Natural Language Interfaces to Databases

Prepared for Knowledge Engineering Review, Special Issue on the Applications of Natural Language Processing Techniques, in press.

Ann Copestake and Karen Sparck Jones

Computer Laboratory, University of Cambridge
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK

September 1989

Acknowledgements

This paper follows from work done under Alvey Project IKBS 019, SERC Grant GR/C/98825. We are grateful to Derek Bridge for his comments.

Abstract

This paper reviews the current state of the art in natural language access to databases. This has been a long-standing area of work in natural language processing. But though some commercial systems are now available, providing front ends has proved much harder than was expected, and the necessary limitations on front ends have to be recognised. The paper discusses the issues, both general to language and task-specific, involved in front end design, and the way these have been addressed, concentrating on the work of the last decade. The focus is on the central process of translating a natural language question into a database query, but other supporting functions are also covered. The points are illustrated by the use of a single example application. The paper concludes with an evaluation of the current state, indicating that

future progress will depend on the one hand on general advances in natural language processing, and on the other on expanding the capabilities of traditional databases.

1 Introduction

Database front ends were one of the first applications of automatic natural language processing. Researchers believed that the task was restricted enough to be feasible, that natural language interfaces would be very useful, and that the experience gained would be a valuable basis for attempting more complex language processing tasks. One of the earliest systems was Green's BASEBALL (Green et al 1961), but the LUNAR system (Woods et al 1972, Woods 1978), developed at the beginning of the seventies was the first major landmark, and one which greatly influenced most subsequent work. However, although many research systems have been built and commercial systems have been available for a decade, the practical use of natural language interfaces is certainly not widespread. Building effective front ends has proved much more of a challenge than the earlier workers expected.

Although it is now quite easy to build an interface which will translate a limited range of questions into queries on a particular database, it is not obvious that such a system is very useful. Very restricted natural language does not work particularly well as a straight substitute for a traditional formal query language; it is very difficult to devise a sublanguage which is sufficiently expressive, yet avoids ambiguity and seems reasonably natural, furthermore the limitations on coverage will not be obvious to the user and so the language may be difficult to learn. However an interface which will handle virtually unrestricted natural language and interpret it appropriately, which supports all the other facilities needed by an inexperienced user, and which can be easily customised to different databases, is way beyond the current state of the art.

In the first part of this article we examine the nature of the task itself and identify some of the major problems. Some of these issues are common to all work on natural language interpretation, but others are specific to database front ends. The assumptions made about the context of use of the interface affect the relative importance of these problems and their possible solutions; we will try to illustrate this, but much of the published literature has been very unclear on this point. We give examples of the type of input that we might expect a reasonably sophisticated interface to handle. Some of the properties of the underlying database management system which will affect the design of the interface are also mentioned.

The second and third parts of the article examine the way in which implemented systems have attempted to address these issues. The second section is concerned with the central problem of translating the natural language question into the database query, the third section considers some of the ancillary functionality which is required to support this process. These sections concentrate on the more recent work in the field; although many of the solutions adopted have a much longer history we will only discuss this when it seems the best way of presenting a particular concept. Furthermore we will only briefly mention the issues which are also relevant to the more general application of natural language processing and concentrate on the topics which are particularly important for interfaces to databases. The concluding section evaluates the current state of the art in research systems, and to a lesser extent commercial systems, and suggests how progress might be influenced by general research into natural language processing and, in contrast, by work on expanding the capabilities of traditional databases.

There is a very considerable body of literature on natural language interfaces to databases. The following systems, listed in roughly historical order, are the most relevant to our discussion PLANES (Waltz 1978), LADDER (Hendrix et al 1978), PHLIQA1 (Bronnenberg et al 1980, Scha 1983), TQA (Damerau 1980, 1981), CO-OP (Kaplan 1982), TEAM (Grosz et al 1987), IRUS (Bates, Moser and Stallard 1986), TELI (Ballard and Stumberger 1986), LOKI (Binot et al 1988), JANUS (Weischedel 1989) (the references are in general to the papers which best describe the systems as wholes or to reports on the most recent work). Some of these systems have been used in a commercial environment; commercial systems which have been sold more generally include INTELLECT (derived from Harris's ROBOT (Harris 1977; Harris 1984)), Q&A (developed by Hendrix and others following the experience of LADDER and NANOKLAUS (Haas and Hendrix 1983)) and Natural Language (known earlier as DATATALKER, derived from the system described in Ginsparg (1983), and marketed by Natural Language Inc (Manferdelli 1989)). We also draw on our own experience in building natural language interfaces; the earlier work is presented in Boguraev and Sparck Jones (1983, 1984) and the more recent work in Copestake and Sparck Jones (1989).

There have been a number of conference panels on natural language interfaces, the most useful discussions are reported in Sondheimer et al 1981, Moore et al 1982, Reiter et al 1983 and Sparck Jones et al 1984. There is also some tutorial literature (eg Bates and Weischedel 1987) and relevant papers have appeared in a number of collections (eg Grosz, Sparck Jones and Webber 1986) and special issues such as the ACM Transactions on Office Information Systems (TOOIS) **3** (1985) issue on transportable natural language processing. Database access figures in some natural language processing textbooks (eg Tennant 1981, Allen 1987), but with an emphasis on language-processing techniques and on the broader question-answering context. It has also been explicitly reviewed, notably by Perrault and Grosz(1986).

2 Issues

There are perhaps four main arguments for the utility of natural language interfaces: the user need not learn a special-purpose query language, the range of database queries that can be formulated is potentially equivalent to that of any formal query language (although in practice full equivalence may not be achieved), queries about the domain structure (*metalevel* queries, Storrs et al 1985) can be accepted, and dialogue and query refinement is possible. No other query language has this combination of characteristics. However interfaces cannot yet fully exploit this potential; for an interface to provide any of these advantages the natural language coverage must be relatively unconstrained, and this raises a wide variety of problems. The topics which we will discuss are a combination of linguistic issues, which will apply to other types of natural language interface, and those which are task-specific. The solutions to these problems are affected by a variety of real world constraints, those that arise from interfacing to a standard DBMS, and those which arise from the context of use of the DBMS, how much training the users can be given and so on.

In this section we raise some of these issues informally by illustrating the sort of functionality that we might expect from a natural language interface. There are limitations on what should be expected from an interface to a database, and we try and give some indications of what these are.

There is a very worrying degree of overestimation of the abilities of natural language interfaces. For example,

> "What can you say to a software package that speaks English better than the average gas-station attendant ..."
> "In all Natural Language may be just what the English language needed: a reason to make people heed precise and meaningful language."
> (Alan Southerton, article in Unix World, May 1989, about the "Natural Language" interface system.)

Now obviously this is a piece of journalistic hype, particularly since the article reveals that the system described, like every other natural language interface, works by coercing a natural language question into a query on a very impoverished domain. Statements like "speaks English better than the average gas-station attendant" are presumably not meant to be taken seriously (especially as Natural Language does not speak at all) but there does seem to be a grotesque misjudgement of the capabilities of natural language systems and underestimation of how complex human language is. The interface is not making people "heed precise and meaningful language", it is forcing them to use the vocabulary and concepts which the interface designer set up.

Quite apart from the disturbing implications of how people may have come to think about computer systems there is a serious practical point. To be usable any interface must make its limitations obvious to its users. There is a specific problem with natural language interfaces; the obvious way for a user to model something that accepts natural language input is as though it were an intelligent, rational agent. But no computer system remotely approaches this ideal. The hope is therefore that users can adjust to this and find a reasonable way of modelling such an interface and recognising its limitations. If the system's limitations seem obscure to the user (as might be the case, for example, if it attempts to coerce any question, no matter how inappropriate, into a database query) it will be difficult to use even though it may appear to have a wide coverage.

2.1 An illustrative domain

So what sort of input should a state of the art natural language interface be able to handle? We will illustrate this and subsequent sections with examples taken from the relational database shown in Figure 1. This is assumed

PEOPLE	<u>Id</u> Name Address
STUDENTS	<u>**StudentId**</u>
LECTURERS	<u>**LecturerId**</u> Salary
COURSES	<u>CourseId</u> CourseName **LecturerId**
TAKES	<u>**StudentId CourseId**</u> Grade
REQUIRES	<u>**CourseId RequiredCourseId**</u>

Keys are underlined, foreign keys are in bold type.

Figure 1: An example database schema

to contain data about the students and lecturers of a university department and the courses which are given. The PEOPLE, STUDENTS and LECTURERS relations describe the personnel; students and lecturers are subsets of people, but these subsets are not necessarily disjoint. Each person has a unique identifier, but this identifier might not have any significance outside the database. The attributes of these relations are straightforward but notice that the STUDENTS relation contains no columns other than the student identifier; the relation is not redundant however as it allows a person's status as a student to be recorded even if he or she takes no courses. The COURSES relation contains the information about which lecturer teaches each course. It is assumed that each course has exactly one lecturer (we exclude the possibility of null values and assume that the database is normalised). In contrast the relationship between students and courses, recorded in TAKES, is many-many. The student's grade in each course is expressed by a percentage. The REQUIRES relation stores information about which courses have prerequisite courses, in other words students wishing to take some courses may be required to take some more basic courses first.

Although we have chosen a relational database as an example, most of what we have to say also applies to other data models. The term *database schema* is used to refer to the sort of skeleton structure shown in Figure 1. One point which we will emphasise in this article is that no conventional database model captures all the specialised knowledge needed to fully understand the database structure; for example the knowledge that students and lecturers are subtypes of people cannot be adequately captured in the standard relational model, nor can the constraint that a student cannot be registered for a course without passing all its prerequisite courses. In subsequent sections we will give many more examples of specialised knowledge of this sort which we refer to as being part of the *domain* of the database[1]. The term *data* is used for the information stored in the database records; the data must not be duplicated in the interface's domain model since this would cause inefficiency and/or inconsistency.

The central issue that we have to consider is the process of transforming a natural language question to a valid database query, where this is possible. Very loosely the problem is that the question can be arbitrarily "far" from the eventual query. Consider the question:

Who teaches Smith mathematics?

A possible query on our example database, expressed in SQL, would be:

```
SELECT LPEOPLE.NAME
FROM PEOPLE LPEOPLE, PEOPLE SPEOPLE, TAKES, COURSES
WHERE LPEOPLE.ID = COURSES.LECTURERID
AND (COURSES.COURSENAME = "Basic Maths"
     OR COURSES.COURSENAME = "Algebra")
AND COURSES.COURSEID = TAKES.COURSEID
AND TAKES.STUDENTID = SPEOPLE.ID
AND SPEOPLE.NAME LIKE '%Smith';
```

This would retrieve one or more lecturer's names if Smith does take at least one of the mathematics courses. The definition of "teach" in terms of the two more basic relationships between a lecturer and a course and between a course and a student has to be made available to the interface in some way, as does the information that "Basic Maths" and "Algebra" are both mathematics courses (these are further examples of domain knowledge).

[1]This terminology is fairly standard in the natural language literature but this use should not be confused with the domain of an attribute in standard relational database terminology.

2.2 The advantages of natural language querying

The simple example which we have just described should illustrate that the advantage of natural language interpretation is not mainly that it avoids the syntax of a language like SQL (after all that difficulty can be ameliorated in many ways, Query By Example (QBE) being perhaps the most popular), but that it can hide the structure of the database from the user; in this case one verb ("teach") is translated into a query effectively involving three joins. The user of a natural language interface should not have to be aware of the database schema. But if the user has nevertheless to learn a specialised sublanguage, enshrined by the developer of the interface but not necessarily intuitively obvious, then some of the advantage of ease of learning has been lost. If the interface's coverage of natural language is very limited or very uneven the user will need a lot of training or patience or both. Currently the most successful application of natural languages interfaces is in adventure games; trying to work out what sort of commands the system will accept is part of the puzzle. But natural language interfaces to databases are usually promoted as being suitable for infrequent users who have no time to learn another query language, and expecting them to explore the capabilities of the interface is therefore unreasonable. This also relates to the range of questions that can be answered. The vocabulary and grammar available has to be rich enough for all the frequently asked types of question to be phrased fairly naturally; if they are too constrained it may be impossible to produce some queries which would be possible in a formal query language. However it is difficult to construct large grammars and lexicons, and wider coverage increases the problems of ambiguity.

To illustrate the range of coverage which might quite plausibly be needed, consider the following questions, which are equivalent as far as our sample database is concerned.

> List the lecturers who teach the courses which more than eight people take!
> Whose courses have more than eight students?
> Whose lectures have registrations of more than eight?
> Who lectures on a course that over eight students go to?

There is a problem here in that the last question might signify a misunderstanding about the scope of the database: perhaps the user believes that actual attendance at lectures is recorded rather than just registration for a course.

We are assuming that the goal is to support ad-hoc queries, and essentially to cover the same facilities as the standard query language. In fact many databases are rarely used in this way but are set up to support a limited range of queries efficiently. In such cases the sort of natural language processing which we intend to discuss is mostly inappropriate. Not only does it represent overkill, but the limitations will not be as obvious to the users as if a menu-based system were used. With full natural language support query optimisation becomes very important because the user cannot be expected to have any idea how expensive a query will be to process, but this is a problem for any front end that facilitates access by inexperienced users, not just a natural language one.

The third advantage of natural language interfaces mentioned at the start of the section was that metalevel questions can be answered. The user may deliberately ask about the structure of the database, for example:

> Is any information kept about student's grades?

In other cases the user may intend to query the database itself, but if the the question is in some way misguided the interface should detect the problem and its response should include the same sort of information about the actual domain structure as it would supply if asked a direct metalevel question. For example, if the question were

> Which course has more than one lecturer?

the interface must recognise that the structure of the domain precludes any particular course having more than one lecturer and so the question should not be translated into a database query. Instead it should produce a metalevel response which informs the user of the constraint. It is perfectly possible to produce a formal query corresponding to this question; the need to recognise constraint violation and respond appropriately is thus not specific to natural language interfaces but can arise with any front end that caters for non-expert users. The issue of representing constraints in the front end is important for several aspects of processing (it will be mainly discussed in the next section from the viewpoint of resolution of ambiguity).

The fourth potential advantage of natural language interfaces is the support that can be easily offered for a rather limited notion of dialogue; resolution of pronouns, for example, makes it much easier for a user to track down the information required when it may be difficult to formulate a single question. For example:

> Who teaches more than three courses?
> > a list of lecturers names
> How much are they each paid?

It is also quite normal to find support for fragmentary input, such as:

> Which students are registered for Basic Maths?
> > a list of students
> Algebra?

This sort of fragment has to be handled by the grammar, but there is also the quite distinct problem of dealing with incomplete input which is not licensed by previous discourse context, but which might be interpretable all the same. The latter is part of the general issue of how to cope gracefully with ungrammatical input.

Support for more extensive dialogue facilities which would be needed to handle questions such as

> Which of the lecturers who you listed earlier in response to my question about database courses earn more than £15,000?

is more difficult. For example most front ends have been able to almost ignore the problems of interpreting tense because most databases do not record temporal information explicitly (though see Hafner 1985, Clifford 1988), but if this type of dialogue is to be handled a fuller treatment may be required (Whittaker and Stenton 1989).

2.3 Limitations and constraints of the database interface task

There are limitations on interfaces to databases. It is not part of their function to act as advisor systems, for example. A question like:

> I want to take the course on particle physics. What courses should I take first?

might be interpreted as a query about which courses are requirements for particle physics. For this the interface would need some way of recording user supplied information (in this case, moreover, information about a goal which is outside the immediate task of database querying) and then some way of reasoning about this. This would currently only be feasible in a rather crude and limited manner, but we would argue that it is inappropriate to attempt it, even if the interface were thus enabled to respond to a somewhat wider range of questions. The difficulty is that to expand the front end so that it could answer some of these questions would give a misleading impression to the user about the capabilities of the system as a whole. While the user who asked such a question might know that they were querying a database which stored formal course requirements, the stating of the goal suggests that they want a more personal response, and that they have generally unrealistic expectations about the system's capabilities. To really act as an advisor system not only would the functionality of the interface have to be enhanced (in a way which is well beyond the current state of the art) but a great deal more knowledge which is outside the domain of the database, as we have defined it, would have to be incorporated, which would have serious consequences for portability. Of course our example database might form part of an advisor system, but general-purpose and portable database front ends cannot be expected to provide such capabilities.

In the next sections we will make the assumption that we are primarily concerned with translation into a single query on an existing relational DBMS. There are several reasons why it is easiest to concentrate on the relational model; it is now very widely used, the majority of new database developments are based on it, much of the natural language interface work has concentrated on it, and it has reasonably well-defined semantics. As noted earlier, our discussion in general carries over to other data models; we will make an explicit comparison with Prolog based systems in 3.5.

At the moment the assumption that a single input question is translated (if possible) into a single database query has to be made if the implemented system is to be efficient[2]. There are two main reasons why we might wish to relax this constraint; the first is that certain "recursive" queries which could potentially be handled by the database cannot be expressed in a relational query language. One example on this database would be:

> Which courses have to be taken before the course on databases?

where we are potentially not only interested in the immediate prerequisite courses but also in their prerequisites and so on. The second reason for relaxing the single query constraint is that there are situations where it might be useful to access the database in order to determine the correct translation of an input question, to resolve ambiguity or to carry out inference, for example.

One last topic, which is very important in practice, is how portable the interface is designed to be. By portability we are referring here to moving the system to new domains. Other types of portability, using the system with different underlying DBMS, for example, may also have to be considered when building practical front ends, but these have fewer theoretical implications. No sophisticated system can be built which can be adapted to a new domain without some human intervention: the issues are how much training and experience that person needs and how long the process is supposed to take. If it is intended that the porting can be done by people other than the interface builders it is reasonable to assume that such people will have detailed knowledge of the new database, but little or no linguistic knowledge. Support for porting in this case will require extensive tools and a certain amount of training. In most cases the actual process of porting will be incremental, with basic functionality being provided quite quickly and then expanded, possibly in response to problems arising in actual use.

[2]Of course one input question may have to be related to another in order to resolve anaphoric references, for example.

In this section we have tried to show that there is far more potential power in natural language processing than using ordinary words and grammatical structures as syntactic sugar in a restricted query language, but also that the further we try and get from this approach the greater the difficulties become. Furthermore, as a natural language interface is most likely to be useful to relatively inexperienced users, some problems will arise because of their limited expertise which are not specific to natural language interfaces. The aim is to provide a system which does not require users to learn a specialised vocabulary, or to know the structure of the database, but though they have to have some idea of the contents of the domain they may have some misconceptions. To be realistic, we cannot expect current systems to cope well with very occasional users; some process of familiarisation at least is required. The next section describes some of the approaches to the central problem of question translation; the other functions of the front end are discussed further in Section 4.

3 Interpretation of a natural language question as a database query

The most basic function of the system is the interpretation as a database query of a natural language question which is linguistically correct and appropriate to the domain. This interpretation covers morphological, syntactic, semantic and pragmatic processing: morphological processing deals with the forms of individual words, syntactic processing is broadly definable as dealing with the structural form of the sentence; semantic and pragmatic processing are both concerned with meaning, but there is no agreed boundary between the two. From a computational viewpoint we can take semantics to refer to the production of a computer-usable meaning representation of the question (or any subpart of the question), and pragmatics to refer to any processing using this representation. However it is defined, the pragmatic processing carried out by database front ends is currently very task-dependent, and so, although linguistic theories of syntax and semantics are relevant, those of pragmatics are less directly pertinent.

3.1 Architecture

All systems have to do some syntactic processing, although the relative importance given to this varies between systems. It is necessary to be able to distinguish between

Who taught Smith?

and

Who is taught by Smith?

for example. In simpler systems the semantics may be defined directly in terms of the database query, but more sophisticated interfaces make use of intermediate representations, which are not tied to particular databases, and in fact are not specific to database access at all. To some extent which approach is taken is connected with the issue of coverage; if the coverage is small it is easier to build a special-purpose grammar, and to give lexical items a translation directly in terms of the database schema. The grammars that can be built in this way (somewhat misleadingly called *semantic grammars*) are very closely tied to the database structure and to the requirements of straightforward translation to a database query. This approach to building grammars was taken by the LADDER system and its successors (including Q&A), which incorporated ingenious tools to make the process of grammar building relatively easy for a non-expert user. But, as we have shown, it is desirable to be able to handle questions which are very indirectly connected to the database structure, and facilities other than direct interpretation as a database query are, at least, very useful. Since semantic grammars are not portable it is rarely worth the effort of providing a grammar with a large coverage; it is obviously desirable to make use of as much general-purpose information as possible, so in more recent systems the grammar has not been domain-specific.

We will therefore concentrate here on the architecture where an intermediate representation is produced; this is normally expressed in some form of logic. This architecture is illustrated in Figure 2. The *analyser* carries out syntactic and semantic processing, although as we will see later the semantic processing is not necessarily complete. (We use the term analyser rather than parser, because parsing is frequently taken to refer to syntactic processing alone.) The *translator* is responsible for producing the database query; the processing involved is primarily pragmatic by the definition given above, but in many ways this is an inappropriate term to use because the central process of translating from one formal representation to another more limited one does not have any counterpart in linguistic theories of pragmatics. In general the processing carried out in the translator is highly task-specific.

Of the systems listed in the introduction PHLIQA1, TEAM, IRUS, TELI, LOKI, JANUS and Natural Language can be said to conform to the architecture shown in Figure 2, as does our own system (they were all intended to be portable between domains, albeit with varying amounts of effort). Producing an intermediate form has a number of advantages: since the grammar is, to a large extent, general purpose, it can be used for other domains and for other applications than database interfacing. An intermediate representation can be more suitable for functions that do not require database access, such as response to metalevel questions, and it can also be better suited to represent-

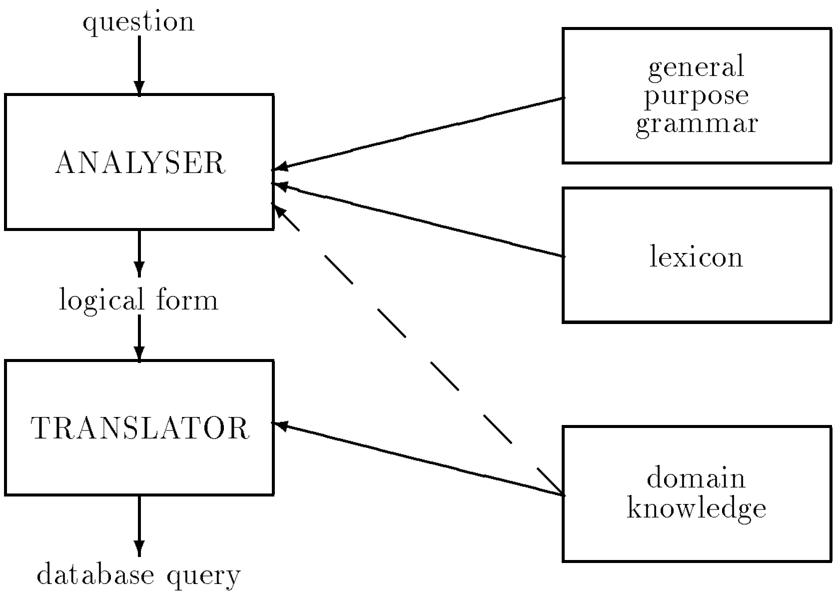

Figure 2: Typical system architecture

ing the less straightforward questions which may require extensive pragmatic processing, including inference, to interpret.

Most of the issues involved in the production of the intermediate representation, for example the type of analyser and the details of the grammar formalism, are common to all natural language interpretation systems and we will not discuss these in any detail, since they are well covered in the literature. (See Gazdar and Mellish 1989 and Allen 1987 for example). A state of the art example of a general-purpose system for producing a natural language meaning representation is SRI's Core Language Engine (CLE) (Alshawi et al 1989). Recent front ends, such as JANUS, are built around general-purpose analysers, indeed the JANUS system as a whole can be used as an interface to knowledge bases as well as conventional databases. We will therefore concentrate here on the issues on which the nature of the application has a direct bearing.

A general-purpose grammar should be linguistically plausible, both to be extensible and to take advantage of work in computational linguistics. Linguistic theories have become much more computationally oriented recently, and some large, general-purpose grammars of English are now available (eg the Alvey Natural Language Tools grammar (Grover et al 1989)). But the grammar has to be sufficiently robust to cope gracefully with sentence fragments and ill-formed input. It is also very important in the database context to have a good treatment of proper names, domain-specific identifiers, abbreviations and other items which cannot be part of a general-purpose lexicon (this is discussed below).

The details of the representation produced by the analysing stage vary greatly between systems. In order that the intermediate representation has at least the same power as the full relational query languages it is necessary to have some way of specifying universal and existential quantification. Many interfaces make use of a more powerful intermediate representation than is needed for database query, since natural language in general requires greater expressiveness (and in systems such as JANUS the extra power would be needed for interfacing to more powerful knowledge based systems); the transformation of the intermediate representation into a database query will then involve a certain amount of simplification, but even in this case the extra power, to deal with tense for example, could be required for dialogue, as noted earlier.

We will discuss two issues which are most relevant to the specific task in some detail; the first is the nature of the lexicon, the degree to which it is domain-specific and how it may be acquired, and the second is the problem of ambiguity.

3.2 The lexicon

Even in systems with a domain-independent grammar the lexicon is normally domain-dependent to some extent. Many systems have a core vocabulary which is supposed to be general purpose; this should include as a minimum the *closed class words*, that is determiners, prepositions, conjunctions and so on and verbs like "be" and "have", though domain-specific information may have to be added for these words. The core vocabulary will have to be more extensive if the interface handles complex dialogue and metalevel query. The rest of the vocabulary may be regarded as being domain-specific to the extent that it is replaced whenever the system is ported to a new domain[3]. TEAM is the paradigm example of this approach. It incorporates tools for vocabulary acquisition which are essentially driven by the structure of the database, but the syntax of the word to be defined is elucidated by a series of questions which require the user to know very little about linguistics, and instead ask for grammaticality/meaning judgements on example sentences. For example, when acquiring the verb "cover", the user would be asked whether:

> An area is covered.

is equivalent to:

[3]This applies whether vocabulary items retain their normal senses or have special application meanings, and whether they belong to technical vocabularies or to the common vocabulary

Something covers an area.

This approach is also taken in other systems and has been developed further in TELI and the CLE but there are problems with it. It is difficult to extend to verbs taking verbal and sentential complements for example, and although it was argued (in Grosz et al 1987) that these are not needed in database contexts this seems over-optimistic. For example:

Which courses require students to take "Basic Maths"?

We will argue later that it is better to have a notion of domain structure than to translate the logical form directly into database terms; if this is accepted the process of lexical acquisition would have to be prompted by the domain structure rather than the database structure and this is considerably more complex, although it could lead to better coverage. The availability of large lexicons for English, derived from machine readable dictionaries (Carroll and Grover 1989), may mean that acquisition of syntactic information from the user becomes less important though that of acquiring other information remains.

Besides vocabulary which refers to the structure of the database, the interface also has to cope with items corresponding to field values, which will normally be far more numerous than the structural ones. It is useful to distinguish between three classes of field values which require somewhat different treatment; first, numerical fields (eg a lecturer's salary — 15000) and identifiers (eg a course code — C23), second, proper names (eg a persons name — "T. P. Smith"), and third, strings corresponding to ordinary words (eg a course name — "Databases"). We cannot assume that all these items will have complete prespecified lexical entries since new values will be added when the database is updated, and if the database is large the size of the lexicon might become prohibitive.

The first class, identifiers and numerical items, presents no particular problem since these normally have a well-defined format; if this is specified as part of the domain-dependent knowledge, the potentially valid strings can be recognised and classified during morphological processing. The third class, ordinary words which can occur as field values, have to be specified in the lexicon, since their morphological, syntactic and semantic behaviour has to be properly specified in order to allow reasonable flexibility in question phrasing, so the actual database can only really be used as a prompt. For example if "Databases" is the name of a course the word "database" should be in the lexicon and the grammar should recognise "database course", "course on databases" and so on. It is reasonable to assume that there is a limited set of such words for any particular database and that these are explicitly enumerated somewhere, otherwise producing some formal queries would require trial and error.

The best way to treat proper names is somewhat less clear. The structures of personal names, for example, should be part of the domain-independent knowledge, so that if the database field contains initials and surname, strings like "Mr. T. Smith", "Mr Thomas Smith", and "Smith" should be recognised as potentially (but not necessarily) equivalent to "T. P. Smith", for example. Proper names are frequently obtained from the database itself. (INTELLECT in effect uses the database as the main part of its lexicon, but this only supports very limited coverage.) There are disadvantages with this; firstly there may be serious problems with efficiency and/or consistency, especially if the database is large and frequently updated; secondly, without some facilities for dealing with unrecognised items, some questions would not be accepted:

> Is there a lecturer named Tomlinson or Thompson?

Alternatively it is generally possible to recognise that strings which are not in the lexicon are proper names from the context of the rest of the sentence, but this may fail if the question is problematic in other ways. (See Kaplan, Mays and Joshi (1980) for a discussion of these issues and some possible approaches).

It is also very desirable to allow vocabulary which is less directly related to database concepts. In the example question:

> Who teaches Smith mathematics?

mathematics was defined in terms of two courses "Basic Maths" and "Algebra" and "teach" was defined in terms of the relationship between lecturers and courses and that between courses and students. It is difficult to foresee what vocabulary is needed when the translation into database terms is indirect but without such vocabulary being defined queries may be very cumbersome, as in:

> Who lectures whichever of the courses "Basic Maths" and "Algebra" Smith takes?

Some work, in TELI for example, has concentrated on allowing words describing such relationships (as well as synonyms of existing lexical items) to be defined, by the user, when they arise. However this does not seem very feasible if the end user may make only very occasional use of the system.

It is largely because of the importance of this type of question, and the problems of acquiring lexical knowledge in general, that attempts have been made to use general knowledge, albeit of a rather limited kind, and a general purpose lexicon. Such information, together with the constraints imposed by the rest of the question, can sometimes be used to construct a query even if the words have not been previously defined. Ginsparg(1983) and Boguraev and Sparck Jones(1983) describe work along these lines that utilises general-purpose information about words which is expressed in terms of semantic primitives. Copestake and Sparck Jones (1989) describes an attempt to extend this approach by using limited inference on restricted knowledge connecting word senses, with less reliance on primitives alone. For example if algebra is known to be a type of mathematics it is possible to infer that a question about mathematics could refer to the algebra course. Both this process and the definition of words like "mathematics" in domain-dependent terms are facilitated by having a fairly rich description of the domain in a knowledge base, as will be described later. However acquiring the domain-independent knowledge needed even to test that this approach is worthwhile is a serious bottleneck. Again machine-readable dictionaries may eventually provide at least a partial source of such information, see Alshawi(1989) and Wilks et al(1989).

3.3 Ambiguity

The second major issue that we wish to discuss is ambiguity. (In fact a further difficulty with providing a large general lexicon is that it increases this already serious problem.) The various sources of ambiguity and the possible ways of treating them are important for any non-trivial natural language system. We will distinguish two main classes of linguistic ambiguity, *lexical ambiguity* and *structural ambiguity*. Ambiguities resulting from either cause may or may not be resolved when a sentence is interpreted with respect to a particular domain. We will consider those that can be resolved first. For example:

> Give me a list of students and lecturers with salaries of less than £12,000!

is syntactically (structurally) ambiguous between an interpretation where both the students and the lecturers have salaries of less than £12,000 and the interpretation where only the lecturers' salaries are specified. Although general world knowledge does not resolve this ambiguity, in our example domain only the second interpretation is possible. Similarly "course" is lexically ambiguous but

> How many courses are there?

can only refer to lecture courses in this domain (and not golf courses, navigational courses, courses of bricks or courses of a meal, for example).

The issues here are how to determine and represent the domain constraints and how to apply them to resolve the ambiguity. There are three classes of constraint to be considered; those which are explicitly recorded in the DBMS, those which are implicit in the database schema, and those which may be part of the knowledge of an expert user but are not derivable in any way from the DBMS. The relative importance of these classes will depend on the sophistication of the underlying DBMS; many commercial systems have very limited facilities to represent and enforce constraints, but this is an important area of research in database systems (see Date 1983). It is obviously desirable that as few constraints as possible need to be supplied specifically for the natural language interface so an underlying DBMS which supports declarative descriptions of constraints is very useful; however we will not assume the existence of such facilities here.

The restriction needed to disambiguate the example above, that only lecturers have salaries, could be determined directly from the database schema and enforced either in a special purpose grammar (which is undesirable for the reasons we mentioned at the beginning of the section) or by excluding one possible logical form (we will consider this below). Other restrictions may be more subtle. Consider, for example:

> Which courses does every lecturer teach?

This is ambiguous between

> List the courses such that all lecturers teach each course.
> For all lecturers list the courses that they teach.

(Notice how difficult it is to phrase such questions unambiguously.) Our example domain includes the constraint that there is only one lecturer per course and, assuming the user knows this, the question is not ambiguous. But this constraint is only implied by the database schema. To make it explicit to the interface the relationship between lecturers and courses has to be specified as domain knowledge. The schema cannot be regarded as though each relation is an isolated frame. This particular type of constraint is not straightforward to encode in a special-purpose semantic grammar.

Some further constraints may not be specified in the schema at all. If there is a domain constraint that no-one can be enrolled for a course without fulfilling the formal requirements then:

> List the students who take the operating system course and the ones who take the database course who haven't fulfilled the operating system course requirements!

is unambiguous. But consider

> List the students who take the operating system course and the ones who take the database course who haven't fulfilled the C programming course requirements!

where the information that C is a prerequisite for the operating system course but not the database course is stored in the database. Here the information is properly data rather than knowledge (and is contingent rather than necessary in the domain); it should not be duplicated in the interface, so the ambiguity cannot be resolved without a database query.

As we mentioned in the introduction, representation of constraints is important for reasons other than the resolution of ambiguity. We will come back to the general issue of knowledge representation later. Assuming that constraints are represented in some way, there are essentially two ways of applying them to resolve ambiguity within the architecture that we have been describing. Some information can be used during analysis to exclude or order the analyses that would otherwise be produced. This often involves the use of selectional restrictions, for example "study" might be restricted to taking a student as an agent and a course as an object. This would resolve the ambiguity in:

> List all mathematics lecturers and students who study mathematics!

Alternatively selectional restrictions may be ostensibly domain-independent, although sets of selectional restrictions which are detailed enough to give a good degree of disambiguation always seem to exclude a high proportion of reasonable sentences. For example if "require" is specified as only taking a human agent:

> Which courses require students to take COBOL first?

will not be analysed.

It is simpler to use only domain-independent information during analysis, especially if translating the logical form into the database query may involve inference. In this case, to maintain some efficiency, the representation produced by the analyser may be a pseudo logical form which is not fully disambiguated; a single "packed" structure can represent various possibilities for quantifier scoping and prepositional phrase attachment (see Tomita 1985, Alshawi et al 1989). A packed structure for representing lexical ambiguities has been used by Hirst (1987). (Anaphoric references will also be unresolved and the relationship between the parts of compound nominals may be left unspecified.) Disambiguation is then regarded as part of the translation process.

Problems of ambiguity get worse very quickly with more complex domains. Although the example database is very simple, it is actually rather more structurally complex than the databases that many researchers report having used, and it seems that some issues of ambiguity (and certain other problems) have been underestimated because of this. As we mentioned earlier some ambiguities are not resolved by domain knowledge. For example:

Which courses does every student study?

is structurally ambiguous.

Who takes the course on databases?

could be asking about students or about the lecturer. Here there is a lexical ambiguity in "take"; "who" is not genuinely ambiguous but is underspecified with respect to its interpretation in the domain, since it refers to people and does not differentiate according their status as student or lecturer. (Obviously a more specific noun phrase like "which students" would make the sentence as a whole unambiguous.) But the example given illustrates one of the potential problems of an over-restricted vocabulary; if only one translation of "take" has been specified the system would not detect the ambiguity and considerable difficulties could be caused if the system's interpretation is not the one that the user intended. If the ambiguity is recognised, and there is no prior context such as a series of questions about students, the only reasonable course of action is to get the user to choose between the alternative readings. But a very similar query

Who takes more than one course?

is much more likely to be about lecturers than about students if practically all the students take several courses. Trying to use such information in a motivated way to get preferred readings is very difficult, and could be dangerous as the system's conclusions cannot be certain.

It is thus important that ambiguity is detected and the user is consulted before going ahead with a (possibly expensive) database query, especially since the user may well not be able to tell from the results returned that the question has been misinterpreted. It is therefore necessary to find some way of specifying the database query, unambiguously, to the user and, since this cannot be in terms of the formal query language, one of the options is generating unambiguous paraphrases of the query. In general natural language generation is much less well studied than interpretation, and unambiguous sentence generation is far from easy, but to some extent a rather simple, template-based approach will work here (eg Waltz 1978, Codd et al 1978). But it is hard to obtain paraphrases which are comprehensible and unambiguous in this way, and more powerful and hence more flexible generators (which may also be relatively portable) appear preferable, though they are hard to control (McKeown 1983, Mueckstein 1983, Boguraev and Sparck Jones 1984). (It may also be desirable to generate a natural language response to metalevel questions or in circumstances where there appears to have been a user misconception, or possibly in order to summarise the results of a database search; if these functions are supported, generation will have to be more complex, as we will consider briefly later.)

3.4 Translating the logical form into a database query

We now turn to the problem of translating a logical form into a database query. If the predicates used are all completely defined in terms of the database and they are combined in a way that does not conflict with any of the database constraints, this process is a fairly straightforward matter of combining the various parts of the query in the specified way. But a considerable range of problems can make things much more difficult. As we mentioned above the logical form may be underspecified in that ambiguities have not necessarily been resolved, and the relationship between the parts of freely-formed compound nominals, such as "mathematics lecturers", will not have been determined. Furthermore many lexical items such as "who" and "have" will have non-unique interpretations as far as the database is concerned.

The treatment of these problems in general depends on the application of a combination of the domain constraints and the constraints from the rest of the question, to resolve the various sources of vagueness. For example:

> List the maths lecturers!

where the relationship between maths and lecturers is not known, can be interpreted by interpolating the "teach" relationship as being equivalent to

> List the lecturers who teach mathematics courses!

and as illustrated earlier this will need further interpretation to recover the individual titles of the courses. If the logical form for a question violates some domain constraint it may be possible to coerce it into a valid form on the assumption that it is in some sense metonymic. Thus

> Who are Smith's lecturers?

can be interpreted as

> Who lectures the courses which Smith takes?

even if a direct relationship between students and lecturers has not been specified. However it is difficult to control such coercion especially in a complex domain, so it should be regarded as a last resort. It is much better to try and recognise the potentially important relationships in advance, even if they are not obvious from the structure of the database, so that they can be prespecified. For example, if the domain relationship corresponding to "teach" is specified, the coercion of the sentence above is less problematic, because no intermediate entity need be postulated.

These issues have been referred to as *local pragmatics* (Hobbs and Martin 1987), and are the sort of problems for which an extensive domain knowledge base may be useful. We have frequently referred to information which is domain-specific but not encoded in the database schema, and indeed the case for some sort of database-independent domain representation has been made several times, for example by Konolige(1979, 1981). However it is only comparatively recently that systems have been built which assume that a rich domain model (as distinct from the database schema) exists, which represent that model explicitly using a well-defined formalism and which store the information in a centralised knowledge base. In TEAM, for example, the details of the representation used to implement the so-called *conceptual schema* are unclear and some of the domain-specific information appears to have been distributed among processing modules. By contrast IRUS (in its later revisions, Stallard 1986) and JANUS make use of the NIKL formalism in order to represent this kind of knowledge; in combination with the logical form and the database contents the whole can be regarded as a hybrid system analogous to KRYPTON (Brachman, Gilbert and Levesque 1985). We also regarded our own system in this way (Copestake and Sparck Jones 1989). The existence of such an explicit and localised knowledge base makes it much easier to expand the system's capabilities without having to duplicate the information in different modules.

We think that the provision of an explicit domain model is essential as a buffer between an initial domain-independent interpretation of a sentence and the final interpretation in terms of the database schema. The central point is that although the relational data model at least has relatively clean, well-defined semantics the basic objects do not correspond directly to the sort of ontology which appears to be intuitively assumed in ordinary circumstances[4], and therefore do not correspond directly with natural language. The needs of the interface for a natural representation of the database domain are similar to those of the designer of the database. The similarities between the more structured AI approaches to knowledge representation and semantic data models such as the Entity-Relationship model and the various binary relationship models have been frequently noted. In the work reported in Copestake and Sparck Jones (1989) we constructed domain models for our interface which were essentially based on binary relationship models of the databases.

We illustrate this approach in Figure 3 which describes the central structure of the example domain.

In this domain model the most important relationships are represented explicitly. These are modelled in the relational database illustrated in Figure 1 by using foreign keys which in effect implement the notion of *feasible joins*. But this is not adequate for all purposes as the relationships are not explicit and their semantics cannot be completely specified in the relational model itself. By considering a domain model, rather than the database itself,

[4]See Kent(1978) for an important discussion of the naturalness of various data models

is a non lexical entity type

is a lexical entity type

is a relationship (may also be an entity)

is a subtype of

partial-functional dependency — eg a person has at most one address

cannot have null value

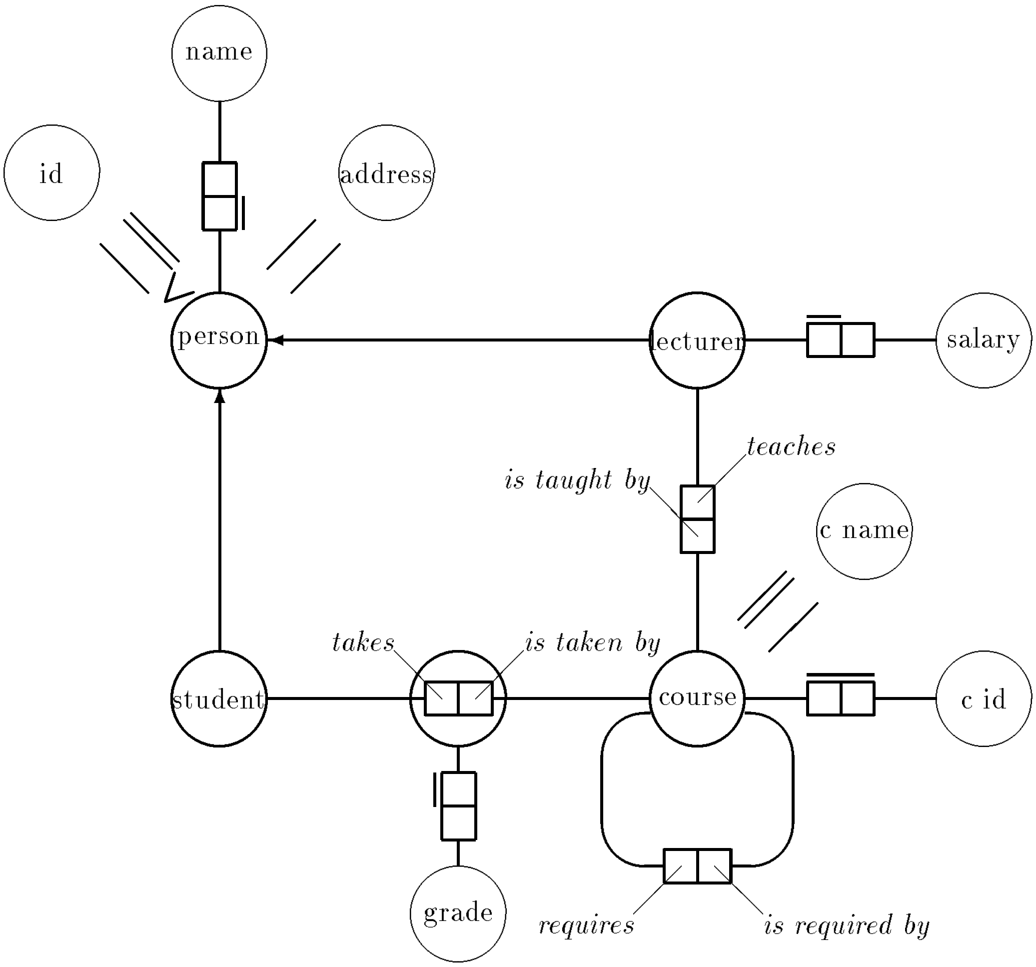

Figure 3: The central structure of the example domain

it is easier to see which relationships are important and therefore what sort of vocabulary is most needed by the interface.

A query representation in terms of the domain model can be constructed as an intermediate between the domain-independent logical form and the schema-dependent query. This enhances modularity because little change is needed for the interface to be used with non-relational models and it insulates the interface from the details of the database schema, which even in relational databases may have been determined for operational reasons. For example instead of having LECTURER and STUDENT relations as well as the PEOPLE relation as shown in Figure 1, we could have chosen to have just a PEOPLE relation with a ROLE column which could take values indicating whether the person was a student or a lecturer (or both) and a SALARY column which could take a null value.

Concepts which are most naturally thought of as entities can be represented as entities in the domain, even though they may be only partially represented in the database schema. For example the SHIPS database used in TEAM had a boolean valued column which just indicated whether or not there was a doctor on board a ship and there was no other information about doctors in the database. It is easier to isolate the problem illustrated by a question such as:

> Who is the doctor on the Vincennes?

if "doctor" is a valid entity in the domain model and the question is first translated into domain terms.

The local pragmatic issues mentioned above can be regarded as involving specialised inference on the domain knowledge base, as can the more systematic process of translating question words which are defined in terms of other domain concepts and not directly in terms of the database. Metalevel queries can also be evaluated against the domain model. The possibility of using inference to determine the domain-specific meaning of words from general-purpose knowledge alone was mentioned earlier. However the restriction that the database may not be accessed until the question has been fully interpreted implies that the inference will be limited to what can be achieved using the domain knowledge alone without the ground clauses in the database. It is obviously desirable that this restriction be lifted if this could be done without sacrificing too much efficiency. This issue of connecting knowledge bases and databases to allow efficient inference is important for many more applications than natural language interfaces (see for example Jarke 1986 and more generally Brodie and Mylopolous 1988), and is probably best investigated in this more general context, before attempting to apply the results to natural language processing. In the context of research into natural language query systems in general, rather than access to conventional databases, it is simpler to assume that there is no such constraint, and though it may be desirable to maintain a conceptual distinction between data and knowledge there is no reason to assume that there is such a rigid distinction between actual operations on them.

3.5 Interfaces to Prolog databases

In the context of systems with tighter connections between front-end and database, natural language interfaces to Prolog databases are interesting, because the knowledge/data distinction is not built into the architecture and so need not be enforced. (Of course this has disadvantages; it must be emphasised that although there are severe limitations on the knowledge representation capabilities of conventional databases, the exploitation of these limitations gives them an inherent efficiency advantage over more powerful systems. Some Prolog systems are being developed which incorporate some of the facilities provided by conventional databases. Such facilities will be necessary before Prolog can be used in anything other than small scale database applications.)

It is necessary to distinguish between the various possible notions of a Prolog database. They may be straightforwardly analogous to relational databases and just contain ground clauses. If we assume that the interface functions by sending off a single query, which may then be optimised, there are no really significant theoretical differences from the relational database case as far as the natural language interface is concerned. Chat-80 (Warren and Pereira 1982) can essentially be thought of in this way.

If the Prolog database essentially contains ground clauses alone but there is no restriction on the interface's access to the data then the design of the front end will be somewhat different. The database interface built, for illustrative purposes, on top of the CLE is of this type, for example (Alshawi et al 1989). Prolog's built-in inference mechanism can be used to carry out some of the processes that we have suggested are necessary during translation. Such a system might of course be very inefficient; this can be avoided by, in effect, programming the knowledge base, but this will not be feasible if the knowledge base is to be built by a database expert rather than the interface designer. So here again it is useful to think in terms of a higher-level domain model that can be constructed by the database expert and automatically compiled into Prolog clauses. There will thus be limitations on the type of knowledge stored and on the inference carried out, but these restrictions need not be as strict as if the data is only accessed after a full query has been produced.

If the database itself contains a large number of non-ground clauses, that is if it takes on more of the characteristics of a knowledge base, it becomes far more difficult to specify the domain. This is because the limitations of such a knowledge base are far less obvious than those of a simple database, and it is therefore difficult to ensure that there will be no discrepancies between any knowledge provided in the interface and the existing knowledge in the underlying system. But extra knowledge will be needed if a natural language interface is added, to allow interpretation of a sufficiently broad vocabulary for example. This is, in effect, the situation that has to be addressed when building a natural language interface to an existing expert system. Frost(1989) describes the use of the functional data model to describe the domain in such circumstances. But producing such a domain description is not straightforward; in general terms it is much easier to build an interface if the conceptual distinction between knowledge and data can be maintained.

4 Other front end functions

4.1 Robustness and response to problematic input

The main topic we will consider here is how to detect input that is in some way "incorrect" and how best to react to it. In the discussion we will distinguish between real mistakes, that is spelling errors and so on; misunderstandings, where the user is not aware of the nature of the domain and asks a question which is for some reason inappropriate; and system failures, where the user's question is appropriate but the system cannot interpret it. Of course the interface cannot distinguish these classes of problems, but it is important that its response is as helpful as possible. In the case of trivial mistakes, such as typographical errors, the interface should suggest corrections (the corrections should not be imposed since there is no certain way of distinguishing spelling errors from unrecognised but valid words). If the problem appears to be more serious the user should be given relevant information so that they can reformulate the question (or abandon it if it is completely inappropriate). Some of the potential problems with inappropriate input were touched on earlier, in the discussions on vocabulary, the grammar and ambiguity. Codd's RENDEZVOUS system (Codd et al 1978) was an early example of a system which took this problem seriously. Since then a considerable amount of work has been done, but inadequate response to problematic input is one of the main failings of current systems. We can only attempt to highlight some of the more important points here; see Webber(1986) for a detailed discussion of the whole topic and Stenton(1987) for a review of work on co-operative dialogue.

Practical natural language interfaces have to include spelling correction; the basic technology is well known (although there is considerable research going on) but it can very easily be overapplied. It is not just pointless but counterproductive to try and coerce every word in the input into one which is in the system's lexicon. If the user has not in fact made a spelling mistake but has used a word outside the system's vocabulary it could easily make the system's reaction completely mystifying. One advantage of having a larger dictionary available, even though not completely connected to the domain, is that it is easier to tell which words are misspelt, rather than not in the domain-specific lexicon. Idiosyncratic abbreviations may also occur in the input; these are very awkward to deal with but are discussed along with spelling errors in Means(1988).

A variety of strategies have been tried for coping with input which is grammatically ill-formed (which may sometimes be the result of spelling or typographical errors not being detected because they produce a valid although incorrect word). Mellish(1989) reports work on a grammar-independent heuristic, other techniques are grammar-dependent. Several relevant papers appeared in the American Journal of Computational Linguistics **9**(3–4) (1983) special issue on ill-formed input. It seems fair to say that all we have at the moment is a series of heuristics which cope with a proportion of syntactically ill-formed input but which will fail a considerable proportion of the time.

The second class of problem mentioned above was misunderstanding; there are a variety of ways in which the user may misunderstand the database's scope or contents. One important area is that a question, which can be translated into a database query, may nevertheless indicate that the user expects a certain set to have an extension in the database, when it is in fact empty. Consider for example:

> Which lecturers who earn more than £20,000 per year teach the courses on user interfaces?

If this is translated directly into a database query and returns a null result this might be because there are no courses on user interfaces, or because no lecturers earn more than £20,000. The user should be informed which part of the query returns a null result, as in Kaplan's (1982) work on unsatisfied presuppositions. Kaplan's technique could also be applied to input in a formal language, since it involves checking the subparts of the database query when a null result is returned. However it does not distinguish between sets which just happen to be empty and those which cannot have any members because of a constraint on the database domain. For example the lecturers' salary scale may stop below £20,000. We have mentioned the importance of specifying constraints in the interface (even if they are not actually implemented in the DBMS) earlier in the discussion of ambiguity. This is also relevant here because a query which appears to presuppose the existence of information which violates a constraint may indicate that the user does not fully understand the domain. Furthermore, for efficiency it is important to detect constraint violation before accessing the database. On the other hand an experienced user might want to check that the database has not become inconsistent if the constraint is not actually enforced in the DBMS, so it is not desirable to block such queries entirely.

There are other classes of user error; for example the user may ask about a type of information that is not in the database:

> Where does Dr. Robinson lecture?

It would be useful if the interface could produce a description of what it does cover, (McKeown 1985) rather than fail to translate the question (or even worse produce Dr. Robinson's address, or a list of courses, in response). The kind of domain model discussed in the previous section can be used as a basis for generating such a description, but it is difficult to make the description pertinent because in general the interface will not be able to infer exactly what the gaps in the user's knowledge are. Describing the entire database domain in detail will not be feasible unless it is trivially simple. In order to respond constructively to domain scope misconceptions additional information on the domain boundary would have to be provided in the knowledge base and this is difficult (Sparck Jones 1988). To be realistic we currently have to assume that the user has a good idea of what is in the database domain and will not ask questions that are not covered by it, or will be able to cope with less than fully informative responses.

The last area then is situations where the system's limitations prevent it responding to a question appropriately though the question content is legitimate. This may be because an input item is not in its vocabulary or because some particular relationship has not been encoded (for example a definition of "teach" might have been supplied so that lecturers teach courses but do not teach students). Acquisition of lexical knowledge from the user was discussed earlier, but the prior problem is helping the user to realise that it is a vocabulary failure rather than a database scope problem that is preventing the question from being recognised. This sort of difficulty is a major source of user frustration; exploitation of a large general-purpose lexicon (discussed in 3.2) could ameliorate the problem by making complete vocabulary failures relatively unlikely.

We have mentioned that it is useful for the system to generate natural language paraphrases of its interpretation of the question in contexts where the system has detected that the input is in some way problematic (because it seems ambiguous or contains a possible spelling error) but can nevertheless produce a query that seems likely to be appropriate. Since the system will not be able to detect some problems it is useful to produce such feedback as a matter of course, so the user can check that the question has not been misinterpreted. This facility, and the generation of cooperative responses mentioned above may require a sophisticated generator. Indeed cooperative responses especially may require multi-sentence output, and therefore the generator will need language processing capabilities, for maintaining thematic cohesion for example, which do not have analogues in the analyser.

4.2 Support for the inexperienced user

One area which is very important for natural languages interfaces, although in no way specific to them, is that of query optimisation. This is important just because the interface users are not expected to know any details of the system's implementation, and will therefore not know how expensive a query is likely to be. Although the relational model itself is supposed to hide such considerations from users, it is clear that in practice users of any database do have to have some idea of how costly a query is, especially on large databases. Naturally all the standard techniques for optimising the query which the front end produces can be used. The type of information which standard DBMS optimisers use might also be used to give the user some estimate of the cost of the query before it is run. Certain types of optimisation can make use of semantic information which is not stored in conventional DBMS (see King(1981) for example); it seems that at least some of this knowledge is of the type that may be required by the natural language front end anyway, and so could naturally be exploited.

A further related issue is packaging of the results. A long list of tuples may not be very informative, and it may be better to attempt to summarise the information as a graph or in some other way (see for example Brennan 1988). Although this issue is again not specific to natural language interfaces it may be desirable to generate a response in natural language. Summary of responses is considered in Kalita, Jones and McCalla(1986). There may be cases where it is impossible to package or summarise the response adequately, a very long list of names for example, where it seems that the user may have misjudged the contents of the database. This is similar to the optimisation issue in that the user can be given an estimate of tuple numbers before processing the query.

It is at least desirable that natural language interfaces should support database update as well as querying. However this creates some problems of its own. There are few references specifically about update, (see Davidson and Kaplan 1983, Salveter and Maier 1982), although several systems have supported at least limited update. Constraint enforcement is obviously critical here and there needs to be sufficient connectivity between the database and the front end to be able to inform the user of any problems with update in a comprehensible manner. Update also has implications for system architecture if querying relies on the maintenance of views (as was the case with TEAM) since carrying out update on these is problematic.

So although the central problem, on which most of the research in this field has concentrated, is that of transformation of an input question into a database query, it can be seen that the other issues discussed here are very important to the design of a usable interface. Most of them also require information to be supplied about the database domain. Some of them are not specific to natural language interfaces but their potential effects on the architecture are worth considering if they can make use of the same information as the natural language interface.

5 The practical uses of natural language interfaces to databases

As we said in the introduction it has turned out to be much more difficult to build useful natural language interfaces to databases than was originally expected. INTELLECT was the first general-purpose commercial system to be widely available. Although it was reasonably successful it was highly priced and required considerable customisation. A certain amount of user adaptation and experience was needed to offset the lack of flexibility caused by the heavy dependence of its language processing on the underlying database. In contrast Q&A was marketed as a low cost interface for personal computers. The customisation was carried out by the end user but as in INTELLECT the resulting interface mirrored the database very closely. Q&A has sold well but this is at least partly because its non-natural language facilities were good compared to the competition and in fact many of its users did not adopt the natural language interface. One or two other low cost natural language systems are available but Query-By-Example style interfaces are much more popular ways of attempting to provide user friendliness in the small system market. It seems likely that the single user market is rather limited; porting a natural language interface that gives a high level of flexibility to a complex database will almost certainly take longer than learning the formal query language and will require at least equivalent skills.

Natural language systems are certainly not automatically user friendly. We have suggested that they are less appropriate than a menu system for very restricted querying. One exception to this is the system described by Tennant et al (1983) which allowed the user to build up a natural language question from computer-generated subphrases displayed using menus; this had the advantage that the restrictions on the query were made explicit. Usable speech recognition would change the potential context and mode of use of natural language interfaces completely, since speech is useful in environments where menus, and even typing, are not feasible. Despite this the issues which we have discussed could nearly all apply to speech driven as well as text driven systems. However the difficulties of accurate speech recognition are so large that the technology involved in existing systems (even those such as VODIS

(Proctor and Young 1987, Young 1989) which are described as interfaces to databases) has very little in common with the natural language interfaces we have considered here. Current systems, especially those that attempt to handle connected speech uttered along noisy connections by users which the system has not been trained to recognise, can only function if the input is very highly constrained; the type of natural language techniques which have been developed to deal with input that is much less constrained are mostly inappropriate overkill in this situation. Speech input for other than highly trained users also presents problems like 'restarts' which do not figure in typed input. Thus although it is clear that syntactic, semantic and domain-specific pragmatic knowledge as used in natural language interfaces could in theory be used to provide constraints to improve speech recognition it is also clear that the architecture of the systems would need to be radically changed.

Some database domains are not suitable targets for natural language interfaces. One example is databases for CAD systems that are primarily dealing with graphical information for which a visual interface is far more appropriate. Even on tractable databases it can be difficult to express some complex queries in natural language, particularly because it is difficult to build up queries piecemeal. Applicability of natural language is also limited by the context in which many large conventional database systems are normally used, where improving the accessibility of data to untrained users is not a major priority. Natural language interfaces might improve access to data but it seems highly unlikely that they can replace significant numbers of specialist staff, and in fact the inverse effect is probably more likely.

But although it is reasonable to be cautious about how wide the utility of natural language interfaces to databases currently is, the commercial situation is more interesting than it has been for some time. Natural Language Inc's interface is the first of the more sophisticated natural language interfaces to be widely marketed (Manderfelli 1989). It aims at a higher-level market than Q&A since it is being sold for Unix workstations at a cost of \$5000 upwards (plus \$10000 upwards for the porting tools). It has a much wider coverage than previous generally available commercial interfaces, but in attempting to derive a query from any input where the words are in its vocabulary it can produce some strange results. It is not clear how easy it is to port it to a substantially different domain. But it does show that the sort of techniques which have largely been confined to research establishments can be used on a commercial system. More recently IBM have announced that they have been working on a Prolog-based system which may become a product. A considerable number of other systems have been used in a commercial environment on a somewhat smaller scale; in general customisation is carried out by the interface builders. New systems can be built with far less effort than was possible a few years ago by utilising the sort of general-purpose language processing systems and tools mentioned in Section 3. The kind of coverage and facilities discussed in Section 2 could therefore be fairly generally available.

It is however very difficult to evaluate the performance of existing systems as there is very little published literature on the practical testing of natural language interfaces to databases. Although it is important that systems have a wide coverage, it is clear that measurements of the percentage of sample questions correctly translated are no substitute for serious testing with real users. Even if the sample questions are representative it is impossible to predict how easy it is to learn to use the interface and how adequate its response to problematic input is. But the results of experiments with users in a particular enviroment may not be widely applicable because of the many contextual factors involved. Variations in domain or usage patterns, for example, make it hard to extrapolate from one application to another and also to compare different interfaces. Simple domains or undemanding uses may mask real system limitations.

Many research systems, including our own, are however not complete or robust enough to be evaluated extensively with real users. This is unavoidable, since large scale resources are needed to build a full system and to carry out this sort of experiment, and limited testing is not unreasonable for exploratory work, as long as the objectives and results are clear enough to be potentially applicable to usable interfaces. Such research systems should not be regarded as wholes but rather as test frames which are just sufficiently complete to support the main work. It is far better to do this than to compromise objectives and to obscure results by trying to expand an unsuitable existing system to support new research.

One way of studying actual users without constructing a complete system is to pretend to the experimental subjects that they are interacting with a computer system but to actually translate their questions manually; these are referred to as "Wizard of Oz" experiments (see for example Carbonell 1983). This is valuable in that it can provide realistic testing material for later use with a computer system, and to some extent experiments can be done on how users learn under various restrictions. Comparisons with other types of interfaces, for particular applications, can also be made. But it would probably be impossible for a human to adequately simulate possible system responses other than results from a database search.

One point that is clear is that changing the human-computer interface by allowing natural language queries can have far-reaching effects on the whole process of interaction. Systems in commercial use, such as INTELLECT have had to provide supplementary information displays to accompany natural language interaction (Harris 1984), and the implications of speech recognition would be far more extensive. Current research on multi-media interfaces (Wahlster 1989) illustrates the importance and the advantages of considering the interface design as an integrated whole.

It also seems likely that natural language interfaces may be useful in the context of less traditional database applications and models. As we have already seen, in Section 3, the distinction between databases and knowledge bases is narrowing and some recent front ends are not restricted to conventional databases. Apart from the development of such more powerful question-answering systems, access to databases of the kind considered in this paper may also be envisaged as one function of a so-called integrated information system offering the user different types of information resource including, for instance, document and text bases as well as conventional databases. Taking a natural language question as a single entry point for different kinds of information search, say on data and documents, as crudely done in LUNAR and suggested in Sparck Jones and Tait (1984), is an attractive idea; but in the light of the material constraints we have found apply to natural language database access, is also one that needs much more and very careful investigation.

6 Conclusions

As we stated at the beginning, natural language interfaces to databases are practical possibilities because databases provide explicitly limited domains. However the history of the research done in this area has shown that unless attempts to exploit this are very carefully motivated, the resulting techniques will not be sufficiently general to apply to different domains, let alone scale up to more complex ones. Building a good, portable, natural language interface to databases is difficult and time consuming and currently the usefulness of such systems is far more limited than we would like. For every attempted short-cut there are penalties. Whether the problems are unacceptable can ultimately only be determined by testing the interface in the type of environment for which it is designed. It is far from obvious what the constraints of the task of database querying are, and how they can be exploited (Sparck Jones at al 1984). Researchers primarily interested in dialogue, for example, quite rightly regard database querying as less suitable for their purpose than interaction with an advisor system. But this does not mean, in general, that the issues investigated can never arise in database querying, only that it is more difficult to find natural-sounding examples. So the restrictions on the database querying task are not obvious and probably not absolute; we can assume that there are some aspects of natural language processing which would hardly ever arise in the context of database querying, but this may be somewhat irrelevant since there are plenty of problems in providing the more basic capabilities which are generally agreed to be necessary. The conclusion is that the restrictions which we must attempt to exploit are not restrictions on the task but restrictions on the domain of any particular database.

It is not possible to interface a general-purpose natural language interface to a standard DBMS and get the sort of capabilities needed without a significant intermediate interface incorporating a knowledge base that encapsulates the restricted domain knowledge. There will be a tradeoff between portability and coverage, but an interface developed for a specific database, which incorporates too many database specific short cuts, will probably not be able to cope with uses of the database that the designers did not foresee (which are inevitable and desirable with flexible interfaces), and with the evolution and development of the database.

As we have shown many of the issues that arise when considering natural language front ends are potentially applicable to any database interface which facilitates access by inexperienced users. Any such interface may need to give the user an estimate of the time a query will take to process, and the number of tuples which will be output, to detect 'presupposition' failures, to cope with semantic constraint violation and to give metalevel information. Conversely the more motivated natural language front ends to databases give some indications of the problems of providing interfaces to other application systems. The differences between conventional databases and deductive (Prolog) databases were mentioned earlier; the important advantage of a conventional DBMS which is not shared by more general knowledge base systems is that the limitations on its domain can be made explicit relatively easily.

The most important result of the work on natural language interface to databases has been to indicate those aspects where future advances will come from improvements in general natural language processing or in database technology, and aspects which are specific to natural language interfaces to databases. Large scale (ie commercial) projects applying existing research techniques, and integrating the natural language interface with the further functionality needed to support a naive user, will result in better commercial systems. It is not possible to determine what proportion of the potential need such interfaces would satisfy, but it seems unlikely that they would be able to cope with an untrained user attempting to access a complex database. The amount of basic research which is specifically aimed at natural language interfaces to databases has declined in recent years; this is probably appropriate since building a complete, testable, interface is a very large scale project and it has become much clearer how the requirements for a sophisticated front end are shared by other natural language interfaces and by "intelligent" databases. Much of the research which will eventually contribute to improvements in front ends to databases can and should be done in more general contexts. It seems likely, for example, that significant theoretical advances will come from work involving interfacing databases and knowledge bases, the use of machine readable dictionaries to aid knowledge acquisition, and a better understanding of dialogue, all of which have much more general relevance.

References

Allen, J(1987) *Natural language processing,* Benjamin/Cummings, Menlo Park, CA

Alshawi, H et al(1989) *Research programme in natural language processing, final report,* SRI Cambridge Computer Science Research Centre, Cambridge

Alshawi, H(1989) 'Analysing the dictionary definitions' in Boguraev, B and Briscoe, T (eds), *Computational lexicography for natural language processing,* Longman, London, pp153–169

Ballard, B and Stumberger, D(1986) 'Semantic acquisition in TELI', *Proceedings of the 24th Annual Meeting of the ACL,* New York, pp20–29

Bates, M, Moser, M G and Stallard, D(1986) 'The IRUS transportable natural language database interface' in Kerschberg, L (eds), *Expert database systems,* Benjamin/Cummings, Menlo Park, CA, pp617–630

Bates, M and Weischedel, R(1987) *Tutorial: Evaluating natural language interfaces,* 25th Annual Meeting of the ACL, Stanford CA (Bolt, Berank and Newman Inc Cambridge, MA)

Binot, J L et al(1988) *LOKI: A logic oriented approach to data and knowledge bases supporting natural language interaction,* Scicon Ltd, London

Boguraev, B K and Sparck Jones, K(1983) 'How to drive a database front end using general semantic information', *Proceedings of the Conference on Applied Natural Language Processing,* Santa Monica, CA, pp81–88

Boguraev, B K and Sparck Jones, K(1984) 'A natural language front end to databases with evaluative feedback' in Gardarin, G and Gelenbe, E (eds), *New Applications of Databases,* Academic Press, New York, pp159–183

Brachman, R J, Gilbert, V P and Levesque, H J(1985) 'An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON', *Proceedings of the 9th IJCAI,* Los Angeles, pp532–539

Brennan, S E(1988) 'The multi-media articulation of answers in a natural language database query system', *Proceedings of the 2nd Conference on Applied Natural Language Processing,* Austin, pp1–8

Brodie, M L and Mylopolous, J(1988) *Readings in artificial intelligence and databases,* Morgan Kaufman, San Mateo, CA

Bronnenberg W J H J et al(1980) 'The question answering system PHLIQA1' in Bolc, L (eds), *Natural language question answering systems,* Macmillan, London, pp217–305

Carbonell, J G(1983) 'Discourse pragmatics and ellipsis resolution in task-oriented natural language interfaces', *Proceedings of the 21st Annual Meeting of the ACL,* MIT, Cambridge, MA, pp164–168

Carroll, J and Grover, C(1989) 'The derivation of a large computational lexicon for English from LDOCE' in Boguraev, B and Briscoe, T (eds), *Computational lexicography for natural language processing,* Longman, London, pp117–133

Clifford, J(1988) 'Natural language querying of historical databases', *Computational Linguistics,* **14**(4), 10–34

Codd, E F et al(1978) *RENDEZVOUS Version 1: An experimental English-language query formulation system for casual users of relational databases,* Research Report RJ2144 IBM Research Laboratory, San José, CA

Copestake, A A and Sparck Jones, K(1989) *Inference in natural language front ends to databases,* Technical Report 163, Computer Laboratory, University of Cambridge

Damerau, F(1980) *The transformational question answering (TQA) system: description, operating experience and implications,* Report RC8287, IBM Thomas J Watson Research Center, Yorktown Heights, NY

Damerau, F(1981) 'Operating statistics for the transformational question answering system', *American Journal of Computational Linguistics,* **7,** 30–42

Date, C J(1983) *An introduction to database systems: Volume 2,* Addison-Wesley, Reading, MA

Davidson, J and Kaplan, S J(1983) 'Natural language access to databases: Interpreting update requests', *American Journal of Computational Linguistics,* **9,** 57–68

Frost, D P(1989) *The design of a natural language interface for medical expert systems,* PhD thesis, University of London (University College and Middlesex School of Medicine

Gazdar, G and Mellish, C(1989) *Natural language processing in LISP,* Addison-Wesley, Reading, MA

Ginsparg, J(1983) 'A robust portable natural language database interface', *Proceedings of the Conference on Applied Natural Language Processing,* Santa Monica, CA, pp25–31

Green, B et al(1961) 'BASEBALL: An automatic question answerer', *Proceedings of the Western Joint Computer Conference,* **19,** 219–224; reprinted in Grosz, Sparck Jones and Webber (1986) pp545–549

Grosz, B, Sparck Jones, K and Webber, B L (eds)(1986) *Readings in natural language processing,* Morgan Kaufman, San Mateo, CA

Grosz, B et al(1987) 'TEAM: an experiment in the design of transportable natural-language interfaces', *Artificial Intelligence,* **12,** 173–243

Grover, C et al(1989) *The Alvey Natural Language Tools grammar (second release),* Technical Report 162, Computer Laboratory, University of Cambridge

Haas, N and Hendrix, G G(1983) 'Learning without being told: Acquiring knowledge for information management' in Michalski, R S, Carbonell, J G and Mitchell, T M (eds), *Machine Learning,* Tioga Publishing, Palo Alto, CA, pp405–427

Hafner, C D(1985) 'Semantics of temporal data and temporal queries', *Proceedings of the 23rd Annual Meeting of the ACL,* Chicage, IL, pp1–8

Harris, L R (1977) 'User oriented database query with the ROBOT natural language query system', *International Journal of Man-Machine Studies,* **9,** 697–713

Harris, L R (1984) 'Experience with INTELLECT: Artificial intelligence technology transfer', *The AI Magazine,* **5**(2)**,** 43–50

Hendrix, G G et al(1978) 'Developing a natural language interface to complex data', *ACM Transactions on Database Systems,* **3,** 105–147; reprinted in Grosz, Sparck Jones and Webber (1986) pp563–584

Hirst, G(1987) *Semantic interpretation and the resolution of ambiguity,* Cambridge University Press, Cambridge

Hobbs, J R and Martin P(1987) 'Semantics of temporal data and temporal queries', *Proceedings of the 10th IJCAI,* Karlsruhe, West Germany, pp520–523

Jarke, M(1986) 'Control of search and knowledge acquisition in large-scale KBMS' in Brodie, ML and Mylopolous, J (eds), *On Knowledge Base Management Systems,* Springer Verlag, New York, pp507–522

Kalita, J K, Jones M L and McCalla, G I(1986) 'Summarising natural language database responses', *Computational Linguistics,* **12,** 107–124

Kaplan, S J, Mays, E and Joshi, A K(1980) *A technique for managing the lexicon in a natural language interface to a changing data base,* Technical Report MS-CIS-80-10, Department of Computer and Information Science, University of Pennsylvania

Kaplan, S J(1982) 'Cooperative responses from a portable natural language query system', *Artificial Intelligence,* **19,** 165–187

Kent, W(1978) *Data and reality,* North Holland, Amsterdam

King, J L(1981) 'QUIST: A system for semantic query optimization in relational databases', *Proceedings of the 7th International Conference on Very Large Databases,* Cannes, France, pp510–517

Konolige, K(1979) *A framework for a portable natural-language interface to large databases,* Technical Note 197, SRI International

Konolige, K(1981) *The database as model: A metatheoretic approach,* Technical Note 255, SRI International

Manderfelli, J L(1989) 'Natural languages', *Sun Technology,* Summer 1989**,** 122–129

McKeown, K R(1983) 'Paraphrasing questions using given and new information', *American Journal of Computational Linguistics,* **9,** 1–10

McKeown, K R(1985) *Text generation,* Cambridge University Press, Cambridge

Means, L G(1988) 'Cn yur cmputr raed ths', *Proceedings of the 2nd Conference on Applied Natural Language Processing,* Austin, Texas, pp93–100

Mellish, C S(1989) 'Some chart-based techniques for parsing ill-formed input', *Proceedings of the 27th ACL,* Vancouver, pp102–109

Moore, R C et al(1982) 'Panel: Natural language access to databases — theoretical/technical issues', *Proceedings of the 20th ACL,* Toronto, Ontario, pp44–66 and 169–171

Mueckstein, E-M M(1983) 'Q-TRANS: Query translation into English', *Proceedings of the 8th IJCAI,* Karlsruhe, West Germany, pp660–662

Perrault, C R and Grosz, B J(1986) 'Natural language interfaces', *Annual Review of Computer Science, Vol 1,* **,** 47–82

Proctor, C and Young, S(1987) 'Dialogue control in conversational speech interfaces' in Taylor, M M, Néel, F and Bouwhuis, D G (eds), *The structure of multimodal dialogue,* North Holland, Amsterdam, pp

Reiter, R et al(1983) 'A panel on AI and databases', *Proceedings of the 8th IJCAI,* Karlsruhe, West Germany, pp1199–1206

Salveter, S and Maier D(1982) 'Natural language database updates', *Proceedings of the 20th Annual Meeting of the ACL,* Toronto, Canada, pp67–73

Scha, R J H(1983) *Logical foundations for question answering,* PhD thesis

Sondheimer, N K et al(1981) 'Panel: Evaluation of natural language front ends to databases', *Proceedings of the 19th ACL,* Stanford, CA, pp29–42

Sparck Jones, K et al(1984) 'Panel: Natural language and databases, again', *Proceedings of the COLING 84, 10th International Conference on Computational Linguistics, 22nd Annual Meeting of the ACL,* Stanford, CA, pp182–193

Sparck Jones, K and Tait, J I(1984) 'Linguistically motivated descriptive term selection', *Proceedings of the COLING 84, 10th International Conference on Computational Linguistics, 22nd Annual Meeting of the ACL,* Stanford, CA, pp287–290

Sparck Jones, K (1988) *A note on robustness in front ends,* Computer Laboratory, University of Cambridge

Stallard, D G(1986) 'A terminological simplification transformation for natural language question answering systems', *Proceedings of the 24th ACL,* New York, pp241–246

Stenton, S P(1987) 'Dialogue management for co-operative knowledge based systems', *The knowledge engineering review,* **2,** 99–122

Storrs, G, du Boulay, B and Gray, P M D(1985) *A metadata advisor: some sample queries,* Department of Computer Science, University of Aberdeen

Tennant H(1981) *Natural language processing,* Petrocelli, New York

Tennant, H et al(1983) 'Menu-based natural language understanding', *Proceedings of the 21st Annual Meeting of the ACL,* MIT, Cambridge, MA, pp151–158

Tomita, M(1985) 'An efficient context-free parsing algorithm for natural languages', *Proceedings of the 9th IJCAI,* Los Angeles, pp756–764

Wahlster, W(1989) 'User and discourse models for multimodal communication', *Proceedings of the Hewlett-Packard Laboratories 1989 European Scientific Symposium,* Paris, pp115–131

Waltz, D L(1978) 'An English language question answering system for a large relational database', *Communications of the ACM,* **21,** 526–539

Warren, D and Pereira, F (1982) 'An efficient easily adaptable system for interpreting natural language queries', *American Journal of Computational Linguistics,* **8,** 110–122

Webber, B L(1986) 'Questions, answers and responses: Interacting with knowledge base systems' in Brodie, ML and Mylopolous, J (eds), *On Knowledge Base Management Systems,* Springer Verlag, New York, pp365–402

Weischedel, R M(1989) 'A hybrid approach to representation in the JANUS natural language processor', *Proceedings of the 27th ACL,* Vancouver, British Columbia, pp193–202

Whittaker, S and Stenton, P(1989) 'User studies and the design of natural language systems', *Proceedings of the 4th EACL,* Manchester, pp116–123

Wilks, Y et al(1989) 'A tractable machine dictionary as a resource for computational semantics' in Boguraev, B and Briscoe, T (eds), *Computational lexicography for natural language processing,* Longman, London, pp193–228

Woods, W(1972) *The Lunar sciences natural language information system,* Final Report, Bolt, Beranek and Newman, Cambridge, MA

Woods, W (1978) 'Semantics and quantification in natural language question answering' in Yovits, M (eds), *Advances in Computers,* Academic Press, New York, pp1–87

Young, S J(1989) *Final report: Alvey/SERC project MMI003, voice operated database inquiry systems, speech input,* Engineering Department, University of Cambridge