In: H. Winter (ed.): Artificial Intelligence and Man-Machine Systems. Berlin: Springer.

The Role of Natural Language in Advanced Knowledge-Based Systems

Wolfgang Wahlster

Department of Computer Science
University of Saarbrücken
D-6600 Saarbrücken 11
Federal Republic of Germany

Abstract: Natural language processing is a prerequisite for advanced knowledge-based systems since the ability to acquire, retrieve, exploit and present knowledge critically depends on natural language comprehension and production. Natural language concepts guide the interpretation of what we see, hear, read, or experience with other senses. In the first part of the paper, we illustrate the needed capabilities of cooperative dialog systems with a detailed example: the interaction between a customer and a clerk at an information desk in a train station. It is shown, that natural language systems cannot just rely on knowledge about syntactical and semantical aspects of language but also have to exploit conceptual and inferential knowledge, and a user model. In the remainder, we analyze and evaluate three natural language systems which were introduced to the commercial market in 1985: Language Craft™ by Carnegie Group Inc., NLMenu by Texas Instruments Inc., and Q & A™ by Symantec Inc. The detailed examination of these systems shows their capabilities and limitations.
We conclude that the technology for limited natural language access systems is available now, but that in the forseeable future the capabilities of such systems in no way match human performance in face-to-face communication.

1. Introduction

Natural language processing is a prerequisite for advanced knowledge-based systems since the ability to acquire, retrieve, exploit, and present knowledge critically depends on natural language comprehension and production. Natural language concepts guide the interpretation of what we see, hear, read, or experience with other senses. In artificial intelligence (AI) we are working under the assumption that natural language understanding and production are knowledge-based processes. Consequently, the existing natural language systems belong to the class of *knowledge-based AI systems,* which inter alia include expert systems and model-driven vision systems.

By definition natural language systems are no expert systems because every human knows at least one natural language without him having to be an expert for that reason.

The knowledge base of a natural language system includes both *linguistic (e.g.* lexicon, grammar, dialog rules) and *nonlinguistic* (e.g. a description of the objects in the domain of discourse) subparts. Whereas, ideally, the construction of the nonlinguistic part of the knowledge base is based on joint research of computer scientists and application specialists, the design and implementation of the linguistic parts relies on cooperation among computer scientists and linguists. For centuries linguists have gathered knowledge about various natural languages. In most cases unfortunately, this knowledge cannot be used directly in

The preparation of this paper was supported by the SFB 314 Research Program of the German Science Foundation (DFG) on AI and Knowledge-Based Systems, and DEC's EERP. The contents of the paper benefited from discussions with Johannes Arz. Thanks go to Heide Dornseifer for typing and editing the manuscript and its revisions.

natural language systems because it is represented in computationally untractable formats or because it is not detailed enough for transformation into algorithmic systems. Thus, it is often a collaborative effort among linguists as 'experts for language' and computer scientists as 'experts for the formal representation of knowledge' to construct linguistic knowledge sources for natural language systems.

In AI a piece of software is called a *natural language system,* if

(a) a subset of the input and/or output of the system is coded in a natural language
(b) the processing of the input and/or the generation of the output is based on knowledge about syntactic, semantic, and/or pragmatic aspects of a natural language.

According to condition (a), there are natural language systems which combine e.g. pointing gestures on a terminal screen with linguistic descriptions. XTRA (eXpert TRAnslator, cf. Kobsa et al. 1986), a dialog system currently under development in our laboratory, will assist the user in filling in his annual income tax form. Here, a typical input contains a tactile gesture: *'Should I add my expenses for business trips here* (pointing gesture on the income tax form displayed in a window of the terminal screen) *or is it better to increase this* (another pointing gesture) *value?.*

Furthermore, most natural language systems commercially available today do not generate natural language. Their output consists of formatted data retrieved e.g. from a database or fixed text strings. Other systems use natural language input but generate graphical output. Yet another class of systems generates natural language descriptions for purely visual input.

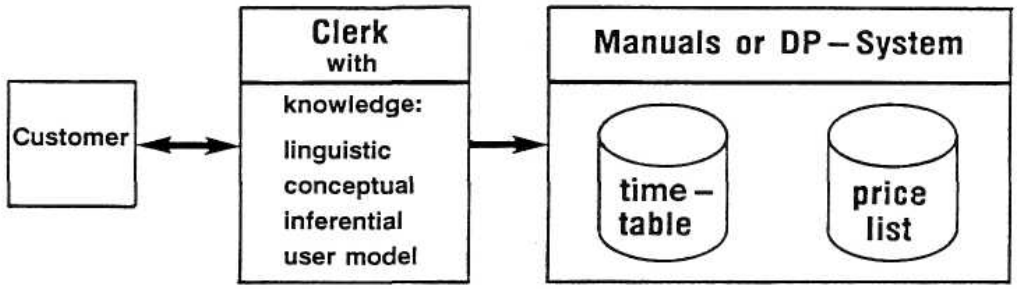
Condition (b) is very important in that it rules out systems which process natural language purely as strings of characters without understanding its content (e.g. text editors, statistical packages).

It should be pointed out that today most natural language systems are based on written input and output. Although hardware for medium quality speech synthesis (without much prosodic features) is available, the speaker-independent recognition of continuous speech is still a major research problem.

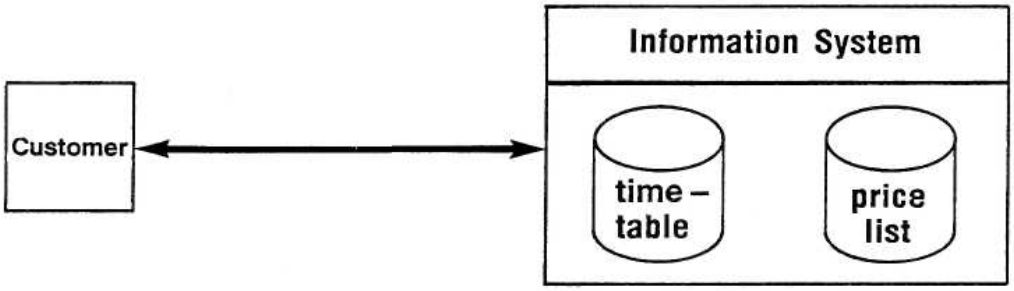
2. Natural language understanding as a knowledge-based process

Let us use a simple example to demonstrate the role of knowledge-based access systems in advanced information systems (see Fig.1).

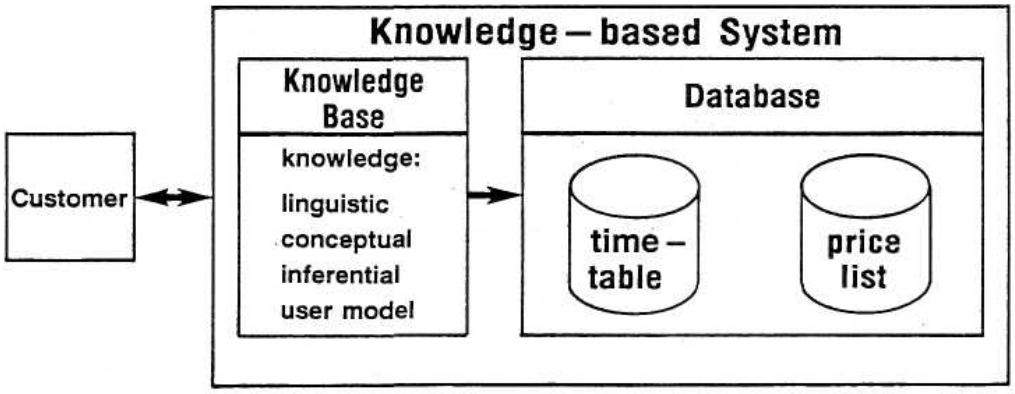
Usually, a clerk at an information desk in a train station uses a time-table and a price list to respond to questions of an information-seeking customer. As formatted mass data, these tables and lists are contained in manuals or in a database system. As external data, they are not a part of his internalized knowledge. Since for the clerk, the access to these tables and lists is of critical importance, it is clear that he must be familiar with the organization of the manuals or the database.



Usual situation at an information desk in a train station



Unacceptable service quality as a result of the clerk's replacement by a conventional information system



The combination of data – and knowledge bases results in an increased consultative capacity

Fig. 1: The need for knowledge- based consultation systems

For adequate consultation however, the clerk must, aside from such *access knowledge,* activate other areas of knowledge as well (see Fig. 1)

(a) If the customer says 'My son must be on the train to Saarbrücken on Monday. Does it have intercity connections?', he shortens his second sentence by using the pronoun 'it' instead of 'the train'. The clerk's linguistic knowledge helps him to select the correct referent for the pronoun. He is able to rule out 'My son', 'Saarbrücken', and 'Monday' as possible antecedents for 'it'. Furthermore, the clerk can use his *linguistic knowledge* about speech acts to recognize that the client does not just expect a yes/no answer, but also the departure times of suitable intercity trains to Saarbrücken.

(b) If the customer asks 'What is the difference between a sleeping-car and a couchette car' the clerk cannot find the answer in his time-table or the price list. But the clerk *can* use his *conceptual knowledge* to compare both concepts and to identify the distinguishing features of both alternatives for spending the night on a train.

(c) If the customer says 'I am going to Greece on an excursion together with my professor and our archeological seminar. Can I use a Eurorail ticket?', the clerk exploits his *inferential knowledge* referring to a rule like 'If the client is a student and is under 27 years old, then he can buy a Eurorail ticket'. He can apply this rule in a backward chaining mode, which means, that he has to test the if-part of the rule. Using an inference rule like 'If someone is attending a university course, then he is a student' the clerk can infer that the customer is a student. Then the clerk can respond with 'Yes' if the customer gives an affirmative answer to the clerk's question 'Are you under 27?'.

(d) If the customer says 'A return ticket to the Hanover Fair, please' the clerk will most probably offer him a first-class ticket. This response is based on a *user model* which the clerk derives from the assumption that a visitor to a professional fair is on a business trip. With the user class 'businessman' the clerk associates certain stereotypical knowledge, e.g. that travel costs are usually reimbursed for business trips. Thus the clerk assumes that a first-class ticket will be preferred by the customer.

In the 70's, there were many computer science projects which tried to replace the clerk by an information system. In such a scenario, the customer formulated his request in a query language and a DBMS evaluated the query and retrieved the relevant data from a database in which the time-table and the price list were stored (see Fig. 1) - the result of which being unacceptable service quality for most customers. For someone who uses a train twice a year, it is unreasonable to have to learn a formal query language (even if the query language is very simple like pushing a combination of four buttons out of a menu of 50) for getting information on railway connections. Even if the customer would spend the time to learn the query language, the lack of expressive power of current database query languages compared with natural language, e.g. the absence of indirect speech acts and anaphoric devices as exemplified in paragraph (a), makes such a dialog with an information system rather frustrating.

The questions presented in (b) - (d) above cannot be answered by a conventional information system based on database technology. No currently available DBMS can record and maintain complex conceptual knowledge structures, apply inference rules, and build up a user model as illustrated in examples (b) - (d).

From the AI point of view, the information system has to be complemented by a knowledge-based access system in order to become acceptable for the end-user (see Fig. 1).

In our example scenario, a knowledge-based consultation system could provide an increased consultative capacity by combining a database system with a knowledge base, containing a formal representation of the linguistic, conceptual, and inferential knowledge of the clerk, as well as stereotypes for user modeling.

Although presently for each problem mentioned in (a) - (d) at least one experimental knowledge-based access system which can adequately handle this type of question exists, it is, of course, a long way from implementing the broad-based universal communication capabilities of a clerk into an integrated and robust real time system.

A particular problem, far from being solved in any AI system, is the permanent knowledge acquisition of humans. In our example, one usually assumes that a clerk reads newspapers or watches TV news programs. Therefore he may know that this year the Davis Cup final takes place in Munich. Thus, if a customer asks for a return ticket to the Davis Cup final, the clerk will probably be able to offer him a return ticket to Munich. For a knowledge-based system today, it would be unrealistic to assume that the system daily updates its knowledge base with information potentially relevant for consultation purposes.

Although at present, the consultation capabilities of a knowledge-based system in no way match human performance, our example demonstrates that, if there were a real bottleneck in consultation capacity, a knowledge-based access system might help.

The example shows also that there are applications in which knowledge-based systems have to be combined with database systems. Without access to the time-table and price list, the best natural language system would be worthless in the scenario previously discussed. Furthermore, it would be clearly inadequate to represent the formatted mass data in one of the knowledge representation languages developed in AI.

In the following we analyze and evaluate three natural language systems which were introduced to the commercial market in 1985: Language Craft ™ by Carnegie Group Inc. (see Carnegie Group 1985b), NLMenu by Texas Instruments Inc. (see Texas Instruments 1985b) and Q & A ™ by Symantec Inc. (see Kamins 1985). It is interesting to note that each of these commercial systems was developed under the direction of a well-known scientist with considerable research experience in NL processing. The principle designer of Language Craft is J. Carbonell, who previously developed natural language systems such as XCALIBUR (cf. Carbonell et al. 1983) at Carnegie-Mellon University. NLMenu was designed by H. Tennant, who formerly was a member of the design team for the natural language system JETS (cf. Tennant 1980) at the University of Illinois at Urbana Champaign. G. Hendrix, as a principle designer of the natural language interfaces LIFER and LADDER (cf. Hendrix et al. 1978) at SRI International, has built the Q & A system for Symantec.

3. Language Craft: a transportable interface based on caseframes

Language Craft is a tool for constructing natural language interfaces to interactive systems, including databases and expert systems (see Carnegie Group 1985a, 1985b). Language Craft consists of PLUME™ and the Grammar Writer's Workbench™. PLUME includes a parser which contains some built-in knowledge of English morphology and syntax. For each new application, the developer of the interface has to write a domain-specific grammar, which mainly consists of a set of caseframes. The Grammar Writer's Workbench is a software tool for assistance in creating, editing, testing, and compiling domain-specific grammars.

In addition to a parser, which maps the natural language input onto a caseframe representation, PLUME provides a translator to convert the caseframes into Lisp function calls, driving the application. The rule-based translator works with two sets of rules, which the grammar writer has to specify. One set contains rules which collapse different parses into a single parse by rearranging caseframes to map into a canonical form. The other set of rules converts the caseframes into the actual Lisp forms to be passed on to the interactive application system.

Language Craft includes components for spelling correction and ellipsis resolution. For example, the system (S) uses the context provided by sentence (3) in Fig. 2 (see Carnegie Group 1985b) to fill in the missing information in (4).

When an input is ambiguous or a paraphrase is requested by the user (U), Language Craft uses a simple natural language generator which verbalizes the caseframes instantiated during parsing. As illustrated by example (6) in Fig. 2, it can paraphrase several readings of the input and ask the user to choose between them.

(1) U: Set up a Friday meeting for me at 2:00 with Mary Smith and Ron Brown.
S: Following meeting scheduled:
Friday March 29 2:00 to 3:00 Mary Smith, Ron Brown, Bob Jones

(2) U: What meetings does Smith have?
S: Monday March 25 10:00 to 12:30 J.P. Roberts, Mary Smith Tues. March 26 4:00 to 5:00 Mary Smith, Bob Jones Friday March 29 2:00 to 3:00 Mary Smith, Ron Brown, Bob Jones

(3) U: List all my meetings on Tuesday.
S: Tues. March 26 1:00 to 1:30 J. P. Roberts, Bob Jones, Jane Arran Tues. March 26 4:00 to 5:00 Mary Smith, Bob Jones

(4) U: Friday meetings
S: Friday March 29 2:00 to 3:00 Mary Smith, Ron Brown, Bob Jones

(5) U: Cancel the meeting at 1 pm Tuesday
S: Following meetings cancelled:
Tues. March 26 1:00 to 1:30 J.P. Roberts, Bob Jones, Jane Arran

(6) U: Give me 10:00 meeting with Smith Friday.
S: Sorry! Found 2 possible interpretations for your input:
1) Schedule a meeting for me with Smith on Friday at 10:00.
2) List a meeting for me with Smith on Friday at 10:00.
Please enter the number of the correct interpretation (0 for none of the above): U: 1 S: Following meeting scheduled:
Friday March 29 10:00 to 11:00 Mary Smith, Bob Jones

Fig. 2: Sample dialog with a meeting scheduling system

The domain-specific grammatical knowledge is specified mainly in the form of case-frames. Caseframe definitions include syntactic knowledge which is expressed in a powerful pattern matching language. In addition to various pattern operators e.g. for disjunction, optionality, negation, assignment, and wildcards, the pattern language is comprised of so called non-terminal patterns.
Non-terminal patterns refer to complex patterns which are repeatedly used in a grammar.
For example:

(1) (schedule) →(schedule !! (&verb set) up !! arrange)

If (schedule) appears in the grammar, it is replaced by the right-hand side of the rewrite rule (1).
The parser treats '!!' as an 'exclusive or' and stops as soon as it finds a match with an element of the disjunction.

For regular cases, morphological processing is automatically done by PLUME, so that in (1), the grammar writer only needs to specify the infinitive form for verbs. Thus, the terminal pattern 'schedule' in (1) will match e.g. 'schedules', 'scheduled' and 'scheduling'.

If a terminal pattern consists of several words (e.g. 'set up' in (1)), the grammar writer has to specify the word on which morphological transformations are to be performed (e.g. (&verb set) in (1)).

The rewrite rules for non-terminal patterns can be recursive, as illustrated by (2):

(2) ⟨directory⟩ (%slash $?⟨directory⟩)

The operator '?' is used to indicate optionality and '$' is a wildcard which matches any single symbol. The symbol % slash represents the character V in a pattern. The above pattern matches UNIX ™ directory names like '/man' or '/man/ray'.

Since the PLUME parser is mainly based on caseframe instantiation, it can handle inputs with odd word orders. PLUME distinguishes between sentential, nominal, and adjectival caseframes. Sentential caseframes usually represent sentences containing verbs, nominal caseframes represent noun phrases, and adjectival caseframes represent adjectives or relative clauses. Each caseframe contains a set of patterns (called header) which, if matched, activate the case frame. PLUME tries to fill the case slots of the activated case frames.

For every case, each caseframe definition contains a filler specification, which is either a pattern to look for in the input, or a name of another caseframe. There are other slots that tell the parser how to locate the case in the input, and in which order the cases should be checked.

The following example shows parts of the caseframes for 'schedule' and 'meeting':

```
* schedule*
        :type sentential
        : header < schedule >
        :cases
              (initiator
                  :positional subject :
                  :filler *name*)
              (meeting
                  :positional direct - object
                  :filler * meeting*)
              (day
                  :type minor
                  :semantic - class time
                  :case - marker ? < on >
                  :filler <days>)...]
```

```
[* meeting*
        :type nominal
        :header   <meeting>
        :cases
              (participants – with
                  :case - marker in
                  :adjective *none*
                  :filler *name*
              (room
                  :case – marker
                  :filler *room*
              (duration
                  :case marker for !! of
                  :filler *duration*)…]
```

The caseframe '*schedule*' refers to '*name*' and '*meeting*' as two nominal case-frames. It uses the non-terminal patterns ⟨schedule⟩, ⟨on⟩ and ⟨days⟩. The ':positional' slot of the initiator case tells the parser to identify the *right* most unmarked
case to the left of the verb as a possible slot filler. The ':semantic-class' field, which has 'thing', 'person', 'place', and 'time' as possible values, helps the parser to handle Wh-questions with 'where' and 'when', and to differentiate between 'what' and 'who'. The ':case-marker' slot tells the parser that the

pattern (on) may immediately precede the filler (days). Day is treated as a minor case, i.e. it is not examined unless there is input left over after the instantiation of all preferred and major cases. In the nominal caseframe '*meeting*' the value of the ':adjective' slot indicates that a prenominal use of this case is not permitted (e.g. 'the with Roberts meeting').

Given the input sentence 'Schedule a meeting with Schmidt in room 4' PLUME uses inter alia the caseframes shown above to construct the following caseframe instance:

```
{* schedule*
    (%mood declarative (meeting
        {"meeting*
            (% determination * indefinite)
            (participants - with
                {*name*
                    (lname Schmidt)})
            (room
                {'room*
                    (% number (singular))
                    (room - number 4)})
                (%number (singular))})
    (% voice active)
    (% tense (present))}
```

The parser scans the input and finds 'schedule' as a caseframe header of '*schedule*'. It picks up 'meeting' as a header of the direct object, filling the 'meeting' case in the '*schedule*' caseframe. In the '*meeting*' caseframe, it finds 'with' as a casemarker for the 'participants-with' case and 'in' for the 'room' case. In the '*name*' and '*room*' caseframes, it assigns 'Schmidt' and '4' as fillers to their slots 'lname' and 'room-number', respectively.

PLUME covers modes, tenses, voice, and negation automatically, so that the grammar writer only has to provide the caseframe definitions for each application domain.

The parser handles 'Yes/No' questions and 'who'-, 'what'-, 'where'-, 'when'-, and 'which'-questions using the same caseframes applied to parse declarative sentences.

Quantification and determination which are also covered by the grammatical knowledge built into the PLUME parser, appear as cases named '%quantification' and '%determination'. The quantifiers and determiners recognized by PLUME are 'most, almost all, more, any, each, none, few, all, every' and 'a, an, the, this, that, these, those, either, neither', respectively.

Release 2.0 of Language Craft does not include a pronoun resolution mechanism. It is implemented in Common Lisp and runs on Symbolics™ Lisp machines and the VAX™ under VMS™.

4. NLMenu: a Menu-based natural language interface

Presently, a major problem with most natural language systems is that they allow the user to input well- formed expressions which the system cannot understand. Thus, there is a mismatch between the user's linguistic competence and the system's limited coverage of natural language.

The basic idea of Menu-based NL interfaces like NLMenu is that instead of typing natural language in an unconstrained way, the user is guided by a set of dynamically generated menus to use the subset of natural language that the system understands (see Thompson 1984). The items of the menus are words, phrases, or so-called 'interaction experts', which are included in angle brackets (see Fig. 3).

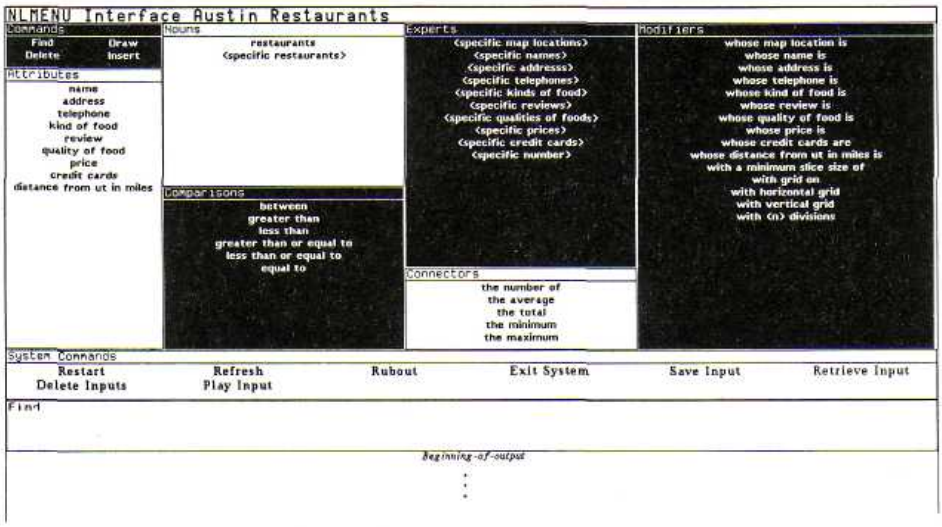


Fig. 3: The first selection phase

To select a menu item, the user moves the mouse cursor to the desired item and clicks on the mouse. After each selection by the user, NLMenu examines the natural language expression constructed so far, and decides which windows to activate next and which items to place in the windows. A selection of windows displayed in reverse video is not possible. As soon as the user has constructed a string, which presents a complete sentence according to the grammar controlling the NLMenu driver, the 'Execute' command appears as a selectable item in the window, displaying applicable commands. This approach has the advantage that the user's input is generated to be both syntactically and semantically meaningful to the system. Figs. 3-5 (taken from Texas Instruments 1985a) illustrate the Menu-based approach using NLMenu as an interface to a restaurant database.

In Fig. 3, the user has selected 'Find' from the 'Command' menu; the only active window at the beginning of a new input. 'Find' appears in the sentence construction window at the bottom of the NLMenu screen shown in Fig. 3. NLMenu makes the 'Command' menu 'inactive' and activates the 'Attributes', 'Nouns', and 'Connectors' windows.

After two selection phases in which the user first chooses 'restaurants' from the 'Nouns' menu, and then, 'whose kind of food is' from the 'Modifiers' menu, NLMenu presents the screen shown in Fig. 4.

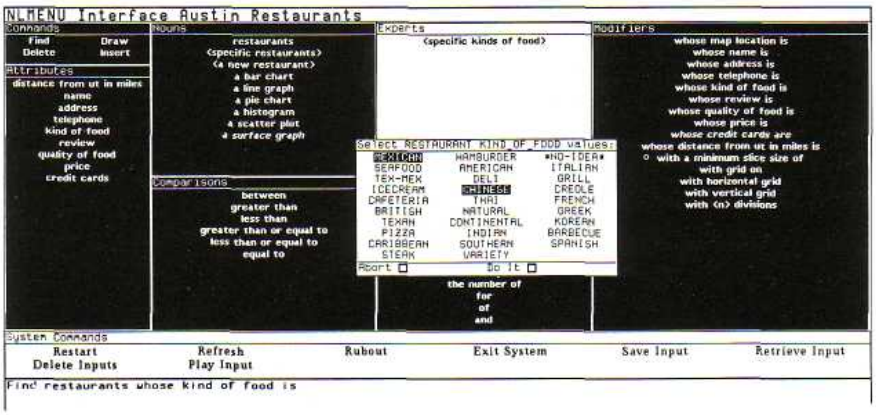


Fig. 4: Using an 'interaction expert'

If the user chooses '(specific kinds of food)' a special window pops up with actual database values as shown in Fig.4. The user selects 'Mexican' and 'Chinese', so that his query now reads 'Find restaurants whose kind of food is Mexican or Chinese' (cf. sentence construction window in Fig. 5). NLMenu recognizes this as a complete sentence and presents the 'Execute' option in the 'Systems commands' window. The user may 'Execute' the query or continue it by selecting an item from the 'Connectors' menu. If he decides to 'Execute' his query, the system retrieves an answer from the database.

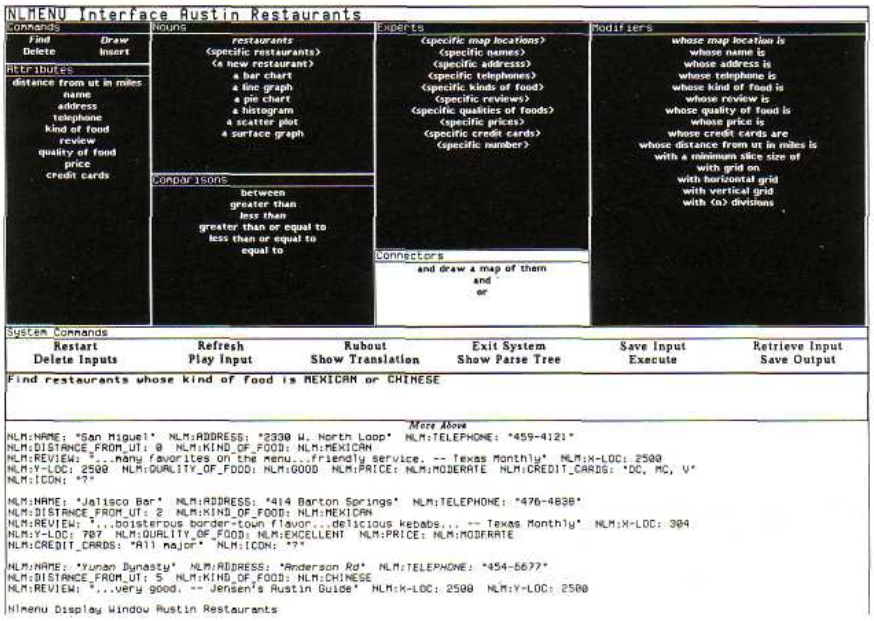


Fig. 5: The user executes the query

A common problem in natural language interlaces to databases is the recognition of database values when they appear in the user's input. Due to storage limitations, storing database values in the lexicon is not feasible for large domains. For example, consider an aircraft maintenance system for which the name of each aircraft has to be stored both in the database and the lexicon.

The developer of a NLMenu-based interface can use so-called interaction experts to support the user in referring to a specific database value. The 'interaction experts' are procedural augmentations to a context-free grammar. NLMenu provides a library of procedural 'interaction experts', e.g. for type-in

windows or menu windows, in which items coming from the database are displayed. In Fig. 5, an 'interaction expert' was used to let the user specify a disjunction of a single-valued database field (Kind_of_food).

The most important knowledge sources, which the developer of a NLMenu interface to a new application has to specify, are the grammar and the lexicon. The grammatical knowledge is represented by an attributed context-free semantic grammar, which does not contain cycles and empty productions. The basic grammar rule format is:

("A→B $_1$ B $_2$...B $_n$" (⟨Translation function 1⟩...⟨Translation function m⟩))

The translation functions, which represent the semantic component of the grammar, are Lisp expressions where numbers correspond to the position of the symbols on the right-hand side of the rule.

The lexicon of NLMenu defines the items listed in the selection menus and relates them to the terminal symbols in the grammar. In addition, it contains information on the application-dependent meaning of a menu item in the form of so-called translations.

The basic lexical entry format is:

(⟨Terminal Symbol⟩ "⟨menu-item⟩" ⟨menu-window⟩ "⟨Translation⟩")

⟨Menu-window⟩ is the name of the menu in the selection window to which the item maps. ⟨Translation⟩ is a lambda expression or an atomic expression.

The translation functions specified in the grammar rules refer to the translations stored in the lexicon. For example, if a translation function reads (1 2 5) and the first, second and fifth symbol of the right-hand side of the corresponding rule is a terminal symbol of the grammar, the translation portion of the lexical entry for the first symbol is applied with the translation specified in the lexicon for the second symbol. That result becomes the function that is applied with the translation for the fifth element as its argument. Note that the application of a lambda expression to an argument produces another lambda expression or an atomic expression.

Consider a relational database which includes a relation RUNWAY with attributes 'Name' and 'Length'. Suppose we want to write a NLMenu interface to the database which is accessible via the SQL query language. Then, the task of NLMenu is to transform a natural language input like (1) into the SQL query (2).

(1) Find name and length of runways
(2) (SELECT NAME, LENGTH FROM RUNWAY)

In the following, this simple example is used to discuss the main idea behind the translation process.

Assume the developer of the NLMenu interface has specified the following somewhat simplified grammar (the parentheses indicate an optional sequence of symbols) and lexicon for this application:

Grammar:

(a) ("S find MODIFIERS for MODIFIABLE-NP" ((4 (1 2))))
(b) ("MODIFIERS ATTRIBUTE (attr-and MODIFIERS)" ((1) (2 1 3)))
(c) ("MODIFIABLE-NP runways"((l)))
(d) ("ATTRIBUTE name"((l)))
(e) ("ATTRIBUTE length"((1)))

Lexicon:

```
(find "find" COMMAND "lambda f Select f from")
(name "name" ATTRIBUTES "name")
(length "length" ATTRIBUTES "length")
(attr-and "and" CONNECTORS "lambda a lambda b a,b")
(for "of" CONNECTORS nil)
(runways "runways" NOUNS "lambda r (r RUNWAY)")
```

Beginning with the lowest, rightmost subtree, the translator traverses the following parse tree built by the parser:

```
S (4 (1 2))

  (1) find
        "find"                          lambda f          Select f from

  (2) MODIFIERS (2 1 3)
      (1) ATTRIBUTE
            name (1)
               "name"
      (2) attr-and
            "and"                       lambda a lambda b a,b
      (3) MODIFIERS
            ATTRIBUTE (1)
               length (1)
                "length"

  (3) for
       "of"

  (4) MODIFIABLE-NP
            runways (1) "runways"       lambda r (r RUNWAY)
```

The translator then gets the translation list for the rule that constructed this subtree and gets the translations for the corresponding lexical entries. In our example it begins with rule (e). The corresponding translation list of lenght 1 has an atomic translation that is simply passed up the tree unchanged. The translator then moves up the tree one rule. After another translation step in which this atomic translation is passed up the tree (using rule (b) without the optional part), the translator applies the translation function (2 1 3) to the lexical entries for 'attr-and', 'name', and to the result of the previous translation which, in our example, is the atomic expression 'length'. The result of ("lambda a lambda b a,b" "name" "length") is "name, length". The translation function of rule (a) tells the translator to first apply the lambda expression representing "find" to the expression representing the translation of the MODIFIERS-subtree. This produces the result "Select name, length from". This expression is then used as an argument for the lambda expression specified in the lexicon as a translation for "runways". The application of "lambda r (r RUNWAY)" to "Select name, length from" produces the SQL query (2) as the final output.

NLMenu provides several software tools for the developer of a new interface (see Texas Instruments 1985b). The grammar writer's toolkit includes e.g. a conflict checker and a cycle checker for the grammar. Other tools assist in the completion and modification of the lexicon and in building the selection windows.

An interesting feature of NLMenu is the database interface generator which uses a NLMenu interface to acquire knowledge about a new database from the developer of a new application. When the developer has specified the necessary domain-dependent information, the interface generator uses a generic 'core' grammar and lexicon to automatically generate the complete grammar and lexicon for the application. The database interface generator is based on a two-level grammar approach. The 'core grammar' of NLMenu is a set of hyper-rules, i.e. rule templates that can be expanded to a set of context-free rules. The developer of a new interface only needs to specify values for a set of meta-rules, which in turn provide the slot fillers that instantiate the hyper-rules. Presently, the interface generator can only be used for simple relational databases since the 'core' grammar and lexicon of NLMenu is quite small.

A disadvantage of NLMenu is that the sentences constructed sound somewhat stilted. A principal reason for this is that offering various paraphrases would clutter the screen too much.

With NLMenu, no elliptical sentences can be directly constructed. But the user can point with the mouse to words or phrases in his previous input generated by interaction experts and change the items in the pop-up menus.

The current version of NLMenu does not provide morphological analysis, pronoun resolution or cooperative response generation. NLMenu runs on the Explorer™ Symbolic Processing Systems produced by Texas Instruments.

5. Q & A: natural language database access on a personal computer

Q & A combines a text editor and a simple database system with a natural language interface in one software package. The natural language interface called 'intelligent assistant' (IA) supports retrieval and update operations on the database. Unlike Language Craft and NLMenu which run on powerful, but relatively expensive workstations or mainframes, Q & A runs on IBM Personal Computers or compatibles requiring a minimum of 512k RAM.

In contrast to the general relational data model, databases in Q & A are restricted to a single relation. In Q & A, each tuple of the relation stored in the database is called a 'form'. An attribute of a relation is referred to as a 'field'. The field values of a Q & A database are taken from a fixed set of domains which includes 'Numbers', 'Money', 'Dates', 'Hours', 'Text strings', and 'Yes/No'. In the database schema, which degenerates to a form template in Q & A, the user has to specify the names of the fields together with the corresponding domains from which values can be taken to fill out fields of a form.

IA has a built-in vocabulary of some 400 words. In addition, it uses the database-as-dictionary approach, i.e. all nonnumeric field values of the stored forms are inverted, providing an index of all the words and phrases used in the database and the fields in which they occur. The information contained in the database schema (field names and associated domains) is also used as part of the lexicon. Finally, there is an interactive knowledge acquisition component based on a 'Lesson Menu' in which the user can specify:

- a set of generic terms that refer to forms in the database (e.g. {employee, staff member, person}, {enrollment, sign up})

- a set of unique alternate field names (e.g. {supervisor, boss} for the 'manager' field specified in the database schema)

- a set of verbs, which like alternate field names, refer uniquely to a field (e.g. the verb 'earn' can be associated with the 'salary' field, so that 'How much does John earn' is interpreted as a paraphrase of 'What is John's salary')

- measurement terms that can be associated with numeric fields (e.g. 'days' for the numeric field 'accrued vacation')

- a set of adjectives which can be associated with specific fields form the domains 'Numbers' and 'Money'. For each new adjective, the user has to specify whether it describes a low value or a high value on the according numerical scale. Note that this classification is field-specific, e.g. 'excellent' may refer to a low value of the 'gas consumption' field but a high value of the 'salary' field. The semantic interpretation for the unmarked forms of the adjectives is 'higher (or lower) than average', for comparative forms 'higher (or lower) than the comparative value', and for superlatives 'the maximal (or minimal) value'.

In the Menu-based knowledge acquisition dialog, the user can also provide IA with three types of semantic information not contained in the database schema by marking fields that

- can be used as definite descriptions, i.e. that uniquely identify a specific form in the database (these fields are included in all detailed reports)

- contain locations (required for answering 'Where is'-questions)

- contain people's names (e.g. first name, last name, title; required for answering 'Who'-questions).

The user of IA can define various sorts of synonyms and paraphrases, which he may introduce during the natural language dialog (e.g. by 'Define total pay as salary plus bonus) or in a stylized subdialog. Words like 'well', which may be ignored in the input, can be defined as 'empty' synonyms (e.g. 'Define well to be"").

Whereas the user of Q & A can apply the various methods discussed above to enhance IA's lexical knowledge, he cannot change or extend the grammatical knowledge of the system.

The built-in grammatical knowledge of IA is represented by an attributed context-free grammar which, in the distributed version, is compiled into a more efficient but equivalent form. IA works with a semantic grammar, i.e. most nonterminals are semantic categories specific to the database application. Left-recursion is not allowed in the rules. During a parse of a sentence like 'Add 100 $ to Schmidt's bonus' the following rule is applied:

⟨Revision⟩ → (ADD (T4⟩ TO ⟨T4_2⟩)
(SHOW-FIELD (COMPOSE_TERM ⟨T4⟩ ' + ⟨T4.2⟩))

The right-hand side of a rule can include individual words ('ADD' and 'TO' in the example above), predicates (e.g. for numbers) and/or nonterminals (⟨T4⟩ and ⟨T4_2⟩ in the example above).

Associated with each rule is a LISP expression, specifying a structure to be produced by the parser. Because the nonterminals are used as variables in the attached LISP code, it is necessary to introduce new terminals if otherwise there would be a conflict (as in the above example). The LISP expression can also contain conditions which, if not met, can have the effect of blocking a rule.

Before the parser is applied to the user's request, a scanner maps the input onto a canonical representation. The scanner includes a morphological component which recognizes regular verb endings, regular comparative and superlative forms of adjectives, and regular plurals of nouns and possessives. It substitutes the definition for all synonyms and paraphrases recognized in the input. Whereas all other morphological information is discarded by the procedure called 'lexical stripper', only the information contained in the suffices for comparatives, superlatives and possessives is forwarded to further processing stages.

The backtracking parser, which is implemented in IQLISP, transforms the natural language input into a sequence of so-called 'packets'. Packets are words or phrases in the input which form a semantic unit corresponding to constructs of Q & A's data manipulation language. For example, the input (1) is transformed into the packet sequence (2).

(1) What is the average age of females with salary >30000.
(2) ((AGG A) (? AGE) (R SEX =FEMALE) (R SALARY >30000))
(3) ((TYPE SUPPRESSED)
(FIELDS (AGE A))
(RESTRICTION (SEX =FEMALE) (SALARY >30000)))

Here, the parser has recognized four packets: the aggregate (AGG) function 'average' (A), the database field 'AGE' as the focus of the request, and two restrictions (R) on the field values.

A component called 'semtop' translates this sequence of packets into a semantic representation called 'logical form' (see (3) above). This representation is the input to a 'rephrase' procedure which generates a stylized English paraphrase of what the system intends to do. If the user gives his OK for the paraphrase, the logical form is input to a procedure which generates the so-called functional form of the user's request. The result of this final translation process is submitted to the database management system.

IA has no problems with partial sentences, i.e. input in a telegraphic style like 'Secretary's salaries' (which IA interpretes as 'Show me the names and salaries for all secretaries'), because its semantic grammar is not based on the notion of complete sentences but on packets as smaller units.

If a phrase in the user's input is ambiguous, IA highlights it and asks the user for clarification. It presents a numbered list of the paraphrases of all readings found, and the user can select the intended request by one of the numbers.

When encountering an unknown word which is not enclosed in quotes (used in update requests for marking new information), IA highlights it and asks the user for a clarification or a correction.

IA's built-in grammar is relatively large (more than 250 nonterminals) and covers a significant portion of possible retrieval and update operations on Q & A databases. It copes with declarative, imperative, and interrogative sentences. The system handles 'What'-, 'Who'-, 'Where'-, and 'How many' -questions. It covers restrictive relative clauses and some forms of conjunction and disjunction on the level of noun groups, verb groups, and clauses, but excludes embedded possessives like 'Gary's manager's age'.

IA can handle simple follow-up questions (see Fig. 6) which include singular or plural forms of personal pronouns (e.g. they) or possessive pronouns (e.g. her). Only the most recent request/response pair is checked for antecedents of a pronoun.
The instruction manual gives a crisp summary of IA's limitations: "You are infinitely smarter than the assistant,

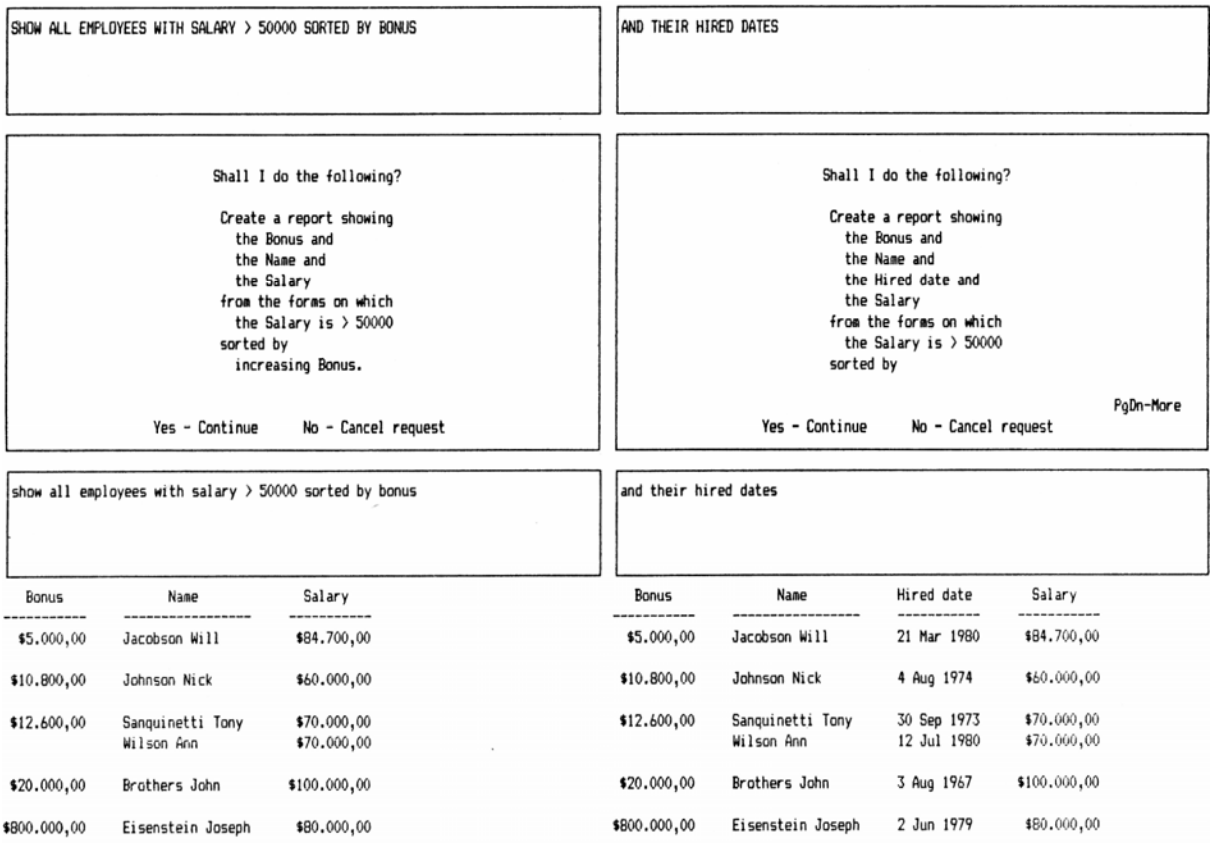


Fig. 6: A sample interaction with Q & A

have a far larger vocabulary, and possess a much stronger grasp of the English language" (see Kamins 1985, p. IA-3).

6. Concluding remarks

Fig. 7 summarizes the distinguishing features of the three systems discussed in this paper. It is important to keep in mind that a 'Yes' in the table does not indicate, that the corresponding problem has been solved in its full generality.

The commercial systems compared in Fig. 7 offer the promise of easy transportability from one domain to another by providing various tools for easy acquisition of domain-specific information. They try to have the domain-dependent knowledge sources clearly seperated from the more general domain-independent knowledge sources.

Two of them (NLMenu and Language Craft) go beyond domain-independence in the attempt to provide a core system that is not restricted to database access but can serve as a basis for interfaces to a variety of interactive systems (e.g. expert systems, command languages).

One of the major prerequisites for a wide-spread application of most language access systems is that the system allows its adaptation to new applications by nontechnical people who know the application domain, but are neither experts on the system nor specialists in AI or computational linguistics. This prerequisite is not yet fulfilled by the transportable systems NLMenu and Language Craft. On the other hand, Q & A fulfills this criterion, but it is restricted to database access. Moreover, some knowledge about a particular database system is built in at a very low level of processing.

	Language Craft	NLMenu	IA
Handling input with odd wordorder	Yes	Not possible	Yes
Handling partial input/ellipsis	Yes	No	Yes
Simple pronoun resolution	No	No	Yes
Generation of paraphrases	Yes, if input is ambiguous or on user's demand	No	Yes, always
Spelling corrector	Yes	Not necessary	No
Recognition of database values in the input	must be included in the lexicon	'interaction experts'	database is used as an extension of the lexicon
Morphological analysis	Yes	No	Yes
Expandable grammar	Yes	Yes	No
Type of grammar	patterns in caseframes	attributed context - free grammar	attributed context - free grammar
Type of parser	caseframe instantiation	bottom - up, incremental	top-down, backtracking
Target language of parser	instantiated caseframes	application - dependent	logical form
Target system	interactive software	interactive software	Q&A databases

Fig. 7: A functional comparison of three commercially available NL interfaces

One of the major prerequisites for a wide-spread application of most language access systems is that the system allows its adaptation to new applications by nontechnical people who know the application domain, but are neither experts on the system nor specialists in AI or computational linguistics. This prerequisite is not yet fulfilled by the transportable systems NLMenu and Language Craft. On the other hand, Q & A fulfills this criterion, but it is restricted to database access. Moreover, some knowledge about a particular database system is built in at a very low level of processing.

A general-purpose natural language dialog system should be adaptable to applications that differ not only with respect to the domain of discourse, but also to dialog type, user type, and intended system behavior. In Wahlster and Kobsa 1986, we call such systems, which are transportable and adaptable to diverse conversational settings, transmutable systems. A first attempt to build a transmutable system was our design of the experimental dialog system HAM-ANS (see Hoeppner et al. 1984), whose dialog behavior can be switched from a 'cooperative' mode (e.g. the system answers questions about a traffic scene) to a 'interest-based' mode (e.g. the system tries to persuade the user to book a room in a particular hotel).

When people communicate, they do so for a purpose specific to the conversational situation. On the other hand, the systems discussed in this paper have no interest beyond providing the information-seeking user with relevant data. In the long run, natural language systems as components of advanced knowledge-based systems must perform a greater variety of illocutionary and perlocutionary acts: they may teach, consult, or persuade the user, inspire him to action or argue with him (see Bates and Bobrow 1984, Wahlster 1984, Webber 1986, Woods 1984). The major problem builders of trans-mutable systems are confronted with, is the lack of a representational vocabulary for the declarative description of the relationship between the system and the user, the system's intended dialog behavior and the associated conversational tactics.

Although in the forseeable future, natural language AI systems will not be able to behave exactly like the clerk at the information desk, discussed in the first part of this paper, the technology for transportable natural language access systems is available now and begins to have a major impact on the design and application of man-machine systems.

References

Bates, M., and R.J. Bobrow (1984): Natural Language Interfaces: What's Here, What's Coming, and Who Needs it. In: Reitman, W. (ed.): Artificial Intelligence Applications for Business. Norwood: Ablex, 179-194.

Carbonell, J.G., W.M. Boggs, M.L. Mauldin, P.G. Anick (1983): The XCALIBUR Project: A Natural Language Interface to Expert Systems. In: Proceedings of the International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 653-656.

Carnegie Group (1985a): Language Craft™: An Integrated Environment for Constructing Natural Language Interfaces. Pittsburgh, PA.

Carnegie Group (1985b): The Language Craft™ Manual. Release 2.0. Pittsburgh, PA.

Hendrix, G.G., E.D. Sacerdoti, D. Sagalowicz and J. Slocum (1978): Developing a Natural Language Interface to Complex Data. In: ACM Transactions on Database Systems.

Hoeppner, W., Th. Christaller, H. Marburger, K. Morik, B. Nebel, M. O'Leary and W. Wahlster (1983): Beyond Domain-Independence: Experience with the Development of a German Language Access System to Highly Diverse Background Systems. In: Proceedings of the International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 588-594.

Kamins, S. (1985): Instruction Manual Q & A. Version 1.0. Document No.: 1-001. Cupertino: Symantec.

Kobsa, A., J. Allgayer, C. Reddig, N. Reithinger, D. Schmauks, K. Harbusch, W. Wahlster (1986): Combining Deictic Gestures and Natural Language for Referent Identification. Technical Report, SFB 314, Dept. of Computer Science, University of Saarbrücken, West Germany (forthcoming).

Tennant, H. (1980): Syntactic Analysis in Jets. Advanced Automation Group Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Working Paper 26.

Texas Instruments (1985a): Explorer™ Natural Language Menu System. Data Systems Group, Technical Report No. 2533593-0001, Austin, Texas.

Texas Instruments (1985b): Natural Language Menu User's Guide. Digital Systems Group, Dallas, Texas.

Thompson, C.W. (1984): Using Menu-Based Natural Language Understanding to Avoid Problems Associated with Traditional Natural Language Interfaces to Databases. Texas Instruments, Computer Science Laboratory, Technical Report No. 84-12, Dallas, Texas.

Wahlster, W. (1984): Cooperative Access Systems. In: Future Generations Computer Systems, Vol.1, No.2, 103-111.

Wahlster, W. and A. Kobsa (1986): Dialog-Based User Models. In: G. Ferrari (ed.): Special issue on natural language processing. IEEE Proceedings.

Webber, B.L. (1986): Questions, Answers and Responses: Interacting with Knowledge Base Systems. In: Brodie, M., J. Mylopoulos (eds.): On Knowledge Base Management Systems. New York: Springer.

Woods, W.A. (1984): Natural Language Communication with Machines: An Ongoing Goal. In: Reitman, W. (ed.): Artificial Intelligence Applications for Business. Norwood: Ablex, 195-209.