A Voice Interface System for Database Accesses using French

Lyne Lahaye

A Major Technical Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

January 1989

ISBN 0-315-49089-6

ABSTRACT

A Voice Interface System for Database Accesses using French

Lyne Lahaye

A French natural language interface to a database using voice input and voice output has been developed. The system consists of three modules: natural language menu along with speech recognition for input, SQL query generation, text generation and voice generation for output. This work integrates different commercially available software and hardware units, on personal computers such as Voicescribe (speech recognizer), Televox (French speech synthesizer), and Arity SQL (relational database) into the inhouse developed software. The emphasis of this report is in the implementation of French text generation. It is based on entity relationship model and transformational grammar. The integration of different modules of the system is done using two PC Ats and the low cost PC Quick Net.

ACKNOWLEDGEMENTS

I express my sincere thanks to my supervisor Dr. T. Radhakrishnan. His precious advice and guidance were invaluable. I thank all those people who helped me in this work, in particular, Mr. Ian Menzies, Mme Christina Pitula, Mr. Cliff Grossner, and Mme Galina Kolesova. Special thanks are due to Gordon Wall and my parents for their continued moral support throughout my studies.

Table of contents

Chapter 1 The overall system

1.0 Introduction 1
1.1 French Natural Language Menu using voice drive 2
1.2 SQL query generation 5
1.3 Sentence generation and voice output 9

Chapter 2 Subsystems

2.0 Introduction 10
2.1 Speech recognition subsystems 10
2.2 Natural language menu subsystem 16
2.3 Speech synthesizers 21
2.4 Arity/SQL 24

Chapter 3 Generation of deep structure from SQL queries

3.0 Introduction 27
3.1 Historical review 28
3.2 System's requirements 29
3.3 Design approach 32
3.4 One level query with one relation 40
3.5 One level query with multiple relations 44
3.6 Two level queries 55

Chapter 4 Surface structure generation

4.0 Transformational component 63
4.1 Model of a reader 71
4.2 Model of discourse 72
4.3 Data dictionary 74
4.4 Two complete examples 77
4.5 Problems specific to French Language 81
4.6 Limitations and possible improvements 84

Chapter 5 Integration

5.0 Introduction 87
5.1 Environment 87
5.2 Communication between modules 89
5.3 Complete integration of the modules 92

Chapter 6 Conclusion 95

References 98

Appendix 1 100

CHAPTER 1

THE OVERALL SYSTEM

1.0 Introduction

In this age of growing applications of computers, vast amount of research is done in the field of man-machine communication. As the use of computers becomes more and more wide spread, a large amount of information is generated and kept in computer readable form.

In the not too distant future, we hope, voice interfaces to computer systems will be popular for accessing these computer stored information. Although, commercial systems are available, many problems are yet to be solved in both speech recognition and synthesis. However, at this stage, it is possible to use commercially available speech recognizers and synthesizers and build usable man-machine interfaces.

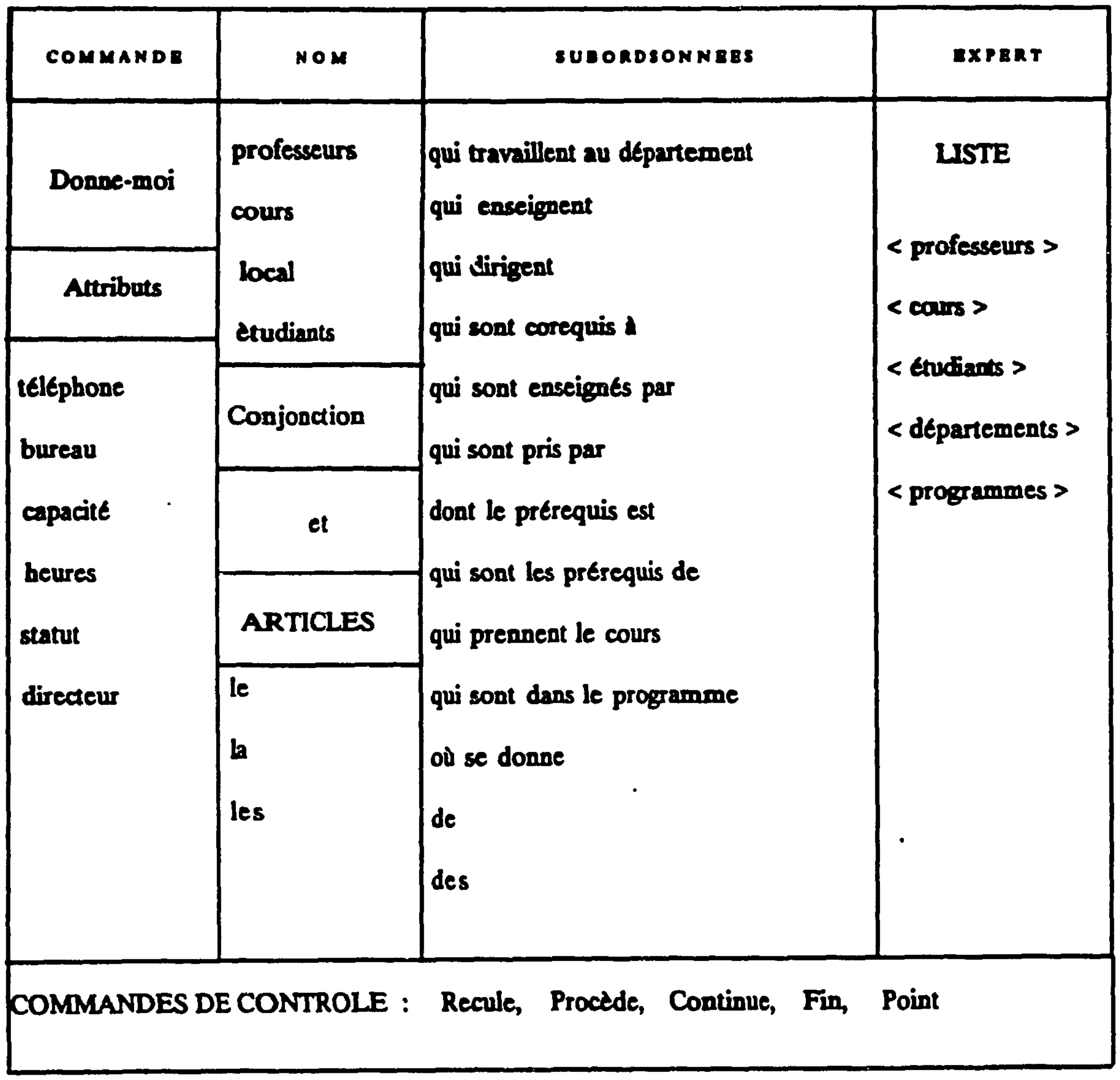
In this report, we describe a man-machine interface system using voice I/O and French natural language. It was developed on two PC ATs using software tools such as Arity Prolog, Arity SQL, Microsoft C, and PC Quick Net. The Dragon system's speech recognition board and Televox's speech synthesizer board are also used. There are three major subsystems in the development of this voice interface: Natural Language input based on NL-menu using speech recognition; SQL query generation; and thirdly Natural language generation in French and its output in spoken form. The integration of these subsystems is done on two machines connected by a network. PC Quick Net is the network tool used by the two machines.

Most of the programs were written in Arity Prolog. The programming language C was used for writing the interface driver for the Dragon system's speech recognition software.

1.1 French Natural Language Menu Using Voice Drive

Basic operation of natural language menu (NL-Menu) with Voicescribe is done as follows: a menu (see fig. 1) of words and/or phrases are displayed on a screen when the user selects a database for querying. As each word is selected by the user from his menu, a partial query is constructed and displayed in a specific window. A new list of permissible words and/or phrases are highlighted after each selection is made. The user continues in this fashion until his total query is formulated. In the process of constructing his query, the user is permitted to go back on his selection of words and change them at any stage. A menu normally contains some subdivisions which pop up and they can be scrolled up and down. If a mistake is made during this process, a word can be cancelled by a simple voice command. When the sentence is completed, the user has four options that he can choose: "procede, fin, continue and recule."

Figure 1. NL_Menu window



COMMANDE	NOM	SUBORDSONNEES	EXPERT
Donne-moi	professeurs	qui travaillent au département	LISTE
	cours	qui enseignent	
Attributs	local	qui dirigent	< professeurs >
	ètudiants	qui sont corequis à	< cours >
téléphone	Conjonction	qui sont enseignés par	< étudiants >
bureau		qui sont pris par	< départements >
capacité	et	dont le prérequis est	< programmes >
heures		qui sont les prérequis de	
statut	ARTICLES	qui prennent le cours	
directeur	le	qui sont dans le programme	
	la	où se donne	
	les	de	
		des	

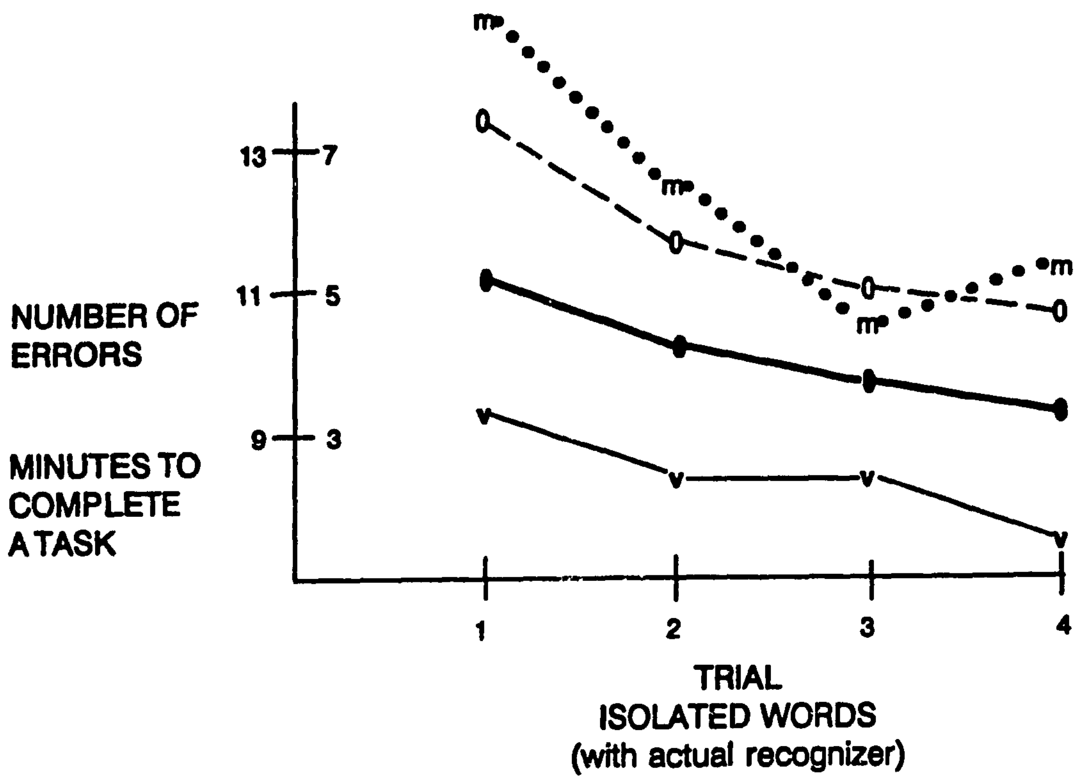
COMMANDES DE CONTROLE : Recule, Procède, Continue, Fin, Point

There are many obvious advantages with NL-menu approach. Ambiguity in the construction of a query is rare. The only requirement of the user is the ability to read and speak. Minimum learning time is required as the user is practically guided through each step and no typewriting experience is necessary. Spelling errors do not exist because of voice command and the fact that all commands are selected from the display on the screen. Voice input can be approximately twice as fast as typing. Physically disabled persons would find this mode of input/output more suitable. Simultaneously another task can be performed while the user is talking to the computer. It has also been shown that fewer errors occur in complex problems when solved by voice communication compared to other modalities (see fig. 2). [6]

There are also certain disadvantages to this means of input: the user's vocabulary and queries are controlled by the natural language menu, thus constraining him. With the current technology only isolated word recognition is possible and the system has to be pre-trained for each user. The size of the vocabulary of words that can be recognized is limited. Background noise has significant impact on the recognition accuracy.

The implementation of a French NL-menu created additional problems when compared to the English menu. Accents were needed both in the menu and the vocabulary file. The screen space restricted the use of long descriptive phrases commonly used in French, so more pop up menus were necessary to alleviate this problem. Detailed description of these problems and the solutions are discussed in chapter 2.

Figure 2. Isolated word recognition vs human interaction [5]



m = Manual errors

0 = Manual times

v = Voice errors

● = Voice times

1.2 SQL Query Generation

Structured Query Language is one of the most popular query languages among relational database systems. SQL has an English like structure. In our case, it is used as an intermediate language between the man-machine interface and the DBMS. SQL query has three major clauses: Select clause, From clause, and Where clause. The Select clause states what is to be found, and the From clause states which relations in the database are involved. The Where clause specifies a set of predicates that must be satisfied in the selection of tuples for response. Queries can be nested into multiple levels.

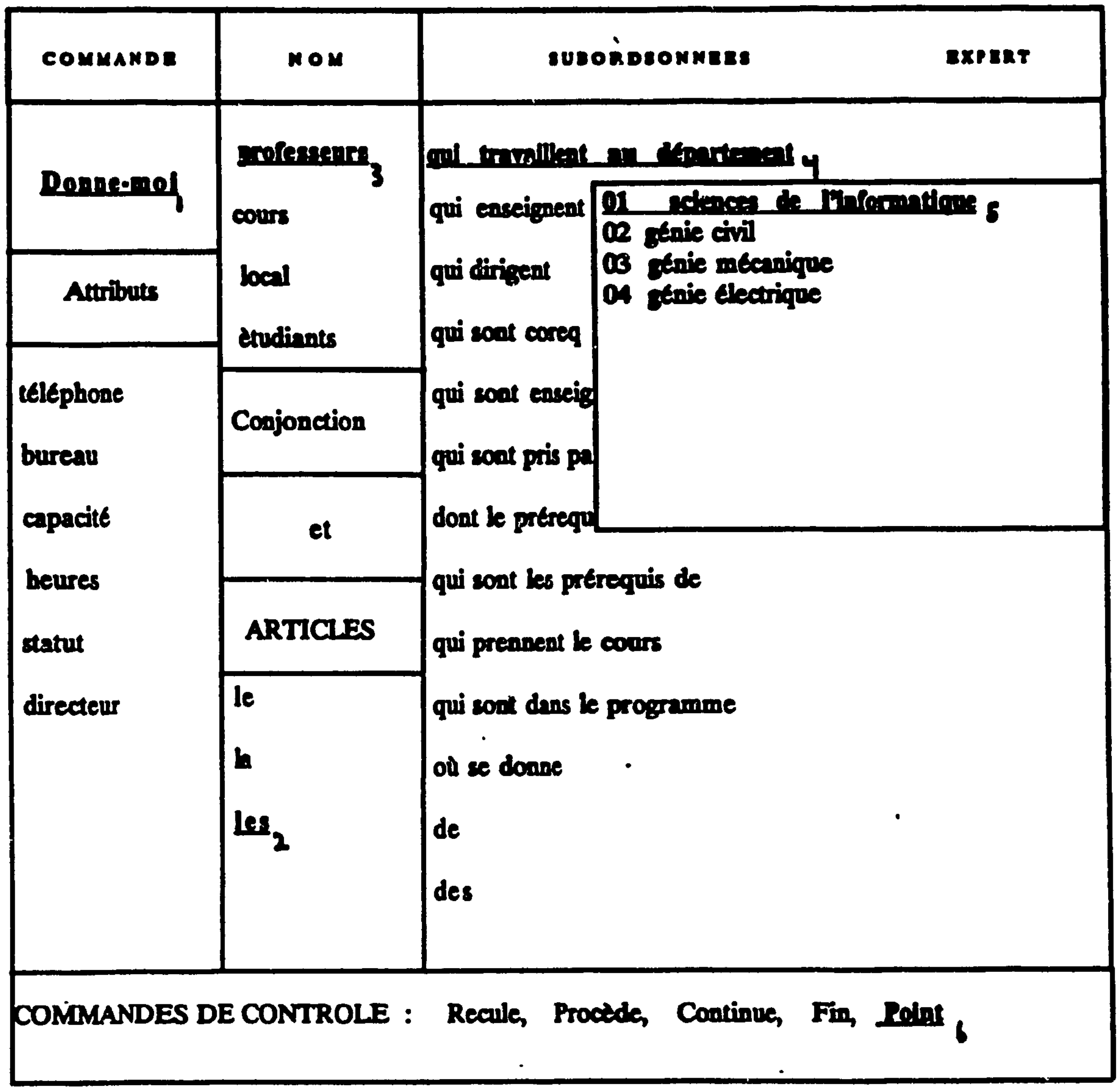
In order to generate an SQL query from the natural language input, it is necessary to structurally analyse and parse the input natural language according to a grammar. The complexity of the grammar is directly related to the menu. Imperative commands lead to less complexity in programming yet furnish equal information when compared to interrogative questions beginning with who, what, where, why, when, how and do. This is because the imperative statement uses the total database while the interrogative statements are limited to a part of the database. The attributes of the Select clause are extracted from the nounphrase (np) grammar rule (see fig. 3) while the relations for From clause and predicates for Where clause are derived from the "rel-clause" of the input sentence. Translations of natural language sentences to SQL queries are aided by the keywords in the NL_menu.

figure 3. Grammar rules (Partial program "transla.ari")

```
sentence --> verb, np, rel_clause.

verb --> [$Donne-moi $].
np --> determiner, noun,(np;no) .
np --> conj, determiner,noun,(np ; no).
conj --> [$ et $].
no(S,S).
```

```
determiner --> [$le $].
determiner --> [$la $].
determiner --> [$les $].
noun --> [$telephone $],
         { recordz(select,$telephone$,_)}.
noun --> [$bureau $],
         {recordz(select,$bureau$,_)}.
noun --> [$capacite $],
         { recordz(select,$capacite$,_)}.
noun --> [$heures $],
         { recordz(select,$dheure,fheure$,_)}.
noun --> [$directeur $],
         {recordz(select,$president$,_)}.
noun --> [$statut $],
         { recordz(select,$statut$,_)}.
noun --> [$local $],
         { recordz(select,$nolocal$,_)}.
noun --> [$professeurs $],
         {recorda(select,$profnom$,_)}.
noun --> [$cours $],
         {recorda(select,$nocours$,_)}.
noun --> [$etudiants $],
         {recorda(select,$etudnom$,_)}.

rel_clause -->  [$de $, Term,_], establish the from and
                                  where clause
rel_clause --> [$des $,Noun], from clause, subordonnee.
rel_clause --> subordonnee.

subordonnee --> (([$qui $],verbl) ; ([$dont $],nphrase);
                ([$ou $],verb2)),(pt ; conj_rel).

pt --> [$.$].
conj_rel --> [$ et $],subordonnee.

verbl  -->  [$travaillent $,$au  $,$departement  $,Noun],from
                                                  and where
verbl --> [$dirigent $,Etudiant],from and where
verbl --> [$enseignent $,Cours],from and where
verbl --> [$sont $,$enseignes $,$par $,Prof],from and where
verbl --> [$sont $,$pris $, $par $,Etud],from and where
verbl --> [$sont $,$les $, $prerequis $, $de $,Crs],from and
                                                      where
verbl --> [$prennent $,$le $,$cours $,Cours],from and where
verbl -->  [$sont $,$dans $, $le $,$programme $,Prog],from
           and where

verbl --> [$sont $,$corequis $,$a $,Noun],from and where

verb2 --> [$se $,$donne $,Cours],from and where

nphrase --> determiner, nounph.

nounph --> [$prerequis $,$est $,Cours], from and where
```

Example: "Donne-moi les professeurs qui travaillent au departement sciences de l'informatique." (Desired by the user)



Note 1,2,.... is the order in which the user has selected the options.

The generated SQL query is:

```
select pnom
from travailler
where dnom = 'sciences de l informatique'
```

1.3 Sentence Generation and Voice Output

Sentence generation is achieved in two steps: deep structure generation and surface structure generation based on transformational rules. For the generation of deep structure, we make use of the data dictionary of the stored database and the database's semantic-knowledge represented in the form of entity-relationships graphs. Several algorithms are devised for generating the deep structure of the desired sentence. Transformational rules are applied to this deep structure and a natural language sentence (surface structure) is generated. These steps are described in chapters three and four.

After sentence generation, Voice output is achieved through the use of a commercial speech synthesizer called Televox. Chapter two includes the specifications and operational details of this speech synthesizer.

CHAPTER II

SUBSYSTEMS

2.0 Introduction

A good communication between man and machines requires a complex system. There are several problems encountered in building such a system. These days several off-the-shelf subsystems are available to build good man-machine interfaces. In this chapter we review four such subsystems that are relevant to our present work.

2.1 Speech Recognition Subsystems

Commercially available speech recognizers are becoming more and more popular for the following reasons: they create a more familiar environment to the casual user using voice. No typewriting is necessary. The productivity increases due to faster access to information and because another task can be performed simultaneously. This section gives a brief overview of the different types of speech recognizers and their problems. The Voicescribe speech recognizer is particularly emphasized as it was chosen for this prototype.

There are four types of speech recognizers : isolated word, word spotting, continuous speech and speech understanding. Isolated word recognizer is the least complex and commercially the most succesful form of recognition. It requires the spoken words to be separated by pauses, thus simplifying the recognition of the start and end points of each word. The unit of recognition is the word. Speaking words in isolation may be less natural but words are pronounced more carefully than words in continuous speech.

The second type is known as word spotting. Each word to be recognized is represented by a model or template. The recognizer attempts to match these models with the incoming speech stream. No distinct pauses are required. This type is more complex due to the difficult recognition of start and end points of each word.

Continuous speech is another type of recognition. It breaks the speech stream into smaller units. These units of recognition could be words, phonemes, etc that together make up words. Continuous speech recognizers should be able to recognize word boundaries. The word-based continuous speech recognizer can be used for word spotting, but this approach is not practical when dealing with large vocabularies. Phonemes-based continuous speech recognizer recognizes individual phonemes and identifies words as sequences of phonemes. This requires a careful study of phonetic rules, vocabulary and syntax rules of the spoken language.

The speech understanding is the most complex and ambitious type. Its goal is not only to recognize the speech units but also to understand speech as humans do. This system emphasizes the meaning of speech rather than the mere recognition of individual words. In order to understand the speech , one needs lexical, semantical and world knowledge.

There are many common problems encountered by all types of speech recognition systems. No two persons talk in the same fashion, resulting in a speaker dependency. Another problem is the voice patterns of individuals could vary from one day to the next which increases the error rate in recognition. Similar or rhyming words create a problem of phonetic ambiguity. Background noise such as people talking, closing doors interferes with the speech input signal which reduces the recognition accuracy.

Voicescribe is an isolated word recognizer marketted by Dragon Lab Inc in Massachusettes, USA. It requires user-training of the vocabulary to be recognized. Its components are Voicescribe 1000 speech board, Dragon Lab software, Dragon Key utility, a library of low level functions, and VOCL compiler. The Voicescribe 1000 speech board is a single circuit board which can be inserted into one of the PC's expansion slots. The Dragon Lab utility is a training program that helps the user to train the system on the vocabulary program. It also gives feedback to the user pertaining to the confidence levels and other variables related to the recognition process.

Dragon Key is a development utility that runs in background while another application is running. It provides pop-up menus allowing the user to train and recognize words from within another application, or to see the next list of words to be recognized at any given state in the application. It provides ready-to-use language description files for use with DOS, Dbase and Lotus packages.

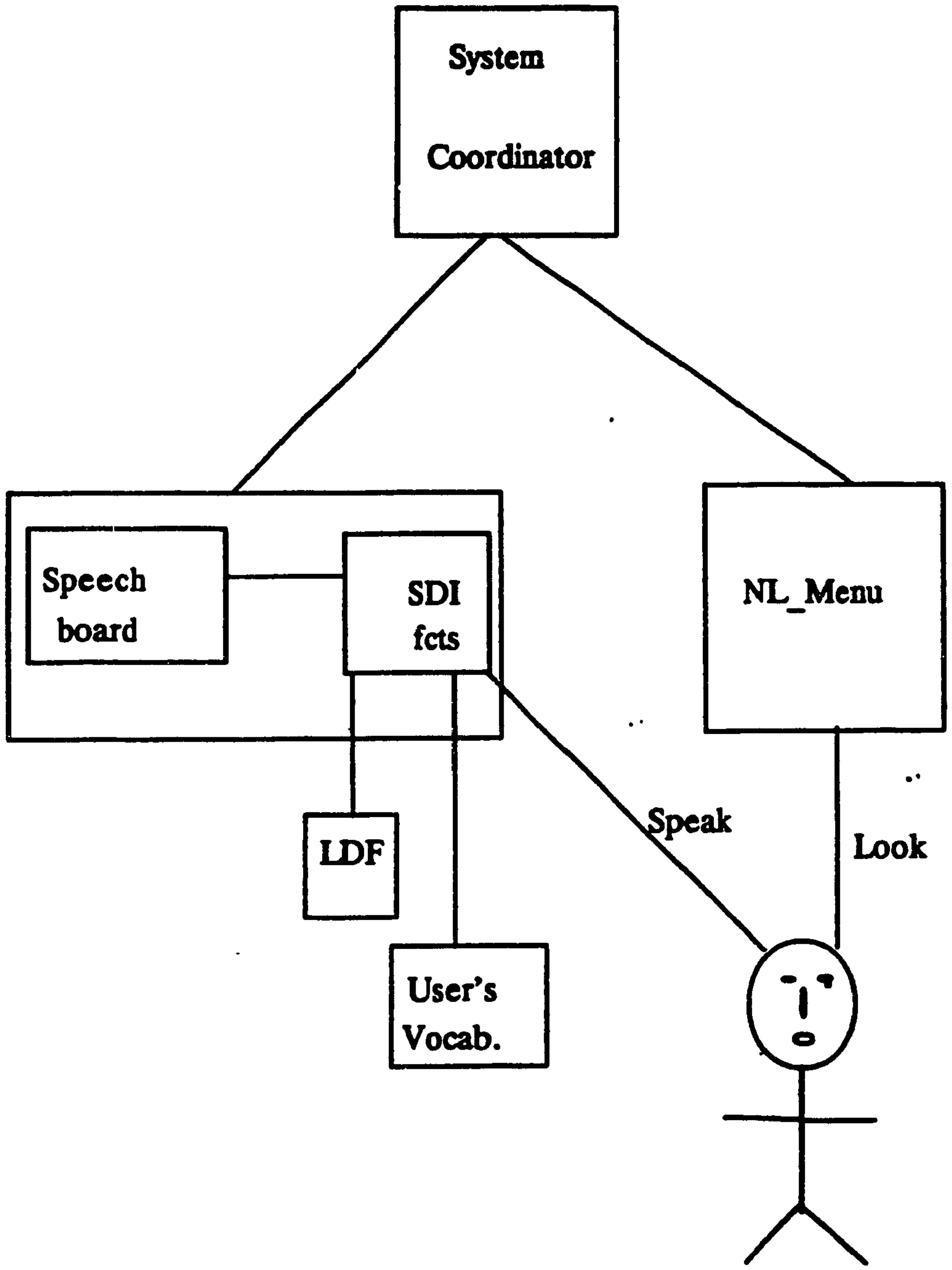
Voicescribe also offers low level voice board functions which allow easier modifications and they are not as limited as the Dragon Lab or Dragon Key is. The "speech driver" is managed directly by the programmer. He can manipulate recognition process, build specialized training routines, and modify the system's parameters. The speech driver functions are written in C and they are together called Speech Driver Interface (SDI) functions.

Voicescribe also contains a language compiler called VOCL. The VOCL compiles a given language Description File (LDF) into a useful intermediate form. The input specifications are transformed by VOCL into a network of states and production rules. Phonetic ambiguity is then minimized because the network moves from one state to the next limiting the choice of subsequent words. This can be seen with the highlighting on the screen. An example LDF is shown in the appendix.

We have built a training routine using SDI functions, and added it to Voicescribe. It builds speech patterns of words through repetitions and stores them in the user's vocabulary file. Words selected from the LDF are spoken repetitively and the speech parameters obtained from the various repetitions of a word are averaged, and this average is employed to recognize the words at a later stage. If a background noise or bad pronunciation occurs, the user may be asked to repeat the word a number of times. It is absolutely necessary that all words in the LDF go through this training process.

The system coordinator is a module written in Prolog which synchronizes the state-to-state transition of the word recognition process with the highlighting of the NL-menu. (see fig. 4) It monitors the speech board's recognition process, aaccepting words that surpass a minimum confidence level and displays them on the screen. If a word is rejected, an appropriate message is displayed in a separate window.

Fig 4: Conceptual diagram of NL_menu with voice drive



2.2 Natural Language Menu

Research in natural language interfaces to database systems became intensified in the mid 1970's. The input query in natural language is parsed, interpreted and executed and then a response to that query is also produced in the natural language such as English or French. In order to understand how we arrive at a compromise with the NL_menu approach, it is necessary to do a review of some problems in natural language understanding and generation.

One of the problems in natural language processing is encountered at parsing. Many people can say the same thing in different ways. In other words, one meaning maps many syntactic structures of sentences. It is difficult to cover all the utterances. However, most natural language systems do not require a complete coverage of the entire scope of the natural language. A limited subset of natural language could be adequate for most applications.

Another problem is related to the semantics. The meaning of a word can vary in different contexts, resulting in a possible misinterpretation of the query. To alleviate this problem both limited scope and context dependent interpretation are used. The Natural language understanding systems thus deal with a limited subset of natural language covering the domain of information contained in the database. However, limiting to a subset of natural language does not solve the problems of ambiguity entirely, but usually these ambiguities can be handled by some application specific approaches. This is especially true in the case of a small limited subset of natural language. The first systems developed used an approach called domain-specific knowledge. This knowledge is directly derived from the database structure. It consists of words and phrases referring to the information and data contained in the database. Names and fields of the database are incorporated in the grammar used for parsing, resulting in a customized language for the application. This approach helps solve misconceptions that would occur when certain words can be used in more than one context. It also creates some inconveniences such as making it necessary for the user to know the database contents to formulate a query. It limits the portability of the domain knowledge to another database world due to the dependency of its grammar. It still does not solve all types of syntactic and semantic ambiguities.

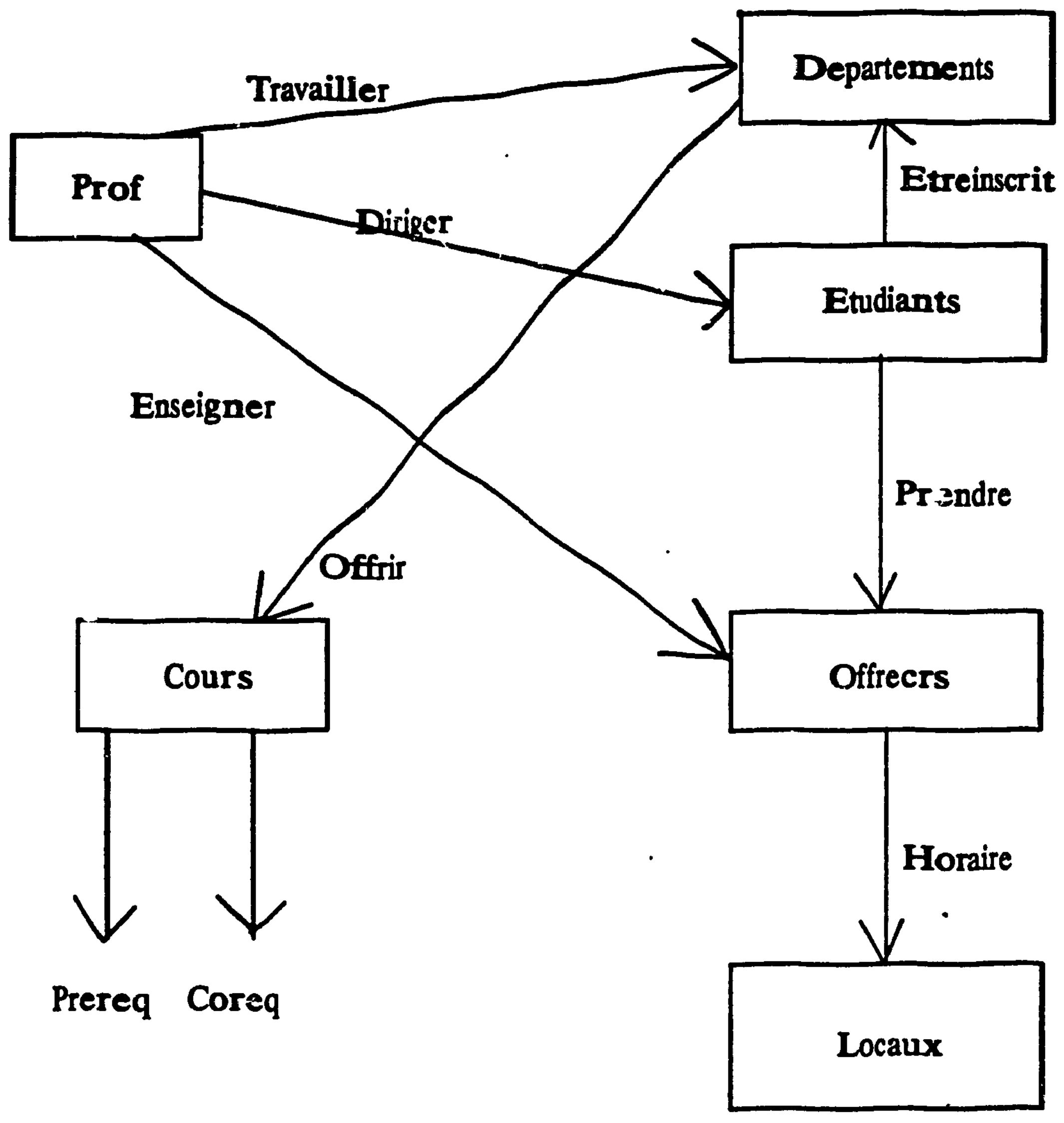
Domain-specific knowledge alone is not enough to handle the ambiguities. We need the world knowledge. This knowledge is based on how people make use of "well known concepts and meanings" while communicating with others. World knowledge is independent of aplications domain but representing it is hard.

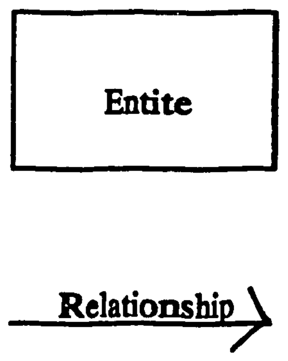
NL_menu is a completely different approach to Natural language understanding. It is a middle ground between two extremes. In this case, there is no need to train the user to the database contents for the formulation of queries. There is no mispelling errors and no unclear queries .

In NL_menu, a menu of words or phrases is displayed on the screen. These words or phrases refer directly to the information contained in the database. At every stage of query formulation, the user may select any word or phrase highlighted on the screen, using a pointing device. As words are selected, the partial query is built and displayed, and the next possible list of words or phrases are highlighted on the screen. This process goes on until a complete query is formulated. Pop-up menus are used to let the user select values for database variables where enumeration is either difficult or impossible. Usually Pop-up menu occupies a portion of the screen structured as a window.

We explain a NL_menu with an example database that is related to an university environment. It consists of six entities and nine relationships. Figure 5 shows the database schema.

Figure 5. Database schema

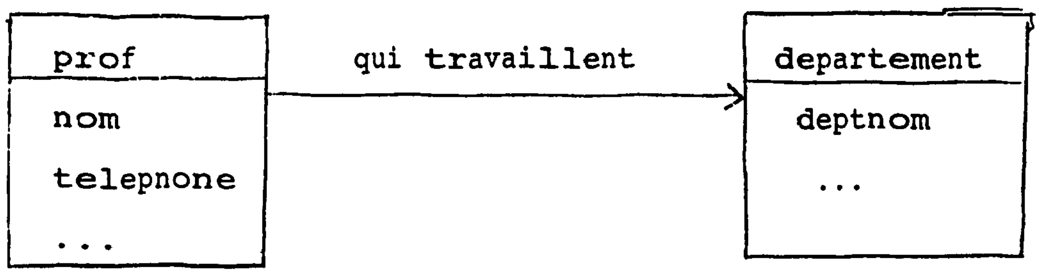




The design of the screen display for NL_menu is based on the database schema and the database itself. The first step is to determine which questions can be derived from the database schema and the values and attributes names in the database itself. For instance, by looking at the entities "prof" and "departements", and the values of the attribute "deptnom", the following query can be formulated:

"Donne-moi les professeurs qui travaillent au departement sciences de l'informatique."

By analyzing such queries we select a list of words and phrases that can be used.



The next step is to place these phrases in appropriate boxes on the screen. Effective use of screen space is important at this stage. The prototype's screen menu in our case is composed of eight different boxes : " commande, nom, attribut, subordonnees, expert, article, conjonction et commandes de controle." All queries end with the word "point". For instance, the partial query is "Donne-moi les professeurs qui travaillent au d partement sciences de l' informatique". The user at his discretion may finish his query by selecting the word "point" or continue his query by selecting the word "et" in the conjonction box.

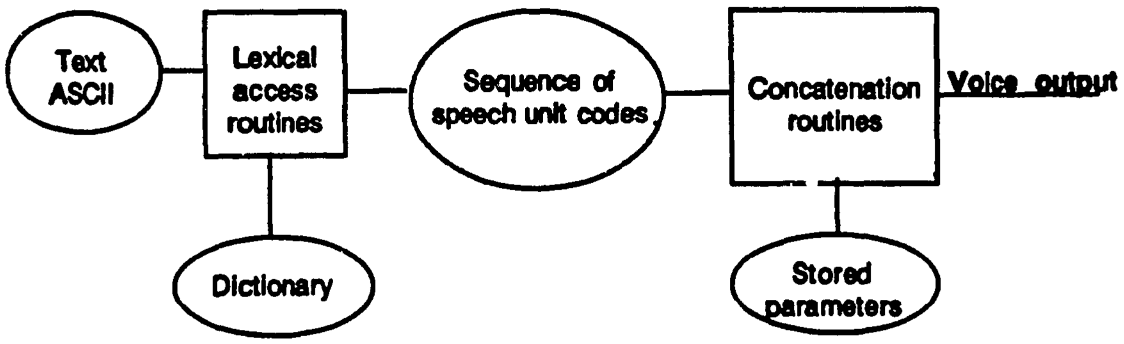
The use of French language poses some unique problems: one problem is the article l' placed in the article box, because the apostrophe is not actually pronunced but only understood, the user would have had to pronounce l' as the letter l followed by a pause and the noun. Since this is not considered natural French language, the plural form is used. The second problem pertains to the character space in the attribut box and the screen space management. Some attribute names occupied more space than was available, making it necessary to have pop-up windows, resulting in a more complex menu structure.

2.3 Speech Synthesizers

There is a lot of progress in integrated circuit technology (VLSI) and in the methodology of speech synthesis. As a result, there are commercial speech synthesizers that can generate intelligible speech. This section introduces two applications of speech synthesis and their advantages and disadvantages. Although several board level products for speech synthesis in English, based on SC-02 chip, are readily available, speech synthesizers in French are not that common. The latter is also about ten times more expensive. The product Televox used in our prototype is described below:

Voice response systems and text-to-speech systems are two of the applications of speech synthesis. Voice response systems handle text of limited vocabulary while text-to-speech systems handle unlimited vocabularies. Voice response systems are basically speech coders or vocoders that store bits stream in memory and playback through a decoder when output speech is needed. Speaking toys, warning systems and automatic telephone directory use voice response. While voice response systems do not use extensive linguistic processing, text-to-speech synthesizers convert an input text to appropriate speech units (figure 6) using extensive linguistic processing.[11]

Figure 6. Generation of voice



A good speech synthesizer maximizes speech quality, while minimizing memory space, computation speed, and algorithm complexity.

Memory size is a clear limitation to those methods which are based on the concatenation of words as speech units. The intonation and the contextual time dependency on words, need to be stored with all the words or phrases of a language, requiring a lot of memory. However, this approach would lead to higher quality of speech.

Current speech synthesizers based on VLSI chips use smaller speech units in order to minimize the memory requirements. The most common speech units are phonemes since most languages have only thirty to forty phonemes. There are other possible speech units such as syllable, demisyllable, and diphones (used in Televox). Diphones are obtained by dividing a speech waveform into phoneme sized units, and then by cutting in the middle of each phoneme. In order to reproduce a sounding speech, the spectral features of the coarticulated speech units must be smoothed at their boundaries. Researchers find advantageous to use diphones over phonemes due to equal size of unit. Because of this equality, the smoothing algorithm at the boundaries is much simpler. Phonemes having variable length, require complex smoothing rules. Also a pronunciation of a phoneme in a word or phrase is dependent not only on the neighbouring phonemes, but on the intonation and speaking rate (called phonetic context). The phonetic context influences greatly the complexity of the algorithms. Not enough is understood about the effect of the context on the articulation of a phoneme. The diphone systems found a way around this problem by storing the parameter transitions from one phoneme to the next, since the effects of coarticulation influence only the adjacent phonemes.

The memory size is dependent on the synthesizer algorithms. The algorithms require memory for storing the dictionary of speech parameters and the rules for concatenation of speech units. By reducing the size of the speech units intended for concatenation, the memory requirement is minimized.

Using LPC (Linear Predictive Coding) has its own advantages. It is an effective form of compressed speech digital data, that helps to increase the naturalness of synthetic speech.

Televox is a text-to-speech French synthesizer as mentioned earlier. It is based on diphones. Regarding the quality of the speech generated by Televox, it is understandable but lacks naturalness. This may be compensated partially by using the control parameters for intonation and rhythm that are available on the Televox synthesizer. These control parameters can be modified by the user and included in the sentence sent to the speech driver. Other advantages of Televox are the ease of its installation and the software available with it. However, there is a restriction, Televox can only read one sentence and not a paragraph, due to its internal memory limitations.

2.4 Arity/SQL

Arity/SQL is a software package that is used in our case to build a relational database and to process SQL queries. Arity/SQL is not designed to be a stand-alone relational database management system. It translates SQL statements into an intermediate form that is executed by the SQL compiler. A powerful library facility is available with Arity/SQL. Arity/SQL is fully compatible and is embedded into Arity Prolog. It is a single user system and supports these data types :

- 16 bit signed integer type
- 64-bit float type
- a char(n) type where n is the length of the string, greater than 0 and less than 16K bytes.

An attractive feature of Arity/SQL package is in its permitting a relational database to be divided into units called "worlds". This division may be done for two reasons:

1. If the relational database is larger than 4Mb of data, another "world" is absolutely necessary. A total of 256 worlds with a limit of 4Mb in each allows an application of up to one gigabyte of addressable space.

2. Time and space could be saved by placing the relational database away from the Prolog's database in a different world. This allows more room for the application's database and the relational database can be retrieved very quickly by a command.

Another feature of Arity/SQL is its capability for saving all responses to SQL queries in the Arity/Prolog database. Arity/SQL has an example mode which allows the user to test queries directly from the relational database for testing purposes.

A disadvantage with Arity/SQL is that it is not fully debugged and not completely documented. We discovered that Arity/SQL would not accept a second predicate in the inner level of the nested two level queries. In the following example, because two predicates (progrnom and statut) were used, it was not accepted.

```
(i.e.,) Select etudnom, telephone
        from etudiants
        where progrnom,statut in (
           select progrnom,statut
           from etudiants
           where etudnom = 'Lyne Lahaye');
```

Although many packages can be obtained in the market that are more professional than Arity/SQL, we chose this because of its availability, low cost, low memory requirements, and it being fully embedded in Prolog. Also it saves programmer's time by not requiring the creation of extensively large programs for database search and management.

CHAPTER III

GENERATION OF DEEP STRUCTURES FROM SQL QUERIES

3.0 Introduction

Researchers are currently placing a great deal of emphasis on text generation systems. When applied to database systems, the reason for this emphasis is apparent: Interpreting a complete sentence is much easier and more natural than understanding the structured output from a computer, particularly for a novice. It is also well suited for the handicapped and the illiterates. There are two possible approaches for producing natural language text. The first consists of storing canned text, and the second translates a limited set of semantic structures into sentences. Each method has certain advantages as well as disadvantages. A text generation system employing either of these approaches requires four basic components: a knowledge representation method, a linguistically justified grammar, a model of the reader, and a model of discourse. My project is based on the second approach. The process employed to generate text is first described using examples. Our implementation handles certain problems specific to the French language. Some of the limitations and possible improvements to the proposed text generation system are also discussed.

3.1 Historical Review

In the early days of text generation systems, the technique employed consisted of planning the necessary text in advance and storing it as text strings for future display. This technique has several major disadvantages. Designers must anticipate all answers to all possible questions. This method cannot be employed in the case of large databases due to memory constraints. Another potential problem is the possible inconsistencies between the information actually accessed by the program, and the sentences generated. In other words, the matching of text strings to certain knowledge structures employed by the system could produce incorrect or unacceptable answers. Finally, the program has no conceptual model of what it is saying, and therefore no way of distinguishing between any of the text strings. However, there are certain advantages: this technique guarantees that the text can be complex as well as elegantly written, which is not necessarily guaranteed with the second approach.

The second technique produces text by directly translating knowledge structures. As all possible answers need not be anticipated, this method can handle large databases. Inconsistency is no longer a problem; the relation between the knowledge structures being transformed and the structures used in the program's algorithms ensure that the sentences produced are consistent with the information being extracted. The closure problem due to the nonexistence of a conceptual model, is overcome by transformations that handle large classes of knowledge structures. However, this technique does not guarantee the production of complex text. The quality of the text is largely dependent on how the knowledge is structured and how the algorithms make use of them. A lot of research must still be done before elegant and complex text can be generated. In spite of these short comings, this technique is the most popular.

3.2 System's Requirements

The goal of any text generation system is to produce natural language text. The generated text must be understandable to the reader, well written, and correspond to the reader's intentions.

Text generation systems require some method to represent the knowledge they employ to generate text. Two sources of knowledge are necessary in this process : domain specific knowledge, and world knowledge. The domain knowledge in the case of database applications, consists of both factual and semantic knowledge about the data. World knowledge includes information about the natural language. These two sources of knowledge are the basic constituents of the reasoning process, which consults them to create its semantic structures. Consultation involves an existing formalism of these knowledges. Many methods have been proposed to represent this knowledge: predicate logic, propositional logic, semantic networks, scripts, frames, entity relationship model, etc.

All these formalisms attempt to represent the entire world, but rarely succeed in representing more than some aspects of it. The main problem that arises resides in abstractions such as time, space, cause, obligation, negation, possibility, and quantification, among others. Formalisms are generally evaluated according to how well they represent the reasoning process, how easily they can be examined and updated, and their relevancy to solving particular problems. The quality of the generated text is limited by the chosen knowledge representation method and the algorithms which make use of them.

Employing an appropriate knowledge representation method does not guarantee grammatically correct answers. Other components are necessary to generate correct responses. The goal of text generators is to express ideas in the form of sentences constructed from a set of words. A comprehensive, linguistically justified grammar is a necessary component that determines the ways in which these words are combined into sentences. It produces sentences by applying rules of arrangements of the set of words. The three principle grammars proposed in the literature for this purpose are Transformational Grammars, Augmented Transformational Networks (ATN), and Case Grammars. Although a good grammar and knowledge representation method are employed in a text generation system, the generated output may not completely satisfy the reader's expectations. Therefore, a model of the reader is necessary to clarify the context.

Most text generation systems do not employ a model of the reader. This component is still in its infancy, although it is essential to the text generation process. Its purpose is to avoid misleading and to provide useful answers by taking into consideration the reader's knowledge. Developing a model of the reader requires four types of knowledge [7]:

- what is obvious
- what has been told so far and what is obvious from that
- what the other party believes
- what is currently in the reader's attention

The quality of generated text is improved by recognizing the reader's knowledge and customizing the responses to it. The quality of the text can be further improved by employing a model of discourse.

Text generation systems may express ideas in more than one sentence. Therefore, discourse rules are required to organize the sequence of sentences into paragraphs and to order the paragraphs to express complex ideas. If multiple sentences are involved, this component is essential to produce well structured text. The generated sentences follow an order established by the discourse model.

There are four components which are considered essential in determining the quality of the text produced. The first component, knowledge representation, helps construct meaningful sentences whose syntax is controlled by the grammar. The model of the reader helps to build an understandable and meaningful answer. The model of discourse organizes the group of sentences into paragraphs to express complex ideas.

3.3 Design approach

The goal of any text generation system is to generate a precise, well structured, and understandable piece of prose. Our design achieves this goal by employing the following approach. It generates French text by translating a limited set of representational structures. The generation process is the following: an SQL query and the factual knowledge are mapped into entity and relationship network graphs to construct the semantic structures. We chose the entity relationship model as the knowledge representation method for various reasons that will be discussed later. The grammar component controls the syntax of these semantic structures, and builds the natural language text. An adaptation of the Transfornational grammar is used for this purpose. The design employs a model of the reader, and of discourse to produce good quality text. A data dictionary that contains linguistic information and information about the contents of the database is consulted during the generation process. These components of the text generation system are explained through several examples.

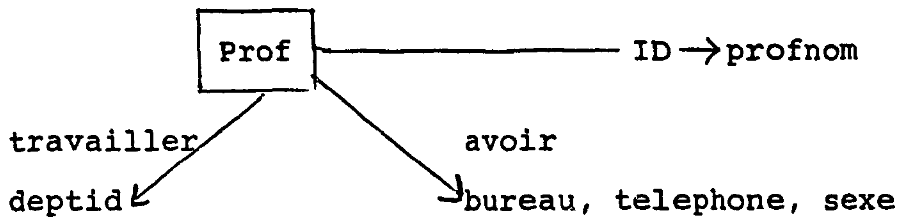
Knowledge Representation

A designer's first consideration is the selection of an appropriate knowledge representation method. We chose the entity relationship model developed by P. Chen. The evaluation criteria on which this choice is based, and a description of its implementation follow.

Criteria such as how easily the stored knowledge can be accessed, examined and updated have been mentioned in this literature. Another important consideration specific to our application is how well the chosen method represents relational database. The entity model satisfies all these requirements. Its representation is close to that of a relational database. Its entity relationship network graphs embody the semantics of the database and are easily examined. The information contained in these graphs gives a complete conceptual view of the database that can be easily accessed and modified.

The entity relationship model can be used to both help build the database, and to represent knowledge about it. The semantics of our data are represented by means of entities, and relationships between entities. The entities represent objects, whereas relationships represent the relationship between two entities. All entities and relationships have their own attributes and sets of values that are stored in the database. The objects are the tables in our database, and consist of etudiants, prof, cours, departements, offrecrs, locaux. The relationships are: diriger, etreinscrit, travailler, offrir, enseigner, prendre, horaire, prereq, coreq. A conceptual view of the database is represented in the database schema in figure 5.

The entity relationship model employs entity and relationship network graphs. These graphs provide the semantic structures that will be employed to construct French sentences. The entity network graph is composed of the entity and its attributes. The entity has labelled arcs going towards the attributes. The labels correspond to transitive verbs. For instance, the entity network graph for "prof" is:

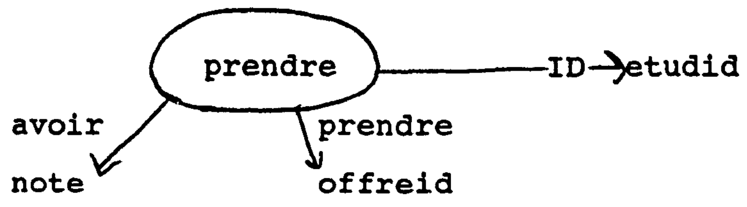


The direction of the arcs has a syntactic meaning. The attributes are the deep structure "object", whereas the attribute "ID" is the deep structure "subject". Also the label "ID" is used to denote the primary key of the relation. This knowledge will help us produce the deep structure from the SQL query: For 'select profnom, bureau from prof':

Subject of the sentence will be profnom
Object of the sentence will be [avoir , bureau]

The relationship network graphs are similar to entity graphs. They represent the relationship between entities.

Example: The "prendre" relationships network graph between the entities "etudiant" and "offrecrs".



Etudid is the primary key of the entity etudiants, and offreid is the primary key of the entity offrecrs.

The entity relationship model is a knowledge representation method that is easy to implement and appropriate to our application. Its closeness to the relational database, and the ease with which it can be examined and modified, makes it the most suitable choice. Our next consideration is the choice of grammar that will control the syntax of these semantic structures.

Grammar

A good grammar is an important component in the design of a text generation system. It helps to generate syntactically correct sentences. As mentioned earlier, there is a wide range of grammars to choose from, such as Transformational Grammars, ATNs, and Case Grammars. We have chosen a Transfomational grammar for the following reasons, and made certain adaptations to it in our implementation. Our requirements:

1- that the grammar be capable of generating sentences for any type of SQL query. 2- that it clearly express the constituents of the sentences (subject, verb, object, etc). This requirement is necessary due to the chosen knowledge representation method.

A transformational grammar satisfies these two requirements. The transformational grammar is primary designed for natural language generation, and has those characteristics defined by the second requirement. The second requirement cannot be satisfied by either an ATN or a case grammar. The case grammar combines the factual and semantic knowledge resulting in a conflict with our knowledge representation method. The transformational grammar is the most suitable choice for this reason.

The transformational grammar was first developed by Chomsky. It consists of two components : a basic component, and a transformational component. The basic component, which is a context free grammar, generates a deep structure. The deep structure conveys the semantics of the sentence. The transformational component, which is a set of transformational rules that operate on deep structures, generates a grammatically correct sentence which is called the surface structure. We use Chomsky's transformational grammar, and a modified set of transformational rules.

Basic Component

This module relates the semantic structures (entities and relationships network graphs) to the SQL query and the answer (factual knowledge) by using algorithms, and then generates the deep structures. These algorithms are developed for each class of query. Some clarifications are required at this point about the basic form of a deep structure and the classes of queries before describing the algorithms.

The basic form of a deep structure is the following.

```
Sentence ->  {Relation_name, Type, Subject, Pp, Object}
Subject ->    Relation_name, Type| attribute name, value|
               'certains',attribute name
Pp ->         verb, Predicates, {Conj, Pp} | empty
Predicates -> operator, attribute name, value {Conj,
                Predicates}
Conj -> and | or
Object ->  verb, Attr_list, {Object} | empty
Type -> entite| relation|rr
Attr_list -> {attribute name, value}

BNF notation : | = optional   {} = 0 or more times
```

A deep structure can be composed of many sentence structures having a relation name, type, subject, pp and object. The possible types are entity, relationship or RR. RR is a special case that corresponds to travailler, etreinscrit, enseigner, offrir. These relationships do not have new attributes when the entities are joined. Therefore a view of these relationships is used. If an SQL query has the relation travailler, then we process two entities rather than one relationship for reasons of efficiency. The subject is usually an attribute name having "ID" label when mapped to the appropriate network graph. The Pp component of a sentence considers only the elements in the where clause, whereas the Object component considers the elements in the select clause. The reason for this separation is to avoid repeating the same conditions for each sentence, when they are identical.

The classes of queries that can be handled by our system are :

One level query or base class (they are applied in the algorithm of other classes)

class 1 :	one entity relation
class 2:	one relationship relation
class 3:	one relation without where clause

One level query multiple relations

class 4 :	two entities relations
class 5:	one entity and one relationship relations
class 6:	two relationships relations
class 7:	one entity and one rr relations
class 8:	one relationship and one rr relations

Two level queries

	Outer Query	Inner query
class 9:	one relation	same relation
class 10:	one relation	one relation
class 11:	one relation	two relations

Having identified these classes, it is possible for us to build algorithms for them that will interpret the SQL query, and generate the deep structures.

The sources of knowledge needed by the algorithms are the data dictionary and the entity and relationships network graphs. Some lexical entries are controlled by the grammar, such as "meme", "anaphoric", but contrary to Marakakis, no article is inserted. The articles are handled by the transformational component. Algorithms are based upon the type of query such as one or two level; the number of relations at each level; and the type of relations. The basic algorithms that are used by other algorithms are the one for one level query with one relation, and one for one level query without any where clauses. Another specification is the algorithms consider the following elements in order to generate proper responses.

- the type of attributes in the query (primary keys or not)
- the place in the query (select or where clause)
- the type of boolean operators in where clause (and,or)
- the network graph corresponding to the query.

The attributes, relations, and conditions are extracted from SQL query before the algorithms are applied. Each attribute has the value produced by the database search.

The algorithms pertaining to the various query classes specify how the subject, object, and pp clauses of the target sentences are selected. 3.4 One level query with one relation

Class 1 One entity relation

Subject selection:

1) If there is an attribute name in the select clause having "id" as label in the entity network graph, then it is the subject.

2) If there is an attribute name in the where clause which has "id" as its label and it is not followed by or, then it is the subject.

3) If there is no "id" in both clauses, then the relation is the subject.

The end user knows the values in the "where" clause, and he is interested in the attributes specified in the "select" clause.

Object selection:

The remainder of the attributes in the select clause accompanied with their verbs (labels) are chosen as objects.

Pp selection:

Pp is the remainder of the attributes in the where clause accompanied with their verbs (labels) that are not referred in the select clause.

Example:

```
select bureau,telephone
from prof
where profnom = 'Shinghal';
```

Entity network graph of prof is :

```
schema(prof,id,[profnom]).
schema(prof,avoir,[bureau,telephone]).
```

Data dictionary required:

relations(entite,prof).

The deep structure of the entity prof is :

[prof,entite,[profnom,["Shinghal']], [],[[avoir, [[bureau, [H961-9]], [telephone,[8483000]]]]]]

Subject = [profnom,['Shinghal']]

Pp = []

Object = [[avoir,[[bureau,[H961-9]],[telephone,[8483000]]]]]

The surface structure is:

<u>Le professeur Shinghal a le bureau H961-9 et a le telephone 8483000.</u>

Class 2 One relationship relation

<u>Subject selection:</u>

1) same as in class 1 except we refer to the relationship network graph instead of the entity network graph.

2) same as in class 1

3) if there is no id in both clauses, the subject is the attribute name in the graph having "id"

<u>Object selection:</u>

same as in class 1

<u>Pp selection:</u>

same as in class 1.

<u>Example:</u>

select note

from prendre

where offreid = 'COMP771287AA';

Entity network graph of prendre is

```
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(prendre,avoir,[note]).
```

Data dictionary is

```
relations(relation,prendre).
```

Deep structure of prendre is :

Subject = [certains,etudid]

Pp = [pp,[[prendre,[[offreid,[COMP771287AA]]]]]] Object = [[avoir,[[note,[A,B,B]]]]]

The surface structure is :

<u>Certains etudiants prennent le cours COMP771287AA et ont la note A.</u> <u>Certains etudiants prennent le cours COMP771287AA et ont la note B.</u>

Class 3 One relation without where clause

A) Relation is an entity

1) If there is one attribute in the select clause , then the subject is the entity, and the object is the attribute name.

2) If there are many attributes in the "select" clause and one attribute has the label id then it becomes the subject. The other attributes are the parts of Object grouped with their verbs.

3) If there is no id in both clauses, then the subject is the entity and the object is all the attributes.

B) The relation is a relationship

1) If there is only one attribute in the select clause and it has a label "id" in the relationship network graph, then it is the subject. The object is taken from the graph. It corresponds to the attribute entity being involved in this relationship.

2) same as in class 3 A

3) If there is no id, the subject is the attribute entity having a label "id" in the graph, and the object is the attribute in the select clause.

<u>Example:</u>

```
select etudid,note
from prendre;
```

Relationship network graph of orendre is :

```
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(prendre,avoir,[note]).
```

Data dictionary required

```
relations(relation,prendre).
```

The deep structure is

```
[prendre,relation,[etudid,['Regis Cardin','Lyne Lahaye']],
[], [[prendre, [[offreid,[novalue]]]], [avoir, [[note,
[A,B]]]]]]
```

```
Subject = [etudid,['Regis Cardin','Lyne Lahaye']]
Pp = []
Object = [[prendre, [[offreid,[novalue]]]], [avoir, [[note,
```

[A,B]]]]]

The surface structure is :
<u>Regis Cardin prend un cours et a la note A. Lyne Lahaye prend un cours et a la note B.</u>

3.5 One level query with multiple relations

This group of algorithms handles each relation in the following way. We apply the base class algorithms 1,2 or 3 to each relation in order to form deep structures which are then modified. The modifications for each class are described below.

Class 4 Two entities relations

A join operation is performed on the key common to the two entities. A relationship network graph of the two entities creates a new deep structure (Subject, empty, Object). The subject and object are determined by the relationship network graph of these two entities. The head of the labelled arc between these entities is the object, whereas the tail of the arc is the subject, resulting in a new deep structure. This new structure will have as relation name the relationship name, and its type will be RR. This new deep structure will be placed between the two entities of the structures in a sequence that is established by the model of discourse. If two structures have the same subject then the second structure's subject will have the word "anaphoric" added to it. If the object in the new deep structure is the subject or object of the following structure, then this entity structure's subject or object will be modified by adding anaphoric. The common key in the entity structure will be deleted if it is not a primary key. This avoids repetition. An example follows.

The SQL query is :

```
   Select pnom, bur, pres
   from travailler
   where did = 'INFO';
```

The entities network graphs are:

```
schema(prof,id,[profnom]).
schema(prof,avoir,[bureau,telephone]).
schema(departements,id,[deptnom,deptid]).
schema(departements,avoir,[pt_etud,tp_etud,president]).
```

Data dictionary required:

```
relations(entite,prof).
relations(entite,departements).
```

The relationship "travailler" is translated to two entities such as "prof" and "departements". The algorithm then applies the step 1. Two structures are constructed.

```
   Relation name : prof
   Type: entite
   Subject : [profnom, [Radhakrishnan,Suen]]
   Pp: [pp, [[travailler,[[=,deptid,INFO]]]]]
   Object: [[avoir,[[bureau,[H961-9,H961-16]]]]]
```

Relation name : departements

Type: entite

Subject: [deptid, INFO]

Pp : []

Object: [[avoir,[[president, ['Bui','Bui']]]]]

We find the common key "deptid". The algorithm deletes "deptid" associated to the verb "travailler" in the prof's deep structure. The subject of departements's deep structure is modified to [anaphoric,[deptid,INFO]], and a new deep structure is added between prof's structure and departements's structure.

Relation name : travailler

type : RR

Subject: [anaphoric,[profnom,[Radhakrishnan,Suen]]

Pp: []

Object: [[travailler,[[deptid,INFO]]]]

The surface structure is :

<u>Le professeur Radhakrishnan a le bureau H961-9. Il travaille au departement INFO. Ce departement a comme president Bui. Le professeur Suen a le bureau H961-16. Il travaille au departement INFO. Ce departement a comme president Bui.</u>

Class 5 One entity and one relationship relation

The two network graphs are joined in order to establish the common key. We then find the label of this common key in the relationship network graph. If the label is "id", meaning it is the subject of this structure, "anaphoric" is added to the subject. If the label is not "id", the common key can be placed in Pp or Object. After finding the place of the key, it adds anaphoric to the common key. Anaphoric specifies that the attribute has been previously referred to, so a pronoun or "adjectif demonstratif" will be added. The sequence of the structures is fixed according to the following rule. The entity's structure will always precede the relationship's structure, as the relationship always refers to the entity structure.

Example:

```
select offreid,note
from etudiants,prendre
where etudnom = 'Lyne Lahaye' and etudiants.etudid =
prendre.etudid ;
```

The network graphs are:

```
schema(etudiants,id,[etudid,etudnom]).
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(prendre,avoir,note).
```

Data Dictionary

```
relations(entite,etudiants).
relations(relation,prendre).
```

Each relation (etudiants and prendre) passes through the algorithms of one level query with single relation. The result is :

```
[etudiants,entite,     [etudnom,[Lyne     Lahaye]],
```

[pp,[[avoir,[[etudid, [no_value]]]]]], []]

[prendre,relation, [etudid, [no_value]], [], [[prendre,[[offreid, [COMP772485AA,COMP771287AA]]]], [avoir, [[note,[B,[]]]]]]]

After the joining of the two structures, the result is : [[etudiants,entite, [etudnom,[Lyne Lahaye]], [pp,[[avoir, [etudid, [no_value]]]]],[]], [prendre,relation, [anaphoric, [etudid, [no_value]]], [[prendre,[[offreid, [COMP772485AA,COMP771287AA]]]], [avoir, [[note,[B,[]]]]]]]]

The surface structure is :
<u>Lyne Lahaye a un code. Il prend le cours COMP772485AA et a la note B. Lyne Lahaye a un code. Il prend le cours COMP771287AA et a la note en progres.</u>

Class 6 Two relationships relations

The two relationships network graphs are joined in order to establish the intersection key. The position of the intersection key will establish the precedence of the relationships.

a) If the common key is the primary key (subject) in one structure, whereas the key is not the primary key in the other structure, then the precedence goes to the structure having that key as non-primary key. The sequence will be : Subject 1 Verbl Objecl. Objectl as subject2 Verb2 Object2. This form of the sentence is chosen arbitrarily, and the reverse sequence will also do. However a good quality response can be generated by considering, on which subject should the focus lie for the opening sentence.

b) Both relationships network graphs have the common key as primary key. Therefore, any relationship can precede.

c) In both cases a and b, the second structure is modified. The word "anaphoric" is added to the common key of the second structure as this key has been referred to in the first structure.

Example:

```
select etudid
from prendre,horaire
where prendre.offreid = horaire.offreid and nolocal = H-843
```

Network graphs required are:

```
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(horaire,id,[offreid]).
schema(horaire,etre,[nolocal,jour]).
```

Data dictionnary:

```
relations(relation,prendre).
relations(relation,horaire).
```

The deep structures before the joining are:

```
[prendre,relation,[[etudid,[1310151,2222222]],
[pp,[[prendre, [[=,offreid,no_value]]]]],[]]

[horaire,relation,[offreid,[no_value]],   [pp,[[etre,   [[=,
nolocal, H-843]]]]],[]]
```

The inter key is offreid, and offreid is the id of horaire. Therefore the precedence of structures is prendre followed by horaire. Horaire has one modification to the inter key: adding "anaphoric". The joined deep structures:

[[prendre,relation,[[etudid,[1310151,2222222]], [pp, [[prendre, [[=,offreid,no_value]]]]],[]],[horaire, relation, [anaphoric, [offreid,[no_value]]], [pp,[[etre, [[=, nolocal, H-843]]]]],[]]]

The surface structure is :
<u>L'etudiant avec code 1310151 prend un cours. Ce cours est au local H-843. L'etudiant avec code 2222222 prend un cours. Ce cours est au local H-843.</u>

Classes 7 and 8

The class 7 and class 8 are special cases of class 4 and class 5 respectively. They are special in the sense that one of the E type is replaced by an RR type.

Class 7 One entity relation and rr relation

This algorithm generates a deep structure of type RR.

Select the entity of RR, here after denoted as Err, that is involved with the entity (E) relation by mapping the relations network graphs, and at the same time find the intersection key (<u>inter key</u>).

Remove the inter key from E or RR's deep structures which has a label different from "id" in order to eliminate the repetition.

A new deep structure having type RR and a relation name corresponding to the relationship between Err and E relations is created. The new structure's subject is the attribute name of the entity (E or Err) that is the tail of the arc represented in the relationship network graph. The attribute name can be positioned as Subject, Object or Pp in its deep structure, resulting in a search for its value. The new structure's object is the head of the arc. Its value is picked up through a search in the following positions : Subject, Object or Pp. There is an easy way to search for the attribute name and its value in a position. The subject usually has "id" as a label, so we find the label of the attribute name. If it is different to "id", we search through Object and Pp. The verb for this new deep structure is picked up from the relationship network graph.

Modifications may be made to the new RR's deep structure, depending upon whether the Subject or Object have been previously referred. Before doing the modifications, tests are made to check the position, and possible elimination of existing structures, such as the E or old RR structures. Potential elimination of a structure is performed if a structure has no PP or Object. This case of no Object or PP is possible after deletion of the inter key in that structure. The position of the existing structures, such as E and new RR, are determined by the model of discourse. The position of these structures depends on their relation to the old RR's structure.

<u>Example:</u>

```
select pnom,dnom
from travailler,offrecrs
where travailler.pnom = offrecrs.profnom and offreid =
COMP771287AA;
```

Travailler consists of two entities, prof and departements. After the process of prof and departements, we have this structure with type RR.

```
[travailler,rr,[profnom,['Shinghal']],              [],
[[travailler,[[deptnom,['sciences de l informatique']]]]]]
```

This structure is then joined with offrecrs's structure which is:

```
[offrecrs,entite,[offreid,[COMP771287AA]],  [pp,[]],  [[etre
enseigner, [[profnom,[Shinghal]]]]]]
```

The offrecrs network graph is :

```
schema(offrecrs,id, [offreid]).
schema(offrecrs,etre enseigner,[profnom]).
```

The entity prof of travailler is involved with the entity offrecrs, and the inter key is profnom. We remove the profnom from offrecrs for the following reason: Profnom is not an id. A new deep structure is the following:

```
[enseigner,rr,[anaphoric,[profnom,[Shinghal]]],  [],
[[enseigner, [[offreid,[COMP771287AA]]]]]]
```

And we delete the offrecrs structure which doesn't correspond to a complete sentence. [offrecrs, entite, [[offreid, [COMP771287AA]]],[], []].

The surface structure is:

<u>Le professeur Shinghal travaille au departement sciences de l'informatique. Il enseigne le cours COMP771287AA.</u>

Class 8 One relationship relation and one RR relation

Select the entity of the RR relation involved in the relationship relation (R) by mapping the appropriate network graphs, and find the intersection key (inter key). The inter key in the relationship's structure is modified by adding the label "anaphoric" to it. This modification is necessary as the RR's structure is placed in front of the relationship's structure, to introduce it.

<u>Example:</u>

```
select etid
from etreinscrit,diriger
where etreinscrit.etid = diriger.etudid and profnom =
'Radhakrishnan' and dnom = 'sciences de l informatique';
```

The 'etreinscrit' relation is divided into two entities : etudiants and departements; and those two entities are processed through the class 4 algorithm giving this deep structure:

[etreinscrit,rr,[etudid,[1310151]], [], [[etreinscrit, [[deptnom, ['sciences de l informatique]]]]]].

This deep structure is then joined with the relationship diriger structure. Diriger's structure is the following.

```
[diriger,relation, [profnom, [Radhakrishnan]], [pp,[]], [[
diriger, [[etudid,[1310151]]]]]]
```

The entity in the RR involved with diriger is etudiants. The inter key is etudid. The sequence of the structures is etreinscrit followed by diriger. The etudid word is modified in the diriger's structure; we add anaphoric. The result is :

```
[[etreinscrit,rr,[etudid,[1310151]], [], [[etreinscrit,
[[deptnom, ['sciences de 1 informatique]]]]]],
[diriger,relation, [profnom, [Radhakrishnan]], [pp,[]], [[
diriger, [[anaphoric,[etudid,[1310151]]]]]]]]
```

The surface structure is :

L'etudiant avec code 1310151 est inscrit au departement sciences de l'informatique. Le professeur Radhakrishnan dirige cet etudiant.

Network graphs required are:

```
schema(diriger,id,[profnom]).
schema(diriger,diriger,[etudid]).
schema(etudiants,id,[etudid,etudnom]).
schema(etudid,etreinscrit,[deptid]).
schema(departements,id,[deptnom,deptid]).
```

3.6 Two level queries

There are several possible types of two level queries. We consider in our work only those two level queries which contain one relation in the outer query and one or more relations in the inner query. This leads to three classes, referred to as class 9, class 10, and class 11 below. These queries can be recognized by having a predicate called link_attribute in the outer query's "where" clause corresponding to the attribute of the inner query's "select" clause. Arity/SQL allows a limitation of predicates in the "where" clause connected to the attributes in the inner query's select clause. There is a common part to the classes (9, 10, 11): The outer query is processed normally using one of the algorithms of one level queries, producing deep structures. The inner query with their appropriate attributes and conditions are processed using the base algorithms for one level queries with a single relation. The deep structures produced are related to the outer query's deep structures through the link_attribute situated in the outer relation's deep structure.

Class 9 Outer query: relation Inner query: same relation

In this class, the relation can only be an entity or relationship, but the link_attribute cannot be a primary key.

1) Find the label relating the link_attribute to the relation in the network graph.
2) If the link_attribute is in the outer query's select

clause, it is selected as Object and the word "meme" is added to the link_attribute in the outer relation's deep structure. Otherwise it is selected as Pp and the word "meme" is added to the link_attribute at that position in the outer relation's deep structure.

3) The inner query's deep structure precedes the outer query's deep structure. The outer query's deep structure has the word meme in front of its link_attribute.

Example:

```
select etudnom,etudid
from etudiants
where progrnom
   in( select progrnom
       from etudiants
       where etudnom = 'Lyne Lahaye');
```

Etudiants network graph is

```
schema(etudiants,id,[etudid,etudnom]).
schema(etudiants,etre,[sexe,progrnom]).
```

The outer's deep structure is :

```
[etudiants,entite, [etudnom,[Lyne Lahaye,Jasmine Blanchet]],
[pp, [[etre, [[=,progrnom,no_value]]]]], [[id,[[etudid,
[1310151, 2222222]]]]]]
```

The inner query's structure is

```
[etudiants, entite, [etudnom, [Lyne Lahaye]], [pp, [[etre,
[[progrnom, [no_value]]]]]]
```

The verb accompaning 'progrnom' is etre. The place of progrnom in the outer query is Pp. The word "meme" is added to progrnom in the outer query's structure. The sequence of structures is inner query followed by outer query. The final result is :

[[etudiants, entite, [etudnom, [Lyne Lahaye]], [pp, [[etre, [[progrnom, [no_value]]]]]], [etudiants,entite, [etudnom,[Lyne Lahaye,Jasmine Blanchet]], [pp, [[etre, [[meme, [=,progrnom,no_value]]]]]],[id,[[etudid, [1310151, 2222222]]]]]]].

The generated output is :
<u>L'etudiant Lyne Lahaye est dans un programme. L'etudiant Lyne Lahaye est dans le meme programme. Jasmine Blanchet est dans le meme programme.</u>

Class 10 Outer query: relation Inner query: relation

This class does not constrain the relation in the inner query to be the same as in the outer query. The possible choices of relations are entity to entity, relationship to relationship, and one entity to relationship. The link_attribute joining two levels must be a primary key for at least one relation. Knowing the structure of the two relations, we apply the algorithms defined for one level queries with multiple relations with certain modifications. The modifications to the one level algorithm are as follows: first, when the link_attribute is the primary key in the relation and an anaphoric is added to it, a test is performed on the link_attribute to determine if the word anaphoric has already been attached to it in order to avoid repetition. Secondly, if there is a second level, and the link_attribute is also in the outer query's select clause, then we replace the values of the link_attribute of the inner relation's structure by the values of the link_attribute of the outer relation's structure. The word anaphoric may also be added to the link_attribute. However, the information about the values of the link_attributes will help determine whether the anaphoric reference is plural or singular.

<u>Example:</u>

```
select offreid, etudid
from prendre
where offreid in (
   select offreid
   from horaire
   where jour = 'MAR');
```

Network graphs needed are

```
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(horaire,id,[offreid]).
schema(horaire,etre.[nolocal,jour]).
```

The class 2 algorithm is applied to each relationship giving:

```
[prendre,relation, [etudid,[1310151, 2334454]], [],
[[prendre, [[offreid, [COMP771287AA, COMP771287AA]]]]]]
```

[horaire, relation, [offreid, [no_value]], [pp,[[etre, [[=, jour, MAR]]]]], []].

We apply the class 6 algorithm to perform the joining of the two structures with two modifications as follows: add anaphoric to the link_attribute that has not already been marked anaphoric, and change the "no_value" of the link_attribute in the inner query by the values of the same link_attribute in the outer query. The result is :

[[prendre,relation, [etudid,[1310151, 2334454]], [], [[prendre, [[offreid, [COMP771287AA, COMP771287AA]]]]]], [horaire, relation, [**anaphoric**,[offreid, [**COMP771287AA, COMP771287AA**]]], [pp,[[etre, [[=, jour, MAR]]]]], []]].

Anaphoric is the modification done by class 6. Class 10 algorithm changes the "no_value" to COMP771287AA and COMP771287AA.

The surface structure is :
<u>L'etudiant avec code 1310151 prend le cours COMP771287AA. Ce cours est le Mardi. L'etudiant avec code 2334454 prend le cours COMP771287AA. Ce cours est le Mardi.</u>

Class 11 Outer query: one relation
Inner query: multiple relations

In this class, the relation in the outer query can be an entity, a relationship, or a RR relation. The inner query can have a mixture of entity to entity, entity to relationship, or relationship to relationship relations. An RR relation cannot occur at the second level, as the two entity relations would have to be processed before doing the join of the two levels.

1) Find the relation in the second level which has a relationship with the relation of the outer query. This selected relation in the second level, and the relation in the outer level are processed using the algorithm for one level queries with multiple relations, with the modifications mentionned for class 10. 2) The other relations in the second level with their deep structures, are processed sequentially with their related, second level relations. In other words, a second level relation is selected depending on its relationship to the previously selected second level relation. This process is repeated until there are no more unprocessed relations left in the second level query.

Example:

```
select etudnom, telephone
from etudiants
where etudid in (
   select etudid
   from prendre,horaire
   where prendre.offreid = horeire.offreid and offreid =
   COMP771287AA and jour = MAR);
```

Network graphs required are:

```
schema(etudiants,id,[etudnom,etudid]).
schema(etudiants,avoir,[telephone, bureau]).
schema(prendre,id,[etudid]).
schema(prendre,prendre,[offreid]).
schema(horaire,id,[offreid]).
schema(horaire,etre,[nolocal, jour]).
```

The class 1 to 3 algorithms are applied to each relation in the SQL query.

[etudiants, entite, [etudnom,[Lyne Lahaye, Regis Cardin]], [pp, [[id, [[=,etudid, no_value]]]]], [[avoir, [[telephone, [3425816, 6775664]]]]]]

[prendre,relation, [etudid, [no_value]], [pp, [[prendre, [[=, offreid, COMP771287AA]]]]], []]

[horaire, relation, [offreid, [COMP771287AA]], [pp, [[etre, [[=, jour, MAR]]]]], []]

1) Prendre has a common key "etudid" with etudiants. The class 10 algorithm is applied to join these two structures giving:

[[etudiants, entite, [etudnom,[Lyne Lahaye, Regis Cardin]], [pp, [[id, [[=,etudid, no_value]]]]], [[avoir, [[telephone, [3425816, 6775664]]]]]], [prendre,relation, [anaphoric, [etudid, [Lyne Lahaye, Regis Cardin]]], [pp, [[prendre, [[=, offreid, COMP771287AA]]]]], []]].

A matching of horaire's structure is still to be done. It has a common key which is "offreid" with prendre. We join the new structure of prendre with horaire's structure by applying the class 10 algorithm. The result is:

```
[[etudiants, entite, [etudnom,[Lyne Lahaye, Regis Cardin]],
[pp, [[id, [[=,etudid, no_value]]]]], [[avoir, [[telephone,
[3425816, 6775664]]]]]], [prendre,relation, [anaphoric,
[etudid, [Lyne Lahaye, Regis Cardin]]], [pp, [[prendre, [[=,
offreid, COMP771287AA]]]]], []], [horaire, relation,
[anaphoric, [offreid, [COMP771287AA]], [pp, [[etre, [[=,
jour, MAR]]]]], []]]
```

The generated surface structure is :

<u>L'etudiant Lyne Lahaye a un code et a le telephone 3425816. Il prend le cours COMP771287AA. Ce cours est le Mardi. L'etudiant Regis Crdin a un code et a le telephone 6775664. Il prend le cours COMP771287AA.</u>

This basic component described above produces the deep structures for each relation in the SQL query. The following component, called the transformational component, will take care of the syntax, insert the required prepositions and articles, and conjugate the verbs, nouns, and articles, etc, by successively applying transformational rules.

CHAPTER IV

Surface structure generation

4.0 Tranformational component

The transformational component applies rules to the deep structure in order to create the surface structure. The transformational rules have two components: the structural description (SD) and the structural change (SC). The structural description lists the elements in the deep structure that will be transformed according to the transformational rules. The structural change represents the changes made to the structural description by the rules. These rules control the sentence's syntax, the insertion of prepositions and articles, and the conjugation of verbs, articles, nouns, etc.

A. The following rules refer to the repetition of a sentence.

```
1) SD: [R,  Type,  [Attr_name, Values],  PP,  [[V,Obj1]]]
        X1   X2         X3 (1,2,..N) X    X5       X6
   SC:  0    0      X3(plural) 1,2..et N X4 et   X5
```

The names X1, X2 etc. denote objects of that type and 0 denotes deletion under the structural change. group of elements in the deep structure. This operation groups all the values of the subject together to form a single sentence. The attr_name is put into its plural form. Obj1 represents a single attribute with one value. R and Type are deleted.

Example:

```
[travailler,   rr,   [profnom,   [Shinghal,Radh]],   [],
     X1         X2        X3         X41      X42       X5

[[travailler, [[deptid,[INFO]]]]]]
               X6
```

```
Les professeurs Shinghal et Radh travaillent au departement
X3              X41        X42                _ X6
INFO.
```

```
2) SD: [R, Type, Subjectl, Pp, [[V,[[Attr_name,Values]]]]]
        X1  X2     X3       X4   X5     X6       (1,2,..N)  N>1
   SC:  0   0      X3       X4 et X5    X6(plural) 1,2,.. et N
```

This rule generates a sentence having one subject and multiple value objects that are enumerated.

Example:

[prendre, relation, [etudid,[1310151]], [],[[prendre,[[offreid, [COMP1, COMP2]]]]]]

L'etudiant avec code 1310151 prend les cours COMP1 et COMP2.

```
3) SD: [R, Type, Subject,  Pp,  Obj]
        X1  X2    X3i to N X4   X5i to N    where N >= 1
   SC:  0   0     X31      X4   X51
        0   0     X32      X4   X52
        ----------------------------
        0   0     X3N      X4   X5N
```

This rule specifies that this sentence is repeated for each different value until all values in the list have been processed.

Example:

[prendre, relation, [etudid, [1,2,3]], [],[[avoir, [[note, [A,B,C]]]], [prendre, [[COMP1, COMP2, COMP3]]]]]

L'etudiant avec code 1 a note A et prnd le cours COMP1.
L'etudiant avec code2 a note B et prend le cours COMP2.
L'etudiant avec code 3 a note B et prend le cours COMP3.

B. The following group of transformational rules refers to substitution and conjugation of words that are the subject of the sentence.

```
1) SD: [[ certains, attr_name],  Pp, Obj]
          X1          X2           X3   X4
   SC:    X1(G,plur) X2(G,plur)  X3   X4
```

The word "certains" is in the plural form with the same gender as attr_name. Attr_name is also plural.

Example:

[prendre, relation, [certains, etudid], [], [[prendre, [[offreid,[COMP1, COMP2]]]]]]

Certains etudiants avec code prennent le cours COMP1.
Certains etudiants prennent le cours COMP2.

```
2) SD: [[R,                  Type],                 Pp, Obj]
        X1                    X2                    X3  X4
   SC:Art indef.(G,singular) id of X1(G,singular) X3  X4
```

This rule substitutes [R,Type] with an indefinite article and the noun which expresses the id of the relation R. The noun has a gender and must be singular. The indefinite article must be singular and agree with the gender of the noun.

Example:

[prof, entite, [prof, entite], Pp, Obj]

Un professeur ...

```
3) SD: [ ..[attr_name,Values],   Pp,   Obj1]
               X1          X2i        X3      X4   where i = 1 to N
   SC: Art def.(G,sing) X1(G,sing,A_V,subject) X2i X3 X4
or
   SC: Art def.(G,sing)  X1(G,sing,V_A,subject) X2i X3 X4
```

This rule inserts the definite article that agrees with the gender of X1. The attribute name is inserted in front of the value because of its specification A_V, and has a specific noun as subject. The i'th value is inserted for each i'th occurrence of the sentence.

Example: Case of A_V(Attribute followed by Value)

[.. [etudnom, [Lyne Lahaye]], Pp, Obj]

L'etudiant Lyne Lahaye ..

```
4) SD: [..[anaphoric,Subject], Pp, Obj]
              X1          X2     X3   X4
   SC:    0  Ce(G,P) X2(inanimate,G,P) X3 X4
   or
   SC:   0 Pronoun(masculin,P) 0 X3 X4
```

This rule adds "ce" or "ces" if the subject is inanimate, or a pronoun if the subject is animate. The gender of the pronoun is always masculin even if it should be feminine. It would be necessary to perform another database access to extract that person's sex, or maintain a dictionary of first names for females to determine the correct gender. As this would result in inefficient processing time, it is not implemented. This problem does not exist with inanimate nouns.

Example:

[.. [anaphoric, [offreid, [COMP]]] ...]

Ce cours ...

C. The following transformational rules refer to the Pp component of the deep structure.

```
1) SD: [pp,[[Verb, [=,At_name, Val], or,[=,At_name,Va]]]]
           X1    X2     X3    X4        X5    X6  X7    X8      X9

   SC:  0   X2(Pl)  0{Prep(V,Att,P),{art(Prep,Type)}}
                       X4(G,P,object,A_V)
                                    X5     ou   0     0      X9

   or

   SC:  0   X2(Pl)  0 artdef(G,P) X4(G,P,objet,A_V)
                                    X5     ou                X9
```

This rule conjugates the verb to the plural of the subject. A preposition and article are inserted in front of the attribute name if so required by the verb. The verb and the attribute name determine the article's type. The preposition has the same plural form of the objet. The position of the attribute name is determined by A_V or V_A. The "{" and "}" brackets indicate the possibility that certain elements in the sentence depend on the satisfaction of certain conditions in the where clause of the query.

Example:

```
[pp, [[avoir, [=, note, []], or, [=,note, B]]]]]
If Pl is singular, then:
 a la note en progres ou B

2) SD: .. [Verb, [[ Oper, Att,  Value]] ..
            X1         X2   X3    X4
   SC:    X1(P)        X2   X3(G,Pl,objet,A_V) X4
or
   SC:    X1(P)        X2   X4  X3(G,Pl,Objet,V_A)
```

This rule is applied in the general case when the operator in the condition is not an equal sign: "=". It also handles quantity. No preposition is inserted between the verb and attribute name, as our vocabulary does not require it.

Example:

```
 [avoir, [[>=, inscrit,25]]...
   a plus de 25 etudiants inscrits

3) SD: .. [Oper, capacite,  Value]..
            X1        X2        X3
   SC: art indef(G,P) X2(G,P) de X1(G,P) X3  X1 different of =
   or
   SC: art indef(G,P) X2(G,P) de  X3         X1 = "="
```

This rule is an exception due to our vocabulary requirements. The word "capacite" requires an indefinite article along with the word "de".

Example:

```
[ >, capacite, 25]..
```

une capacite de plus de 25

```
4) SD: ..[=, Att_name,  no_value]..
          X1   X2          X3

   SC:     0 {Prep(V,Att)}artidef(G,Punique) X2(G,Punique)0
```

This rule inserts an indefinite article before the object. The number (singular or plural), is determined by the relation between the attribute name and the relation name. For instance, a professor works in only one department. The relation name is travailler. In this case the attribute "department" requires a singular form. A preposition may be necessary between the verb and the attribute name.

Example:

```
[= deptid, no_value] ..
```

un departement ..

```
5) SD: ..  [[meme, [Oper, Att_name, Value]]..
               X1      X2     X3        X4

   SC:artdef(G,P) X1(P)0 X3(G,P,objet) 0
```

The word "meme" agrees with the plural of the attribute.

Example:

```
[[meme, [=, progrnom, Bacc]]] ..
```

le meme programme ..

```
6) SD: .. [[anaphoric, [Oper,  Att_name, Value]]]..
                X1       X2       X3       X4
  SC:           0 ce(P)  0  X3(P,A_V,objet) 0
```

This rule asserts "ce" or "ces", because the attribute name and its value have been referred previously.

<u>Example:</u>

[anaphoric, [=, offreid, COMP1]]

<u>.. ce cours ..</u>

```
7) SD: .. [pp, [V,  Cond,..]..]..
             X1  X2    X3
   SC:        0    X2(P) {Prep(V,att of Cond) art(Prep,Type)}
                                X3
```

This rule analyses the verb and condition in order to insert a preposition and article between the verb and condition. The condition can be analyzed further.

<u>Example:</u>

[pp, [etreinscrit, [=,deptid, INFO]]

<u>est inscrit au departement INFO</u>

Rules related to generating other components of the sentence such as the object component are not mentioned here. The object component is analyzed the same way as the Pp component except no operators are involved.

French sentences are then generated by applying successive rules to the deep structure. The rules perform lexical insertions of prepositions, articles,and words, control the syntax of the sentences, and delete certain components. The transformational component also controls the sequence in which sentences are produced, i.e., a set of sentences is repeated for each tuple of values when necessary. Marakakis [7] and Chomsky accomplish lexical entires as part of the basic component, but we find it to be easier to insert lexical entries with the transformational component.

4.1 Model of a reader

In an interactive session wherein a sequence of queries are posed by the user, it is possible to develop a model of the user. It can be constructed from the knowledge of what is being asked, what has been told, what the other believes, and what the reader's attention is currently on. However in our present work this model is not employed in full detail.

What is being asked is contained in the attributes of the select clause of the input SQL query. A study shows that 80% of queries to a database correspond to the structure of the database. The NL-menu approach implicitly displays the database's structure. As this information about the structure is evident from the menus, the reader can easily understand it. Knowledge about what has been told is therefore not considered. Knowledge about the beliefs of others is not useful in this application. The questions do not refer to abstract concepts such as philosophy or politics, among others. They refer to records of students, professors, departments, etc. Therefore, the possible questions are straightforward. Other knowledge about what is in the reader's attention in our case corresponds to the reader's structures. This knowledge is useful when the reader misunderstands some parts of the database. However, to recognize the misunderstanding, we need some information of what has been told. As we do not store information pertaining to a previously asked query, it is not possible to determine what the user does not understand.

4.2 Model of discourse

This model is useful whenever multiple sentences are generated. It could help to organize the sequence of sentences into a consistent and well readable text. At present, in my implementation, transformational rules at the level of deep structures determine how sentences are placed in a sequence.

The following rules are applied to the group of sentences each of which has the following basic form: Subject, Verb, Object. Object can be a series of objects (1,2,3). The sentences can be compound sentences having the following form : Subject, Verb1, Objects, and Verb2, Objects.

<u>Rule 1</u>

Input :

```
   S1 = Subject1  V1   O1,..
   S2 = Subject1  V2   O2
```

The sentences S1 and S2 have the same subject. The order is immaterial but the subject in the second sentence is transformed into an appropriate pronoun.

<u>Rule 2</u>

Input:

S1 = Subject1	V1	O1, Obj
S2 = O1	V2	Objs
S3 = SUbject1	V3	Objets

The object of sentence S1 is the subject of the sentence S2. Therefore, S2 follows S1. S3 is not positioned immediatly after S1 as it would cut the flow of ideas. In the following sequence: S1,S3,S2, S2 is not related to S3. S2 would appear out of context if placed immediately after S3.

<u>Rule 3</u>

Input:

S1 = Subject1	V1	Obj1
S2 = Subject2	V1	meme Obj1

The sentence S1 precedes S2 due to the reference made in S2 to the common object "Obj1".

These rules of sentence ordering, generally improve the readability of a multiple sentence response. But further improvements can be achieved by changing the structure of some of the sentences.For example, additional clauses can be introduced in lieu of a preceeding or a following sentence.

4.3 Data Dictionary

The designer must provide a data dictionary that will be consulted by the generation process for different purposes. The data dictionary in this application contains information about entities, relationships, attributes names, values of the attributes, and some linguistic information which is independent of the database domain. A brief description of what the data dictionary contains is given below with examples.

1. Each relation has its type : entity or relationship.

   schema: relations(type, rel_name)

   <u>example:</u>relations(entite,etudiants)

2. Each relation is associated with its list of attribute names.

   schema: "rel_name"(list of attributes)

   <u>example:</u> prendre([etudid,offreid, note])

3. Abbreviated values have their corresponding complete textual values.

   schema: value_attr(attr_name,value,compl_val)

   <u>example:</u>value_attr(note,[],en cours)

4. Information about conjugated attribute.

   schema: "attr_name"(conj_att,place,gender,plural)

   <u>example:</u> titre(cours,sujet,masculin, singulier)

   titre(titre,objet,masculin,singulier).

An attribute may occupy the subject or object position in a sentence. Some attribute names require a different noun for different positions. For example, the attribute name "titre" has different nouns in the following different positions:

1) Le cours COMP671 a titre intro. aux syst mes experts.
"titre" occupies the position objet.
2) Le cours intro. aux syst mes experts vaut 3 cr dits.
"titre" occupies the position subject, and is omitted.

5. Information about the position of the value associated with the attribute.

schema: v_a(attr_name, v_a)

example:v_a(pt_etud,v_a).

v_a = value followed by the attribute

a_v = attribute followed by the value

6. Information about each conjugated verb.

schema: verbco(verb,tense,3rd,plural,conj_verb)

example:verbco(avoir,indicatifpresent,3rd,singulier,a)

In certain sentences, the passive voice is used. When using the passive voice the French language requires that the gender and number of the subject correspond to the gender and number of the verb. One problem in our application is that if the sentence refers to a person , sex is not available. Therefore, we assume the gender is masculine, and conjugate accordingly.

7. Translation of an operator in natural language.

schema: ope(operator, value).

example:ope(>, plus de)

8. Few operators require conjugation.

schema: conj_ope(gender,plural,oper,conj_oper)

example:conj_ope(feminin,singulier,<=, inferieure ou egale a)

9. Information about prepositions and articles required after a verb.

schema: preposition(verb, attr_name,preposition,article)

example:preposition(avoir,pr sident,comme,no)

10. Information about the mapping.

schema: unique(rel_name,att_name,plur_subj, plur_obj)

example:unique(prendre,offreid,singulier,pluriel)

unique(travailler,deptid,pluriel,singulier)

This predicate refers to one to one mapping or one to many mapping. The one to one mapping is explained this way: professors can only work in one department. The one to many mapping refers to a student taking many courses. The predicate "unique" is used by the grammar in the case of an indefinite object.

The following linguistic information is provided.

11. Conjugated prepositions.

schema: prep(preposition,plural,conj_prep)

example:prep(au, pluriel, aux)

12. Conjugated pronouns

schema: pronoun(gender,plural,pronoun)

example:pronoun(masculin,singulier,il)

13. Conjugated definite articles.

schema: artdefini(article,gender,plural)

example:artdefini(le, masculin,singulier)

14. Conjugated indefinite articles.

schema: artindefini(article,gender,plural)

example:artindefini(une,feminin,singulier)

15. Conjugated operators.

schema: conj_ope(gender,plural,operator,conj_ope)

example:conj_ope(feminin,singulier,<=,inferieure ou egale a)

The data dictionary employed by this application is different from the usual data dictionaries. Data dictionaries in general are used to support database design, whereas we use it to contain meta data that helps in the sentence generation process. A complete view of the generation process is shown below through two examples.

4.4 Two complete examples

Complete Example 1

The following example demonstrates how the design components generate French sentences. First, a database [illegible]ch in response to the SQL query finds data values. This is stored in a record. Then the system parses the input query and extracts the attributes from the "select" clause, the relation names from the "from" clause, and the conditions from the "where" clause. The attributes, values and conditions are grouped with their respective relations by consulting the data dictionary. At this point, if a composite relation such as "travailler", "offrir", "enseigner", or "etreinscrit" is recognized, the system replaces it by its two component entities. Based on this structure of the query, its class (Class1 through class 11 described earlier) is determined. Then appropriate algorithms of that class are applied in order to generate the deep structure.

Example: SQL query:
```
select offreid, jour
from horaire
where offreid in
  (select offreid
   from prendre
   where etudid = '2222222')
```

The example is a two level query consisting of one outer query and one inner query, each with one relation. Therefore, the class 10 algorithm is employed. The generator also applies the class 2 algorithm which handles the relationships between "horaire and prendre" relations. The following deep structures are produced by the class 2 algorithm.

```
[horaire, relation, [offreid, [COMP771287AA, COMP772287AA,
COMP244285T]], [], [[etre, [[jour, [MAR, MER, VEN]]]]]]

[prendre, relation, [etudid,['2222222']], [pp,[]],
[[prendre, [[offreid,[no_value]]]]]]
```

The first relation horaire has "offreid" as subject, and "jour" as object. Offreid is the chosen subject as it is the primary key in the horaire relationship network graph. The object of the generated sentence is derived from the remainding attributes in the select clause. The object's verb corresponds to the labelled arc between "jour and horaire". The condition offreid is deleted from the horaire structure. The second relationship works in the same way by mapping the relationship network graph to the relation, and forming the subject, Object and pp components of the deep structure.

The next step is the join of the two relationships network graphs to find the link_attribute, which in this case is offreid. Knowing that the primary key of horaire is offreid and that this is not the primary key for prendre, the sentence sequence produced is prendre followed by horaire. After establishing this sequence, the position of the link_attribute in prendre is found in the position object component, and the actual values of the no_value attribute, COMP771287AA, COMP772287AA, COMP244285T, are substituted. Finally, the word "anaphoric" is added to the link_attribute in the horaire structure. The deep structures look like this:

[[prendre, relation, [etudid,['2222222']], [pop,[]], [[prendre,[[offreid, [COMP771287AA, COMP772287AA, COMP244285T]]]]]], [horaire, relation, [anaphoric, [offreid,[COMP771287AA, COMP772287AA, COMP244285AA]]], [], [[etre, [[jour, [MAR, MER, VEN]]]]]]]

The next step is to apply the transformational rules that produce French sentences.

L'_etudiant_avec_code_2222222_prend_le_cours COMP771287AA. Ce cours est le Mardi. Il prend le cours COMP772287AA. Ce cours est le Mercredi. Il prend le cours COMP244285T. Ce cours est le Vendredi.

There are some undesired repetitions in this sequence of sentences. However, the sentences convey the information unambiguously.

Complete Example 2

Example: SQL query:
```
select nolocal, capacite, offreid
from locaux, horaire
where horaire.nolocal = locaux.mnolocal
```

This query involves multiple relations consisting of one entity and one relationship. The class 5 algorithm handles this type of query. The deep structures for each relation is:

[locaux, entite, [nolocal, [H-843, H-539-2, H-843]], [pp, []], [[avoir, [[capacite, [25, 20, 25]]]]]]

[horaire, relation, [offreid, [COMP771287AA, COMP772287AA, COMP67128[illegible]]], [], [[etre,[[nolocal, [H-843, H-539-2, H-843]]]]]]

The next step is to find the common key to both relations which is "nolocal". After finding the common key, the word "anaphoric" is added to the relation horaire that follows the local's deep structure. The result is the following.

```
[[locaux, entite, [nolocal, [H-843, H-539-2, H-843]], [pp,
[]], [[avoir, [[capacite, [25,20,25]]]]]], [horaire,
relation, [offreid, [COMP771287AA, COMP772287AA,
COMP671287AA]], [], [[etre, [[anaphoric, [nolocal, [H-843,
H-539-2, H-843]]]]]]]]]
```

These deep structures are transformed into surface structures by the transformational rules. The generated sentences are:

Le local H-843 a une capacite de 25. Le cours COMP771287AA est a ce local. Le local H-539-2 a une capacite de 20. Le cours COMP772287AA est a ce local. Le local H-843 a une capacite de 25. Le cours COMP671287AA est a ce local.

These two examples give a brief overview of how a group of sentences is generated.

4.5 Problems specific to French language

The French language is different from the English language in several ways. In French, the gender and number of the noun should agree with the conjugation of the verb, preposition, definite and indefinite articles, or adjectives. This prototype handles these cases. The gender can be determined from the data dictionary or from the data value in the database. If there is a series of nouns with different genders, the masculine gender is automatically assumed.

There is no neuter nouns in French. The English has three genders: masculine gender describing a man, feminine gender describing a woman, and a neuter gender for all other cases. The article has a gender in French such as feminine and masculine. The gender of the article should agree with the gender of the noun being qualified. There are three types of article in French : definite article (le, la, les), indefinite article(un, une), and partitive article (du, de la , des). The definite article l' is a special case for French. If the noun starts with a vowel, then l' is used instead of le or la. However, the English language has two types of articles: definite article (the), and indefinite article (a, an). The rules for incorporating articles in English differ from French.

Another difference is that the adjective in French has to agree with the noun that it qualifies.

The French and English pronoun have gender, number and person. There is an exception witn the relative pronoun in English, such as that, that, whose, who, whom. These relative pronouns appear in textual form as they are in English whereas the French relative pronouns need to be conjugated (laquelle, lesquelles). French regular verbs are generally grouped in four classes ending in er, ir, oir, and re. Compound tenses ("passe compose, plus-que-parfait, passe anterieur, futur anterieur, conditionel passe, subjonctif passe, subjonctif plus-que-parfait") are conjugated with the auxiliary avoir and the past participle, except reflexive verbs and the most usual intransitive verbs (like "aller, arriver", etc.) which are conjugated with etre. The example of a compound tense is "J'ai recu un cheque.". "ai" is the auxiliary avoir, and the past participle is recu. The tense of the verb "recevoir" is passe compose (compound tense). The French past participle always agree with the noun to which it is either an attribute or an adjective.

<u>Example:</u>

The boy was punished, "Le garcon fut puni".

The French past participle agrees with the object of a verb conjugated with avoir, only when the object comes before it.

The English regular verbs have only three endings: "s" for the third person singular of simple present, "ed" for the simple past and past participle (always invariable), "ing" for the present participle. Its tenses are much easier to implement than the French tenses where we have to find which group of verbs the verb belongs to.

4.6 Limitations and possible improvements

This text generation system handles only certain types of two level queries. Each level can have more than one relation. The "where" clause has a maximum of one "or" connective. This limitation is more due to program design than the system design.

The tense of the verbs are assumed to be simple present (present). The past tense is necessary if a student has taken a course and has a grade. The implementation considers only simple present for all cases. However, modifying the program to integrate the past tense for this case, is possible.

The data dictionary contains the singular and plural for each noun. The memory space taken by the dictionary can be reduced by storing the singular noun and yes or no for regular noun. An algortihm will add "s" to the singular noun for regular noun. Irregular nouns can be checked for their plural in the dictionary. Our data dictionary does not have a lot of nouns. Therefore, we can keep the plural and singular for each noun. This method is very useful with a large data dictionary.

The generated output consists of simple sentences or compound sentences. No relative clauses are used. A possible improvement can be to apply new tranformational rules to the deep structures building relative clauses. The relative clause transformational rule can be employed if one sentence has an object which is the subject of the following sentence. The relative clause consists of "qui" followed by the verbs of the following sentence and the objects. Another improvement to simple sentences is to group some types of sentences, such as having the same subject in one sentence or the following type. Lyne Lahaye a une identification, Il prend le cours COMP771287AA. Regis Cardin a une identification, Il prend le cours COMP771287AA. The result would be Lyne Lahaye et Regis Cardin ont une identification et prennent le cours COMP771287AA. The course COMP771287AA is the common object for each value of the subject. Therefore, by post processing of the sentences, we can group the sentences in this fashion to improve the quality of the text.

There is monotony when same type of sentences is displayed to the user. Our implementation uses only active voice. Using passive voice, other structures of sentences, or synonyms avoid monotony of the answers. New transformational rules and a dictionary of synonyms can be developed to solve that problem.

Another limitation is the repetitive use of the same verb. To improve the quality of the sentence, a transformational rule can check the repeated use of a verb and modify appropriately.

For a given query, when the database search fails, the four following cases are possible.

1) Generate a stereo type answer. For instance, "The

database does not contain information about your question."
2) Generate the deep structure if possible and an appropriate surface structure for it, explaining the database search failure.
3) Confirm partial information stated in the query. For instance, the user asks about which courses is Lyne Lahaye taking. There could be no Lyne Lahaye or no courses taken by her. The appropriate answers for these two cases is: Lyne Lahaye is not a student at Concordia, or Lyne Lahaye does not take any courses at Concordia but is registered.
4) Direct the user to some other source of information possible.

A future implementation of this project is to develop a bilingual system (French, English). Some changes are necessary to achieve this goal. The possible changes are the modifications of transformational rules. There will be no conjugation of articles, prepositions, past participles, and adjectives. Some new structures of sentences can be added. The position of the value for each attribute can be different for English language. The semantic structures remain the same. Therefore, no modification is done to the basic component of the grammar, only at the transformational component. A conversion of the meaning of data contents of the database to English words is to be performed. The entity model remains the same. These modifications will allow us to have a bilingual text generation system.

CHAPTER V

INTEGRATION

5.0 Introduction

The different modules of the software system for voice interface to access database are integrated using two personal computers and the PC Quick Net interconnecting them. The interaction between the two PCs and which software goes on which machine are described in this chapter. Natural Language input based on Natural Language menu and speech recognition, SQL query generation, Natural Language generation in French, and its output in spoken form are handled by the two PCs together. The full implementation of the network and the modules are described.

5.1 Environment

The 640K active memory of one AT was not enough to run the complete software system. Using two ATs and the PC Quick Net network, we were able to meet the resource needs of the various modules. PC Quick Net is a combination of hardware and software that allows communication between the two PCs. It allows a group of PC users to share data, programs and peripherals. It joins up to six computers via their RS232 serial ports. It consists of modular RS232-RJ11 (Telephone cable) connectors and standard flat telephone cable. The network controller hardware is contained inside the RS232-

RJ11 connectors. PC Quick Net does not require any special communications board. It runs in background and uses MSDOS commands. The software consists of the following files.

devnet.sys	(device driver)
setnet.com	(network configuration program)
netacc.com	(access tool)
readme.doc	(information about Quick Net)
lantest.com	(test utility for the network system)
install.exe	(install program)
install.lst	(list of incompatiobilities)
inmac.exe	(creates the install banner)
filelock.doc	(information on multiuser applications)

This software is easy to install by simply using the command install. Following the command install, one must activate the network on the PC through the setnet command which assigns and reassigns virtual drives and printers. This command is also used for initialization. To access remote programs, we use the drive letter assigned to the drive of the remote PC. PC Quick Net maintains data integrity by allowing many computers to read a disk, but only one computer to write on the disk. This product is simple to install, cheap, but it is slow.

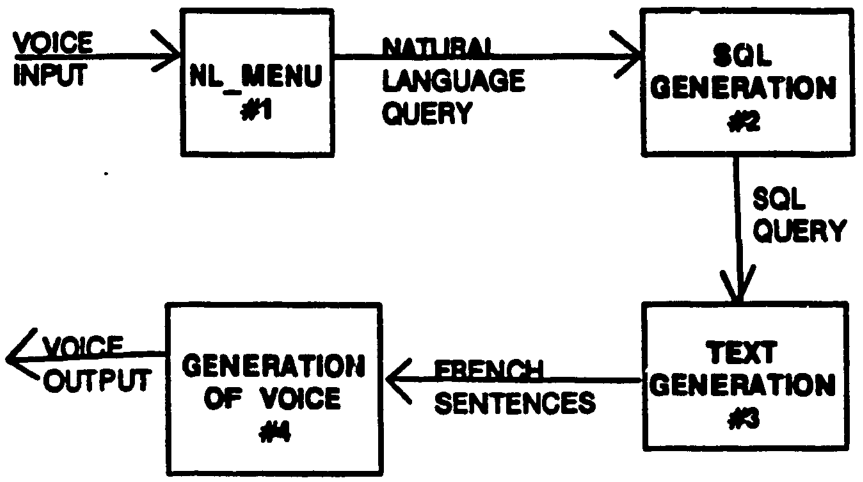
Each PC AT has an assigned virtual drive which corresponds to a real drive of the other machine. References to virtual draive access the remote drive. The two PC ATs share two files in our application. One machine finishes writing on one file and then transfers it to the other machine. The receiving machine can start to read the file. There is a problem of timing if the receiver starts reading the file while the sender has not finished sending. We alleviate this problem by locking the file by using the extension lck while writing. After the writing, the file is renamed with the extension "ari" which enables reading by the other machine. The communication is done by sending the file to the other machine. The voice command "fin" from the user will close all the files on both machines and terminate the session.

One of the two machines called Challenger, contains Dragon system (speech recognition system), Televox system (Speech synthesizer system), Arity Prolog, Microsoft C, and Flush software. These hardware and software have been described earlier, except Flush. Flush is a program that flushes the contents of the buffer used by Televox. The buffer end is marked by "flmk". The other machine called Columbia, contains Arity Prolog and Arity SQL. All these software are used by different modules.

5.2 Communication between modules

The communication between these modules is shown in figure 7. The "NL_menu module" receives as input a voice command. The user sees a NL_menu screen with highlighted words which he can select. When a sentence is completed, the user says "procede" in order to process the query, or "fin" to terminate the session. This module is written in Arity Prolog and Microsoft C, and requires Dragon System in background. The second module, SQL query generation receives the output of the first module, and translates it to an SQL query. This module is written in Arity Prolog exclusively. The text generation module takes the SQL query, accesses the database to receive an answer to that query. It then translates the "semantic structures" into a sequence of French sentences. This module requires Arity/SQL and is written in Prolog. The output, French sentences, are passed to a program called "parler" that generates spoken output through Televox, the speech synthesizer. This data flow is repeated for each SQL query from the user. However, these modules cannot run on one PC AT due to lack of memory.

Figure 7. Dataflow of the modules



	Development	Software Tools
#1	In-house developed by earlier thesis	Arity Prolog Voicescribe (isolated word speech recognition) Microsoft C
#2	Designed and developed by this thesis	Arity Prolog
#3	Designed and developed by this thesis	Arity Prolog
#4	Off the shelf system	Arity Prolog Televox (French Speech Synthesizer)

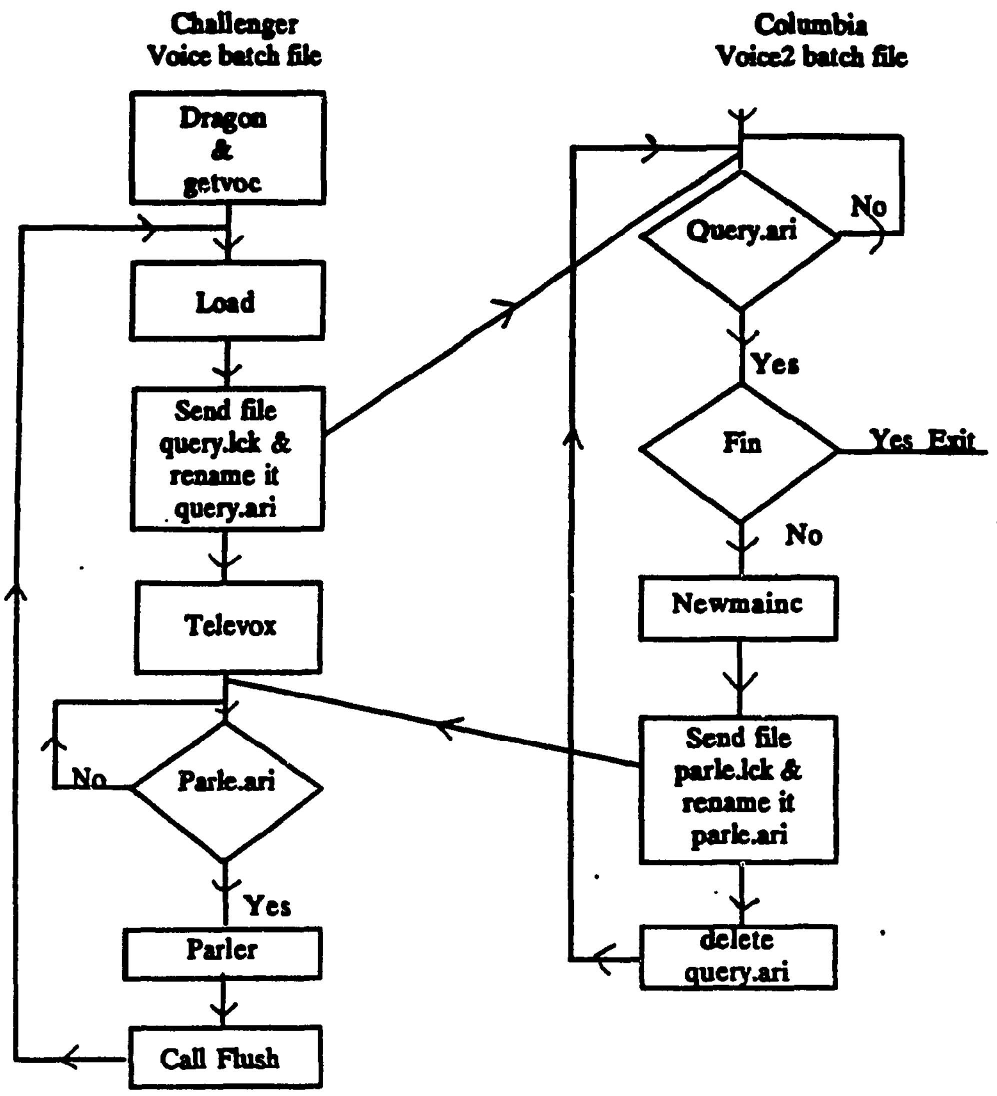
5.3 Complete integration of the modules

Two PCs AT are interconnected through a network called PC Quick Net that allows us to manage and meet the needs of the modules. The integration is achieved through the network using two batch files which transfers shared files between the machines. The boot-up procedure on each machine activates PC Quick Net automatically. The user starts up the batch file called "voice" on Challenger, and "voice2" on Columbia. The batch file on Challenger loads Dragon, and asks the user the name of its vocabulary file. It then executes the program "Load" which displays the NL-menu screen and responds to the voice command. When the user ends his sentence by "procede", the module "SQL generation" is invoked. It generates the SQL query and stores it in the file "query". In the case of a termination of the session, the word "fin" is stored in the file "query" in order to warn the other machine to stop the session. The batch file "voice" then sends the file "query" with the extension lck to Columbia. It then renames it with the extension ari, and deletes it from its own drive. Challenger checks if that file contains the word "fin" to proceed to the end of this session. If it is not the end of a session, Challenger installs the marker in memory by calling "flmk". It loads the synthesizer Televox with the command "tlvx". Challenger stays in a waiting loop until the file "parle.ari" is sent from Columbia. When Challenger receives the file "parle.ari", it runs the program "parler" which generates the voice output. When the program "parler" ends, the batch file "voice" deletes the file "parle.ari", flushes Televox out of the memory, and returns back to the beginning of the loop waiting to be called again. The loop ends when the word "fin" is written in the file "query.ari".

The "voice2" batch file starts by executing a waiting loop. It waits until it receives the file "query.ari" from the Challenger machine. It checks if the file contains the word "fin" in order to end the session. If this is not the case, it then runs the program "newmainc" which reads the file "query.ari", generates French sentences and stores it in the file "parle.ari". This program takes a while to run. The file "query.ari" is deleted, and the file "parle.ari" is copied on Challenger machine with the extension lck. It is then renamed with the extension ari in order to be processed by the program "parler". Then the batch file returns control to the beginning of the loop. It waits until it receives the file "query.ari". The figure 8 shows the data flow between the two machines.

This integration works perfectly but has one bug, yet to be removed. After a session of three consecutive questions, the memory seems to be messed up. It happens always after the execution of the file parler. This file requires more memory to be allocated for the stack. Because Dragon, Televox and PC Quick net are in background and each of one is using a lot of memory, it was necessary to reduce the environment of "parler".

Figure 8. Dataflow Charts of batch files



CHAPTER VI

CONCLUSION

In this project, we developed an integrated software system for voice interface in French to access databases. It is implemented on two personal computers AT. The system composes of three modules: French natural language menu using voice drive, SQL query generation, Sentence generation and voice output. These modules use off-the-shelf subsystems. The French NL_menu makes use of Voicescribe system (isolated word recognizer). The sentence generation and voice output use Arity/SQL and a speech synthesizer respectively.

My contribution to this project is to develop a French text generation module and integrates all the modules into a system. Text generation system makes use of the entity relationship model as the vehicule for knowledge representation.

A given SQL query is put into one of the 11 classes by means of a simple analysis of its structure. For each of these 11 classes we have presented an algorithm that will generate a suitable deep structure of the target sentences. The transformational component will transform the deep structure into natural language sentences in French. These sentences are then presented in the form of voice output.

Another minor contribution of my project is to develop

SQL queries from the NL_menu input. The NL_menu input was done in an earlier thesis.

The integration of all the software modules is achieved using two personal computers and a network called PC Quick Net. Two batch files run on these machines that control the software modules resident on them. One machine (A) having the NL_menu and SQL query generation modules sends the output file containing the SQL query to the other machine (B). The machine B processes the output file and runs the text generation module which creates a new output file with a French sentence while the machine A is waiting. The new output file is then transferred back to machine A. Machine A generates the voice output while machine B is waiting for a new SQL query.

This integrated software system demonstrates the feasability to develop a voice interface system to access a small database on personal computers. The user can talk to the machine, and the machine answers him back using a synthesized voice. The constraints of isolated word recognizer, The NL_menu language input, the quality of generated text, and the quality of synthesized speech are all far from "ideal", but they contribute a step towards voice communication with computers to access the information stored in databases. Several improvements are suggested in chapter 4 to obtain better quality of generated French text.

In order to generate good text, we need to understand the context and what is required by the user. Generation of good text becomes as difficult as Natural Language understanding. There is hope to solve the underlying problems at least under the database domain.

REFERENCES

1. Appelt D.E., Planning English Referring Expressions, Artificial Intelligence, no. 26, 1985, 1-31.

2. Brathwaite K.S., Analysis, Design & Implementation of Data Dictionaries, McGraw-Hill Book Co., 1988.

3. Chomsky N., Aspects of the theory of syntax, Cambridge Mass., MIT Press, 1965.

4. Harris M.D., Introduction to Natural Language Processing, Reston Publishing Co., 1985.

5. Hutchins W., Dusek L., How vocabulary is generated determines speech quality, Computer Design, Feb, 1984, pp. 89-96.

6. Lea W.A., Voice Controlled Office Machines: A Critical Review, AFIPS Office Automation Conference Digest, 1984, pp. 159-174.

7. Mann W., Text Generation, American Journal of Computational Linguistics, vol.8, no. 2, April-June 1982.

8. Marakakis E.J., Entity Relationship Approach to the generation of sentences for database queries, M.Comp.Sci.,Thesis, Concordia University, 1984.

9. McKeown K.R., Discourse Strategies for Generating Natural-Language Text, Artificial Intelligence, vol. 27, 1985, pp. 1-41.

10. Menzies I., Voice Input for querying a database, M.Comp.Sci., Major Report, Concordia University, 1988.

11. O'Shaughnessy D., Speech Processing, Prentice Hall,1987.

12. O'Shaughnessy D., Archambault D., Barbeau L., Bernardi D., Speech Synthesis Using Diphones, MONTECH 88, Nov., 1987,

pp. 191-194.

13. PC Quick Net Reference Manual.

14. Radhakrishnan T., Marakakis E.J., Generation of Natural Language Response to Database Queries, Proc. of IEEE International Conference on Systems, Man and Cybernetics, pp. 825-829, 1983.

Appendix 1

```
% This program defines the ATN grammar used by Voicescribe.
% It defines  all the words that the user can say and the
%  order in which they should appear in a sentence.
% The terminal words are in lower case, and non terminal
%  words are in upper case. The terminal words are the words
% that the user can say. The non terminal words are
%  variables. A space between words indicates that the user
%   have to say those words in sequence. A comma between
%   words  indicates that the user can choose one of the two
%   words.

#keystroke ai 130;  % keystrokes define the french accents
#keystroke aa 133;
#keystroke es 138;
#keystroke hu 151;


%recule is an active word that the user can use at any time
Q[recule] = donne_moi (Q1, Q2, Q3, Q4);
Q1 = ATTRIBUT1;
Q2 = les NOM;
Q3 = ATTRIBUT2;
Q4 = le local (LOCAL1, LOCAL2) EX_OFFREID point;

NOM = (N1, N2, N3);

N1 = professeurs (PROF_SUB,PROF_SUB1);
N2 = cours CRS_SUB point;
N3 = etudiants (ETU_SUB, ETU_SUB1);

PROF_SUB = (S1,S2,S3) point;
PROF_SUB1 = ((S1 et S2),(S2 et S1)) point;
CRS_SUB = (S4, S5, S6, S7, S8);
ETU_SUB = (S9,S10) point;
ETU_SUB1 = ((S9 et S10),(S10 et S9)) point;

S1 = qui travaillent au departement EX_DEP;
S2 = qui enseignent EX_OFFREID;
S3 = qui dirigent EX_ETUD;
S4 = qui sont corequis a EX_CRS;
S5 = qui sont enseignes par EX_PROF;
S6 = qui sont pris par EX_ETUD;
S7 = dont le prerequis est EX_CRS;
S8 = qui sont les prerequis de EX_CRS;
```

```
S9 = qui prennent le cours EX_OFFREID;
S10 = qui sont dans le programme EX_PROG;

LOCAL1 = de;
LOCAL2 = ou se donne;

EX_CRS = comp (six, sept) DIGIT DIGIT;
EX_OFFREID = DIGIT DIGIT;
EX_PROF = (zero, un, deux) DIGIT;
EX_PROG = DIGIT;
EX_DEP = DIGIT;
EX_ETUD = (zero, un, deux) DIGIT;
DIGIT = (un, deux, trois, quatre, cinq, six, sept, huit, neuf, zero

ATTRIBUT1 = (PROF_ATT1,ETUD_ATT,CRS_ATT, DEP_ATT) point;
PROF_ATT1 = ((A1 (et A2) #), (A2 (et A1)#)) de EX_PROF;
ETUD_ATT = A5 de EX_ETUD;
CRS_ATT = (A3,A4) de EX_OFFREID;
DEP_ATT = A6 de EX_DEP;
A1 = le telephone;
A2 = le bureau;
A3 = la capacite;
A4 = les heures;
A5 = le statut;
A6 = le directeur;

ATTRIBUT2 = (PROF_ATT2, ETUD_ATT2);
PROF_ATT2 = ((A1 (et A2)#), (A2 (et A1)#)) des N1;
ETUD_ATT2 =  A5 des N3;
*SYSTEM = (procede, continue, fin);

donne_moi                        "Donne-moi ";
professeurs                      "professeurs ";
cours                            "cours ";
local                            "local ";
bureau                           "bureau ";
telephone                        "t'ai'l'ai'phone ";
prerequis                        "pr'ai'requis ";
dont                             "dont ";
est                              "est ";
sont                             "sont ";
etudiants                        "'ai'tudiants ";
travaillent                      "travaillent ";
au                               "au ";
departement                      "d'ai'partement ";
qui                              "qui ";
enseignent                       "enseignent ";
dirigent                         "dirigent ";
enseignes                        "enseign'ai's ";
par                              "par ";
pris                             "pris ";
le                               "le ";
les                              "les ";
la                               "la ";
de                               "de ";
prennent                         "prennent ";
dans                             "dans ";
```

programme	"programme ";
donne	"donne ";
recule	"recule";
comp	"Comp";
zero	"0";
un	"1";
deux	"2";
trois	"3";
quatre	"4";
cinq	"5";
six	"6";
sept	"7";
huit	"8";
neuf	"9";
procede	"proc'es'de";
continue	"continue";
fin	"fin";
et	" et ";
point	".";
des	"des ";
se	"se ";
corequis	"corequis ";
a	"'aa' ";
ou	"o'hu' ";
heures	"heures ";
statut	"statut ";
capacite	"capacit'ai' ";
directeur	"directeur ";