

Whole ALTO World Newsletter

Technology and Tools

XEROX

February 28, 1978

SPECIAL NOTE

NEW OPERATING SYSTEM - As previously announced, the timestandard implementation is being changed. This last week a new OPERATING SYSTEM and BRAVO were released. As future releases of this and other subsystems will not necessarily operate properly with the old operating system, *you should change over as soon as possible* by retrieving and executing NEWOS.cm from your local IFS or MAXC. You will need about 300 free pages on your disk. Check with your local support people for special procedures. The documentation, <AltoDocs>OS.press, has been revised.

GENERAL NOTES

WHOLE ALTO WORLD MEETING - The Whole Alto World meeting was hosted by Liz Bond of XEOS in Pasadena on February 7, 1978. Fifty-five people, representing virtually every Alto using group, attended.

The Distributed Message System(DMS), an upcoming, Alto-based replacement for the MAXC MSG system, was described by Frank Ludolph. Under DMS, messages are stored on IFS stations (or MAXC for the immediate future) only during transit. Received messages will be stored on the user's Alto disk in one or more user-designated files managed by Alto resident software. The user interface will be familiar to Alto users, consisting of several windows, menus, and Bravo style editing facilities. Although it is a research project, it is expected that DMS will be available to MAXC MSG users this summer.

Dick Sonderegger, SD Support, reports that MESA is now available through the Whole Alto World coordinator on a limited basis, by specific request, and depending on the proposed application. The language is still evolving and should not be used for long term development projects. Questions and problems should also be channeled through the coordinator's office to <SDsupport>.

Barry Smith, Sheldon Raizes, Terry Anderson, and Irv Keschner, lawyers with the Xerox Patent Department, attended the meeting to discuss the methods used to protect intellectual property, trade secrets, patents, and copyrights, as they apply to the Alto. Barry will be working with WAW in the near future to develop written material on this subject. The material will be printed in the Newsletter when it becomes available.

Terry Haney spoke on SPG's board repair activity. Boards should be sent to Terry, *along with a description of the problems* and, if from Orbit or Dover, a copy of the printer's output. Boards are logged and their repair scheduled in conjunction with SPG's other activities.

Jim Hall announced that his 1200 group is very interested in providing maintenance service for as many Altos as possible. Existing spares inventories can be turned in for credit. Contact Jim for pricing particulars.

The 7th Alto build will proceed on schedule according to Doug Stewart. This will be the last Alto build. There has been some difficulty obtaining 7000 bases for the Dover build (marketing has been quite successful in placing them recently), but it is not expected to significantly delay Dover deliveries.

Sam Losh of XEOS reports that Sequoia development is continuing. He requested that organizations interested in obtaining Sequoias contact him. If there is sufficient interest, deliveries could begin in the fall.

John Ellenby briefly described Advanced Systems Division's role in marketing test probes based on Alto technology. ASD has requested information on the Fuji Xerox mag brush developer, used on their 7200, for possible retrofit to Dover. The unit would improve solid area development. Additional information will be printed in the Newsletter as it is available.

The reasons for developing Altos as gateways were outlined by Ted Strollo. Essentially, the current Novas present maintenance problems, the small memory (32K) prevents further software development, and the Nova operating system is not as malleable as the Alto's. The number of gateways is expected to increase to as many as ten this year. Though the D0 will eventually be used in this capacity, they will not be available for this application for some time.

The meeting was then adjourned to permit attendees to see the Boca Raton Insurance tape, hosted by John Ellenby, and demonstrations of the touch screen (Dave Moulding), Smalltalk (Alan Kay), FIRST (Bob Datolla), and HSIL (Marion Suggs, Paul Lam).

ALTO MAINTAINERS MEETING - A meeting of Alto maintainers was held of February 8th, 1978 at El Segundo. The meeting was hosted by Doug Stewart, SPG. The primary subjects of discussion were hardware problem areas, centralized repair reporting, and SPG repair service.

The biggest problem area seemed to be the disk drives. Typical adjustments for the read gate are 460/440 n sec for the long/short one shots though this may vary from drive to drive. Also, the write head current is normally cut past track 128 due to the reduced track length. Cutting resistor F-63 on the J-10 board to raise write head current is common but Diablo advises against it suggesting instead that the value of resistor H-64 (part of the same voltage divider network) be varied starting with 1K and working down as necessary. The heads should be cleaned periodically (approximately 3 months) using a lint free material such as TexWipes. Q-Tips should not be used as they will leave fibers on the head. Alignment is generally performed after cleaning heads. Some groups keep spare heads for replacements.

The 15 volt Sorenson power supply (and to an extent the 12 volt supply) is the other major problem area. As these units are under a five year warranty they should be returned to the manufacture. Sorenson will also update units returned for repair.

It's useful to have a few memory chips on hand as this is the most common chip failure and it is easy to repair. Bad 16K memory chips should be returned to Terry Haney for failure evaluation. Memory Chips may be purchased from SPG. These are the only chips available from that group.

Keyboards have multiple character and mechanical sticking problems. The Keytest diagnostic can be used to adjust the key producing multiple characters. The electronics are on the AIM module and keyboard printed wiring board.

Modules will be repaired by SPG in conjunction with their other activities. No headcount is specifically assigned to repair activity. Boards should be returned to Terry Haney *along with a description of the problem* and, if applicable, a copy of printer output.

There is considerable interest in developing a repair data base. The only data of this type currently available is maintained by Jim Hall's 1200 group on the machines maintained by them under contract. Doug Stewart will set up a mechanism for collecting failure information including net address, date, subsystem affected, serial number (if applicable), failure symptoms, and corrective actions. Jim Iverson reports that a paper log is currently being kept for each machine in his group for the convenience in the multiple user environment and to identify recurring failures in a specific unit.

It was requested that Frank Ludolph set up a system to more quickly disseminate maintenance information.

ALTO MAINTAINER'S MESSAGE LIST - An immediate result of the Alto maintainers meeting is the establishment of a MAXC MSG distribution list file, <Secretary>AltoMaintainers.msg, to simplify the communication of general interest information among Alto maintainers. To use this feature when sending a message, the response to "TO:" is "**↑b** <Secretary>AltoMaintainers.msg CR CR". The rest of the sndmsg procedure is normal. The message will be sent to all accounts listed in the .msg file.

MESSAGING FOR SPECIAL INTEREST GROUPS - The same messaging mechanism referred to in the preceding item is used by several special interest groups including; AIS, PROM (ProLog users), SIL, and AltoMaintainers. If you are actively involved in any of these are sndmsg to Jennette <Jenkins> for inclusion. A complete listing of all distribution lists can be retrieved from [MAXC]<Secretary>All.masterlist.

DON'T LEAVE YOUR DISK IN AN ALTO - As pointed out in a recent issue of SDD's Random Items, a disk is locked inside the Alto's disk drive when power is removed from the unit or when the 15 volt power supply fails. Failure of this supply is one of the most common Alto ailments. *It is suggested that you not leave your disk in an Alto overnight.*

TOOLS

HARDWARE

NEW HARDWARE MANUAL - The Alto Hardware manual has been revised and made available in Press format. If your print server does not have two disk drives, the file <AltoDocs>AltoHardware.press may have to be broken into two pieces using PRESSEEDIT and the pieces sent to the printer.

ORBIT BUG - Severo Ornstein reports that there is a timing problem in the Orbit adapter. It appeared that Pimlico had problems aligning the successive color passes on a page. In reality the Output Scanline counter (SLN/SLWN) wasn't resetting properly due to a race situation resulting from anding the clock pulse with the clear level using an N163 (synchronous clear). This situation also exists with Dover but isn't very noticable because it shifts the image by only a fraction of a band.

The fix is simple; replace the three N163s in locations G2, G3, and G4 on the Input board with N161s. *Severo suggests that the fix be made on all Orbits*, regardless of attached printer, because it could create a very difficult bug to locate someday if printers are exchanged.

Whole ALTO World Newsletter

4

MAKING A DUAL-DRIVE ALTO - Doug Stewart has written a memo listing the items necessary to connect a second drive to the Alto. All items can be ordered directly from Diablo. The memo is appended to the Newsletter.

DISK DIAGNOSTIC DOCUMENTATION - Jim Cucinitti recently wrote some documentation for the Model 31 disk diagnostics that have been in use for quite sometime. It describes diagnostic initiation, use of the debugger, understanding the failure data, and modification of the diagnostics. *It is intended for maintainers only.* The document, which includes assembler listings of the programs, can be retrieved from [MAXC]<AltoDocs>DiskDiag.press.

PROM DESTRUCTION BY THE PROLOG PROGRAMMER - Tom Chang informs us that the ProLog PROM programmer will often destroy the last location of a PROM in socket PM when powered down. The ProLog people advise that *PROMs should always be inserted and removed from the socket with the power on.*

COLOR DISPLAYS AND THE ALTO - Every now and then Dick Shoup is asked about the use of color displays with the Alto. While it has been done, the results were generally unsatisfactory. Dick has written a memo on this topic which has been attached to this Newsletter.

SOFTWARE

In general, the subsystems, packages, and documentation indicated here will be available from your local IVY server under the directories <Alto> and <AltoDocs>. If they are not available, or if you are in doubt as to the version, they may be retrieved from [MAXC] (same directories). Files stored under other directories are on [MAXC] unless otherwise indicated, e.g. [XEOS].

NEW RELEASE: CONDENSE.RUN - This recently released program by Keith Knox will retrieve the screen bitmap from the SWAT and SWATEE files. The bitmap can be displayed or output to a file in either AIS or PRESS format. Documentation will be forthcoming shortly, but isn't really required for operation as the menus tell all. Retrieve [WRC]<IPA>Subsystems>Condense.run.

NEW RELEASE: AISdump.run - A new dump program, part of the AIS System, will write out the pixels as decimal values for an 8 bit/pixel or 1 bit/pixel AIS file to a file on the Diablo disk. Since the dump file is a text file, it is a 4:1 or 16:1 expansion, so be careful how large a window you choose. Retrieve [WRC]<AIS>Subsystems>AISdump.run.

ReReleases - Subsystems

AISmagnify - This new version, 2.0, has a menu, runs a little faster, and some new features. Retrieve [WRC]<AIS>Subsystems>AISmagnify.run. The documentation is [WRC]<AIS>MEMOS>AISmagnify.press.

BRAVO - The new version, 7.1, contains bug fixes and implements the new time standard. It will be retrieved and installed automatically when installing the new operating system. Documentation on the new color facilities can be retrieved from [IRIS]<Bravo>ColorBravoChanges.bravo.

CHAT - This rerelease, TTY version 9, Display version 15, contains bug fixes and minor improvements. Retrieve <Alto>Chat.run. The documentation, Chat.tty, is updated to include the I and O commands which toggles the USER.cm entry TYPESCRIPTCHARS.

COPYDISK - This subsystem, found on boot servers, has been updated to include the new time standard.

DMT - This subsystem, found on boot servers, has been updated to include the new timestandard. Also, the bug in the Dec. 10 version, which fails to indicated the bad RAM chip location, has been corrected.

IFS - The new release, 1.14, includes commands for accessing and manipulating file protections. Users should retrieve and read <IFS>HowToUse.press.

PRESSEEDIT - An experimental release of this subsystem can be found on <Newman>PressEdit.run. The documentation PressEdit.tty is on the same directory. It provides a new, simpler method of combining illustrations with text documents. Official release will occur in March after sufficient testing. The experimental version is reasonable robust.

PROM - The nature of the changes is unknown to me. Retrieve <Alto>PROM.run. New documentation is available on <EOD>PROM.bravo.

SCAVENGER - The nature of the changes is unknown to me. The documentation is unchanged. Retrieve <Alto>Scavenger.run.

SETTIME - The new version implements the new timestandard. Is is automatically retrieved when installing the new operating system with NewOS.cm.

SIL - Several changes have been made and are summarized in <SIL>SILupdates.press. Retrieve <SIL>SIL.run. The documentation SILmanual.press and SILsummary.press, also on <SIL>, have been revised.

ReReleases - Packages

ALTODEFS, ALTOFILESYS, DISKS, STREAMS, SYSDEFS - These definition files have changed in conjunction with the new operating system. If they currently reside on a disk they will be updated when NewOS.cm is run. For a description of changes see the change history in the rereleased OS Manual.

TECHNOLOGY

This month's paper, *GUS, A Frame-Driven Dialog System* by Daniel Bobrow, Ronald Kaplan, Martin Kay, Donald Norman, Henry Thompson, and Terry Winograd, is the third in a series on methods of making machines more amenable to the naive user. While this work is strictly research and not being performed on Altos, it gives us a glimpse of possible future directions.

The Understander project at PARC is exploring the process of language comprehension and the cognitive structures and operations which underlie it. GUS was written to assess progress and suggest area of future effort.

GUS, A Frame-Driven Dialog System

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay,
Donald A. Norman, Henry Thompson, Terry Winograd¹

*Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304*

GUS is the first of a series of experimental computer systems that we intend to construct as part of a program of research on language understanding. In large measure, these systems will fill the role of periodic progress reports, summarizing what we have learned, assessing the mutual coherence of the various lines of investigation we have been following, and suggesting where more emphasis is needed in future work. GUS (*Genial Understander System*) is intended to engage a sympathetic and highly cooperative human in an English dialog, directed towards a specific goal within a very restricted domain of discourse. As a starting point, GUS was restricted to the role of a travel agent in a conversation with a client who wants to make a simple return trip to a single city in California.

There is good reason for restricting the domain of discourse for a computer system which is to engage in an English dialog. Specializing the subject matter that the system can talk about permits it to achieve some measure of realism without encompassing all the possibilities of human knowledge or of the English language. It also provides the user with specific motivation for participating in the conversation, thus narrowing the range of expectations that GUS must have about the user's purposes. A system restricted in this way will be more able to guide the conversation within the boundaries of its competence.

MOTIVATION AND DESIGN ISSUES

Within its limitations, GUS is able to conduct a more-or-less realistic dialog. But the outward behavior of this first system is not what makes it interesting or significant. There are, after all, much more convenient ways to plan a trip and, unlike some other artificial intelligence programs, GUS does not offer services or furnish information that are otherwise difficult or impossible to obtain. The system is interesting because of the phenomena of natural dialog that it attempts to model and because of the principles of program organization around which it was designed. Among the hallmarks of natural dialogs are unexpected and seemingly unpredictable sequences of events. We describe some of the forms that these can take below. We then go on to discuss the modular design which makes the system relatively insensitive to the vagaries of ordinary conversation.

¹ This work was done by the language understander project at the Xerox Palo Alto Research Center. Additional affiliations: D. A. Norman, University of California, San Diego; H. Thompson, University of California, Berkeley; and T. Winograd, Stanford University. To appear in *Artificial Intelligence*, Spring 1977 (8:1)

Problems of natural dialog

The simple dialog shown in Figure 1 illustrates some of the language-understanding problems we attacked. (The bracketed numbers are for reference in the text). The problems illustrated in this figure, and described in the paragraphs below, include: allowing both the client and the system to take the initiative, understanding indirect answers to questions, resolving anaphora, understanding fragments of sentences offered as answers to questions, and interpreting the discourse in the light of known conversational patterns.

Mixed Initiative. A typical contribution to a dialog, in addition to its more obvious functions, conveys an expectation about how the other participant will respond. This is clearest in the case of a question, but it is true of all dialog. If one of the participants has very particular expectations and states them strongly whenever he speaks, and if the other always responds in such a way as to meet the expectations conveyed, then the initiative remains with the first participant throughout. The success of interactive computer systems can often be traced to the skill with which their designers were able to assure them such a dominating position in the interaction. In natural conversations between humans, however, each participant usually assumes the initiative from time to time. Either clear expectations are not stated or simply not honored.

GUS attempts to retain the initiative, but not to the extent of jeopardizing the natural flow of the conversation. To this extent it is a mixed-initiative system (see Carbonell, 1970a, 1970b). This is exemplified in the dialogue at [1] where the client volunteers more information than GUS requested. In addition to his destination, the client gives the date on which he wants to travel. Line [3] illustrates a case where the client takes control of the conversation. GUS had found a potentially acceptable flight and asked for the client's approval. Instead of either giving or denying it, the client replied with a question of his own.

Indirect answers. It is by no means always clear what constitutes an answer to a question. Frequently the purported answer is at best only a basis on which to infer the information requested. For example, when GUS asks "What time do you want to leave?" it is seeking information to constrain the selection of a flight. The client's response to this question, at [2], does constrain the flight selection, but only indirectly. In general, the kinds of inference necessary to interpret indirect answers can be very complex. GUS can handle the problem in its simpler manifestations, thanks largely to the very narrow expectations it has about the subject matter and the client's goals.

Resolving anaphora. A central issue in dialog is ensuring that both parties understand a given word or phrase as referring to the same object or event (see Norman and Rumelhart, Chapter 3, 1975 for a discussion of the general problem of reference). One problem occurs when a short phrase refers back to something that was introduced earlier. GUS provides for this problem in some of its simpler manifestations. At line [3], for example, the client's query refers to the flight mentioned in GUS's immediately preceding utterance. In [4] there is a reference to the flight mentioned earlier in the conversation, at [2]. Note that "next flight" in [3] was to be interpreted relative to the order of flights in the airline guide whereas "first one" in [4] refers to the order in which the flights were mentioned. Another

implicit referent underlies the use of "Friday" to specify a date in [6]. Resolution of this reference requires some complicated reasoning involving both the content and the context of the conversation. Since May 28th has been given as the departure date, it must presumably be the following Friday that the client has in mind. On the other hand, suppose that the specifications were reversed and Friday had been given as the departure date at line [1]. It would then be most readily interpretable as referring to the Friday immediately following the conversation.

-
- GUS: Hello. My name is GUS. I can help you plan a simple trip by air.
Where do you want to go?
- [1] *Client*: I want to go to San Diego on May 28
- GUS: What time do you want to leave?
- [2] *Client*: I must be in San Diego before 10 AM
- GUS: Would you like P.S.A. flight 102 that arrives at 9:15 AM?
- [3] *Client*: What is the next flight?
- GUS: Air California flight 310 that leaves at 8:30 AM and arrives at 10:00 AM
- [4] *Client*: I'll take the first one
- GUS: In what name should I make a reservation, please?
- [5] *Client*: Dan Bobrow
- GUS: I have confirmed the following flight:
P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego
that leaves at 7:30 AM and arrives at 9:15 AM.
What date do you want to return on?
- [6] *Client*: On Friday in the evening
- GUS: Would you like the flight that leaves at 7:45 PM?
- [7] *Client*: That's fine.
- GUS: I have confirmed the following flight:
P.S.A. flight 307 on Friday May 30 from San Diego to San Jose
that leaves at 7:45 PM and arrives at 9:30 PM
Thank you for calling. Goodbye.

Figure 1. A transcript of an actual dialog with GUS

Sentence fragments. Utterances in natural conversation are by no means always complete sentences. This is not to say that there are no rules governing the ways in which fragments can be used. We collected a number of dialogs between people and examined the sentence

fragments that occurred: most appeared as answers to direct questions. Furthermore, a rule can almost invariably be derived from a question that will convert a fragmentary answer into a complete sentence expressing the same information. For example, the client's response in [5] to the request for a name is not a sentence but, when inserted in the blank space in the skeleton "You should make the reservation in the name of _____", it yields a sentence. Normal processing of the sentence so constructed gives the required interpretation of the fragment. This works even for the fragment in [6] which is not even a complete phrase.¹

¹The SRI speech system (Walker, et al., 1975) uses a number of other techniques for handling a different set of fragments.

These skeletons are systematically related, in the sense of transformational grammar, to the corresponding questions. The blank space in the skeletons usually occurs at the end. If Sgall and the linguists of the modern Prague school are right, then this follows from a strong tendency to organize sentences so that given information comes at the beginning and new information at the end. In this case, the given information is clearly that which is shared by the question and its answer.

Conversational patterns. Conversations conform to patterns, which are still only poorly understood, and there are specialized patterns that are used in special circumstances such as those that obtain in a travel agency. Realism requires that GUS fit its conversational strategy to these patterns. For example, flights are usually specified by departure time, but in response to [2], GUS specifies an arrival time, because the client had specified the arrival time to constrain the choice of flights. This is in accordance with a typical conversational convention; a speaker says as little as will suffice to communicate the point to be made. Grice [1975] calls these conventions conversational postulates and implicatures.

It seems also to be important to use conversational implicatures with respect to the goals of the client and the system in interpreting and generating the dialog (see Gordon & Lakoff (1972) for a general discussion of this issue). For example, in [1] the client says where he wants to go. GUS interprets this as a request for an action, that is, inserting the appropriate information into the travel plan being generated.

Principles of program organization

One of the major methodological issues we addressed in designing and building GUS was the question of *modularity*. We realize that language understanding systems, and other systems exhibiting some degree of intelligence, will be very large and complicated programs, and the flow of processing within them will be correspondingly complex. As Simon (1969) has pointed out, one way of reducing the complexity of a system is to decompose it into simpler, more readily comprehensible parts, and to develop and debug these in isolation from one another. When the separate modules have been constructed, however, the task of integrating them into a single system still remains. This can be difficult: truly complex systems are more than just the sum of their parts. The components, when put together, interact in subtle

but important ways. We implemented GUS in order to determine whether a modular approach for a dialog system was at all feasible and to test our notions of what reasonable lines of decomposition might be. We are aware of alternative decompositions, and are not committed to this one; it was convenient given the program modules already available, and the issues we wished to focus on. GUS provided a context in which to explore tools and techniques for building and integrating independent modules.

The major knowledge-oriented processes and structures in GUS--the morphological analyzer, the syntactic analyzer, the frame reasoner, and the language generator--were built as independent processes with well defined languages or data structures to communicate across the interfaces. They were debugged separately, and tied together by means of an overall asynchronous control mechanism.

Control: The organization of the system is based on the view that language-understanding systems must operate in a multiprocess environment (Kaplan, 1973b, 1975). In a system with many knowledge sources and a number of independent processes, some part of the mechanism must usually be devoted simply to deciding what shall be done next. GUS puts potential processes on a central *agenda*. GUS operates in a cycle in which it examines this agenda, chooses the next job to be done, and does it. In general, the execution of the selected task causes entries for new tasks to be created and placed on the agenda. Output text generation can be prompted by reasoning processes at any time, and inputs from the client are handled whenever they come in. There are places at which information from a later stage (such as one involving semantics) are fed back to an earlier stage (such as the parser). A supervisory process can reorder the agenda at any time. This process is similar in function to the control module in the BBN Speechlis system (Woods, 1974; Rovner, Nash-Webber & Woods, 1974), except that it can resume processes which are suspended with an active process state. Preserving the process state is necessary because the flow in the system is not unidirectional: for example, the state of the syntactic analysis cannot be completely abandoned when domain dependent translation starts. If a semantically and pragmatically appropriate interpretation of an utterance cannot be found from the first parsing, the syntactic analyzer must resume where it was suspended. INTERLISP's coroutine facility makes it possible to completely preserve the active state of the various processes (Teitelman, 1976; Bobrow & Wegbreit, 1974).

Procedural attachment. Broadly speaking, procedural attachment involves redrawing the traditional boundary between program and data in such a way as to give unusual primacy to data structures. Most of the procedures that make up a program, instead of operating on separate data structures, are linked to those structures and are activated when particular items of data are manipulated in particular ways. This technique lies at the heart of the reasoning component which is described in more detail later. It provides a natural way of associating operations with the classes or instances of data on which they are to operate. It is in some ways extensions of ideas found in SIMULA (Dahl & Nygaard, 1966) and SMALLTALK (Goldberg and Kay, 1976).

Monitoring and debugging: In a multiprocessing system with processes triggered by procedures attached to complex data structures, special tools are needed for programmers to

monitor the flow of control and changes in the data structures. Tightly linked with the agenda scheduler there is a central monitor with knowledge about how to summarize the current actions of the system. The monitor interprets special printing instructions associated with potential actions and particular items of data. In effect, the principle of procedural attachment has been extended to debugging information.

External data-bases: We believe that an important application of specialized dialog systems like GUS may be to help users deal with large files of formatted data. In the travel domain, the *Official Airline Guide* is such an external data-base. GUS can use an extract of this data-base, but the information in the file does not form part of its active working memory for the same reason that the information in the *Official Airline Guide* does not have to be memorized by a travel agent. Only that portion of the data base relevant to a particular conversation need be brought into the working memory of the system.

PROCESSES AND KNOWLEDGE BASES

Figure 2 illustrates the knowledge structures and processes in GUS. Each numbered row corresponds to a single knowledge based process in the system. The input to each process is shown in the left hand column. Each input is labelled with a number in parentheses indicating the row number of the process which produces it. Processes usually provide input to the ones listed below them. The third column names the process which produces the output structures specified in the fourth column, using for the processing the permanent knowledge bases specified in column two.

Input Structures	Permanent Knowledge Structures	Processes	Output Structures
1. Text String word (input) structures	Stem dictionary; Morphological rules	Dictionary lookup; Morphological analysis	Chart of data
2. Query context(6); of a Chart(1) sentence	Transition net grammar	Syntactic analysis	Parsing
3. Parsing of a frame sentence (2)	Case-frame dictionary	Case-frame analysis	Case- structure
4. Case-frame Frame change structure (3) description	Speech patterns; Domain specific frame forms	Domain dependent translation	
5. Frame change Frame change descriptions(4,5); descriptions Current frame response instances (5) descriptions;	Prototype frames and attached procedures	Frame reasoning	Output Current

frame

instances

6. Output response	Dialog query map;	Response	
English text;			
description (5)	Flight description	generation	Query
context			
	template		

Figure 2. Knowledge structures and Processes in GUS

Figure 3 shows the output structures of the earlier stages of processing of the sentence "I want to go to San Diego on May 28". Starting with an input string of characters typed by the client, a sequence of words is identified by a lexical analyzer consisting of a dictionary lookup process and a morphological analysis. The analysis program has access to a main dictionary of more than 3,000 stems and simple idioms and a body of morphological rules specifying how the information in the dictionary can be used to partition character sequences into known lexical items (Kay & Kaplan, 1976). The output of this stage is a *chart* (Kay, 1973), a table of syntactic and semantic information for use by the parser.

CLIENT: I want to go to San Diego on May 28

```
[S MOOD =DCL                                ...the syntactic analysis of the input
  SUBJ =[NP HEAD =[PRO CASE =NOMIN NUMBER =SG ROOT =I]]
  FVERB =[V TENSE =PRESENT ROOT =WANT] HEAD =WANT
  OBJ =[S MOOD =FOR-TO
        SUBJ =I
        HEAD =[V TENSE =PRESENT ROOT =GO]
        MODS =(
          [PP PREP =[PREP ROOT =TO]
            POBJ =[NP HEAD =[NPR PROPERTYPE =CITY-NAME
                          ROOT = SAN-DIEGO]]]
          [PP PREP =[PREP ROOT =ON]
            POBJ =[NP HEAD =[NPR PROPERTYPE =DATE-NAME
                          MONTH =MAY DAY =28]]]]]
```

```
[CLIENT DECLARE                                ...the case-frame structure
  (CASE FOR WANT/E (TENSE PRESENT)
    (AGENT (PATH DIALOG CLIENT PERSON))
    (EVENT (CASE FOR GO (TENSE PRESENT)
      (AGENT (PATH DIALOG CLIENT PERSON))
      (TO-PLACE (CASE FOR CITY
        (NAME SAN-DIEGO)))
      (DATE (CASE FOR DATE
        (MONTH MAY))
```

(DAY 28]

```

CMD: [ CLIENTDECLARE           ... the domain dependent translation, a
      (FRAME ISA TRIP-LEG      ... frame change description
      (TRAVELLER (PATH DIALOG CLIENT PERSON))
      (TO-PLACE (FRAME ISA CITY
                (NAME SAN-DIEGO)))
      (TRAVEL-DATE (FRAME ISA DATE
                  (MONTH MAY)
                  (DAY 28])

```

Figure 3. Processing the client's first utterance

The syntactic analyzer is based on the General Syntactic Processor (Kaplan, 1973a). Using a transition-network grammar and the chart, the parser builds one or more canonical syntactic structures, depending on whether or not the sentence is syntactically ambiguous. It finds one parse, and can continue to find others if the sentence is ambiguous and the first parse is rejected as uninterpretable by a later process. The syntactic analysis of the input sentence is shown in Figure 3.

The case-frame analysis uses linguistic knowledge associated with individual lexical items to relate their appearance in canonical syntactic structures to their uses in a semantic environment. It uses a dictionary of case-frames based on the ideas of case grammar originated by Fillmore (1968; see Bruce, 1976 for a general review of case systems). This component uses knowledge about such things as selectional restrictions and the mapping between surface cases (including prepositions) and semantic roles. As seen in Figure 3, the cases for GO are AGENT, TO-PLACE, and DATE.

As we have already observed, interpretation of an utterance must include knowledge of conversational patterns for the appropriate domain. Domain dependent interpretations of utterances were implemented by a simple structure-matching and reconstruction program that operates on case-frames. The example in Figure 3 illustrates how the domain-dependent translation module handles a common conversational pattern for the travel domain: it interprets a statement of desire (the WANT/E) as an instruction to insert the specified event into the trip plan being constructed. In addition, the case frame involving GO is transformed into a description of the TRIP-LEG which is part of the planned trip, with the AGENT of GO becoming the TRAVELLER in the TRIP-LEG and the DATE becoming the TRAVEL-DATE. This simple translation mechanism is obviously very limited; in a more realistic system, the purposes of the client would have to be understood more deeply.

The frame reasoner component of the system was the focus of most of the research and development. It was based on the assumption that large scale structures closely tied to specific procedures for reasoning constitute a framework for producing a mixed initiative dialog system. It uses the frame change description (labelled CMD in Figure 3) to fill in the appropriate information in the trip plan it is building and trigger associated reasoning, as

described later.

The generation of output English is guided by a query-map, a set of templates for all the questions that might be asked by the system. GUS uses a table lookup mechanism to find the appropriate template and generates the English by filling in the template form. This simple generation mechanism is sufficient for the dialog system; generation was not one of the areas of substantial work.

The module that generates questions for the client simultaneously produces one or more skeletons into which his responses can be inserted, if they do not prove to be sentences in their own right. What is being done here is surprisingly simple and works well for most of the fragments we have encountered in response to simple WH-questions. Note that the language generator communicates with the syntactic analyzer using English phrase fragments rather than using a specially constructed formalism. This contrasts with other approaches to the fragment problem, in which the various components of the system are more deeply affected.

THE REASONING COMPONENT

Frames: It is widely believed in artificial intelligence that intelligent processing requires both large and small chunks of knowledge in which individual molecules have their own sub-structure. Minsky's 1975 paper on *frames* discusses the issues and suggests some directions in which to proceed. But, as Minsky stated, his ideas were not refined enough to be a basis for any working system. Our intuitions about the structure of knowledge resemble Minsky's in many ways, and we have appropriated the word *frame*. However, our conceptions are by no means identical to Minsky's, and the two notions should not be confused. The frame structures used in this system were a first step towards a more comprehensive knowledge representation language whose current development is described in Bobrow and Winograd (1977).

Frames are used to represent collections of information at many levels within the system. Some frames describe the sequence of a normal dialog, others represent the attributes of a date, a trip plan, or a traveller. In general, a frame is a data structure potentially containing a *name*, a reference to a *prototype* frame, and a set of *slots*. Frame names are included primarily as a mnemonic device for the system builders and are not involved in any of the reasoning processes. In fact, names are not assigned to any of the temporary frames created during a dialog.

If one frame is the prototype of another, then we say that the second is an *instance* of the first. A prototype serves as a template for its instances. Except for the most abstract frames in the permanent data base, every frame in GUS is an instance of some prototype. Most instances are created during the process of reasoning, although some (for example those representing individual cities) are in the initial data base.

A frame's important substructures and its relations to other frames are defined in its slots. A slot has a *slot-name*, a *filler* or *value*, and possibly a set of attached procedures. The value of a slot may simply be another frame or, in the case of a prototype, it may be a description constraining what may fill the corresponding slot in any instance of the given frame. Figure 4 shows the prototype frame for *date* and the specific date *May 28*, which has no external name. The fact that it is an instance of *date* is indicated by the keyword *ISA* followed by the prototype name.

The *date* prototype illustrates several of the ways in which the values for instance slots can be described. For example, the slot labelled *MONTH* specifies that only a *name* can be used as value; that is, only a literal LISP atom. GUS interprets a standard set of type terms such as *name*, *integer*, *list*, and *string*. The slot for *WEEKDAY* stipulates that a value for that slot must be a member of the list shown in the frame. The slot *DAY* can only be filled by an integer between 1 and 31. The terms *BOUNDED-INTEGER* and *MEMBER* have no special meaning to the interpreter. Any LISP function may occur in this position as a predicate whose value must be non-NIL for any object filling the slot.

Not all of the slots of an instance frame need to be filled in. For example, in *May 28*, only the *MONTH*, and *DAY* are filled in, and not the *WEEKDAY*. A prototype frame provides slots

as placeholders for any data that might be relevant, even though it may not always be present. Only those slot values which are required for the current reasoning process need be put into instances.

```
[DATE
  MONTH   NAME
  DAY     (BOUNDED-INTEGER 1 31)
  YEAR    INTEGER
  WEEKDAY (MEMBER (SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY
                  FRIDAY SATURDAY))]
```

a. Prototype for date

```
[ISA DATE
  MONTH   MAY
  DAY     28]
```

b. The instance frame for May 28

Figure 4. Examples of frames

Procedural attachment: We have already referred to *procedural attachment*, a concept first discussed by this name by Winograd (1975), as a central feature of GUS. Procedures are attached to a slot to indicate how certain operations are to be performed which involve either the slot in the given frame or the corresponding slot in its instances. We have found that there are many slots for which some processing is best done by idiosyncratic procedures. For example, there may be special ways of finding fillers for them or for doing other kinds of reasoning about them. This might include verifying that the value in an instance is consistent with other known information or propagating information when the slot value is obtained.

The procedures associated with slots fall into two general classes: *servants* and *demons*. *Demons* are procedures that are activated automatically when a datum is inserted into an instance. *Servants* are procedures that are activated only on demand. The expanded date prototype in Figure 5 contains examples of both classes. On the slot WEEKDAY there is a demon marked by the keyword WHENFILLED and a servant marked by the keyword TOFILL. When a value is filled into the WEEKDAY slot of a date instance, the WHENFILLED statement on the prototype causes the interpreter to invoke the demon FINDDATEFROMDAY. This procedure attempts to compute the appropriate date to fill the other slots in the frame, using the name of the day just entered and contextual information to identify the value uniquely.

The servant GETWEEKDAY on the same slot is only invoked when the name of the week day

is needed. The requirement is satisfied by calling the LISP procedure GETWEEKDAY with the current instance as an implicit argument. The servant attached to the slot YEAR indicates how a default value can be filled in. If the year is given by the client, then this servant will never be activated. However, if the client does not mention the year explicitly, the system will fill in the default value 1975 when any part of the reasoning process calls for it.

```

-----
[DATE
  MONTH   NAME
  DAY     (BOUNDED-INTEGER 1 31)
  YEAR    INTEGER (TOFILL ASSUME 1975)
  WEEKDAY (MEMBER (SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY
                  FRIDAY SATURDAY))
          (WHENFILLED FINDDATEFROMDAY)
          (TOFILL GETWEEKDAY))
  SUMMARY (OR (LIST MONTH DAY) WEEKDAY)]

```

Figure 5. The frame for date with attached procedures and summary form

The system provides a number of standard servant procedures. ASKCLIENT causes the client to be asked for information that will determine the value of the slot. CREATEINSTANCE indicates that a new instance of a specified prototype should be created and inserted at that location. Some of the values of the newly created frame may be filled in by the procedure, others may be left to be filled through later reasoning or interaction with the client. In addition to standard servants, the builders of the system can program special procedures to compute appropriate values, such as the GETWEEKDAY mentioned earlier.

Summarizing data structures. In Figure 5, the frame for date includes a slot with the special name SUMMARY. A SUMMARY slot appears only in a prototype frame, never in an instance. It gives a format for describing the instances of the prototype to help programmers monitor and debug the system. Thus, instances of date will be described by printing the month and day, e.g. (May 28) or, if they are not known, just the day of the week.

USING FRAMES TO DIRECT THE DIALOG

Frames are used at several levels to direct the course of a conversation. At the top level, GUS assumes that the conversation will be of a known pattern for making trip arrangements. To conduct a dialog, the system first creates an instance of the **dialog** frame outlined in Figure 6. It goes through the slots of this instance attempting to find fillers for them in accordance with the specifications given in the prototype. When a slot is filled by a new instance of a frame, the slots of that instance are filled in the same way. GUS follows this simple depth-first, recursive process, systematically completing work on a given slot before continuing to the next. This is how GUS attempts to retain the initiative in the dialog. Notice, however, that slots may occasionally be filled out of sequence either through information volunteered by the client or by procedures attached to previously encountered slots.

In Figure 6, boldface atoms are frame names, representing pointers to other frames. (Substructures for the frames for **Person**, **Date**, **City**, **PlaceStay**, **TimeRange**, and **Flight** are not shown.) Each of the slots shown in Figure 6 must be filled in during the course of the dialog, usually by invoking a servant attached to the prototype slot. The servants for some slots calculate the desired values from other known data, or (as in the case of frames like **TripSpecification**) simply create a new frame. The servant **ASKCLIENT** obtains information needed to fill a slot by interrogating the client. The default organization of a dialog is determined by the order of the slots which have **ASKCLIENT** as servant, since appropriate questions will be asked if those slots have not been filled by the time they are encountered.

Now let us follow the system as it goes through part of a dialog, with special emphasis on the process of filling in the slots of frames. The dialog and the relevant information about the state of the system are shown in Figure 7. This figure is the beginning of an actual transcript of a session, and the information shown there is provided to allow us (in the role of system builders) to follow the actions of the system.

The dialog starts when GUS outputs a standard message ("Hello. My name is GUS. I can help you plan a simple trip by air."). At this point, GUS knows that it is about to conduct a dialog on travel arrangements, so it creates an instance of the prototype **Dialog** frame shown in Figure 6 and starts to try to fill its slots. (From now on, all numbers in brackets refer to the corresponding lines of the frames of Figure 6. All references to the dialog refer to Figure 7.) The slot **CLIENT** at [1] contains a servant which fills this slot, when necessary, by creating a new instance of **Person**. This is indicated in the first line of the transcript of Figure 7, where the instance of person is shown as {ISA PERSON}. After the slot is filled in, a demon associated with the **CLIENT** slot is triggered, which then puts the same person instance in the **TRAVELLER** slot in [16]. GUS fills the **NOW** slot in [2] by constructing a frame instance for today's date. It then creates a **TripSpecification** instance [3], summarized by **ROUNDTRIP TO ?** in the transcript of Figure 7, to fill the **TOPIC** slot [3].

	Slots	Fillers	Servants	Demons
Dialog				
[1]	CLIENT	Person	Create	Link to TRAVELLER
[2]	NOW	Date	GetDate	
[3]	TOPIC	TripSpecification	Create	
TripSpecification				
[4]	HOMEPORT	City	Default - Palo Alto	
[5]	FOREIGNPORT	City		Link to OUTWARDLEG, AWAYSTAY, INWARDLEG
[6]	OUTWARDLEG	TripLeg	Create	
[7]	AWAYSTAY	PlaceStay		
[8]	INWARDLEG	TripLeg	Create	
TripLeg				
[9]	FROMPLACE	City	FindFrom HOMEPORT	
[10]	TOPLACE	City	AskClient	
[11]	TRAVELDATE	Date	AskClient	
[12]	DEPARTURESPEC	TimeRange	AskClient	Propose-Flight- By- Departure
[13]	ARRIVALSPEC	TimeRange		Propose-Flight- By- Arrival, Link to DEPARTURESPEC
[14]	PROPOSEDFLIGHTS	(SetOf Flight)		
[15]	FLIGHTCHOSEN	Flight	AskClient	
[16]	TRAVELLER	Person	AskClient	

Figure 6. An outline of key frame structures for our dialog

At this point the Dialog frame has been completely filled in so GUS proceeds to fill in the slots of the TripSpecification frame. In [4], a HOMEPORT which is a City is required; GUS assumes, on the basis of an attached servant, that the home port is Palo-Alto. There is no attached servant to find the FOREIGNPORT in [5], so GUS just leaves that slot empty for the moment. When a TripLeg instance is created for the outward leg of the journey, GUS begins trying to fill its slots. A servant for FROMPLACE specifies that it should be filled with the city used for HOMEPORT in the TripSpecification frame, so PaloAlto is filled in. The first slot which has an ASKCLIENT servant is at [10], which requires a city to fill the TOPLACE in the TripLeg, which is the OUTWARDLEG of the TripSpecification [6]. GUS issues the command (CMD) shown at the bottom of Figure 7, which directs the generation of the English question. This is done by a rather elaborate table look up: the result is shown as the last line of Figure 7.

GUS: Hello. My name is GUS. I can help you plan a simple trip by air.

CLIENT = {ISA PERSON} in {ISA DIALOG}
 TODAY = (MAY 15) in {ISA DIALOG}
 TOPIC = (ROUNDRIP TO ?) in {ISA DIALOG}
 HOME-PORT = PALO-ALTO in (ROUNDRIP TO ?)
 FROM-PLACE = PALO-ALTO in (TRIP TO ?)

CMD: (GUSQUERY (DIALOG TOPIC TRIP-SPECIFICATION OUTWARD-LEG TRIP-LEG
 TO-PLACE CITY))

GUS: Where do you want to go?

Figure 7. The beginning of the transcript for the dialog

We continue the trace of the analysis in Figure 8, starting with the client's response to the question. The domain dependent translation contains the information needed to fill the frame slots. The result of the client's English input is that both the `TOPLACE` [10] and the `TRAVELDATE` [11] of the `TripLeg` are filled in.

The system then continues working its way through the entire tree specified by the frames, asking questions of the client. Many of the slots have demons which propagate information to other places in the data structure. For example, when the city that fills the slot `FOREIGNPORT` [5] is found, GUS will insert that same `City` as the place to stay in the `AWAYSTAY` [7]. The `FOREIGNPORT` city also serves as the destination of the `OUTWARDLEG` of the trip and the starting point of the return trip (the `INWARDLEG`). To handle this information, GUS establishes two instances of the frame `TripLeg`, one for the outward leg, the other for the inward leg, and puts the city names in the appropriate slots.

Once a departure specification (some time range before, near or after the desired flight departure) is determined, a demon attached to `DEPARTURESPEC` calls a program which uses this information to propose a flight. Each proposed flight is added to the slot for `PROPOSEDFLIGHTS` [14]. This slot can be used to resolve anaphoric references to flights, based on the order of their mention in the conversation. GUS then tries to determine which of the flights is appropriate to fill in the `FLIGHTCHOSEN` slot [15]. When that has been determined, it will ask for the name of the traveller and confirm the flight.

Many of the slots are marked in such a way that they need not be filled for the dialog to be completed. For example, the arrival specification [13] in each `TripLeg` frame is never requested. This slot is provided as a place to put constraints about the arrival of the flight, if the client volunteers information constraining the desired arrival time. Demons associated with that slot would then be activated to propose a flight based on the arrival time. In a

similar way, the AWAYSTAY slot in the trip specification [7], is never asked for. If the client specifies something about the time range of the AWAYSTAY, as he did in the dialog of Figure 1, there is a place to store that information in the frame structure and a demon to put it into the appropriate TripLeg.

CLIENT: I want to go to San Diego on May 28

```

CMD: [ CLIENTDECLARE                                 ... the domain dependent translation
      (FRAME ISA TRIP-LEG
        (TRAVELLER (PATH DIALOG CLIENT PERSON))
        (TO-PLACE (FRAME ISA CITY
                   (NAME SAN-DIEGO)))
        (TRAVEL-DATE (FRAME ISA DATE
                     (MONTH MAY)
                     (DAY 28])

```

```

TO-PLACE =SAN-DIEGO in (TRIP TO ?)                 ... filling in the requested information
TRAVEL-DATE =(MAY 28) in (TRIP TO SAN-DIEGO) ... and the volunteered information
downhen TO-PLACE is put in (TRIP TO SAN-DIEGO) ...propogating information to other slots
(LINK TRIP-SPECIFICATION FOREIGN-PORT CITY)

```

Figure 8. The reasoning from the first input utterance

Figure 9 illustrates how a sentence fragment is processed. GUS asks "What date do you want to return on?" Generation of the question also generates a context for the expected interpretation of the next answer. The context is an inverted form of the question; that is, "I want to return" is a potential prefix to the next response. The preposition "on" may be optionally inserted in this prefix. The client responds "on Friday in the evening". Since this is not a sentence, the question context is used in the interpretation and the actual parsed structure which is interpreted is derived from the sentence "I want to return on Friday in the evening."

The time is taken as a departure specification and the date is specified in terms of the day of the week. The day of the week is filled into the appropriate place and date, and then the demon associated with that slot in date is activated. That demon computes the date relative to the previous date specified in the conversation. The phrase *evening* is taken as being equivalent to "around 7:30 PM". From this departure specification, GUS proposes the flight that leaves nearest to that time. Information is provided to the client about the leaving time, not the arrival time, because the client constrained the choice of flight by leaving time.

GUS: What date do you want to return on? ... a query generated by GUS

The context of the next answer is:
 (I WANT TO RETURN ((ON) (*SKIP*))) -- ... The expected context of the query response

CLIENT: On Friday in the evening

CMD: [CLIENTDECLARE ... the domain dependent translation, including context
 (FRAME ISA TRIP-LEG
 (TRAVELLER (PATH DIALOG CLIENT PERSON))
 (TRAVEL-DATE (FRAME ISA DATE
 (WEEKDAY FRIDAY)))
 (DEPARTURE-SPEC (FRAME ISA TIME-RANGE
 (DAY-PART EVENING])

WEEKDAY =FRIDAY in {ISA DATE}
 down when WEEKDAY is put in {ISA DATE} ... triggering a demon to find the Friday's date
 (FINDDATEFROMDAY)
 DAY =30 in (MAY 30)
 DAY-PART =EVENING in {ISA TIME-RANGE} ...evening is interpreted as around 7:30 PM
 DEPARTURE-SPEC =(AT 7 30 PM) in (TRIP TO PALO-ALTO)
 down when DEPARTURE-SPEC is put in (TRIP TO PALO-ALTO)
 (PROPOSE-FLIGHT-BY-DEPARTURE) ... this demon proposes a flight using a departure spec

GUS: Would you like the flight that leaves at 7:45 PM?

CLIENT: That's fine.

Figure 9. Processing a sentence fragment

This sample dialog illustrates how GUS attempts to control a conversation by fitting it to the mold laid down in a structure of related frames. It has a place prepared in this structure for each piece of information that might potentially be used for making travel arrangements. It also has a strategy that will cause the pieces of information that the client must supply to be elicited in a natural order. The sequence of slots in the frames determines the usual course of the conversation, but it will change if, for example, the client volunteers information or asks questions.

REAL AND REALISTIC DIALOGS

There is an important difference between *real* and *realistic* conversations. The simple dialog in Figure 1 is a *realistic* conversation that was actually carried on with GUS. It is much too easy to extrapolate from that conversation a mistaken notion that GUS contained solutions to far more problems than it did. To get an idea of some problems that GUS does not approach, we collected a variety of travel dialogs that clients of a full-fledged system (perhaps the final version of GUS) might expect to conduct. We did this by simulating the system, asking the clients to arrange for round trip air flights between Palo Alto and San Diego, typing all queries and responses on the computer terminal, and pretending that a computer system was interacting with them. In fact, the role of GUS was played by an experimenter sitting at another computer terminal, airline guide, travel books, and calendar in hand, responding to the client.²

² The experimental dialogs were collected by Allen Munro in the LNR research laboratory at the University of California, San Diego.

GUS Do you want a flight leaving at 4:00 PM
 CLIENT Do you have something a little closer to 7
 GUS Do you want the flight at 7:00 PM

a) Interpreting politeness

GUS Do you want the flight arriving at 8:00 PM
 CLIENT When does it leave?
 GUS 6:30 PM
 CLIENT How much?
 GUS \$25.50 round trip

b) Some pronominal reference problems

GUS When would you like to return?
 CLIENT I would like to leave on the following Tuesday, but I have to be back before my first class at 9 AM.

c) Giving a reason for flight preference

Figure 10. Fragments of real dialogs, with a person simulating the role of GUS

The two participants -- client and experimenter -- were each seated in independent,

individual sound-isolated experimental booths. They communicated with a special experimental program (designed for tutorial instruction) that presented the experimenter's responses in a block presentation, so it appeared as a realistic approximation of a computer output, without the slow typing rate that would occur otherwise. The system delays were approximately what one would expect for the operation of a complex program (10 to 60 seconds response time).

Some of the problems we found were unexpected. For example, people spent a lot of time telling us about their thought processes and reasons. They made excuses for changing their minds. They hedged a lot about what they wanted. Figure 10a illustrates a type of conversational interaction our current system cannot even begin to handle. When the system proposes a flight at 4 PM, the client requests something *a little closer to 7*. A literal interpretation of that request would be to find a flight that is as close to 4 PM as possible, but in the direction of 7 PM: perhaps the 5:00 PM flight. That, of course, is not at all what was desired by the client. The human experimenter made the natural response of offering the flight that left at 7.

Figure 10b indicates some pronominal reference problems which we did not attack at all. When the client says "when does *it* leave" it is quite obvious that he wants the departure time of the flight referred to in the previous sentence. For his question "how much," a response that "all of the plane leaves" seems somewhat inappropriate. In this case, the client is not referring to the previous system response, but rather is asking about the cost of the flight. But a response such as "how much" can sometimes refer to the previous system response. Suppose the system had just stated "They serve food on that flight." In this case, the client's query could be appropriately interpreted by the system as referring to the quantity of food. GUS cannot solve the problem of determining when a response is meant to refer to the previous question and when it is not.

Figure 10c illustrates how people provide extra information about their motivations. In a system with a better model of human needs and desires, this would be useful for suggesting alternatives that might otherwise be ruled out.

CONCLUSION

Computer programs in general, and programs intended to model human performance in particular, suffer from an almost intolerable delicacy. If their users depart from the behavior expected of them in the minutest detail, or if apparently insignificant adjustments are made in their structure, their performance does not usually change commensurately. Instead, they turn to simulating gross aphasia or death. The hope, which has been at least partially realized in GUS, is that the notions of procedural attachment and scheduling, as well as being realistic cognitive models, will make for more robust systems. We were pleased, for example, by the way the system's expectations could evolve in the course of a single conversation. The client would occasionally seize the initiative, volunteering information that was not asked for or refusing to answer a question as asked and GUS was able to respond appropriately in many cases. It would be misleading to press these claims too far. GUS never reached the stage where it could be turned loose on a completely naive client, however

cooperative. But, to one familiar with other systems of the same general kind, the impression of increased robustness is clear.

GUS represents a beginning step towards the construction of an intelligent language understanding system. GUS itself is not very intelligent, but it does illustrate what we believe to be essential components of such a system. An intelligent language understander must have a high quality parser, a reasoning component, and a well structured data base of knowledge. The knowledge is of several types, from language specific information and expertise in the topic areas in which it can converse to broad general knowledge of the world that must be used to interpret people's utterances. This knowledge tends to be taken for granted by most native speakers of the language, hence often left for the listener to infer. The system must be capable of giving direction to the conversation, but it must also be flexible enough to respond to novel directions set by the clients. The system must be able to make use of a large external data base and to understand what information must be retrieved and processed in depth. There must be an intimate connection between its representation of structural knowledge and the procedures used to process knowledge. A general framework for representing knowledge must be able to encompass all the different necessary forms of knowledge. In our future studies of GUS, we intend to broaden the general framework for representing knowledge, as well as to increase the power of the components of the system. Preliminary steps in this direction include the development of improved systems for language analysis (Kay & Kaplan, 1976) and a knowledge representation language (KRL: Bobrow & Winograd, 1976).

References

- Bobrow, D. G. & Collins, A. M. (Eds.) *Representation and Understanding: Studies in Cognitive Science*. New York: Academic Press, 1975.
- Bobrow, D. G. & Wegbreit, B., A model and stack implementation of multiple environments, *Communications of the ACM*, 1973, 16, 591-603
- Bobrow, D. G. & Winograd, T. An overview of KRL, a Knowledge Representation Language. *Cognitive Science*. Vol 1. No 1. 1977
- Bruce, B., Case systems for natural language. *Artificial Intelligence*, 1975, 6, 327-360.
- Carbonell, J. R. AI in CAI: An artificial intelligence approach to computer-aided instruction. *IEEE Transactions on Man-Machine Systems*, 1970, MMS-11, 190-202.
- Carbonell, J. R. Mixed-initiative man-computer instructional dialogues. Unpublished Ph.D. dissertation. Cambridge, Mass: Massachusetts Institute of technology, 1970.
- Dahl, O. J., & Nygaard, K., SIMULA--an ALGOL-Based Simulation Language, *Communications of the ACM*, 1966, 9, 671-678.
- Fillmore, C. The case for case. In E. Bach and R. T. Harms (Eds.), *Universals in Linguistic Theory*. New York: Holt, 1968.
- Goldberg, A. & Kay, A. (Eds.) SMALLTALK-72 instruction manual. Xerox Palo Alto Research Center SSL-76-6. Palo Alto, Ca. 1976
- Gordon D., & Lakoff, G., Conversational postulates, Papers from Seventh Regional Meeting, Chicago Linguistic Society, Chicago: University of Chicago Linguistics Department, 1972.
- Grice, H. P. Logic and conversation. In P. Cole & J. L. Morgan (Eds.), *Studies in Syntax*, Volume III. New York: Seminar Press, 1975.
- Kaplan, R. A general syntactic processor. In R. Rustin (Ed.), *Natural language processing*. New York: Algorithmics Press, 1973a.
- Kaplan, R. A multi-processing approach to natural language. *Proceedings of the 1973 National Computer Conference*. Montvale, N.J: AFIPS Press, 1973b.
- Kaplan, R. On process models for sentence analysis. In Norman, D. A., Rumelhart, D. E., and the LNR Research Group. *Explorations in cognition*, San Francisco: Freeman, 1975.

Kay, M. The MIND system. In R. Rustin (Ed.) *Natural language processing*. New York: Algorithmics Press, 1973.

Kay, M. & Kaplan, R. Word recognition. Palo Alto, California: Xerox Palo Alto Research Center, 1976.

Minsky, M. A framework for representing knowledge. In P. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 1975.

Reddy, D. R., Erman, L. D., Fennell, R. D., & Neely, R. B. HEARSAY speech understanding system: An example of the recognition process. *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

Norman, D. A., Rumelhart, D. E. and the LNR Research Group, *Explorations in cognition*. San Francisco: Freeman, 1975.

Rovner, P., Nash-Webber, B., & Woods, W. A. Control concepts in a speech understanding system. *Proceedings of the IEEE Symposium on Speech Recognition*, Carnegie-Mellon University, April 1974.

Simon, H. *Sciences of the artificial*. Cambridge: Massachusetts Institute of Technology Press, 1969.

Teitelman, W. INTERLISP reference manual. Palo Alto, California: Xerox Palo Alto Research Center, December, 1975.

Walker, D., Paxton W., Robinson, J., Hendrix, G., Deutsch, B., Robinson, A. Speech understanding research. Annual report, Project 3804. Artificial Intelligence Center. Stanford Research Institute, 1975.

Winograd, T. Frames and the declarative-procedural controversy, In D. G. Bobrow and A. M. Collins (Eds.), *Representation and Understanding*. New York: Academic Press, 1975.

Woods, W. A. Motivation and overview of BBN SPEECHLIS: An experimental prototype for speech understanding research. *Proceeding of the IEEE Symposium on Speech Recognition*, Carnegie-Mellon University, April 1974.