TOWARDS AUTOMATING EXPLANATIONS

R. E. Culllngford, M. W. Krueger, M. Selfridge, and M. A. Bienkowski

Department of Electrical Engineering
and Computer Soience
The University of Connecticut
Storrs, CT 06268

ABSTRACT

This paper discusses an approach to the modelling of the explanation process within the framework of a graphics-based CAD system currently under development, which can describe its own use, including the common ways to make and recover from errors. With a coordinated textual and pictorial display, the system, CADHELP, simulates an expert demonstrating the operation of the graphical features of the CAD tool. It consults a knowledge base of feature scripts, built up using situational script and commonsense algorithmic methods, to explain a feature, generate prompts as the feature is being operated, and to give certain types of "help" when a feature is misused. CADHELP provides these services by summarizing the feature script in different ways depending upon what it has told the user previously. The summarization process is based upon a series of "sketchification" strategies, which prescribe which parts bf a knowledge structure, a causal chain, or a single concept can be thrown away, since the listener should be able to infer them.

I. Introduction

The ability to explain things is an important, but poorly understood, component of intelligent behavior. This paper discusses an approach to the modelling of the explanation process within the framework of a graphics-based CAD system, called CADHELP, which can explain its own operation to its user, including the common ways to make and recover from errors. Functionally, the CAD system is a familiar sort of graphical tool for the design domain of digital logic circuits. It provides a standard set of graphical features, such as drawing, dragging and rubber-band line techniques, and accepts user input during, design from such standard devices as terminal, light pen and data tablet.

CADHELP contains detailed knowledge Structures, which simulate an expert user's knowledge of how the basic graphical features work, or may fail to work. CADHELP explains itself by demonstration, i. e., by mimicking an expert operating the device while explaining it. The explanation mechanism contains a set of natural-language generation heuristics which allow it to create explanatory text or prompts from the underlying knowledge structure in a context-sensitive manner. CADHELP is primed to tell a user only what he, as a result of an explicit command or by dialog history, does not already know. Therefore, subsequent explanations reflect the knowledge the user gained from earlier ones by omitting detailed restatement of concepts he should now understand.

The research summarized here was sponsored by the Advanced Research Projects Agency of the Department of Defense, and monitored by the Office of Naval Research under Contract N00014-79-C-0976. The support of these agencies is gratefully acknowledged.

II. An Example

In this section, we present a sample explanation which CADHELP has produced. The user selects the graphical feature to be explained, and the level of the explanation (viz., summary, normal operation, or errors) with simple typed input. CADHELP cannot "understand" natural language input, and thus cannot engage in mixed-initiative dialogue [CARB70] with its user. Our desire was not to produce an intelligent computer-aided instructional (CAI) system, but rather to focus on the rather poorly understood generative component that any such system will require:

CADHELP produces explanations by expressing "concepts" (encoded in a conceptual dependency format [SCHA77]) selected from the knowledge structure describing the selected feature. The text generated is not canned, but varies in significant ways according to the context it is being produced in. (In the current implementation, however, the display oomponent of the explanation is prestored.) Suppose the user asks to have explained the graphical feature of "dragging" objects (in this domain, logic devices) into position in the design area using the light pen. In a "summary" explanation, the normal operation of the feature is described in the highest-level terms available:

> To move an object to a location, use the drag feature.
>
> Select the drag feature.
> Select the object.
> Drag the object into the design area.
> Position the object.
> Exit from the drag feature.

In the "normal" mode of explanation, the operation is expanded in a greater level of detail:

To move an object to a location, use the drag feature.

(Light pen and parts flash as they are named)
The light pen hangs on the hook on the graphics device.
The hook is to the right of the screen.
The light pen has a sensor.
The light pen has a handle.
The light pen has a connection to the system.

Select the drag feature by pointing the light pen.
Grasp the light pen.
Move the light pen to the screen.
Touch the light sensor to the drag symbol.
The symbol flashes.
Select the object by pointing.
Drag the object into the design area.
Move the light pen into the design area.
The object follows.
Remove the light pen when the object is at the location.
Exit from the drag feature by pointing.

An explanation of "errors" describes how common mistakes Involving the feature can be made, and how to recover:

When you move the object into the design area, the system receives an input from the light pen when the light pen touches the object.
The system predicts where you are moving the light pen to.
The system redraws the object at the place.
The system waits to receive an input from the light pen.
If you move the light pen too fast, the system does not get an input.
The system cannot redraw the object.
To continue dragging, point to the object,.

III. Representing Expert Knowledge

In designing a mechanism which simulates the explanations generated by an expert, we have assumed that what is to be communicated is a knowledge structure (KS), a large body of interconnected declarations (propositions or assertions) describing a knowledge domain. Explanation, then, can be conveniently modelled as a process by which a more or less complete copy of a declarative KS is moved from the memory of the explainer to that of the listener.

Several claims oan be made about the characteristics of a KS suitable for CADHELP-style explanations. First, it should be language free, since we wish to be able to generate a number of paraphrases of the same underlying event. We would also like to be able to drive both the text generator and the display device from the same representation.

The units of the descriptions of graphical features are called conceptualizations. The conceptualizations are connected together at the lowest level of the KS into causal chains. The type of causal connection between the events is expressed in terms of the primitive causals of the commonsense algorithmic representation [RIEG77] for meohaniams. KS's for explanation, as bundles of causal chains Including well-understood actors and objects, conform well to the notion of a situational script [SCHA77]. The KS describing the operation of a graphical feature is called a feature script, and the objects appearing in them which differ from use to use of the script are called script variables. Feature scripts are built according to the "standard" methods descibed, for example, in [CULL78], and so will not be pursued further here. More details of CADHELP script structure can be found in [CULLBO].

IV. Explanation Strategies

The key observation concerning explanations of feature scripts is that they contain dozens of conceptualizations and causal relations which could be selected for expression. Thus, the process of explaining appears to be one of deciding what not to say rather than what to say. For example, the feature script $LPPOINT, which describes "pointing to a graphical object with the light pen," contains fourteen conceptualizations, interconnected by ten causal relations. Each assertion and relation is potentially expressible, but an explainer seems to focus on only certain elements of the KS during the explanation.

Note that the speaker has the dual problem to the inferenoe problem the understander has. The understander is expected to fill in what the speaker left out, by various types of inferences [e.g., CHAR72, RIEG75, CULL79, WILE79]. The speaker seeks as economical an expression of the thought to be communicated as possible. He leaves "sketchy" the parts of the utterance the hearer should be able to infer. We call this process conceptual sketchification. In examining what sounds natural in CADHELP-style explanations, it's clear that speakers sketchify what they say at every level. Selection and suppression occur in the use of the top-level pointer to the KS, causal chains to express an operation, unit events from a causal chain, and, finally, conceptual cases in a conceptualization.

Examples of sketch ification are easy to find in the example explanations of Section II. Consider the use of the term "point," for example. In the "normal" mode of explanation, CADHELP initially assumes that the user knows so little about pointing that the instrument used, the light pan, must be located and described. Then the mainpath sequence of actions comprising a pointing episode is expanded. In the second use of "point," the instrument is named, but the action sequence is not given again. In the final use of "point," the description is reduced to the minimum. The instrument and the actor, which can be inferred, are not named. But the object pointed to, which is variable from use to use of $LPPOINT, must be named.

The role of sketchifying can be seen within

single concepts, aa well. Consider the syntactic phenomenon oalled the "imperative". Conceptually, this Involves the suppression in the surface form of the conceptual actor. But from the point of view of explanation, it's known to both parties who the actor la: the user, Beoause of thia, the explanation need not explicitly name the actor, since this can be inferred. Similarly, oertaln uses of the infinitive construction are sketchy expressions of an inferrable fact about the underlying concept. If I say "I decided to go home," my hearer is expected to infer that the unnamed person who will go is the same as the one who decided. It appears that many syntactic constructions are the surface manifestations of underlying redundancies of these sorts,

CADHELP'a explanation strategies are organized into a hierarchy containing three levels of sketchification. The KS-summarization level uses intent/summary conceptualizations associated with feature scripts for descriptions of KS's which have already been explained. The explainer keeps a small working memory of the featurea which have been the subject of an "explain" command, or which were Imbedded in a feature which was explained.

The user Phenoaenolofgoai level explains a KS in terms of the user's actions and the system responses he can see directly. This type of explanation is Implemented by a small set of demon-like rules (on the order of a dozen) which examine the causal chains associated with the feature, paring these down to concepts in which the user is explicitly named as actor, or which are mental-information transfer events by the user in response to system actions. Thus, during the explanation of "dragging", various objects are made to flash at various points as the feature is operated. The system also redraws the selected object to give the illusion of the object's moving. The system aa actor is suppressed by a sketchif 1-catlon rule, however, which eventually forces the language generator to use actorleas construotions such as: "the symbol flashes" or "the object follows," Phenonenological level summarization is used for descriptions of "normal" operation, including the generation of prompts during the actual execution of a graphical feature.

The third, most detailed level expands each malnpath and support conceptualization in a causal chain. This- is called the user/system conversational level because it explaina the KS in terms of the give and take between the user and system, which is a kind of converaation. Becauae the details of how the system reacts to the user's manipulations are important in giving an understanding of how the user has gone wrong, conversational level summarization is used in error/recovery explanations. As in the phenomenological-level summarization, the technique of causal-chain sketchification is carried out by a set of rules, operating as demons, which select concepts to throw away. (The rules are, in fact, a subset of the ones used in phenomenological summariztion.)

V, Generating English

A, Background

By comparison with natural language understanding and inference, the research subarea of natural language generation has received relatively little attention. Although several language generators exist (e.g., [SIMM72], [CHES76], [MCD080]) which are capable of impressive fluency, these begin with a syntactic representation of the string to be created, including the words to be used. Thus, they are unsuited for use in a system such as CADHELP, which must generate from a conceptual representation of a feature to be explained. Goldman's BABEL [GOLD75] is one of a very limited number of examples of a sentence generator whioh starts with a conceptual representation of the thought to be uttered (in conceptual dependency format) and maps it into a surface English string, Goldman's approach, however, is basically suited for sentence at a time generation. The present researoh was motivated by the desire to build a generator capable of producing paragraph-length texts describing a knowledge structure, in which the factors that make for fluency could be explicitly studied.

B. Generation in CADHELP

CADHELP's generator, CGEN, has data and control structures which are very similar to the CA language analyzer described in [BIRN80], Its primary data structure is a short term memory, called the C-LIST, which the generator accesses in an iterative process of looking UP words to express the meaning of a concept currently at the focus of attention (the "front" of the C-LIST); and of inserting left-over subconcepts, perhaps with associated function words, in appropriate places on the C-LIST. Initially, the C-LIST contains a conceptualization which has survived the pruning process the KS-level sketchifiers apply to a feature script.

English has various conventions which govern the order in whioh words should be said. These conventions are stored in CGEN's dictionary as positional constraints on where the constituents expressing sub-conceptualizations must appear with respect to a word which spans part of the current concept. Consider the following simplified dictionary definition for the word "move," as in "move the stylus to the tablet:"

```
move:  (PTRANS ACTOR (NIL) OBJECT (NIL) TO (NIL))

       ACTOR — (PRECEDES PARENT)

       OBJECT — (FOLLOWS PARENT)
                (PRECEDES TO-SLOT-FILLER)

       TO – (FOLLOWS PARENT)
            (FOLLOWS OBJECT-SLOT-FILLER)
            (FOLLOWS FUNCTION/WORD:TO)
```

This definition states that "move" spans a concept based on a physical transfer of location (a PTRANS) which an ACTOR makes an OBJECT undergo.

The predicates PRECEDES and FOLLOWS are used to indicate where on the C-LIST the associated conceptual case fillers are to be inserted. Here, the predicates speoify a default ordering of ACTOR, the word "move", OBJECT, then the filler of the TO slot, following the function word "to". This approach to handling the details of English syntax is admittedly oversimplified. It has the great benefits of simplicity and uniformity, however, and thus has allowed a progressive approach to the addition of new language structure knowledge. For example, handling the active and passive forms of a verb turned out to require straightforward additions to the generator once a notion of "conceptual focus" was worked out.

CGEN's basic generation cycle be described by four rules:

1) If the front of the CLIST is empty, then there is nothing to generate; return.

2) If there is a word on the front of the CLIST then "say" the word by saving it on a special list to be returned when the generation cycle is complete.

3) If there is a concept on the front of the CLIST then remove the concept and try to find a word in the dictionary to express that concept. The look-up process is similar to the discrimination-net approach used in BABEL.

4) If the current concept is completely spanned by the word(s) found, replace it with those words. Otherwise, insert the leftover fillers into the C-LIST using the positional constraints stored with the word found.

There is a fifth rule which concerns the decision not to say something. This decision is embodied in the actions of the collection of concept-level sketchifiers which continuously monitor the front of the C-LIST for concepts which can be expressed more economically than Rules 1-4 above would prescribe. Thus, the basic model of generation contained in CADHELP is that of an "exhaustive" algorithm (Rules 1-4) being <u>restrained</u> by sketchifying rules.

VI. An Extended Example

Here we illustrate the generation process with computer output, edited for readability, showing the generator CGEN expressing the same concept at several levels of sketchiness. The concept is one that is selected repeatedly by the explanation mechanism (in a process not shown here) as it applies user-phenomenological summarization to produoe prompts for the user as he selects a graphical feature.

First, we show what CGEN produces if most of the concept-level sketchifiers are turned off, and the system is run in a special "verbose" mode. Comments are indicated by """:

```
ULISP V1.4    Copyright,1978,R.L.Kirby
Eval: (genverbose 'selomdpO)

"The current top of c-list is the input concept
    which states that the user is transferring
    to the system the information that he has the
    goal that the system instantiate one of the
    graphical features ($cadfeat), and the inst-
    rument of the transfer is the user applying
    a force to a command block with the stylus.

CGEN: top of c-list is cO:
(mtrans actor (#person role (*user)) mode (t)
        from (*cp*  part *user)
        to (*cp* part (#person role (*sys)))
        mobj (s-goal actor *user mode (t)
                     goal ($cadfeat actor *sys
                         featname (nil) mode (t)))
        inst (propel actor *user
                     obj (#inst role (*stylus))
                     to (*perpto* part
                           (#loc role (&cmdblk)
                              locname (nil)))
                     mode (t) manner (forceful)))

"CGEN finds "tell" in its dictionary as spanning
~part of the input concept
CGEN: using
(tell)

~Following the instructions found under the word,
"CGEN rebuilds the c-list:
CGEN: current c-list
(cO actor)    "the actor of the mtrans should be
              "said first (the user)
(tell)        "then the lexical item "tell"
(cO to part) "then the concept in the (to part)
              "of the mtrans (the system)
(that)        "then "that"
(C18 mobj)     "then the concept in the mobj slot
(by the action that)     "then "by the action that"
(cO inst)     "then the instrumental concept

CGEN: top of c-list is c50:
(#person role  (*user))  "the actor of the mtrans

CGEN: using
(you)

CGEN: current c-list  "now the c-list has two
(you)     "lexical items on top which can be
(tell)    "popped off



CGEN: top of c-list is c55: "mtrans to part
(#person role (*sys))       "concept is next

CGEN: using                "realized by "cadhelp"
(cadhelp)

CGEN: current c-list       "contains two more
(cadhelp)                  "words to be said
(that)


"Now the goal stative in the mtrans mobj slot
"reaches the top
CGEN: top of c-list is c29:
(s-goal actor (#peraon role (*user)) mode (t)
        goal ($cadfeat actor (lperson role (*sys))
                featname (nil) mode (t)))
```

```
CGEN: using
(want)              "expressed with "want"

CGEN: current c~list
(c29 actor)         ~whose actor goes first
(want)              ~then want itself
(that)              ~then "that"
(c29 goal)          ~then the goal concept
(by the action that) ~then the mtrans inst
(cO inst)

CGEN: top of c-list is e72
(#person role (*user))  ~the  user reappears

CGEN: using
(you)

"Now the "execute a cad feature" script concept
"reaches the top
CGEN: top of c-list is c42:
($eadfeat focus (actor) actor (lperson role (fsys))
       featname (nil) mode (t))

CGEN: using  "CGEN has not been able to find a
(execute)    "word which expresses the concept
            "directly so it uses the generic
             "verb "execute"
CGEN: top of c-list is c48: "system appears
(#person role (*sys))       "again

CGEN: using
(cadhelp)

"Now the nominalized form of the "feature script"
"concept bubbles up from where "execute" put it...
CGEN: top of c-list is c102:
($cadfeat nomform (nil) focus (actor)
          actor (#person role (*sys))
          featname (nil) mode (non))

CGEN: using  "dictionary has a word for this form
(feature)

CGEN: top of c-list is c107: "indefinite refer-
(indef)                      "ence

CGEN: using
(a)

"Finally the propel concept comes up for
"expression
 CGEN: top of c-list is c148:
 (propel actor (lperson role (•user))
         obj (#inst role (•stylus))
         to (•perpto* part (#loc role
               (&cndblk) locname (nil)))
         mode (t) manner (forceful))

 CGEN: using  "the aot itself is "press"
 (press)
"after several more cycles, the result...
Value:
(you tell cadhelp that you want that cadhelp
 execute a feature by the action that you press
 the stylus on a command block)
```

Next, we hand the above oonoept to the standard generator three tines in Succession. Each tine the realization returned is shorter.

```
Eva1: (gen 'SelemdpO)
"The inst sketchifier looks at the instrumental
    concept and notes that its actor is the same
    as the input concept's. Therefore, a gerund
    form can be used to express the instrument
    economically...
inst: examining inst concept c12

"The imp sketchifier notes that user-phenom
    summarization is going on (as is normal
    in prompts), and that the actor is the
    user; therefore the imperative is ok...
imp: squashing actor *user in cO

"The dictionary returns "select" as the mapping
    word. "Select" takes care of more of the
    input concept than "tell" does,
    so this is chosen
CGEN: using
(select)

"In the c-list as rebuilt by "select", the actor
    has disappeared because of imp, "select" only
    requires expression of the (mobj goal)
    subconcept, and inst has set up a special form
    for the expression of the instrumental conoept
CGEN: current c-list
(select)          "the lexical form "select"
(cO mobj goal)    "the $cadfeature concept
                  "which "select" has nominalized
(by)              -"by"
(cO inst)         "a nominalized form of the
                  "propel concept

CGEN: top of c-list is c44:
($cadfeat actor (lperson role (*sys))
          featname (nil) mode (nom))
CGEN: using
(feature)    "concept is expressed as "a feature"
             "as before
CGEN: using
(a)

"The instrumental concept reaches the top.
    Note the syninstr marker which the inst
    sketchifier added to get the gerund form
CGEN: top of c-list is c72:
(propel syninstr ($prog) actor (nil)
        obj (#inst role (•stylus))
        to (*perpto* part (#loc role (&cmdblk)
                  locname (nil)))
         mode (t) manner (forceful))

 CGEN: using  "the progressive realization
 (pressing)

 "the cycle continues, with the result...
 Value:
 (select a feature by pressing the stylus
  on a command block)

 "Now express the conoept again
 Eval: (gen 'selcmdpO)
 "This time the standard inst sketchifier
     disqualifies Itself because it's already had
     a shot at this oonoept. Thus entity-instrument
     gets a chance to modify the concept. Its test
     is: is an instrumental object being used in a
     "normal" function in an instrumental concept?
 entinst: examining inst oonoept in oO
```

"Imp does its usual dirty work:
imp: squashing actor •user in cO

~with the result:
Value:
(select a feature with the stylus)

""Once more into the breach...
Eva1: (gen 'selecmdpO)
"Both the standard inst and entity-inst
sketchlfiers disqualify themselves because
they've already had a shot at this concept.
Thus kill-instrument gets in and erases the
instrumental concept altogether...
killinst: squashing inst concept in cO

imp: squashing actor *user in cO

"with the result:
Value:
(select a feature)

VII. Implementation Note

The natural-language and control modules of CADHELP are programmed in Franz LISP. The high-level graphics component was designed using the facilities of the University of Toronto GPAC [REEV77]. These parts of CADHELP currently run under UNIX on a VAX-11/780 computer. The graphics display itself is generated by a DEC VT-11 vector display device controlled by a PDP-11/0M computer. The graphics, natural language and explanation modules are coordinated by a general-purpose integration package called a hierarchical task manager [CULL81].

To give a feeling for run-time, CGEN produces the verbose expression of the command-select concept discussed in Section VI in about 5 seconds when it is used alone; and in about 15 seconds when it is part of the complete CADHELP system.

VIII. Conclusions

This paper describes research into the mechanisms that appear to be needed for generating explanations intelligently. The micro-world chosen, that of interactive graphics in support of CAD, is a particularly attractive one because the needed knowledge structures can be designed by extending existing representational formalisms: scripts and mechanism-simulation causal relations. Within this domain, methods of summarization have been developed which oan be coupled to the level of detail requested by the user, and the demands of local context. Summarization is based upon the notion of "sketchification," a multilevel process of selecting conceptual items for expression, then suppressing components of an item based upon whether the user should be able to infer them. A collection of concept-level sketchlfiers interacts with a natural language generator, CGEN, of a novel design. These sketchlfiers prescribe syntactic constructions which are more economical than those whioh would normally be used by the generator.

REFERENCES

[BIRN79] Bimbaum, L. and Self ridge, M. Problems in Conceptual Analysis of Natural Language. CSTR-168, Yale U., New Haven, CT.

[CARB70] Carbonell, J. R. An Artificial Intelligence Approach to CAI. IEEE Trans, Man-Machine Systems. MMS-11.

[CHAR72] Charniak, E. Towards a Model of Children's Story Comprehension. AITR-266. M.I.T. Cambridge, MA.

[CHAR77] Charniak, E. Ms Malaprop, A Language Comprehension Program, Proc. Fifth Int- Joint Conf.• on *Alt* Cambridge, MA.

[CHES76] Chester, D. Translating Mathematical Proofs into English. Artificial Intrilifflnoe. November.

[CULL78] Cullingford, R. Script Application: Computer Understanding of Newspaper Stories. CSTR-116. Yale U., New Haven, CT.

[CULL79] Cullingford, R. Pattern-Matching and Inference in Story Understanding. Discourse Processes. Vol. 2, No. 4.

[CULL80] Cullingford, R., Krueger, M. and Selfridge, M. Automated Explanations as a Component of a Computer-Aided Design System. Proc.. Int. Conf. on Systems. Man & Cybernetics, Cambridge, MA. October.

[CULL81] Cullingford, R. Integrating Knowledge Sources for Computer 'Understanding' Tasks. IEEE Trans.. SM&C. February.

[GOLD75] Goldman, N. Conceptual Generation. In Schank, R. (ed.), Conceptual Information Processing. North Holland.

[MCD080] McDonald, D. Language Production as a Process of Decision-Making under Constraints, AI Lab. TR, M.I.T. Cambridge, MA.

[REEV77] Reeves, W. GPAC User's Manual. Computer Systems Research Group. Univ. of Toronto, Toronto, Canada.

[REIG75] Rieger, C. Conceptual Memory. In R. Schank (ed.), Conceptual Information Processing. North Holland.

[RIEG77] Rieger, C and Grinberg, M. The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms. Proc Fifth Int. Joint Conf, on i l. Cambridge, Mass.

[SCHA77] Schank, R. and Abelson, R. *SBEJS&SL* Plana Goals and nnd^rstanding, Erlbaum. Hillsdale, NJ.

[SIMM72] Simmons, R. and Slocum, J. Generating English Discourse from Semantic Networks. £ojUL* AQ4. Vol. 15, No. 10.

[WILE79] Wilensky, R. Understanding Goal-Based Stories. CSTR-UO. Yale Univ., New Haven, CT.