G. E. Heidorn

Automatic Programming Through Natural Language Dialogue: A Survey

Abstract: This paper describes and compares four research projects whose goal is to develop an automatic programming system that can carry on a natural language dialogue with a user about his requirements and then produce an appropriate program. It also discusses some of the important issues in this research area.

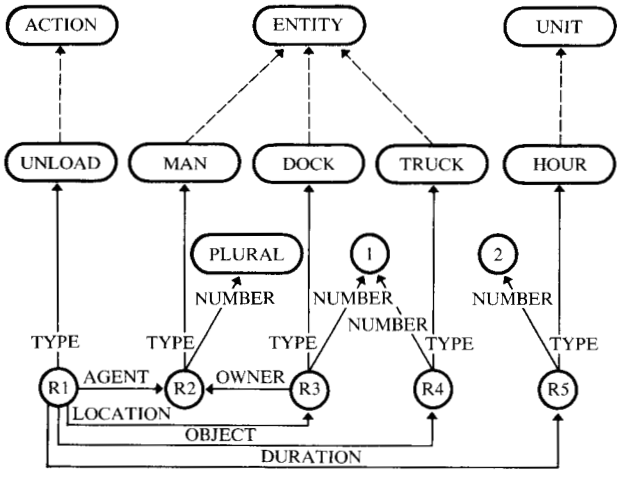
Introduction

Since the early days of computing, effort has been put into automating more and more of the programming process. (Reference [1] describes some of the most recent work.) The ultimate objective in automatic programming is a system that can carry on a natural language dialogue with a user (especially a nonprogrammer) about his requirements and then produce an appropriate program for him. Although the basic idea of "programming in English" has often been expressed in the literature [2–4], only in recent years have any serious attempts been made toward producing such a system.

Three major research efforts of this sort are currently in progress. One is at the Information Sciences Institute (ISI) of the University of Southern California; another is at Project MAC at the Massachusetts Institute of Technology (MIT), and the third is at IBM's Thomas J. Watson Research Center. A fourth effort of interest, although it has been discontinued, was at the Naval Postgraduate School (NPGS) in Monterey, California. Whereas the broadly stated objectives of these projects are the same and their techniques are similar, they do differ markedly in the details.

This paper describes and compares these four projects. The NPGS work is presented first and in the greatest detail because to date it is the only one for which there is a complete running system. Then the ISI and MIT projects are discussed, followed by a description of the work being done at IBM. After a brief comparison of the four projects, some of the important research issues are considered. (See note [5] and references [6–8].)

Figure 1 Portion of a semantic network (NPGS).



NPGS

The NPGS work was actually begun at Yale University in 1967 as a doctoral dissertation and then was completed at NPGS during the years 1968-1972 [9–11]. The goal of this project was to develop a system that would generate a GPSS simulation program after carrying on an English dialogue with a user about a simple queuing problem. A general purpose natural language processing system called NLP was developed and was then used to develop the automatic programming system for queuing problems, called NLPQ, by furnishing it with an appropriate grammar and information about queuing.

A sample problem presented to NLPQ and taken from [10] is shown in Tables 1-3. The dialogue has been divided into three parts to illustrate the main steps required to produce a program. All lower case typing was

done by the user and all upper case typing by the computer. Table 1 shows the dialogue through which the system acquired a description of the problem. It can be seen that the user can make statements, give commands, answer questions, and ask questions, and that the system can ask and answer questions and respond to commands. Table 2 shows an English description of the complete problem "in the computer's own words," which can be helpful to the user for checking the computer's "understanding." Table 3 shows the GPSS program produced, complete with English comments and meaningful symbolic names. This sample problem used about $3\frac{1}{2}$ minutes of virtual CPU time on an IBM System 360/model 67 and about 350 K bytes of virtual storage. (On an IBM 370/168, it uses only about 33 seconds of virtual CPU time.)

The data structure used by NLP is a form of semantic network. It consists of a collection of objects called records, each of which is just a list of attribute-value pairs. These records represent such things as concepts, words, physical entities, and probability distributions. The attributes of a particular record depend upon what it represents. The value of an attribute is usually a pointer to another record, but it may be simply a number or a name.

The processing to be done by NLP is specified by rules written in a rule language designed especially for this system. "Decoding" rules specify how strings of text are to be converted into records, and "encoding" rules specify how records are to be converted into text. These rules are basically phrase structure grammar rules, but they are augmented with arbitrary conditions and structure building actions.

Prior to a dialogue, NLP is given a set of about 300 "named" records containing information about some words and concepts relevant to simple queuing problems. Also, it is furnished with a set of about 300 English decoding rules and 500 English and GPSS encoding rules. (NLP augmented with these records and rules is in fact NLPQ.) As the dialogue progresses, the system uses the information it obtains from the user to build a subnetwork called the Internal Problem Description (IPD).

Basically an IPD describes the flow of mobile entities, such as vehicles, through a system consisting of stationary entities, such as pumps, by specifying the actions that take place there and their interrelationships. Each action is represented by a record whose attributes furnish such information as the type of action, the entity doing the action (i.e., the agent), the entity that is the object of the action, the location where it happens, its duration, its frequency of occurrence, and what happens next. For example, the action "The men unload the truck at a dock for two hours" could be represented (approximately) by the record:

Table 1 Problem acquisition (NPGS).

1. when a vehicle arrives at a station, it leaves there immediately if the length of the line at a pump in the station is not less than 2.
2. 75 percent of the vehicles are cars and a fourth are trucks.
3. there is just 1 pump.
4. a simulation run time of 8 hours is desired.
5. ask questions for further info.
6. HOW OFTEN DO THE VEHICLES ARRIVE AT THE STATION?
7. the arrivals of vehicles are normally distributed with a mean of 8 minutes.
8. WHAT IS THE STANDARD DEVIATION OF THE NORMAL DISTRIBUTION?
9. 1 minute.
10. OTHERWISE, WHAT DO THE VEHICLES DO?
11. what?
12. AFTER ARRIVING AT THE STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS NOT LESS THAN 2, THE VEHICLE WILL LEAVE THE STATION. OTHERWISE, WHAT DO THE VEHICLES DO?
13. they are serviced.
14. WHERE ARE THE VEHICLES SERVICED?
15. at the pump.
16. HOW LONG ARE THE VEHICLES SERVICED AT THE PUMP IN THE STATION?
17. vehicle service times are exponential.
18. THE TIME FOR THE VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY DISTRIBUTED. WHAT IS THE MEAN OF THE EXPONENTIAL DISTRIBUTION?
19. 5 minutes for cars and 9 minutes for trucks.
20. AFTER BEING SERVICED AT THE PUMP IN THE STATION, WHAT DO THE VEHICLES DO?
21. leave the station.
22. WHAT IS THE BASIC TIME UNIT TO BE USED IN THE MODEL?
23. 30 seconds.
24. THE PROBLEM STATEMENT IS COMPLETE.
25. how often does a vehicle arrive at the station?
26. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 8 MINUTES AND A STANDARD DEVIATION OF 1 MINUTE.
27. the standard deviation of the time between arrivals should be 2 minutes.
28. are vehicles serviced at the pump?
29. YES, THE VEHICLES ARE SERVICED AT THE PUMP IN THE STATION.

R1:	Type	unload
	Agent	men
	Object	truck
	Location	dock
	Duration	2 hours

In the usual graphical notation for semantic networks, a record is depicted as a node with a set of arcs emanating from it. The attribute names appear as labels on the arcs, and the attribute values appear at the arc termini. A portion of such a semantic network for the action just stated is shown in Fig. 1. The nodes in the top two rows

Table 2 Model verification (NPGS).

30. state the complete problem in english.
31. THE VEHICLES ARRIVE AT THE STATION. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 8 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES. 75 PERCENT OF THE VEHICLES ARE CARS, AND THE REST ARE TRUCKS. AFTER ARRIVING AT THE STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS LESS THAN 2, THE VEHICLE WILL BE SERVICED AT THE PUMP IN THE STATION. OTHERWISE, THE VEHICLE WILL LEAVE THE STATION. THE TIME FOR THE VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 5 MINUTES FOR THE CARS, AND 9 MINUTES FOR THE TRUCKS. AFTER BEING SERVICED AT THE PUMP IN THE STATION, THE VEHICLES LEAVE THE STATION.

    THE SIMULATION IS TO BE RUN FOR 8 HOURS, USING A BASIC TIME UNIT OF 30 SECONDS.

of the figure show some of the named records that must be available prior to a dialogue. The dotted arcs imply that one or more nodes representing concepts intermediate to those shown have been left out (e.g., mobile entity between truck and entity). The nodes in the lower part of the figure (i.e., R1, R2, etc.) are part of the IPD built during a dialogue. The owner arc between nodes R3 and R2 indicates that: "The men own the dock."

The English dialogue for specifying the problem must furnish all of the information needed by the system to enable it to build a complete IPD. The flow of mobile entities through the queuing system must be described by making statements about the actions that take place in the system and how these actions are related to one another. Each mobile entity must "arrive" at or "enter" the system. Then it may go through one or more other actions, such as "service," "load," "unload," and "wait." Then, typically, it "leaves" the system. The order in which these actions take place must be made explicit by the use of subordinate clauses beginning with such conjunctions as "after," "when," and "before," or by using the adverb "then." If the order of the actions depends on the state of the system being simulated, an "if" clause may be used to specify the condition for performing an action. Then a sentence with an "otherwise" in it is used to give an alternative action to be performed when the condition is not met.

The English dialogue must also furnish other information needed to simulate the system, such as the various times involved. It is necessary to specify the time between arrivals, the time required to perform each activity, the length of the simulation run, and the basic time unit to be used in the GPSS program. Inter-event and activity times may be given as constants or as probability distributions, such as uniform, exponential, normal, or empirical. The quantity of each stationary entity should also be specified, unless 1 is to be assumed.

The user may either state the complete problem immediately, or he may state just some part of it and then let the system ask questions to obtain the rest of the information, as was done in Table 1. The latter method results in a scanning of the partially built IPD for missing or erroneous information and the generation of appropriate questions. Each time the system asks a question, it is trying to obtain the value of some specific attribute that will be needed to generate a GPSS program. A question may be answered by a complete sentence or simply by a phrase to furnish a value for the attribute. The user may ask the system specific questions also to check on specific pieces of information in the IPD. Answers are generated from this information. In order to check the entire IPD as it exists at any time the user may request that an English problem description be produced, as was done in Table 2.

The user of NLPQ is constrained to using words and grammatical constructions known to the system. Part of the vocabulary has words for about 25 actions and entities. In addition to grammatical information about each word, such as its part of speech and how the plural or past participle is formed, semantic information is furnished. This primarily specifies whether an entity is mobile or stationary and whether an action is an event or an activity. The vocabulary also includes about 200 other words, such as attribute names, time units, certain prepositions, pronouns, conjunctions, and forms of to be. This information is entered in the form of named records.

The grammar for the system, embodied in the decoding and encoding rules, has both syntactic and semantic aspects, with the syntactic reflecting general English usage and the semantic being more narrowly oriented toward queuing problem jargon. For instance, verb phrase syntax has been treated fairly thoroughly, including various tenses, passives, negatives, and interrogatives. Most reasonable orderings of phrases in clauses and clauses in sentences are accommodated.

It is important to realize, however, that even though NLPQ can handle a fairly wide range of inputs, there are many more that it does not handle. As a specific example, the following are *some* of the ways in which statement 17 in Table 1 *could* have been made to NLPQ for that problem:

Vehicle service times are exponential.
Service times are exponentially distributed.
The time to service vehicles is exponential.

The time for vehicles to be serviced is exponential.
The time to service vehicles at the pump is exponential.

In the above, "exponential" and "exponentially distributed" are interchangeable. The following are *some* of the ways in which statement 17 *could not* have been made to NLPQ:

Vehicle service times have an exponential distribution.
Service times are distributed exponentially.
Service times may be described by an exponential distribution.
Service times are drawn from an exponential distribution.
The service times of vehicles are exponential.

To extend the system to handle these forms would require furnishing it with additional English decoding rules. Whereas syntactic extensions of this sort would be fairly straightforward, arbitrary semantic extensions generally would be more difficult.

An English problem description such as in Table 2 is essentially a "dump" of the information in the IPD put out by the English encoding rules. The overall flow of the process is for the system to make one or more statements about each action. For an action with no predecessor, a new paragraph is begun with a simple action sentence containing information about the type of action, its agent and/or object, and its location. This may be followed by a statement about the inter-event time or duration. Then, a statement beginning with "after" is produced from the successor attribute. A conditional successor may result in two sentences, with the first one having an if clause in it and the second one beginning with otherwise. After all of the actions have been described, a separate one-sentence paragraph is produced with the values of the run time and the basic time unit.

The GPSS program shown in Table 3 was produced from the IPD using the GPSS encoding rules. Definitions are put into the beginning of the GPSS program for the stationary entities, mobile entities, and distributions. Then for each action a comment consisting of a simple English action sentence is produced, followed by the GPSS statements appropriate to this action. For example, an "arrive" usually produces a GENERATE and an ASSIGN, a "leave" produces a TABULATE and a TERMINATE, and most activities produce a sequence like QUEUE, SEIZE, DEPART, ADVANCE, and RELEASE. These are usually followed by some sort of TRANSFER, depending upon the type of value the action's successor attribute has. Finally a standard "timing loop" is produced to govern the run length of the simulation.

Although this project was "completed," a system ready for production use was not developed. The NLPQ prototype has been demonstrated several times on a variety of problems, but usually with the author as the user. Although the capabilities of the system implemented are limited, the research did establish an overall framework for such a system, and useful techniques were developed. Enough details were worked out to enable the system to perform in an interesting manner, as evidenced by the sample problem in Tables 1-3.

This project was about a five man-year effort and was partially supported by the Information Systems Program of the Office of Naval Research. The primary documentation is a 376-page technical report [9], but introductory papers are available also, e.g., [10, 11].

Table 3 Automatic coding (NPGS).

```
32.  produce the gpss program.
33.          SIMULATE
             RMULT      277, 423, 715, 121, 655, 531, 999, 813
     STAT1   EQU        1, F, Q
     PUMP2   EQU        2, F, Q
     CAR2    EQU        2, T
     2       TABLE      M1, 1, 1, 2
     TRUC3   EQU        3, T
     3       TABLE      M1, 1, 1, 2
     1       FUNCTION   RN1, C24
    0.0, 0.0/0.100, 0.104/0.200, 0.222/0.300, 0.355/
    0.400, 0.509/0.500, 0.690/0.600, 0.915/0.700, 1.200/
    0.750, 1.390/0.800, 1.600/0.840, 1.830/0.880, 2.120/
    0.900, 2.300/0.920, 2.520/0.940, 2.810/0.950, 2.990/
    0.960, 3.200/0.970, 3.500/0.980, 3.900/0.990, 4.600/
    0.995, 5.300/0.998, 6.200/0.999, 7.000/1.000, 8.000/
     2       FUNCTION   RN2, C29
    0.0, -3.000/0.012, -2.250/0.027, -1.930/0.043, -1.720/
    0.062, -1.540/0.084, -1.380/0.104, -1.260/0.131, -1.120/
    0.159, -1.000/0.187, -0.890/0.230, -0.740/0.267, -0.620/
    0.334, -0.430/0.432, -0.170/0.500, 0.0/0.568, 0.170/
    0.666, 0.430/0.732, 0.620/0.770, 0.740/0.813, 0.890/
    0.841, 1.000/0.869, 1.120/0.896, 1.260/0.916, 1.380/
    0.938, 1.540/0.957, 1.720/0.973, 1.930/0.988, 2.250/
    1.000, 3.000/
     3       FUNCTION   RN3, D2
    0.750, CAR2/1.000, TRUC3/
     4       FUNCTION   P1, D2
    CAR2, 10/TRUC3, 18/
     1       FVARIABLE  16 + 4*FN2
     *
     *       THE VEHICLES ARRIVE AT THE STATION.
             GENERATE   V1
             ASSIGN     1, FN3
             TEST L     Q$PUMP2, 2, ACT2
             TRANSFER   ,ACT3
     *
     *       THE VEHICLES LEAVE THE STATION.
     ACT2    TABULATE   P1
             TERMINATE
     *
     *       THE VEHICLES ARE SERVICED AT THE PUMP.
     ACT3    QUEUE      PUMP2
             SEIZE      PUMP2
             DEPART     PUMP2
             ADVANCE    FN4, FN1
             RELEASE    PUMP2
             TRANSFER   ,ACT2
     *
     *       TIMING LOOP
             GENERATE   960
             TERMINATE  1
             START      1
             END
```

ISI

The ISI work began in 1972 with a large report [12] describing the form that an automatic programming system could take. Such a system would have four phases: problem acquisition, process transformation, model verification, and automatic coding. The first phase would consist of a natural language dialogue in problem domain terms. In the second phase the system would manipulate the information obtained during the first phase to transform it into a high level process for solving the problem. The third phase would be used to verify that this process was the one desired and that it was adequate for the problem solution. Finally, the fourth phase would optimize the process and produce the actual code to solve the problem. (The titles on Tables 1-3 were chosen to show how NLPQ fits within this framework.)

By early 1974 a prototype implementation of such a system was underway [13]. A key feature of this work is its emphasis on "domain-independence." This means that prior to the dialogue the system has *not* been primed with information about a specific problem area (e.g., queuing simulation or accounts receivable) but must obtain all of this information. The dialogue consists of the user initially stating his problem, from which the system constructs a "loose model." Then the system, through a process called "model completion," attempts to transform this loose model into an operational, interpretable form called the "precise model." The model completion process usually requires further dialogue with the user.

In this system knowledge is represented as stored tuples, which may be considered to be a form of semantic network. The processing is specified in AP/1 (an extension of the list processing language LISP) developed specifically for this project [14]. The language AP/1 supports associative relational data bases, strongly typed variables, compound pattern matches, and failure control.

In late 1974 this group decided to limit their implementation efforts to a very specific task domain, i.e., military message distribution [15], and one year later succeeded in generating their first program [16]. The example that their system handled is shown in Table 4. The program generated consists of about 6 pages of AP/1 code and took about one hour of CPU time on a Digital Equipment Corp. PDP/10 to produce.

So far this group has been concentrating their efforts on the processing required to convert an imprecise functional description of a task into a precise program rather than on the initial acquisition of the task description in natural language. Consequently, at this time, each input sentence must be *manually* translated into a parenthesized format that segments each clause and noun phrase. Table 5 shows this input form for the example in Table 4. Workers on this project intend to eventually replace the use of this form with an "off-the-shelf" natural language interface.

The processing that this system does is driven by trying to produce a viable program. First the system extracts intra-sentence information about the domain and the actions that occur there; it then builds a semantic network to represent this information. Next it does inter-statement processing to organize the actions into an appropriate control structure. This whole process requires 1) the filling in of omitted details and 2) the recognition of what is being referred to by the various phrases and clauses in the problem description. To do this the processor makes heavy use of both static and dynamic program well-formedness criteria.

Although the ISI group has been concentrating on the particular task domain of military message distribution, they are still concerned with domain independence and have made a strong effort to keep information about the domain separate from the more general information. By mid-1976 they hope to have done examples in several different domains to test their techniques. They are presently not concerned with generating optimized programs.

This project is sponsored by ARPA. The group at ISI currently consists of three people, although it has had as many as six. The references already cited give a reasonably good idea of what this group is trying to do and how they are going about doing it. Reference [17] provides an especially good, concise progress report.

MIT

In 1972 work was begun at MIT's Project MAC toward the goal of a natural language automatic programming system for business applications. In the first progress report [18], an overview of Protosystem I, a partially implemented system, was given. The user's interaction with this system begins with a questionnaire, but one that allows constructive responses rather than just multiple-choice answers. The user's particular application is constrained to being an instantiation of a general model of a business procedure, such as billing, constructed in a relational modeling language called MAPL. After acquiring the user's description of his application, the system guides him in the construction of an appropriate block diagram. He is then allowed to explore the resulting procedure through simulation. Finally, the block diagram is translated into an optimized PL/1 program.

MAPL was intended to be a language in which relational models of the world could be built and was designed especially for this system. This form of knowledge representation is basically a semantic network also. A routine for translating natural language text into a MAPL expression was also designed. It uses an aug-

Table 4 Message distribution example (ISI).

MESSAGES RECEIVED FROM THE AUTODIN-ASC ARE PROCESSED FOR AUTOMATIC DISTRIBUTION ASSIGNMENT.

THE MESSAGE IS DISTRIBUTED TO EACH ASSIGNED OFFICE.

THE NUMBER OF COPIES OF A MESSAGE DISTRIBUTED TO AN OFFICE IS A FUNCTION OF WHETHER THE OFFICE IS ASSIGNED FOR ACTION OR INFORMATION.

THE RULES FOR EDITING MESSAGES ARE (1) REPLACE ALL LINE FEEDS WITH SPACES (2) SAVE ONLY ALPHANUMERIC CHARACTERS AND SPACES AND THEN (3) ELIMINATE ALL REDUNDANT SPACES.

IT IS NECESSARY TO EDIT THE TEXT PORTION OF THE MESSAGE.

THE MESSAGE IS THEN SEARCHED FOR ALL KEYS.

WHEN A KEY IS LOCATED IN A MESSAGE, PERFORM THE ACTION ASSOCIATED WITH THAT TYPE OF KEY.

THE ACTION FOR TYPE-0 KEYS IS: IF NO ACTION OFFICE HAS BEEN ASSIGNED TO THE MESSAGE, THE ACTION OFFICE FROM THE KEY IS ASSIGNED TO THE MESSAGE FOR ACTION. IF THERE IS ALREADY AN ACTION OFFICE FOR THE MESSAGE, THE ACTION OFFICE FROM THE KEY IS TREATED AS AN INFORMATION OFFICE. ALL INFORMATION OFFICES FROM THE KEY ARE ASSIGNED TO THE MESSAGE IF THEY HAVE NOT ALREADY BEEN ASSIGNED FOR ACTION OR INFORMATION.

THE ACTION FOR TYPE-1 KEYS IS: IF THE KEY IS THE FIRST TYPE-1 KEY FOUND IN THE MESSAGE THEN THE KEY IS USED TO DETERMINE THE ACTION OFFICE. OTHERWISE THE KEY IS USED TO DETERMINE ONLY INFORMATION OFFICES.

Table 5 Actual input for message distribution example (ISI).

```
* ((MESSAGES ((RECEIVED) FROM (THE "AUTODIN-ACS"))) (ARE PROCESSED) FOR (AUTOMATIC
DISTRIBUTION ASSIGNMENT))

* ((THE MESSAGE) (IS DISTRIBUTED) TO (EACH ((ASSIGNED)) OFFICE))

* ((THE NUMBER OF (COPIES OF (A MESSAGE) ((DISTRIBUTED) TO (AN OFFICE)))) (IS) (A FUNCTION OF (WHETHER ((THE OFFICE) (IS ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

* ((THE RULES FOR ((EDITING) (MESSAGES))) (ARE) (: ((REPLACE) (ALL LINE-FEEDS) WITH (SPACES)) ((SAVE) (ONLY (ALPHANUMERIC CHARACTERS) AND (SPACES))) ((ELIMINATE) (ALL REDUNDANT SPACES))))

* (((TO EDIT) (THE TEXT PORTION OF (THE MESSAGE))) (IS) (NECESSARY))

* (THEN (THE MESSAGE) (IS SEARCHED) FOR (ALL KEYS))

* (WHEN ((A KEY) (IS LOCATED) IN (A MESSAGE)) ((PERFORM) (THE ACTION ((ASSOCIATED) WITH (THAT TYPE OF (KEY))))))

* ((THE ACTION FOR (TYPE-0 KEYS)) (IS) (: (IF ((NO OFFICE) (HAS BEEN ASSIGNED) TO (THE MESSAGE) FOR ("ACTION")) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS ASSIGNED) TO (THE MESSAGE) FOR ("ACTION"))) (IF ((THERE IS) ALREADY (AN "ACTION" OFFICE FOR (THE MESSAGE))) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS TREATED) AS (AN "INFORMATION" OFFICE))) (((LABEL OFFS1 (ALL "INFORMATION" OFFICES FROM (THE KEY))) (ARE ASSIGNED) TO (THE MESSAGE)) IF ((REF OFFS1 THEY) (HAVE (NOT) (ALREADY) BEEN ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

* ((THE ACTION FOR (TYPE-1 KEYS)) (IS) (: (IF ((THE KEY) (IS) (THE FIRST TYPE-1 KEY ((FOUND) IN (THE MESSAGE)))) THEN ((THE KEY) (IS USED) TO ((DETERMINE) (THE "ACTION" OFFICE)))) (OTHERWISE (THE KEY) (IS USED) TO ((DETERMINE) (ONLY "INFORMATION" OFFICES)))))
```

mented transition network approach and pays special attention to verb case frames. The process of PL/1 code generation and optimization is described in this report too, using an inventory system example.

From the second year's progress report [19] it became apparent that the emphasis had shifted from trying to build a single integrated system to studying the pieces somewhat independently. Currently there are basically three prototypes being constructed. One is OWL, which is intended to be a very general system for dealing with knowledge representation and natural language processing. Another is a system for putting together packages of programs configured according to answers a user gives to a multiple choice questionnaire, for the domain of planning and scheduling in production and distribution systems. The third prototype deals with automatically optimizing data organization and accessing procedures for data processing systems operating on large keyed files [20].

A related effort, which was completed in 1974, dealt with understanding how a user might interact with a knowledge-based application system [21]. A series of experiments were performed in which a manager was given a problem to solve by obtaining information in unrestricted English from a data base about a hypothetical small company. The experimenter would intercept the questions and answer them with the aid of a more restricted system. By recording and analyzing these dialogues, much was learned about the capabilities required of an actual system of this sort.

Table 6 Hypothetical dialogue with Susie software (MIT).

HELLO, I AM SUSIE. WHAT IS YOUR NAME?

bill. i need a program written to manipulate blocks.

I CAN HELP YOU SOLVE THE PROBLEM OF WRITING A PROGRAM WHICH BUILDS A SCENE USING THE BLOCKS FROM ANOTHER SCENE.

that is what i want.

OK. I NEED A DESCRIPTION OF THE INITIAL AND FINAL SCENES.

the final scene contains 3 blocks: B1, B2, B3

OK.

what else do you want to know about the final scene?

WHAT ARE THE POSITIONS OF B1, B2, and B3 WITH RESPECT TO EACH OTHER?

B1 is on top of B2 and B3 is on top of B1.

OK.

initially, all of them are on the ground.

DO YOU MEAN IN THE INITIAL SCENE?

yes. can you write a program from that?

I HAVE ENOUGH INFORMATION TO START WRITING THE PROGRAM. I CAN'T DETERMINE ALL OF THE INFORMATION WHICH I WILL NEED UNTIL I WRITE THE PROGRAM.

ok. start writing it.

DO YOU WANT ME TO DECIDE IMPLEMENTATION ISSUES SUCH AS WHAT TO GRASP A BLOCK WITH?

yes, if i can ask you about them later.

YOU CAN. IS THERE A BLOCK ON TOP OF B2 IN THE INITIAL SCENE?

the initial scene only contains B1, B2, and B3.

I SEE. I HAVE FINISHED THE PROGRAM. DO YOU WANT ME TO DESCRIBE THE PROCEDURE?

yes

FIRST I MOVE B2 WITH MY LEFT HAND, THEN I PUT B1 ON TOP OF B2 WITH MY LEFT HAND, THEN I PUT B3 ON TOP OF B1 WITH MY LEFT HAND.

how did you decide to use your left hand to move B2?

I NEEDED TO CHOOSE A HAND WITH WHICH TO GRASP B2. BOTH OF MY HANDS ARE FREE AT THAT POINT IN THE PROBLEM. YOU TOLD ME TO DECIDE THE IMPLEMENTION ISSUES. I CHOSE THE FIRST HAND WHICH I THOUGHT OF.

OWL is considered to be central to the MIT project and has been described in considerable detail [22–24]. The OWL language, which has replaced MAPL for representing knowledge, has as its major feature that English has been used as a guide in the definition of its data structures. It is still basically a form of semantic network, however.

Table 7 Definition of PUT-ON-TOP in OWL (MIT).

```
(LEARN (DEFINE PROCEDURE (PUT-ON-TOP-OF BLOCK-1))
       (AGENT (PUT-ON-TOP-OF BLOCK-1) PERSON-1)
       (INSTRUMENT (PUT-ON-TOP-OF BLOCK-1) HAND-1)
       (PART AGENT HAND-1)
       (SPECIFIC-POSITION (PUT-ON-TOP-OF BLOCK-1)
                          (ON-TOP-OF BLOCK-2)
       (PRINCIPAL-RESULT (PUT-ON-TOP-OF BLOCK-1)
                         (POSITION OBJECT SPECIFIC-POSITION))
       (METHOD (PUT-ON-TOP-OF BLOCK-1) (FIND SPACE-1))
       (POSITION SPACE-1 SPECIFIC-POSITION)
       (BENEFICIARY SPACE-1 OBJECT)
       (THEN (FIND SPACE-1) (GRASP OBJECT))
       (THEN (GRASP OBJECT)
             (MOVE (INSTRUMENT-1 (GRASP OBJECT))))
       (DESTINATION (MOVE INSTRUMENT-1) POSITION-1)
       (RESULT (MOVE INSTRUMENT-1)
               (POSITION OBJECT SPECIFIC-POSITION)
       (THEN (MOVE INSTRUMENT-1) (LET-GO-OF OBJECT))
       (Y-COORDINATE POSITION-1
             (PLUS 2
                   (Y-COORDINATE (POSITION (OBJECT (FIND SPACE-1))))
                         (MEASURE (HEIGHT OBJECT))))
       (X-COORDINATE POSITION-1
             (X-COORDINATE (POSITION (OBJECT (FIND SPACE-1)))))))
```

Two basic structural devices are used in the OWL formalism: specialization and restriction. Specialization says essentially that one concept is a-kind-of another concept (e.g., a dog is a kind of animal). Restriction has to do with giving properties to a concept (e.g., a dog has four legs). The use of case relations, such as agent, object, location, and duration, is basic to OWL also.

Effort has been put into building an augmented transition network parser [25] for translating English sentences into OWL data structures [19, 22]. For debugging purposes, this group is attempting to write a program in OWL capable of carrying on the dialogue shown in Table 6. The OWL language also provides for the specification of procedures in such a manner that they can be executed for their effect or merely inspected for their information. Table 7 shows an OWL procedure relevant to the dialogue of Table 6.

This project also is sponsored by ARPA and currently involves 12 faculty members and students. In addition to the cited references, there are a number of internal memos and student papers describing various aspects of the work.

IBM

The work in this area of automatic programming at IBM took on project status in 1974, although much of the groundwork was laid prior to that [26]. The long range goal of the Computer Assisted Application Definition Group is to develop a system that will permit users to create business application programs by holding an informal, interactive dialogue with the computer. Currently under development is a more modest system that will help a user to customize a set of highly parameterized application programs for business accounting by means of a natural language dialogue.

An example of the sort of dialogue that this system is expected to support is shown in Table 8. It can be seen that this dialogue has similar characteristics to the ones

shown in Tables 1 and 6, namely that both the user and the computer make statements and ask and answer questions. There is also some verification included. It is intended that the user also be able to request a simulated execution of his application program to explore its behavior under various conditions.

The data structure used by this system is a form of semantic network too [27]. This network is considered to have basically three parts, called the program model, the application model, and the linguistic model. The program model furnishes an abstract description of the parameterized programs available to be customized, including information about the various options, using concepts from the Business Definition System, BDS [28, 29]. The application model provides information about concepts relevant to business, such as that invoices and statements are kinds of documents that normally are sent to customers. The linguistic model provides information about the words of the English language and how they are used. The interrelationship of these models is important. Each object in the program model is linked to its "related application object," which serves as a sort of conceptual explanation for it, and each object in the application model (and possibly some in the program model) is linked to its "related linguistic object," which is used for communicating in natural language.

A simplified example of a very small portion of such a semantic network is shown in Fig. 2. Each node represents an object and has a name in it for reference purposes. The relationships among objects are shown as directed arcs, sometimes labeled. All of the arcs are actually two-way links, but the label names shown are meaningful only in the direction shown.

The unlabeled arcs in the figure represent the "instance" relation, which serves the same purpose as the type relation in Fig. 1, although it is drawn in the opposite direction. For example, F1 is an instance of Field (i.e., the type of F1 is Field), and Field is an instance of POB (Program OBject). Because of the transitivity of the instance relation, every object in the program model is ultimately an instance of POB. Similarly, every object in the application model is ultimately an instance of AOB (Application OBject), and every object in the language model is ultimately an instance of LOB (Language OBject). In the full network there tends to be many more levels between the most general objects at the top and the most specific objects at the bottom than appears in this small example.

The portion of program model shown in the figure may be interpreted as saying that some field F1 is derived by performing an operation OP3 using fields F2 and F3 as operands. By following the RAO (Related Application Object) arcs it can be seen that in terms of application concepts this calculation is EXT-PR =

Table 8 Hypothetical customizing dialogue (IBM).

IN ORDER FOR ME TO CREATE A BILLING PROGRAM FOR YOU I WILL NEED TO GET CERTAIN INFORMATION FROM YOU ABOUT WHAT YOU WANT. FEEL FREE TO ASK QUESTIONS AS WE GO ALONG.

WOULD YOU LIKE YOUR SYSTEM TO BE TRANSACTION ORIENTED OR BATCH ORIENTED?

what is a transaction oriented system?

IN A TRANSACTION ORIENTED SYSTEM AN INVOICE IS PRODUCED IMMEDIATELY WHEN AN ORDER ARRIVES.

i want invoices to be produced just once a day at 5 p.m.

DO YOU WANT A SEPARATE INVOICE FOR EACH ORDER OR DO YOU WANT THE ORDERS GROUPED BY CUSTOMER?

separate invoices.

ARE THE CUSTOMER'S NAME AND ADDRESS FURNISHED ON EACH ORDER OR MUST THEY BE LOOKED UP IN THE CUSTOMER MASTER FILE?

looked up; is it possible to specify a different address on an order?

NO, NOT IF THEY ARE TO BE LOOKED UP IN THE CUSTOMER MASTER FILE.

how would i handle an order from a new customer?

BY "NEW CUSTOMER" DO YOU MEAN ONE WHO IS NOT IN THE MASTER FILE?

yes.

YOU WOULD HAVE TO MAKE AN ENTRY FOR HIM IN THE CUSTOMER MASTER FILE BEFORE THE ORDER IS PROCESSED.

okay.

DOES THE SAME TAX RATE APPLY TO ALL CUSTOMERS?

yes, 5%.

TO SUMMARIZE YOUR REQUIREMENTS:
INVOICES ARE TO BE PRODUCED IN A BATCH AT 5 PM EACH DAY.
A SEPARATE INVOICE WILL BE PRODUCED FOR EACH ORDER.
THE CUSTOMER'S NAME AND ADDRESS WILL BE LOOKED UP IN THE CUSTOMER MASTER FILE.
A TAX RATE OF 5% APPLIES TO ALL CUSTOMERS.

DO YOU HAVE ANY FURTHER QUESTIONS?

what does an invoice look like?

(SYSTEM PRINTS A SAMPLE INVOICE)

how is extended price calculated?

THE EXTENDED PRICE FOR AN ITEM IS CALCULATED BY MULTIPLYING THE UNIT PRICE BY THE QUANTITY.

that's all.

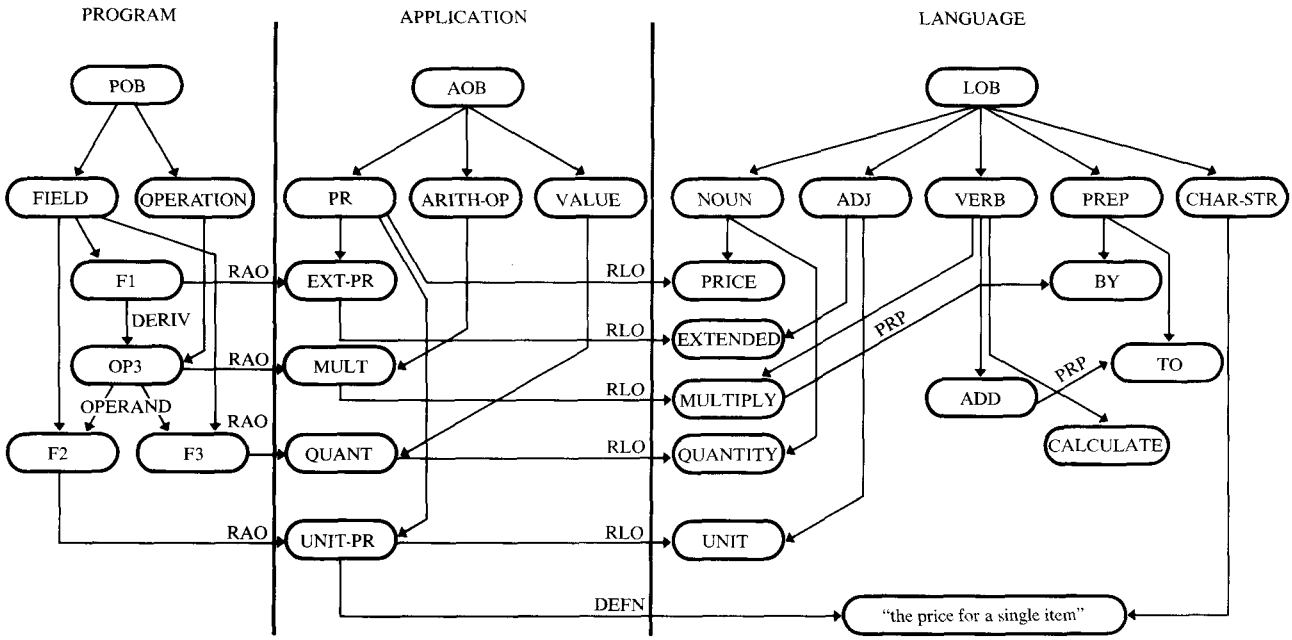


Figure 2 Portion of a semantic network (IBM).

UNIT-PR * QUANT. By making use of the relationships given in the program and application models, along with the RAO's and RLO's (Related Language Objects) given, it is possible to generate the sentence, "Extended price is calculated by multiplying unit price by quantity." The semantic network for this system is still in its early stages of development, and undoubtedly its final form will be somewhat different from that shown here.

The dialogue is driven by the options given in the program model. In its simplest form this is not much different from providing the user with a questionnaire of the sort that is central to the Application Customizer Service, ACS [30]. However, in this case the questioning is dynamic, with later questions being dependent upon information supplied by the user earlier in the conversation. Also, the user may ask questions about terms he does not understand and about the effects of making certain choices.

The natural language processing in this system is being done by an expanded version of NLP. Whereas the original version of NLP used for the queuing problem application described earlier was implemented in FORTRAN, this new version has been implemented in LISP so that the facilities of this more powerful list processing language may be taken advantage of. In order to support the large amount of network manipulation required by the system described here, a companion special purpose language, called THINKER, which has some of the same features as ISI's AP/1, has been implemented in LISP also [27]. The business application programs are written in the BDS language.

To observe the kinds of questions a user of this system might ask, a series of actual dialogues with a manually simulated system have been recorded and analyzed [31, 32]. This manual system is also providing a framework for building the actual system. As appropriate techniques are developed, parts of the system are automated, with the eventual goal being to completely eliminate the need for manual intervention.

This project is funded internally by IBM and currently has six people on it. In addition to the references already cited, an overview is also available [33].

Comparison

It should be apparent by now that none of these groups is trying to develop what might be called "an English-like programming language." (After all, that is what some people would say COBOL is.) Rather, what they are trying to do is develop knowledge-based systems that can "understand" a user's statement of a problem or a procedure in his own terms and convert it into a computer program. As stated by Balzer [13], "the main distinction between conventional and automatic programming is the latter's use of a semantic model of a domain to structure the dialogue between the system and the user, to understand the user's responses, and to translate the user's responses into actions."

A tabular summary of information about the four projects just described is presented in Table 9 for quick reference and comparison. The philosophy underlying all of these projects is that the ultimate automatic programming system is one that carries on a natural language

Table 9 Summary of the four projects.

	NPGS	*ISI*	*MIT*	*IBM*
Location	Monterey, CA.	Marinadel Rey, CA.	Cambridge, MA.	Yorktown Heights, NY.
Sponsor	ONR	ARPA	ARPA	IBM
Principal investigator	George Heidorn	Robert Balzer	William Martin	Irving Wladawsky
Time period	1968-72	1972-	1972-	1974-
People currently on project	0	3	12	6
Problem domain	queuing simulations	any	business applications	accounting applications
Task	generate progs.	generate progs.	gen. or cust. progs.	customize progs.
Data structure	semantic network	semantic network	semantic network	semantic network
Nat. lang. technique	aug. phrase struc.	none yet	aug. trans. net.	aug. phrase struc.
Computer used	360/67	PDP-10	PDP-10	370/168
Language used	FORTRAN	LISP	LISP	LISP
Language developed	NLP	AP/1	MAPL, OWL	THINKER
Target language	GPSS	AP/1	PL/1	BDL
Current status	completed prototype	completed prototype for message distribution	implementing three prototypes	implementing prototype
Relevant references	9-11	12-17	18-25	26-33

dialogue with a user about his requirements and then produces an appropriate program for him. They also share the philosophy that the way to bring this about is by trying to build extendable prototype systems that will support this processing for at least a limited class of applications.

Except for the NPGS project, detailed documentation about the techniques being used is lacking. Also, that which does exist becomes out-of-date pretty quickly because this work is constantly breaking new ground. Consequently, it would be difficult and probably not very useful to compare these projects in great detail at this time. However, some general comments can be made.

Of the three current projects, two (IBM and MIT) are concerned with business applications, and the other (ISI) is striving to be "domain independent." (However, the ISI group has been concentrating on a military message distribution application.) The ISI system is intended to *generate* programs "from scratch," whereas the IBM system is intended initially just to *customize* parameterized programs. The MIT system is intended to do both.

All three of the current projects are using LISP as their basic implementation language, but each is also developing a higher level language embedded in LISP to make the processing easier to specify. All are using basically some form of semantic network representation for their knowledge base and some form of procedural specification for their natural language processing. Because each of these groups is in its early stages of prototype implementation, nothing can be said yet about their relative performance.

Research issues

As implementation progresses these researchers are faced with many problems. For instance, saying that a semantic network representation is being used is a rather general statement. What is the specific form that is best for the system? What are the specific concepts that must be represented? How are these concepts related to each other? How many might there be? Will this form of representation support the large number of inductive and deductive inferences likely to be required? Some interesting work on a formalism for semantic networks (or "conceptual graphs") that eventually may be useful in projects of this sort is described by Sowa [34]. An excellent discussion of many of the issues involved with the use of semantic networks appears in [35].

The natural language processing to be done by these systems requires techniques that are more advanced than those currently available, and it is likely that the only way these techniques are going to be developed is by work on such systems. A communication view of language must be taken, rather than being concerned with parsing and interpreting sentences in isolation as is done in most query systems [e.g., 36, 37]. In an automatic programming system the user and the computer engage in a dialogue with the mutual goal of finding a match between the user's requirements and what the system can provide. They must enter into the conversation with a certain amount of knowledge in common, and then each must help the other to know more. In any conversation the actual words and sentences uttered provide only a very small part of new information. Primarily they serve as keys that enable the listener to open up new paths through information he already has.

The important point is that a dialogue takes place for a purpose. This sets the overall context. As the dialogue progresses, many sub-purposes are established and satisfied, each setting its own local context. (Actually, this may be multi-layered.) By knowing these purposes at

each point in time, a listener is able to set up expectations that help him to understand what is being said. Making this notion of context operational would seem to be crucial for supporting a truly natural language dialogue in an automatic programming system. In our work at IBM we feel that this will be possible because we are dealing with a very restricted domain of discourse and a system with a very specific purpose. Also, because the dialogue is driven by options in the program model, we have a good basis for establishing local contexts throughout the dialogue. A technique for dealing with context in task-oriented dialogues is described in [38].

In addition to devising an operational notion of context, another problem to be faced in the natural language processing part of such a system has to do with giving the user freedom in the way he expresses himself, which is especially important when the need for training him is to be minimized. Ultimately this means that the system should be able to process completely, both syntactically and semantically, every user utterance (i.e., it would "understand" anything that was said). Clearly this would be extremely difficult, if not impossible, to bring about. What is needed is a technique for partially processing utterances, to make it possible for the system to get something out of an utterance when it is not able to do a complete syntactic and semantic analysis. To do this, it would have to be able to recognize some words and phrases and make assumptions about the unrecognized portions. The idea would be for the system to get enough information out of such an utterance to be able to respond in a manner that the user would feel is reasonable. In many cases this response could take the form of one or more fairly specific clarifying questions [39].

Debugging can be a major difficulty with any program generator. How could we ever be sure that all of the huge number of programs that might be produced would be correct? The usual approach is to test each piece independently and then put them together "very carefully," performing some tests on their interaction. When the input is in natural language, there is an additional level of difficulty. How can we be sure that the computer interprets correctly each of the essentially infinite number of different statements that the user might make? It would seem essential to provide facilities to enable a user to readily check for himself the programs produced but without requiring him to have an intimate knowledge of the programming language. The computer-produced English verification and the comments and symbolic names in the GPSS program from NLPQ are considered to be initial steps in this direction, but additional facilities, such as the computer-assisted running of test cases, will have to be provided.

This debugging issue is not of such great concern in our initial work at IBM because of the fact that the system under development will customize programs rather than generate them. This makes it possible to completely check out the various programs that can result beforehand. Also, as Mikelsons has pointed out [27], in this system the burden of matching a procedure to a task is being placed more on the user than on the machine, and he is better suited for doing it. The planned capability of the system to do simulated execution of an application program should be helpful to the user in this regard, too.

There will probably always be the danger that a computer conversing in English may give the appearance of being more knowledgeable than it actually is, thus instilling false confidence on the user's part. It might be able to discuss an application beautifully, but produce an erroneous program that would be accepted simply because "it came from the computer." Higher level considerations such as this will have to be dealt with in addition to the more technical issues discussed above before natural language automatic programming can become a practical reality.

Acknowledgments

This paper benefitted from discussions with R. Balzer, A. Brown, C. Green, A. Malhotra, W. Martin, M. Mikelsons, J. Sammet, P. Sheridan, N. Sondheimer, and I. Wladawsky.

References and notes

1. A. W. Biermann, "Approaches to Automatic Programming," *Advances in Computers* **15**, Academic Press, New York, 1975.
2. J. E. Sammet, "The Use of English as a Programming Language," *Commun. ACM* **9**, 228 (March 1966).
3. M. Halpern, "Foundations of the Case for Natural-Language Programming," *AFIPS Conf. Proc., Fall Jt. Comput. Conf.*, Spartan Books, Washington, D.C., 1966, p. 639.
4. J. E. Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.
5. There is another project at the Artificial Intelligence Laboratory of Stanford University that should be mentioned here, too. This group is currently building a system for automatically generating concept formation programs through natural language dialogue. Although this work is very much in line with the topic of this survey, it was begun only recently and has not yet been described in any published documents. References [6–8] describe earlier related work.
6. C. C. Green, et al, "Progress Report on Program-Understanding Systems," *Memo AIM-240*, Artificial Intelligence Laboratory, Stanford University, Stanford, California, August 1974.
7. C. C. Green and D. R. Barstow, "A Hypothetical Dialogue Exhibiting a Knowledge Base for a Program-Understanding System," *Memo AIM-258*, Artificial Intelligence Laboratory, Stanford University, Stanford, California, January 1975.
8. C. C. Green and D. R. Barstow, "Some Rules for the Automatic Synthesis of Programs," Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, 1975, p. 232.

9. G. E. Heidorn, "Natural Language Inputs to a Simulation Programming System," *Technical Report NPS-55HD72101A*, Naval Postgraduate School, Monterey, California, October 1972.
10. G. E. Heidorn, "English as a Very High Level Language for Simulation Programming," Proc. Symp. on Very High Level Languages, *Sigplan Notices* **9**, 91 (April 1974).
11. G. E. Heidorn, "Simulation Programming through Natural Language Dialogue," *North-Holland/Tims Studies in the Management Sciences* **1**, *Logistics*, edited by M. A. Geisler, North-Holland Publishing Co., Amsterdam, 1975, p. 71.
12. R. M. Balzer, "Automatic Programming," *Technical Report RR-73-1*, USC/Information Sciences Institute, Marina del Rey, California, September 1972.
13. R. M. Balzer, et al, "Domain-Independent Automatic Programming," *Information Processing* **74**, North-Holland Publishing Co., Amsterdam, 1974, p. 326.
14. R. M. Balzer, et al, "AP/1—A Language for Automatic Programming," *Technical Report RR-73-13*, USC/Information Sciences Institute, Marina del Rey, California, 1974.
15. R. M. Balzer, "Imprecise Program Specification," *Technical Report RR-75-36*, USC/Information Sciences Institute, Marina del Rey, California, June 1975.
16. R. M. Balzer, N. M. Goldman, and D. S. Wile, "Specification Acquisition from Experts," Presentation transparencies, USC/Information Sciences Institute, Marina del Rey, California, September 1975.
17. "Automatic Programming," *Annual Technical Report SR-75-3*, USC/Information Sciences Institute, Marina del Rey, California, June 1975, p. 24.
18. "Automatic Programming Group," *Project MAC Progress Report X*, Massachusetts Institute of Technology, Cambridge, Massachusetts, July 1973, p. 172.
19. "Automatic Programming Group," *Project MAC Progress Report XI*, Massachusetts Institute of Technology, Cambridge, Massachusetts, July 1974, p. 107.
20. G. R. Ruth, "Optimization in Protosystem I," Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, July 1975
21. A. Malhotra, "Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis," *Technical Report TR-146*, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1975.
22. W. A. Martin, "OWL, A System for Building Expert Problem Solving Systems Involving Verbal Reasoning," unpublished course 6.871 notes, Massachusetts Institute of Technology, Cambridge, Massachusetts, Spring 1974.
23. L. Hawkinson, "The Representation of Concepts in OWL," Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, 1975, p. 107.
24. W. A. Martin, "Conceptual Grammar," *Automatic Programming Group Memo 20*, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, October 1975.
25. W. A. Woods, "Transition Network Grammars for Natural Language Analysis," *Commun. ACM* **13**, 591 (October 1970).
26. M. Mikelsons, "Computer Assisted Application Definition," Unpublished memo; IBM Thomas J. Watson Research Center, Yorktown Heights, New York, August 1973.
27. M. Mikelsons, "Computer Assisted Application Definition," *Proc. Second ACM Symp. on Principles of Programming Languages*, Palo Alto, California, January 1975.
28. W. G. Howe, V. J. Kruskal, and I. Wladawsky, "A New Approach for Customizing Business Applications," *Research Report RC 5474*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, March 1975.
29. M. Hammer, W. G. Howe, V. J. Kruskal, and I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications," *Research Report RC 5583*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, August 1975.
30. *Application Customizer Service Application Description*, GH20-0628-5, IBM Data Processing Division, White Plains, New York, 1971.
31. J. C. Thomas, "A Method for Studying Natural Language Dialogues," *Research Report RC 5882*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, February 1976.
32. A. Malhotra and P. B. Sheridan, "Experimental Determination of Design Requirements for a Program Explanation System," *Research Report RC 5831*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, January 1976.
33. I. Wladawsky, "The Mentor for Business Applications (MBA): A Natural Language Automatic Programming System," Unpublished memo, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, March 1975.
34. J. F. Sowa, "Conceptual Graphs for a Data Base Interface," *IBM J. Res. Develop.* **20**, 336 (1976, this issue).
35. W. A. Woods, "What's in a Link?" *Representation and Understanding: Studies in Cognitive Science*, edited by D. G. Bobrow and A. Collins, Academic Press, New York, 1975.
36. W. J. Plath, "REQUEST: A Natural Language Question-Answering System," *IBM J. Res. Develop.* **20**, 326 (1976, this issue).
37. S. R. Petrick, "On Natural Language Based Computer Systems," *IBM J. Res. Develop.* **20**, 326 (1976, this issue).
38. B. G. Deutsch, "Establishing Context in Task-Oriented Dialogs, *Amer. Journ. Computational Linguistics*, Microfiche 35, 1975.
39. E. F. Codd, "Seven Steps to RENDEZVOUS with the Casual User," *Data Base Management*, edited by J. W. Klimbie and K. L. Koffeman, North-Holland Publishing Co., Amsterdam, 1974, p. 179.

Received November 25, 1975

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.