Towards an adaptive natural language interface to command languages

S. E. MICHOS, N. FAKOTAKIS AND G. KOKKINAKIS
University of Patras, Greece

(*Received 26 January 1996; revised: 29 July 1996*)

Abstract

Operating system command languages assist the user in executing commands for a significant number of common everyday tasks. On the other hand, the introduction of textual command languages for robots has provided the opportunity to perform some important functions that leadthrough programming cannot readily accomplish. However, such command languages assume the user to be expert enough to carry out a specific task in these application domains. On the contrary, a natural language interface to such command languages, apart from being able to be integrated into a future speech interface, can facilitate and broaden the use of these command languages to a larger audience. In this paper, advanced techniques are presented for an adaptive natural language interface that can (a) be portable to a large range of command languages, (b) handle even complex commands thanks to an embedded linguistic parser, and (c) be expandable and customizable by providing the casual user with the opportunity to specify some types of new words as well as the system developer with the ability to introduce new tasks in these application domains. Finally, to demonstrate the above techniques in practice, an example of their application to a Greek natural language interface to the MS-DOS operating system is given.

1 Introduction

It is not quite true that Natural Language Interfaces (NLI) are always easier to learn than formal command languages, but they are sometimes easier to use (Shwartz, 1987). While commercially available NLIs traditionally have been used as a front-end for database management systems (Waltz, 1975; Hendrix *et al.*, 1978), recent NLI products focus on interfacing to Operating Systems (OS) and Robot Languages (RL).

Many efforts have been made in the past towards creating flexible NLIs to popular OSs. A recent potential success story for MS-DOS is DOSTALK (Obermeier, 1989), a LISP-based program. Though endowed with several value-added capabilities, it provides no knowledge-based expandability mechanisms. The UC (Wilensky *et al.*, 1984, 1988) developed at Berkeley, helps users learn correct UNIX commands through English dialogue. It is a pragmatic analysis system[1] with advanced and

[1] These systems attempt to derive the deeper meaning/implications of natural language sentences by performing inference on pragmatic discourse elements, such as the goals of dialogue participants, social protocols associated with a given situation, etc.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

sophisticated features, and its only disadvantage seems to be the absence of the facility to execute UNIX commands to UC. NALIGE (Manaris and Dominick, 1993) is the most up-to-date significant effort for the development of a user interface management system which can generate a NLI to an OS by simply accepting a set of well-formed specifications. Though it is a system allowing a user to custom-tailor the system's linguistic capabilities, knowledge-based expandability mechanisms are restricted to the definition of lexical synonyms and syntactic paraphrases by the end user. Finally, FRIENDOS (Michos and Fakotakis, 1994), a Greek NLI to MS-DOS that translates users' Greek expressions into MS-DOS commands, in its first version provided the user with suitable expandability mechanisms to cope with unknown words, as well as satisfactory error-recovery mechanisms to deal with ill-formed input user messages. However, it could not handle complex Natural Language Expressions (NLE) and define new task domain concepts.

On the other hand, there have been several attempts in artificial intelligence to develop a NLI for robots (Huffman and Laird, 1993; Winograd, 1972; Barbenhenn *et al*., 1994). The crucial disadvantage in these systems is that they translate the natural language commands into a sequence of primitive robot actions, but fail to generate motions that can be executed in real world. This is because real-world constraints like collision avoidance need geometric analyses in continuous domains, which have high complexity to be adequately tackled by such systems. A NLI with a limited vocabulary has been developed for an electronic circuit simulator to specify output data (Samad, 1986). However, it does not learn. A learning user interface for commanding robot motions in a natural-language like form is presented in Chen and Hwang (1994). Although it has the ability to learn how to execute a motion command from the demonstration given by the user, it employs a limited vocabulary and cannot handle combinations of commands.

As most command languages comply with a generally adopted command-organization strategy, modelling this strategy for a NLI to such a language would offer a high portability across similar command language systems and applications. Section 2 describes how this goal is accomplished by means of suitable and flexible knowledge representation techniques. In section 3 linguistic parsing techniques are presented to assist in handling simple commands, as well as combinations of commands, that are given through complex NLEs. On the other hand, in spite of the fact that the subset of a natural language used for communication with a command language is quite restricted, it is very difficult to predict in advance all linguistic structures and lexical entities that may be useful for the NLI. Furthermore, the necessity for treatment of new task domain concepts in command languages which are continually updated with new facilities requires the provision of sufficient mechanisms to cope with these needs. For this reason, machine learning techniques are given in section 4. The application of the above techniques to FRIENDOS (Michos and Fakotakis, 1994) is given in section 5. This interface in its latest version addresses precisely these needs, and demonstrates how the adoption of the above techniques could considerably improve the performance of similar NLIs to popular command languages. Finally, some conclusions and directions for future work are discussed in section 6.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

2 Knowledge representation techniques

To ensure a satisfying organization of both the command language grammar and the lexicon of terms, we adopted the following scheme for the tasks supported by a NLI like the above described: we defined some basic Task Domain Objects (TDO) as well as some Task Domain Actions (TDA) related with these objects. To be more specific, the objects are those appearing in commands (e.g. *file*, *directory*, *location*, *box*, etc.) and the actions refer to common operations (e.g. *deleting a file*, *making a directory*, *approaching a location*, *moving a box*, etc.). This scheme offers an easy way to classify things in such an application domain, especially when a new object or action becomes available either with an updated version of a known command language, or with the first version of an entirely new one.

On the other hand, a conceptual representation for TDOs is required to distinguish conceptually similar but practically distinct TDOs, as well as to classify new TDOs in such application domains. Common sense and real world practice shows that a satisfying representation should comprise TDOs that are preceded by a descriptor indicating the specific role they play in a given situation, as follows:

descriptor_TDO

For instance, in the case of approaching a location where it is necessary to determine the source and destination locations, these two TDOs will be internally represented as *s_location* and *d_location* (s = source, d = destination).

2.1 Commands codification

Several strategies for command organization have emerged, but guidelines for choosing among these are only recently beginning to be discussed (Schneiderman, 1992). However, some general criteria have been adopted and are presented below:

(a) Each command is chosen to carry out a single task, so that the number of commands matches the number of tasks.
(b) Each command is followed by one or more arguments that indicate objects to be manipulated.
(c) Commands may have options to indicate special cases; the arguments may also have options.
(d) The full set of commands is organized into a tree structure, like a menu tree where the first level might be the command action, the second might be an object argument, and the third might be a destination argument.

Although a major part of the tasks performed by mainstream OS environments are conceptually equivalent (i.e. a large subset of their semantic domains coincide), the corresponding command languages are, in most cases, diametrically different in terms of linguistic characteristics. For instance, consider the conceptual action of requesting a five-copy printout of a specific file. Figure 1 presents the corresponding realizations of this conceptual action in the UNIX (UNIX, 1986), VAX/VMS (DEC, 1988) and

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

UNIX:	lpr - #5 steve.doc
VAX/VMS:	print /copies = 5 steve.doc
MS-DOS:	print steve.doc /c:5

Fig. 1. Various realizations of the PRINT command.

VAL:	appro p1, 50
AML/2:	pmove (p1 + 50)
VAL II:	approach p1, 50 MM

Fig. 2. Various realizations of the APPROACH command.

MS-DOS (Close, 1989) command languages. Consequently, a user may know the semantics of a task to be performed, but may lack the experience to express it in the appropriate OS command language.

The same is true for the great majority of RLs. For instance, consider the conceptual action of approaching a location within a safe distance of 50 mm. Figure 2 presents the corresponding realizations of this conceptual action in the VAL (Asfahl, 1991), AML/2 (Asfahl 1991) and VAL II (Groover, 1992) command languages.

Hence, the problem is that of investigating the possibility of conceptual modelling of similar mainstream command languages. After a thorough investigation of their semantic domains, the following observations have been made:

(i) No matter what application we have (i.e. an OS, a robot gripper, an Internet navigator, etc.), the command language syntax consists of an *identifier* specifying the name of the command, one or two sets of *compulsory arguments*, one set of *optional arguments*, and a set of *options*. For example, for the previous command languages, we have found that:
   – *lpr*, *print*, *appro*, *pmove*, and *approach* are the identifiers;
   – *steve.doc* and *p1* are one set of compulsory arguments;
   – *50* and *50 MM* are a second set of compulsory arguments;
   – -#5, */copies = 5*, and */c:5* are options.

(ii) What differentiates one command language from another, except for some commands, is the order in which the identifier, the corresponding arguments and the options appear in the command string. Hence, a conceptual representation could be as follows:

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

{<IDENTIFIER><CAS1><OAS><CAS2><OPS>}

where <IDENTIFIER> = identifier, <CAS1> = first set of compulsory arguments, <OAS> = set of optional arguments, <CAS2> = second set of compulsory arguments and <OPS> = set of options. The only thing we are looking for now to render such a NLI as portable as possible to similar command languages is a flexible way to represent the order of appearance of all these parameters.

The above observations have led us to adopt the following knowledge representation scheme:

<IDENTIFIER><CAS1LIST><OASLIST><CAS2LIST><OPSLIST><PLIST>

The new item is <PLIST>, which is a priority list. Thanks to this last list, we are able to determine for such command languages as well as for future ones with similar structure the right order of appearance of the above parameters in the command string. So, the knowledge structure that captures these findings could be as follows[2]:

application_pattern(' <IDENTIFIER> ',
[<*CAS1LIST*>],
[<*OASLIST*>],
[<*CAS2LIST*>],
[<*OPSLIST*>],
[<*PLIST*>])

Before closing this section, two things need to be mentioned: (a) all the lists in the *application_pattern* knowledge structure, except the OPSLIST, should be composed of TDOs in the form of *descriptor_TDO*; and (b) the selection of these TDOs should depend upon the syntax of the command in the specific command language. Consequently, the importance of this structure is twofold:

(a) During analysis of a NLE, it helps us locate the potential compulsory and optional arguments, as well as the expected options, for a recognized command identifier.
(b) During generation of the command in the specific command language, it assists us in ordering correctly all the above parameters in the command string, regardless of their order of appearance in the NLE.

2.2 Commands identifier recognition

One of the important problems in designing a NLI to a specific command language is the recognition of the correct command identifier. It is well known that the same NLEs we use to execute a TDA in a specific command language often need to be

[2] All structures are given in Prolog notation.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

mapped to different command identifiers. Things get harder when elliptical expressions are given by the user, which may lead to more than one acceptable commands. For instance, the words 'Display' and 'contents' could equally be used in the following NLEs, which are mapped to different MS-DOS commands:

Regular NLEs

- '*Display* the *contents* of the directory GAMES.' <—> *dir*
- 'I would like you to *display* me the *contents* of the directory GAMES.' <—> *dir*
- Could you please *display* me the *contents* of the memory?' <—> *mem*
- *Display* the time.' <—> *time*
- *Display* the date.' <—> *date*
- *Display* the version.' <—> *ver*

Elliptical NLEs

- '*Display* me the *contents*.' <—> *dir* or *mem*
- '*contents*.' <–> *dir* or *del* or *print* or *copy*

To cope with this problem, we have adopted the following methodology:

After determining the exact set of TDOs and TDAs that will be supported in a selected application domain as well as the representative set of NLEs with which these TDOs and TDAs can be described, we define ***knowledge structures*** *for both the verbs describing TDAs and the nouns describing TDOs which comprise all the possible command identifiers in which they may appear.*

Such a knowledge structure will have the following form:

bkey(*WORD*,[*COMMAND_IDENTIFIER_1*,...,*COMMAND_IDENTIFIER_N*])

So, for example, the knowledge structure for the word 'Display' in the MS-DOS command language application domain could be as follows:

bkey('*Display*',['*dir*','*cd*','*date*','*doskey*','*mem*','*time*','*tree*','ver','vol'])

The advantages of such a structure are many:

(i) This structure is flexible and easily expandable in the future.
(ii) It offers memory economy, since otherwise nine similar structures should be defined.
(iii) Ellipsis phenomena can now be easily handled, since there is a set of possibly intended commands directly related to some words.

Additionally, if we employ similar structures for all the other critical nouns which are likely to appear in the NLE given by the user, we are able to extract the corresponding command identifier, or a small set of command identifiers in case of elliptical expressions, by means of an intersection algorithm. For instance, if we have the following NLE in the MS-DOS command language application domain

'*Display the contents of the directory GAMES.*'

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

the knowledge structures for the critical words of this expression are[3]

bkey('*Display*',['*dir*','*cd*','*date*','*doskey*','*mem*','*time*','*tree*','*ver*','*vol*'])
bakey('*contents*',['*dir*','*mem*','*del*'])
bgkey('*directory*',['*dir*','*chkdsk*','*cd*','*copy*','*del*','*print*','*rd*','*tree*','*type*', '*xcopy*'])

and the output of the intersection algorithm will give 'dir' for the expected command identifier.

Two things should be cleared here. First, it is our convention to utilize capital letters for the names of files, directories, drives, etc., to distinguish them from the other words of the NLE. Second, when elliptical NLEs have been used and the output of the intersection algorithm gives more than one command identifiers, the system developer is responsible for deciding on what the right course of actions will be for the system to clarify the user message (e.g. it will ask the user to select the intended action from a helpful menu, it will use default knowledge to select automatically a concrete command, etc.).

3 Linguistic parsing techniques

Traditionally, many NLIs that were developed in such application domains aimed at two targets:

(a) transforming some possible NLEs into some internal representation by usually relating each *phrase structure* or *semantic* or *augmented semantic* grammar rule directly to a NLE (Manaris and Dominick, 1993); and
(b) allowing for only stereotyped NLEs corresponding most of the time to single commands.

As a consequence, the interfaces developed, although they manage to accomplish their initial goal quite satisfactorily, could not always tackle the unpredictability of the natural language successfully. Furthermore, the user was usually allowed to type the command only in imperative mood, and could not give combinations of commands.

In this section, special emphasis is put on (a) the significant reduction of the number of grammar rules needed to handle the possible NLEs given by the user, and (b) the utilization of an advanced linguistic parser that can cope with both simple and complex NLEs.

3.1 Determining the NLEs

The most challenging part in developing a NLI is identification of the natural language subset to be modeled. Past work suggests that following an empirical approach, such as surveys of actual human usage, may be the most effective way to discover the NLEs likely to be input by a user (Furnas *et al.*, 1987; Keller, 1984). We conducted a survey of the target user population, namely a cross-section of

[3] The *bakey* and *bgkey* structures are used to distinguish nouns in *a*ccusative and *g*enitive cases, respectively. We need these two distinct structures since the Greek language is a highly inflected one.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

undergraduate and postgraduate students who had attended computer science courses in the past. The results of this survey have led us to the following conclusions:

- users with sufficient proficiency in command languages find it convenient to express a NLE usually in imperative mood, while less experienced users seem to prefer positive and interrogative NLEs;
- although the command languages themselves do not allow the typing of complex commands (i.e. combination of at least two different commands) – except for the use of batch procedures – many users would like to have a NLI providing them with this capability, especially when this interface could be integrated into a corresponding speech one;
- finally, no matter how fluent in one or more command languages a user is, (s)he may use elliptical, semantically or pragmatically wrong NLEs[4] due to carelessness, briefness or even actual lack of computer literacy.

Taking into consideration the above conclusions, both objective and subjective criteria should be used in determining the NLEs that would be handled by the interface. These criteria could be summarized as follows:

(i) a NLE should be selected for inclusion if at least half the participants of similar surveys supply it;
(ii) a NLE should describe one or more given TDAs as well as possible in terms of the command language syntax consistency and natural language brevity; and
(iii) a NLE is considered to be different from another NLE only in semantic content (i.e. regardless of the sentence mood and voice and the order of the constituent words of these sentences).

3.2 Creating the semantic grammar

It is generally accepted that semantic-grammar based formalisms are quite useful in the development of robust natural language processing applications, especially in situations where the semantic and linguistic domains are tangible (Allen, 1987). Semantic grammars were invented by Burton (Burton, 1976; Burton and Brown, 1979) for use in the SOPHIE natural language understanding system. These grammars are equivalent in descriptive power to context-free grammars. The only difference is that they incorporate both syntactic and semantic aspects of natural language understanding knowledge into a single framework.

Current augmented semantic grammar formalisms model the syntactic/ semantic structure of the natural language subset to be 'understood' by the NLI in terms of production rules (Manaris *et al*., 1994). For each rule, they identify a semantic action which derives the meaning of corresponding NLEs in terms of the semantic domain specification components. Although the above formalisms are considered by many researchers as appropriate for linguistic parsing in such limited application domains,

[4] Semantically incorrect messages are as follows: 'Delete the drive A of the file STE.TXT.', while pragmatically wrong messages are as follows: 'Check the drive F.' (F is a network drive that cannot be checked.)

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

they sometimes prove to be inefficient due to the freedom of natural-language expressive style. Besides, a dramatic and perhaps prohibitive increase in the number of production rules required to describe complex linguistic phenomena – such as *complex sentences*, *sentence mood* and *voice analysis*, *anaphora resolution*, and *free word order* – that may be encountered in a given NLE, is inevitable in both formalisms.

To deal efficiently with this problem, we proceed as follows:

> No matter what the surface structure of a NLE may be, semantically equivalent NLEs are mapped to the same internal user message (IUM)[5] with the aid of a suitable linguistic submodule. Then, as soon as the command identifier of the original NLE has been recognized, the corresponding *application_pattern* helps us locate the potential compulsory and optional arguments as well as the expected options for the recognized command. Thus, the production rules are now used to map possible natural language words of the IUM to appropriate *descriptor_TDOs*.

Following this scheme, all the subsequent NLEs (simple or complex) – except the second clause of the fourth:

Simple	'*Display the files of the directory GAMES.*' (dir)
Simple	'*I would like you to display me the files of the directory GAMES.*' (dir)
Simple	'*Could you please display me the files of the directory GAMES*?' (dir)
Complex	'*Display the files of the directory GAMES and then print them.*' (dir,print)
Simple	'*To be displayed the files of the directory GAMES.*' (dir)
Simple	'*The files of the directory GAMES to be displayed.*' (dir)
Simple	'*Show the contents of the list GAMES.*' (dir)

should be mapped to the same IUM

'*Display the files of the directory GAMES.*'

In doing so, synonymous words and syntactic paraphrases of the NLEs do not require additional production rules. Therefore, a significant decrease in the number of production rules is now achieved that allows the treatment of even complex linguistic phenomena in such application domains.

It should be made clear that the mapping from NLEs to IUMs, as well as from IUMs to command language, has been based on the following features: *pattern-matching*, *unification* and *list processing*. These features are also characteristics of the programming language we used in this work (i.e. Prolog).

3.3 Embedding an enhanced linguistic parser

With a view to assisting the treatment of simple commands as well as combinations of commands that are given through complex NLEs, an enhanced linguistic parser is necessary. By endowing a NLI with such a parser, complex linguistic phenomena can now be addressed. Furthermore, apart from being able to be integrated into a future

[5] The IUMs are distinct canonical NLEs that are derived from the survey that was presented earlier.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

speech interface, this interface can facilitate and broaden the use of command languages to a large audience.

A parser that meets the above requirements is described in detail in Michos *et al.* (1994, 1995). This parser allows for:

- the analysis of both simple and complex sentences;
- the treatment of sentence mood and voice;
- the resolution of some anaphoric pronouns; and
- the handling of even quasi free word order languages[6].

Specifically, this parser is capable of parsing long sentences in quasi-free word order languages. It operates in two stages, each of which is carried out by the same parsing engine. In the first stage, a NLE is split up into its constituent clauses. The type of these clauses is then marked. In the second stage, phrase identification is carried out for each of the above constituent clauses. However, this parser cannot yet handle long compound nouns and some appositional structures.

Hence, instead of writing additional production rules of doubtful value and generality to tackle as many NLEs as possible, we employ a robust and efficient parser to recognize a specific TDA or a valid combination of more than two TDAs in an application context, regardless of the word order and syntactic structure of the given NLE.

4 Machine learning techniques

In a conventional NLI the system's linguistic knowledge base must be prepared in advance by a human expert. This requirement results in two qualities of the linguistic knowledge base: (a) it must be complete before the system starts working; and (b) it has a human-oriented structure, imposed by the expert's way of thinking about the language.

For a NLI to be self-expandable, its knowledge base components should be augmented during its interaction with the user with facts relative to the language being learnt. The adopted learning paradigm that is presented in this paper is a modified version of the *Symbolic Concept Acquisition* (Schalkoff, 1990; Shapiro, 1987), which we decided to call *User-Assisted Symbolic Concept Acquisition*. The former paradigm proceeds by modifying previously stored representations of concepts or constructing entirely new ones when presented with unknown NLEs. Although this task proceeds quite well, it is a fully automated procedure and thus liable to common mistakes when morphological or semantic problems arise during learning. On the contrary, though the latter paradigm employs its own means to guide learning, it allows the user (casual or system developer) to take an active part in the learning process by providing useful information, especially when difficult cases occur during the learning process. In this respect, we manage to restrict the eventual mistakenly updated knowledge as much as possible.

This is a considerable improvement, since the only way to correct possible mistakes in a self-expandable system is to give the new information again, fine-tuned and

[6] Such languages are characterized by a fixed intraphrasal word order and a free interphrasal one. The Greek language belongs to this category of languages.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

corrected without avoiding possible inconsistencies with previously acquired information. Moreover, the latter paradigm is very flexible, as it makes no distinction between the casual user and system developer, by helping the former provide general information without knowing the internal structure of the system knowledge base components, and by offering the latter the ability to provide specific information without requiring the rewriting of some of the system's source code procedures.

The NLI learning scheme could be carried out by the following modules: (a) the *lexical checker*, (b) the *morphological analyser*, and (c) the *learner*. It should include learning processes such as learning vocabulary, learning new TDAs relating to existing TDOs, and probably defining new TDOs. By supplying such mechanisms to cope with these matters, the treatment of new task domain concepts in command languages which are continually updated with new facilities is now feasible and easily done.

5 A clarifying example

To demonstrate how the adoption of the above techniques could considerably improve the performance of NLIs to popular command languages, we applied the described methodology to FRIENDOS (Michos and Fakotakis, 1994). FRIENDOS, which initially translates NLEs given by the user into MS-DOS commands and then executes them in real-time, is briefly described in this section. A detailed presentation of FRIENDOS, which also helps novice users master MS-DOS, is given in Magoulas and Fakotakis (1995) and Michos and Magoulas (1995).

However, it should be stressed here that the aim of this work is to illustrate the above theoretical techniques in practice, and not to supply a detailed step-by-step analysis of FRIENDOS' operation.

5.1 TDAs and TDOs

The task domain knowledge comprises the following elements:

(a) There are 23 initially supported TDAs: *cd*, *cdr*, *cls*, *chkdsk*, *copy*, *date*, *del*, *dir*, *diskcopy*, *doskey*, *format*, *label*, *md*, *mem*, *print*, *rd*, *ren*, *time*, *tree*, *type*, *ver*, *vol*, *xcopy*;

(b) There are 18 defined TDOs. They include: *content*, *file*, *drive*, *directory*, *path*, *screen*, *date*, *doskey*, *commands*, *label*, *memory*, *status*, *classification*, *printer*, *queue*, *time*, *structure*, *version*;

(c) Finally, there are 11 descriptor_TDOs: *s_file*, *s_path*, *s_drive*, *d_file*, *d_path*, *d_drive*, *d_date*, *d_time*, *s_label*, *s_doskey*, *s_coms* (s = source, d = destination).

5.2 System knowledge base

Due to the different levels of knowledge to which such an interface has access, it appears natural to conceptualize a system architecture which reflects this stratification. The FRIENDOS system environment consists of five knowledge base components:

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

- GWORD.DBA: this database comprises some general-purpose words (e.g. verbs, articles, etc.) as well as some morphological information.
- AWORD.DBA: in this database we have stored some application-specific words (e.g. nouns, adjectives, etc.) that are closely related to the language of the OS.
- SYNONYM.DBA: contains all the critical words that correspond to previously defined IUMs together with lists which are updated with synonymous words that may appear during the interaction of the NLI with the end user.
- NLTODQL.DBA: includes the direct mappings of some natural language words to MS-DOS final queries, one-to-one and one-to-many mappings of some TDOs and TDAs to MS-DOS commands, and finally, some other knowledge structures relating natural words to TDOs.
- DOSLAB.DBA: this is composed of the *application_pattern* knowledge structures of all the MS-DOS commands currently available in the NLI.

5.3 Linguistic parsing modules

Additionally, the FRIENDOS system environment can be subdivided into three major system modules, namely:

(a) TOKENIZER: splits the NLE into a list of tokens.
(b) GENERAL PARSER: is in charge of a number of tasks. For this reason, it consists of a number of submodules, namely:
  – *Special words handler*: finds all the special words (i.e. filenames and extensions, drive and directory names, numbers, etc.) contained in the NLE and stores them in appropriate structures;
  – *Lexical checker*: looks for unknown words.
  – *Morphological analyser*: when an unknown word has been located by the lexical checker, this submodule undertakes to find out the morphological category of the unknown word.
  – *Learner*: learns synonymous words and new TDAs relating to existing TDOs.
  – *Spelling checker*: this module corrects wrong article-noun agreement due to typographic errors.
  – *Linguistic parser*: can handle even quasi-free word-order languages. Among other tasks, it divides complex sentences into simple ones, it handles sentence mood and voice, and it disambiguates some anaphoric pronouns.
  – *Command's identifier recognizer*: recognizes the command identifier(s) of a simple or complex NLE given by the user with the help of the intersection algorithm described in section 2.2.
  – *Many-to-one transformer*: irrespective of the typed NLE, this module is responsible for converting all semantically equivalent NLEs to a corresponding IUM.
  – *Context checker*: analyses the IUM looking for possible semantic and pragmatic inconsistencies. It can also handle elliptical messages and

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

messages with redundant information. Finally, it collects all appropriate OS-oriented information from the IUM by means of the set of semantic grammar rules.

(c) CODE GENERATOR: generates the appropriate MS-DOS command or combination of MS-DOS commands, and may warn the user of the intended action(s) before invoking the MS-DOS system in order to execute it.

Finally, there are no centralized control modules for knowledge-base management and error handling. Instead, these procedures are incorporated in any one of the above modules supplying a form of distributed control.

5.4 Semantic grammar rules

As previously mentioned, as soon as the command identifier of the NLE has been recognized, the production rules are used to locate possible natural language words that correspond to appropriate *descriptor_TDOs* of the *application_pattern* of the recognized command. This process is being applied to the IUMs to which the original NLEs have been converted.

The implemented semantic grammar exhibits two important qualities:

(i) We have written rules that refer to all available TDOs. In this sense, an unknown TDA that relates to an existing TDO is simple for the system to learn, as it only has to store some facts in the appropriate databases.

(ii) We endowed the system with the capability of automatically cutting out all unnecessary information from a NLE. In this sense, we have managed to limit the number of available rules for any TDO. For instance, if the original NLE was

'*I would like you to display me the files of the directory GAMES.*'

the words 'I', 'would', 'like', 'you', 'to' and 'me' would be considered as redundant, and would be deleted before conversion of this NLE to an IUM.

A small portion of the set of available rules concerning the identification of *directories* in an IUM is shown in Figure 3 (the rules are given in Prolog code). These two rules

```
loc_cp1([H1,H2,H3|T],H,T) :- H="s_path",gword(H1,"arthro","or",_,_,"ait",_,_),
        conv_ait_to_onom(H2,O),commap(O,"path"),isa_sw(H3),
        sword(H3),asserta(reg_cwl("s_path",H3)),!.
loc_cp1([H1,H2,H3|T],H,T) :- H="s_path",gword(H1,"arthro","aor",_,_,"ait",_,_),
        conv_ait_to_onom(H2,O),commap(O,"path"),isa_sw(H3),
        sword(H3),asserta(reg_cwl("s_path",H3)),!.
```

Fig. 3. Production rules concerning directories.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

are concerned with the identification of an article, definite or indefinite, followed by a noun that is a valid word for directories in an IUM.

5.5 Learning processes

FRIENDOS learning processes are carried out by the following submodules of the General Parser module: lexical checker, morphological analyser, and learner. This interface is capable of learning both vocabulary and new TDAs relating to existing TDOs. In what follows, we give a brief description of how these claims are accomplished during the interaction of the NLI with the casual user or the system developer, respectively.

5.5.1 Learning vocabulary

As soon as the lexical checker finds an unknown word in the NLE, it presents a warning box to the casual user, as illustrated in Figure 4. If the user selects the second choice, the morphological analyser begins looking for the morphological category of this word. When it manages to recognize it, it presents to the user a complete list of all the known words belonging to the morphological category of the unknown word, and asks the user to determine which of these words has the meaning closest to the unknown word.

If the user does select a word, the learner is responsible for updating the appropriate knowledge-base components. Hence, according to the morphological category of the unknown word, the updating process takes place as follows (again, all knowledge structures are given in Prolog notation):

A. VERBS
In this case, the verb is being written in the following knowledge structure:

ngword(*NEWWORD*,'*verb*',*SYNI*,*SYNI*,*SYNI*,*SYNI*,*PRAI*,*SEMI*)

together with other syntactic (SYNI), semantic (SEMI) and pragmatic (PRAI) information concerning the specific OS. All this information is automatically transferred to this knowledge structure from the corresponding knowledge structure of the known word (i.e. the *gword*). This structure is then stored in the GWORD.DBA. Additionally, the verb is being written in a knowledge structure with all verbs of the word previously chosen by the user being synonymous. This structure looks like

vsynonyms(*CHOSENWORD*,[*VERBLIST*])

and it is stored in the SYNONYM.DBA.

B. NOUNS-ADJECTIVES
In this case, the noun or adjective is written in the following knowledge structure:

bnaword(*NEWWORD*,'*noun or adjective*',*SYNI*,*SYNI*,*SYNI*,_,_)

together with other syntactic information regarding the gender, number and case of this word. Again, this information is automatically transferred to this knowledge

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

structure from the corresponding knowledge structure of the known word. Other structures like

conv_ait_to_onom(*ACCUSATIVE*,*NOMINATIVE*)

conv_gen_to_onom(*GENITIVE*,*NOMINATIVE*)

are used to convert the accusative or genitive case of the unknown word into the nominative case. This is due to the intrinsic inflectional richness of the Greek language. All these structures are then stored in the AWORD.DBA. Additionally, the noun or adjective is written in a knowledge structure with all nouns or adjectives, respectively, of the word previously chosen by the user synonymous. These structures look like:

nsynonyms(*CHOSENWORD*,[*NOUNLIST*]) or

asynonyms(*CHOSENWORD*,[*ADJECTIVELIST*])

and they are stored in the SYNONYM.DBA.

5.5.2 Learning new TDAs relating to existing TDOs

If there are no known verbs semantically equivalent with the unknown verb, the learner is entangled in a simple interaction with the system developer in order to clarify a possible new TDA relating to an existing TDO, as well as to store all suitable information in the corresponding knowledge-base components. To be more concrete, the information that needs to be acquired from the system developer is as follows:

- some morphological information;
- the command identifier of the new TDA;
- what are the possible descriptor_TDOs of this new TDA, which one of them is compulsory and which one is optional and where (i.e. in what lists) it should be stored; and
- what will be the order of these descriptor_TDOs in the <PLIST> of the new application-pattern that will be created

As an example, let us assume that, up to now, FRIENDOS knows nothing about *running a file*. If the NLE is

'*Run the file WORD*.'

the following knowledge structures are stored automatically in the appropriate FRIENDOS knowledge-base components after completion of the interaction with the system developer:

ngword('*Run*','*verb*',*SYNI*,*SYNI*,*SYNI*,*SYNI*,*PRAI*,*SEMI*) → *GWORD.DBA*
morpho('*Run*','*Subordinate-Present-Singular-Second person*') → *GWORD.DBA*
bkey('*Run*',['*run*']) → *NLTODQL.DBA*
bakey('*file*',['*run*','*dir*','*chkdsk*','*copy*','*del*','*print*','*ren*','*type*']) →
NLTODQL.DBA
application-pattern('*run*',
['*CAL1*','*s_file*'],
['*OAL*'],

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

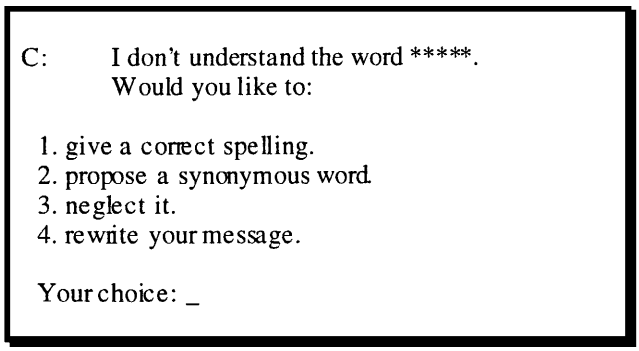


Fig. 4. Warning box for an unknown word.

['*CAL2*'],
['*OL*'],
['*PL*','*identifier*','*s_file*','*options*']) → *DOSLAB.DBA*
noid('*run*') → DOSLAB.DBA

In doing so, this NLI exhibits the following feature: it is able to learn new TDAs from examples given by the system developer in real-time. This on-line learning capability is very important for two reasons: (a) a quite small-sized knowledge base is required before the NLI starts working in real-time; and (b) any new facilities of the specific OS which become available with a new updated version can now be learnt on-line by the NLI without any previous training from the system developer.

5.6 Performance

FRIENDOS has been developed in the Prolog language and runs on 80486DX (and higher) Intel machines. It requires a small amount of memory ($\approx$ 300 Kb RAM) and has a real-time response[7] for all the MS-DOS commands supported. Although it has been implemented for the Greek language, it could equally well be used for other inflectionally rich or languages.

It employs 130 production rules. Before adoption of the techniques presented, FRIENDOS employed 860 rules. Hence, the decrease in the number of production rules required to map possible natural language words of the IUM to appropriate *descriptor_TDOs* is in the order of 85% (i.e. 1 out of 7). On the other hand, the system knowledge base initially included about 500 distinct words (i.e. general-purpose words, application-specific words, TDOs and TDAs). After application of the machine learning capabilities, this knowledge base was augmented to 1500 words.

The NLI presented is endowed with satisfactory error-recovery mechanisms to cope efficiently with problematic IUMs. Such messages comprise:

(a) *elliptical messages*;
(b) *semantically wrong messages*;
(c) *pragmatically wrong messages*;
(d) *messages with redundant information*.

[7] Recognition of the original NLEs as well as their execution in the MS-DOS environment is being carried out in real time.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

In the case of elliptical messages, the NLI presents to the user a helpful menu of all possible intended actions, and waits for him to make a choice. So, the user is not confused by possible unintentional mistakes. Regarding semantically incorrect messages, after locating the problem the system warns the user of its real cause with the help of a warning box, and proposes a solution. Special emphasis should be put on pragmatically incorrect messages. Although the NLI successfully recognizes the command identifier, the MS-DOS error handling mechanism undertakes to warn the user of its erroneous nature. Finally, messages with redundant information do not cause any problems, since all irrelevant information does not affect the command's identifier recognition process.

6 Conclusions and future work

The motivation for this research stems from the significant loss of productivity experienced by interactive computer environment users as a result of the syntactic inflexibility and relative linguistic inconsistencies of well-known command languages. Besides, no matter how restricted the subset of a command language may be, it is hard to anticipate in advance all NLEs which are likely to be used during the interaction of a NLI to such a command language with a casual user.

The main contribution of this research is the utilization of advanced techniques in a NLI to popular command languages to facilitate and broaden their use to a large audience. Such an adaptive NLI, apart from being able to be integrated with a future speech interface, can be (a) portable to a large range of command languages, (b) used to handle even complex commands thanks to an embedded linguistic parser, and (c) expandable and customizable by providing the casual user with the opportunity to specify some types of new words, as well as the system developer with the ability to introduce on-line new tasks in these application domains.

The advantages of the proposed NLI can be summarized as follows:

(a) it is a mixed system, applying knowledge prepared by a human expert in conjunction with knowledge acquired through machine learning,
(b) it offers extended morphological analysis combined with enhanced learning capabilities for unknown words and newly available facilities,
(c) it is a portable NLI, easily adaptable to other existing or future command languages attached to the standard syntax format paradigm,
(d) its embedded linguistic parser can assist in the disambiguation of simple as well as complex NLEs given by the user,
(e) the fact that semantically equivalent NLEs can lead to the same IUM has resulted in a significant decrease in the number of production rules,
(f) it can also handle elliptical sentences.

The disadvantages of the proposed NLI include the following:

(a) the introduction of ambiguity in NLEs may result in unexpected performance of the NLI,
(b) for the time being, it is impossible to define new TDOs,
(c) portability is only achieved for similar command languages.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

However, in response to the above disadvantages, the following observations can be made:

(a) free-style expressiveness of natural language causes some inevitable ambiguity problems;
(b) by enhancing appropriately the embedded linguistic parser these problems can be eliminated;
(c) the concrete version of the programming language we have used does not allow the on-line assertion of production rules for the definition of new TDOs;
(d) if a new command language with an entirely different syntax format paradigm appears, the knowledge representation scheme could be used with necessary slight modifications.

Our research has been currently focused on four directions: (a) enhancing the linguistic parser to cope with problematic anaphoric expressions; (b) defining new TDOs; (c) learning from NLEs existing or new TDAs to newly defined TDOs; and (d) learning new words other than verbs, nouns and adjectives that can be critical for the command's identifier recognition process. We hope that the outcome of this research will further amplify the utilization of similar NLIs to interactive computer environments.

References

Allen, J. (1987) *Natural Language Understanding*. Benjamin/Cummings.
Asfahl, D. (1991) *Robots and Manufacturing Automation*. Reading, MA: Addison-Wesley.
Barbehenn, M., Chen, P. C. and Hutchinson, S. (1994) An efficient hybrid planner in changing environments. *Proc. IEEE International Conference on Robotics and Automation*, 2755–2760.
Burton, R. R. (1976) Semantic grammar: an engineering technique for constructing natural language understanding systems. *PhD Thesis*, University of Irvine.
Burton, R. R. and Brown, J. S. (1979) Toward a natural-language capability for computer-assisted instruction. In: H. O'Neil (ed.), *Procedure for Instructional Systems Development*. Academic Press, 273–313.
Chen, P. and Hwang, Y. (1994) Generating natural-language like robot motion command set through machine learning. *Proc. 11th International Conference on Machine Learning*, 114–119.
Close, K. S. (1989) *MS-DOS and PC-DOS – A Handbook of Essentials*. Cincinnati, OH: South-Western.
DEC (1988) *VAX/VMS DCL Dictionary*, Digital Equipment Corporation.
Furnas, G. W., Landauer, T. K., Gomez, L. M. and Dumais, S. T. (1987) The vocabulary problem in human-system communication. *Communications of the ACM*, **11**: 964–971.
Groover, M. P. (1992) *Robot Programming*. Kluwer Academic.
Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D. and Slocum, J. (1978) Developing a natural language interface to complex data. *ACM Transactions on Data Base Systems*, **3**: 105–147.
Huffman, S. B. and Laird, J. E. (1993) Learning procedures from interactive natural language instructions. *Proc. 10th International Conference on Machine Learning*, 143–150.
Keller, J. F. (1984) An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Office Information Systems*, **2**: 26–41.
Manaris, B. J., Glanville, R. and Gillis, T. E. (1994) Developing natural language interfaces through NALIGE. *Proc. 6th IEEE International Conference on Tools with Artificial Intelligence*, 260–266.
Manaris, B. Z. and Dominick, D. W. (1993) NALIGE: a user interface management system for

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press

the development of natural language interfaces. *International Journal of Man-Machine Studies*, **38**: 891–921.

Michos, S. E. and Fakotakis, N. (1994) FRIENDOS: a self-extending natural language interface to MS-DOS Operating System. *Proc. 14th International Conference AI*'94, 13–22.

Michos, S. E. and Magoulas, G. D. (1995) A hybrid approach to knowledge representation and learning in a natural language interface to operating systems. *Proc. 5th GCS Hellenic Conference on Informatics*, 431–440.

Michos, S. E., Fakotakis, N. and Kokkinakis, G. (1994) A novel method for parsing complex sentences in syntactically-free languages. *Proc. 6th IEEE International Conference on Tools with Artificial Intelligence*, 253–259.

Michos, S. E., Fakotakis, N. and Kokkinakis, G. (1995) A novel and efficient method for Parsing unrestricted texts of quasi free word order languages. *International Journal on Artificial Intelligence Tools*, **4**(3): 301–321.

Michos, S. E., Magoulas, G. D. and Fakotakis, N. (1995) A hybrid knowledge representation model in a natural language interface to MS-DOS. *Proc. 7th IEEE International Conference on Tools with Artificial Intelligence*, 480–483.

Obermeier, K. K. (1989) *Natural Language Processing Technologies in Artificial Intelligence*. Chichester, UK: Ellis Horwood.

Samad, T. (1986) *A Natural Language Interface for Computer-Aided Design*. Kluwer Academic.

Schalkoff, J. R. (1990) *Artificial Intelligence*: *An Engineering Approach*. McGraw-Hill.

Shapiro, S. C. (ed.) (1987) *Encyclopedia of Artificial Intelligence*. John Wiley & Sons.

Shneiderman, B. (1992) *Designing the User Interface*: *Strategies for Effective Human-Computer Interaction*. Reading, MA: Addison-Wesley.

Shwartz, S. C. (1987) *Applied Natural Language Processing*. Princeton, NJ: Petrocelli Books.

UNIX (1986) *Programmer's Manual* 4.3 *BSD*, Berkeley Software Distribution.

Waltz, D. (1975) Natural language access to a large data base: an engineering aid. *4th IJCAI*, 868–872.

Wilensky, R., Arens, Y. and Chin, D. (1984) Talking to UNIX in English: an overview of UC. *Communications of the ACM*, **27**: 574–593.

Wilensky, R., Chin, D., Luria, M., Martin, J., Mayfield, J. and Wu, D. (1988) The Berkeley UNIX Consultant Project. *Journal of Computational Linguistics*, **14**: 35–85.

Winograd, T. (1972) *Understanding Natural Language*. Academic Press.

https://doi.org/10.1017/S1351324997001344 Published online by Cambridge University Press