Domain Information System, and as outlined in the previous section of this paper. Teaching material on a Goal/Task node generally just consists of pointers to the relevant objects in the Domain Information system, or, in more complex cases, messages to the Mentor who organises appropriate material from the Domain Information system. It is important to realise that the Goal/Task Objects in the present system are mainly there as points of intersection of Objects from many type-systems, thereby acting as nodes for an Object network (the Goal/Task network) that describes epistemological relationships in the subject domain to be taught.

The Goal/Task network provides a description of the operations that students must perform to complete learning tasks from a subject domain. Often this network will map directly onto the Domain Information System - the network that describes the knowledge that the student is to acquire through the performance of those tasks. For example, in Figure 2, N1 should map directly onto a node in the Domain Information System specifying that the number of the subject of a sentence must be singular or plural. N2 should map onto a similar node relating to verbs. Finally, N should map onto a Domain Information System node specifying that there is subject/verb number agreement in a phrase if and only if the number of the subject is the same as the number of the verb.

A Goal/Task network will only map directly in this manner onto a Domain Information System network if each element of knowledge that the student is to learn is to be taught by a single operation that the student is to perform and if each operation in turn relates only to a single element of domain knowledge. There will be many domains for which these conditions do not hold.

The idea of basing lessons on explicit descriptions of the knowledge to be taught has been explored through the DABIS system (Webb, 1986). DABIS utilises a form of and/or digraph, the feature network, to represent the epistemological structure of the domain to be taught. Thus, DABIS lessons are based on explicit descriptions of the knowledge they are to teach (the Domain Information System) rather than on explicit descriptions of the teaching strategy that is to be pursued (the Goal/Task Hierarchy) as is the case with ECCLES. Figure 9 illustrates the difference. This allows DABIS to construct far better student models than ECCLES in domains for which there is not a one to one correspondence between the Goal/Task hierarchy and the epistemological structure of the domain.

However, DABIS has the shortcoming that it is difficult to specify instructional flow that differs substantially from the structure of the Domain Information System. Thus, like ECCLES, and for the converse of the same reason, DABIS operates best in domains in which the Goal/Task hierarchy maps directly onto the Domain Information System. As a result, tutorials for both systems tend to be selected where these conditions hold. This, unfortunately, tends to obscure the fundamental differences between the semantics of the underlying structures utilised by the two systems. However, as described in (Webb, 1986), these differences are both real and significant.

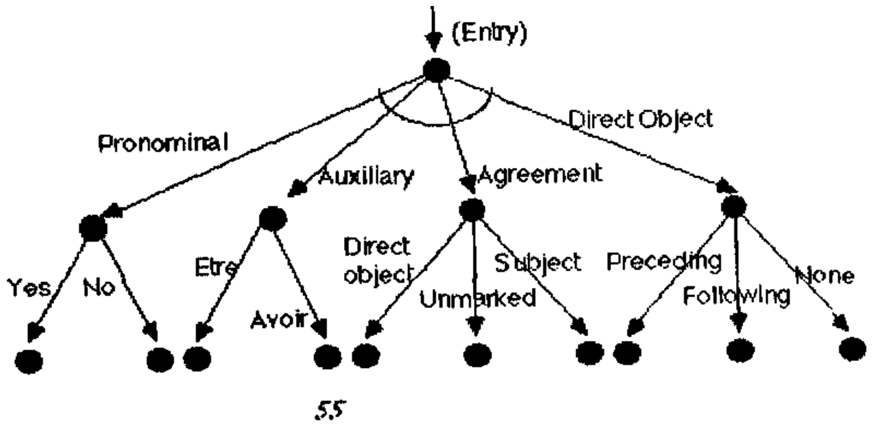

Figure 9: Part of a DABIS feature network for the French 'passe compose'. This covers much the same features as the ECCLES tree in Figure 8. The loop joining the links coming down from the top node indicates that it is an AND-node: all four subtrees below it must be visited, not just one.

The present project, EXCALIBUR, can be seen as unifying the ECCLES and DABIS approaches to allowing the explicit description of both a Goal/Task network and a Domain Information System within the one system. What EXCALIBUR adds to the other systems is:

- explicit Goal/Task networks (missing from DABIS);
- explicit knowledge structures (missing from ECCLES);
- the allowance that student paths through the Goal/Task network need not be constrained by the necessary-condition vectors of that network, but by pointers in DoFirst and NoNext slots in the Goal/Task Objects;
- the provision an Object-oriented architecture for the construction of ICAL lessons around the Goal/Task networks.

The reason for treating the Domain Information System separately is just that in general a lot more information and reasoning ability is needed in a teacher than the curriculum they teach (see the Introduction, above.) However, the Domain Information system can, in the Object-structured architecture of the present system, be coupled as closely as needed with the Goal/Task network. In some cases it could be dispensed with entirely; as where the tutorial author provides the necessary parse data for the problem French sentences.

8 CONCLUSION: FURTHER RESEARCH

The design of a suitable architecture is just the starting point of the EXCALIBUR Project. Current active research by the authors consists of:

- building small tutorial systems along the above lines, as different in subject area and tutorial approach as possible, as test beds for the design of Goal/Task hierarchies and nodes, and of the various Object networks and their interaction (Richards, 1986b).
- abstracting from those cases to an adequate common Object formalism.

Other current EXCALIBUR work comprises:

- design of Student Model systems that incorporate different theories of cognitive structure and learning; with the aim of using EXCALIBUR as empirical tests of those theories (Cumming, 1986). The use of ICAL systems as test beds for cognitive theories has been explored by (Anderson et al., 1984).
- the problem of incorporating current work on "Instructional Design' - the study of how to design good teaching vehicles and learning environments — into EXCALIBUR as features available by default or on the author's choice (Barrett, 1986).
- natural language processing for the system (Sussex, 1967).
- user-computer discourse management, including answer negotiation, logic of dialogue, and modelling student beliefs (Girle, 1986).

Acknowledgements

This research is supported by grants from the Apple Educational Foundation, Australian Telecommunications and Electronics Research Board, La Trobe University School of Mathematical and Information Sciences, and the Victorian Department of Industry, Technology and Resources.

The authors wish to thank Jan Reed, Dennis Hill and Roly Sussex for valuable discussions.

References

[1] Anderson, J.R., Boyle, C.F., Farrell, R. and Reiser, B.J. (1984) **Cognitive Principles in the Design of Computer Tutors**. *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, Boulder, Colorado.

[2] Barrett, J. (1986) **Intelligent CAL - some development considerations**. *Proceedings of 1986 CALITE conference*, 23- 27, University of Adelaide.

[3] Brown, J.S., Burton, R.R. and De Kleer, J. (1981) **Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III**. In Sleeman and Brown, op cit., Ch.11, *Byte*, (1986). [Special issue on OOPL's].

[4] Cumming, G. (1986) **The Mentor Component in EXCAL**. In *Proceedings of the First Round Table Conference, Technical Report* 3, EXCALIBUR Project, La Trobe University.

[5] Girle, R.A. (1986) **Dialogue and discourse**. *Proceedings of 1986 CALITE conference*, 123-135, University of Adelaide.

[6] Goldstein, I.P. (1981) **The genetic graph: a representation for the evolution of procedural knowledge**. In Sleeman and Brown, op cit, Ch.3. (1986*). Proceedings of the OOPSLA '86 Conference.* Constituting SIGPLAN Notices, September 1986.

[7] Richards, T.J. (1986a) **A Description of the Educational Expert Systems Project**, *Technical Report* 1, EXCALIBUR Project, La Trobe University.

[8] Richards, T.J. (1986b) **Knowledge representation in expert system based computer assisted learning**. *Proceedings of 1st Australian Artificial Intelligence Congress*, Section H, Deakin University.

[9] Richards, T.J. and Webb, G.I. (1985) **ECCLES - An "Expert System" for CAL**. *Proceedings of the 1985 Western Educational Computing Conference*, Oakland, CA Educational Computing Consortium, 151-157.

[10] Sleeman, D. and Brown, J.S. (eds) (1982) *Intelligent Tutoring Systems*, Academic Press, New York.

[11] Sleeman, D.H. and Hendley, R.J. (1981) **ACE: a system which analyses complex explanations**. In Sleeman and Brown, op cit., Ch. 5.

[12] Sowa, J.F. (1984) *Conceptual Structures*, Addison Wesley, Reading, MA.

[13] Stevens, A., Collins, A. and Goldin, S.E. (1981) **Misconceptions in students' understanding**. In Sleeman and Brown, op cit., Chapter 1.

[14] Sussex, R. (1987) **Natural Language Interfaces and Representations**, *Technical Report* 4, EXCALIBUR Project, La Trobe University.

[15] Webb, G.I. (1986) **Knowledge representation in computer-aided learning**, *PhD. Thesis*, Department of Computer Science, La Trobe University.