Chapter 10

Database Interfaces

Ion Androutsopoulos and Graeme Ritchie[1]

10.1 Introduction

Natural language database interfaces (NLDBIs) are systems that allow users to access information stored in a database by formulating requests in natural language. For example, a NLDBI connected to the personnel database of a company would typically be able to answer questions like the following. (We show user entries in *italics*, and system replies in **bold**.)

> *Who is the youngest employee in the sales department?*
John Smith.
> *What is his salary?*
$25,000.
> *Does any employee in the sales department earn less than John Smith?*
Yes, George Adams.

NLDBIs have received particular attention within the natural language processing community (see [8], [28], and [63] for previous reviews of the field[2]), and they constitute one of the first areas of natural language technology that have given rise to commercial applications.

This chapter is an introduction to key concepts, problems, and methodologies in the area of NLDBIs. It focuses on issues that are specific to NLDBIs, as opposed to general natural language processing methods, but it also provides examples from NLDBIs where the general techniques of the previous chapters of this book are used. As with the rest of this book, this chapter does not cover issues related to speech processing. This reflects the assumption in most current NLDBIs that the user's requests will be typed on a keyboard. As we comment briefly in section 10.2, however, we believe that speech technology will play an important role in future NLDBIs.

The rest of this chapter is organised as follows: section 10.2 provides a brief history of NLDBIs; section 10.3 presents the typical architecture and the main components of modern NLDBIs; section 10.4 discusses portability issues; section 10.5 highlights some advanced issues: the "doctor-on-board" problem, database updates, and meta-knowledge, modal, and temporal

[1] To appear in R. Dale, H. Moisl, and H. Somers (Eds.), "*A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*", Marcel Dekker Inc.

[2] Some of the material in this chapter originates from [8].

questions; section 10.6 discusses NLDBIs with restricted input, namely NLDBIs with controlled languages, and menu-based NLDBIs; section 10.7 concludes by reflecting on the state of the art and the future of NLDBIs.

10.2 Historical background

The first NLDBIs appeared in the late sixties, with the most well-known system of that period being LUNAR [83, 84, 85, 86], a NLDBI to a database that contained chemical analyses of moon rocks. LUNAR demonstrated convincingly that usable NLDBIs could be built. It also introduced some innovative techniques (e.g. in the treatment of quantifiers) that had a significant impact on subsequent computational approaches to natural language.

Several other NLDBIs had appeared by the late seventies (e.g. RENDEZVOUS [26], TORUS [61], PLANES [79], PHILIQA1 [68], and LADDER [45]).[3] Some of these early systems (e.g. PLANES, LADDER) used *semantic grammars*, an approach where – roughly speaking – non-terminal symbols of the grammar reflect categories of world entities (e.g. *employee_name*, *question_about_manager*) instead of purely syntactic categories (e.g. *noun_phrase*, *sentence*; see [2] for more information). Semantic grammars allowed selectional restrictions (discussed in section 10.3.5 below) to be encoded easily, and the resulting parse trees could be very close to logical formulae, eliminating the need to map from syntactic to semantic constructs. Semantic grammars, however, proved difficult to port to new knowledge domains (e.g. modifying a NLDBI to be used with a database about train schedules rather than employees), and were gradually abandoned.

In the early eighties, CHAT-80 was developed [80]. CHAT-80 incorporated some novel and ingenious techniques, and its implementation responded to queries very promptly. Its code was circulated widely, and it became a *de facto* standard demonstration of NLDBI capabilities. It also formed the basis of the experimental NLDBI which we will use as a source of illustrative examples here – MASQUE/SQL [5, 7]. We use examples from MASQUE/SQL, mainly because we can say with authority what it would do, and partly because we regard its architecture and mechanisms as typifying a large class of NLDBIs.

By the mid-eighties, NLDBIs had become a very popular research area, and numerous research prototypes were being implemented. Portability issues (see section 10.4) dominated much of the NLDBI research of the time. TEAM [36, 37, 59], for example, was designed to be configured by database administrators with no background in linguistics, and ASK [75, 76] allowed end-users to "teach" it new words at any point. ASK was actually an integrated information system, with its own built-in database, and the ability to interact with external applications (e.g. external databases and e-mail programs). The user could control any application connected to ASK via natural language requests. JANUS [22, 46, 81] had similar abilities to interface to multiple systems. DATALOG[4] [38, 39], EUFID [71, 72], LDC [13, 14], TQA [29], TELI [12], were also among the numerous research prototypes of the same period.

We note at this point that some natural language researchers use the term "database" to mean just "a lot of data". In this chapter, we mean much more than that. Most importantly, we assume that the data constitute a principled attempt to represent part of the world, and that they are structured according to a formally defined model. Database systems have evolved substantially over recent decades. The term "database systems" now denotes (at

[3] More information on the historical evolution of NLDBIs can be found in [11].

[4] This NLDBI has nothing to do with the subset of Prolog that is used as a database language [77].

least in computer science) much more complex and principled systems than it used to denote in the past. Many of the "database systems" of early NLDBIs would not deserve to be called database systems by today's standards.

Until the early eighties, the standard way to interact with database systems was to use special database programming languages (e.g. SQL [60]), which are difficult for end-users to master. NLDBIs were seen as a promising way to make databases accessible to users with no programming expertise, and there was a wide-spread optimism about their commercial prospects. In 1985, for example, Ovum Ltd. predicted that "*By 1987 a natural language interface should be a standard option for users of database management system and 'Information Centre' type software, and there will be a reasonable choice of alternatives.*" [49].

Since then, several commercial NLDBIs have appeared (e.g. Linguistic Technology's ENGLISH WIZARD, a descendant of INTELLECT[5] [43], which was in turn based on experience from ROBOT [40, 41, 42]; BBN's PARLANCE, derived from the RUS [21] and IRUS [16] systems; IBM's LANGUAGEACCESS [62]; Q&A from Symantec; NATURAL LANGUAGE; BIM's LOQUI [20]; and ACCESS ELF).[6] Some of these systems are claimed to have been commercially successful. The use of NLDBIs, however, is much less wide-spread than it was once predicted, mainly because of the development of alternative graphical and form-based database interfaces (see, for example, the discussion of Zloof's "query by example" technique in [77]). These alternative interfaces are arguably less natural to interact with, compared to NLDBIs. It has also been argued (e.g. [27], [44]) that queries that involve quantification (e.g. "*Which company supplies every department?*"), negation (e.g. "*Which department has no secretaries?*"), or that require multiple database tables to be consulted are very difficult to formulate with graphical or form-based interfaces, while they can be expressed easily in natural language. Nevertheless, graphical and form-based interfaces have largely out-marketed NLDBIs, probably because their capabilities are often clearer to the users (see section 10.3.3 below), and they are typically easier to configure (see section 10.4 below). Experiments on the usability of NLDBIs, and comparisons with alternative database interfaces, are discussed in [18], [23], [32], [48], [69], and [82].

Perhaps as a result of their difficulties in the market, NLDBIs are no longer as fashionable a topic within academic research as they were in the eighties. There is, however, a growing body of research on integrating speech recognition, robust interpretation, and dialogue-handling techniques, with the goal being to implement systems that engage users in spoken dialogues to help them perform certain tasks (see chapter 15). We expect that this line of research will have a significant impact on future NLDBIs, giving rise to systems that will allow users to access databases via spoken dialogues, in situations where graphical and form-based interfaces are difficult to use. This could lead, for example, to NLDBIs able to answer spoken queries over the phone (see, for example, [1]; also see [87] for information on the ATIS domain, where users make flight arrangements by interacting with a computer via spoken dialogues).

10.3 Architecture and main components

This section discusses the typical architecture and the main modules of modern NLDBIs.

[5] INTELLECT is currently owned by Platinum Technology.

[6] Few of these systems appear to be currently in the market. Consult `http://users.aol.com/elfsoft/elfsoft.htm` and `http://www.englishwizard.com` for more information on two NLDBIs available at the time of writing.

10.3.1 Architectural overview

Ignoring some details, the architecture of most current NLDBIs is similar to that of figure 10.1. Roughly speaking, the part of the system that is labelled *linguistic front-end* translates the natural language input to an expression of some intermediate meaning representation language (MRL). The MRL expression is subsequently passed to the *database back-end*. This translates the MRL expression into a database language that is supported by the underlying database management system (DBMS; this is the part of the database system that is responsible for manipulating the information in the database). The resulting database language expression is then executed by the DBMS, to satisfy the user's request. In the case of questions, the execution of the database language expression retrieves information from the database, which is reported back to the user.

This architecture has several portability advantages (discussed in section 10.4). It also allows inferencing components to be added between the linguistic front-end and the database back-end to allow, for example, the NLDBI to deduce new facts from the contents of the database (we discuss this briefly in section 10.5.1). In systems where the natural language input is mapped directly to database language expressions, this inferencing would have to be carried out in the database language, which is particularly difficult because database languages are not designed to facilitate machine reasoning.

We discuss below the components of the architecture of figure 10.1 in more detail.

10.3.2 Pre-processing

The natural language input first undergoes a pre-processing phase. This tokenises the input, morphologically analyses the words, and looks them up in a lexicon to retrieve their syntactic and semantic properties. The techniques of chapters 1 – 3 apply here. (We note, however, that most NLDBIs allow the user to type only single-sentence requests. Hence, sentence segmentation is usually not an issue. Punctuation is also typically ignored.)

Proper names (e.g. person names like "*Adams*", or flight numbers like "*BA742*") constitute a particular problem for NLDBIs. For example, the personnel database of a large company may contain information about thousands of employees. To be able to parse questions that contain employee names (e.g. "*What is the salary of George Adams?*") the NLDBI must have entries in its lexicon for all these names (possibly separate entries for first names and surnames). Inserting manually an entry in the lexicon for each employee name is tedious, and it would also mean that the lexicon would have to be manually updated whenever new employees join the company.

One possible solution is to provide some mechanism that would automatically compute lexicon entries from proper names that appear in the database. (In this case, the architecture of figure 10.1 has to be modified, to allow the pre-processor to access the database. The *preprocessing rules* of figure 10.1 would also have to be removed. They are used in an alternative pre-processing method, which is discussed below.) In the personnel database case, whenever a word of the natural language input is not found in the lexicon, that word would be checked against a list of employee names held in the database. If the word is found in that list, an appropriate lexicon entry would be generated automatically using the database's information. For example, all the words that appear in the list of employee names could be assigned lexicon entries that classify the words as singular proper-name noun-phrases. If the database includes gender information, the lexicon entries could also show the gender of the

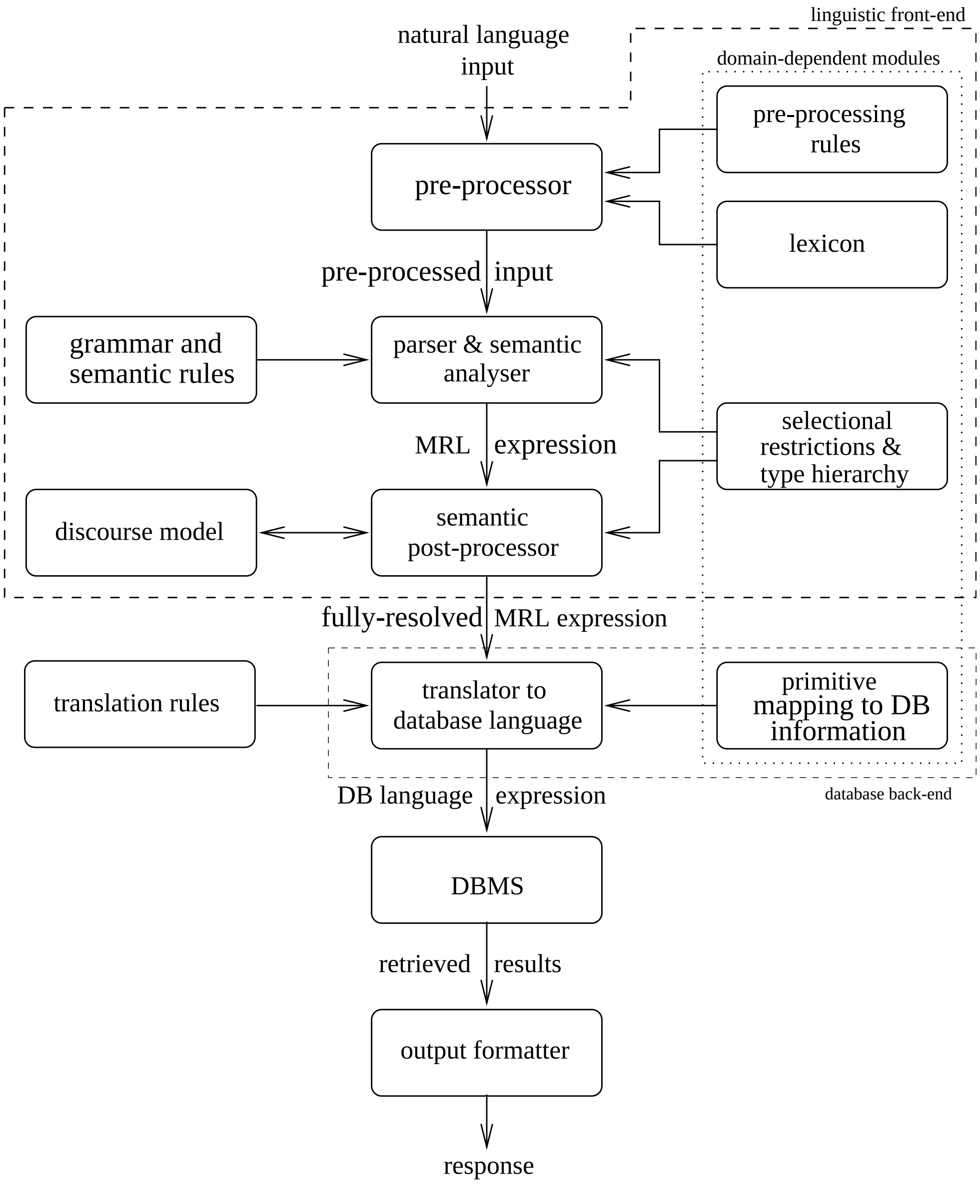

Figure 10.1: Typical NLDBI architecture

denoted employee (this is useful for anaphora resolution; see section 10.3.4).

This approach has the disadvantage that it introduces an additional database look-up during the pre-processing. In databases with a large number of proper names (e.g. databases for telephone directory enquiries) this additional look-up may be computationally expensive. This approach also runs into problems with questions that contain proper names not mentioned in the database. If, for example, the database does not contain the name "*Johnson*", the NLDBI would fail to parse "*Is Johnson in the database?*" instead of responding negatively.

An alternative approach is to employ pattern-matching pre-processing rules and typographic conventions. In questions directed to a database about flights, for example, it may be safe to treat any word that consists of two or three letters followed by three or four digits (e.g. "*BA742*") as a flight number. Any word that matches this pattern would be assigned a lexicon entry that classifies the word as singular proper-name noun-phrase. In the case of the personnel database, person names could be identified by asking the user to type all words in lower-case letters, apart from person names whose first letter is to be capitalised, and by using a suitable pattern-matching rule.

This second approach does not run into problems with proper names not mentioned in the database. Without a database look-up, however, it may be difficult to obtain some information that we wish to be included in the generated lexicon entries for proper names. As mentioned above, for example, including gender information in the entries for employee names is useful in anaphora resolution (section 10.3.4). It is generally difficult, however, to determine the genders of employees by applying pattern-matching rules on their names. It is also useful (e.g. for the mechanisms that will be discussed in section 10.3.5) to include in the proper name entries information indicating the semantic types of the named entities (e.g. whether the proper name corresponds to a flight or a spare-part). In some domains, proper names corresponding to different types of entities may be typographically very similar (e.g. there may be flight numbers like "*BA742*" and spare-part numbers like "*DX486*"). In those cases, it may be difficult to determine the types of the named entities using pattern-matching rules, and one may be forced to introduce unnatural typographical notations.

10.3.3 Parsing and semantic analysis

The pre-processed input is subsequently parsed and analysed semantically using the techniques of chapters 4 and 6. This generates an expression in a meaning representation language (MRL; typically some form of logic) which is intended to capture formally what the NLDBI "understands" to be the semantics of the natural language input. MASQUE/SQL [5, 7], for example, maps (10.1) (which is directed to a database containing geographical information) to (10.2). (MASQUE/SQL's MRL is a subset of Prolog. Terms starting with capital letters are variables.)

(10.1) What is the capital of each country bordering Greece?

(10.2)
```
answer([Capital, Country]):-
   is_country(Country),
   borders(Country, greece),
   capital_of(Country, Capital).
```

(10.2) shows that the user's question is a request to find all pairs `[Capital, Country]`, such that `Country` is a country, `Country` borders Greece, and `Capital` is the capital of `Country`.

In most NLDBIs (especially early ones), the parsing and semantic analysis are based on rather ad hoc grammars. There are, however, systems built on principled computational grammar theories (see, for example, [6], [24] and [65] for information on NLDBIs that are based on HPSG [64]). The grammars of these systems tend to be easier to comprehend and to extend. Of course, no matter how principled the grammar and how extensive the lexicon of the NLDBI are, there will always be user requests that the system fails to parse and analyse semantically in their entirety (e.g. sentences with unknown words, syntactic constructs not covered by the grammar, ungrammatical sentences). Natural language researchers are becoming increasingly interested in *robust processing* techniques, that allow systems to recover from such failures (see chapter 5). These techniques may, for example, combine the fragments of the natural language input that the system managed to process, to make reasonable guesses about the user's goals [70]. (Consult also [53] for an alternative method that can be used in NLDBIs.)

Most current NLDBIs, however, do not employ such mechanisms, generating simply an error message whenever they fail to process fully the user's input. If the failure was caused by an unknown word, the error message may report that word. In many other cases, however, the message may carry no information to help the users figure out which parts of their requests caused the failure. The situation is made worse by the fact that the linguistic coverage of most NLDBIs is not designed to be easily understandable by end-users. For example, MASQUE/SQL is able to answer "*What are the capitals of the countries bordering the Baltic and bordering Sweden?*", which leads the user to assume that the system can handle all types of conjunctions. However, the system fails to process "*What are the capitals of the countries bordering the Baltic and Sweden?*". (MASQUE/SQL can handle conjunctions only if they involve progressive verb phrases.) The result is that the users are often forced to rephrase their requests, until a phrasing that falls within the linguistic coverage of the NLDBI is found. Since the users do not have a clear view of the NLDBI's linguistic coverage, several rephrasings may be needed. This is obviously very annoying for the users, and it may be one of the main reasons for the small market-share of NLDBIs. In contrast, the capabilities of graphical and form-based interfaces are usually clear from the options that are offered on the screen, and typically any request that can be input can also be processed. We believe that it is crucial for NLDBIs to incorporate robust parsing techniques, in order to overcome this problem (we return to this issue in section 10.7). An alternative approach is to restrict severely, deliberately, and explicitly the natural language inputs that the users are allowed to enter, so that the the linguistic capabilities of the system will be clearer to the them. We discuss this approach in section 10.6.

We should note, at this point, that there are also NLDBIs that appear to be using pattern-matching techniques instead of parsing and semantic analysis. (The authors of some of those NLDBIs seem unwilling to confirm this.) To illustrate a pattern-matching question-answering approach, let us assume that the database contains the following table that holds information about countries and their capitals. (In the examples of this chapter, we assume that the database is structured according to the *relational model* [25], in which information is stored in *relations*, intuitively tables consisting of rows and columns. Most of the techniques of this chapter apply to other database models as well.)

countries	
country	*capital*
Australia	Canberra
Greece	Athens
...	...

A simplistic pattern-matching system could use a rule like the following, which would map "*What is the capital of Greece?*" to an appropriate MASQUE/SQL-like MRL expression:

> If the input string matches the following pattern:
> ... "*capital*" ... <*country_name*> ...
> then generate the MRL expression:
> `answer([Capital]):-`
> `capital_of(`<*country_name*>`, Capital).`

The main advantage of the pattern-matching approach is its simplicity. The linguistic shallowness of this method, however, very often leads to irrelevant answers. In (10.1), for example, the pattern-matching rule above would cause the NLDBI to report the capital of Greece.

10.3.4 Semantic post-processing

The MRL expressions that are generated by the semantic analysis may be initially underspecified, i.e. they may leave the semantic contribution of some linguistic mechanisms unclear. For example, they may not specify the exact entities to which anaphoric expressions refer, or they may not specify the exact scopes of logical quantifiers (see the discussion in chapter 6). There is usually one or more additional post-processing phases (figure 10.1), that resolve such underspecified elements of the logical expressions (e.g. they modify the logical expressions so that the referents of anaphoric expressions and the scopes of logical quantifiers are explicit; see [3] for a detailed discussion of how this is achieved in the CLE system).

Anaphoric expressions (e.g. "*him*", "*these*", "*the project*") are supported by several NLDBIs. They are particularly useful because they allow follow-up questions to be short. For example, in the following interaction between the user and LOQUI (from [19]), the user refers to the people who report to E. Feron as "*they*", instead of re-typing the five names.

> *Who leads* TPI*?*

`E.Feron`

> *Who reports to him?*

`C.Leonard, C.Willems, E.Bidonnet, P.Cayphas, J.P.Van Loo`

> *What do they work on?*

project	worker
DOCDIS	C.Willems
	J.P.Van Loo
	P.Cayphas
EURS	C.Leonard
	C.Willems
	E.Bidonnet

> *Which of these are leaders?*

`J.P.Van Loo`

Anaphoric expressions are resolved by consulting a discourse model (chapter 7). The discourse models of most NLDBIs are rather simplistic, as they are typically used only to resolve anaphoric expressions. They can be simply lists of entities that have been mentioned, along with information on the properties of these entities, and the locations in the discourse where the entities have been mentioned. For each anaphoric expression, the NLDBI searches this list until it finds an appropriate referent (e.g. for the pronoun "*him*", a male singular entity

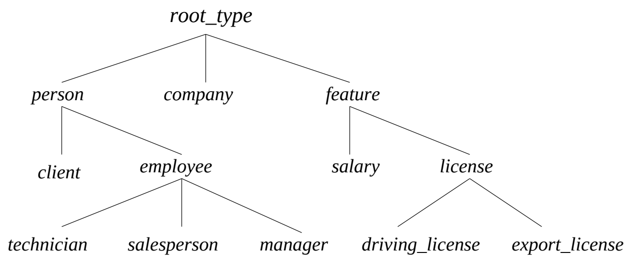

Figure 10.2: A type hierarchy of world entities

has to be found). Various heuristics can be employed in cases where an anaphoric expression has multiple possible referents in the list (e.g. the most recently mentioned possible referent may be preferred.). (See [15] for information on the anaphora resolution module of a linguistic front-end. [47] also discusses relatively simple methods for resolving pronoun anaphora that can be used in NLDBIs.)

More elaborate discourse models and reference resolution rules are needed to handle elliptical sentences, like "*How about* MIT*?*" and "*The smallest department?*" in the following interaction (from [17]) between the user and PARLANCE:

> *Does the highest paid female manager have any degrees from Harvard?*

```
Yes, 1.
```

> *How about* MIT*?*

```
No, none.
```

> *Who is the manager of the largest department?*

```
 Name       Dept.  Count
 Patterson  045    40
```

> *The smallest department?*

```
 Name      Dept.  Count
 Saavedra  011    2
```

Consult [2] and [3] for a discussion of practical ellipsis resolution techniques.

10.3.5 Type hierarchy and selectional restrictions

During the semantic analysis and the post-processing, a type hierarchy of world entities, and a set of selectional restrictions are often employed. We discuss these below.

The type hierarchy shows the various types and subtypes of world entities that are modelled in the database. (The term "entity" should be interpreted with a broad meaning. It may include, for example, salaries or ages.) Figure 10.2 shows a fragment of a possible type hierarchy that could be used when interfacing to a company's database. (Similar hierarchies are used in TEAM [36, 37], MASQUE/SQL [5, 7], and the CLE [3].) The hierarchy of figure 10.2 shows that, in the context of that particular database, a person can be either a client or an employee, that employees are divided into technicians, sales-persons, and managers, etc. (The hierarchy does not need to be a tree. There could be, for example, a common subtype of *employee* and *client* for employees that are also clients of the company.)

The selectional restrictions typically specify the types of entities that the various predicate-arguments of the MRL may denote. To illustrate the use of the type hierarchy and the selectional restrictions, let us consider the hierarchy of figure 10.2, and let us assume that the database stores the salaries of all employees, but not the salaries of clients. (For simplicity, let us assume that an employee cannot also be a client.) Then, (10.3) will retrieve no information, because the database does not store salaries of clients. We would like the NLDBI to be able to detect that the question is conceptually problematic from the database's point of view.

(10.3) What is the salary of each client?

In MASQUE/SQL, (10.3) would receive an MRL expression like (10.4):

(10.4)
```
answer([Salary, Person]):-
   is_client(Person),
   salary_of(Salary, Person).
```

The selectional restrictions could specify that: (i) the argument of `is_client` must always denote an entity of type *client*, (ii) that the first argument of `salary_of` must always denote a *salary*, and (iii) that the second argument of `salary_of` must always denote an *employee*. Then, restriction (i) and the second line of (10.4) require the variable `Person` to denote a client. This, however, leads to a violation of restriction (iii) in the third line of (10.4): restriction (iii) requires `Person` to denote an employee; this is incompatible with the requirement that `Person` must denote a client, because neither *client* nor *employee* are subtypes of each other. The system would, thus, be able to detect that the question is conceptually anomalous. It is relatively easy to generate a message like "`A client does not have a salary.`" to report this anomaly to the user. This could be achieved, for example, by associating a message template of the form "`A` X `does not have a salary.`" with constraint (iii), and by replacing X with the type-name of the second argument of `salary_of` whenever the constraint is violated.

In contrast, (10.5) would receive (10.6). Assuming that the argument of `is_technician` must denote an entity of type *technician*, (10.6) does not violate any restriction, because *technician* is a subtype of *employee*.

(10.5) What is the salary of each technician?

(10.6)
```
answer([Salary, Person]):-
   is_technician(Person),
   salary_of(Salary, Person).
```

The type hierarchy and the selectional restrictions are also useful in disambiguation. For example, from a NLDBI's point of view (10.7) is potentially ambiguous: "*with a driving licence*" may refer either to "*employees*" or to "*company*". The two readings correspond to (10.8) and (10.9) respectively.

(10.7) List all employees in a company with a driving licence.

(10.8)
```
answer([Employee]):-
   is_employee(Employee),
   is_company(Company),
   is_licence_for(driving, Licence),
   in(Employee, Company),
   has(Employee, Licence).
```

(10.9)
```
answer([Employee]):-
   is_employee(Employee),
   is_company(Company),
   is_licence_for(driving, Licence),
   in(Employee, Company),
   has(Company, Licence).
```

Of course, humans can figure out immediately that the intended reading is (10.8), because typically persons and not companies have driving licences. (In "*List all employees in a company with an export licence.*", however, the situation would be reversed.) This knowledge can be encoded using selectional restrictions. One could specify that the arguments of `is_employee` and `is_company` must denote entities of types *employee* and *company* respectively, that when the first argument of `is_licence_for` is `driving`, the second argument must denote a *driving_licence*, and that if the second argument of `has` denotes a *driving_licence*, the first argument must denote a *person*. This would rule out (10.9), where the second argument of `has` denotes a *driving_licence*, and the first one a *company*.

Anaphora resolution methods can also exploit the type hierarchy and the selectional restrictions. Knowing, for example, that only employees have salaries, limits the possible referents of "*his*" in "*What is his salary?*" to male employees, excluding any previously mentioned male clients.

Type hierarchies and selectional restrictions have been employed in many natural language processing areas. In systems that target broad knowledge domains (e.g. newspaper articles), type hierarchies and selectional restrictions are often difficult (if at all possible) to construct, because of the very large number of involved entity types, and the enormous variety of possible relations between those types (see [2] for related discussion). Databases, however, typically store information about very few types of entities, and hence type hierarchies and selectional restrictions become manageable. In fact, very often the entity types that are relevant to the database and many of their relations will have already been identified during the design of the database (e.g. in the form of "entity-relationship" diagrams [77]), and this information can be exploited when specifying the type hierarchy and the selectional restrictions of the NLDBI.

10.3.6 Paraphrasing the input

Whenever the NLDBI "understands" a natural language request to be ambiguous, several MRL expressions are generated at the end of the post-processing, each corresponding to what the NLDBI considers to be a possible reading of the input. (The input may be truly ambiguous, as in (10.10). It may also be – as it is often – the case that the NLDBI has generated more readings than the input can actually have.)

(10.10) Which disabled and female applicants speak German?

Some NLDBIs employ heuristics in the form of preference measures to determine the most likely reading (see [3] for related discussion). Another common technique is to generate an unambiguous paraphrase of each reading, asking the user to select the intended one. In (10.10), for example, an NLDBI could ask the user to select one of the following:

(10.11) Which applicants that are both disabled and female speak German?

(10.12) Which disabled and which female applicants speak German?

In the case of questions, if there are relatively few possible readings and the answers are short, the NLDBI may generate the answers to all the readings, showing which answer corresponds to which paraphrased reading, without asking the user. Paraphrasing the input is useful even when the NLDBI generates only one reading. In the following interaction (from [75]) between the user and ASK, for example, ASK repeats the user's input with the pronoun replaced by what the system assumes to be its referent. This prevents the user from being misled, in case the system has not chosen the referent the user had in mind.

> *Is there a ship whose destination is unknown?*
`yes`
> *What is it?*
`What is [the ship whose destination is unknown]?`
`Saratoga`

Repeating the user's input with anaphoric expressions replaced by their referents is relatively easy. (We can augment the list of previously mentioned entities of the discourse model to record the natural language expressions that introduce the entities; see section 10.3.4.) In the general case, however, one has to use natural language generation techniques (see chapter 8; [31] and [55] also discuss paraphrasing techniques in NLDBIs).

10.3.7 From meaning representation to database language

The MRL expressions that are generated at the end of the semantic post-processing are subsequently translated into expressions of a database language (typically SQL [60]) that is supported by the underlying database management system (DBMS). (The DBMS is the part of the database system that is responsible for manipulating the information in the database). Various methods to translate from forms of logic to and from database languages have been proposed (see, for example, [2], [5], [33] [56], [58], [67], [77], [78]). The full details of some of these proposals are highly technical, and hence beyond the scope of this chapter. Here, we only attempt to highlight some of the central ideas on which several of these proposals are based.

Let us return, for example, to the MRL expression of (10.2), repeated here as (10.13):

(10.13)
```
answer([Capital, Country]):-
   is_country(Country),
   borders(Country, greece),
   capital_of(Country, Capital).
```

Before MRL expressions like (10.13) can be translated into database language, one has to link primitive MRL expressions, like constants (e.g. `greece`) and predicate functors (e.g. `is_country`, `borders`), to database constructs. For MRL constants, one has to specify a systematic way to map them to database values (e.g. the country denoted by the MRL constant `greece` may be represented in the database as *Greece* or *GR*). In the case of predicate functors, one must link each functor which may appear in the MRL expressions to a database construct that shows for which arguments the predicates of the functor hold. (There are some cases where this linking is impossible. We discuss this in section 10.5.1.)

Assuming, for example, that the database contains the following tables, one can simply link the functor `borders` to the table *borders_info*, meaning that `borders(X, Y)` is true if and only if *borders_info* contains a row for `X` and `Y`. (We make a *closed-world assumption*, i.e. we assume that if a pair of two geographical areas is not included in *borders_info*, the two areas do not border each other.)

countries_info		
country	*capital*	*currency*
Italy	*Rome*	*Lira*
Australia	*Canberra*	*Dollar*
Greece	*Athens*	*Drachma*
...	...	...

borders_info	
borders1	*borders2*
Australia	*Pacific*
Greece	*Bulgaria*
Albania	*Greece*
Bulgaria	*Greece*
...	...

The database constructs to which functors are linked need not be stored directly in the database. They may be computed indirectly from the contents of the database. The functors `is_country` and `capital_of`, for example, could be linked to the tables *countries* and *capitals*. These are obtained by retaining only the first column or by dropping the third column of *countries_info* respectively. (In SQL, these derived tables can be defined as "views" [60].)

countries
country
Italy
Australia
Greece
...

capitals	
country	*capital*
Italy	*Rome*
Australia	*Canberra*
Greece	*Athens*
...	...

Once primitive MRL expressions have been linked to the information of the database, MRL formulae can be translated into database language using a set of translation rules. These are of two sorts: base (non-recursive) rules that translate atomic MRL formulae (e.g. `borders(Country, greece)`), and recursive rules that translate non-atomic MRL formulae (e.g. (10.13)) by recursively calling other rules to translate their subformulae.

In (10.13), a base rule could map `borders(Country, greece)` to (10.14), which generates the table of (10.15). (10.15) means that the resulting table should have the same contents as *borders_info*, except that only rows whose *borders2* value is `Greece` should be retained. Intuitively, (10.14) shows all the argument values for which `borders(Country, greece)` holds. (In this case, the second argument is a constant, and hence the second column of (10.15) contains the same value in all rows.)

(10.14)
```
SELECT *
FROM borders_info
WHERE borders2 = 'Greece'
```

(10.15)

(result of (10.14))	
borders1	*borders2*
Albania	*Greece*
Bulgaria	*Greece*
Turkey	*Greece*
...	...

The base rule that maps `borders(Country, greece)` to (10.14) could be formulated as follows: each atomic formula $\pi(\tau_1, \tau_2, \tau_3, \ldots, \tau_n)$ (where π is a predicate functor and $\tau_1, \ldots, \tau_n$ predicate-arguments) is to be mapped to the SQL expression of (10.16), where *table* is the name of the database table to which the predicate functor is linked, $\kappa_1, \ldots, \kappa_n$ are all the MRL constants among $\tau_1, \ldots, \tau_n$, $h(\kappa_1) \ldots, h(\kappa_n)$ are the database values that correspond to these constants, and ζ_1, ζ_n are the names of the columns that correspond to the predicate-argument positions where $\kappa_1, \ldots, \kappa_n$ appear. (If there are no constants among $\tau_1, \ldots, \tau_n$, the `WHERE` clause is omitted.)

(10.16)
```
SELECT *
FROM table
WHERE ζ1 = h(κ1) AND ζ2 = h(κ2) AND ζ3 = h(κ3) AND ... AND ζn = h(κn)
```

(10.16) would also map the `is_country(Country)` and `capital_of(Country, Capital)` of (10.13) to (10.17) and (10.18) respectively.

(10.17)
```
SELECT *
FROM countries
```

(10.18)
```
SELECT *
FROM capitals
```

As an example of a recursive translation rule, an MRL conjunction of the form:

$$\pi_1(\tau_1^1, \ldots, \tau_{n_1}^1),\ \pi_2(\tau_1^2, \ldots, \tau_{n_2}^2),\ \ldots,\ \pi_k(\tau_1^2, \ldots, \tau_{n_k}^2)$$

could be mapped to (10.19), where $trans(\pi_1(\tau_1^1, \ldots, \tau_{n_1}^1)), \ldots, trans(\pi_k(\tau_1^k, \ldots, \tau_{n_k}^k))$ are the SQL translations of $\pi_1(\tau_1^1, \ldots, \tau_{n_1}^1), \ldots, \pi_k(\tau_1^2, \ldots, \tau_{n_k}^2)$, and *variable_constraints* stands for an expression that requires columns corresponding to the same variable to have the same values. For example, (10.20) would be mapped to (10.21), where "((10.17))", "((10.14))", "((10.18))" stand for the expressions of (10.17), (10.14), and (10.18) respectively. (10.21) generates (10.22), that intuitively shows all the combinations of predicate-argument values in (10.20) that make (10.20) true. (`r1.country` refers to the *country* column of the table generated by (10.17), `r2.borders1` refers to the *borders*1 column of the table generated by (10.14), etc.) (Several improvements can be made to our translation rules, for example to remove identical columns from (10.22).)

(10.19)
```
SELECT *
FROM trans(π1(τ1^1,...,τn1^1)), trans(π2(τ1^2,...,τn2^2)), ..., trans(πk(τ1^k,...,τnk^k))
WHERE variable_constraints
```

(10.20)
```
is_country(Country),
borders(Country, greece),
capital_of(Country, Capital)
```

(10.21)
```
SELECT *
FROM ((10.17)) AS r1, ((10.14)) AS r2, ((10.18)) AS r3
WHERE r1.country = r2.borders1 AND r2.borders1 = r3.country
```

(10.22)

(result of (10.21))				
r1.country	*r2.borders1*	*r2.borders2*	*r3.country*	*r3.capital*
Albania	*Albania*	*Greece*	*Albania*	*Tirana*
Bulgaria	*Bulgaria*	*Greece*	*Bulgaria*	*Sofia*
Turkey	*Turkey*	*Greece*	*Turkey*	*Ankara*
...	...	...	...	...

Another recursive rule would generate the SQL expression for the overall (10.13), using (10.21) (the translation of (10.20)). In MASQUE/SQL, the resulting SQL expression would generate a table containing only the last column of (10.22).

In some systems (e.g. [56]) the MRL expressions are translated first into more theoretical database languages (e.g. relational calculus [77]). Although these languages are not supported directly by the DBMSs, they are closer to logic-based MRLs, which simplifies the formulation of the translation rules and the proof of their correctness. In these cases, there is a separate mapping from the theoretical database language to the language that is actually supported by the DBMSs (e.g. SQL). This extra mapping is often trivial (e.g. part of SQL is essentially a notational variant of the tuple relational calculus).

At the end of the translation from MRL to database language, the generated database language expression is executed by the DBMS. This retrieves the appropriate information from the database (in the case of questions), or modifies the contents of the database (in the case of requests to update the database; we discuss these in section 10.5.2).

10.3.8 Response generation

So far, we have focussed mainly on the interpretation of the natural language input. Generating an appropriate response to the user is also very important

In the case of questions, simply printing the database constructs that were retrieved by the execution of the database language expression (table rows in the case of relational databases) is not always satisfactory. For example, the retrieved database constructs may contain encoded information (e.g. department codes instead of department names). An output formatter (figure 10.1) is needed in this case, to convert the information to a more readable format. The retrieved information may also be easier to grasp if presented in graphical form (e.g. pie-charts). Commercial NLDBIs often provide such graphical output facilities.

In other cases, the NLDBI may fail to "understand" the user's request. The cause of the failure (e.g. unknown word, syntax too complex, conceptually ill-formed input) should be explained to the user. (Some methods to detect conceptually ill-formed requests were discussed in section 10.3.5.)

A more challenging task is to generate what is known as *cooperative responses*. These are needed when the user's requests contain false presuppositions, or do not express literally what the users want to know. These cases are illustrated in the following interaction with LOQUI (based on examples from [19] and [20]):

> *Does every person that works on 20 projects work on* HOSCOM*?*
`There is no such person.`
> *Does David Sedlock work on something?*
`Yes. BIM_LOQUI, MMI2, NLPAD.`

In the first question, the system has detected the false presupposition that there are people working on 20 projects, and it has generated a suitable message. In the second and third questions, the system did not generate a simple "`yes`" or "`no`"; it also reported additional information that the user probably wanted to know.

In some cases, such cooperative responses can be generated using relatively simple mechanisms. Let us consider, for example, yes/no questions that contain expressions introducing existential quantifiers (e.g. "*something*", "*a flight*"), like the second question of the dialogue above. Whenever the answer is affirmative, one strategy is to print a "`yes`", followed by the answer to the question that results when the expressions that introduce existential quantifiers are replaced with suitable interrogatives (e.g. "*what*", "*which flight*"; this is easier to achieve by operating on the MRL representation of the question). In the second question of the dialogue above, this generates a "*yes*" followed by the answer to "*On what does David Sedlock work?*".

When the answer is negative, one can generate a "*no*", followed by the answer to the question that results when a constraint of the original question is modified. In the following dialogue (based on an example from [49]), the NLDBI has replaced the constraint that the flight must be an American flight with the similar constraint that the flight must be a United flight.

> *Do American Airlines have a night flight to Dallas?*
`No, but United have one.`

One must be careful to limit the types of constraints that can be modified. In the previous dialogue, for example, the NLDBI must not be allowed to modify the constraint that the flight must be to Dallas, otherwise (as pointed out in [49]) the dialogue could have been:

> *Do American Airlines have a night flight to Dallas?*
`No, but they have one to Miami.`

Kaplan [50, 51, 52] discusses relatively simple mechanisms for generating cooperative responses, including methods to detect false presuppositions. It is not always possible, however, to generate reasonable cooperative responses using simple mechanisms. In some cases, to generate an appropriate cooperative response, the NLDBI must be able to reason about the user's intentions, as discussed in chapter 7. This requires an inferencing component. We return to this issue in section 10.5.1.

10.4 Portability

A significant part of NLDBI research has been devoted to portability issues. We discuss below three kinds of NLDBI portability. Most of the discussion relates to the architecture of figure 10.1.

10.4.1 Knowledge-domain portability

Current NLDBIs can cope only with natural language requests that refer to a particular *knowledge domain* (e.g. requests only about the employees of a company, or only about train-schedules). Before a NLDBI can be used in a new knowledge domain, it has to be *configured*. This typically involves modifying the pre-processing rules (section 10.3.2), the lexicon, the type hierarchy and selectional restrictions (section 10.3.5), and the information that links primitive MRL expressions to database constructs (section 10.3.7). One of the advantages of the architecture of figure 10.1 is that it separates clearly these *domain-dependent* modules from the rest of the system. Different assumptions about the skills of the person who will configure the NLDBI are possible:

Programmer: In some systems a (preferably small and well-defined) part of the NLDBI's code has to be rewritten during the configuration. This requires the configurer to be a programmer.

Knowledge engineer: Other systems provide tools that allow the configuration to be carried out without any programming. They still assume, however, that the configurer is familiar with knowledge representation techniques (e.g. logic), databases, and linguistic concepts. For example, MASQUE (an enhanced version of CHAT-80 from which MASQUE/SQL was developed) provides a "domain-editor" [10], that helps the configurer to "teach" the system new words and concepts that are common in the new knowledge domain. The use of the domain-editor is illustrated in the following dialogue, where the configurer teaches the system the verb "*to exceed*" (as in "*Does the population of Germany exceed the population of Portugal?*"):[7]

```
editor> add verb
what is your verb? exceed
what is its third singular present? exceeds
what is its past form? exceeded
what is its participle form? exceeding
to what set does the subject belon? numeric
is there a direct object? yes
to what set does it belong? numeric
is there an indirect object? no
is it linked to a complement? no
what is its predicate? greater_than
```

In the dialogue above, apart from teaching the system the morphology of the various forms of the verb, the configurer has also declared that the verb introduces MRL predicates of the form `greater_than(X,Y)`, where `X` and `Y` denote entities that belong to the *numeric* type of the type hierarchy (section 10.3.5).

Some NLDBIs (e.g. PARLANCE [17], the CLE [3]) provide morphology modules that allow them to infer the various forms of the new words without asking the configurer. Other systems have large built-in lexicons that cover the most common words of typical database domains, minimising the need to elicit lexical information from the user. (According to the vendor of one current commercial NLDBI, their system comes with a built-in lexicon of 16,000 words; see also the discussion in chapter 2.) In some cases, it is also possible to construct automatically lexicon entries from the contents of the database (see, for example, the discussion on proper-names in section 10.3.2).

Database administrator: In practice, NLDBIs may often be configured by the administrators of existing databases. In that case, it is reasonable to assume that the configurers will be familiar with database concepts and the targetted database, but not with MRLs or linguistics. This is the approach adopted by the designers of TEAM [36, 37]. In the following dialogue (based on an example from [36]), TEAM collects information about the database table ("`file`") `chip`. Notice that TEAM uses database terminology (e.g. "primary key"), but not terms from linguistics or logic (e.g. "participle" and "predicate" in the MASQUE dialogue above).

```
file name:  chip
fields: id maker width
```

[7] MASQUE's domain-editor does not actually shield completely the knowledge-engineer from the programming language. The knowledge engineer is still required to write some Prolog to relate MRL expressions to the database.

```
subject: processor
synonyms for processor: chip
primary key: id
Can one say ''Who are the processors?': no
Pronouns for subject (he, she, it,they): it
field:  MAKER
type of field (symbolic, arithmetic, feature): symbolic
...
```

End-user: Other NLDBI designers emphasise that the configuration can never be complete, and that the end-users should be allowed to teach the system new words and concepts using natural language. This approach is illustrated in the following interaction with PARLANCE [17], where the user defines the terms "*yuppie*" and "*rich employee*":

> *define: a yuppie is a person under 30 with a graduate degree who earns over 5K per month*
> *define: a rich employee is an employee who earns over 100K per year*

Although large built-in lexicons and configuration tools have made NLDBIs much easier to port to new knowledge domains, configuring a NLDBI for a new knowledge domain is often still a complicated task, and it may still require at least some familiarity with basic natural language processing concepts. Alternative graphical and form-based database interfaces are usually easier to configure. As we noted in section 10.2, this is probably one of the main reasons for the small market-share of NLDBIs.

10.4.2 DBMS portability

NLDBIs that adopt widely-supported database languages (e.g. SQL [60]) can often be used with different DBMSs that support the same database language with only minor modifications. If the original database language is not supported by the new DBMS, the part of the NLDBI that is responsible for translating from MRL to database language (the *database back-end* of figure 10.1) has to be rewritten. In that case, one of the advantages of the architecture of figure 10.1 is that no modifications are needed in the linguistic front-end, provided that the knowledge domain remains the same.

The architecture of figure 10.1 also leads to the development of generic linguistic front-ends, that apart from NLDBIs can be used as components of other natural language processing systems. (An example of such a system is the CLE [3], that has been used as a component of both a NLDBI and a machine translation system.)

10.4.3 Natural language portability

Almost all of the existing NLDBIs require the user's requests to be formulated in English. Modifying an existing NLDBI to be used with a different natural language can be a formidable task, because the NLDBI may contain deeply embedded assumptions about the targetted natural language. [54] provides some information about an attempt to build a Portuguese version of CHAT-80 [80]. Information about Swedish, French, and Spanish versions of the CLE can be found in [3] and [66].

10.5 Advanced issues

Having introduced the basic components of the typical NLDBI architecture, we now move on to some more advanced issues, namely the "doctor-on-board" problem, database updates, meta-knowledge, modal, and temporal questions.

10.5.1 The "doctor-on-board" problem

The discussion of section 10.3.7 assumed that each predicate functor can be linked to a database construct (a table in the relational model) that shows the arguments for which the predicates of the functor hold. That database construct, it was assumed, is either stored directly in the database, or can be computed from the contents of the database. There are some cases, however, where this assumption does not hold. The "doctor-on-board" problem [63] is a well-known example of such a case.

Let us imagine a database that contains only the following table, which holds information about the ships of a fleet. The third column shows if a doctor is on board the ship ("*y*") or not ("*n*").

ships_info		
ship	*crew*	*doctor*
Vincent	420	*y*
Invincible	514	*y*
Sparrow	14	*n*
...	...	...

Let us also consider (10.23), which would receive the MASQUE/SQL-like MRL expression of (10.24). (10.24) shows that the answer should be affirmative iff there is a doctor `D` that is on board Vincent.

(10.23) Is there a doctor on the Vincent?

(10.24)
```
answer([]):-
    is_doctor(D),
    on_board(D, vincent).
```

To apply the MRL to database language translation method of section 10.3.7, one needs to link `is_doctor` to a database table that shows the entities that are doctors. This is impossible, however, because the database (which contains only *ships_info*) does not list the doctors. Similarly, `on_board` has to be linked to a table that shows which entities are on board which ships. Again, this table cannot be computed from the information in the database.

What is interesting is that (10.24) is equivalent to (10.25), where `doctor_on_board(D,S)` is intended to be true iff `is_doctor(D)` and `on_board(D,S)` are both true. Unlike (10.24), (10.25) poses no problem for the translation method of section 10.3.7, because *doctor_on_board* can be linked to a version of *ships_info* without the *crew* column.

(10.25)
```
answer([]):-
    doctor_on_board(D, vincent).
```

The problem is that (10.23) cannot be mapped directly to (10.25). The MRL expression (10.24) is typical of what most natural language front-ends would generate, with predicates which are introduced by linguistic constituents (e.g. `is_doctor(X)` is introduced by the noun phrase "*a doctor*", and `on_board(X, vincent)` by the prepositional phrase "*on Vincent*"). What is needed is a mechanism to convert the MRL expressions that are generated by the linguistic front-end to equivalent MRL expressions that contain only predicate functors which can be linked to appropriate database tables. [67] proposes a mechanism which can, among other things, perform this conversion. Roughly speaking, the conversion is carried out by an inferencing component that employs rules like (10.26).

(10.26) $(\texttt{doctor}(\xi_1),\ \texttt{on\_board}(\xi_1,\ \xi_2)) \equiv \texttt{doctor\_on\_board}(\xi_2)$

(10.26) allows any MRL expression of the form $\texttt{doctor}(\xi_1),\ \texttt{on\_board}(\xi_1,\ \xi_2)$ (where ξ_1 and ξ_2 stand for MRL terms) to be replaced by $\texttt{doctor\_on\_board}(\xi_2)$. (10.26) would license the conversion from (10.24) to (10.25). (10.25) would then be translated into database language as discussed in section 10.3.7.

Inferencing components are also useful in cases where the NLDBI has to reason about the user's intentions to generate cooperative responses (section 10.3.8). They can also be employed to allow the NLDBI to deduce new facts from the information in the database (e.g. with appropriate reasoning rules, a NLDBI connected to a hospital's database could answer "*Which patients need attention overnight?*" even if the database does not flag explicitly patients who need overnight attention). In some NLDBIs (e.g. INTELLECT), the inferencing component can be a full expert system, which acts as an intermediate layer between the linguistic front-end and the database back-end.

[4] provides a detailed description of CLARE, a NLDBI with reasoning capabilities. CLARE is based on the CLE [3], and includes the inferential mechanism of [67].

10.5.2 Database updates

Apart from answering questions, some NLDBIs also allow the user to *update* the information in the database. This is illustrated in the following dialogue with ASK [75] [76]:

\> *What is the home port of the Tokyo Maru?*
`Yokohama`
\> *Home port of the Tokyo Maru is Hong Kong.*
`Yokohama has been replaced by Hong Kong as home port of Tokyo Maru.`

ASK knows that a ship may have only one home port, and hence in the second request it replaced Yokohama by Tokyo Maru. In contrast, it knows that a ship may carry many types of cargo. Hence, the input "*The Tokyo Maru carries coal.*" would lead ASK to add coal to the previously known cargoes of the Tokyo Maru, instead of replacing the previous information.

Natural language database updates can be difficult to process, because they may lead to unanticipated side-effects. These side-effects are due to database constraints that the user is not aware of. This is illustrated in the following dialogue (the example is borrowed from [30]).

\> *List the employees and their managers.*

employee	manager
Adams	Fisher
White	Baker
Brown	Jones
Smith	Jones

\> *Change Brown's manager from Jones to Baker.*

`Done.`

> *List the employees and their managers.*

employee	manager
Adams	Fisher
White	Baker
Brown	Baker
Smith	Baker

Notice that Smith's manager also changed from Jones to Baker, although this was not requested. This happens because in the database (which contains the two tables below) employees are linked to managers indirectly, through their departments.

employees_table		
employee	*salary*	*department*
Adams	3000	*inventory*
White	3500	*marketing*
Brown	2500	*sales*
Smith	2500	*sales*

departments_table	
department	*manager*
sales	*Jones*
marketing	*Baker*
inventory	*Fisher*

To satisfy the user's request to change the manager of Brown, the system has changed the sales manager from Jones to Baker. This caused all other employees in the sales department (e.g. Smith) to receive Baker as their new manager. Users not aware of the database structure would find this behaviour hard to explain. (NLDBI users are usually not informed about the structure of the database. Not having to be aware of the database structure is supposed to be an advantage of NLDBIs over graphical or form-based interfaces.)

The indirect link between employees and managers actually makes the update request in the dialogue above ambiguous: instead of changing the sales manager from Jones to Baker, the NLDBI could have moved Smith from the sales department (which is managed by Jones) to the marketing department (which is managed by Baker). PIQUET [30] maintains a model of the user's conceptual view of the database. Whenever a user input reflects awareness or presupposition of a database object, link, or restriction, the model is modified accordingly. If the user enters an ambiguous request (in the above sense), the NLDBI consults the model to select the reading that has the fewest side-effects with respect to the user's current view of the database. In the dialogue above, after "*List the employees and their managers.*" has been answered, the user's view of the database corresponds to the table:

employee	*manager*
Adams	*Fisher*
White	*Baker*
Brown	*Jones*
Smith	*Jones*

If "*Change Brown's manager from Jones to Baker.*" is interpreted as a request to change the sales manager from Jones to Baker, the answer causes the user's view to become as shown below on the left. A side-effect (shown in bold) occurs. If, in contrast, the sentence is interpreted as a request to move Brown from the sales department to the marketing department, the user's view becomes as shown below on the right.

employee	*manager*
Adams	*Fisher*
White	*Baker*
Brown	*Baker*
Smith	**Baker**

employee	*manager*
Adams	*Fisher*
White	*Baker*
Brown	*Baker*
Smith	*Jones*

The second interpretation introduces no side-effects to the user's view, and would have been preferred by PIQUET.

10.5.3 Meta-knowledge, modal, and temporal questions

This section discusses briefly some types of questions that present particular interest.

Meta-knowledge questions: Apart from questions about the entities that are represented in the database, the user may want to submit queries about the conceptual organisation of the database's knowledge (e.g. "*What information does the database contain?*", "*What are the properties of employees?*"). These can be considered *meta-knowledge questions*, as they refer to knowledge about the database's knowledge. The following example (from [75]) shows how ASK reacts to a question of this kind.

> *What is known about ships?*

```
 Some are in the following classes:  navy, freighter, tanker
 All have the following attributes:  destination, home port
 ...
```

Most NLDBIs do not support meta-knowledge questions.

Modal questions: Database systems usually enforce a set of *integrity constraints* to guard against contradicting or incorrect information (e.g. an employee cannot have two dates of birth, and in practice, the age of an employee is never greater than 80), and to enforce corporate policies (e.g. that employees cannot earn more than their managers, or that all employees must be over 20). [57] presents a method that exploits these integrity constraints to answer *modal questions*, questions that ask if something may or must be true (e.g. "*Can an employee be 18 years old?*", "*Is it the case that J.Adams must earn more than T.Smith?*").

Temporal questions: Most NLDBIs do not provide adequate support for temporal linguistic mechanisms. For example, users are often allowed to use very few (if any) verb tenses, temporal adverbials (e.g. "*for two hours*", "*before 5:00pm*"), or temporal subordinate clauses (e.g. "*while J.Adams was personnel manager*"). [6, 9] discuss how some of these mechanisms can be supported when constructing NLDBIs for temporal databases (databases designed to handle time-dependent data).

10.6 Restricted input systems

We noted in section 10.3.3 that one of the main problems of NLDBIs is that their linguistic coverage is usually not obvious to the user. This often has the annoying consequence that users are forced to rephrase their requests several times, until a phrasing that falls within the linguistic coverage of the system is found. To overcome this problem, most NLDBI developers attempt to expand the linguistic coverage of their systems, hoping that eventually it will be possible to process successfully most of the users' inputs. An alternative approach, which we discuss in this section, is to restrict drastically the linguistic coverage of the NLDBI, and to do this in a manner that allows the users to obtain a clearer view of the system's linguistic capabilities.

10.6.1 Controlled languages

One way to make the linguistic capabilities of NLDBIs clearer is to support only a natural language fragment whose syntax is made explicitly known to the user. (The term *controlled language* is often used to refer to such fragments.) The fragment must be selected carefully, to make its syntax easy to explain. A well-chosen fragment may also allow a direct mapping from natural language to database language.

In PRE [34], for example, the user was allowed to input only questions of the following pattern:

what is/are
conjoined noun phrases
  nested relative clauses
    conjoined relative clauses

The user could, for example, enter the following question (simplified example from [34]). Line (1) contains the conjoined noun phrases. Lines (2) – (4) contain the nested relative clauses (they are nested, because (3) refers to "*schedules*" in (2), and (4) refers to "*appointments*" in (3)). Lines (5) – (7) contain the conjoined relative clauses (these are not nested; they all refer to the "*orders*" of (4)).

what is/are	
the names, ids, and categories of the employees	(1)
who are assigned schedules	(2)
that include appointments	(3)
that are executions of orders	(4)
whose addresses contain 'maple' and	(5)
whose dates are later than 12/15/83 and	(6)
whose statuses are other than 'comp'	(7)

In contrast, the following question was not accepted by PRE. In this case, line (2) contains the only clause of the nested relative clauses part. Both (3) and (4) are conjoined relative clauses referring to the "*schedules*" of (2). However, (5) is nested with respect to (4), because it refers to the "*orders*" of (4), not the "*schedules*" of (2). PRE's question pattern does not allow conjoined relative clauses to be followed by other nested relative clauses.

what are	
the addresses of the appointments	(1)
that are included in schedules	(2)
whose call times are before 11:30 and	(3)
that are executions of orders	(4)
whose statuses are other than 'comp'	(5)

Epstein claims that PRE's question pattern is easy to remember, though this is probably arguable.[8] PRE's question pattern is also chosen to simplify the retrieval of information from the database. PRE adopts an entity-relationship-like database model [77]. Figure 10.3, for

[8] APPEAL by Winformation Software adopts a similar controlled language approach, but with a more restricted and easier to remember syntax; see `http://wwwiz.com/home/mather`.

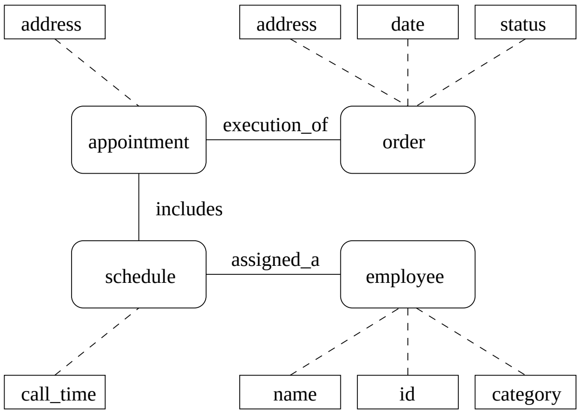

Figure 10.3: The structure of a PRE-like database.

example, shows the structure of the database to which the previous two examples refer.[9] Oval boxes correspond to entity types, rectangular boxes correspond to attributes of entities, and continuous lines correspond to relations between entities.

The conjoined noun phrases of PRE's question pattern correspond to a projection operator [77]. Projection operators specify which attributes (e.g. *name*, *address*) of an entity are to be reported. The nested relative clauses correspond to traversals of continuous line links (relationships between entities). Finally, the conjoined relative clauses correspond to select operators. These pick individual entities of particular types according to certain criteria.

For example, to answer the first question of this section, PRE first transforms the conjoined relative clauses to a select operator. This selects all the *order* entities that contain '*maple*' in their *address*es, and that have *date*s later than 12/15/83, and *status*es other than '*comp*'. Next, PRE transforms the nested relative clauses to a sequence of relationship (continuous line) traversals. In our example, this connects the *order*s that were selected in the previous step to *appointment* via the *execution_of* relationship, *appointment*s to *schedule* via the *includes* relationship, and *schedule*s to *employee*s via the *assigned_a* relationship. Finally, PRE transforms the conjoined noun phrases to a projection operator. For each *employee* entity reached during the previous step, the system reports its *name*, *id*, and *category*.

The main disadvantage of NLDBIs with controlled languages is that the user has to be taught the syntax of the supported fragment. In applications where the users need to be able to submit complex queries, it may not be easy to define a natural language fragment which is both rich enough to allow these queries and easy to understand.

10.6.2 Menu-based systems

Another approach is to require the users to form their requests by choosing words or phrases from menus displayed on the screen. Figure 10.5, for example, shows the initial screen of NLMENU [73, 74] (the example is from [74]). The highlighted border of the *commands* menu indicates that the first word of the user's request has to be chosen from that menu. (Symantec's Q&A offered a similar menu-based mode of interaction.)

[9][34] does not provide much information about PRE's database. Figure 10.3 may not reflect the exact form of PRE's database model.

commands	***nouns***	***modifiers***
Find Delete	suppliers	whose part city is
Insert	parts	whose color is
attributes	shipments	whose part name is
weight	<specific suppliers>	whose part# is
quantity	<specific parts>	whose supplier city is
city	<specific shipments>	whose supplier name is
color	***comparisons***	whose supplier supplier# is
name	between	whose shipment part# is
part#	greater than	whose shipment supplier# is
supplier#	less than	whose supplier status is
status	greater than or equal to	whose part weight is
	less than or equal to	whose shipment quantity is
	equal to	which are shipments of
	connectors	which were shipped by
	the number of of	who ship
	the average and	who supply
	the total or	which are supplied by
>		

Figure 10.4: The initial NLMENU screen.

NLMENU used a context-free semantic grammar (section 10.2). Whenever a new word or phrase was selected, NLMENU used the grammar to determine which words or phrases could be attached to the string that the user had assembled up to that point, to form a request that the system could process. These words and phrases were then shown in the menus. Figure 10.5, for example, shows the NLMENU screen after entering "*Find color and name of parts*". Notice how the *modifiers* menu has changed to contain only completions that the system can handle.

The main advantage of menu-based NLDBIs is that only requests that the systems can handle can be input, and the users can obtain an understanding of the systems' capabilities by browsing the menus. This approach works well in domains where small dictionaries and short questions are adequate. With large dictionaries and longer questions, however, the menus proliferate, and become lengthy and difficult to use.

10.7 Conclusions

This chapter has attempted to serve two purposes: to introduce the reader to the field of NLDBIs by describing some of the central issues, and to highlight the state of the art in NLDBIs by outlining the facilities, methods, and problems of typical implemented systems. In the light of the discussion in the previous sections, the following observations seem valid to us.

The area of NLDBIs is mature enough, to the extend that usable systems can be constructed for real-world applications, and indeed several commercial NLDBIs have appeared.

Although usable NLDBIs can and have been constructed, NLDBIs currently hold a rather small market-share, mainly because of competition from graphical and form-based interfaces. Despite their limitations, these alternative interfaces are often more appealing, because their capabilities are usually clearer to the users, and they are easier to configure. To improve the

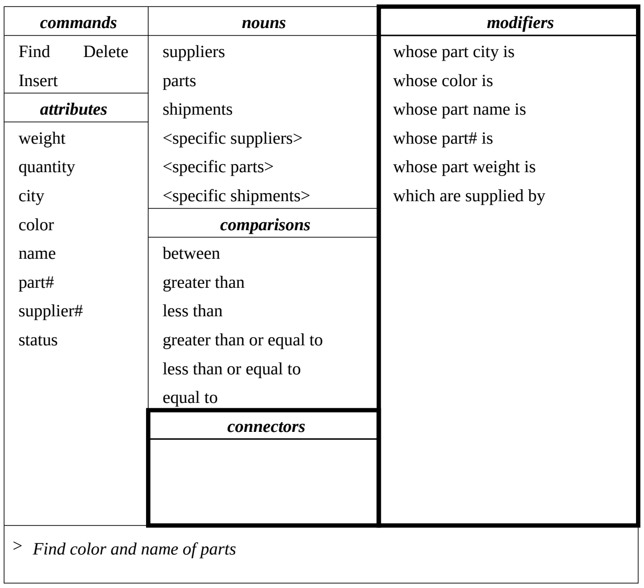

Figure 10.5: NLMENU's screen at a later stage.

position of their systems in the market, NLDBI developers need to address these two issues, and to explore applications where the alternatives to NLDBIs are difficult to use.

One example of such an application might involve remote access to databases by telephone. There is a growing body of research (e.g. [1]) integrating speech recognition, robust interpretation, and dialogue-handling techniques, with the goal of implementing systems that engage users in spoken dialogues to assist with certain tasks (see chapter 15). This line of research is likely to have a significant impact on future NLDBIs, giving rise to systems that will allow users to access databases via speech, in situations where graphical and form-based interfaces are difficult to use (cf. the ATIS domain [87]).

Regarding the obscurity of the capabilities of NLDBIs, we believe that this problem can be overcome by incorporating robust language interpretation methods and dialogue-handling techniques (chapters 5 and 15). These will allow NLDBIs to make reasonable guesses about the content of any user inputs which cannot be "understood" in their entirety. Also, such systems can engage in reasonably cooperative dialogues that will both help the users to realise the capabilities of the NLDBI and also assist the system in determining the task to be performed.

The configuration difficulties of NLDBIs are more of an obstacle. Portability has been a primary research focus of NLDBI research for almost two decades. Although significant improvements have been made, NLDBIs still cannot compete with graphical or form-based interfaces in terms of ease of configuration. The situation may become worse if NLDBIs adopt robust interpretation and dialogue-handling techniques as suggested above, since many of these techniques rely heavily on domain-dependent information. There could be a possible niche for NLDBIs, nevertheless, in accessing a small number of very large, centrally held, widely accessed databases (e.g. telephone directories). The configuration of the NLDBIs for these databases could be assigned to NLDBI experts, with the understanding that it will be a lengthy and resource-consuming effort, the cost of which will be balanced by the subsequent wide use of the NLDBIs.

Bibliography

[1] D. Albesano, P. Baggia, M. Danieli, R. Gemello, E. Gerbino, and C. Rullent. DIALOGOS: A Robust System for Human-Machine Spoken Dialogue on the Telephone. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Munich, Germany*, 1997.

[2] J.F. Allen. *Natural Language Understanding.* Benjamin/Cummings, Menlo Park, USA, 1995.

[3] H. Alshawi, editor. *The Core Language Engine*, Cambridge, Massachusetts, 1992. MIT Press.

[4] H. Alshawi, D. Carter, R. Crouch, S. Pulman, M. Rayner, and A. Smith. CLARE – A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine. Final report, SRI International, Cambridge, UK, 1992.

[5] I. Androutsopoulos. Interfacing a Natural Language Front-End to a Relational Database. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[6] I. Androutsopoulos. *A Principled Framework for Constructing Natural Language Interfaces to Temporal Databases.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1996.

[7] I. Androutsopoulos, G. Ritchie, and P. Thanisch. An Efficient and Portable Natural Language Query Interface for Relational Databases. In P.W. Chung, G. Lovegrove, and M. Ali, editors, *Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Edinburgh*, pages 327–330, Langhorne, PA, U.S.A., 1993. Gordon and Breach Publishers Inc.

[8] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural Language Interfaces to Databases – An Introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[9] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Time, Tense and Aspect in Natural Language Database Interfaces. *Natural Language Engineering*, forthcoming.

[10] P. Auxerre and R. Inder. MASQUE Modular Answering System for Queries in English – User's Manual. Technical Report AIAI/SR/10, Artificial Intelligence Applications Institute, University of Edinburgh, 1986.

[11] B. Ballard and M. Jones. Computational Linguistics. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 203 – 224. John Wiley and Sons, New York, 1992.

[12] B. Ballard and D. Stumberger. Semantic Acquisition in TELI. In *Proceedings of the 24th Annual Meeting of ACL, New York*, pages 20–29, 1986.

[13] B.W. Ballard. The Syntax and Semantics of User-Defined Modifiers in a Transportable Natural Language Processor. In *Proceedings of the 22nd Annual Meeting of ACL, Stanford, California*, pages 52–56, 1984.

[14] B.W. Ballard, J.C. Lusth, and N.L. Tinkham. LDC-1: A Transportable, Knowledge-based Natural Language Processor for Office Environments. *ACM Transactions on Office Information Systems*, 2(1):1–25, January 1984.

[15] F.A. Barros and A. De Roeck. Resolving Anaphora in a Portable Natural Language Front End to a Database. In *Proceedings of the 4th Conference on Applied Natural Language Processing, Stuttgart, Germany*, pages 119–124, 1994.

[16] M. Bates, M.G. Moser, and D. Stallard. The IRUS transportable natural language database interface. In L. Kerschberg, editor, *Expert Database Systems*, pages 617–630. Benjamin/Cummings, Menlo Park, CA., 1986.

[17] BBN Systems and Technologies. *BBN Parlance Interface Software – System Overview*, 1989.

[18] J.E. Bell and L.A. Rowe. An Exploratory Study of Ad Hoc Query Languages to Databases. In *Proceedings of the 8th International Conference on Data Engineering, Tempe, Arizona*, pages 606–613. IEEE Computer Society Press, February 1992.

[19] BIM Information Technology. *Loqui: An Open Natural Query System – General Description*, 1991.

[20] J.-L. Binot, L. Debille, D. Sedlock, and B. Vandecapelle. Natural Language Interfaces: A New Philosophy. *SunExpert Magazine*, pages 67–73, January 1991.

[21] R.J. Bobrow. The RUS System. In *Research in Natural Language Understanding, BBN Report 3878*. Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1978.

[22] R.J. Bobrow, P. Resnik, and R.M. Weischedel. Multiple Underlying Systems: Translating User Requests into Programs to Produce Answers. In *Proceedings of the 28th Annual Meeting of ACL, Pittsburgh, Pennsylvania*, pages 227–234, 1990.

[23] R.A. Capindale and R.G. Crawford. Using a Natural Language Interface with Casual Users. *International Journal of Man-Machine Studies*, 32:341–361, 1990.

[24] N. Cercone, P. McFetridge, F. Popowich, D. Fass, C. Groeneboer, and G. Hall. The SystemX Natural Language Interface: Design, Implementation, and Evaluation. Technical Report CSS-IS TR 93-03, Centre for Systems Science, Simon Fraser University, Burnaby, BC, Canada, 1993.

[25] E.F. Codd. A Relational Model for Large Shared Data Banks. *Communications of ACM*, 13(6):377–387, 1970.

[26] E.F. Codd. Seven Steps to RENDEZVOUS with the Casual User. In J. Kimbie and K. Koffeman, editors, *Data Base Management*, pages 179 – 200. North-Holland Publishers, 1974.

[27] P.R. Cohen. The Role of Natural Language in a Multimodal Interface. Technical Note 514, Computer Dialogue Laboratory, SRI International, 1991.

[28] A. Copestake and K. Sparck Jones. Natural Language Interfaces to Databases. *The Knowledge Engineering Review*, 5(4):225–249, 1990.

[29] F. Damerau. Operating Statistics for the Transformational Question Answering System. *American Journal of Computational Linguistics*, 7:30–42, 1981.

[30] J. Davidson and S.J. Kaplan. Natural Language Access to Data Bases: Interpreting Update Requests. *Computational Linguistics*, 9(2):57–68, 1983.

[31] A.N. De Roeck and B.G.T. Lowden. Generating English Paraphrases from Formal Relational Calculus Expressions. In *Proceedings of the 11th International Conference on Computational Linguistics, Bonn, Germany*, pages 581–583, 1986.

[32] S.M. Dekleva. Is Natural Language Querying Practical? *Data Base*, pages 24–36, May 1994.

[33] C. Draxler. *Accessing Relational and Higher Databases through Database Set Predicates in Logic Programming Languages.* PhD thesis, University of Zurich, 1992.

[34] S.S. Epstein. Transportable Natural Language Processing Through Simplicity – The PRE System. *ACM Transactions on Office Information Systems*, 3(2):107–120, 1985.

[35] B. Grosz, K. Sparck Jones, and B. Webber. *Readings in Natural Language Processing.* Morgan Kaufmann, Los Altos, USA, 1986.

[36] B.J. Grosz. TEAM: A Transportable Natural-Language Interface System. In *Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California*, pages 39–45, 1983.

[37] B.J. Grosz, D.E. Appelt, P.A. Martin, and F.C.N. Pereira. TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *Artificial Intelligence*, 32:173–243, 1987.

[38] C.D. Hafner. Interaction of Knowledge Sources in a Portable Natural Language Interface. In *Proceedings of the 22nd Annual Meeting of ACL, Stanford, California*, pages 57–60, 1984.

[39] C.D Hafner and K. Godden. Portability of Syntax and Semantics in Datalog. *ACM Transactions on Office Information Systems*, 3(2):141–164, April 1985.

[40] L.R. Harris. User-oriented Data Base Query with the ROBOT Natural Language Query System. *International Journal of Man-Machine Studies*, 9:697–713, 1977.

[41] L.R. Harris. The ROBOT System: Natural Language Processing Applied to Data Base Query. In *Proceedings of the ACM'78 Annual Conference*, pages 165–172, 1978.

[42] L.R. Harris. Experience with ROBOT in 12 Commercial Natural Language Data Base Query Applications. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence, Tokyo, Japan*, pages 365–368, 1979.

[43] L.R. Harris. Experience with INTELLECT: Artificial Intelligence Technology Transfer. *The AI Magazine*, 5(2):43–50, 1984.

[44] L.R. Harris. Proliferating the Data Warehouse Beyond the Power User. White paper, Linguistic Technology Corporation, 1996.

[45] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a Natural Language Interface to Complex Data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978. Reprinted in [35], pages 563–584.

[46] E.W. Hinrichs. Tense, Quantifiers, and Contexts. *Computational Linguistics*, 14(2):3–14, 1988.

[47] C.H. Hwang and K. Schubert. Resolving Pronoun References. *Lingua*, 44:311–338, 1978. Reprinted in [35], pages 339–352.

[48] M. Jarke, J.A. Turner, E.A. Stohr, Y. Vassiliou, N.H. White, and K. Michielsen. A Field Evaluation of Natural Language for Data Retrieval. *IEEE Transactions on Software Engineering*, SE-11(1):97–113, 1985.

[49] T. Johnson. *Natural Language Computing: The Commercial Applications*. Ovum Ltd., London, 1985.

[50] S.J Kaplan. Indirect Responses to Loaded Questions. *Theoretical Issues in Natural Languae Processing*, 2:202–209, 1978.

[51] S.J. Kaplan. Cooperative Responses from a Portable Natural Language Data Base Query System. *Artificial Intelligence*, 19:165–187, 1982.

[52] S.J. Kaplan. Cooperative Responses from a Portable Natural Language Database Query System. In M. Brady and R.C. Berwick, editors, *Computational Models of Discourse*, chapter 3, pages 167–208. MIT Press, Cambridge, Massachusetts, 1983.

[53] M.C. Linebarger, L.M. Norton, and Dahl D.A. A Portable Approach to Last Resort Parsing and Interpretation. In *Human Language Technology – Proceedings of the ARPA Workshop, Princeton, NJ*, pages 31–36. Morgan Kaufmann, 1993.

[54] G.P. Lopes. Transforming English Interfaces to Other Languages: An Experiment with Portuguese. In *Proceedings of the 22nd Annual Meeting of ACL, Stanford, California*, pages 8–10, 1984.

[55] B.G.T. Lowden and A.N. De Roeck. REMIT: A Natural Language Paraphraser for Relational Query Expressions. *ICL Technical Journal*, 5(1):32–45, 1986.

[56] B.G.T. Lowden, B.R. Walls, A.N. De Roeck, C.J. Fox, and R. Turner. A Formal Approach to Translating English into SQL. In Jackson and Robinson, editors, *Proceedings of the 9th British National Conference on Databases*, 1991.

[57] B.G.T. Lowden, B.R. Walls, A.N. De Roeck, C.J. Fox, and R. Turner. Modal Reasoning in Relational Systems. Technical Report CSM-163, Dept. of Computer Science, University of Essex, 1991.

[58] R. Lucas. *Database Applications Using Prolog.* Halsted Press, 1988.

[59] P. Martin, D. Appelt, and F. Pereira. Transportability and Generality in a Natural-Language Interface System. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, Germany*, pages 573–581. Morgan Kaufmann, 1983. Reprinted in [35], pages 585–593.

[60] J. Melton and A.R. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[61] J. Mylopoulos, A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, and H. Wong. TORUS: A Step Towards Bridging the Gap Between Data Bases and the Casual User. *Information Systems*, 2:49–64, 1976.

[62] N. Ott. Aspects of the Automatic Generation of SQL Statements in a Natural Language Query Interface. *Information Systems*, 17(2):147–159, 1992.

[63] C.R. Perrault and B.J. Grosz. Natural Language Interfaces. In H.E. Shrobe, editor, *Exploring Artificial Intelligence*, pages 133–172. Morgan Kaufmann Publishers Inc., San Mateo, California, 1988.

[64] C. Pollard and I.A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Stanford, 1994.

[65] F. Popowich, P. McFetridge, D.C. Fass, and G. Hall. Processing Complex Noun Phrases in a Natural Language Interface to a Statistical Database. In *Proceedings of the 15th International Conference on Computational Linguistics, Nantes, France*, volume 1, pages 47–52, 1992.

[66] M. Rayner, D. Carter, and P. Bouillon. Adapting the Core Language Engine to French and Spanish. In *Proceedings of the Conference on Natural Language Processing and Industrial Applications, Moncton, New Brunswick, Canada*, 1996.

[67] Manny Rayner. *Abductive Equivalential Translation and its Application to Natural Language Database Interfacing.* PhD thesis, Royal Institute of Technology, Stockholm, 1993.

[68] R.J.H. Scha. Philips Question Answering System PHILIQA1. In *SIGART Newsletter, no.61.* ACM, New York, 1977.

[69] D.W. Small and L.J. Weldon. An Experimental Comparison of Natural and Structured Query Languages. *Human Factors*, 25(3):253–263, 1983.

[70] D. Stallard and R. Bobrow. The Semantic Linker – A New Fragment Combining Method. In *Human Language Technology – Proceedings of the ARPA Workshop, Princeton, NJ*, pages 37–42. Morgan Kaufmann, 1993.

[71] M. Templeton. EUFID: A Friendly and Flexible Frontend for Data Management Systems. In *Proceedings of the 17th Annual Meeting of ACL*, pages 91–93, 1979.

[72] M. Templeton and J. Burger. Problems in Natural Language Interface to DBMS with Examples from EUFID. In *Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California*, pages 3–16, 1983.

[73] H.R. Tennant, K.M. Ross, M. Saenz, C.W. Thompson, and J.R. Miller. Menu-Based Natural Language Understanding. In *Proceedings of the 21st Annual Meeting of ACL, Cambridge, Massachusetts*, pages 151–158, 1983.

[74] R. Tennant, K.M. Ross, and Thompson C.W. Usable Natural Language Interfaces through Menu-Based Natural Language Understanding. In *Proceedings of CHI'83, Conference on Human Factors in Computer Systems, Boston*. ACM, 1983.

[75] B.H. Thompson and F.B. Thompson. Introducing ASK, A Simple Knowledgeable System. In *Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California*, pages 17–24, 1983.

[76] B.H. Thompson and F.B. Thompson. ASK is Transportable in Half a Dozen Ways. *ACM Transactions on Office Information Systems*, 3(2):185–203, April 1985.

[77] J.D. Ullman. *Principles of Database and Knowledge-Base Systems – Volume 1*. Computer Science Press, Rockville, Maryland, 1988.

[78] A. Van Gelder and R.W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, 1991.

[79] D.L. Waltz. An English Language Question Answering System for a Large Relational Database. *Communications of ACM*, 21(7):526–539, July 1978.

[80] D. Warren and F. Pereira. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *Computational Linguistics*, 8(3-4):110–122, 1982.

[81] R. Weischedel. A Hybrid Approach to Representation in the JANUS Natural Language Processor. In *Proceedings of the 27th Annual Meeting of ACL, Vancouver, British Columbia*, pages 193–202, 1989.

[82] S. Whittaker and P. Stenton. User Studies and the Design of Natural Language Systems. In *Proceedings of the 4th Conference of the European Chapter of ACL, Manchester, England*, pages 116–123, April 1989.

[83] W.A. Woods. Procedural Semantics for a Question-Answering Machine. In *Proceedings of the Fall Joint Computer Conference*, pages 457–471, New York, NY, 1968. AFIPS.

[84] W.A. Woods. Lunar Rocks in Natural English: Explorations in Natural Language Question Answering. In A. Zampoli, editor, *Linguistic Structures Processing*, pages 521–569. Elsevier North-Holland, New York, 1977.

[85] W.A. Woods. Semantics and Quantification in Natural Language Question Answering. In M. Yovitz, editor, *Advances in Computers*, volume 17. Academic Press, New York, 1978. Reprinted in [35], pages 205 – 248.

[86] W.A. Woods, R.M. Kaplan, and B.N. Webber. The Lunar Sciences Natural Language Information System: Final Report. BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

[87] V. Zue, S. Seneff, J. Polifroni, M. Phillips, C. Pao, D. Goddeau, J. Glass, and E. Brill. PEGASUS: A Spoken Language Interface for On-Line Air Travel Planning. In *Proceedings of the ARPA Human language Technology Workshop, Princeton, New Jersey*, 1994.