INTELLIGENT FRONT ENDS

Alan Bundy

D A I RESEARCH PAPER NO. 227

Commissioned paper for the Pergamon Infotech State of the Art Report "Expert Systems" Pergamon Infotech Ltd., 1984.

July 1984

Intelligent Front Ends

by
Alan Bundy

Abstract

An intelligent front end is a user-friendly interface to a software package, which uses Artificial Intelligence techniques to enable the user to interact with the computer using his/her own terminology rather than that demanded by the package. Several such systems exist and provide interfaces for finite element, statistical and simulation packages, and the area is an important area of growth for expert systems. In this paper we discuss the techniques required in an intelligent front end and whether general tools can be provided for their construction.

Acknowledgements

I would like to thank Bob Muetzelfeldt, David Probert and Mike Uschold for their valuable comments on early drafts of this paper.

This research was supported by SERC grant, number GR/C/06226.

1. Introduction

An intelligent front end, IFE, is a kind of expert system. It is a user-friendly interface to a software package which would otherwise be technically incomprehensible and/or too complex to be accessible to many potential users. It will be convenient to use the term 'package', throughout this paper, to refer to the target of the IFE, but this term is meant to be interpreted in a very general sense. It might be a traditional software package, such as a finite element, statistics or mathematical modelling system, but it might be a database, a compiler or a computer network.

An intelligent front end builds a model of the user's problem through a user-oriented dialogue, which is then used to generate suitably coded instructions for the package. It allows users to explain their problems in language familiar to them and then translates this into a language suitable for the software package.

Intelligent front ends are likely to become of major importance in the immediate future. The falling price of computers has made it possible for a much wider audience to have access to them. A large amount of existing software is potentially useful to this audience, but the audience may be not be able to use it. The software that was built before the advent of cheap computers was designed to be used by a small elite of computer experts and can be incomprehensible to the layperson. IFEs promise to act as a translator between this software and the lay user and, hence, enable the widespread use of powerful software. If IFEs succeed they will be of major commercial and social significance.

The architecture of a typical IFE is given in figure 1-1. The specification of the user's task or problem is extracted during a dialogue between the user and the program. Instructions to run the package are then synthesised from the task specification and the package is run. The results of the package are then interpreted into the language of the task specification and explained to the user as part of the dialogue.

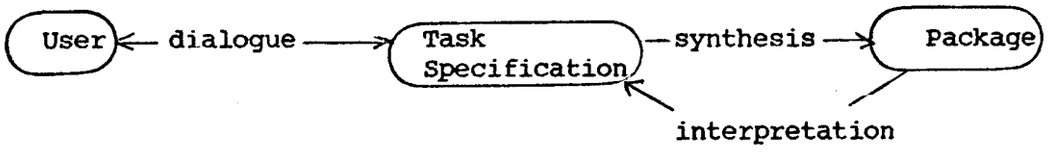

Figure 1-1: A Typical Intelligent Front

The distinguishing characteristic of an intelligent front end, as opposed to a merely rational or well-engineered front end, is the explicit representation of the user's problem in the task specification. It is this which enables the user to state the task in a different terminology from that used by the package. In addition, an IFE might also have a model of the user, representing, for instance, the user's understanding of the package. It might also have a model of the package, describing what kind of task the package can and cannot cope with.

Examples of IFE systems are:

SACON, [Bennet and Englemore 79], which advises on the use of a finite element package;

ELAS, [Weiss et al 82], which assists in the analysis and interpretation of well log data for determining the likely presence of hydrocarbons.

φ_0, [Barstow et al 82], which helps a petroleum scientist develop a mathematical model;

- ECO, [Uschold et al 84], which helps an ecologist build FORTRAN simulation models;

ASA, [O'Keefe 82], which helps a psychologist find a suitable statistical analysis of his/her experiment;

REX, [Gale and Pregibon 83] which assists in the use of a statistics package; and

ADVISOR, [Genesereth 79], which is an automated consultant for MACSYMA.

These example IFEs fit the idealized architecture of figure 1-1 with varying degrees of precision. For instance, many of them do not have an explicit task specification, but instead integrate the synthesis and dialogue subsystems into one. In fact, figure 1-1 has a normative as well as a generalizing aspect - we will argue that this is how IFEs must be constructed if they are to free the user from the language of the package.

In this ideal IFE the following kinds of Artificial Intelligence and Expert System

techniques are called for:

- Knowledge Representation, to represent models of the task, the user and the package;

- Problem Solving, to synthesise the package instructions from the task specification and to interpret the results of the package;

  Natural Language Understanding, which may or may not be used in the dialogue with the user.

Thus the IFE area involves the integration of several different areas of AI. It also impinges on non-AI areas of Computer Science.

Man/Machine Interaction is concerned with the provision of well-engineered front ends, and the experience of this area must be integrated with the AI techniques to produce the best possible system.

Computer Aided Instruction has considerable experience of interfacing software to naive users. Intelligent CAI has experimented with techniques for user modelling. This experience must be harnessed in IFE building.

Relational Database work has been moving into the IFE area as the user interfaces to large databases have become more 'user friendly'.

That so many different areas of Computer Science have been moving in the same direction tends to reinforce the argument for the significance of this development.

The area of intelligent front ends has been made a research theme of the UK government sponsored, Alvey Programme for Information Technology. Apart from the general promotion of IFEs, one of the main aims of this theme is to develop general-purpose software to assist the building of IFEs. The inspiration of this aim is the provision of EMYCIN-type, expert system shells, which assist in the building of diagnosis systems. IFEs are another kind of expert system. Can a similar shell be provided? Failing this, can a meccano kit of techniques be provided which can be rapidly assembled into an IFE? Since the area of IFEs is wide and somewhat vague around the edges, can we separate out a sharp sub-area for which such shells and/or meccano kits can be more readily provided? In subsequent sections of this paper we explore these possibilities, discuss what kind of software would be required, and whether any of it is already available.

2. Types of Intelligent Front End

IFEs can be of many types depending on the package to be interfaced to, the kind of user, etc. In this section we discuss some of the dimensions of variation. This discussion draws on the report of the first workshop of the Alvey IFE Theme. [Bundy & Uschold 83].

In our definition we have assumed some conceptual distance between the

language in which the user prefers to describe his/her problem and the language in which the operations of the package are naturally expressed. For instance, in ECO the user specifies his/her problem in ecological terms, e.g. "deer graze grass", "deer biomass depends on respiration", etc, and this problem is represented in the task specification. The target 'package' is a FORTRAN compiler which requires input in the form of loops, conditionals, arithmetic expressions, etc. The task specification represents the user's view of the task and this must be translated into the language of the package by the synthesis subsystem.

In some early IFEs, e.g. SACON, REX and ELAS, the conceptual distance between the user input language and the package was minimal. In SACON and ELAS the small amount of translation required was done implicitly by the EMYCIN and EXPERT production rules, and there was no explicit task specification. Since EMYCIN-type shells* have no explicit mechanism for translation between representations and no explicit representation of the task, they are unlikely to be sufficient as shells for IFEs in which there *is* a large conceptual distance between user and package.

In future IFEs the conceptual distance between user and package may be so large that a sequence of task representations and translation procedures are required, analogous to the sequence of scene representations and translation procedures used in current vision systems.

The information provided by the user may not be exactly the information required to decide what instructions to give to the package, even after it has been translated into the language of the package. In this case some 'gap bridging' processing will be required. This might be done by inference in the task specification language using, for instance, an EMYCIN-type production rule subsystem. Some 'jumping to conclusions' may be required, which will call for non-monotonic or default inference.

In some applications it may be impossible to avoid communicating with the user in the language of the package. For instance,

the user may have to choose between alternative inputs to the package, although even in this case it may be possible to present the choice in the user's preferred terminology;

the user may want to see the output provided by the package, although again the significance of this output might be explained in the user's preferred language; or

user may want to be instructed in the direct use of the package.

Another dimension of variation is given by the nature of the input to the package. This can vary from simple one-line commands to arbitrarily complex

*By 'EMYCIN-type shells' I mean a package whose inference engine is a production rule system. My assertions about what cannot be done in such systems should be interpreted as assertions about what cannot be straightforwardly done within the spirit of such systems. Most of these things can be done by exploiting tricks or ad hoc patches, but such tricks and patches will not lead directly to an understanding of how to build custom-made IFE tools.

computer programs. The more structured the input to the package the more processing is required of the synthesis subsystem. AI planning and automatic programming techniques can sometimes be used to synthesise such structured package-input. Since EMYCIN-type shells do not embody such techniques they are unlikely to be sufficient for IFEs in which the package input is highly structured.

If the IFE is aimed at a range of users of different levels of skill, or if the user's skill can be expected to improve over the course of a session, then it will be necessary to make the IFE adaptable. This can be done in several ways.

(a) Users may have commands at their disposal which can change the dialogue subsystem. For instance, the IFE can give more or less explanation and instruction to the user.

(b) These same commands may be called by the system after it has interrogated the user about his/her skill level. (REX does this, for instance.)

(c) The commands may be called by the system after it builds its own hypothesis about the user's skill on the basis of his/her performance in using the system.

(b) and (c) make progressively less demands on the user, and thus are more suitable for the novice. If the default is set for the novice then (a) also makes no demands on the user. (c) requires a user model to be formed and maintained. A danger with (c) is that the users may improve their skill levels faster than the IFE can adapt, resulting in hunting behaviour.

3. Dialogue Handling

We can divide the problem of dialogue handling into two parts:

(a) controlling the overall structure of the dialogue and

(b) determining the mode of the interchanges.

In (a) we are concerned with what questions should be asked and what answers given, when and by whom. In (b) we are concerned with whether natural language, graphics, etc should be used to ask and answer these questions. This factoring of the dialogue handling problem will simplify the explanation of the range of techniques available.

One popular solution to (a) is to use a menu. This determines a tree structure for the dialogue. Each non-terminal node represents a multi-choice question from the program to the user and each arc represents one of the possible answers to this question, which may then lead to a further question. The terminal nodes represent actions to be taken, e.g. instructions to be sent to the package, or information to be stored in the task specification. The standard menu solution to (b) is for the computer to ask the questions in the form of canned text specifying a character for each of the possible replies, and for the user to respond with one of these characters. Many IFE systems, e.g. ELAS, REX, φ_0, make use of menu-

driven dialogue, but with much more complex interactions with the user at each node.

3.1 The Control of the Dialogue

Menu trees can be generalised to a directed graph structure, which we will call the **transition net** (see figure 3-1). Joins in the transition net will mean that there are alternative routes to the same question. Loops will give the possibility of repeated asking of the same question (we hope in different contexts). The current node of the transition net summarises the findings of the program so far and allows subsequent dialogue to be interpreted in the light of this context, which keeps that dialogue brief. Workers in MMI, [Alty 83, Edmonds 82], have produced aids to help the IFE builder design transition nets. For instance, path algebras provide a formalism for representing nets and these algebras are employed in tools for interactive net design.

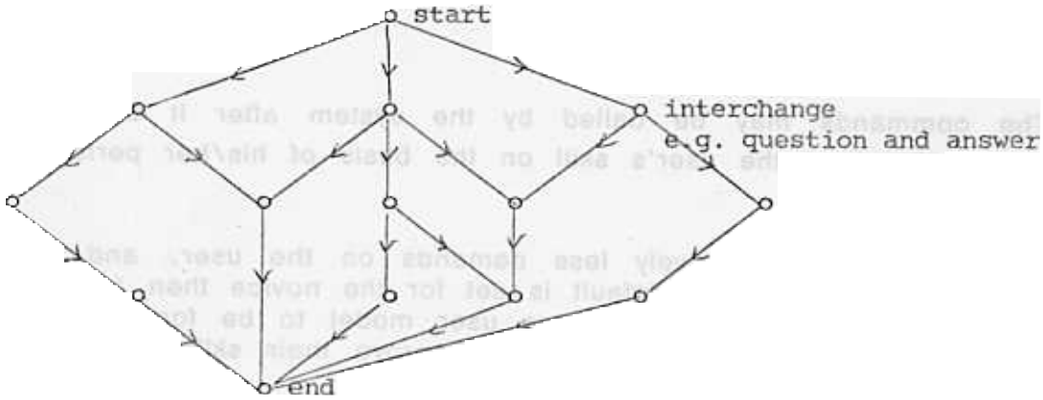

Figure 3-1: A Transition Net

A further generalisation of menus is to generate the transition net dynamically as a byproduct of a problem solving process. For instance, the search tree generated during a run of EMYCIN can be regarded as a transition net. The terminal nodes represent questions to be asked of the user or answered by laboratory data. The non-terminal nodes represent questions to answered by appeal to production rules. SACON uses this technique. Dynamic generation of the transition net means that it can be responsive to context and can be radically different on different occasions (although each net generated can be thought of as a subnet of some, possibly infinite, master net). The cost of this flexibility is that the transition net cannot be designed per se, but that design is restricted to the production rules, etc, which generate the net.

In all the above techniques the initiative remains with the program. Users are constrained to answer the questions asked by the program in an order determined by the program. The input from the user is kept within narrow limits. This eases the job of the program designer and may be welcomed by the novice user. However, it may not be suitable for some kinds of IFE. In some domains the transition net may not be a suitable representation of the contextual information required to process the user's dialogue, for instance, this context may not vary much during the session. Thus, it may be better for the user to have the initiative and volunteer information for the program to store as the task specification. We will call such a

dialogue user-driven, because the program does not know what is coming next. In user-driven dialogues the *mode* of the user input must be constrained, or the program designer will have an impossible job. ECO has a user-driven dialogue system.

Hybrid systems are sometimes desirable, especially where the IFE is intended for users at different skill levels. The novice user may prefer to be guided through the transition net, whereas the expert user may prefer to use a user-driven dialogue to jump straight to the desired node of the transition net.

In many IFEs the dialogue is controlled as a side effect of some more dominant process, e.g. an inference process or the synthesis of the input to the package. The EMYCIN dialogue can be thought of in this way. We will see another example in section 4. Intelligent Browsers provide another example. In RABBIT [Tou et al 82], for instance, the dominant process is the attempt by the program to select a few examples from a large database on the basis of the user's, possibly changing, description of his/her needs. The dialogue is a process of changing the user's description in response to the examples provided by the system at each stage.

3.2. The Mode of the Interchange

We now turn to (b) - the mode of the interchange. This is one of the main areas where IFE designers must look to other disciplines for help. The techniques chosen will depend on the domain of the IFE and not much can be said in general, except to survey some of the possibilities.

In addition to the keyboard input and vdu output available on most computers, special hardware might be used to allow input/output of graphics, touch screen, speech, etc. Each of these is a major area of research in its own right and will not be dealt with here. In the discussion below we restrict ourselves to text input/output. This may range from simple canned text to natural language.

The traditional menu system outputs multi-choice questions as canned text and the user responds with single character input. The context is completely set by the current node in the transition net and the user has no initiative at all.

To make use of a context, external to the transition net, the program output must be dynamically generated in some language. Similarly, to allow the user more initiative there must be a non-trivial, user input language. These languages can be natural or artificial. There is a continuum of possible languages - from a formal, logical language, like predicate calculus, to a natural language, like English. An artificial language which reads like a subset of English, e.g. REX's pseudo-English query language, can combine the ease of parsing of a formal language with the readability of natural language.

If the range of allowed user inputs is narrow then a formal language may be more appropriate than a natural one for the following reasons.

- Allowing some natural language input may mislead the novice user into thinking that any natural language input is allowed.

  Formal language statements are often shorter than the corresponding

natural language ones and put a lighter burden on the non-expert typist.

A simple formal grammar is easily learned by users.

The tradeoffs between the two will depend on:

the training time available for users to learn any formal language.

- the frequency with which users use the system;

their typing skill;

the range of user input required; and

the availability of a suitable natural language parser.

This formal grammar can be as simple as a few phrases or words. For instance, when answering SACON's question: "What is the material composing the total wing?" the user is restricted to a range of phrases like "high-strength-aluminium". SACON can be requested to list the acceptable replies. ECO uses a simple template grammar. An example template is: "<state-variable> depends on. <process>", where <state-variable> might be "deer biomass" and <process> might be "respiration".

Some IFEs, e.g. ELAS, φ_o, use form filling or editing as the mode of interchange. Several windows may be displayed on a bit map display. One of these may describe the current overall state of the dialogue. One may offer a menu for moving to another node of the transition net. One may be a form with blank entries to be filled in with information from the user, or with partially filled entries to be edited by the user. This mode can be very natural and easy to use if the forms correspond to the way in which the user usually thinks of his/her problem, cf spreadsheets.

The ADVISOR consultation system for MACSYMA avoids a lot of explicit dialogue with the user by analysing the his/her initial attempt to use MACSYMA. From this analysis, ADVISOR builds a plan of the user's intentions which is then fleshed out by an explicit dialogue to form the task specification. REX avoids explicit dialogue by analysing the user's statistical data to determine whether its properties fit the preconditions of the various statistical techniques available.

4. Translation, Synthesis and Inference

The synthesis subsystem has several related jobs:

to translate the task specification into the language required by the package;

2. to synthesise the structure required by the package as input; and

3. to fill in any gaps in the information provided by the user.

The first two of these jobs can often be tackled together by using problem-reduction/planning/automatic-programming techniques to instantiate and link together methods which represent package operations. The last of these jobs calls for inference techniques, e.g. deduction, production rules, non-monotonic reasoning, etc.

For instance, suppose the package is a statistics package. The methods might represent different statistical operations each with its own preconditions and effects, and the instructions that will be sent to the package. The language of the preconditions and effects will be the same as that used in the task specification, whereas the language of the instructions will be that used by the package. Thus the methods provide a bridge between the two languages.

To initiate the synthesis process the user's goals, stored in the task specification, are compared with the effects of each method until a match is found. Where there is a choice of method an intelligent browser may be used to assist the user select a method. The instructions associated with the method are then placed into the emerging plan and the preconditions are either seen to be satisfied in the task specification or are set up as new subgoals. The process recurses until all the subgoals are true in the task specification. At any stage, inference may be needed to bridge the gap between the information in the task specification and that required by the methods. Where the task specification does not already contain the required information the dialogue handler may be triggered to request it from the user. Thus the synthesis process provides an architecture for the whole IFE - calling the dialogue, inference and translation processes as subroutines, as required. The whole process is represented diagrammatically in figure 4-1.

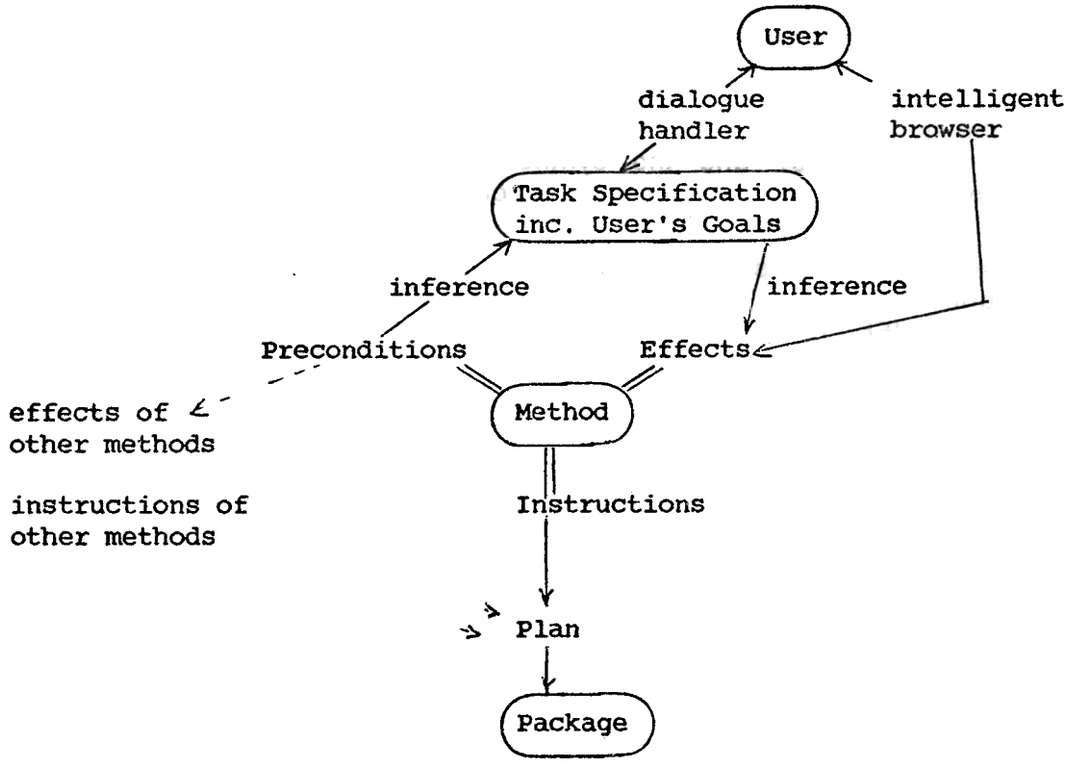

Figure 4-1: The Process of Synthesising the Package Input

φ_0. ECO, ASA and ADVISOR use this kind of means/ends analysis driven synthesis of the input to the package. SACON and ELAS use production rules to bridge the gap between the user's problem description and the package input. The MECHO system used both means/ends analysis and inference rules to synthesise sets of equations from an English specification of a mechanics problem, [Bundy et al 79].

The same basic process can be used if the package requires: equations to solve; or procedures to interpret; or a plan to execute. The matching of the methods to the task specification via inference bridges the gap between the information provided by the user and that needed by the package and enables the translation into the package language and the building of the structured input that the package requires.

Production rules can also be used to translate, but are limited in their ability to synthesise. Translation can be done by having the condition of a rule match the task specification or input from the user and having the action of a rule call the package. ELAS works like this. However, the production rule system would have to be enhanced to enable the rules to build up a structured input to the package, rather than a series of simple commands.

5. Conclusion

Intelligent front ends are a commercially and socially important type of expert system requiring the integration of several areas of AI and Computer Science. This importance has been recognised by the UK Alvey programme.

The ease with which IFEs can be built will depend on the tools available to build them. Ideally, we would like to provide an expert system shell like the EMYCIN family of shells for diagnosis. EMYCIN itself has been used as an IFE shell, e.g. in SACON, but it has limitations as a general IFE shell. In particular, it does not have a problem reduction component which would enable it to handle two of the jobs of the synthesis subsystem: the translation of the task specification into the language of the package; and the synthesis of a structured input to the package. Nor does it provide for an explicit task specification.

But just as EMYCIN has proved an over-restrictive framework for the development of many diagnosis systems, so an IFE shell is unlikely to be universally useful. For instance, if the IFE requires a broad-bandwidth communication with the package, e.g. with access to the package's error handler, then any IFE shell would require extensive tailoring and modification. More promising would be the provision of a meccano kit of procedures and subsystems which can be drawn from during the building of the system.

Among the candidates for an IFE meccano kit are:

Knowledge representation languages, e.g relational/assertional database, frames, inheritance hierarchies.

Procedures for dialogue handling, e.g natural language front end, transition net building tools

Procedures for synthesis and inference, e.g. problem reduction, planner, automatic programmer, deduction engine, production rules, non-monotonic logic system.

Many of these are generally useful AI tools. Further research is required to see if they can be specially tailored to the needs of IFE. I suspect that the IFE area is coherent enough that it will be both possible and beneficial to develop special software for it.

References

[Alty 83] Alty, J.L.
Path Algebras - a useful CAI/CAL analysis technique.
Research Report 125, University of Strathclyde, 1983.

[Barstow et al 82]
D. Barstow, R. Duffy, S. Smoliar, S. Vestal.
An Overview of PHINIX.
In *National Conference on Artificial Intelligence.* American Association for Artificial Intelligence, Pittsburgh, Pennsylvania, August, 1982.

[Bennet and Englemore 79]
Bennet, James S. and Englemore, Robert.
SACON: A Knowledge-Based Consultant for Structural Analysis.
In *Proceedings of the sixth IJCAI*, pages 47-49. International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.

[Bundy & Uschold 83]
Bundy, A. and Uschold, M. (editor)
Intelligent Front End Workshop.
SERC, 1983.
Available from Mr W.P. Sharpe, Rutherford Appleton Laboratory, Didcot, Oxon, OX11 0QX.

[Bundy et al 79] Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.
Solving Mechanics Problems Using Meta-Level Inference.
In Buchanan, B.G. (editor), *Proceedings of IJCAI-79*, pages 1017-1027. International Joint Conference on Artificial Intelligence, 1979.
Reprinted in 'Expert Systems in the microelectronic age' ed. Michie, D., Edinburgh University Press, 1979. Also available from Edinburgh as DAI Research Paper No. 112.

[Edmonds 82] Edmonds, E.A.
The man-computer interface: a note on concepts and design.
Int. J. Man-Machine Studies 16:231-236, 1982.

[Gale and Pregibon 83]
William Gale and Daryl Pregibon.
Building an Expert Interface.
1983.
Bell Telephone Laboratories, Murray Hill N.J. Unpublished

[Genesereth 79] Genesereth, M.
The Role of Plans in Automated Consultation.
In Buchanan, B.G. (editor), *Proceedings of IJCAI-79*, pages 311-319. International Joint Conference on Artificial Intelligence, 1979.

[O'Keefe 82] O'Keefe, R.A.
Automated Statistical Analysis.
Working Paper 104, Dept. of Artificial Intelligence, Edinburgh, 1982.

[Tou et al 82] Tou, F.N., Williams, M.D., Fikes, R., Henderson, A. and Malone, T.
RABBIT: An intelligent database assistant.
In Waltz, D.L. (editors), *Procs. of AAAI-82*, pages 314-8. American Association for Artificial Intelligence, 1982.

[Uschold et al 84]
Uschold, M., Harding, N., Muetzelfeldt, R. and Bundy, A.
An Intelligent Front End for Ecological Modelling.
Research Paper 223, Dept. of Artificial Intelligence, Edinburgh 1984.
To appear in ECAI-84.

[Weiss et al 82] Weiss S., Kulikowski C., Apte C., Uschold M., Patchett J., Brigham R., and Spitzer B.
Building Expert Systems for Controlling Complex Programs.
In *Proceedings of AAAI-82*, pages 322-326. American Association for Artificial Intelligence, 1982.