A general effect representation for operating system commands*

Paul Mc Kevitt and Zhaoxin Pan

Computing Research Laboratory
Dept. 3CRL, Box 30001
New Mexico State University
Las Cruces, NM 88003-0001, USA
E-mail: INTERNET: {paul,zpan}@nmsu.edu; Phone: 505-646-5109

0. Abstract

OSCON (Operating System CONsultant) is a computer program which answers, in English, English queries about computer operating systems. The program answers queries in less than 2.5 seconds. The program accepts input in the form of typed English queries and answers queries queries on over 40 commands. OSCON is intended to be a consultant for various types of users who may ask vague and detailed queries. OSCON is programmed in Quintus Prolog. Unique to this operating system consultant is the fact that it answers queries on more than one operating system. This is enabled by the general effect representation used for describing the effects of commands. The representation contains information about the effect name, objects of the effect, and the location where the effect will apply. The representation is used for describing commands from the UNIX† and MS-DOS† operating systems.

1. Introduction

This paper describes the representation of operating system commands used in the OSCON (Operating System CONsultant) program (OSCON[1.2]). OSCON is a natural language interface which answers English queries about computer Operating Systems (see Mc Kevitt 1986; Mc Kevitt 1988; Mc Kevitt & Wilks 1987; Guthrie, Mc Kevitt & Wilks 1989). OSCON allows the user to enter English queries and then answers them in English. The program is written in Quintus Prolog and the maximum time taken to answer a query is 2.5 seconds. OSCON runs on a Sun-3/ME-4 computer and answers on over 40 commands from the UNIX and MS-DOS Operating Systems. There are four basic types of queries that users tend to ask and the system handles all of these. OSCON will also answer queries about options on UNIX commands and complex queries about command compositions. The system is intended to be used by varying types of users with different levels of expertise. The architecture of OSCON is modular so that it can be easily updated and mapped over to new domains.

One of the characteristics of Operating Systems is that they all incorporate basic operations like displaying, removing and transferring data. There are a number of commands which are primitive to different Operating Systems. For example, the command ‘‘copy’’ in MS-DOS basically performs the same function as ‘‘cp’’ in UNIX. The function is to copy files from one location to another.

∗ **This research is currently funded by U S *WEST* Advanced Technologies, Denver, Colorado, under their Sponsored Research Program.**

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

† MS-DOS is a trademark of Microsoft Corporation.

The representation of information about commands in OSCON consists of (a) Effects or Postconditions, (b) Preconditions, (c) Command Syntax, and (d) Command Names. For different Operating Systems one would expect different Preconditions, Syntax and Command Names. However, the description of the Effect of a command should basically remain the same. This happens because of the similarity of function for command primitives from different systems.

2. Knowledge representation in OSCON

One of the problems in building natural language interfaces is to organise the knowledge of the domain in some form which will be effective. There are two types of knowledge stored in OSCON: (1) knowledge about Operating Systems, and (2) knowledge about language.

Detailed knowledge about Operating Systems is contained in a module of OSCON called DataCon. This type of knowledge includes, command effects, command preconditions, command syntax and the names of commands. Also, stored here is (1) knowledge about options for commands, (2) English text description of Operating System concepts like ‘‘files’’ and ‘‘directories’’, and (3) knowledge about plans or possible command combinations. Knowledge is stored here for the the UNIX and MS-DOS Operating Systems.

Also, OSCON has stored knowledge about language which includes words used to refer to command actions. For example, a user may use the words ‘‘remove’’, ‘‘delete’’ and ‘‘get rid of’’, and so on to ask a query about deleting files and directories. These words must be stored under the general category of remove. Also, there are many ways in which people ask queries about actions. A user asking about copying a file will probably specify the file which must be copied. A user asking about displaying will most likely specify the location of display. This type of knowledge is called ‘‘understanding knowledge’’ and is stored within a module of OSCON called KnowCon.

3. Knowledge for solving (DataCon)

The knowledge for solving in OSCON consists of files of data that describe detailed information about Operating Systems. There are four types of knowledge stored here: (1) Basic command representation, (2) Option representation, (3) Concept representation, and (4) Plan representation. We shall discuss the first type of representation here. The others are described in Guthrie, Mc Kevitt & Wilks (1989).

For both UNIX and MS-DOS commands there are basically four types of information stored. These are (1) Effects or postconditions, (2) Preconditions, (3) Syntax, and (4) Command Names. In this paper we are mainly

concerned with the Effects or postconditions of commands.

3.1. Effects

Effects, or postconditions, are definitions of the outcome of commands. The effect is defined by a predicate which has a name and three arguments. The predicate name is the action and the arguments are (1) object, (2) object modifier, and (3) location. It turns out that this representation is useful for both the UNIX and MS-DOS operating system. This happens because in all operating systems there is a basic set of actions and objects that they act upon. Shown below are Prolog predicates for the effects of some UNIX commands. Rule [1] shows the effect for the command ‘‘more’’. The object for ‘‘more’’ is ‘‘file’’ and and its modifier ‘‘contents’’. The location of output of more is the ‘‘screen’’. One case of the ‘‘cat’’ command [2] has the same effect as ‘‘more’’. The other effect case of ‘‘cat’’ [3] is defined as concatenate and describes the concatenation of files. The command ‘‘ls’’ will either display directory contents [4], or file information [5] on the screen. The displaying information command ‘‘users’’ will display usernames on the screen. Rule [7] describes the ‘‘gemacs’’ command which creates files, and rule [8] the ‘‘rm’’ command which deletes them.

[1] unixeffect(more, display(file, contents, screen)).
[2] unixeffect(cat, display(file, contents, screen)).
[3] unixeffect(cat, concat(file1, file2, file3)).
[4] unixeffect(ls, display(directory, contents, screen)).
[5] unixeffect(ls, display(file, info, screen)).
[6] unixeffect(users, display(usernames, @, screen)).
[7] unixeffect(gemacs, create(file, @, loc)).
[8] unixeffect(rm, remove(directory, @, loc)).

We show below the Effect similarities for some commands from UNIX and MS-DOS. Effect representations are predicates with two arguments. The predicate name depicts the relevant Operating System. The first predicate argument is the Command Name and the second argument the Effect of the command. Each Command Effect consists of a predicate with three arguments: (1) Object, (2) Object Modifier, and (3) Object Location. There are three sets of commands below. These represent examples of (1) displaying, (2) copying, and (3) removing commands respectively.

The first Command Effect specification below [1] describes the effect for displaying files. More specifically, the effect specification describes the displaying of files on the screen. Note that the effect specification is the same for both the UNIX and MS-DOS Operating Systems. The difference is in the command names. The second Command Effect specification describes the effect for copying files. Both files and directories can be copied and there are Effect specifications for each case here. In UNIX the

command is ‘‘cp’’ while in MS-DOS it is ‘‘copy’’. The third [3] Command Effect representation represents the removal of files in Operating Systems. The Effect representation for remove declares that a file can be removed from some location. The command in UNIX is ‘‘rm’’ while in MS-DOS it is ‘‘del’’.

[1] unixeffect(more,display(file,contents,screen)).
doseffect(type,display(file,contents,screen)).

[2] unixeffect(cp,copy(file,x,loc1)).
unixeffect(cp,copy(directory,x,loc1)).
doseffect(copy,copy(file,x,loc1)).
doseffect(copy,copy(directory,x,loc1)).

[3] unixeffect(rm,remove(file,x,loc0)).
doseffect(del,remove(file,x,loc0)).

Therefore, it is possible to define generic Effect definitions for Operating Systems and use these definitions for different command names in different Operating Systems. We hope to add in more information about other Operating Systems such as VAX VMS† to further demonstrate the generality of OSCON.

3.2. Preconditions

Preconditions are lists of objects which are necessary for a command to be executed. Here are some examples of preconditions for commands from UNIX. Rules [1] and [2] show that ‘‘more’’ and ‘‘cat’’ have the precondition ‘‘file’’. The command ‘‘mkdir’’ has the precondition ‘‘directory’’ and ‘‘cp’’ has no precondition.

[1] precon(more, [file]).
[2] precon(cat, [file]).
[3] precon(mkdir, [directory]).
[4] precon(cp, []).
Of course the preconditions for MS-DOS commands have different command names. Shown below are the equivalent MS-DOS commands.

[1] dosprecon(type, [file]).
[2] dosprecon(type, [file]).
[3] dosprecon(md, [directory]).
[4] dosprecon(copy, []).

† VAX VMS is a Trademark of the Digital Equipment Corporation.

3.3. Syntax

The syntax of commands is defined as a structure which contains the name of the command and then its syntactic definition of use. Shown below are some examples of the syntax for UNIX commands. The syntax rules are three place lists containing (1) Command name, (2) Optionname (filled in from context), and (3) Syntax description.

[1] unixsyn(more, Optionname, "[more <filename>]").
[2] unixsyn(cat, Optionname, "[cat <filename>]").
[3] unixsyn(lpr, Optionname, "[lpr <filename>]").
[4] unixsyn(ls, "[ls <filename>]").
[5] unixsyn(ls, -l, "[ls -l <directoryname>]".
The equivalent syntax for commands from MS-DOS are shown below.

[1] dossyn(more, X, [more, X, "< <filename>"]).
[2] dossyn(type, X, [type, X, "<filename>"]).
[3] dossyn(type, X, [type, X, "<filename> > prn"]).
[4] dossyn(dir, X, [dir, X, "<filename>"]).
[5] dossyn(dir, X, [dir, X, "<directoryname>"]).

4. The architecture of OSCON

The architecture of the OSCON system is defined as six distinct modules. There are two arguments for modularising any system: (1) It is much easier to update the system at any point, and (2) it is easier to map the system over to another domain. The six modules in OSCON are as follows: (1) ParseCon: natural language syntactic grammar parser which detects query-type, (2) MeanCon: a natural language semantic grammar which determines query meaning, (3) KnowCon: a knowledge representation for understanding, (4) DataCon: a knowledge representation for solving, (5) SolveCon: a solver for resolving query representations against knowledge base representations, and (6) GenCon: a natural language generator for generating answers in English.

ParseCon consists of a set of predicates which read natural language input and determine the type of query being asked by the user. For each type of query there are tests for characteristic ways of asking that query.

MeanCon consists of predicates which check queries for important information. There are predicates which check for mentioned (1) command names (e.g. ‘‘ls’’, ‘‘more’’), (2) command-effect specifications (e.g ‘‘see a file’’), and (3) concepts (e.g. ‘‘file’’, ‘‘directory’’). In case (2) there are specific types of information searched for: (1) **verb** specifying action (e.g. ‘‘see’’, ‘‘remove’’), (2) **object** of action (e.g. ‘‘file’’), (3) **modifier** of object (e.g. ‘‘contents’’), and (4) **location** of object (e.g. ‘‘screen’’). MeanCon also

checks for option verbs (e.g ‘‘number’’) and option verb objects (e.g. ‘‘lines’’). MeanCon contains a dictionary of English words that define categories such as ‘‘person’’, ‘‘modifier’’, ‘‘article’’, ‘‘quantifier’’ and ‘‘prepositions’’.

KnowCon consists of a set of data files to represent the knowledge about the domain language used for understanding English queries. Files contain information about verbs which categorise types of command or action. Examples of categories of action are: (1) creating, (2) screenlisting, (3) printerlisting, (4) sending, (5) transferring, and (6) removing. KnowCon also contains grammar rules for Operating System objects like ‘‘date’’, ‘‘file’’ and ‘‘directory’’. The grammar rules encode characteristic ways in which people talk about the objects in English.

DataCon consists of a set of data files defining detailed information about Operating System commands. This information is stored for the UNIX and MS-DOS Operating Systems. The data for UNIX is split among seven files: (1) command effects, (2) command preconditions, (3) command syntax, (4) command names, (5) command precondition options, (6) command effect options, and (7) command name options. The first four files contain basic data about commands while the last three contain data for options. For MS-DOS, data is contained in just four files which are similar to the first four here.

SolveCon is a solver which constructs and matches representations of user queries (called Formal Queries) against DataCon and produces an instantiated Formal Query which serves as an answer for the query. SolveCon is the driver of the OSCON program because it contains the information for mapping English sentences into instantiated answers. It contains a set of complex rules which call other OSCON modules to determine (1) query type, and (2) the instantiated Formal Query for that query.

GenCon is the natural language generator for OSCON and maps instantiated information from SolveCon into English answers. Here, there are algorithms for printing out (1) effects, (2) preconditions, and (3) syntax of commands. Also, there are predicates for printing out examples of the use of commands and command compositions. The type of query asked by the user determines the information mapped to the user.

5. Query coverage of OSCON

The problem with building effective natural language interfaces is that there are many ways of asking English queries. The system must attempt to capture all the different possibilities. One way to do this is to capture the basic types of queries that people ask. This gives the system the power of answering a large number of queries when it caters for each type.

There are four basic types of query that people ask about Operating Systems. These are: (1) request-for-attribute of mentioned command (e.g. ‘‘What does rm do?’’), (2) request-for-explanation(command) (e.g. ‘‘What is more?’’), (3) request-for-command for mentioned effect (e.g. ‘‘How do I see my file on the printer?’’), and (4) request-for-explanation(concept) (e.g. ‘‘What is a file?’’). There are three cases of type (1): (1) request-for-attribute(effect) (e.g. ‘‘What does rm do?’’), (2) request-for-attribute(syntax) (e.g. ‘‘What is the syntax of cp?’’), and (3) request-for-attribute(precondition) (e.g. ‘‘What is needed for rm?’’). Each of these basic query types can also be asked in terms of options. Examples are, ‘‘What option of ‘ls’ shows the number of bytes in my files?’’ (request-for-option + mentioned command), ‘‘What does ls -l do?’’ (request-for-effect + option), ‘‘How do I rename a file without having reported errors?’’ (request-for-command + option), ‘‘What are the options on ls?’’ (request-for-options of command), ‘‘What does the -i option normally do?’’ (request-for-explanation(concept)). Users can also ask queries involving command compositions. An example is, ‘‘How do I list my files and print them on the printer?’’ This query involves a query about listing files (request-for-command) and then printing them on the printer (request-for-command).

The OSCON program currently answers (1) the four basic query types, (2) queries about options, and (3) command composition queries for both the UNIX and MS-DOS Operating Systems. The fact that queries are of a given type aids in understanding and generating answers to them. For example, queries of type (1) above will always include a command name. Therefore, the parser for OSCON could check for command names and if it found them, then discover that the query was of type (1). Also, the generator would generate an answer, in a particular format, depending on the type of query. Rules of thumb such as these also speed up the time it takes OSCON to answer queries. Although one can add such rules of thumb into the interface it does not reflect a short-cut to natural language parsing. For example, there is no such short cut to understanding the query, ‘‘How do I print a file on the Imagen with no page burst?’’ Understanding queries is a combination of both (1) filtering the query type, and then (2) understanding the query. Examples of queries answered by OSCON[1.2] are shown in Appendix A. These examples are listed by query type.

6. The solving algorithm

The Solver basically searches queries for three types of information: (1) Command Names, (2) English Descriptions of command effects, and (3) Concepts. The search process in conducted in the following order.

[1] SolveCon checks to see if a command name is mentioned in the query. Then, (a) SolveCon checks if an option (e.g. -l) is mentioned. If (a) fails then (b) SolveCon checks if the query is request-for-option. This check is

done by having ParseCon check the syntax of the query, having MeanCon check for an English Description of an option effect. If either (a) or (b) are satisfied SolveCon will retrieve from the database Option Preconditions, Option Effect, Option Syntax, and Option Name.

If (a) and (b) have both failed then (c) SolveCon checks if the query is a request-for-attribute(precondition), request-for-attribute(effect) or request-for-attribute(syntax) query. Here, SolveCon checks the syntax again using ParseCon. If (c) fails, then (d) SolveCon checks the query for request-for-explanation(command) query. ParseCon is involved here too. If either (c) or (d) are satisfied SolveCon will retrieve Command Preconditions, Command Effect, Command Syntax, and Command Name from the database. If (d) fails then SolveCon moves on to step [2].

[2] SolveCon checks the query semantics. In this case the user must have asked an English query with no command names. (i) SolveCon has ParseCon check the syntax of the query. (ii) Then, SolveCon calls MeanCon to check for a Primary Verb, Verb Object, Modifier, and Location. SolveCon will retrieve Command Preconditions, Command Effect, Command Syntax, and Command Name from the database. Next, (iii) SolveCon has MeanCon check for a Secondary Verb (option action), and Secondary Verb Object. SolveCon will retrieve from the database Option Preconditions, Option Effect, and Option Syntax. If step [2] fails then SolveCon goes on to step [3].

[3] SolveCon checks the query semantics. In this case the user must have asked an English query involving no command names. Also, the query must be about command combinations, or pipes, otherwise step [2] would have passed. SolveCon checks for the existence of a command combination in the user query. SolveCon has MeanCon check for the existence of a sentence connector like ‘‘and’’. If this occurs then is it possible that the query involves command combination. SolveCon then calls the SolveCon algorithm again for (1) the piece of the query before the connector, and (2) the piece of the query after the connector. The data returned from (1) and (2) is integrated. If [3] fails then SolveCon tries step [4].

[4] SolveCon checks query syntax through ParseCon. Then MeanCon searches for concepts mentioned in the query. Examples of such concepts are ‘‘ada’’ and ‘‘protection’’.

6.1. Structures returned

The step of SolveCon which succeeds will return an instantiated Formal Query to the generator. In step [1], if cases (a) or (b) succeed, an instantiated Formal Query will be returned containing the following: (1) Option Preconditions, (2) Option Effect, (3) Option Syntax, (4) Option Name, and (5) Query Type. In step [1], if cases (c) or (d) succeed, the instantiated

Formal Query contains: (1) Command Preconditions, (2) Command Effect, (3) Command Syntax, (4) Command Name, and (5) Query Type.

In step [2] the Formal Query returned will contain slots for: (1) Command Preconditions, (2) Command Effect, (3) Command Syntax, (4) Option Preconditions, (5) Option Effect, (6) Option Syntax, and (7) Query Type. The complete structure will be instantiated when step [2] involves options. However, only parts (1),(2),(3), and (7) are instantiated when there is no mention of options.

With step [3] a list containing two instantiated Formal Queries is returned. Each formal query will contain: (1) Command Preconditions, (2) Command Effect, (3) Command Syntax, (4) Command Name, and (5) Query Type.

In step [4] a Formal Query with three pieces of information is returned. The structure contains (1) the Concept Name, (2) Concept Description, and (3) Query Type.

7. Natural language generation (GenCon)

The final phase of the OSCON program is to map an instantiated formal representation into an English answer. There are two types of answer which may be returned to the user: (1) Stored English sentences describing some concept which are contained in the DataCon knowledge base, and (2) English sentences mapped out from instantiated Formal Queries.

The natural language generator for the OSCON system is used to map instantiated formal queries into English answers. The generator has five primary components:

[1] **psyntax:** gives the syntax for a command
[2] **peffect:** gives the effect of some command
[3] **pexample:** gives an example on the use
of some command
[4] **ppre:** gives the preconditions for some command
[5] **ppipe:** gives the commands involved in some
piping example and an example of the piping

For each of the major query types various configurations of printing components are used. There are three types of request-for-attribute query: (1) request-for-attribute(precondition), (2) request-for-attribute(effect), and (3) request-for-attribute(syntax). In these cases the components [1], [2], [3] and [4] are combined in order. Printing out the syntax for some command is trivial. The syntax is already stored in the DataCon knowledge base. This is just returned to the user. Printing out preconditions is quite trivial too as all GenCon has to do is to print those preconditions retrieved from the DataCon precondition information.

Generation for request-for-attribute(effect) queries is more complex. GenCon will print command effects by (1) checking to see if the output should be in plan/pipe form, and if it is, then generating the answer in plan/pipe form; (2) generating the (a) Command Syntax, (b) Effect and (c) Preconditions for the command. The Effect is generated from the instantiated Formal Query produced by SolveCon which contains action, object, object modifier and object location. The latter information is generated in sentence form. Some interleaving information such as the output of prepositions between object modifier and location are handled too.

For request-for-command queries the latter algorithm is used. For request-for-explanation(command) components [1], [2], [3] and [4] are used. For request-for-explanation(concept), the answer is output from a stored piece of text. Often users ask queries about commands as concepts. The generation of these is simple as the definitions of such concepts are just stored as English descriptions in the first place. Therefore, all GenCon has to do is to map the stored sentences into English answers. We have written a simple algorithm which maps the English text into pretty format on the screen.

8. An example

In this section we show an example of how the query ‘‘How do I see my files with numbered lines?’’ is understood and answered by OSCON. First, SolveCon attempts to find out the type of query being asked. Initially, SolveCon tries step [1] to match the query as one mentioning a command and there is no match. Then step [2] is tried and a match occurs. The query is a semantic one. ParseCon is called forth and a syntactic match is found. ‘‘How do I’’ matches the syntactic form [how, do, i] for what-command queries.

Next, SolveCon calls MeanCon which analyzes the meaning of the sentence. (i) Findverb checks for a verb and gets ‘‘see’’. From KnowCon, ‘‘display’’ is marked as the action. Then, (ii) findobj checks ‘‘How do I’’ and ‘‘my files with numbered lines’’ for objects. ‘‘Files’’ is matched as an object. Data from the query (i.e. **display(file,contents,*)**) is matched against the DataCon database Effects and a match is found with **display(file,contents,screen)**. This effect match from the query data to DataCon will allow SolveCon to retrieve (1) Command Preconditions ([file]), (2) Command Effect (display(file,contents,screen)), (3) Command Syntax (cat -n <filename>), and (4) Command Name (cat).

Next, the query is checked for existence of a secondary action. The findverb predicate retrieves ‘‘numbered’’ as a secondary action and its object is retrieved as ‘‘lines’’. This representation is matched against the option database and number(lines) from the query matches number(lines) in the

Option Effect definition in the database. The DataCon database is referenced and (1) Option Preconditions, (2) Option Effect, and (3) Option Name are returned. The data retrieved is integrated to form an instantiated Formal Query which is passed to the generator. The representation for this query is as shown below.

```
cquery(display(file,contents,screen),      % Command Effect
    "with numbered lines",          % Option Effect
    [cat -n <filename>],            % Syntax
    [file]),                        % Preconditions
    cat),                           % Command Name
    -n).                            % Option Name
```

The GenCon generator takes the Formal Query and maps it into an English answer. (i) psyntax will print out ‘‘cat -n <filename>’’, (ii) peffect will display ‘‘will display file contents on the screen.’’, and (iii) popeffect will display ‘‘with numbered lines.’’ Therefore, the complete answer generated for this query is:

‘cat -n <filename>’ will display file contents on the screen with numbered lines.

9. Current state of the implementation

OSCON can answer queries in 2.5 seconds when the Sun computer is at low load. OSCON answers four major types of query (1) request-for-attribute of mentioned command, (2) request-for-explanation(command), (3) request-for-command for mentioned effect, and (4) request-for-explanation(concept). There are three cases of type (1): (1) request-for-attribute(precondition) (2) request-for-attribute(effect), and (3) request-for-attribute(syntax). In real terms this brings the query types covered up to six. The system answers the latter three query types with options and will soon answer the other types with options. OSCON’s database contains information on 40 UNIX and 40 MS-DOS commands with their respective preconditions, effects, syntax and command names. OSCON has 20 grammar rules for understanding the ways that users ask queries about categories of commands. OSCON also contains 10 plan sets for possible combinations of commands.

10. Comparison to other systems

There are basically two other natural language consultants for Operating Systems. These are the Unix Consultant (UC), and the Sinix Consultant (SC).

The Unix Consultant (UC) (see Chin 1988, Wilensky et al. 1984, 1986, 1988) is a natural language consultation system for UNIX, and is in many ways similar in scope and intent to OSCON. However, there are some key differences: (1) While OSCON is a specialised system with the sole goal of providing detailed expert help, UC has the broader goal of studying knowledge representation and planning (as well as natural language understanding) at a basic level. As such, UC takes a fundamentally different approach to knowledge representation, in that employs a much more general knowledge representation and planning mechanism, with a somewhat cognitive flavor. UC contains only one knowledge base of Operating System concepts which must serve both the understanding and the solving phases of query resolution, while OSCON incorporates specialised representations for each function. Hence, UC must sacrifice some of the specialisation built into OSCON.

(2) Not unexpectedly, an examination of the examples presented in Wilensky et al. (1986) suggests that UC will have to perform far more extensive procedures to solve a given query, and that it may not be able to answer some of the very detailed queries that OSCON will address; (3) UC is not intended to be a consultant which will help users migrating from one system to another. UC is a consultant for the UNIX Operating System. We do not claim that the UC system could not act as a consultant on other Operating Systems. We claim that the Berkeley team have not concerned themselves with putting information from more than one Operating System in their program; (4) It is a characteristic of computer Operating Systems that commands can be combined in various ways to execute complex processes. In Wilensky et al. (1986, p. 6-10) there are a number of examples of the queries which the UC system handles. These examples are intended to show the kinds and scope of requests answered by the system. None of the examples involve command compositions but operations of single commands. The UC system has no capability for answering complex natural language queries which involve command sequencing; UC and OSCON are not truly competitors, but rather systems with different emphases.

The Sinix Consultant (SC) (see Kemke 1986, 1987; Hecking et al., 1988) is a broad-based UNIX help system It is similar to OSCON in that it is designed from the start to be an Operating System consultant rather than a more general system into which a UNIX model is embedded. Although SC contains a rich knowledge base, which reflects the technical aspects of the domain as well as the users view of the system, the focus of SC, like UC, is to provide help on the use of individual commands, rather than the interconnection of several. Our knowledge representation of plans has more emphasis on answering complex queries. There is no mechanism whereby the SC system will understand complex command sequences. However, the authors do predict in Hecking et al. (1988) that future development of the SINIX

Knowledge Base will include ‘‘combinators’’ for I/O-redirection and pipelining. The SC system has a dialogue modeling component, for (1) handling elliptical queries, and (2) anaphora.

There are several other consultation systems for Operating Systems, including CMS-HELP developed by Yun (1984), TVX by Billmers & Garifo (1985), Wizard (see Shrager & Finin 1982; Finin 1983), and USCSH by Matthews & Pharr (1987). However, all of these systems appear to be either far less ambitious in scope than OSCON, or else have a totally different emphasis. In particular, all appear to employ rather simple surface models of UNIX.

11. Conclusion and further work

It is concluded that it is possible to build a computer program which will answer natural language user queries about Operating Systems. We have shown how this is done by the OSCON system. The OSCON program answers English queries in English. Each query is answered within seconds.

The development of any program like OSCON is enhanced if the program is built in a modular form where each module has a distinct function. This has been done for OSCON which is divided up into six distinct modules. This methodology allows for easy update of the program and also will allow the possibility of mapping the program over to a new domain.

In comparing OSCON to the other three systems available today OSCON turns out to have a different emphasis. The OSCON system is more concerned with the detail of Operating Systems and natural language processing rather than an experiment on cognitive modeling. Also, OSCON is the only system which answers queries involving command combinations.

There are three areas of further work proposed for OSCON which are all part of a dialogue interface to be added to the system. These are: (1) context storage mechanisms, (2) reference determination algorithms, and (3) user modeling capabilities (see Chin, 1988). Dialogue management is important and it will permit the user to ask queries without having to spell out queries in elaborate English. Initial thoughts on this research are reported in Ball et al. (1989).

12. Acknowledgements

We would like to thank Yorick Wilks for guidance on the natural language aspects of OSCON and Louise Guthrie for programming parts of the system. Also, Catherine Marshall, Hans Brunner, Andy Parng and Scott Wolff of the Intelligent Customer Assistance Project at U S *WEST* Advanced Technologies are thanked for consultations on this research.

13. References

Ball, Jerry, John A. Barnden, Sylvia Candelaria de Ram, David Farwell, Louise Guthrie, Cheng-Ming Guo, Stephen Helmreich, Paul Mc Kevitt, and Liu Min (1989) *The need for belief modelling in natural language processing*. In Proc. of the International Conference on Cross-Cultural Communication (ICC-CC-89), Trinity University, San Antonio, Texas, March.

Billmers, Meyer A. & Michael G. Carifio (1985) *Building knowledge-based operating system consultants*. Proceedings of the Second Conference on Artificial Intelligence Applications, pp. 449-454, Miami Beach, December.

Chin, David (1988) ‘‘Exploiting user expertise in answer expression’’ In Proceedings of the Seventh National American Conference on Artificial Intelligence (AAAI-88), St. Paul, Minnesota, Vol. 2, 756-760, August.

Guthrie, Louise, Paul Mc Kevitt & Yorick Wilks (1989) *OSCON: An operating system consultant*. In Proc. of the Fourth Annual Rocky Mountain Conference on Artificial Intelligence (RMCAI-89), Subtitled ‘‘Augmenting Human Intellect By Computer’’, 103-113, Registry Hotel, Denver, Colorado, June.

Hecking, M. C. Kemke, E. Nessen, D. Dengler, M. Gutmann & G. Hector (1988) ‘‘The SINIX consultant —a progress report’’ Memo Nr. 28, Universitat des Saarlandes, FB 10 Informatik IV, Im Stanwald 15, D — 6600 Saarbrucken 11, Fed. Rep. of Germany, August.

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.

Kemke, Christel (1987) *Representation of domain knowledge in an intelligent help system*. In Human-Computer Interaction — INTERACT ’87, H.J. Bullinger and B. Shakel (Eds.), pp. 215-220. Amsterdam: Elsevier Science Publications B.V. (North-Holland).

Matthews, Manton & Walter Pharr (1987) *Knowledge acquisition for active assistance.* Preprints of the First International Workshop on Knowledge representation in the UNIX help domain, University of California, Berkeley, California, December.

Mc Kevitt, Paul (1986) *Formalization in an English interface to a UNIX database*. Memoranda in Computer and Cognitive Science, MCCS-86-73, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001.

Mc Kevitt, Paul (1988) *Rules of inference in an operating system consultant*. In Proc. of the First Irish National Conference on Artificial Intelligence and Cognitive Science (AI/CS-88), Vol. 1, University Industry Center, University College Dublin, Dublin, Eire (Republic of Ireland), European Community (EC), September.

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, 569-575, Milan, Italy, August.

Shrager, Jeff & Tim Finin (1982) *An expert system that volunteers advice*. Proceedings National Conference on Artificial Intelligence (AAAI-82), pp. 339-340.

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

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

Wilensky, Robert; David N. Chin, Marc Luria, James Martin, James Mayfield and Dekai Wu (1988) *The Berkeley UNIX Consultant project*. Computational Linguistics, Vol. 14, No. 4, 35-84, December.

Yun, David Y & David Loeb (1984) *The CMS-HELP expert system*. Proceedings International Conference on Data Engineering, IEEE Computer Society, 459-466, Los Angeles.

14. Appendix A

This appendix contains a list of some of the queries which OSCON[1.2] currently answers. The maximum time taken to answer any query is 2.5 seconds. The query sets have been divided into four major types (1) request-for-attribute of mentioned command, (2) request-for-explanation(command), (3) request-for-command for mentioned effect, and (4) request-for-explanation(concept).

```
| ?- oscon.
OSCON Program 1.0 (Sun-3, Sun Unix 4.2)
Copyright (C) 1988, Computing Research Laboratory.
```

Dept. 3CRL, Box 30001, NMSU, Las Cruces, NM (505) 646-5466
U S WEST Advances Technology
Please input your question ending with with a ‘?’
Write ‘quit.’ when you are finished

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-attribute(effect)

--> what does ls do?
‘ls <directoryname>’ will display directory contents on the screen.

--> what happens with rm?
‘rm <filename>’ will remove a file from your current directory.

--> what is the effect of date?
‘date’ will display a date on the screen.

--> what does more -c do?
‘more -c’ will display each page after clearing the screen.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-attribute(syntax)

--> what is the syntax of rm?
It has the syntax ‘rm <filename>’.

--> How do i use cp?
It has the syntax ‘cp <file1> <file2>’.

--> how does he use more?
It has the syntax ‘more <filename>’.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-explanation(command)

--> what is ls?
‘ls’ is a command.
It has the syntax ‘ls <directoryname>’.
‘ls <directoryname>’ will display directory contents on the screen.

--> what is more?
‘more’ is a command.
It has the syntax ‘more <filename>’.
‘more <filename>’ will display file contents on the screen.
‘more’ only takes the argument ‘file’

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-command for mentioned effect

--> how do i see my file?
‘more <filename>’ will display file contents on the screen.

--> how do i see my file on the screen?
‘more <filename>’ will display file contents on the screen.

--> how do i see my file on the printer?
‘lpr <filename>’ will display file contents on the printer.

--> how do i see all my files?
‘more <filename>’ will display file contents on the screen.

‘ls <directoryname>’ will display directory contents on the screen.

--> how do i see this file?
‘more <filename>’ will display file contents on the screen.

--> how do i see that file?
‘more <filename>’ will display file contents on the screen.

--> how do i see his files?
‘more <filename>’ will display file contents on the screen.
‘ls <directoryname>’ will display directory contents on the screen.

--> how do i read my files?
‘more <filename>’ will display file contents on the screen.
‘ls <directoryname>’ will display directory contents on the screen.

--> how do i list my files and print my files?
‘more <filename>’ will display file contents on the screen.
‘lpr <filename>’ will display file contents on the printer.
To display file contents on the printer use ‘more <filename> | lpr’

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-explanation(concept)

--> what are aliases?
alias is the ability to establish shorthand names
for frequently used but long-winded commands.

--> quit.

over
yes
| ?-