WRITING A NATURAL LANGUAGE DATA BASE SYSTEM*

David L. Waltz and Bradley A.Goodman
Coordinated Science Laboratory
University of II]inois, Urbana, Illinois 61801

Abstract

We present a model for processing English requests for information from a relational data base. The model has as its main steps (a) locating semantie constituents of a request; (b) matching these constituents against larger templates called concept case frames; (c) filling in the concept case frame using information from the user's request, from the dialogue context and from the user's responses to questions posed by the system; and (d) generating a formal data base query using the collected information. Methods are suggested for constructing the components of such a natural language processing system for an arbitrary relational data base. The model has been applied to a large data base of aircraft flight and maintenance data to generate a system called PLANES; examples are drawn from this system.

1. Introduction

The language processing model for the PLANES system for natural language access to a large data base [9,18-21] has evolved over the past two years to a point where we feel that it is now practical to begin constructing such systems for other data bases. Key ideas and assumptions in the model are described below.

The model is designed to handle requests by real, casual users, whose only programming language is English, but who have some knowledge of the material in the data base.t We have assumed that users will ask questions which are often ungrammatical, which include many abbreviations, both standard and non-standard, and which use ellipsis (omission of information to be understood in context) and pronouns extensively. (See Malhotra [13] for ideas about the types of things users are likely to type in.)

The model is designed to work with a relational model |3,8], Data is viewed as being divided into relations which correspond to files or sets of files in conventional data base terminology. Each relation contains a collection of tuples which correspond to records; each tuple contains one or more domains or fields. A relation can conveniently be thought of as a table, with each row being a tuple and each column a domain. There are two important reasons for using the relational approach:
(1) The relational approach stresses data independence. This means that the user and front end programs are effectively isolated from the actual data base organization. We are now working with only a small subset (approximately 10" bits) of a much larger database (approximately 10_{11} bits); if we were to use our front end with the entire-data base, the data accessing programs would have to be modified, hut, using the relational model, changes need not affect the "data model" seen by users and the natural language front end.
(2) Many data bases arc already internally organized in a tabular form, and are thus naturally suited to a relational data model. We are not in this paper concerned with updating and normalization aspects of the relational data bases.

Most of the examples given in this paper are drawn from the world of the PLANES system [9,18-21j. PLANES is a working system which operates on a data base supplied by 3-M (Maintenance and Material Management), Mechanicsburg, PA. The data base is comprised of detailed flight and maintenance data, plus data summaries, and is organized by month, aircraft, and type of data (scheduled maintenance, unscheduled maintenance, flights, etc.).

2. Key Ideas

Probably the most important contributions of our work are the overall organization of the model, and ideas on how to generate each component of the model for a novel data base. We have attempted in our work to attack on a broad front and integrate solutions to many practical problems including operating speed, dialogue and paraphrase generation, spelling correction and error recovery, browsing and answering of vague questions, answer generation, automatic HELP files, etc.

2.1 Central Assumption

The central novel assumption underlying the model is that a data base request is uniquely determined by the set of semantic constituents in a clause, independent of the order of the constituents.# We need only sufficient grammatical correctness to recognize the phrase boundaries of semantic constituents and clause boundaries (if any). Thus the modei handles grammatical English, "pidgin English," or ungrammatical lists

*Thc research described in this paper was supported by the Office of Naval Research under Contract Number N00014-75-C-0612.

tin the PLANES systems, we have provided easily accessible HELP files to bring a user without data base knowledge to a point where he can use the rest of the system.

There are important exceptions, for example comparative constructions such as "...did plane 3 have more flights than plane 2..."where the order of "plane 3" and "plane 2" in the sentence is important.

of semantic constituents with equal ease. To our knowledge the use of a method such as this for data queries is original with this project.

Given our central assumption, the processing of a user's query involves primarily identifying all the semantic constituents, e.g. (for our data base) time period, plane type, serial numbers, maintenance types, etc. Each of these semantic constituents can be a variable; (i.e. the name of a field for which the system is to find values), or a constant (i.e. a value for a particular field which a data item must satisfy), or a set of constants (i.e. a set of values or range of values). Thus, in "Which planes had engine maintenance in May 1973?" planes is a variable and engine maintenance and May 1973 are constants. An operation on a variable (e.g. "sum of flight hours") functions as a constant. The identification of semantic constituents is handled by a group of ATN subnets, each of which is an "expert" on recognizing dif-- ferent ways of expressing one type of semantic-constituent; thus, there are subnet experts for time period, planetype, etc.

2.2 Concept Case Frames and Context Registers

A second major set of ideas is necessary to handle the following problem: often a user's request will omit information necessary to form an adequate query. This can happen because! (a) a user leaves out information meant to be understood in context (this is called ellipsis), (b) a user uses a pronoun in place of a named constituent, or (c) a user simply neglects to include all the' necessary information. To handle these situations we- use context registers together with concept case frames. Context registers are history keepers; they store all semantic constituents of requests along with answers to earlier questions and other information.

Each concept case frame is a template representing a whole series of questions about the data base. The templates are used as tools to "build" a legal query by filling in the mandatory "slots" of the template with information extracted from the current request, earlier requests, world knowledge or default values. If the set eif semantic constituents found in a user's request does not match a concept case frame exactly, we can look back through past context register values to fill in missing elements (as with ellipsis and pronoun reference), ask the user to pick the appropriate meaning if there is more than one possible way of filling in the concept case frame, or note that a join (combining of more than one relation) is necessary if there are. elements left over after matching any single concept case frame. Certain concept case frames also correspond to requests for HELP files, or note that declarative information is being input (e.g. "From now on consider only plane 3."), or note that the request requires special processing, as in the case of a vague question or one requiring alerting functions. A good way to begin enumerating concept case frames is to look at the domains (field names) for each relation.

2.3 Query Generator

The third important set of ideas is concerned with query generation. The system can generate a formal data base query from the set of semantic constituents whether or not a concept case frame was matched.

The first step is to decide what to return as an answer, and to specify what patterns "hits" must satisfy. To do this, the query generator first notes which constituents are constants or sets of constants and which are variables (see above). Often there will be only one variable, denoted by a question word (e.g. which, when, where, etc.); all variables and sets of constants must be part of the answer returned. The constants and sets of constants are treated as predicates, i.e. as field values against which individual data items or "tuples" must be cemipared to find which are "hits".

The second step is to decide which relations to search. There are about 600 relations in our data base, each corresponding to one month's detailed data of a particular type (e.g. flight-hours) for one particular plane, or to summary data for planes of one class for various periods, and so on. There are also temporary re1ations which are created as a dialogue progresses, and the system must be able to determine when these are being referred to (as in "Of these, ..." and other folleow up questions). Because there is both summary and detailed data, and several types of data for each plane, the? decision of the query generator requires some processing. The[1] query generator intersects the sets of relations referred to by each semantic constituent, in the hope that only a single relation will remain. If no relations remain, then a join (or joins) is required and additional processing is necessary; if more than one relation remains, then heuristics can often help select one (e.g. use summary data rather than detailed data if either is possible) or the user can be offered a choice among the alternatives.

As its third step, the query generator must decide on how to sort the hits when retrieved. Sometimes sorting is specified in queries (e.g. "Give me' total maintenance hours for plane 3 by month"); otherwise it is determined by a heuristic priority scheme.

3. Model Operation

The processing of a user's request is divided into four main phases: parsing, interpretation, evaluation, and response.

(1) In the parsing phase, a set of semantic features with values is formed from the user's request. As part of this phase pronoun reference and ellipsis are resolved, and complex questions are broken up into a sequence of simple queries.

(2) In the interpretation phase, the feature and value representation of the user's request is translated into a 'program,' called a "data base query", to generate the data to answer the request.

The bulk of this paper is devoted to explaining these first two phases of the model's operation.

(3) The evaluation phase uses the query generated in the previous stage to search the data base and return the needed data.

(4) The evaluation portion passes the. resulting data to the response generator.

At each stage of the process, the results are sent to context registers, which consist of a set of stacks of relevant information. These stacks contain the results of each stage (e.g. user's request, paraphrase, etc.), syntactic components (e.g. subject, object, etc.), and semantic/contextual information (e.g. time specification, etc.). This information is made available for later resolving of anaphoric reference, supplying phrases deleted through ellipsit;, and generating responses.

3.1 Parsing

The first phase, "parsing," actually covers several operations. These include

(a) cleaning up the input (correcting spelling, substituting canonical words and synonyms, etc.);
(b) applying the semantic ATNs to the input request and filling in context register-values ;
(c) b re a k in g up questions with e nib e d d e d clauses into two or more "simple queries";
(d) applying concept case frames to "simple queries," resolving ellipsis, pronoun reference and questions involving multiple relation searches.

These operations are explained in more detail in the following sections.

3.2 Cleaning Up the Input

The parser first checks to make sure that each word of the input is known by the system. Roots and inflection markers are substituted for inflected words, canonical words are substituted for synonyms, and single words are substituted for certain phrases (e.g. "USA" for United States of America"). If a given input word cannot be found in the dictionary, then the spelling correction module is called. This module attempts to find dictionary entries "close" to the input word using methods described in |17]; if one of these candidates is correct, it is inserted in place of the misspelled word. If no candidates are found, or if the user rejects all the suggested candidates, a word adding module can be called to try to add the user's word to the dictionary by finding a synonymous word or phrase already known to the system. The user can also tell the system to ignore the word and continue.

3.3 Applying Semantic ATNs

This section together with the next two describe the heart of the language understanding process. It is here that pronoun reference and ellipsis are resolved, and here too that much of the overall programming effort for the system has been expended. The processing in this portion is handled by subnets.

Each subnet is an ATN [23] phrase parser which matches only phrases with specific meaning. For example, in the PLANES world there are subnets for each different semantic object: plane type, date, time period, malfunction, maintenance type, aircraft component, etc. Some examples of phrases with the subnet for "time period" would match are: "between Jan 1 and Feb 28 1972," "1972," "during February and March," "then," and "in the first six months of 1972." Most subnets match noun phrases or prepositional phrases. The construction of subnets is based on Winograd's analysis of noun phrase Quantifiers (e.g. "first," "rest," "more than," "largest," etc.) are handled by a special subnet as are qualifiers (e.g. the italicized words in the phrase "A7s which crashed in May"). Subnets also check for compounds (e.g. "planes 3 and b," "plane 3 or plane 5," etc.), and recognize verb phrases.

Subnets are applied to the Input request one after another. When a subnet matches a phrase, that phrase is saved along with information on which subnet matched it, and attention is shifted to the next portion of the request. Also as part of this phase, "noise words" are matched by a subnet and essentially discarded. "Noise words" refer to phrases like "please tell me," "can you tell me," "would you let me know," "could you find," etc.

Whenever subnets match a phrase, they set the value of a corresponding context register, which acts as a history keeper. Context registers are; used for pronoun reference and ellipsis; if some item(s) in a request have been left unspecified or replaced by pronouns, context register values from previous request are used to supply the missing information or the referent of a pronoun. There are also context registers for the last request, last paraphrase, last query language form, and last answer. Context registers are implemented as stacks; which are pushed down with each new request. Thus an earlier context could be retrieved from a user statement like "A while ago we were talking about skyhawks," by looking back through context registers until a planetype context register value equal to "skyhawk" was found, and then restoring all the other context register values current at that time.

At the end of this phase we arc left with a set of representations of the semantic contents of the phrases in the sentence and a list of the order of the constituents. Unless certain constructs (such as comparatives, e.g. "...greater than...," or embedded quantified clauses) are present, the order of the phrases is ignored. This means that passive, active,and "pidgin English" requests are all represented identically from this point of processing onward. Given this sort of processing, pronoun reference and ellipsis become a relatively difficult task. The parsing of a pronoun does not result in its attachment to a particular semantic category. Ellipsis may be

taking place but it is hard to be sure." Our handling of these issues is described in the next section.

Specific subnets recognize requests for HELP information, and draw HELP files in directly.

Embedded clauses are handled by subnets; this process is described in section 3.9 below.

3.4 Concept Case Frames

Concept case frames enumerate the patterns of questions understood by the system (and can also associate data base query skeletons with question types which cannot be properly handled by the query generator). Our concept case frames are somewhat different than ordinary case frames [2]; each concept case frame consists of the act (typically related to the verb) and a list of noun phrases (referred to by subnet/context register name) which can occur meaningfully with the act. Unlike ordinary case frames, we do not store information about the role (e.g. agent, patient, instrument) played by the various phrases, and each act covers a number of related verbs (e.g. "fly," "log" and "record" map into the same act). Phrases which would have to occur in every concept case frame, such as "time period," are omitted from the internal representation of concept case frames. Together, the subnets and concept case frames form a "semantic grammar" very similar to that used in SOPHIE ['I].

Whenever constituents of a sentence are missing (as in ellipsis) or replaced by pronouns or referential phrases, the model is able to suggest what type of phra.se is necessary to complete the concept by finding all the concept case frames which match the rest of the sentence. If only one concept case frame matches, we are done; if more than one matches, then reference to which constituents were present in the previous sentence is usually adequate to decide among candidates; otherwise, the user can be given the set of possibilities from which to choose the appropriate referents for each phrase. Furthermore, the system can guess that sentences like "How many malfunctions logged more than 10 flight hours" are meaningless because all phrases are recognized but no matching concept case frame exists.

More specifically, the matching is done as follows. After the best concept case frame (i.e. the longest one that matches the most semantic-categories of the phrases) has been chosen and as many slots in the template as possible filled in, pronoun reference and ellipsis must be resolved. If all mandatory slots are filled and no pronouns occur, everything is already resolved.

*0ne clue for ellipsis is finding a question word followed by an action. This sometimes indicates that it is occurring--namely the noun phrase normally expected in that position is missing. The probable ellipsis can be tagged so that further investigation can be taken on later.

Pronouns are resolved by noting which semantic category slots are left over for them. Pronouns can be replaced with items that fit that same semantic category by scanning backwards through the context register values for earlier sentences. When ellipsis and pronouns occur at the same time, it may be impossible to decide which of two or more slots to put a pronoun into. If the frame contains no optional slots, it makes *no* difference where the pronoun is put because the system will have to fill in any other empty slots before proceeding. Should optional slots be included, the task becomes more difficult because the pronoun may go in a required slot or an optional slot. If it should go in an optional slot, the-ellipsis must be occurring for the required slot. If it fills the required slot, then ellipsis may or may not be occurring for the optional slot. The second case is the one that causes a problem- we must know if ellipsis is occurring to be able to extract the full meaning of the query. We resolve this problem by assuming that ellipsis is occurring, and scanning the very recent set of past queries (say the last one or two sentences) to see if the context built up confirms our hypothesis (i.e. see if can find anything to fill the optional slot). If nothing fits the semantic category of the slot, the ellipsis is overruled and the hypothesis dropped. Otherwise we fill in the slot with the information found. If all else fails, we can ask the user to select the appropriate interpretation from among a set of hypotheses. (Being able to choose from among a small set of possibilities is still much simpler for the user than rephrasing!)

Ellipsis alone is a little easier to handle. Any time information is missing from required slots we know that ellipsis is occurring. The past queries can be scanned backwards for information to fill the slots. If we do not find enough information to fill all the slots we can ask the user for the required information. When e11ipsis of optiona1 slots is occurring, we must use a set of heuristics to give us a clue that the omission of a phrase has occurred. Earlier we mentioned an example of such a heuristic-namely finding a question word followed by a verb other than the verb to be without any intervening noun phrase-.

3.5 Construction of a Query

The filled-in concept case frame is next translated into a formal query expression for use with a relational data base system.t The translation involves:
(1) selecting the relations (files and card types) to look at in order to retrieve the information necessary for answering the user's request;
(2) deciding what domains (data fields) to return from the relations which are searched. (In general, more fields are returned than are actually

+ While we have not done so, it should be possible to write similar query generators for other data models.

asked for. For example, if asked about which planes had engine maintenance during some time period, PLANES returns not only the plane identification numbers, but also the dates of maintenance and codes for the exact type of maintenance.);
(3) deciding how to arrange the output data. Typical orderings are by increasing or decreasing size of some field value (like "number of hours down time") or sequentially by date. Other more complex orderings may be specified by the user (e.g. "List maintenances for plane 5 by month.);
(4) deciding which operations should be performed on the fields returned. Examples of operations include list, count, average, sum, and find largest;
(5) translating field values (e.g. for dates, plane types, or actions) into internal data base codes.
(6) Most important, organizing all this material into an expression in the relational calculus [4, 5,8] which can be used to implement the actual data base search.

The general process was described briefly in Section 2.2. For a simple query (one involving no joins, comparatives, listing by special grouping (e.g. by month, plane serial number, repair location, etc.), and no special quantification.) this process involves (a) finding the question phrase(s) (e.g. which planes in "which planes flew more than 10 times?"), (b) inserting it in the answer slot of the general query skeleton,' (c) interpreting other semantic phrases and values (constants and sets of constants) as predicates, (d) inserting these in the general skeleton (e.g. adding (GT FLIGHTS 10.) to the list of predicates, given the question above), (e) completing the answer-description portion of the skeleton with other field values, using both heuristic knowledge about meaningful answer forms and any special user instructions (e.g. "Plot...", "List...", etc.), and (f) deciding upon and filling in a answer sort specification (e.g. by increasing serial number, in time sequence, by groups according to portion of the plane repaired, etc.).

The query construction is more complex for quantified expressions (e.g. "Find all repairs for the 5 planes with the most flight hours), comparatives (e.g. "Did plane 5 log more flight hours than plane 3?"), questions with embedded clauses (e.g. "Which planes that crashed in May had engine maintenance in April ?") and requests involving special operations (e.g. 'Vhich plane flew the most hours in May?").

Of particular interest is the method by which relations to be searched are selected. The system looks at each phrase separately, and notes which relations the phrase could possibly belong to. Some phrases (like plane type and date) are not very useful for this process, since they appear in most relations, but others (like flighthours) appear in only one or two relations. All the relations possible for a clause are then intersected, and if a single relation is selected, the process of selecting a relation is complete for the phrase. Clauses are then considered in pairs, and so on. If more than one relation or a set of relations remains, then the request is ambiguous, and priority scheme or a dialogue is necessary to select the appropriate relation. If no relations remain at any intersection step, then more than one relation must be searched to answer the request, and the results of these searches must then be combined via the relational operation called Joining (constructing a single relation from two different relations). As an example, the request: "Find all planes which had engine maintenances on the same day as a flight" would require searching the maintenance and flight relations, and then joining these relations via the date and plane domains by intersecting the sets of tuples for maintenance and flight and retaining tuples with identical planes and dates.

*The general query skeleton consists of an ordered set of slots for quantification, locally bound variables, answer format, predicates to be satisfied by "hits," and an answer sort specification.

3.6 Constructing a Paraphrase

An important part of any query system's operation is allowing a user to verify whether or not the system has correctly understood his request. To this end, the. system feeds back its understanding of the request, with pronoun reference and ellipsis resolved, for the user's approval. The paraphrase is straightforwardly constructed from the formal query, and any special information associated with the matched concept case frame. "Special information" includes query language skeletons, calls to HELP files, and special functions, such as statistical comparison functions, needed to answer complex questions.

If the user does not approve of the interpretation of his request, he can enter into a clarifying dialogue with the system [6], The system asks whether the user wants this query executed on the data base, if he wishes to continue with the current sentence as context (this is useful for correcting minor errors, e.g. typing the wrong year), or with the previous sentence as context. As a simple example, suppose a user wanted data (for a previously specified question) for January 1972, but (using ellipsis) typed "January 1973" instead. It is simple to correct this by typing "n" when asked by the system "Shall I execute this query on the data base...y or n?", and then simply typing "1972". The system will recognize this as a year, substitute it for 1973, and the user can then have the corrected query executed. Clearly, minimum typing and no remembering of special commands is required for this sort of error correction.

3.7 Retrieving the Data

The query expression generated is expressed in the data sublanguage ALPHA [4], as implemented in LISP by Green [10]. This expression is used by the relational data base system to construct the actual program which retrieves the data. In order to construct the search program, the system

must :
(1) select the files to be searched;
(2) select an order for searching these files;
(3) generate an expression for testing and selecting tuples values to return while searching;
(4) generate a program to combine data, possibly from a number of different relations, so that the proper answer will be returned.
(5) decide when to save the results of a search for future use. This is important in interactive querying, since interesting results can be expected to evoke follow-up queries from a user, and such queries are likely to reference tuples just retrieved.

3.8 Generating an Answer

Once the data has been retrieved, the results are passed to the output module, which decides on an appropriate, display format for the data. If possible it attempts to produce a graph. This can only be done if (1) pairs of items are returned, (2) one item is numerical, (3) the number of items returned is small enough (but not too small) to produce a reasonable graph which will fit on a CRT screen. If a graph is not possible, the system will produce a list or table; if there is too much data to fit on the screen, the results will be automatically output to the line printer.

3.9 Embedded Clauses

Qualifying phrases or qualifiers (see Winograd *[22]*),constitute the most common type of dependent clause. Examples of qualifiers are the underlined parts of "planes which crashed in May," "maintenance performed on A7s," and "planes with poor maintenance records." Qualifiers appear after the main noun in a noun phrase, and are often introduced by relative pronouns such as "which" or "that", or by verb forms ending in -ed or -ing. Prepositional phrases can also serve as qualifiers.

Qualifiers can be found by applying a qualifier subnet to the portion of a request following the main noun of a noun phrase. Because qualifier syntax is fairly restrictive, in many cases merely examining the single word after the main noun may suffice to preclude the presence of a qualifier. If a qualifying phrase or clause may be present, the following actions are taken:
(1) A syntactic parser (based on Woods' LSNL1S parser [24]) is invoked to find if a qualifier is present, and if so what its boundaries are.
(2) If the embedded clause is not grammatical, heuristics are invoked to attempt to bracket the clause.
(3) Once a qualifier is found to be present, processing is suspended on the current clause, and the current context register values are pushed down.
(4) The main noun from outside the qualifying phrase is substituted for the relative pronoun (if any) or is inserted as a phrase element in the qualifier.
(5) The qualifying phrase or clause is processed like a normal request, with the main noun from the clause above serving the role of the requested item. Note that verb forms get changed to a root plus an inflection, so that the exact verb form does not affect this processing. Prepositions as well as verbs can refer to certain case frames, so that, for example, "planes with poor maintenance records" has the same meaning to the system as "planes having poor maintenance records."
(6) The query corresponding to the. qualifier must be integrated with the query corresponding to its surrounding clause to form the overall query. The ordinary meaning of qualifiers seems to suggest that the qualifier query be evaluated on the data base first, and its result should then be used as the scope of search for the other clauses, in fact, either search can in general be performed first.*

Notice that request can involve searching more than one relation, even if there are no qualifiers or dependent clauses, as in the sentence: "Did any planes have a flight on the same day as an engine maintenance?" In this case the relation for flights and the relation for maintenances must both be searched, and the results joined with respect to plane and date values.

3.10 Adding New Questions

Extending system competence within the model is particularly easy although for the most part it must be done by a programmer. To add the ability to handle a new type of sentence, one must only add a concept case frame which expresses that sentence. Variations on this sentence, including active and passive forms, ellipsis, different phrase orderings and the addition of noise words can all be handled with no additional machinery, provided that the proper relation (or relations) is automatically selected by the translation mechanisms discussed above. The addition can be handled in ordinary user mode. If a novel request is encountered (i.e. one which does not match any concept case frame), the query generator can still attempt to handle the request, and feed back a paraphrase to the user. If the user approves of the paraphrase, the request can be added to the concept case frame list. The request can be added in a general way only if there were no pronoun reference and no ellipsis in the request--if there were, the concept case frame generated would be incomplete. If special instructions are necessary, these are attached to the concept case frame but this process probably will involve a programmer for the foreseeable future.

Extending the subnets is also fairly easy; we have written a net editor (described in [20]) which takes a phrase and adds the states and arcs to match this phrase, to a specified subnet, using a minimum number of new arcs and states. Once a

PLANES estimates temporary storage required for each query, and selects the query with minimum requirements to search first. The storage estimates are made on the basis of statistical information stored for each file. Query construction is discussed in more detail in [20] and [10].

new phrasing has been added to a subnet's repertoire, the phrase can of course be matched in any sentence context. To add a new phrase in user mode, the new phrase must be synonymous with some phrase already known by the system.

4. Evaluation of the Model

To our knowledge, no other model proposed or implemented attempts to deal with ungrammatical input; any scheme based on sentence parsing (e.g. Woods et al. [24]) must be restrictive, even if certain ungrammatical constructs are allowed (Woods, for example allowed single noun phrases as requests, interpreting them and "Find <noun phrase>."). The SOPHIE system [1] uses a semantic grammer similar to that in PLANES, and is thus at least conceivably able to deal with non-grammatical input.

A number of other systems have dealt with pronoun reference and ellipsis, including (at least) LUNAR [24], SOPHIE [1], SHRDLU [22], RENDEZVOUS [6], NLPQ 1l], and LIFER [12].

Much of the stimulation for generating our model came from reading Codd [6], and some of the. query generator ideas are similar to those expressed in Sowa [16]. Some language processing ideas were inspired by PARRY [7],

The amount of computation required by the model is relatively modest. PLANES typically requires 1-4 seconds to parse a sentence and generate a formal relational query. Total processing time depends critically on the amount of data to be searched.

Plans for the near future include extensive testing on real potential users.

References

1. Brown, J. S. and Burton, R. R. Multiple representation of knowledge for tutorial reasoning. In Bobrow, D. G. and Collins, A. (ed.) Representation and Understanding, Academic Press, New York, 1975, 311-349.
2. Bruce, B. Case systems for natural language. Artificial Intelligence 6, 4 (Winter 1975), 327-360. .
3. Codd, E. F. A relational model of data for larged shared data banks. Comm. ACM 13, 6 (June 1970), 377-387.
4. Codd, E. F. A data base sublanguage founded on the relational calculus. Proc. ACM-S1GFIDET Workshop on Data Description, Access and Control, Nov. 1971, ACM, New York, 35-68.
5. Codd, E. F. Relational completeness of data base sublanguages. Courant Computer Science Symposium 6: Data Base Systems, Prentice-Hall, New York, 1971, 33-64.
6. Codd, E. F. Seven steps to RENDEZVOUS with the casual user. Proc. IFIP TC-2 Working Conf. on Data Base Management Systems,(April 1974) North-Holland Publ. Co., Amsterdam, 1974, 179-200.
7. Colby, K. M., Faught, B., and Parkison, R. Pattern-matching rules for the recognition of natural language dialogue expressions. Stanford AI Lab. Memo AIM 234, June 1974.
8. Date, C. J. An Introduction to Database Systems, Addison-Wesley, Reading, MA, 1975.
9. Gabriel, R. P. and Waltz, D. L. Natural language based information retrieval. Proc. 12th Allerton Conf. on Circuit and Sys. Theory, Univ. of 111., Urbana (Oct. 1974), 875-884.
10. Green, F. R. Implementation of a query language based on the relational calculus. M.S. thesis, Dept. of Computer Science, Univ. of 111., Urbana, Oct. 1976. (CSL Rpt. T-38).
11. Heidorn, G. E. Natural language inputs to a simulation programming system. Tech. Rpt. NPS-55HD72101A, Naval Postgraduate School, Monterey, CA., Oct. 1972.
12. Hendrix, G. G. LIFER: A natural language interface facility. Stanford Res. Inst. Tech. Note 135, Dec. 1976.
13. Malhotra, A. Knowledge-based English language systems for management support: an analysis of requirements. 4IJCAI (Sept.1975) 842-847.
14. Palermo, F. P. A data base search problem. 4th Intl. Symp. on Computer and Infor. Tech. (Dec. 1972), Plenum Press, N.Y., 1972, 67-101.
15. Schank, R. C. Identification of conceptualizations underlying natural language. In Schank, R. C. and Colb, K. M.(ed.). Computer Models of Thought and Language. Wott, Freeman, San Francisco, CA, 1973, 187-247.
16. Sowa, J. F. Conceptual graphs for a data base interface. IBM J. Res. Develop., Vol. 20, No. 4, July 1976, 336-357.
17. Tenczar, P. J. and Golden, W. M. Spelling, word, and concept recognition. Report. Computer-based Education Res. Lab., Univ. of 111., Urbana, 1972.
18. Waltz, D. L. Natural language access to a large data base: an engineering approach. 4IJCAI (Sept. 1975) 868-872.
19. Waltz, D. L. Natural language to a large data base. Naval Research Reviews XXIX, 1 (Jan. 1976), 11-25. Reprinted in Computers and People 25, 4 (April 1976), 19-26.
20. Waltz, D. L., Finin, T., Green, F., Conrad, F., Goodman, B., and Hadden, G. The PLANES system: natural language access to a large data base. Coordinated Science Lab., Univ. of 111., Urbana, Tech. Rpt. T-34 (July 1976).
21. Waltz, D. L. An English language question answering system for a large relational data base. (Submitted for publication.)
22. Winograd, T. Understanding Natural Language, Academic Press, New York, 1972.
23. Woods, W. A. Transition network grammars for natural language analysis. Comm. ACM 13, 10 (Oct. 1970), 591-606.
24. Woods, W. A., Kaplan, R. M., and Nash-Webber, B. The lunar sciences natural language system: final report. Report No. 2378, Bolt Beranek and Newman, Inc., Cambridge, MA. 1972.