FROM MENUS TO INTENTIONS IN MAN-MACHINE DIALOGUE

Robert F. Simmons

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-84-33 November 1984
AI-TR-84-04

From Menus to Intentions in Man-Machine Dialogue

Rob't F. Simmons[1]
University of Texas, Austin, 1984

Abstract

Operating systems are designed to achieve goals not to recognize user intentions. But the use of *Help* systems and Menu-selection make them more useful and friendly. Natural Language interfaces must go farther by guessing what the user wants and what is meant but not specified. Natural language programming systems -- still in infancy -- promise explicit capability for a user to define his/her intentions explicitly. Text Knowledge systems -- at the research frontier -- bewilder us with the complexity of intentions expressed and implied. Current techniques for recognizing intentions and computing appropriate representations and responses are discussed in this paper.

Of Mouse and Menu

A modern computing system includes hundreds of operators and programs each of which has its own control language. It is a technological marvel for accomplishing my information processing intentions -- providing only that I figure out how to use its capabilities to achieve my goals. The description and documentation of such a system entails several fat volumes of difficult text, so a user may frequently settle for using the system in limited ways to avoid the extensive effort required to learn it thoroughly. The complexity of such systems easily distinguishes between the casual, specialized user who knows an editor, a programming language, a text formatter, etc. from the "wizard-hacker" who can use the system to accomplish vastly more varied operations. The wizard-hacker knows much of the operating system and numerous control languages for its multitudinous operations; he/she is an indispensible consultant and teacher in any computer laboratory.

For the best equipped modern laboratories, specialized work stations have become available -- LISP machines, SUNs, etc. -- which are large, fast microcomputers supported by megabytes of internal memory, hundreds of megabytes of local disk storage, and somewhere in the background a gigantic file server, providing trillions of megs of longterm memory, and network communications with the world. More importantly, they include bit-map displays on the order of 1000 X 1000 pixels and a "mouse" whose motion translates into movements of a pointer on the display. One or

[1]Supported by NSF Grant IST-8403028

more buttons on top of the mouse are used to select a portion of the screen (actually of the array that the screen displays) in order to focus on specific data. A most common use of the mouse is to select a choice from a menu; the pointer is guided to one of the choices that are displayed on the screen, a button is pressed on the mouse, and the system passes control to the procedure selected from the menu. In its turn, this procedure may present additional menus or actually accomplish some task. A menu is usually a box drawn on the screen containing cells with the names of the procedures that can be selected, although it may take other display forms such as a boxed border of the screen in which choices are shown.

Apple Corporation's MacIntosh is correctly advertised as a desk assistant for people who have no desire to program. With a one-button mouse and hierarchies of menus, the bit-map display presents to the user a large set of options for drawing, text preparation, editing, type-setting, display, and printing of data. One or two hour's experience with such a system is sufficient to enable most users to graduate from their typewriter to a high-technology information control device that not only replaces the printing functions of the typewriter, but adds capabilities for drawing diagrams and artwork, for storing files, for graphically displaying and charting data, and for endless creative activities. As long as the user is content with the capabilities provided by the menus, he/she is almost fully protected from asking impossible questions -- only meaningful sequences of menu choices are provided -- and the system is equally protected from user errors.

In general, the operating system, whether mouse and menu-controlled or controlled by formal language commands, provides a set of operations that a user can combine to accomplish a set of information processing goals. These goals, with careful design and a bit of luck, will represent what the system can understand of the user's *intentions*. From the users' viewpoint, it will be seen that these operations are primitives at the bottom of the human goal system. In menu-controlled systems the possible processing goals are completely pre-specified and the user need only select a permissable sequence to accomplish his/her goal if it is provided. In formal-language controlled systems, the user may be offered much more freedom in accomplishing goals within the capability of the system, but at the cost of learning several formal comand languages at the operating system and production program levels, and subject to the ever-present potential of making errors. In a sense, the menu-controlled system anticipates a limited set of users' intentions and provides capabilities to accomplish them, while the formal language system can provide a vast space of possible intentions that the user can realize -- if he/she learns the several operating and production system languages that are required.

In the menu systems, intentionality is explicit and limited; in the formal language system it is largely left to the user (or to analysts who might consider all the possible, legal combinations of operations in order to describe the system). But human intentions are much more complicated, hierarchically ordered systems of abstract and concrete goals. A MacIntosh user, for example, might have a desire to win fame and fortune by publishing a paper communicating a simple description of some hithertoo complex process. Perhaps that high-level goal might have brought him to the purchase of his new computer and to the learning of skills in its use. To write the paper this user may have read widely, invented a technique, and walked long miles while he organized his thoughts for communication. When finally, he selects MacWrite and begins to prepare his paper, a new set of intentions or goals control his behavior. He may wish to illustrate his paper with diagrams, and embellish it with special type fonts; he may wish to include some humor, he may consult dictionaries or thesauri to select vocabulary, and insure correct spelling.

In any case, the operations provided by the computer correspond to only the most primitive and concrete of human goals; so intentions, as provided by the operating system, are only the basic building blocks in achieving vastly more complex intentions by its human users. At the operating system level, computers reliably apply the programs a user calls; user intentions must be translated to program capabilities. The system helps with information and documentation files, but only rarely recognizes the users' goals.

Natural Language Interfaces -- NLI

Natural spoken and written languages play a very large role in human communications and it was completely predictable from the inception of computers that efforts would be made to use natural language to communicate with them. The limited practical problem of querying databases using English became a most attractive goal, and one that resulted in the technically satisfying accomplishment of suitable grammars for translating subsets of English to formal query languages. But immediately following the technical success of working systems, tests with user communities showed that much more was required. One of the first problems to emerge was that many users first had to learn to type and to use a computer at the operating system level. More seriously, the typical user expected that a system that "understood" English, could understand what the user meant in asking a question and, in general, expected the system to be intelligent in a human way -- since it used a human language. In addition the user usually had no way to know the limits of the system's comprehension of natural language and thus no way of restricting the vocabulary and structure of his queries to those limits. The consequence appears to be that current users of NLI find some elementary and short English forms with which the system is successful and ignore the more complex capabilities which are more prone to failure.

A recent experiment [6] evaluating the use of an NLI English interface to a database compared the success of English queries with those of queries in a simple formal query language, SQL. For this experiment a set of problems was chosen for which answers could be derived from the DB whether asked in English or in SQL. Users were required under various experimental conditions to formulate a series of questions either in SQL or English to obtain the answers that would solve the problem. The findings were entirely surprising. The users succeeded in discovering answers to only about 46% of the questions when using SQL and for about 22% using English. Analysis revealed that in SQL, when a question failed there was enough feedback to enable (sometimes) successful revision, but in English the lack of feedback as to what went wrong left the user only the recourse of random paraphrase.[2] Such experiments as this demonstrate that the human factors problems associated with NLI are crucial; technical linguistic success is only a beginning.

Tennant [10] first studied human intentions with regard to natural language query systems and noted how poorly the NLI fitted expectations -- or how unrealistic user expectations were with respect to the capabilities of NLI. He and Thompson [11] adopted menu-control as a possible solution for the problem. In their approach, the user selects an option to query or input data; at that point a menu shows how a command can begin. Selecting an option causes a new menu to appear showing choices for possible continuations. This method insures that the user remains within the English subset and that his queries remain within semantic and pragmatic bounds of the system. Use of a mouse or voice selection largely eliminates typing problems. All in all, the menu-driven NLI results in error-free use, accompanied by satisfactory feedback showing the user the resulting translation to a simple formal language. The authors note, however, that the method will become extremely cumbersome if the NL subset becomes very large. A reader may also notice that the style of the English they use is already reminiscent of the flavor of database formal languages.

We can notice here that a menu containing natural language queries is a decision tree; beginning with say, "How many", branches exist for all categories in the database. For each such category, continuations for further qualifications and constraints are provided as shown below.

```
HOW MANY -- EMPLOYEES ----with-- QTY --CHILDREN
         -- MANAGERS         |-- SALARY of--QTY
         -- SALESMEN         |-- SENIORITY of
                                 --QTY -- years
         -- PARTS            |-- etc.
         -- ASSEMBLIES
         -- etc.
```

The first menu offers EMPLOYEES, MANAGERS, etc. When EMPLOYEES is chosen, the *with* choices appear as shown. If PARTS had been the choice, different continuations would have been presented.

[2]It is important to mention that the version of the NL system used in this study was not fully debugged, and that the computer facility used was sporadically available. Generally such studies show that NL questioning is about 70% satisfactory.

Descriptive Statistics for Study

Subjects	20	
Problems	39	
Tasks		
NLS	42	
SQL	45	
Total		87
Queries		
NLS	656	
SQL	425	
Total		1081

SQL - Formal Query Language
NLS - English Interface

Results:

TASKS	NLS	SQL
Essentially correct	17.1	44.2
Partially solved	34.2	23.3
Not Solved	48.7	32.5
QUERIES		
Essentially correct	22.3	45.6
Correctable	75.5	57.0

Conclusions:

SQL CURRENTLY SUPERIOR TO NLS
NLS BRIEFER THAN SQL
NLS LACKS FEEDBACK TO USER

Subjects performed more poorly in both languages than laboratory studies predicted.

Figure 1:
EVALUATION OF NATURAL LANGUAGE FOR DATA RETRIEVAL

M. Jarke, *et. al.*[6]

Many years ago a noted researcher was laughed out of IBM research for supporting a machine translation system based on an exhaustive list of phrase equivalences for two natural languages. It was obvious that the language used in ordinary texts was too large to yield to such an approach. For NLI applied to database and operating system control, this exhaustive approach is, for the moment at least, more successful because the command languages into which the NL is translated are still relatively small and grammars are used to form economical descriptions of the NL subset.

Another human factors problem concerns the nature of the NLI answers to queries. Obvious to most researchers, the answers should be in a readable NL form, and so they have been in most systems. Not so obvious until studied by Kaplan [7] and Allen [1], is the requirement that the system estimate the users' intentions in asking the question and so respond appropriately. Allen used a train station example like the following:

Query: Is there a train to Zurich?
Response: Track 5 at 2100.

The clerk might have responded directly to the question with the answer, "Yes", but then the user would almost certainly have asked about time and location. The clerk, anticipating the questioner's intention to take or meet the train, returned a cooperative response providing the additional information. If there were no train to Zurich, the clerk might have responded,

"No, but an express bus leaves the bus station
at noon daily."

In this event the clerk again is cooperating by anticipating the questioner's probable intention. Computational methods for dealing with intentions at this level are only beginning to be understood.

Kaplan emphasizes the importance of corrective responses in a database environment. In asking a question to a student database, the questioner may have certain misapprehensions. His example dialogue is paraphrased below:

Q: Which students failed CS135 in Spring 1984?
A: None.
Q: Did anyone get a D in CS135 in Spring 1984?
A: No.
Q: How many students passed CS135 in Spring 1984?
A: None.
Q: Was CS135 offered in Spring 1984?
A: No.

Aha! How much better if the system recognizes our misapprehension and immediately informs us that the course was not offered. Kaplan's computational solution is simple and attractive. Suppose the data structure has the following pattern:

(1:STUDENT 2:GRADE 3:COURSE-NAME)

The following failures are possible:

1: No such student
2: No such grade
3: No such course
1-2: No such grade for this student
1-3: No such course for this student
2-3: No such grade in this course

Reporting one of these reasons for failing a question serves to correct user presuppositions.

Programming in Natural Language, NLP

In the early history of computing, *automatic programming* meant the development of assembly languages that allowed the programmer to use English mnemonics for binary operation codes, and octal or decimal numbers to refer to computer-memory locations. Soon the meaning changed to refer to programming in higher level languages such as Fortran or Lisp. But that reference also changed to the present (confused) notion that automatic programming includes any method for specifying to a computer the requirements of a program in such a manner that it could be automatically compiled, somehow excluding the ordinary methods of specifying the structures and operations directly in a compilable programming language. The current notion seems to be to state what a program should do and allow a very smart compiler to determine the "how". Specification languages of choice include input output arrays, first order logic, and occasionally English and other natural languages.

Several experimental systems have been reported that allow a user to write programs in a strictly restricted English subset. Two recent examples include van Baalan's dissertation [12] in which recursive Lisp functions were compiled from English descriptions, and Biermann and Ballard's [3] experiments comparing the efficacy of programming array-manipulation problems in an English programming-subset and in PLC. The latter study reported the challenging finding that sophomore level programming students were able to program these operations faster and with fewer mistakes in English than in PLC. I take this report as a curiosity much in need of additional confirmation and of further study of conditions under which the findings hold.

As in any programming language, defining a system of programs in English allows the programmer to organize and specify the intentions which are to be achieved by the program. In the preceding section we saw that those intentions might include predictions of the intentions of the user, if for example, the program is to simulate an information clerk at a train station. But the use of an English subset as the programming vehicle suggests that the English compiler may be heavily burdened with the necessity for guessing what the programmer may mean by what he/she did not say directly; English uses pronouns, noun phrases, and elliptical references as ordinary parts of its structure and we would consider an English programming-subset sadly lacking if it did not recognize these. For example in a programming experiment by Yeong Ho Yu [13], translating from an English subset to procedural logic as a programming language, objects in the robot blocks world are defined as follows:

```
There is a table and a robotarm.
[TABLE ISA TABLE]
[ROBOTARM ISA ROBOTARM]
A white block is on the table and a blue block is on
the white block.
[B1 ISA BLOCK][WHITE B1][ON B1 TABLE]
[B2 ISA BLOCK][BLUE B2][ON B2 B1]
There is a red block on the blue block.
[B3 ISA BLOCK][RED B3][ON B3 B2]
The block is a pyramid.
[B3 ISA PYRAMID]
```

In these sentences, the compiler must accomplish the easy task of recognizing that "the table" co-refers to the object "a table" refers to, and similarly recognize the repeated references to "blue block" and "white block". In the last sentence the interpreter must decide that "the block" refers to the red one. In this case, since "the block" is a pyramid which cannot support any block, the conclusion is certain. But if the last statement were "the block is large" we could not determine whether the red or blue blocks were intended without further information.

In another example, Yu states the English rule,

It is always the case that the table is larger
than anything.

The phrase "It is always the case that" is recognized as one way of introducing a rule with no antecedents and "anything" is recognized as a free variable. The translation to a procedural logic rule is:

```
[LARGER TABLE X] <
```

a universally true assertion over all possible values for X. (Note that "<" represents a backward pointing logical implication sign, and that variables to its left are universally quantified.)

Another English rule states,

If a block is larger than another,
the other is smaller than it.

Here, "a block" is interpreted as a free variable, say X; "another" is a free variable, Y; "the other" is coreferential with "another" so it is replaced by Y; and "it" is coreferential with "a block" and so replaced by X; resulting in the translation to a procedural logic rule as follows:

```
[SMALLER Y X] < [LARGER X Y]
```

A more general system could attend to the type, "block" and compile<

```
[SMALLER Y X]
  < [BLOCK X][BLOCK Y][LARGER X Y]
```

What is notable here, is that procedural logic uses variables in a manner not dissimilar to some uses of English pronouns and noun phrases, thus facilitating the translation from English.

Describing robot commands to the blocks world stretches our use of English, requiring some programming jargon. Consider the following definitions:

A block is clear if it is a supporter unless something is
on it.

```
[CLEAR X]
 < [SUPPORTER X] [UNLESS(ON W X)]
```

{Note here, that any variable occurring exclusively to the
right of the backward-implication sign is existentially
quantified.}

A block is a supporter unless it is a pyramid.

```
[SUPPORTER X]
        < [UNLESS (ISA X PYRAMID)]
```

A block is clear if something is on it and that
something is put on the table.

```
[CLEAR X] < [ON W X][PUTON W TABLE]
```

A block is picked up if the block is clear, and
if the robotarm is clear, and if it is deleted
that the block is on something, and if it is
asserted that the block is on the robotarm.

```
[PICKUP X] < [CLEAR X][CLEAR ROBOTARM]
    [DELETE [ON X Y]]
    [ASSERT[ON X ROBOTARM]]
```

A block is put on X if the block is picked up
and if X is clear and if it is deleted that the
block is on the robotarm and it is asserted that
the block is on X.

```
[PUTON Y X] < [PICKUP Y] [CLEAR X]
     [DELETE (ON Y ROBOTARM)]
     [ASSERT (ON Y X)]
```

The concepts "clear", "supporter", and "robotarm" are jargon defined only with respect to the blocks world. The explicit use of multiple "ands" and of the concepts "unless", "deleted", and "added" are required in the subset to simplify the parsing and translation process. The passive form of sentences is chosen to make the descriptions a reasonably acceptable form of natural English.

Yu's experimental NLP system for the blocks world accepts descriptive statements, rules, questions, and commands; in addition it accepts special consistency rules to maintain database integrity. His grammar translates these four types of English statements into procedural logic and runs the resulting programs, providing the user with an English-subset capability for defining, questioning, and commanding a simulated robot in the blocks world.

The grammar for translating these statements into programs is domain independent in the sense that rules concerned with "if-then","unless", "delete", "assert", etc. are equally applicable to describing programs in any domain, not just the blocks world. Similarly, rules for transforming descriptive sentences apply regardless of their content. In contrast, the content of the English descriptions is domain dependent; in describing a tree search algorithm, for example, we might state,

A tree (X.Y) is a root X and branches Y.
A branch (U.V) is a tree U and branches V.
NIL is a branch.
A goal X is solved if X is the root of a tree.
A goal X is solved if X is the root of a branch
of a tree.

The same grammar supported by additional lexical entries and rules can be used to describe a different programming domain.

As a hacker of English grammars, accustomed to the precision required by programming languages, I can feel considerable freedom in my ability to communicate my intentions by writing lexical entries and grammars that in turn recognize some of the intentions a programmer may want to communicate using the English subset. On the other hand it is most doubtful that the person programming in the English subset will feel that there is sufficient freedom for expressing many of his/her intentions. And in the NL programming language, little attention has been paid to informing the user why a particular statement or command doesn't work; not even basic error messages are so far included. (Nor are there documentation or Help systems to guide the user.) Since NLP has received little attention so far, these lacks are hardly surprising, but if NLP is someday to become a viable programming option, user intentions must be anticipated and provision must be made for informative and corrective responses to commands and queries.

NL programming differs from NL interface systems mainly in scope; NLI is concerned with limited control languages for operating systems and databases, while NLP addresses the task of translating indefinitely large subsets of natural language into full scale programming languages. We can notice that there is no natural system for limiting the English subset -- as there is in application to interfaces with databases and operating systems. There is almost equal emphasis on declarative descriptions of data structures and procedural descriptions of operations. Progress in NLP requires many experiments with the limited goals of programming in such microsystems as the blocks world, expert knowledge domains, or in defining data-types such as trees, queues, decision networks, etc. and their operators.

Two attractive applications for NLP are apparent: the first, NLP for expert knowledge systems, is partially illustrated in Teiresius by Davis [4], which allows the user to input and debug diagnostic "if-then" rules in limited English, the second is to use a grammar to map procedures into English descriptions as an aid to documentation. The use of a symmetric grammar in NLP automatically provides the capability for translating in both directions, source-to-target language and target-to-source language (See Simmons [9]). Thus if a grammar is constructed to map the assertions and rules of a logic program into an English subset, that grammar can be used to define those programs in that subset of English as well as to provide English descriptions of the programs. But the intentionality of the program may not be apparent from its English paraphrase; intentions may never have been explicitly stated, and complex inference systems may be required to complete the description.

Natural Language Dialogues, NLD

One microworld much favored at Stanford Research Institute is composed of an air compressor, an apprentice, and a robot expert that understands a subset of English concerning the system and includes models of how it is assembled. This system has been used to develop a significant English subset grammar, a notable planning system by Sacerdoti [8], and studies of the focus and changes in focus in dialogues by Grosz [5]. Goals and intentions of both the apprentice and the expert are the central driving force of experimental programs involving this assembly task. A recent study by Appelt [2] illustrates the extent to which dialogues depend on mutual understanding and how that understanding can be communicated.

Appelt's system, called KAMP, is particularly concerned with multiple agent planning and problem solving. The domain includes two agents, Rob the robot who has a complete understanding of the compressor, and John the apprentice who has limited knowledge. The situation includes also the compressor and a toolbox containing tools. Objects such as tools and compressor parts may be in plain view -- in which case both Rob and John know their perceptual attributes, or they may be obscured and only Rob may know their details. Rob has a model of what Rob knows that John knows. Since Rob is limited to input/output on a computer terminal, his only manner of changing the compressor world is to get John to do it. John knows that Rob knows all about the compressor task, and can obtain information by asking Rob. Rob also knows that John will attempt to answer questions and try to accomplish commands.

As part of a training goal Rob wants John to remove the pump from the compressor assembly. A rule such as the following approximates this goal:

```
(NOT (ATTACHED PU PL))
< (DO JOHN (REMOVE PU PL))
```

In this rule, PU stands for "pump" and PL for "platform". The goal, that the pump not be attached to the platform can be achieved if John removes the pump from the platform of the compressor.

```
(DO JOHN (REMOVE PU PL))
< (INTEND (DO JOHN (REMOVE PU PL)))
                  ...
(INTEND (DO JOHN (REMOVE PU PL)))
< (REQUEST ROB JOHN
   (DO JOHN (REMOVE PU PL)))
```

If Rob uses a speech act commanding or requesting John to accomplish the action John will try to comply, and John's intention to remove the pump is one essential condition to his doing it. (The "..." signifies that more conditions are required.) But of course John must know what a pump is, where it is located, be in the same location, know what tools to use, and finally do the job.

One beauty of KAMP lies in its ability to plan both the action and the communicative acts required to get it done by giving John the information he needs. Rob consults his knowledge of what he knows that John knows. Since the pump assembly is in full view, John knows where it is located. Rob can see in his model of John's knowledge that:

```
(KNOW-WHAT-IS JOHN PUMP)
```

```
(KNOW-WHAT-IS JOHN PLATFORM)
```

Rob now plans what is required for John to remove the pump.

```
 (DO JOHN (REMOVE PU PL))
  < (HAVE JOHN (TOOL B1))
(SAMELOC (LOC JOHN)(LOC PU))
(DO JOHN (UNFASTEN PU PL (TOOL B1)))
```

John can remove the pump if he unfastens it from the platform using tool B1 providing he has tool B1 and is at the pump. Rob discovers that John is at the pump and that both know the following rules concerning HAVE, GET, and MOVE:

```
(HAVE X Y)
< (KNOW-WHAT-IS X Y)(DO X (GET Y))
(GET X Y) < (SAMELOC X Y)
(SAMELOC X Y)
< (DO X (MOVE X (LOC X) (LOC Y)))
```

John can HAVE the tool if he knows what it is and can get it by moving to it. Rob fails to find that John knows what tool to use, or where the tool is located. Rob must inform John:

```
 (DO ROB
  (INFORM ROB JOHN (LOC WR1 TB1)))
```

The function (TOOL B1) evaluates to WR1, wrench1, and its location is TB1, the toolbox. Rob discovers that John knows about the toolbox and does not have to inform him about its location.

Rob need not state the entire plan to John, but simply request, "Remove the pump with the wrench B1 in the toolbox", giving John a request augmented with the informing acts that wrench B1 is to be used and it is located in the toolbox. KAMP accomplishes this line of planning, translating it into language as it completes the reasoning. We should notice that Rob has a model of John that includes the knowledge that John will respond appropriately to speech acts and is otherwise merely a list of what Rob believes John knows.

Let us now see how this methodology might be used to account for the travel clerk's behavior when asked, "Is there a train to Zurich?" We can assume that the clerk is able to access a Transportation Table that contains entries of the following form:

```
(TRANSPORT _VEHICLE  _FROM-CITY
   _TO-CITY _TIME _GATE)
```

with examples such as:

```
(TRANSPORT TRAIN MILAN
    ZURICH 2100 TRACK5)
(TRANSPORT BUS DAVOS
    ZURICH 1200 BUSGATE2)
```

When an inquirer asks the clerk a question about a vehicle, the clerk can assume that the inquirer doesn't know the answer, does have a need to meet the vehicle for some purpose, and therefore needs to know the time and gate. If the query is about a vehicle *from* a city, the clerk may wish to state the destination as well to allow the inquirer additional information that might be used to correct an assumption. (In fact for this overly-simple transport table, the clerk might well have the strategy of reporting all the information just to be safe.)

The question is first translated to the logical form of a transport entry:

```
(TRANSPORT TRAIN _FROM-CITY
  ZURICH _TIME _GATE).
```

Each underscore signifies an unbound variable. The clerk's problem solver has the simple task of proving this TRANSPORT hypothesis by matching it with the entries in the TRANSPORT list. The hypothesis unifies with:

```
(TRANSPORT TRAIN MILAN
  ZURICH 2100 TRACK5)
```

and under the assumptions given above the clerk generates the response, "The train to Zurich arrives at 2100 on Track 5". If several trains to Zurich existed on the schedule, the clerk might have generated, "The next train to Zurich arrives..." and waited for additional questions.

If unification of the question failed in the event there were no train to Zurich, the clerk would then relax the constraint of the vehicle, thus allowing for buses and planes.

```
(TRANSPORT _VEHICLE _FROM-CITY
  ZURICH _TIME _GATE)
      which unifies with:
(TRANSPORT BUS DAVOS
  ZURICH 1200 BUSGATE-2)
```

In this event, the clerk responds, "No, but there is a bus at 1200 leaving from bus-gate 2 in the bus station". This response adds "in the bus station" by asking a LOC relation:

```
(LOC BUSGATE2 X)
      which unifies with,
(LOC BUSGATE2 (BUS STATION))
```

assuming that the inquirer doesn't know the location of bus-gate 2.

This transport example is obviously a simple one, but it demonstrates the general applicability of Appelt's principle of integrating the planning of speech acts with solving a problem. The example only requires the speech acts,

```
(DO INQUIRER
  (ASK CLERK (...)))
(DO CLERK
  (INFORM CUSTOMER (...)))
```

but the calculus of speech acts gets more difficult in dealing with such indirections as, "Could you tell me if there is a train to Zurich" in which it is necessary to recognize that the inquirer knows that the clerk "can tell" and that the question to be answered is the second clause. Of course "Could you pass the salt" requires a gamut of problem-solving activity to conclude that the user wants the salt rather than an informative response. A most important aspect of Appelt's work is the further development of logic for reasoning about such complex intentions and speech acts.

Text Knowledge Systems, TKS

A few current attempts at automatic translation of NL books and scientific articles into text knowledge databases for general browsing, question-answering, and summarizing operations require much more understanding of human intentions than is presently known. Texts are written by authors to achieve several purposes including the obvious informative one. More subtle purposes such as advertising a particular discipline, admiring or criticizing other writers, entertaining the reader, displaying erudition, etc. are also understood (and perhaps ignored) by most readers. Literary, political, and psychoanlytic interpretrations can uncover even subtler intentions. Texts often include reported dialogues, plausible arguments, and narratives. Good writing styles are characterized by the avoidance of unneccessary redundancy, the use of many co-referring expressions, and the elliptic description of events to avoid stating what the author believes is obvious to the reader. Intentions of participants in reported dialogues are rarely reported -- the reader is expected to infer them. Reasons for and results of reported events are also left to be inferred at least as often as they are stated, and intermediate steps in an argument are omitted if the author believes they are apparent. Unfortunately for the ease of computation, most of these omissions must be restored by a discourse analysis system if the text is to be translated into a useful, connected knowledge representation.

Discourse analysis theories and computational techniques are still highly experimental and must receive considerable development before intelligent text knowledge systems (TKS), can be brought to the point of use. Such systems will include grammars for translating sentences of text into appropriate logical formalisms (e.g. semantic network relations), discourse grammars for knitting the resulting propositions together in terms of their logical coherence, paraphrase systems for representing propositional structures in something approaching canonical forms, and question-answering and summarizing logic for extracting information. A TKS must also include browsing capabilities, indexing the elements of the text so that a user may go from segment to segment in any desired fashion, using combinations of keywords to specify a context, and operators to look ahead and behind the presented text. In our own TKS work, the keyword indexing component provides an heuristic function to select portions of text that have the best likelihood for answering a question; these candidate texts are then translated to logic, as is the question, and attempts are made to prove that the candidate text implies the question.

In such work we must anticipate that the user wants to have full access to reading any portion of the text, wants to determine if a text truly answers a question, wants answers to questions, and wants to summarize segments or topics from the text. These high-level intentions of potential users guide the initial design of a TKS. When such systems come into a higher stage of development they will then require more detailed analysis of user intentions, to insure cooperative and corrective responses, and to integrate the process of composing an answer with that of answering the question.

Conclusion

Operating systems are designed to find and execute programs for a user -- to achieve rather than to recognize a goal. But as they increased in capability and complication, it became necessary to support them with various information and *Help* systems that could at least partially aid and instruct their users. Mouse and menu systems are the current farthest advance in providing systems so friendly to the user that it is merely necessary to point to a sequence of desired options. MacIntosh is the phenomenal example of a powerful information handling system with widely varied capabilities that requires no programming skills of the user. Operating systems include no explicit models of their user or user intentions, yet they are designed with the philosophy of providing the user information that will enable successful use of the system.

Natural language interface systems have had to go farther; they must not only guide the user in their application, but must also predict and supply information the user wants, even though the question does not specify the user's intention. They must on occasion correct misapprehensions of the user and provide answers to questions that were not asked.

Systems for programming in natural language are so far initial experiments concerned with technical feasibility. If feasible, they will allow the user to state his intentions in more or less natural language so defining the programs to achieve them. Of course defining a program in English still requires careful design and definition of the program, but the amount of detail that must be specified is minimized by the use of high-level target languages such as Prolog, and will be further minimized by a deeper understanding of how natural language implies rather than states its meanings.

Systems for participating with humans in natural language dialogue are most revealing of how people communicate with language, leaving many intentions implicit. Theories of speech acts and intentions come strongly into play in any attempt to account for real human dialogues and are the current focus of experimentation in every natural language dialogue system. Computational theories of discourse are still in infancy, but they must face a daunting variety of problems concerned with what is meant but not stated. Essentially, they must track the author's focus of attention from clause to clause throughout the discourse, reconstituting the outline structure which guided the author's thought. And the usual communication dilemma is apparent; the author knows what topic he has chosen, but the reader must guess from the following sentence what the topic of the preceding one was. The author is reaching into the future; the reader is reaching back to the past.

Man machine dialogue, just as person to person dialogue, depends ultimately on each participant actively interpreting what was explicitly stated to determine what was implicitly meant.

References

[1] Allen, James F., *A Plan-Based Approach to Speech Act Recognition*, Ph.D. dissertation, Dept. of Comp. Sci., Univ. of Toronto, 1979.

[2] Appelt, Douglas, Planning English Referring Expressions, SRI International, Tech. Note 312, (AI Journal, forthcoming), Menlo Park, Calif., 1983.

[3] Biermann, Alan W., and Bruce W. Ballard, Toward Natural Language Computation, *Amer. J. of Comp. Ling.*, Vol. 6, Nbr. 2, pp. 71-86, 1980.

[4] Davis, Randall, Teiresias: Applications of Meta-Level Knowledge, in Davis, Randall and Lenat, Douglas B. *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

[5] Grosz, Barbara J., Focusing and Description in Natural Language Dialogues, in Joshi, Sag, and Webber (eds.), *Elements of Discourse Understanding*, Cambridge Univ. Press, New York, 1981.

[6] Jarke, M., J.A. Turner, E.A. Stohr, Y. Vassiliou, N.H. White, and K. Michiels, A Field Evaluation of Natural Language for Data Retrieval, New York Univ., Grad. Sch. Bus. Ad., mss. New York, 1984.

[7] Kaplan, S. J., Cooperative Responses from a Portable Natural Language Database Query System, Stanford Heuristic Programming Project, Hpp-79-19, Stanford Univ., 1979.

[8] Sacerdoti, Earl, *A Structure for Plans and Behavior*, Elsevier North-Holland Inc., Amsterdam, 1977.

[9] Simmons, Robert F., *Computations from the English*, Prentice-Hall, New York, 1984.

[10] Tennant, Harry R., *Evaluation of Natural Language Processors*, Ph.D. dissertation, Dept. Comp. Sci., Univ. of Ill., Urbana, 1980.

[11] Thompson, Craig W., *Using Menu-Based Natural Language Understanding to Avoid Problems Associated with Traditional Natural Language Interfaces to Databases*, Ph.D. dissertation, Dept. Comp. Sci., Univ. of Texas, Austin, 1984.

[12] Van Baalen, Jeffrey, *Using Natural Language to Solve Recursive Programming Problems*, M.S. thesis, Dept. Comp. Sci., Univ. of Wyoming, Laramie,

1980.

[13] Yu, Yeongh Ho, *Translating Horn Clauses from English*, M.A. thesis, Dept. of Comp. Sci., Univ. of Texas, Austin, 1984.