CMU-CS-81-146

Automatic Construction of Explanation Networks for a Cooperative User Interface

Ingrid D. Glasner and Philip J. Hayes

November, 1981

Abstract

This paper is concerned with providing *automatically generated* on-line explanations to the user of a functional computer subsystem or *tool* about what the tool can and cannot do, what parameters and options are available or required with a given command, etc.. The explanations are given through the COUSIN interface system which provides a cooperative tool-independent user interface for tools whose objects, operations, input syntax, display formats, etc. are declaratively represented in a *tool description* data base. The explanations are produced automatically from this data base, with no incremental effort on the part of the tool designer, and in a single uniform style for any tool that uses COUSIN as its interface. The explanation facility takes the form of a fine-grained, tightly linked network of text frames supported by the ZOG menu-selection system. Exactly what information the net building program, NB, extracts from a tool description, and the way in which this information is formatted in the text frames is controlled by a second declarative data base called the *aspect description*. The declarative nature of the aspect description makes it easy to adapt NB to changes in and extensions to the tool description formalism, and to experiment with the structure of the explanation network. We also describe how the appropriate network frame can be found and displayed in response to specific explanation requests from the user.

To appear in:
Proc. of Jt. Conf. on Easier and More Productive Use of Computing Systems, AnnArbor, May, 1981, which will be published as a special issue of ACM SIGSOC Bulletin.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1. Introduction

Interfaces to interactive computer systems often appear inflexible and uncooperative to their users. The COUSIN[1] project at Carnegie-Mellon is engaged in a wide-ranging program of research to produce more graceful and cooperative user interfaces. Our work includes research on more flexible and robust parsing techniques [2, 6], communication mechanisms that are both natural and efficient [4], and a number of other topics (see [1, 5, 7]) including the subject of this paper: explanation facilities.

In order to appear cooperative, any user interface must be prepared to offer its user explanations of what it can and cannot do, which command the user should issue to get a particular task done, what parameters and options are available with a given command, etc.. It is these kinds of *static* explanations that we will be concerned with here. Other, *dynamic* types of explanation, such as what command an interface is currently performing, what kind of information it expects the user to input next, what was the result of a command issued two hours ago, are equally vital for a cooperative interface, but will not be covered in this paper.

The most common approach to static explanation facilities in the relatively small proportion of interactive systems that provide them at all has been to use canned text messages. Such messages are either written into the system by the system designer specifically for interactive use as in the SOS editor [11], or extracted by an indexing scheme from an on-line version of the system manual as in the RdMail electronic mail system [9], or the CMULisp system [3]. However, the structure of explanations provided this way is often too grainy, so that the user must search through irrelevant material to get to the information he actually needs, and insufficiently interlinked, so that the user may be unable to locate the information he needs even if he has found a related piece of information, see [5] for a good example of this. Much more ***fine-grained and closely interlinked explanations are necessary.*** Happily, the overall approach to interface construction adopted in the COUSIN system forms an excellent basis for the provision of such explanations. Moreover, the approach allows such explanations ***to be constructed automatically from information that the interface needs in any case for other purposes***, thus providing an interactive system with an explanation facility for zero incremental effort on the part of the interface designer.

Automatic construction of explanation facilities with COUSIN is possible because COUSIN is independent of the particular functional subsystem or tool being interfaced to. In order to use the COUSIN interface, all relevant information about a given subsystem must be represented in a

[1] COUSIN is a new acronym; previous publications describe the system as a gracefully interacting interface.

declarative data base called a *tool description*. This information includes all the commands and object types that the tool deals with, their parameters and components with filler types and defaults, their input syntax and display formats, plus other information, less important for our purposes, about the tool's operation. Using this information, COUSIN can accept commands from the user, check them for validity, fill in defaults, correct some errors and ambiguities, interact with the user to correct the others, and finally transmit the corrected command to the underlying system. More importantly for our purposes, this information, possibly supplemented by text strings to explain the purpose of commands and object types, is just what is needed to construct a fine-grained and highly interconnected static explanation facility for any tool represented by such a tool description.

In what follows, we describe a program, called NB, for automatically constructing a static explanation facility from a tool description. The resulting explanation facility is a network of text frames that the user is able to traverse as he wishes using the ZOG menu-selection system [10]. Which information is extracted from the tool description, and in what format it is presented in the text frames is controlled by a second declarative data base we call the *aspect description*. The representation of this meta-level information in declarative format makes the NB program very easy to adapt to changes in or extensions to the format of the tool description formalism. Because the COUSIN project is highly experimental, such changes occur frequently. The following sections describe in turn the tool description formalism, the organization of the explanation network derived from it, the way in which this network is constructed, and the way in which the appropriate network node may be found automatically in response to a specific request for help by the user. We conclude with a discussion of some more problematic requests for explanation.

2. The Tool Description

As mentioned above, a particular functional sub-system, or *tool*, is characterized for the COUSIN interface program by information about the types of objects it manipulates and the operations or commands it can perform. This information is stored in the *tool description*, a declarative data base provided by the tool system designer. Given the tool description, the COUSIN interface is able to parse the user's commands, check them for validity, fill in defaults, correct some errors and ambiguities, interact with the user to correct the others, and finally transmit the corrected command to the underlying tool system. The tool description consists of schemas, one for each object type dealt with by the tool, and one for each operation that the tool can perform.

Object schemas are "declarations" of the types of data object that the tool knows how to deal with. The following is a partial schema for the object type, Message, of an electronic mail system that we have been using as an example tool for COUSIN.

```
[ StructureType: Object
  ObjectName: &Message
  Components:
      [ Sender: [FillerType: &Person  ComposeAs: CurrentUser]
        Recipient: [FillerType: &Person  Number: OneOrMore]
        Copies: [FillerType: &Person  Number: NoneOrMore]
        Date: [FillerType: &Date  ComposeAs: CurrentDate]
        Subject: [FillerType: &Uninterpreted  Number: NoneOrOne]
        Body: [FillerType: &Uninterpreted  Number:NoneOrOne]
        After: [FillerType: &Date  UseFor: DescriptionOnly]
        Before: [FillerType: &Date  UseFor: DescriptionOnly]
      ]
  Syntax:
      [ SynType: NounPhrase
        Head: (message note <?piece ?of mail>)
        PostMod: (
                  <%From ↑Sender>
                  <%To ↑Recipient>
                  <%CopiesTo ↑Copies>
                  <%Dated ↑Date>
                  <%About ↑Subject>
                  <%After ↑After>
                  <%Before ↑Before>
                  <%Between ↑After and ↑Before>
                  )
      ]
]
```

The precise details of this slot and filler style of notation are not important for present purposes. The points to note are that the Components property list gives each of the components of the structured object, &Message, and for each of these components, the type of object supposed to fill that component, plus any defaults, etc. that are relevant. The Syntax property list says that a message can be described by a noun phrase in which the head noun is chosen from the specified list of words and patterns of words (denoted by angle brackets), and which can be followed by any of the descriptive cases given - percent means word class (defined elsewhere), and up-arrow refers back to one of the components of Message. Other information contained in an object schema, but not shown here, includes display formats for the object, and information on how to resolve a description into an instance. The formalism can also describe primitive object types, i.e. those that have no components, and classes of object types.

A tool description schema for an operation includes a specification of its parameters and the syntax of commands requesting execution of the operation. The following example is the schema for the Forward operation from the same example electronic mail tool system.

```
[ StructureType: Operation
  OperationName: &Forward
  Parameters:
        [ Message: [FillerType: &Message]
          Recipient: [FillerType: &Person  Number: OneOrMore]
          Forwarder: [FillerType: &Person  MustBe: CurrentUser]
        ]
  Syntax:
        [ SynType: Imperative
          Verb: (forward send mail (pass on))
          Object: <↑Message>
          Cases: <%To ↑Recipient>
        ]
]
```

The interpretation of the notation is similar to the example above. Using this schema, COUSIN can determine whether a command specifies all required parameters of the operation, default any missing optional slots, and check on the appropriateness of those parameters that are given. Operation schemas also contain descriptions of output formats, how to transmit an operation request to the tool, and what to do after the operation is finished.

For further details of the tool description and of the way it is used by the COUSIN interface, the reader is referred to [1].

The classes of information in the two examples above are all that are needed to produce a basic static explanation network from a tool description. However, to give more complete explanations about the tool system objects and operations, tool description schemas could also contain additional textual information put in for this specific purpose. This information might include text explaining the purposes behind certain objects or operations, examples of object descriptions and instances, etc.. Since the tool description schemas are represented as lists and property lists in our implementation, augmentation of the schemas with explanatory fields is straightforward. We have not used supplementary textual fields in our current implementation, but Section 4.1 discusses how such additional information could be incorporated into the explanations we currently produce.

There are many similarities between the structure of objects and that of operations, and in particular a correspondence between the components of an object and the parameters of an operation. This similarity is exploited by our program, NB, for constructing an explanation facility from a tool description, so in the discussion that follows to avoid saying "object or operation" and "component or parameter" too much, we will refer to objects and operations collectively as *tool items* or just as items, and we will refer to the components or parameters of a tool item as its *slots*. Note that while a slot name is unique within a particular item, the same name may be used for slots in more than one item (e.g. the Forward, Answer, and Delete operations in our example tool all have Message parameters). This fact cannot be ignored in constructing an explanation facility, because of possible questions like "Is there a default for the Message parameter?". For this reason, when we talk of a slot, we will often be referring to the collection of all slots with the given name.

3. The Structure of an Explanation Network

As mentioned in the introduction, the program NB produces an explanation facility for an interactive functional subsystem or tool from a declarative description of that tool in the formalism we have just described. This explanation facility is in the form of a net of text frames that the user can traverse as he wishes by use of the ZOG [10] menu-selection system. The advantages of having the explanation facility produced automatically are clear. Since the tool designer must provide the tool description in any case in order make use of the COUSIN interface system, his tool will be augmented with an explanation facility with no extra effort on his part. The choice of a network of text frames for the explanation facility requires some justification.

One obvious alternative to constructing a net of text frames is to construct responses to help requests "on the fly". The same techniques could be used to extract the needed information from the data base, and responses could be tailored to the specific question asked, instead of being selected from a predetermined set of text frames. While this approach may be unavoidable sometimes (see Section 5.3), it has serious efficiency problems. Extracting the required information from the tool description proves to be quite an expensive operation, and a practical interface cannot afford to make its user wait too long for responses to his questions. In addition, such an approach does not help the user to obtain information related to the explanation elicited by his last question. If each question is answered separately, there are no easy ways to "poke around" a topic. This violates the goal of interlinked explanations established in the introduction.

Since the explanation facility is intended to be used interactively, the structure of the net of text frames can and should be much richer than the structure of a printed manual containing the same information. Printed documents, being composed of chapters and paragraphs, have a hierarchical structure, appropriate for describing domains in which there is a hierarchy of concepts. Cross-references, pointing to concepts on different "paths" or in higher levels of the hierarchy, are legal, but access to information via cross-references is done in a way distinctly different from access via hierarchical links, which is the "preferred" way.

The inherent structure of a tool system usually does not conform to a strictly hierarchical discipline. If we chose "is-component-of" as one of the basic hierarchical relations between tool items, each "is-fillertype-of" connexion would be a cross-reference. The explanation facility is, therefore, structured as a true network, providing a variety of semantically meaningful connexions, of equal priority, between the information units it contains.

In the remainder of this section we present the structure of explanation networks from an abstract point of view; discussion of the program that constructs them from a tool description is deferred until Section 4.

3.1. Nodes in the Net

The units of information in the explanation facility, i.e. the nodes in the explanation network, do not contain as much information as the schemas in the tool description. In line with the goal set in the introduction, we have chosen a finer granularity in order to be able to answer a user's questions more succinctly, and to avoid presenting him with a confusingly large amount of information all at once. There are two basic kinds of information units in the net, containing aspect information and context information, respectively.

There is an information unit for each relevant *aspect* of each tool item. For our current system, aspects of a (compound) object type include the structure of instances of this type, the syntax of corresponding descriptions, the possible uses of such objects as components or parameters, etc.. Aspects of an operation include its parameter list, the syntax of requests to perform it, the effect of its execution, etc. An aspect is thus not the same as a tool description field, but rather is determined pragmatically as a group of facts about a tool item that fit naturally together from a user's point of view. As described in more detail in Section 4.3, the NB program uses a declarative data base called an *aspect description* to determine for which aspects of a tool item it should construct information units. This level of indirection adds significantly to the flexibility of the NB program.

For each slot mentioned in the tool description, there is an information unit for each of its *contexts*, i.e., for each of its meanings as a component of an object or as a parameter of an operation. These information units describe the properties of the slot in the specific context: its fillertype, its default value, whether it is mandatory or optional (for parameters), etc.. Examples of information units are given in Section 4.

3.2. Edges in the Net

Each tool item and slot name may thus give rise to one or more information units. These information units may be linked together by various types of binary relations which constitute edges in the explanation network. Edges may run between information units associated with different tool items or slots, or between units associated with the same item or slot.

Direct connexions between information units for different tool entities exist in correspondence to certain semantic relations between the tool items they describe. These relations are called *simple semantic relations*. The standard simple semantic relations, existing in any kind of tool, are the following:

- "is-component-of" (between a compound object type and each of its components)
- "is-parameter-of" (between an operation and each of its parameters)
- "is-fillertype-of" (between a slot and its fillertype)

For a specific tool system, there may be additional simple semantic relations. This depends on whether there are additional kinds of tool items e.g., classes of object types, and semantic relations involving them, e.g., "is-member-of"; it also depends upon which of the semantic relations existing in the tool domain are considered important enough to represent by edges in the net (instead of by sequences of edges, or not at all). Because which edges are produced is specified declaratively in the aspect description (see Section 4.3), it is easy to experiment with different combinations of edge types, and to provide extra edge types for tool descriptions with more information without changing the basic NB program.

For each item and each slot, one of the corresponding information units, called the entity's *base unit*, is distinguished as being the unit that provides the primary information about it. There are edges from the base unit to each of the remaining information units for the same entity. Conversely, each information unit contains a reference to its base unit. This implies that all information units describing the same entity are accessible (possibly indirectly) from each other, and each of them can be accessed via the entity's base unit.

4. Automatic Construction of an Explanation Network

In this section, we turn to the question of how the explanation network for a given tool is actually constructed from its tool description. To simplify our task, and to avoid needless duplication of effort, we chose to build networks in a way that allowed use of the well-developed software support of the

ZOG system.

4.1. Implementation of an Explanation Network as a ZOG Net

ZOG [10] is a rapid-response, large-network, menu-selection system for man-machine communication also developed at Carnegie-Mellon. In ZOG-nets, information is chunked into *frames*, i.e., portions of text small enough to be displayed as a whole on a video terminal. In addition to the actual information, each frame contains a menu of options from which the user can select the next topic and thus the next frame to be shown. Frames are identified internally by unique frame identifiers, but the user refers to frames by selecting their content. Thus, ZOG is well suited to handling networks like ours in which connexions between frames are semantic.

The ZOG system accepts files containing descriptions of frames using the BH formalism [8], and uses these descriptions to produce the actual frames with appropriate indexing and interconnexions. For each frame, the external BH format indicates the layout and contents of the information part of the frame, plus the frames to which it should be connected and short text strings to describe the connexions. The internal representation of frames and frame connexions, the mechanism for actually displaying the frames on the screen, and the implementation of other operations on the net are hidden from the producer of the external net description.

In the implementation of an explanation network as a ZOG net, each information unit is represented by a frame (or several frames, depending on its size). Here are some examples of the frames constructed by NB for our example tool system from its tool description. First, a frame for the structure aspect of the object type, Message:

```
MESSAGE (GENERAL STRUCTURE)

Objects of type Message have components:

      1.  Sender
      2.  Recipient
      3.  Copies
      4.  Date
      5.  Subject
      6.  Body

D. how is a Message described?
U. how is a Message used?
```

This frame, together with a line of general options not shown, takes up a complete display screen[2].

[2] In the complete COUSIN system, the frames are actually shown in one window or pane of a larger screen.

When the user selects another frame by typing 1, 2, 3, 4, 5, 6, D, or U, this frame is replaced by the one corresponding to the digit or letter typed, e.g. typing '1' results in the display of a frame describing the Recipient component of a Message:

```
RECIPIENT (in Message)

The Recipient component of object Message
contains one or more objects of type Person.


     F. about object type Person
     O. about object type Message
     M. about other meanings of Recipient
```

Note that this frame is not the same as the frame for Recipient as a parameter of the operation, Forward:

```
RECIPIENT (in Forward)

The parameter Recipient of operation Forward
contains one or more objects of type Person.

     F. about object type Person
     O. about operation Forward
     M. about other meanings of Recipient
```

Typing 'M' to either of these Recipient frames would, however, display the base frame for the Recipient slot, which lists both of these frames, together with similar frames for all other valid contexts of the Recipient slot.

All of the text in all of these frames is generated automatically from prestored word patterns filled out by slot and item names extracted from the tool description. The screen layouts are also prestored, and are subject to the conventions of the underlying ZOG system. In addition to the types of frame shown, the frames generated by the present system include frames giving the abstract syntax for each tool item, and frames showing in which other objects (operations) an object appears as a component (parameter). Another interesting possibility, not yet implemented would be frames giving example inputs for each object and operation; we believe these examples could be derived automatically from the abstract syntax and vocabulary specifications.

The example tool description we have been using contains only information that is also needed by other parts of the COUSIN interface. A more complete explanation network would result from the addition of information describing e.g. the purpose of various tool items. The design of NB makes such an extension easy. It would simply be necessary to define a new aspect for the tool description, say Purpose, and modify the aspect description (see Section 4.3) to accomodate this extra aspect. In this way, it would be straightforward to translate an extra property in the tool description schema for Message:

```
Purpose: "enable the transfer of some text (the Body) from a Sender
          to one or more Recipients, with extra copies to the users
          specified in Copies."
```

into corresponding extra lines in the Message frame above (or in a supplementary frame), prefaced by "The purpose of a Message is to ...".

Supplementing a tool description formalism with these kinds of slots for information in the form of prose has an additional benefit. It offers a very structured way for the tool designer to document parts of his system, and the ways he intends them to be used. This approach to documentation also requires less effort from the tool designer because he can let the NB program take care of integrating his individual comments into, what is essentially, a carefully structured, on-line manual.

4.2. NB - the Net Building Program

Our program to construct explanation networks from a tool description is called NB for Net Builder. NB operates off-line from the rest of the COUSIN system. It takes a tool description as input, and from it produces a ZOG net which can be used as an interactive explanation facility for the tool in question. NB itself does not interact with the end user of the COUSIN interface. The choice of the ZOG system as the support for the explanation network means that all that NB actually has to do is to produce the BH file corresponding to the network it wants to construct. All the actual displays, and all the necessary bookeeping operations are handled by the ZOG system. NB's task then reduces to:

- extracting the information for each frame from the tool description and translating it into the form in which it is to be displayed, and
- specifying the connexions between frames (see Section 3.2 for a list of connexion types).

NB is composed of a hierarchy of functional modules which produce different levels of detail of the network. There are three main levels in this hierarchy:

1. functions for building the net
2. functions for building a frame, including:
   - *aspect modules* (one for each kind of aspect)
   - *context modules* (one for object contexts and one for operation contexts)
3. functions for building frame components

The functions of level 1 take care of the bookkeeping necessary for correctly connecting the frames built. In addition to the actual frames, NB produces four lists of indexing information:

- two *Base Lists* associating each tool item and slot name with the frame identifier of its base unit, and
- two *Frames Lists* associating each item/aspect pair and each slot/context pair with the frame identifier of the corresponding aspect unit or context unit.

Besides being necessary for the action of NB, these lists are also important in finding the correct text frame with which to answer specific help requests from the user (see Section 5.1).

The level 2 functions are executed for each aspect of each tool item and for each context of each slot name. They pick the appropriate pieces of information from the tool description and call functions of level 3 to produce the appropriate BH output. This BH representation for the explanation net is then compiled into a ZOG net, ready for use as the explanation facility for the interactive tool from whose description it was derived.

4.3. The Aspect Description

As mentioned earlier, NB does not produce frames for each possible aspect of each tool item in the tool description. It chooses only those aspects for which there is an entry in a second declarative data base called the *aspect description*. This arrangement insulates NB from changes in or additions to the tool description formalism; only the aspect description and not the code for NB need be modified in such circumstances. An aspect description, then, does not provide information about a specific tool like a tool description, but rather gives "meta-information", about the tool description NB is given to process and about the form of the explanation network to be constructed from it. In particular, it answers the following questions:

- What kinds of tool items (e.g. objects, operations, primitive objects) are there?
- About which aspects of each tool item should the system give explanations? I.e., for which aspects should NB construct corresponding information units in the explanation network?
- How is the information to be extracted from the tool description?
- Which fields of which tool description schemas should be used?
- What level 2 function of NB will produce the frame for a given aspect?

An aspect description is composed of schemas, one for each type of tool item. An aspect schema for a specific item type contains sub-schemas for each information unit to be derived from that kind of tool item and incorporated into the net produced by NB. In our example system, the aspect schema for the Object item type contains sub-schemas for Structure, Description, and Uses. Each of these sub-schemas indicates from which fields in the description of its own and other object types its information unit is derived. It also gives the name of the functional module from level 2 of NB that does the actual building, plus some other information used to construct the resulting frame. Here is an abbreviated version of the aspect schema for items of type Object.

```
[ Entity:  Object
  Structure: [ Module:  BuildStruct
               OwnFields:  (Components)
               Crossrefs: (Description Uses)
               SelChar: "S"
             ]
  Description: [ Module:  BuildDescr
                 OwnFields:  (Syntax DescrExamples)
                 Crossrefs: (Structure)
                 SelChar: "D"
               ]
  Uses: [ Module:  BuildUse
          OtherFields: [Object: (Components)
                        Operation: (Parameters)
                       ]
          Crossrefs: (Structure Description)
          SelChar: "T"
]
```

It indicates information units are to be constructed for the Structure, Description, and Uses aspects of all tool description items of type Object, using the NB level 2 functions, BuildStruct, BuildDescr, and BuildUse, respectively. The fields of Object items from which these functions are to obtain their information are given by OwnFields, or in the case of Uses by OtherFields, which says in this case that all Operations and other Objects must be searched for uses of the object. CrossRefs and SelChar are used in the obvious way for setting up the links between the three information units constructed for each item of type Object. Naturally, the information in the aspect description depends heavily on the format of the tool description. For instance, if there was a UsedAs field in the tool description of each item of type Object, e.g. in the case of Message something like

```
UsedAs:  [ParameterOf:  (Forward Reply)]
```

then the corresponding portion of the aspect description for Object could be

```
Uses: [ Module:  NewBuildUse
        OwnFields:  (UsedAs)
        Crossrefs: (Structure Description)
        SelChar: "T"
      ]
```

with function NewBuildUse being much simpler than BuildUse. So there is a certain tradeoff between the size of the tool description and the amount of information to be provided by the tool designer on the one hand, and the complexity of the aspect modules and of the net building process on the other.

While the aspect description itself is declarative, there is non-declarative information associated with it in the form of the NB function names in the Module fields. The aspect description would insulate NB from changes in and extensions to the tool description formalism even more if that information was also made declarative, i.e. if the information to be extracted from the specified fields of the tool description was itself specified declaratively, along with the way it was to be incorporated into the resulting net frames. Of course, NB would still need to provide functions to interpret such declarative specifications, but its actual code could be completely independent of any particular tool description format. We intend to pursue this line of research in the future.

5. Using an Explanation Network

Now that we have discussed how an explanation network is constructed from a tool description, we turn to an examination of how the COUSIN interface can use the network to respond to a user's interactive requests for explanations. Presently, COUSIN uses the network in a way that requires the user to expend an unnecessarily large amount of effort to obtain the explanation he desires. However, we also describe a way for COUSIN to use the net more intelligently, so that a user will have to expend less effort to find out what he wants to know.

The net access mechanism currently used by COUSIN requires the explanation net to have a distinguished *root frame* and a set of *index frames*. The root frame points off to a general help frame which explains how to move around in the ZOG network, and to two sequences of index frames, one for tool items, and the other for tool slot names. The index frames in turn point off to the base frames for each of the tool items and slot names. The root and index frames are, of course, produced automatically by NB. When the user asks for help, COUSIN switches to ZOG mode and displays the root frame, from where the user himself, by making appropriate selections, has to find his way to the frame containing the desired information.

This way of handling help requests requires only minimal involvement on the part of the parsing component of COUSIN: the parser only has to identify help requests as such, without analyzing them further. Also, the explanation component of COUSIN does not need to know anything about the net - ZOG can be made to display the root frame automatically each time the net is entered. But obviously, this mechanism is far from being cooperative or graceful, since all the work has to be done by the user, through stepwise selection of the right frame.

A more adequate reaction by the explanation facility of COUSIN would be to display directly the aspect or context frame that provides the information asked for. We have designed a mechanism to do this. It depends on the Base and Frames lists produced by NB in the course of constructing the explanation net (see Section 4.2). The Base lists associate each tool item and slot name with its base unit frame, and the Frames lists associate each item/aspect pair and each slot/context pair with the corresponding aspect unit frame or context unit frame. The way in which these lists can be used to provide access to the appropriate frame depends on the type of explanation requested. We distinguish three types of request: *simple* and *indirect* which can be answered by a single frame of the network, and *complex* which cannot.

5.1. Simple Requests

A *simple request* is a question for which the answering frame can be found by direct lookup in the Base lists or Frames lists. This means that the subject inquired about must be one of:

- an aspect of an item
- a context of a slot
- basic information about an entity.

The following are examples of simple requests (with the answer frame indicated in brackets):

What does a message look like?
[Structure aspect of object type Message]

Where do messages occur?
[Uses aspect of object type Message]

What parameters are needed forward a message?
[Parameters aspect of operation Forward]

(in the dialogue context of being prompted for the parameters of operation Forward)
What does 'recipient' mean?
[Recipient as parameter of operation Forward]

What is a message?
[basic information about object type Message]

While it would be possible for a tool designer to anticipate all these types of requests and include rules to parse them in the grammar for recognizing commands and object descriptions (see Section 2 for examples of how this grammar is specified in the tool description), it appears unnecessary to impose this extra burden on him. The grammar description must, in any case, be preprocessed into a form more suitable for the flexible pattern-matching parser [6] used by COUSIN, and it appears feasible to modify the preprocessor to supplement the grammar rules it produces with rules to recognize explanation requests. This would not only free the tool designer from having to be concerned with all the different forms that help requests could take, but would also mean that the forms of input recognized as help requests would be uniform across all tools using the COUSIN interface.

Since we have already provided through the aspect description a way of insulating the net building program, NB, from changes in or extension to the tool description formalism, it would be only logical to provide the same degree of insulation to the grammar preprocessor. This would involve providing grammar patterns for each information unit that the aspect description instructs NB to produce. The grammar patterns for a given information unit would recognize requests for explanation which are best answered by display of that unit. This arrangement would also ensure that the simple explanation requests that could be handled would correspond exactly to the information available in the explanation net.

5.2. Indirect Requests

The concept of simple request does not cover all kinds of help requests that a user might come up with. Consider a question like:

How do I specify the recipient of the forwarding operation?

It can be answered by displaying the Description aspect frame for Person (since Person is the filler type for the Recipient parameter of operation Forward). But in the question itself, the object type Person is not mentioned at all; it is identified by making use of the simple semantic relation "is-fillertype-of" (cf. Section 3.1). If all the explanation facility could do was to look in the Frames or Base lists for items or aspects mentioned directly in requests for explanations, the user would have to decompose his question into a sequence of two simple requests:

What is the filler type of the Recipient parameter of Forward?
[Answered by displaying the frame for Recipient as a parameter for Forward]

How can I specify a Person?

This does not create the impression of a cooperative interface.

This problem can be solved by allowing the user to specify tool items by chains of slot specifications of length two, as in the example above, or greater, as in:

How do I specify the host of the recipient of the message in resend?

Questions with such chains of slot specifications are called indirect requests for explanation, and should be included in the coverage of the automatically produced grammar discussed in Section 5.1. In general, we expect to be able to accomodate within this grammar all questions, but the most obscure, that can be answered by displaying a single frame of the explanation network.

5.3. Complex Requests

While simple and indirect requests for explanation can be answered by displaying a single frame of the explanation network, this is not true in general:

Which components of a message are of type mailbox?

There is no single frame in the explanation network we have been considering that shows all components of Message together with detailed information about their filler types. This is a reflection of our principle of making information units very small, in order to avoid bothering the user with more information than he actually asked for. But whatever organization were chosen for the net, there would inevitably be questions whose answers were not contained in a single frame.

If the answer to the above question is to be a single frame, the explanation facility would have to create this frame dynamically. However, it is unclear how this could be done in a style consistent with the other frames in the net, and in any case, constructing the frame may result in an unacceptably long delay in answering the question. An alternative is to produce the answer in the form of a single text string, However, this will result in two radically different kinds of responses to requests for explanation, and such non-uniformity, especially if it cannot be predicted by the user, is probably undesirable. A third alternative is for the explanation facility to determine which frames in the explanation net would contribute to the answer of a complex question, inform the user that more than one frame is involved, and provide a simple mechanism to allow him to examine each of the frames at his leisure. The production of answers to complex questions is a research topic we intend to pursue.

6. Summary

This paper has been concerned with providing an explanation facility for an interactive subsystem or *tool* to answer such "static" questions as what the tool can and cannot do, what parameters and options are available or required with a given command, etc.. We addressed the problem in the context of the COUSIN interface system which provides a cooperative tool-independent user interface for tools whose objects, operations, input syntax, display formats, etc. are declaratively represented in a *tool description* data base. Our approach was to construct the explanation facility automatically from this data base, thus allowing the facility to be produced with no incremental effort on the part of the tool designer, and in a single uniform style for any tool using COUSIN as its interface. The resulting explanation facility took the form of a network of text frames supported by the ZOG menu-selection system. This format allowed us to meet our goals of fine-grained and closely interlinked explanations. The network of frames was produced from a tool description by the net building program, NB. Exactly what information NB extracted from the tool description, and exactly how this information was formatted in the text frames was controlled by a second declarative data base called the *aspect* *description*. The declarative nature of the aspect description made it easy to adapt NB to changes in and extensions to the tool description formalism, and to experiment with the structure of the explanation network. We also showed how the appropriate network frame could be accessed in response to specific explanation requests from the user.

Acknowledgments

Don McCracken and George Robertson showed us how to interface to the ZOG system, and gave us useful insights into exploiting its considerable power to best advantage.

References

1. Ball, J. E. and Hayes, P. J. Representation of Task-Independent Knowledge in a Gracefully Interacting User Interface. Proc. 1st Annual Meeting of the American Association for Artificial Intelligence, Stanford University, August, 1980, pp. 116-120.

2. Carbonell, J. G. and Hayes, P. J. Dynamic Strategy Selection in Flexible Parsing. Proc. of 19th Annual Meeting of the Assoc. for Comput. Ling., Stanford University, June, 1981, pp. 143-147.

3. *TOPS LISP*. Carnegie-Mellon University Computer Science Department, 1978.

4. Hayes, P. J. Anaphora in Limited Domain Systems. Proc. Seventh Int. Jt. Conf. on Artificial Intelligence, Vancouver, 1981, pp. 416-422.

5. Hayes, P. J., Ball, J. E., and Reddy, R. "Breaking the Man-Machine Communication Barrier." *Computer 14*, 3 (March 1981).

6. Hayes, P. J. and Mouradian, G. V. Flexible Parsing. Proc. of 18th Annual Meeting of the Assoc. for Comput. Ling., Philadelphia, June, 1980, pp. 97-103.

7. Hayes, P. J., and Reddy, R. An Anatomy of Graceful Interaction in Man-Machine Communication. Tech. report, Computer Science Department, Carnegie-Mellon University, 1979.

8. Newcomer, J. BH - A General Information Organization Program. Carnegie-Mellon University Computer Science Department, 1976.

9. *RdMail Message Management System*. Carnegie-Mellon University Computer Science Department, 1980.

10. Robertson, G., Newell, A., and Ramakrishna, K. ZOG: A Man-Machine Communication Philosophy. Tech. Rept. , Carnegie-Mellon University Computer Science Department, August, 1977.

11. *Son of StopGap (SOS)*. Carnegie-Mellon University Computer Science Department, 1978. Originally developed at Stanford AI Lab; the help facility was added at CMU.