The OSCON Operating System Consultant

Paul Mc Kevitt
Center for PersonKommunikation (CPK)
Fredrik Bajers Vej 7-A5
Institute of Electronic Systems (IES)
Aalborg University
DK-9220, Aalborg, Denmark.
E-mail: pmck@cpk.auc.dk

Abstract. OSCON (Operating System CONsultant) is an operating system consultant which gives English answers to English queries about computer operating systems. The program currently answers queries for over 40 commands from the UNIX[1] and MS-DOS[2] operating systems. OSCON answers a wide variation of queries that users may wish to ask. 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 and answers queries in less than 2.5 seconds. An empirical study with the Wizard-of-Oz technique provides important data for the further development of OSCON.

Key words: Natural Language Processing (NLP), operating systems, intelligent help, UNIX

1. Introduction

This paper describes the OSCON (Operating System CONsultant) program and its capabilities. OSCON is a natural language interface which answers English queries about computer operating systems. Although a comprehensive description of OSCON is given here further details are found in Mc Kevitt (1986, 1987, 1988), Mc Kevitt and Wilks (1987), Mc Kevitt and Pan (1989), and Mc Kevitt and Ogden (1989a). The user enters English queries to OSCON and the program answers them in English. The program is written in Quintus Prolog, runs on a Sun-3/ME-4 computer, and the maximum time taken to answer a given query is 2.5 seconds. OSCON answers queries for over 40 commands from the UNIX and MS-DOS operating systems. OSCON has been constructed to answer three basic types of query although recent studies indicate that there are at least nine basic query types (see Mc Kevitt and Ogden, 1989a,b). 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

[1] UNIX is a trademark of X/Open, Inc.
[2] MS-DOS is a trademark of Microsoft Corporation.

so that it is easily updated and can be easily mapped over to other domains.

2. 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 try and 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.

OSCON was designed to cater for three basic types of query. These query types were decided a priori before any data analysis was conducted. These query types are: (1) *request for information* (e.g. "What does rm do?"), (2) *request for explanation* (e.g. "What is a pipe?"), and (3) *request for direction* (e.g. "How do I see my file on the printer?"). Three cases of type (1) are considered important: (1) *request for information[effect]* (e.g. "What does rm do?"), (2) *request for information[form]* (e.g. "What is the syntax of cp?"), and (3) *request for information[precondition]* (e.g. "What is needed for rm?"). Two cases of type (2) were analyzed: (1) *request for explanation[command]* (e.g. "What is more?"), and (2) *request for explanation[concept]* (e.g. "What is a file?"),

Each of these basic query types can also be asked in terms of options in the UNIX operating system. Examples are, "What option of 'ls' shows the number of bytes in my files?" (*request for information[option]*), "What does ls -l do?" (*request for information[effect]*), "How do I rename a file without having reported errors?" (*request for direction*), "What are the options on ls?" (*request for information[options]*), "What does the -i option normally do?" (*request for information[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 direction*) and then printing them on the printer (*request for direction*).

The OSCON program currently answers queries constituting (1) the three 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, *request for information* queries usually include a command name. Therefore, the parser for OSCON could check for command names and if it found them, this would be an indication 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 Laserwriter 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 are shown in Appendix A. These examples are listed by query type.

3. The architecture of OSCON

The architecture of the OSCON system is defined as six distinct modules. There are two arguments for modularizing 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 programs which read natural language input and determine the type of query being asked by the user. We have described the three basic types of query in section 2 above. For each type of query there are tests for characteristic ways of asking that query.

MeanCon consists of programs 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 categorize 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 preconditions, (2) command effects, (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) preconditions, (2) effects, and (3) syntax of commands. Also, there are routines 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.

4. Knowledge representation in OSCON

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

The knowledge about language stored includes words used to refer to command actions. For example, a user may use the words "delete", "remove", "get rid of", "erase" and so on to ask a query about deleting files and directories. These words must be stored under the general category of delete. Also, there are many ways in which people ask queries about actions. For example, if you are asking about copying a file you will probably specify the file which you wish to copy. If you are asking about "displaying" you will specify what you wish to display and where you wish to display it. This type of knowledge is called "understanding knowledge" and is stored within a module of OSCON called KnowCon.

More detailed knowledge about operating systems is contained in a database called DataCon. This type of knowledge includes command preconditions, command effects, command syntax, and the names of commands. Also, stored here is (1) knowledge about options for commands, (2) English descriptions of operating system concepts like "files" and "directories", and (3) knowledge about plans or possible command combinations (e.g. 'ls' can precede 'lpr' but the converse is not true). The knowledge stored here is for the UNIX and MS-DOS operating systems. The distinction between the two types of knowledge is that one is language oriented and the other is domain oriented. One type of knowledge is used for understanding queries and the other for solving queries. This is the principle of separation of understanding and solving defined in Hegner (1988, 1998).

4.1. Knowledge for understanding (KnowCon)

There are two types of understanding knowledge stored in the KnowCon module: (1) data on operating system action reference, and (2) data on descriptions of operating system objects. The first type of knowledge includes sets of words or phrases that may refer to some operating system action or command. For example, the words, "print", "print out", and "get a copy" would indicate that the user was referring to printing something on the printer. Such words and phrases are stored as being associated with the general concept of printing. The second type of knowledge is used for defining the ways that users refer to operating system objects. For convenience the data have been split up into two types (a) data on files and directories, and (b) data on other operating system objects. In type (a) there are grammar rules specifying how users refer to files and directories, and in type (b) there are rules for how users refer to other operating system objects.

4.1.1. *Referring to actions*

In asking queries about operating systems users commonly use a well defined set of verbs or verb phrases. For example, if a user wants to know about removing files or directories he/she will use the following verbs and phrases: "delete", "remove", "get rid of", "erase" and so on. Queries about copying may be referenced by: "copy", "transfer", and "move". These phrases should be captured by the OSCON program.

Phrases and words are stored under their categories in the KnowCon module of the program.

It is possible to divide the set of operating systems commands into various categories. We have defined three major command categories and each of these have various subdivisions. The three categories are "listing", "altering" and "compiling" commands. Listing commands are those which display information about the state of files in an operating system. Altering commands are used to alter the state of files in the system. Compiling commands are used to compile files in the system rather than display/alter them. There are two types of listing command, (1) screenlisting, and (2) printerlisting. Screenlisting commands are those which allow the user to see information on the screen and printerlisting will do the same for the printer. There are three subtypes of screenlisting command, (1) display-file (e.g. "more", "nroff", "cat"), (2) display-file/directory-information (e.g. "ls"), and (3) display-system-information (e.g. "users", "who", "ps", "jobs"). There is only one subtype of printerlisting command (e.g. "lpr", "runoff"[3], "itroff").

Altering commands are of three types, (1) creating (e.g. "gemacs", "vi", "mkdir"), (2) removing (e.g. "rm", "rmdir", "kill"), and (3) transferring (e.g. "mv", "cp"). Each of these can be applied to either files or directories. A third type of command is compiling commands. These commands cannot be categorized under altering commands as they are really utilities which do not change files but use them. An example of such a command is "run". It is important to point out that certain commands apply to files which are not directories and others only apply to files which are directories. For example, "more" can only be used on files and not directories and "rmdir" can only be used on directories and not files. Such information can be used to specify the preconditions for commands and enables the system to detect errors in user queries and inform the user about such errors. For example, if a user asked, "How do I use "more" to display a directory?" the system could locate that the precondition for "more" is that it only works over files. Then the system could tell the user this information.

The rules listed below are examples of typical action rules in KnowCon (bracketed items indicate input strings to be matched with). Rules [1] and [2] show typical verb phrasings used to reference the action of screenlisting. Rule [3] is for printerlisting and rules [4] and [5] for creating.

[3] "runoff" is a command defined at the Computing Research Laboratory, New Mexico State University, USA to load text formatting packages for word processing a text file.

```
[1] screenlist --> [see].
[2] screenlist --> [look, at].
[3] printerlist --> [print].
[4] create --> [edit].
[5] create --> [produce].
```

4.1.2. *Referring to objects*

Queries about operating systems often include reference to operating system objects like files and directories. Typically, users will refer to the object which is operated over by some command. The phrasing of the English query will dictate the object present. KnowCon has objects separated into two categories: (1) file/directory object specifications, and (2) other operating system object specifications.

The rules listed below are examples of file/directory object specifications (bracketed items indicate input strings to be matched with and non bracketed items indicate non-terminals in the grammar). File object specifications include grammar rule definitions for files and directories. The first rule definition below [1] specifies that a file can be mentioned in a query by the word "file" preceded by three modifiers. The first modifier can be (1) a quantifier (e.g. "all", "some of"), and the second (2) a possessive (e.g. "my", "our"). Then *filemod* can be a modifier of type of file. Examples are "mail", "device", "plain", "executable" and so on. This grammar rule will capture most of the ways that a user might refer to a file.

The second rule [2] shows that a directory may have two modifiers and then the word referring to directory itself. Rule [3] shows that a file or directory could be in reference. This happens because it is ambiguous as to whether the intended referent is file or directory. The user, by using the plural of file may intend directory[4] rather than the plural for file.

Rule [4] defines location to be a triple: (1) preposition (prep), (2) modifier (mod), and (3) directory (dir). Phrases like "..in my directory", "..in our directory", and "..in the directory" will match here. There are more specific definitions of location for where the location is. Rule [5] is a definition of screen location in terms of (1) preposition (prep), (2) modifier (mod), and (3) screen-output (soutput). Preposition and modifier are self explanatory and output is the location of output. This could be "terminal" or "screen". Rule [6] is the equivalent definition for a printer location. The output specification here is for a printer. Then poutput can be (1) "printer", (2) "imagen" or (3) "laser writer".

[4] A directory is a set of files.

```
[1] file --> mod, mod, filemod, [file].
[2] directory --> mod, mod, [directory].
[3] fileordir --> mod, mod, filemod, [files].
[4] location --> prep, mod, dir.
[5] slocation --> prep, mod, soutput.
[6] plocation --> prep, mod, poutput.
```

The second type of object definitions include objects other than those to do with files and directories. Some examples are shown below. Rule [1] shows the definition of a queue in terms of (1) a modifier, followed by (2) [printer, queue]. There are definitions of users by rules [2] and [3]. Rules [4] and [5] define names. The definitions of objects are simply definitions of the various ways which users refer to the objects.

```
[1] queue --> mod, [printer, queue].
[2] users --> names, mod, [users], prep, mod, [system].
[3] users --> names, [system, users].
[4] names --> mod, [names, of].
[5] names --> [].
```

4.2. 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.

4.2.1. *Basic command representation*

There are basically four types of information about any command held in the database. These are (1) preconditions, (2) effects or postconditions, (3) syntax, and (4) command names. Preconditions are lists of objects that are necessary for a command to be executed. Here are some examples of preconditions for commands from UNIX. Facts [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, []).
```

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. The facts below show some effects for UNIX commands ('@' acts as a slot filler where an extra slot is not required in some cases). Fact [1] shows the effect for the command "more". The object for "more" is "file" 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 together. 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. Fact [7] describes the "gemacs" command which creates files, and fact [8] the "rm" command which deletes them.

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

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

```
[1] syn(more, Optionname, "[more <filename>]").
[2] syn(cat, Optionname, "[cat <filename>]").
[3] syn(ls, Optionname, "[ls <directoryname>]").
[4] syn(users, Optionname, "[users]").
[5] syn(gemacs, Optionname, "[gemacs <filename>]").
```

4.2.2. *Option representation*
DataCon also contains information for option specifications of commands. DataCon has files for (1) Option Preconditions, (2) Option Effects, (3) Option Names. There is no distinction for option syntax as this is practically the same for commands with options or not.

Option preconditions are defined as three-place lists with the (1) Command Name, (2) Option Name, and (3) Precondition List. Shown below are a set of options for the various commands. The Precondition List contains the objects which must be present for the command to be executed. Facts [1], [2] and [3] show that each option for "cat" has "file" as precondition. Facts [4] and [5] show that for one option of "cp" (i) there is no precondition, and for the other option (r), "directory" is the precondition. Facts [6] and [7] show the preconditions for "ls".

```
[1] opprecon(cat, n, [file])
[2] opprecon(cat, s, [file])
[3] opprecon(cat, v, [file])
[4] opprecon(cp, i, [])
[5] opprecon(cp, r, [directory])
[6] opprecon(ls, f, [directory])
[7] opprecon(ls, l, [])
```

The definition of the Option Effects are also stored as strings of English words. This is merely a convenience for generation. The definitions will enable the generator to give more detail about the specific effect of some option. Each fact here contains (1) Command Name, (2) Option Name, and (3) Option Effect with an English description.

```
[1] opeffect(more, #,
        "set the window size to # lines").
[2] opeffect(more, c,
        "display each page after cleaning screen").
[3] opeffect(more, d,
        "prompt to hit space to continue at each screen").
[4] opeffect(cp, i,
        "prompt file name when overwriting").
[5] opeffect(cp, p,
        "preserve the modification information from
        the source").
[6] opeffect(cp, r,
        "copy each subtree rooted at that directory").
```

Option Name definitions are similar in spirit to the Command Effect specifications defined in section 4.2.1 above. However, the definitions here are for specific options. The option definitions have three arguments: (1) Option Effect defined as an action (object) specification, (2) the Command Name, and (3) Option Name. The first three facts [1], [2] and [3] show the option variations on "cat". The various options allow the displaying of file contents in a specific manner. We also show the options for "ls" (facts [4]-[10]) and for "cp" (facts [11]-[13]). Note that a characteristic of options is that sometimes they have the same action as the main action (e.g. display and display(non-printing-characters)) while other times they have a different action (e.g. display and squeeze(blanklines)).

```
[1] opcomm(number(lines), cat, n).
[2] opcomm(squeeze(blank-lines), cat, s).
[3] opcomm(display(non-printing-characters), cat, v).

[4] opcomm(include(hidden-files), ls, a).
[5] opcomm(display(directory-name), ls, d).
[6] opcomm(display(directory-content), ls, f).
[7] opcomm(display(group-ownership), ls, g).
[8] opcomm(display(long-listing), ls, l).
[9] opcomm(sort(file-ages), ls, t).
[10] opcomm(display(subdirectories), ls, R).

[11] opcomm(prompt(overwriting), cp, i).
[12] opcomm(preserve(modification-information), cp, p).
[13] opcomm(copy(subdirectories), cp, r).
```

4.2.3. *Concept representation*

DataCon also contains a set of definitions of the possible concepts which a user may wish to ask about. Such concepts are considered to be operating system objects like files and directories, and concepts like piping and filtering. Concept representations are three place lists with (1) Concept name, (2) Concept name pluralized, and (3) English description of concept. We show the definitions for, "ada", "working directory" and "unix" here.

```
[1] con([ada],[ada],
    ``Ada is developed on behalf of the U.S. Department of
    Defense for use in embedded systems.  Ada is the first
```

```
    practical language to bring together important features
    such as data abstraction, multitasking, exception
    handling, encapsulation and generics.'').

[2] con([working, directory],[working, directories],
    ``The directory you are working in.'').

[3] con([unix],[unix],
    ``UNIX Operating System manages the resources of your
    computer system to perform useful work on your behalf.
    It is composed of three major parts: the kernel,
    the file system, and the shell.'').
```

4.2.4. *Plan representation*

DataCon contains a list of possible Plans for command sequences which users may ask about. These command sequences are defined in terms of predicates which have five arguments. The first three arguments represent (1) an action, (2) an object, and (3) location of output. The last two arguments define the second action and its output location. We show two example rules below. Fact [1] represents displaying directories on the screen and then on the printer and fact [2] likewise for files.

```
[1] plan(display,directory,screen,display,printer).
[2] plan(display,file,screen,display,printer).
```

5. Natural language understanding

The most important part of any program which acts as a natural language interface is the natural language understander. There must be some means of mapping a user query into a good meaning representation of that query. The problem, of course, is that there are very many ways of specifying queries in English. The first job of the natural language interface is to use some type of syntactic filtering which determines the type of query being asked. The filter will check queries for certain objects or phases and works as a fast mechanism for determining query type. The second job is that of determining the meaning of a query, or its content, once the type of query is determined. Let's take a look at the two components of OSCON which tackle these two problems.

5.1. Parsing natural language queries (ParseCon)

The parser called ParseCon has the job of determining the type of query present. There are characteristic ways of asking certain types of queries and the parser checks for these. The parser checks for the three basic types of query described in section 2. The ParseCon module has lists of the characteristic ways which people use to ask queries about particular query types.

One of the query types checked for is *request for explanation.* There are a number of characteristic phrases which are used. Some of these are listed below. The first set of facts [1-4] and facts [1a-1e] define the syntax of what would come before and after some concept respectively. The second set of facts [6-11] only define possible uses of syntax before the concept but do not reflect any after-concept syntax.

```
[1] firstphrase([what, does|X], o1).
[2] firstphrase([what, a|X],o2).
[3] firstphrase([what|X],o3).
[4] firstphrase([what|X],o4).

[1a] secphrase([mean|X],o1).
[2b] secphrase([is|X],o2).
[3c] secphrase([is|X],o3).
[4d] secphrase([are|X],o3).
[5e] secphrase([means|X],o4).

[6] wphrase([what, is, a]).
[7] wphrase([what, are]).
[8] wphrase([what, is]).
[9] wphrase([explain]).
[10] wphrase([describe]).
[11] wphrase([]).
```

Another query type checked for is *request for information[effect].* Here, again there are a number of possible characteristic phrases. We list some here [1-8]. Similarly, there are facts for (i) *request for information[precondition]*, (ii) *request for information[form]*, (iii) *request for direction*, (iv) *request for explanation[command]*, and (v) *request for explanation[concept]* queries. Of course, there are syntax facts which will be compatible for all query types. However, this is not a problem because there are other characteristics of query types which separate them out.

```
[1] firstdesc([what|X],o1).
[2] firstdesc([does|X],o2).
[3] firstdesc([does|X],o3).

[1a] secdesc([does|X],o1).
[2b] secdesc([do|X],o2).
[3c] secdesc([have|X],o3).

[4] desc([what, happens, with|X]).
[5] desc([result|X]).
[6] desc([results|X]).
[7] desc([uses, of|X]).
[8] desc([effect, of|X]).
```

5.2. Determining query meaning (MeanCon)

The function of the MeanCon component of OSCON is to determine query meaning. This is less of a problem for most query types than for *request for direction* queries. These queries involve complex phrasings of English to describe the effects which the user wishes to execute. The MeanCon component of OSCON has the function of determining the occurrence of objects in user queries. There are seven types of object searched for: (1) command name, (2) option name, (3) verb reference, (4) object, (5) object modifier, (6) object location, and (7) concept.

MeanCon has a predicate called *findcmd* which searches for command names. The mention of command name is a good indicator of the type of query being asked. If a command name is present this indicates that the query is probably about (1) command preconditions, (2) command effects, or (3) command syntax. MeanCon also has a predicate called *findopt* which searches for mention of option names in queries.

One of the most difficult types of user query to be understood by the system is *request for direction* queries. These are queries where the user knows what he/she wishes to accomplish but does not know the command to do that. In these cases the user will specify, in English, some process, or effect, which he/she wants to be executed. The process/effect can be described with (1) a primary verb and (2) a secondary verb. The user will use a primary verb to describe the main action/command in question. A secondary verb may be used to describe a restriction on the main action/command. This restriction could be a definition of some option specification for the major command.

Therefore, MeanCon has algorithms which check for (i) Primary Verb, (ii) Primary Verb Object, (iii) Object Modifiers, and (iv) Locations. There are also algorithms which search for (i) Secondary Verb, and (ii) Secondary Objects which usually describe option effects.

In checking for *request for direction* queries the user will usually specify some action with a verb. This will be followed by the mention of an object such as a "file" or "directory". There may be a modifier of the object such as "contents". The location of the object may also be specified such as "printer" or "screen" or "directory".

There's a predicate in MeanCon called *findverb* which searches for verbs in user queries. When a verb is located this will determine the major category of action/command. Therefore, the word "delete" will reference removing, "see" will reference displaying and so on. MeanCon uses the stored verb-action structures in KnowCon to find verbs in queries and their related action representations.

Another predicate called *findobj* searches for the object of a verb. Say, for example, the user had asked, "How do I see a file?" then the findverb predicate will locate the verb "see". The query is split up so that the phrase after the verb is checked for an object. The object "file" is located and marked.

More complex queries may include modifiers and locations. Take the query, "How do I see my file contents on the screen?" In this case, the phrase "|..file contents on the screen?" is checked for an object by *findobj*. Then, "|..contents on the screen?" is checked for modifier by *findmod*. Findmod locates "contents" as a modifier.

A predicate called *findloc* checks for locations in queries. Locations include "screen", "printer" and "directory". In our example, "|..on the screen?" is checked for location and "screen" is uncovered.

MeanCon has a predicate called *findcon* which checks for concepts in user queries. For example, the existence for the concept, "ada" will denote a query asking about, "ada".

6. Rules for solving (SolveCon)

Now that we have defined the data available to OSCON before the system begins to process a query we move on to describe the rules which match user queries to database information. SolveCon determines the answer for a query by (1) building an uninstantiated formal query from the query, (2) matching this structure to the DataCon database, (3) retrieving data from the database, and (4) using the data to build an instantiated formal query which is passed back to the GenCon natural language generator. We shall describe the answering process in two steps: (1) the algorithm used by SolveCon to specify the query and to retrieve data from the database, and (2) the structure of instantiated Formal Queries returned by SolveCon to the GenCon generator.

6.1. 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 direction[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 information[precondition]*, *request for information[effect]* or *request for information[form]* 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”.

6.2. 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 information* query: (1) *request for information[precondition]*, (2) *request for information[effect]*, and (3) *request for information[form]*. 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 information[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 direction* 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 *request for direction* 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 are 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

The OSCON program can answer three major query types and replies to each in seconds: (1) *request for information*, (2) *request for explanation*, and (3) *request for direction*. There are three cases of type (1): *request for information[precondition] request for information[effect]*, and (3) *request for information[form]* and two cases of type (2): *request for explanation[command]*, *request for explanation[concept]*. In real terms this brings the query types covered up to six. The system answers the three query types with options in the case of UNIX. OSCON's database contains information on 40 UNIX and 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 three other natural language consultants for operating systems. These are the Unix Consultant (UC), the Sinix Consultant (SC), and UCC/Yucca.

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. While OSCON is a specialized 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 specialized representations for each function. Hence, UC must sacrifice some of the specialization built into OSCON.

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. 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 has not concerned itself with putting information from more than one operating system in their program. 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. UC has no demonstrated ability for answering complex natural language queries which involve command sequencing. UC and OSCON are not truly competitors, but rather systems with different emphases.

UCC is another natural language UNIX help system (see Douglass and Hegner, 1982) UCC was a prototype system, implemented in Franz Lisp on a VAX-11/780. It used a simple natural language front end based on augmented transition networks. The development of a knowledge base and query solver were not advanced enough so that they could be linked with the front end. UCC generated answers to queries directly from concept case frames rather than from any particular formal language. There were two major problems with UCC: (1) as the front end included a relatively simple knowledge base it was unable to answer sophisticated queries with many constraints involving command options, and (2) the simple augmented transition network method of parsing was not sufficient enough to handle the types of queries posed by many users. A program called Yucca (Hegner and Douglass, 1984) was an attempt to augment the UCC system in two ways. Yucca incorporated a much more sophisticated formal knowledge base and an improved natural language front end. OSCON's design is based very much on principles from UCC and Yucca.

The Sinix Consultant (SC) (see Hecking et al. 1988, Kemke 1986, and Kemke 1987) is a broad-based UNIX help system. It is similar to OSCON in that it is designed for the operating systems domain. 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 any interconnection of several. There appears to be no mechanism whereby the SC system will understand complex command sequences. However, it is predicted in Hecking et al. (1988)

that future development of the SINIX Knowledge Base will include "combinators" for I/O-redirection and pipelining.

There are several other consultation systems for operating systems, including CMS-HELP developed by Yun and Loeb (1984), TVX by Billmers and Carifio (1985), Wizard (see Shrager and Finin 1982, and Finin 1983), and USCSH by Matthews and Pharr (1987). Selfridge (1988) takes an expert system approach to knowledge representation. 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 simpler 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 for over 40 commands. 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 two most related systems available today (UC, SC) 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 answers queries involving command combinations.

Recently, some Wizard-of-Oz studies have been completed for the operating systems domain (see Mc Kevitt and Ogden 1989a). The Wizard-of-Oz technique is one where subjects interact with a computer through typed dialogue and are told that they are conversing with the computer. Subjects utterances are sent to another monitor where a "Wizard" sends back a reply to the subject monitor. Data were collected for 14 subjects who were naive UNIX users. Initial analyses of results look interesting. Seven types of queries have been discovered as opposed to the three types that were defined above a priori. Also, it has been noted that query type is strongly related to levels of user expertise. These data show that certain query types which were not decided a priori would not be handled by OSCON. Also, these query types would cause a major breakdown in communication in the natural language dialogue. Further analyses of this data will be done to help determine the relationship between dialogue models and user models.

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 the queries out in elaborate English. Initial thoughts on this research are reported in Ball et al. (1989).

12. Recent developments

There have been a number of developments with respect to the OSCON system since 1987. Analysis of Wizard-of-Oz data (see Mc Kevitt 1991, Mc Kevitt and Ogden 1989a,b) from natural language dialogues on UNIX gave nine query/intention[5] types (see Table I below). The example UNIX queries in Table I are shown as they were actually typed by users.

Our Wizard-of-Oz empirical data showed that expert subjects have mainly *information* and *description* intentions whereas less expert subjects have much more of the other intentions such as *elaboration*, *explanation* and *repetition*. Also, the data showed that there were statistically significant differences in the frequencies of intention types between expert and non-expert subjects. Based on this grouping of intentions, and the fact that some intention sequences indicate less satisfaction than others, we proposed a weighted formula for a measure of *satisfaction* as follows (where [X $\rightarrow$ Y] represents the frequency of intention pairs from intention X to intention Y; 'I' denotes information intentions, 'De' denotes description intentions, etc.):

$$\begin{aligned} 3([I \rightarrow I]) + [I \rightarrow De] + [I \rightarrow El] + [I \rightarrow Ex] + [I \rightarrow Re] \\ +[De \rightarrow I] + [El \rightarrow I] + [Ex \rightarrow I] + [Re \rightarrow I] \\ +3([De \rightarrow De]) + [De \rightarrow El] + [De \rightarrow Ex] + [De \rightarrow Re] \\ +[El \rightarrow De] + [Ex \rightarrow De] + [Re \rightarrow De] \end{aligned}$$

[5] We have reclassified query types into nine more general intention types.

Table I. Definitions of principal intentions for the consultancy domain

Intention	*General Definition*	*UNIX Domain Definition*
information	An intention requesting a PLAN to achieve a specific GOAL where the GOAL is described[6]. e.g. "How do I cook this dish?"	An intention requesting a UNIX command to achieve a UNIX operation where the operation is described. e.g. "How do I print a file?"
description	An intention requesting a description of an object or concept. e.g. "What is Persia?"	An intention requesting the description of UNIX concepts, objects, or commands. e.g. "what is UNIX?"
instruction	An intention acting as an instruction to achieve a GOAL rather than the PLAN to achieve that GOAL. e.g. "Can you find out how many foreign nationals now live in Kuwait?"	An intention requesting the execution of a UNIX command. e.g. "has oscon been printed?"
elaboration	An intention requesting more information on a PLAN or GOAL. e.g. "Could you tell me more about Iraq?" following "Where is Iraq?"	An intention requesting more information on UNIX commands, or UNIX itself. e.g. "how do i use more?" following "how do i see my file?"
confirmation	An intention requesting confirmation of a belief or some PLAN believed to execute some GOAL. e.g. "Will sanctions stop Saddam Hussain?"	An intention requesting confirmation of a belief about the function of commands in UNIX, or the function of UNIX itself. e.g. "can i remove a directory with files in it?"
explanation	An intention requesting explanation or clarification of an item which occurred during the execution of a PLAN for a GOAL. e.g. "Could you tell me what you mean by U.N. resolution 611?"	An intention requesting explanation of a response from the UNIX shell. e.g. "What does cp -r mean?"
guidance	An intention requesting a PLAN for a GOAL where there is no explicit GOAL expressed. e.g. "What do I do next?"	An intention requesting help with UNIX operations, or UNIX, where there is no operation described. e.g. "I don't understand what i'm supposed to do."
repetition	An intention which is a repeated request. e.g. "How many people live in the Gulf?" followed by "What number of people live in the Gulf?"	An intention repeating another intention. e.g. "How do I print a file" followed by "How do I get a print out of my file?"
nointention	An intention which is not immediately relevant to the domain or not understood by the hearer as being relevant to the domain. e.g. "Where does Strider live?" in the domain of Economics.	An intention not understood by the system. e.g. "How do I eat a file?"

Relative dissatisfaction can be calculated by summing dissatisfaction intention pairs. The formula for a measure of *dissatisfaction* is:

$$\begin{aligned} 3([El \to El] + [Ex \to Ex] + [Re \to Re]) \\ +2([Ex \to El] + [Re \to El] + [Re \to Ex] + [El \to Ex] \\ +[El \to Re] + [Ex \to Re]) \end{aligned}$$

Two new components have been added to OSCON to include the capability of dialogue modelling and user modelling: *DialCon* and *UCon*. *DialCon* is a dialogue modelling component which uses to track query/intention sequences in dialogue. It updates an intention matrix by locating the relevant cell in the matrix representing a given intention sequence and increasing its count. DialCon indexes the cell in the matrix by pairing the current query/intention type with the previous.

UCon is a user modelling component which computes the level of user satisfaction from the intention matrix and provides information for context sensitive and user sensitive natural language generation. *UCon* derives a binary measure of user expertise, *expert* and *novice*. UCon applies a user-modelling function, based on the weighted formulas above, to the intention matrix to determine levels of user *satisfaction* and *dissatisfaction*. Initially, the user is assumed to be an expert. Subsequent changes in the levels of satisfaction and dissatisfaction will result in changes in the level of user expertise. Such information is used by GenCon to generate context-sensitive and user-sensitive natural language responses. OSCON's capability of dialogue and user modelling is demonstrated in Appendix B.

Also, we have developed a version of OSCON which acts as a command interface where after a user asks a given query then the system asks if he/she would like the resultant UNIX command executed (see Lim 1994). If the user replies "y" then the results of the execution of the command are reported.

Acknowledgements

Acknowledgements are due to Louise Guthrie and Zhaoxin Pan for programming parts of OSCON. Stephen Hegner is thanked for many stimulating discussions. Yorick Wilks and the natural language processing team at the Computing Research Laboratory (CRL), New Mexico State University, USA are thanked for numerous comments on this work. Hans Brunner, Andy Parng and Scott Wolff of the Intelligent

[6] A *PLAN* is defined as a set of actions to achieve some GOAL. A *GOAL* is defined as an operation a speaker wishes to achieve.

Customer Assistance Project at U S *WEST* Advanced Technologies are thanked for consultations on this research. This research was funded in part by U S *WEST* Advanced Technologies, Denver, Colorado under their Sponsored Research Program. Peter Norvig, Wolfgang Wahlster, and Robert Wilensky are thanked for comments on an earlier draft of this paper and Stephen Hegner for comments on a later draft.

Notes

[0] This paper is effectively that already published as "Principles and practice in an operating system consultant", in "Artificial Intelligence and Software Engineering (Vol. 1)", Section on "AI Mechanisms and techniques in practical software", Derek Partridge (Ed.), 1991, 439-462. New York: Ablex Publishing Corporation. Copyright ©1991 Intellect Books (originally to Ablex Publishing Corporation).

Appendix A

This appendix contains a list of some of the queries which OSCON currently answers. The maximum time taken to answer any query is 2.5 seconds. The query sets have been divided into three major types (1) *request for information*, (2) *request for explanation*, and (3) *request for direction.*

```
| ?- oscon.
OSCON Program 1.0 (Sun-3, Sun Unix 4.2)
Copyright (C) 1988, Computing Research Laboratory.
All rights reserved.
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-information[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-information[form]

--> 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-explanation[concept]

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-direction

--> 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.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; request-for-direction

--> 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'

--> quit.
over
yes
| ?-
```

Appendix B

This Appendix demonstrates that the OSCON system modifys its natural language responses dynamically based on the types of utterances, and hence intention, that appear in the input.

```
| ?- oscon.
OSCON Program 1.0 (Sun-4, SunOS Release 4.1)
Copyright (C) 1988, Computing Research Laboratory. All rights reserved.
Dept. 3CRL, Box 30001, NMSU, Las Cruces, NM (505) 646-5466, USA.

U S WEST Advances Technology

Please input your question ending with with a '?'
Write 'quit.' when you are finished
OSCON can answer queries for UNIX or MSDOS.
Type one of these to the prompt.
Do you want answers for UNIX or MSDOS?
The default is UNIX [unix]: unix

How can I help you?

--> what does ls do?
```

```
'ls <directoryname>' will display directory contents on the screen.

information  : 0  0  0  0  0   0  =  0
description  : 0  0  0  0  0   0  =  0
elaboration  : 0  0  0  0  0   0  =  0         Satisfaction    = 0
explanation  : 0  0  0  0  0   0  =  0         Dissatisfaction = 0
repetition   : 0  0  0  0  0   0  =  0

nointention  : 0  0  0  0  0   0  =  0

                                     == 0
```

OSCON displays the 'intention matrix' and levels of 'satisfaction' and 'dissatisfaction'. Note that OSCON caters for 5 'real' types of intention, and nointentions. The 'intention matrix' is a 2-dimensional, 6 X 6 matrix. The 'intention matrix' shows all intention-sequence counts to be 0 as only one intention has been entered into the program. Totals for each intention type are also shown. 'Satisfaction' and 'dissatisfaction' levels, computed from the matrix, using the user modelling function, are also 0.

```
--> how do i see my file?

'more <filename>' will display file contents on the screen.

information  : 0  0  0  1  0   0  =  1
description  : 0  0  0  0  0   0  =  0
elaboration  : 0  0  0  0  0   0  =  0         Satisfaction    = 1
explanation  : 0  0  0  0  0   0  =  0         Dissatisfaction = 0
repetition   : 0  0  0  0  0   0  =  0

nointention  : 0  0  0  0  0   0  =  0

                                     == 1
```

This 'intention matrix' shows a total intention sequence count of 1, and the intention count for the sequence 'explanation $\rightarrow$ information' is set to 1. 'Satisfaction' also totals to 1.

```
--> how does he see my file?

'more <filename>' will display file contents on the screen.

information  : 0  0  0  1  0   0  =  1
description  : 0  0  0  0  0   0  =  0
elaboration  : 0  0  0  0  0   0  =  0         Satisfaction    = 2
explanation  : 0  0  0  0  0   0  =  0         Dissatisfaction = 0
repetition   : 1  0  0  0  0   0  =  1
```

```
nointention  : 0  0  0  0  0   0  =  0

                                  == 2
```

This 'intention matrix' shows a total sequence count of 2. Now, a 'information → repetition' sequence has occurred. Note that the syntax of this query does not have to be exactly the same as the previous query to denote a repetition. 'Satisfaction' increases to 2.

```
--> how do they see my file?

'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'.

information  : 0  0  0  1  0   0  =  1
description  : 0  0  0  0  0   0  =  0
elaboration  : 0  0  0  0  0   0  =  0        Satisfaction    = 2
explanation  : 0  0  0  0  0   0  =  0        Dissatisfaction = 3
repetition   : 1  0  0  0  1   0  =  2

nointention  : 0  0  0  0  0   0  =  0

                                  == 3
```

A 'repetition → repetition' intention loop has occurred tilting the level of 'dissatisfaction' higher than 'satisfaction'. As a reaction, more information is returned to the user. Remember that the user modelling function gives intention repetitions which are along diagonals a weight of 3.

References

Ball, Jerry, John A. Barnden, Sylvia Candelaria de Ram, David Farwell, Louise Guthrie, Cheng-Ming Guo, Stephen Helmreich, Paul Mc Kevitt & 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. In *Proceedings of the Second Conference on Artificial Intelligence Applications*, 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)*, Vol. 2, 756-760, St. Paul, Minnesota, August.

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

Finin, Timothy W. (1983). Providing help and advice in task oriented systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, 176-178, Karlsruhe, Germany.

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, Germany, August.

Hegner, Stephen J. (1988). Representation of command language behavior for an operating system consultation facility. In *Proc. of the Fourth IEEE Conference on Artificial Intelligence Applications*, 50-55, San Diego, CA: IEEE Computer Society.

Hegner, Stephen J. (1998). Representations of command language behavior for an operating system expert consultation facility. In Stephen Hegner, Paul Mc Kevitt, Peter Norvig, and Robert Wilensky (Eds.), *Intelligent help systems for UNIX.* Dordrecht, The Netherlands: Kluwer Academic Publishers (this volume).

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/SCEIO)*, 159-161, London, Ontario, May.

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 H.J. Bullinger and B. Shakel (Eds.), *Human-Computer Interaction - INTERACT '87,*. 215-220. Amsterdam: Elsevier Science Publications B.V. (North-Holland).

Lim, Hun Keong (1994). *Translation of OSCON into a command interface.* Master's Thesis, Department of Computer Science, University of Sheffield, Sheffield, England.

Matthews, Manton & Walter Pharr (1987). Knowledge acquisition for active assistance. In *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, US.

Mc Kevitt, Paul (1987). *Artificial Communicators: An operating system consultant.* Master's Thesis, Computer Science Department, Dept. 3CU, Box 30001, New Mexico State University, Las Cruces, New Mexico 88003-0001, US.

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, Ireland, September.

Mc Kevitt, Paul (1991). *Analysing coherence of intention in natural language dialogue.* Ph.D. Thesis, Department of Computer Science, University of Exeter, GB- EX4 4PT, England.

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.

Mc Kevitt, Paul & Zhaoxin Pan (1989). A general effect representation for Operating System Commands. In *Proc. of the Second Irish National Conference on Artificial Intelligence and Cognitive Science (AI/CS-89)*, pp. 50-65, School of Computer Applications, Dublin City University, Dublin, Ireland, September. Also in, Alan Smeaton and Gabriel McDermott (Eds.), *AI and Cognitive Science '89*, Springer-Verlag British Computer Society Workshop Series, 68-85. Heidelberg, Germany: Springer-Verlag.

Mc Kevitt, Paul & William C. Ogden (1989a). *Wizard-of-Oz dialogues for help on operating systems.* Memoranda in Computer and Cognitive Science, MCCS-89-167, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001, US.

Mc Kevitt, Paul & William C. Ogden (1989b). *OSWIZ II: Wizard-of-Oz dialogues in the computer operating systems domain.* Memorandum in Computer and Cognitive Science, MCCS-90-181, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001, US.

Selfridge, Peter G. (1988). How to print a file: An expert system approach to software knowledge representation. In *Proceedings of the Seventh National American Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, 380-385, St. Paul, Minnesota, August.

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

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

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

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

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