Natural-language processing applied to an ITS interface

Enrico Fischetti and Antonio Gisolfi

Dipartimento di Informatica, University of Salerno, Italy

Abstract

The aim of this paper is to show that with a subset of a natural language, simple systems running on PCs can be developed that can nevertheless be an effective tool for interfacing purposes in the building of an Intelligent Tutoring System (ITS). After presenting the special characteristics of the Smalltalk/V language, which provides an appropriate environment for the development of an interface, the overall architecture of the interface module is discussed. We then show how sentences are parsed by the interface, and how interaction takes place with the user. The knowledge-acquisition phase is subsequently described. Finally, some excerpts from a tutoring session concerned with elementary geometry are discussed, and some of the problems and limitations of the approach are illustrated.

Introduction

This paper deals with the way in which users and computers can interact by means of natural language. In particular, we focus our attention on the development of a restricted and simplified, but still potentially useful, natural-language interface for an Intelligent Tutoring System (ITS).

Although natural-language processing technology has become flexible enough to assist in certain problem-solving activities, only powerful systems such as INTELLECT (AIC, 1986) are currently able to tackle it effectively. Yet even such systems are subject to severe limitations (e.g. distinguishing between nouns and verbs), and consequently the problem arises of considering to what extent simple systems handling only a subset of a natural language, and running on inexpensive machines, can be effectively used for interfacing purposes.

So far, many systems for understanding natural language have been developed, and their advantages and disadvantages have been investigated for some time (see, for example, Morik, 1984; Rich, 1985; Ogden, 1986; Shneiderman, 1986). The most potent argument in support of the use of natural language is, of course, its flexibility and 'naturalness'. The merits of natural-language interfaces are well documented as a result of a number of field trials (Turner *et al.*, 1984; Jarke *et al.*, 1985; Capindale and Crawford, 1990; Slator *et al.*, 1986). But one should remember that 'normal' language is not always the best way of describing things. In fact, activities requiring mathematical skills (e.g. creating shapes or solving equations) benefit from the use of a restricted formal language that allows the avoidance of ambiguities inherent in natural languages. On the other hand, when question-answer tasks are to be tackled, natural-language interaction is clearly better suited.

Intelligent Tutoring Systems

Although the goal of a system capable of entirely autonomous teaching is still very far off, recent advancements in both hardware and software technologies mean that systems can now be developed that offer partial but nevertheless potentially effective solutions to problems of flexibility, and that can act as reliable partners for human teachers. It is apparent that the overall result of bringing together multimedia devices and AI technology will represent a significant improvement in the quality of interface systems available for ITSs (Barker and Tucker, 1990).

ITSs originated during the 1970s, and have emerged, among other things, as tests for Artificial Intelligence (AI) concepts. Computer-Assisted Instruction (CAI) systems, despite their success in certain environments, suffer from an inherent rigidity, being unable to adapt themselves to the characteristics of specific students. But AI techniques can be profitably used to improve the performance of CAI systems. ITSs take into consideration individual differences in students, and provide more individualized instruction. This adaptation to individual students has been made possible through the development of certain modules within an ITS (Polson and Richardson, 1988; McFarland and Parker, 1990). The most generalized architecture includes four functionally distinct, though linked, modules:

- Database module, containing knowledge about the topic to be taught, and holding the data created while the ITS is in operation;
- Student module, implementing the student model, and tailoring the system behaviour to the student's needs;
- Teaching module, implementing adequate tutoring strategies according to the information obtained in the student module, and controlling the topic to be taught, and how and when it should present it;
- Interface module, managing student interaction with the system, and supporting the building phase of the knowledge base by the human expert.

The role of the interface module is of critical importance (see Brown, 1989; Laurel, 1990), since it processes the information flow from the user to the system, and vice versa. Thus the presentation of a topic will be made more or less understandable according to the quality of the interface.

The knowledge base is usually built by expert AI programmers, but an important issue is constructing the knowledge base so that a teacher-expert in the field can modify it, even though he or she may lack programming experience. For this, our interface module manages the knowledge acquisition phase so that the ITS is easily maintainable.

The Smalltalk/V programming environment

The Object-Oriented Programming (OOP) paradigm allows systems to be modelled in terms that attempt to match human thinking and language, and considering that computer-based tutorial activity also attempts to simulate the way in which human beings tackle teaching and learning, the importance of the OOP approach with respect to ITS research is clear. The essence of the approach is that software systems are modelled in terms of *objects* and actions on objects by means of *messages.* An object is essentially a set of data and methods (the procedures that act on this data). In defining a type of object, a *class* is obtained whose elements share some characteristics. The classes are hierarchically organized so that their characteristics can be inherited by sub-classes.

We decided to implement our interface module in Smalltalk/V, a dialect of Smalltalk developed by Digitalk Inc. one of whose main advantages is that it runs on PCs. It also offers very rapid coding and testing, and provides a wide range of HCI (Human-Computer Interface) facilities. In particular, the graphical interface is very friendly, the user interacting with the system mainly by means of windows and menus; friendly interfaces combined with powerful facilities are an essential part of ITS development, as has been shown many times (Or-Bach and Bar-On, 1989; Hobbs, 1990; Gisolfi and Moccaldi, 1990; Gisolfi and Moccaldi, 1992; Fischetti and Gisolfi, 1992). Moreover, employing OOP in ITS development is useful for diagnosis. Each concept is implemented as an object, and objects are in turn hierarchically organized so that the attributes needed for diagnosis can be inherited. Smalltalk/V is very economical in code: an impressive amount of activity results from a few lines of code, and one can re-use most lines of pre-written code. Further, editing and correcting errors is accomplished very easily. Of course, OOP systems are not disadvantage-free (see Jones and Thorne, 1988); for instance, dynamic binding allows software rules to be compiled, linked and loaded incrementally, but requires complex structures for its implementation and additional time for execution.

We can give here, in describing our work, only a vague feel for the unique features of Smalltalk that make it quite different from conventional languages. Newcomers to the language may be helped by standard texts on the subject, e.g. Meyer (1988) and Digitalk (1988).

The architecture of the interface module

Our interface module consists of two distinct sub-modules: Expert and User. The Expert sub-module manages knowledge acquisition and allows the human expert to build the knowledge base of the ITS so that it can subsequently be consulted by the student by means of the User sub-module. The hierarchical characteristics of the Smalltalk/V language are exploited: thus, we have defined a sub-class of the class Object, called Modules, and have linked the classes Interface, Engine, Expert Interface and User Interface as depicted in Figure 1.

Although the underlying principles of the interface are simple, their translation into

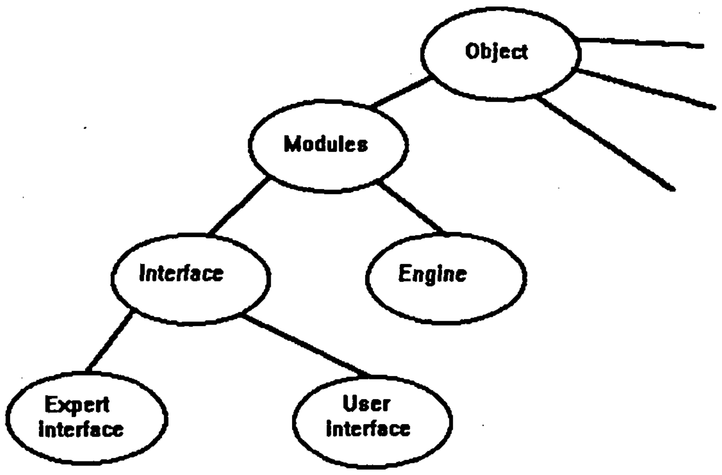

Figure 1: Hierarchical structure of the interface

software brings with it several unavoidable technicalities that use the advanced features of Smalltalk/V. It is not possible in an outline paper of this kind to go into the details of these technicalities, and we confine ourselves here to some conceptual aspects.

The class Engine is implemented by means of the minimal functions of a database module, so that the knowledge base can be built fairly easily. In Smalltalk/V terms, the knowledge base consists of a set of objects having an internal state, as defined by their characteristics (instance variables), and a set of specific methods, for example those defined for the classes String, Dictionary and Set. The knowledge base evolves gradually as its elements are modified: after some initial conditions have been satisfied, changes in the knowledge base are related to the inferences carried out by the database module.

As the interface has to process natural language, its dictionary plays a vital role. The non-specific part of the dictionary is built-in: the human expert has only to add information specifically related to the topic to be taught. The expert is guided in this activity by appropriate windows and menus present in the Smalltalk/V environment. All the terms inserted by the expert are added to the basic dictionary. The subset of the English language recognized by the interface incorporates nouns, verbs, adjectives, conjunctions, articles and prepositions. The first three of these (nouns, verbs, adjectives) contain objects to be considered as 'meaningful' (a semantic value is associated with each of them), whereas the other three (conjunctions, articles and prepositions) are used to link together elements of the 'meaningful' classes. Building the knowledge base has the side-effect of adding to the interface all the elements it needs to carry out its function.

Our prototype ITS is intended for a tutoring activity in elementary geometry. The human expert has first to specify what objects are involved (triangles, rectangles, etc.), and then for each object its syntactic features (singular, plural, gender in the case of Italian) and its semantic ones (subclass of ...). Other information is also linked to each object, for instance the number of sides, the perimeter, the area, and what operations are allowed for the object. For example, operations such as 'area measurement' or 'verify the equivalence' are suitably linked to Smalltalk/V methods, then saved in the appropriate class.

The goal of the interface is to associate a semantic value with the sentence typed by the user, something which can be tackled only if the process is adequately supported by parsing algorithms able to check the syntactic and semantic validity of the sentence. The analysis carried out by the user-interface, faced with the sentence X, can be summarized as follows:

(a) Does X belong to the interface language?

(b) What is the type of X?

(c) What is the semantic value of X?

The following section shows how sentences are parsed and these questions answered.

Parsing sentences

The process which a sentence typed by the user undergoes can be briefly described as follows:

The interface module first analyses the sentence to check whether all its elements belong to the dictionary. Each acceptable word is stored in the dictionary along with its synonyms, allowing the maximum available flexibility, and therefore the maximum available freedom for the user. Although the general structure of parsable sentences is subject-verb-object, the interface can also cope with more complex sentences where there are verbs indicating requests. For example, given the sentence 'Rome is the capital of Italy', the interface recognizes the noun 'Rome' as the value of the instance variables Capital and Italy as instances of the class Nation. If the check is positive, the next step involves a diagnosis as to whether the sentence is the answer to a tutor question or a user's question, and the dialogue state has to be changed accordingly. Now, the parsable sentences can be classified as one of:

(a) Declarative

(b) Direct interrogative

(c) Indirect interrogative

Direct interrogative sentences are recognized by the presence of the final question mark, a crude but workable solution. User's answers are expressed by declarative sentences. Indirect interrogative sentences express an information request by the student. As their structure is similar to that of declarative sentences, we have introduced into the dictionary a set of typical expressions whose presence induces the interface to assign the sentence to the indirect interrogative class. These expressions are, for example: Tell me, Show me, Explain, and so on. The interface manages user's questions by means of the Prompter, a built-in Smalltalk/V window. The sentence typed by the student is stored in the global variable Line, and then the parsing process starts.

A crucial problem to be tackled is the correct management of sudden changes in the dialogue state. A dialogue between two individuals may be composed of questions and answers, but it is hardly realistic that the roles 'asking-answering' of individuals carrying on a question-answer dialogue remain static, something which happens only in quiz-like sessions, and one has thus to consider that, for example, when the student is asked a question, instead of answering, s/he might reply with another question. To cope with such situations, the binary variable Dialogue-state is introduced, whose value changes according to the circumstance that a question or an answer is expected from the student.

Thus, if the Dialogue-state indicates that the student's answer is expected, and an interrogative sentence is typed, the unanswered question is saved into a stack, the Dialogue-state switched, the student's question answered, the tutor's question popped from the stack, and the Dialogue-state switched again.

Associating a semantic value with a sentence X has proven to be one of the most difficult problems. We have tackled it by supposing that the student can ask five types of question, schematically classified as follows:

(1) Tell me if . . .

(2) Tell me how . . .

(3) Tell me why . . .

(4) Tell me which . . .

(5) Tell me how many . . .

The keywords: *if, how, why, which* and *how many* are stored into Set instances, and are declared as global variables in the basic dictionary. It is worth noting that the performance level of the interface is strictly related to the characteristics of the Smalltalk/V environment. This carries with it some disadvantages, but on the positive side, this means that a hierarchy of classes can be developed, each linked to a type of object. In this way, all the knowledge related to the object is contained in a unique class, including internal variables and methods, so that if a particular class is selected, one is not obliged to deal with the entire knowledge base: the methods of the selected class are used to investigate its internal variables and related instances. But one has also to remember that the interface is not a module independent from the others making up an ITS: indeed, there is no sharp dividing line between ITS modules, and managing questions requires suitable interactions between the interface and the inference engine of the database module.

The first type of question ('if') is a confirmation request. Of course, it can also be expressed, like other classes of sentences, by a direct question ('Is a triangle a polygon?'). The interface has simply to check that the relationship between the objects is present and, if this holds true, the student's hypothesis is confirmed.

The second type of question ('how') is concerned with requests for explanations about the correct way of executing a specific operation. In this case, the answer requires that the methods introduced by the expert are suitably communicated to the student. For example, the sentence 'Tell me how the perimeter of the triangle is measured' activates the inference engine to draw the symbolic code associated with the object 'triangle' and the operation 'measurement of perimeter'.

The third type of question ('why') is concerned with requests for explanations about the logical steps that have led to the current state. For example, in the sentence: 'Why does the measure of the area equal 70?', a Smalltalk/V method will show the steps the current situation depends upon.

The fourth type of question ('which') involves situations where the student wishes to know which objects satisfy the constraints present in the question. If, after the preposition 'of', we find the name of the class to be investigated, synonyms management allows the phrase 'surface of the triangle' to be recognized by the system as equivalent to 'area of the triangle', and since 'area' is one of the instance variables of the class Triangles, the object Triangle is scrutinized to get information about area measurement. The reasoning can be iterated, and a similar process investigates the structure of the more complex phrase: 'measurement of the area of the triangle'. The basic structure of the system allows for several kinds of mistake to be detected. For example, it is easy to verify that the sentence 'measure the area of the side' is incorrect because there is no instance variable associated with area in the class Side.

The last type of question ('how many') can be managed in a similar way to the previous one, but the system has additionally to look for all the objects that satisfy the conditions of the question.

Thus, in our architecture, the knowledge base can be regarded as a set of classes, hierarchically organized according to an expert's requirements. Each class is associated with a class variable of type Set in order to contain the instances created during the processing phase. One can declare, for each class, a specific sub-class named Exercises, whose instances are all the exercises concerning that class. The student model benefits from this approach as the analysis concerning student's knowledge can become more specialized as lower hierarchical levels are attained. In this way, the interface becomes more powerful because a sentence can be analysed semantically without using the inference engine. For example, diagnosing that the phrase 'area of the line' is meaningless would be immediate, although the sentence is syntactically correct.

Operating the knowledge-acquisition interface

As mentioned earlier, the dictionary required by the interface consists of a non-specific part which includes basic information (articles, prepositions, auxiliary verbs, request verbs, and so on), and a specific part consisting of information related to the particular topic to be taught. Figure 2 depicts the general structure of the dictionary and its functional links.

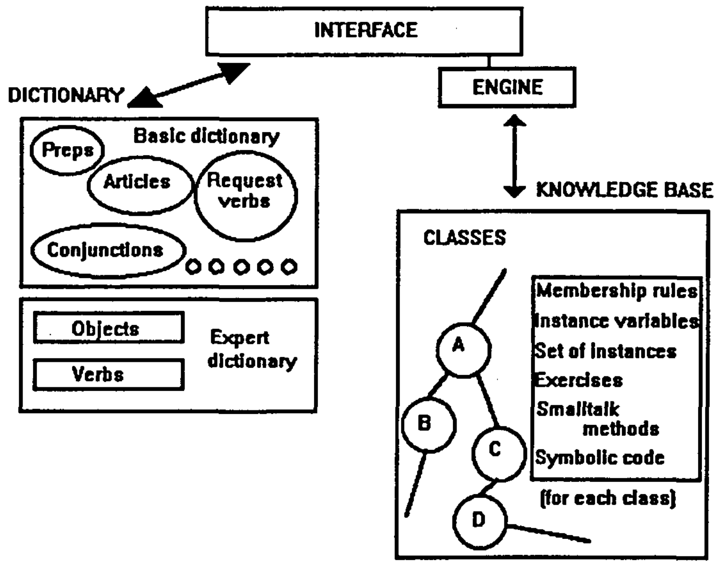

Figure 2: Structure of the dictionary and functional links

The structure of the first part of the dictionary is relatively simple. Instances of the class Set, declared as global variables, are used to contain articles, prepositions, conjunctions and a collection of typical terms that characterize interrogative sentences, such as 'Tell me' or 'Show me'. The second part of the dictionary is more complex because the information furnished by the human expert about the topic to be taught has to be organized in a suitable way. We have defined two basic classes – Things and Actions – that represent the objects and the operations on them respectively. An instance of Things is characterized by the name (noun), its gender (in Italian), its plural and a set of associated synonyms. In a similar way, an instance of Actions consists of the tenses of a verb and a set of synonyms. All the instances of Things are stored in a set called Objects; the instances of Actions are stored in a set called Verbs. For example, as elementary geometry is the topic to be taught, the class Things will include Triangle, Rectangle, Circle, Side, Area, and so on. In turn, in the class Actions, we have verbs such as Compute, Measure and Compare.

The knowledge acquisition phase can be sketched as follows. First, the expert is requested to specify the list of objects the topic consists of. Each topic requires several specifications because part of the information is required by the interface and another part by the inference engine. The expert has to define the syntactic features of the object, i.e. singular, plural, gender and synonyms. Consequently, the instance variables get their value, and the resulting object is added to the set Objects. Subsequently, the relation 'is a' is considered, and in such a way classes get hierarchically organized. For instance, one has (the asterisk denotes that the class has no super-class):

Perimeter	————————	*
Area	————————	*
Polygon	————————	*
Square	————————	Polygon
Triangle	————————	Polygon
Right-angled triangle	————	Triangle

Figure 3 depicts the hierarchical structure of the geometric classes.

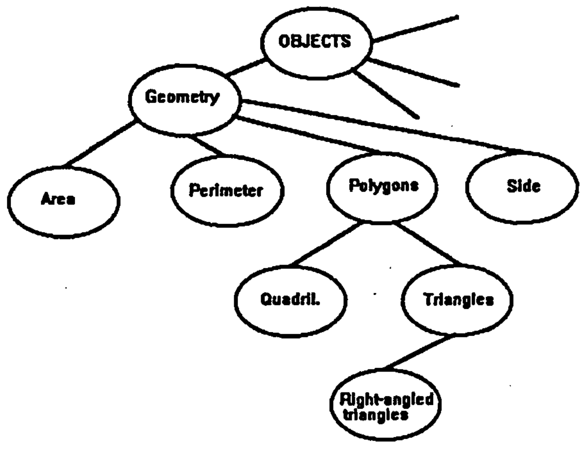

Figure 3: Hierarchical structure of the geometric classes

In order to define the classes, the expert has to specify the features of the objects, and these characteristics become the instance variables of the classes. For example, a polygon is characterized by the number of sides, formulas for evaluating area and perimeter, and so on. More specifically, to manage the peculiar features of the objects, for example triangles and quadrilaterals, two global variables are defined: Features and Differences. In such a way, the expert specifies which features are shared by objects and which characteristics are peculiar to each object. In particular, the variable Differences contains a set of rules that characterize the objects belonging to that class. In other words, the values contained in such a variable represent the necessary and sufficient conditions for an object to belong to that class: a polygon is a triangle if, and only if, three sides are present.

When the building phase is completed from the syntactic point of view (gender, synonyms, singular, plural), the expert has to specify which operations are allowable for these objects, and this happens by means of sentences having the general structure verb-complement. First, each sentence is analysed with respect to its compatibility with the objects present in the dictionary, and if no discrepancy is recognized, the sentence is stored in the dictionary.

The activity of the interface is twofold: while the dictionary is suitably enlarged, the engine receives the necessary information to define the methods related to the instances of the classes. This task is by no means trivial; for example, the complement typed by the expert might be unknown to the dictionary, so that the operation will be utterly meaningless. If this is not the case, and the complement is a known object, the verb is added to the set Verbs, after having been declared as an instance of the class Actions, and its conjugation and synonyms having been specified. If the complement is unknown, the expert is requested to re-type the name of the object, and if it is still unknown, the interface asks whether the new object is to be added to the dictionary. If the answer is Yes, the complement is added, through the above-mentioned procedure, to the set Objects, and the action to the set Verbs. Of course, the expert can alternatively express the complement in a different way if he or she does not want to introduce a new term.

The above-mentioned operations are carried out for each object declared as a class. When all the required information is collected, the interface prepares the Smalltalk/V code representing the methods that operate on the data.

Thus each class gets associated with a set of relevant information. For example, the class Right-angled triangles includes slots for several pieces of data: the names of the sides and vertices of the triangle, the lengths of its sides, and the measurement of its angles, features such as 'number of sides = 3' and 'one angle is 90°', the symbolic code for evaluating its area, perimeter, catheti and hypotenuse. At the end, the dictionary is ready to start the consulting phase.

Operating the user interface

A student activates the ITS and starts interacting with the system. Suppose that the student types:

'Tell me how the area of the triangle can be measured'

The words this sentence consists of are recognized by the interface, and thus the sentence is reduced to a standard form:

'Tell me how area of Triangle is measured'

The system displays this to the student, and asks for confirmation. If the sentence is not confirmed, the interface asks for a new formulation of the question. Otherwise, the module recognizes that this sentence expresses an information request thanks to the term 'Tell me', while the presence of the adverb 'how' allows it to recognize that the student wants to know how a certain operation can be carried out. The operation is 'area measurement', and the presence of the phrase 'of triangle' allows the system to recognize the relationship between the operation (measurement), the feature (area) and the class (triangle).

At this point, the interface is able to ask the engine to draw up the symbolic code associated with the method 'area measurement' defined for the class Triangles. This looks as follows:

[Engine new DrawFrom: Triangle CodOf: area measurement] Interface: Measurement of area of triangle: Base * Height / 2

Suppose now that the student types:

'Measure the area of the quadrilateral abcd'

As the noun 'quadrilateral' is followed by another noun, the interface is able to recognize that the request 'measure' refers to some instance of the class Quadrilateral. Thus the set of instances of this class is investigated to verify that 'abcd' exists, then the data is fetched. If 'abcd' is not an instance, the interface asks the student whether it has to be defined. If the answer is Yes, the interface calls the engine in order to declare a Quadrilateral. The related method gives the appropriate values to the instance variables, provided that the rules present in Differences are not contradicted, and finally the new instance is added to the instance of Quadrilateral. Then the method specific for area measurement is activated:

[Engine newActivate: Area measurement in: Quadrilateral for: abcd].

Suppose that the interface is faced with the following statement:

'Tell me if ptr is a polygon'

This question is a request about a possible class membership. Suppose that 'ptr' is an instance of Triangles but not of Polygons. The interface asks the engine to look for such instance:

[Engine new Search: ptr relay: is a]

and starting from the circumstance that a triangle is a polygon, the question is answered affirmatively.

Finally, to give an idea of how the system works in practice, here is an excerpt from a typical student-interface dialogue:

S: Tell me how the perimeter of the tide can be measured

I: I don't understand "tide"

S: Sorry, how do you measure the perimeter of the side?

I: "Measure perimeter" is not defined for side.

Concluding remarks

We are all too aware of the constraints of our interface, but it is apparent that language-processing capabilities which will be satisfactory in certain circumstances can be achieved by means of a simple conceptual architecture supported by a powerful programming environment. Field tests so far carried out show that users generally have a positive attitude towards the interface. Most errors made by users are directly related to restrictions in the language, and when a mistake occurs, feedback helps the user to understand the language limitations of the interface, so that errors are subsequently avoided. Of course, this kind of positive conclusion has to be drawn cautiously: far more extensive evaluation is required, and we are now actively engaged in testing, evaluating and improving the interface.

References

Digitalk (1988) (anon.), Smalltalk/V 286 tutorial and programming handbook, Los Angeles, Digitalk Inc.

AIC (1986) (anon.), INTELLECT reference manual, Waltham, Educational Technology Publications.

Barker, J. and Tucker, R.N. (eds) (1990), *The Interactive Learning Revolution: Multimedia in Education and Training*, London, Kogan Page.

Brown, J.R. (1989), *Programming the User Interface – Principles and Examples,* Reading, MA, USA, Addison-Wesley.

Capindale, R.A. and Crawford, R.G. (1990), 'Using a natural language interface with casual users', *International Journal of Man-Machine Studies,* **32**, 341–62.

Fischetti, E. and Gisolfi, A. (1992), 'Design and development of the student module of an ITS', *Interactive Learning International,* **8**, 201–11.

Gisolfi, A. and Moccaldi, G. (1990), 'An ITS for the factorization of algebraic expressions in Smalltalk/V', *Educational and Training Technology International,* **27**, 4, 406–13.

Gisolfi, A. and Moccaldi, G. (1992), 'The student module in POSITS, a tutor for the factorization of algebraic expressions', *Journal of Artificial Intelligence in Education,* **3**, 3, 347–58.

Hobbs, D.J. (1990), 'Second-level design of a knowledge-based educational advisor', *Educational and Training Technology International,* **27**, 2, 216–23.

Jarke, M., Turner, J.A., Stohr, E.A., Vassiliou, Y., White, N.H. and Michielsen, K. (1985), 'A field evaluation of natural language for data retrieval', *IEEE Transactions in Software Engineering, SE-II,* **1**, 97–113.

Jones, A.C. and Thorne, M.P. (1988), 'Smalltalk as a vehicle for intelligent computer assisted learning', *Proceedings of the 5th International Conference on Technology in Education,* March 1988, London, CEP Consultants.

Laurel, B. (ed) (1990), *The Art of Human-Computer Interface Design,* Reading, MA, USA, Addison-Wesley.

McFarland, T. and Parker, R. (1990), *Expert Systems in Education and Training*, Englewood Cliffs, USA, Educational Technology Publications.

Meyer, B. (1988), *Object-Oriented Software Construction*, Englewood Cliffs, USA, Prentice-Hall.

Morik, K. (1984), 'Cumers' requirements for natural language systems: results of an inquiry', *International Journal of Man-Machine Studies*, **21**, 401–14.

Ogden, W.C. (1986), 'Implications of a cognitive model of database query: comparison of a natural language, formal language and direct manipulation interface, *SIGCHI Bulletin*, **18**, 2, 51–4.

Or-Bach, R. and Bar-On, E. (1989), 'Probit: developing an intelligent tutor', *Proceedings of the 4th International Conference on AI and Education*, Amsterdam, 185–92.

Polson, C.P. and Richardson, J.J. (eds) (1988), *Foundations of Intelligent Tutoring* Systems, Hillsdale, USA, Lawrence Erlbaum.

Rich, E. (1985), 'Natural language understanding: how natural can it be?, *IEEE Conference on AI Applications*, 372–7.

Shneiderman, B. (1986), 'Seven plus or minus two central issues in human computer interaction' in Mantel, M. and Orbeton, P. (eds), *Human Factors in Computing Systems: CHI '86 Proceedings*, 343–9.

Slator, B.M., Anderson, M.P. and Conley, W. (1986), 'Pygmalion at the interface' *Comm. ACM*, **29**, 7, 599–604.

Turner, J.A., Jarke, M., Stohr, E.A., Vassiliou, Y. and White, N. (1984), 'Using restricted natural language for data retrieval' in Vassiliou, Y. (ed.), *Human Factors and Interactive Computer Systems*, Norwood, USA, Ablex, 163–90.