Natural language meaning representations of queries about operating systems

Paul Mc Kevitt

Computing Research Laboratory∗
Dept. 3CRL, Box 30001
New Mexico State University
Las Cruces, NM 88003-0001, USA.
CSNET: paul@nmsu.edu [(505) 646-5942/5466]

ABSTRACT

An essential component of any natural language consultant program is a mechanism that constructs good meaning representations of user queries. When people ask queries about computer operating systems they often refer to interrelated actions and objects. These temporal interrelations should be recognized and captured within semantic structures. Actions or commands, and relations between them, can be described by embedded action representations. Embedded action representations are constructed from deep case structures. A theory of embedding enables efficient parsing of queries about operating systems. The theory is applied in an operating system consultant called *OSCON*.

∗The Computing Research Laboratory is partially funded by the New Mexico State Legislature administered by its Science and Technology Commission as part of the Rio Grande Research Corridor.

0. Introduction

When people ask queries about computer operating systems they often refer to interrelated actions and objects. It is possible to build a computer program which will answer natural language queries about computer operating systems. An essential component of any such program is a mechanism which constructs good meaning representations of user queries. Such representations will enable the correct understanding, by the program, of what the user is asking about.

A good way to understand the requirements of a natural language understander for operating systems is to build a theory about the way people use English to ask questions on these systems. For example people ask queries like, ''How do I print a file on the Xerox with pageheaders?'', or ''How do I spell a file and then have the mistakes printed on the Imagen?'' The former query has one action, that of *printing*, and the latter has two actions, that of *finding-mistakes* and *printing-on-Imagens*. Of course, some queries have no actions at all. These are queries like ''What is a file?'' or ''What is a pipe?'' Such queries are static and involve answers which are descriptive rather than dynamic. Therefore, operating system queries are about the dynamics of the system, i.e., queries about actions such as printing, removing or deleting; or about the statics of the system, i.e., queries about static objects such as files, file-structure, pipes, and so on. Hegner (1987) has termed the former *dynamic queries* and the latter *concept description queries*. It is necessary to develop some mechanism whereby meaning representations for multiple action queries can be integrated in a sensible manner. That is what this paper is about.

1. The Operating System CONsultant

At the Computing Research Laboratory we are developing a natural language understander for an operating system consultant called OSCON. OSCON is being programmed in Kyoto Common Lisp. The system is intended to help novice and expert users learn operating system concepts. It is hoped that OSCON will answer user queries on many operating systems, although we are focusing on UNIX∗. Other computer operating systems of interest are TOPS-20†, VMS† and VM/CMS‡.

OSCON has a two-module architecture involving a natural language understander and a knowledge base. The natural language understander has the function of understanding and answering English queries. Query responses will be in English too. The knowledge base is detailed and formal and functions as a solving or answering module. The knowledge base is being constructed at the University of Vermont by Dr. Steve Hegner. Work on the knowledge base is discussed extensively in Douglass & Hegner (1982), Hegner & Douglass (1984) and Hegner (1987). The two-module architecture is one of the principle design features of OSCON. As pointed out by Hegner (see Hegner 1987, p. 1) the two-module architecture facilitates an important principle on the separation of understanding and solving.

∗ UNIX is a trademark of AT&T Bell Laboratories.

† TOPS-20 and VMS are trademarks of Digital Equipment Corporation.

‡ VM/CMS is a trademark of International Business Machines Corporation.

2. A theory of embedding

We can assume that any query about operating systems includes a number of actions (which may be zero) and objects manipulated by those actions. Any meaning representation of a query must contain, in some semantic form, the actions and objects and how these are related together.

Dynamic queries containing more than one action are more difficult to deal with. There needs to be some way of relating actions together. Let's look at some more examples. In ''How do I send a troff file to the Imagen?'' there are two actions *troffing-a-file*, and then *sending-it-to-the-Imagen*. In the query, ''How do I remove a file printing on the Imagen?'' there are also two actions. Those are *printing-a-file* and then *removing-it*. What do we notice about the actions in each query? Each action is related temporally to other actions. One action is executed before another and the ordering is important.

To understand queries about operating systems we need to be able to recognize actions and objects in the input. To do that we need to represent actions and objects in the system. We need to represent the meaning of words if the system is to determine whether words in input sentences are actions or objects.

3. The components of a meaning representation

To determine whether words in the input are objects or actions we need to represent the meaning of words in the system. Therefore we need a *dictionary* of words which tells us the type of each word. So, *print* will be represented as an action and so will *delete* and *move*. *Files* and *directories* will be typed as objects. We build a dictionary with entries like *(file location object), (print action), (directory location object)* and *(user actor)*. Notice that *file* and *directory* are marked as being specific types of objects — locations.

The next problem we must worry about is how to define allowable operations of actions. We should build some structures which represent various actions and how they relate to objects. Such structures would recognize incorrect operations of actions. A database of patterns is defined to represent legal action operations. Patterns are semantic templates for expected legal actions. We construct a pattern database with entries like *(observe-obj <actor> <object>)*. This entry tells us that a legal sentence could include actors observing objects. Thus the query, ''How do I see a file?'' would match this pattern. We can call the patterns *action templates*.

To build meaning representations for sentences, we need to be able to link action templates together by some means. To do that there needs to be some precise definition of what each action template means. Each action template should have an associated meaning structure. Such meaning structures should include actions, objects related to actions, types of those objects, and actors. We call such meaning structures *deep case structures* and one is shown in Figure 1.

Figure 1 is a case structure which tells us that to observe an object there can be an observer, an object of observation, and a location for the observation to occur. Each entity may have some *description* tagged to it. The case structure also references two frames. These are the action frames in the system that represent domain specific knowledge about operating systems. The structure and content of action frames is discussed in detail in Mc

```
(observe-obj ((actor _*)
              (description _))
             ((object _)
              (description _))
             ((location _)
              (description _))
             (frames print list))
```

Figure 1. Case structure for observing objects.

Kevitt & Wilks (1987). For this example the frames are print and list which specify printing-objects and listing-objects respectively. Already we note that domain specific information will be selected if this case structure is referenced. However, the frame references in case structures for action frames are not relevant here, although they are useful in other aspects of the OSCON system.

4. Embedded action representations

As already mentioned, user queries about operating system commands contain embeddings of actions. It should be possible to create representations of nested or embedded deep case structures to describe interrelated actions. We can call such representations embedded action representations (EREPs). Operating system commands are related to each other in specific ways. When users ask questions about such actions they usually get these relations correct. Therefore, if we build EREPs from user queries, the EREPs should be a good approximation of the relations. This means that domain-specific structures produced from EREPs should often be correct. In fact, if the domain-specific structures are not correct, the input query also contains relations which are incorrect.

Examples of typical actions which can occur in EREPs are printing, listing and deleting. People can ask questions about UNIX such as, ‘‘How do I print a listing of my directory?’’, or ‘‘I need to print a file.’’ In the former example we build an EREP where the concept *listing* is embedded inside the concept *print* and in the latter case *print* is embedded inside *need*. These are examples of double embedding. Yet, triple embeddings result from queries such as ‘‘How do I delete a listing of my directory, printing on the Imagen.’’ We will now go on to show how EREPs can be used to understand natural language queries.

* We use the symbol ‘‘_’’ to denote an unfilled slot in a case frame.

5. Null embedded queries ($\{A_i\}_{i=0,1}$)

If a query involves no actions, or just one action, then there will be no embedding at all. If we use A_i to represent the number of actions in a query then null embedded queries are denoted by $\{A_i\}_{i=0,1}$. Concept description queries are simple questions about objects with no presence of operating system actions. Therefore, concept description queries will always exhibit null embedding. A typical example of a concept description query is the sentence, ''What is read protection?'' The action template *(be <object>)* is used in deciphering this query. *Protection* is defined under the category object from its dictionary entry i.e., *(protection object)*. The case structure for the *be* action template is instantiated to give the structure in Figure 2.

(be ((object PROTECTION∗)
(description READ))
(frames protection))

Figure 2. Instantiated case structure for 'be'.

Dynamic English queries illustrate null embedding when only one action is mentioned. For example, the query ''How do I delete a file?'' has a representation with no embedded actions at all. The query is parsed into the structure shown in Figure 3.

6. Positively embedded queries ($\{A_i\}_{i\geq 2}$)

There are many types of embedding present in meaning representations resulting from dynamic queries. We have already seen that dynamic queries exhibit null embedding. However they also exhibit positive embedding, which means that the query includes more than one action. We call queries with two or more actions positively embedded queries and they are denoted by $\{A_i\}_{i\geq 2}$. Such queries have at least one positive embedding with one action inside another. Also, we have discovered that there are many types of positive embedding and there are different ways of recognizing and processing these.

6.1. Explicit embedding

Explicit embedding occurs in representations for queries involving two or more actions. For example, the meaning representation for the query ''How do I print a listing of

∗ In the following case structure diagrams capitalized items indicate values filled in from dictionary entries.

(delete-obj (actor USER)
((object FILE)
(description quantity ONE))
(frames remove))

Figure 3. Instantiated case structure for ‘delete-obj’.

my directory?’’, has the concept of listing embedded inside the concept of printing. In processing this query, an *observe-pat* case structure is instantiated to give Figure 4.

(observe-pat (actor USER)
(case (observe-obj (actor _)
((object DIRECTORY)
(description quantity ONE)
(description owner USER))
(frames print list)))
(frames print list))

Figure 4. Meaning representation exhibiting explicit embedding.

Figure 4 shows that deep case representation for listing is nested inside the representation for printing. The actor slot in the outer case representation is instantiated to be USER. The inner case structure contains directory as an object because the actor is asking about listing directories. In Figure 4 we note that the actor slot in the inner case structure is not instantiated. However, this information would be determined from the outer case structure and promoted inwards.

6.2. Implicit embedding

Some word in a user query may indicate implicitly the existence of another action although this action is not mentioned directly. Say, for example, the system is given the query, ‘‘How do I delete mail files?’’ Naively, the system would believe that the user just wants to delete an object called *file* with description *mail*. The system would overlook the fact that another action (in this case *mail*) has created the object. In deriving a meaning representation for this example a first step would be to construct the structure in Figure 5.

(delete-obj (actor USER)
((object FILE)
(description quantity MORE-THAN-ONE)
(description type MAIL))
(frames remove))

Figure 5. Implicit embedding I.

Now, to solve the problem of not recognizing implicit embedding, each object could be checked every time a representation is produced to establish whether that object or action refers to another action template. In this case, *mail* is recognized as possibly referencing another action. Indeed, *mail* (a description on the object *file*) references the action template *send* and its respective case structure. After some processing the EREP in Figure 6 is computed. It is noted that in Figure 6 the concept *send* is embedded inside the concept *remove*. The actor as user is expressed inside each case structure. The send case structure represents the fact that the user wishes to remove more than one file by the quantity descriptor.

6.3. Shadowed embedding

Often actions such as *wanting* or *needing* can shadow the UNIX action which is more important for the system to locate. Although we are primarily concerned with locating UNIX concepts, we do realize the importance of shadowing actions. Such actions are very useful in detecting the goals of the user (see Wilensky et al. 1984, p. 589). The direction of reading the input query is important because shadowing may occur while reading a sentence in one direction although it does not in the other. Examples of shadowing exist in sentences like ‘‘I would like to delete a file’’, and ‘‘I need to print a file’’. On reading the latter query from left to right *need* shadows *print*. However, if the query is read right to left we get ‘‘A file, to print, I need?’’ In this case *print* is not shadowed. Yet, OSCON reads

---

(remove (actor USER)
(case (send (actor USER)
((object FILE)
(description quantity MORE-THAN-ONE))
(frames MAIL)))
(frames remove))

Figure 6. Implicit embedding II.

sentences left to right and therefore it needs to know about shadowing. There are action templates for shadowing verbs in the pattern database such as, (s-verb <actor> <pattern>). For the query, ‘‘I need to print a file?’’ the meaning representation shown in Figure 7 is formed.

(s-verb (need) (actor USER)
(case (observe-obj (actor USER)
((object FILE)
(description quantity ONE))
(frames print list))))

Figure 7. Shadowed embedding.

We note that the s-verb case structure has only one case slot other than *actor*, called *case*. The actual shadowing verb used in the sentence is tagged onto the EREP as it may be useful in later processing. As already mentioned, such information would be useful for discovering the intention of the user.

6.4. The intricacy of redundant embedding

Representations with redundant embedding are more a characteristic of the parsing strategy than a characteristic of English. For example, while parsing the query ‘‘How do I use print to print a file?’’, the case structure for observing objects would become embedded within itself. This happens because of implicit embedding rules. In effect, (1) the user has mentioned printing files, and (2) the user has also mentioned the operation for doing so i.e., *print*. It would certainly be a mistake to embed in examples such as this and OSCON must have strategies to recognize redundant embedding. For this example the system produces the case structure in Figure 8.

(s-verb (use) (actor USER)
(object PRINT)
(case (observe-pat observe-obj) (actor _) ...)
(frames FRAME (object)))

Figure 8. Redundant embedding I.

From the previous example of implicit embedding we notice that the system would find PRINT and believe there should be another embedding of the *observe-obj* case structure. Yet, this is wrong because the case structure for observing objects already exists. There must be another rule which recognizes that implicit embedding is not carried out if there appears to be redundancy. Therefore, a counter rule will dictate that PRINT does not call up another case structure. Yet, we must be careful in applying the counter rule too. For example, ‘‘How do I print listed files?’’ involves an embedding of *observe-obj* inside *observe-obj*. The inner case structure for listing is referenced again by implicit embedding techniques and the problem here is that we really do wish to embed. There seems no way out of all this. But, look again at the example of redundant embedding. We notice that the query contains the shadowing verb *use* and that is what the system needs to look for while applying the counter rule. The system will correctly represent the query ‘‘How do I use print to print a file?’’ as Figure 9. It is noted in Figure 9 that objects have been promoted inwards from the query. The clause ‘‘...to print a file’’ instantiates objects in the inner case structure. No frames are called forward by FRAME (object) because of the counteractive rule for redundancy. Note however, that in a query like ‘‘How do I use print?’’ FRAME (object) would call forward these frames as they are not referenced in any inner embedded case structure.

```
(s-verb (use) (actor USER)
        (object PRINT)
        (case (observe-obj (actor USER)
                           ((object FILE)
                            (description quantity ONE))
                           (frames print list))))
        (frames NIL))
```

Figure 9. Redundant embedding II.

7. Other work on meaning representations

There has been much work on building meaning representations of natural language utterances and we can not claim to do justice to all of those here. We shall begin with representations of natural language utterances on operating systems and then move on to more general approaches to meaning representation.

The theory of how to represent natural language queries in the Unix Consultant (see Wilensky et al., 1986) has evolved over a number of years. Initially, the system used a phrasal analyzer called PHRAN (see Arens, 1986; Wilensky et al., 1984) which read sentences in English and produced representations to decode their meanings. PHRAN contained a knowledge base of pattern-concept pairs where patterns were descriptions of literal utterances that had many different levels of abstraction. For example, <*person*> <*give*> <*person*> <*object*> is a phrasal pattern. Each pattern had an associated conceptual template which is a piece of meaning representation. For example, associated with the phrasal pattern <*nationality*> *restaurant* is a conceptual template denoting a restaurant that serves <nationality> type food. PHRAN's use of patterns and concepts is similar to our use of action templates and case structures. However PHRAN is a general parser and not specifically geared towards operating systems. There was no theory of embedding to contend with our own. Therefore, although PHRAN was a good general mechanism for producing meaning representations of English it was not very efficient as a parser of queries about operating systems.

The latest Unix Consultant implementation (see Wilensky et al., 1986) involves a new parser called ALANA (Augmentable LANguage Analyzer) developed by Cox (1986). ALANA is an extension of the PHRAN parser described above. Although ALANA is a more advanced parser than PHRAN there is no description of how the parser may handle multiple action queries. Any discussion of ALANA shows only how single action queries

are handled. Again, there is no description of an alternative theory that competes with ours of embedding.

Douglass & Hegner (1982) used case frames in the front end for the Unix Computer Consultant (UCC) system. Case frames were templates representing the main action of a clause and the constituents of the action, such as the actor and recipient of the action. The case frames corresponded to logical operations in an operating system, and therefore formed the main link between English-language operating system concepts and the formal semantic definitions of specific UNIX commands. The problem with these case frames was that they were too far removed from natural language input to be useful and also there was no great theory of how to combine case frames together to formulate good meaning representations of complex queries.

The SINIX consultant involves a natural language interface which produces meaning representations of English sentences. Although the SINIX parser (see Kemke 1986, Section 2.6.3) uses case structures to build up sentence case frames we find little description of a theory on how case structures may be combined efficiently.

Fillmore (1968, 1977) discusses how natural language sentences can be understood using knowledge in a form of case structures. Case structures are frames into which verbs may be parsed. Fillmore concerns himself more with the syntax of verbs than their semantics. Different verbs may link to a number of different frames and he explains which verbs are constrained to which cases in which frames. Although Fillmore gives a good description of different verbs and their properties he does not concern himself with the semantic questions of verbs like *print* affecting objects like *files* or directories. He does not describe any theory of embedding where different structures for various verbs can be linked together. He is largely concerned with single action sentences. Fillmore helps us in defining properties of verbs but not how such verbs are integrated in an operating system consultant.

Schank (1975) has worked on a deep representation of natural language sentences called *conceptual dependency*. Schank intends a very deep representation because he wishes to have a language free form. His representation is similar to our deep case structures. Schank's theory entails a reduction of all utterances to combinations of primitive *predicates* chosen from a set of twelve *actions* plus state and change of state, together with the primitive *causation*, and seven role relations or *conceptual cases*. Schank sets up case frames for primitive acts as opposed to Fillmore's concentration on the surface verbs of English.

Wilks (1975a, 1975b, 1976, 1978a, 1978b) developed a natural language understanding program which parsed English text into deep meaning representations. Wilks' parser constructed a meaning representation made up of *templates*, having the basic form of *agent-action-object* which are integrated by the use of *paraplates* and *inference rules*. The *templates* are built up from *formulas* which represent individual word senses. In the discussion of meaning representations above there is no discussion of semantic formulas because information about such word senses would already be maintained in the parser that analyzes English input. Our deep case structures are like Wilks' templates as they contain actions, objects and agents. Wilks' idea of building paraplates from templates parallels ours of building EREPs from case structures. However Wilks would have different templates for different clauses whereas we only have different templates for different verbs.

Also Wilks talks of linking paraplates with cases, whereas we talk of linking case structures by embedding them inside each other to denote temporal relations. In other words, we are talking at a more pragmatic level than a semantic one. Of course the semantic structures do exist in the parser that analyzes input. Another difference between our EREPs and Wilks' paraplates is that the action representations are constructed on the fly whereas Wilks' paraplates already exist in the system.

8. Embedded representations are useful

Embedded action representations are a precise means of formalizing meaning relations between UNIX actions. English queries involving interrelated actions can be understood effectively using these action representations. The most significant feature of EREPs is that because they maintain an implicit notion of time, or ordering of actions, there is no need to represent temporal orderings themselves. These are already inherently provided by the representation itself.

There is much work yet to be done on our meaning representations. For example we believe that negation can be handled as a type of embedding. The query ''I can not delete my file?'' can be interpreted as an embedding of *deleting* inside *not* with the s-verb *can*. We have not defined the rules for matching case structures to output from a parser, or promoting objects form one embedded action to another. There has been no discussion of the mechanisms involved in rejecting incorrect action relations occurring in user queries. This would happen if a user query did not match one of the action templates. For example a PRINT action could never be nested inside a DELETE action when they apply to the same file because if a file is deleted it is not possible to print the file. A lot of work needs to be done on determining what to do when such errors are detected. Early detection of such pragmatic user errors will increase the efficiency of the operating system consultant.

References

Arens, Yigal (1986) *CLUSTER: An approach to contextual language understanding.* Report No. UCB/CSD 86/293, Computer Science Division (EECS), University of California, Berkeley, California 94720, April.

Cox, Charles A. (1986) *ALANA Augmentable LANguage Analyzer*. Report No. UCB/CSD 86/283, Computer Science Division (EECS), University of California, Berkeley, California 94720, January.

Douglass, Robert J. & Stephen J. Hegner (1982) *An expert consultant for the UNIX operating system: Bridging the gap between the user and command language semantics*. In Proc. Fourth National Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI)/SCIEO Conference, Saskatoon, Saskatchewan, pp. 119-127, May.

Fillmore, C.J. (1968) *The case for case*. In Universals in Linguistic Theory, E. Bach and R. Harms (Eds.), pp. 1-90. New York: Holt, Rinehart and Winston.

Fillmore, C.J. (1977) *The case for case reopened*. In Syntax and Semantics, Peter Cole and Jerrold M. Sadock (Eds.), pp. 59-81. New York: Academic Press.

Hegner, Stephen J. & Robert J. Douglass (1984) *Knowledge base design for an operating system expert consultant*. In Proc. of the Fifth National Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI), pp. 159-161, London, Ontario, December.

Hegner, Stephen J. (1987) *Representations of command language behavior for an operating system expert consultation facility*. Technical Report CS/TR87-02, CS/EE Department, University of Vermont, Burlington, Vermont, USA.

Kemke, Christel (1986) *The SINIX Consultant — Requirements, Design, and Implementation of an intelligent Help System for a UNIX Derivative*. Universitat des Saarlandes, KI-Labor (SC-Project), Bericht Nr. 11, October.

Mc Kevitt, Paul & Yorick Wilks (1987) *Transfer Semantics in an Operating System Consultant: The formalization of actions involving object transfer,* In Proceedings of The Tenth International Joint Conference on Artificial Intelligence (IJCAI-87), Vol. 1, pp. 569-575, Milano, Italy, August.

Schank, R.C. (1975) *Conceptual information processing*. Amsterdam: North-Holland.

Wilensky, Robert, Yigal Arens & David Chin (1984) *Talking to UNIX in English: An overview of UC*. Communications of the ACM, Vol. 27, No. 6, June, pp. 574-593.

Wilensky, Robert, Jim Mayfield, Anthony Albert, David Chin, Charles Cox, Marc Luria, James Martin and Dekai Wu (1986) *UC — a progress report*. Report No. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkeley, California 94720, July.

Wilks, Yorick (1975a) *An intelligent analyser and understander of English*. Communications of the Association of Computing Machinery (ACM), Vol. 18, pp. 264-274.

Wilks, Yorick (1975b) *A preferential, pattern-seeking, semantics for natural language inference*. Artificial Intelligence, Vol. 6, No. 1, pp. 53-74.

Wilks, Yorick (1976) *Processing case*. Technical Report, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, United Kingdom. Also in American Journal of Computational Linguistics, microfiche 56.

Wilks, Yorick (1978a) *Good and bad arguments about semantic primitives*. Communication and Cognition, Vol 10., No. 3/4, pp. 181-221.

Wilks, Yorick (1978b) *Making preferences more active*. Artificial intelligence, Vol. 11, pp. 197-223.

Wilks, Yorick; Xiuming Huang and Dan Fass (1985) *Syntax, Semantics and Right Attachment*. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85), pp. 779-784, Los Angeles, California.

Wilks, Yorick (1986) *Projects at CRL in Natural Language Processing*. Memoranda in Computer and Cognitive Science, Memorandum MCCS-86-58, Rio Grande Research Corridor, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001.

Wilks, Yorick; Dan Fass, Cheng-Ming Guo, James E. McDonald Tony Plate & Brian M. Slator (1987) *A tractable machine dictionary as a resource for computational semantics*. Memoranda in Computer and Cognitive Science, Memorandum MCCS-87-105, Rio Grande Research Corridor, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001.