Command Line Interactions using Natural Language

James Kim
Department of Software Engineering
University of Auckland
jkim202@ec.auckland.ac.nz

ABSTRACT

In recent years, graphical user interface (GUI) has rapidly dominated human-computer interaction. Although GUI is favored by many ordinary users, its limitations have been heavily noticed, especially with modern systems that contain rich set of functionalities. In particular, scalability is the key concern, which leads to usability and performance issues. GUI fails to scale with increasing number of functionalities. Another popular user interface is command line, which outperforms GUI in terms of speed. Despite many advantages a command line interface (CLI) has over GUI, CLI is almost merely used by software developers. Typical CLI requires special knowledge such as syntactical rules, command names and optional arguments, therefore ordinary users find it rather complex to use. In order to resolve dilemmas of two major user interfaces, enhanced ways of interacting with command lines have been proposed. Improvements in command line interaction include providing natural language commands, adaptive commands, and global-level command lines. This report explores these enhancements, focusing on the utilization of natural language and the mapping techniques between natural language expressions and commands.

INTRODUCTION

Natural language processing has been attempted in various papers. While researches agree natural language processing helps to build user-friendly command lines, it is still considered extremely difficult (Raskin, 2008). In order to prove the claim which suggests presence of user-unfriendliness in current GUI and CLI, an early study (Hayes, 1985) examined advantages and disadvantages of natural language interfaces and factors that affect the utility of natural language interfaces. Manaris, Pritchard and Dominick (1994) developed a restricted natural language interface for the Unix operating system. Michos, Fakotakis and Kokkinakis (1996) revealed various techniques to achieve an adaptive natural language interface that is portable, can handle complex commands and can learn new commands. Kate, Wong and Mooney (2005) proposed a similar concept of learning transformation rules. Their approach is based on incremental mapping of natural language sentences with formal query or command language. Little and Miller (2006) presented keyword commands which merely focuses on the presence of keywords in a command. They developed a Web-based prototype that demonstrates translation of "keyword commands" into executable code. The idea of attaching keywords or labels to resources such as files, emails, and photos improve the ability of searching commands (Norman, 2007). Norman explored the current search functions in the Web and operating systems. In contrast to previous proposals, Raskin (2008) stretched the concept one step further by illustrating a global-level command line interface called the "linguistic command line".

This report discusses drawbacks of current user interface paradigms and compares various articles that attempt to enhance human-computer interaction through utilization of natural language commands. This report also explores partial details of mapping techniques that have been applied in order to produce their solution. Potentials and limitations of natural language processing have also been identified.

LIMITATIONS OF CURRENT USER INTERFACES

Hayes (1985) suggested that two categories of user interface are graphical and command line or combination of the two. This fact still stays valid today. However, these two major user interface paradigms are facing many problems with demand of both humans and computers today.

Limitations of GUI

Until now, GUI has served as an important user interface. GUI is still considered as a valuable interaction channel between humans and computers, but they fail to scale with the demands of modern systems (Norman, 2007). In GUI, number of menu items is increasing proportionally with growing number of functionalities. An immediate solution to overcome this problem was to divide the system into smaller applications by compartmentalizing the tasks. Each of the applications would provide specific collection of functions. However, different problems emerge through compartmentalization (Raskin, 2008). Firstly, needless compartmentalization forms the necessity to switch between applications, which lead to performance reduction

and user frustration. Also, redundancy concerns caused by overlapping functions are not avoidable. Similarly, scalability issues are also noticed in file systems, where it has become inefficient to find something in typical huge hierarchical file structures. These examples already provide enough evidence that the window-icon-menu-pointing-device (WIMP) paradigm, which forces recognition, does not fulfill the requirements of current systems in terms of usability.

Limitations of CLI

Despite the fact that textual interface has many advantages over a visual interface including functionality and performance, classic CLI is overall less popular than GUI. While command languages assume user to be expert enough to carry out their tasks (Michos et al., 1996), novice and intermediate users experience significant loss of productivity (Manaris et al., 1994). Memorizing command names and command line options are considered most difficult in utilizing typical CLI (Raskin, 2008). Also, strict syntax rules largely contribute in raising the complexity. Little and Miller (2006) found that other barriers involve learning and switching between different command languages and learning the Application Program Interface (API) of the domain, which can be very large, especially for end-user programming.

NATURAL LANGUAGE COMMAND LINE INTERACTION

Despite the presence of difficulty in using command languages, it has more advantages than GUI. In fact, they are still being used in many applications, usually without the users' intentions. People do not notice simple command line interfaces that have been incorporated into systems, such as the search box on the Web and page range box in Microsoft Word's printing options. Unnoticeable interface is a sign of a good interface (Raskin, 2008). Wide range of functions can be effortlessly provided by a text interface compared to GUI, which lack scalability (Raskin, 2008). The key reason is that text is so much lighter than graphics. The ability for pure texts to be easily viewed, edited, copied, pasted, stored and shared is offered in almost every user interface and application (Little & Miller, 2006).

This section discusses how natural language enhances command line interaction, the solutions that have been proposed, the approaches in mapping natural language to command language, and the key findings in researches.

Advantages of Natural Language Utilization

Natural language interface's key advantages over other paradigms had been recognized early (Hayes, 1985). Firstly, commands or queries can be expressed in many ways, hence the expressiveness. Secondly, there is no learning involved since everyone is already familiar with natural language. Another benefit is its naturalness. People are comfortable with "natural" language. Michos et al. (1996) also pointed out that natural language broadens audience by facilitating the use of command languages. Flexibility and robustness are other key advantages of natural language. This overcomes the problems of old CLI outlined in the previous section by allowing variations of commands that are not strictly adhered to syntax rules (Norman, 2007). All of these advantages contribute to increased user-friendliness (Kate et al., 2005).

Proposed Solutions

Prior to building natural language interface, it is important to recognize key factors that may affect the utility of natural language. These have been identified early, which include types of user, interface hardware, application domain, input medium, degree of coverage, and combination with other input types (Hayes, 1985).

At present, it is impossible to fully anticipate the user input. Norman (2007) mentioned that natural language succeeds with well-defined tasks but natural language processing is still considered extremely difficult (Raskin, 2008). Therefore, the following solutions provide close-to-natural-language commands based on restricted natural language interface. Translating natural language expressions to commands is a very sophisticated task. Some papers revealed their mapping procedures on specific domains, which will be discussed later.

Earlier research (Manaris et al., 1994) focused on developing a natural language interfaces for operating systems (NLIOS). Their primary goal was to provide "natural" interaction between users and the operating system. They built a natural language interface for the Unix operating system, using a user interface management system called Natural Language Interface to Operating System Generation Environment (NALIGE). NLIOS integrated synonymy and polysemy of words, which refers to words that have same meaning and words that have more than one meaning, respectively. For example, "create" and "make" or "file" and "files" are correspondingly recognized to have same meaning and treated identically due to synonymy. In contrast, "move" either means rename or change directory depending on the rest of the expression due to polysemy. These attributes create flexibility by enabling variants of natural language commands to be accepted and interpreted equally. The following phrases equivalently create a new directory called x: "create x", "make directory x", "new directory x", "create directory named x", "create new directory x", "make a new directory x", and "make a new directory named x".

Similarly, Michos et al. (1996) proposed a method of interpreting variants of command expressions equivalently, by using semantic grammar rules that remove unnecessary information. The command "I would like you to display me the files of the directory GAMES" would ignore all redundant words such as 'I', 'would', 'like', 'you', 'to' and 'me'. Therefore same function can be expressed in multiple styles. This system also supports synonyms, handling alternatives of commands more effectively. For example, "Show the contents of the list GAMES" executes the same

command as before. In addition to the benefits offered by Manaris et al. (1994), Michos et al.'s system can expand its knowledge base through a learning paradigm called User-Assisted Symbolic Concept Acquisition. It is capable of learning new vocabulary and actions by both casual users and system developers, which significantly increases flexibility.

Self-expandable interface has also been introduced by Kate et al. (2005). They developed SILT (Semantic Interpretation by Learning Transformations) based on two domains, RoboCup Coach Language (CLANG) and database query application. SILT uses a novel approach in mapping natural language sentences into domain-specific commands using transformation rules. The system learns transformation rules by iteratively comparing patterns found in natural language with templates based on command language.

In contrast to previous methods, Norman (2007) anticipated a simple, yet an effective solution, "enhanced searching". Search engines on the Web have transformed into answer services as users are increasingly typing commands, instead of search terms, for prompt answers. Modern search engines support search commands based on natural language syntax. On Yahoo!, the phrase "time in Nagoya" returns the current time in Nagoya and on Live.com, "cars in China" returns "15 per 1000 people". Similarly, definitions from various knowledge bases are immediately returned by expressions like "define: polysemy" on Google. Many search engines are capable of tolerating command variants as well. Grammar and spelling errors are automatically retrieved by returning most-likely results or suggesting possible alternatives. Also, it has been observed that some advanced search engines cope with synonyms as well. These features contribute significantly towards flexibility and usability. Norman discovered that recent operating systems have started to integrate powerful search mechanisms after learning lessons from the Web. Attaching keywords or labels to files increases the performance of search, for example, allowing quick navigation of file system.

Little and Miller (2006) developed a Web-based prototype which demonstrates "keyword commands" being translated into executable code. Their system follows the idea of using keywords. In fact, keyword commands focus only on the presence of keywords in the expression and do not oblige to any language constructs. Downside is that users require prior knowledge of the domain because the keywords recognized by the system are domain-specific. For instance, in Microsoft Word domain, the keyword command "left margin 2 inches" requires the user to know the term "margin", which is the white border around the page. Keyword commands eliminate strict requirements for

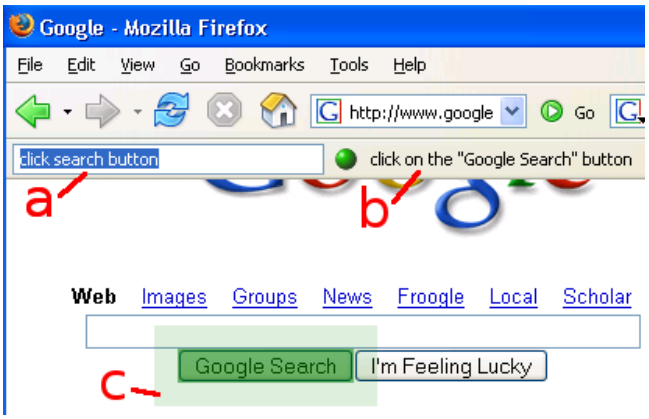

Figure 1. Web-based prototype which demonstrates keyword commands (Little & Miller, 2006)

punctuation and grammar, which allows for many variations of expressions. Their concept is quite similar to method presented by Michos et al. (1996). Verbose expressions such as "set the left margin to 2 inches", and different word ordering like "2 inches, margin left", would equivalently evaluate to the earlier command. A prototype in the Web browsing domain (Figure 1) has been developed to demonstrate automatic translation of keyword commands into executable code. When a (a) command is typed into the textbox, two types of output are generated: (b) textual feedback of the executed command in a form of pseudo-natural language and (c) graphical representation on the target control. For example, when "click search button" is entered into the textbox, the text "click on the Google Search button" is displayed and the "Google Search" button is highlighted.

Unlike all of the approaches outlined above, where commands only worked on domain-specific applications, Raskin (2008) introduced a distinctive CLI called linguistic command line interface, which is currently available at no cost under name of "Enso". Enso provides all functionality at any time regardless of the application you are using. It was designed primarily to eliminate the compartmentalization of applications. Although Enso relies on structured syntax, its commands are easily guessed. Most commands begin with a verb and some are even complete English expressions. For example, the command "translate <selected text> to french" translates the selected text into French using Google's translation service. Figure 2 illustrates interaction of Enso's interface. Users are guided with a list of potential commands dynamically, as keys are being entered.

Figure 2. Enso interface (Raskin, 2008)

Mapping of Natural Language and Commands

This sub-section reports key parts in various mapping techniques between natural language expressions and commands. The methods outlined here are incomplete because further knowledge is required to understand them in depth.

Michos et al. (1996) found that the task of discovering set of natural language expressions to be modeled was most challenging. They identified these by conducting a survey on undergraduates and postgraduates who had computer science background. Their system knowledge base consists of knowledge structures of both verbs and nouns, that is, natural language commands and arguments, respectively. For example, if "Display the contents of the directory GAMES" is requested, the system looks up the knowledge base, examining the following knowledge structures.

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

bgkey('directory',['dir', 'chkdsk', 'cd', 'copy', 'del', 'print', 'rd', 'tree', 'type', 'xcopy'])

The intersection of above knowledge structures is 'dir', which is returned as an expected command identifier. Through an advanced linguistic parser, simple and complex natural language expressions can be handled. Also, semantically equivalent natural language expressions are translated into the same command. As mentioned earlier, their knowledge base may grow through machine learning techniques.

SILT (Kate et al., 2005) accepts two versions of mapping representations. Their approach focuses on learning transformation rules that associate patterns with templates. Each version uses strings of words or trees as patterns to match against natural language sentences. Transformation rules are iteratively learnt through their complex algorithms.

Manaris et al. (1994) constructed NLIOS through several key steps, which include identifying operating system operations, determining natural language expressions, identifying corresponding command language syntax for these operations, modeling the above knowledge, and producing an interface to encapsulate the above knowledge. Similar to Michos et al. (1996), they found that the most challenging task was determining the possible natural language expressions. They also conducted a survey to achieve this. NLIOS is essentially based on three sets of input specifications. Augmented semantic grammar (ASG) specification consists of possible natural language expressions that map to commands in operating system command language (OSCL) specification. Lexicon specification models synonymy and polysemy in relation with ASG.

In solution presented by Little and Miller (2006), keywords are converted into a function tree prior to being translated into executable code. This translation algorithm is broken down into two steps. The first step is tokenizing the input. The input is split into words or "tokens". For example, if the input is "LeftMargin", the system splits it into two tokens "Left" and "Margin". Further separation is performed through a spell checker for common compound expressions. For instance, "login" becomes "log" and "in" tokens. The second step is to feed the token sequence obtained from the previous step into a recursive algorithm that tries to match substring of tokens with function names. Through this algorithm, the function with highest explanatory power, which measures how well the function explain the token sequence, is identified as the best fitting command. Keywords in the expression are used as identifiers. The translation from keyword to command is performed if there is at least a partial match between the keyword and the function name. For instance, a single keyword "inches" is sufficient to be interpreted as InchesToPoints function. The ultimate function is dependent on the arguments. That is, if there exist multiple candidates, the system decides upon the remaining keywords.

Evaluation

There were several interesting findings presented in these researches. Firstly, Michos et al. (1996) conducted performance test on their interface and found that it had a real-time response. Also, the system knowledge base increased from 500 to 1500 words after introducing learning capabilities. Considering that these figures are over ten years old, this is an outstanding improvement. Kate et al. (2005) claimed that SILT performs overall better than previous systems despite the fact that their system had lower score in precision and recall. They suggested SILT is more general and supports multiple target command languages. Little and Miller (2006) conducted a user study on their prototype, which showed that untrained users generated successful keyword commands for 90% of the tasks and succeeded 73% on their first attempt. Although this system requires users to have basic knowledge about the domain, these figures suggest that learning curve of complex syntaxes can be overcome through more "natural" commands.

SUMMARY

Many researches have focused on providing user interface that is more scalable, user-friendly and flexible in order to

overcome the limitations of current user interface paradigms that are becoming more evident. Scalability is increased through developing upon pure textual command lines that are more portable. Usability is increased through natural language utilization, providing more "natural" input. Flexibility is increased through accepting synonyms, handling variants, and learning new commands. The current state of natural language utilization is quite rare nevertheless the paradigm is rapidly expanding from the Web to operating systems and will continuously develop.

FUTURE WORK

Utilization of natural language has enhanced command line interaction. It improves both existing and emerging issues of current user interface paradigms. However, natural language interfaces have their own limitations. As mentioned previously, natural language processing is still considered as an extremely complex task (Raskin, 2008). Firstly, the inability to fully anticipate user input is the key concern. This problem will not be resolved entirely regardless of knowledge base size because current mapping techniques cannot interpret variety of styles used to express ideas with arbitrary English expressions (Little & Miller, 2006). Users still have to learn system's capabilities because free input is restrained (Hayes, 1985). Another weakness is that command line by itself is not efficient in doing certain tasks. English expressions can be verbose, which means it could take many keystrokes than syntactical commands. Balanced combination of GUI and enhanced CLI that accepts both formal and natural language commands could be the key.

REFERENCES

Hayes, P. J. (1985). *The utility of natural language interfaces.* Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computer Systems, San Francisco, California, United States. Retrieved April 10, 2008, from http://doi.acm.org/10.1145/317456.317460

Kate, R. J., Wong, Y. W., & Mooney, R. J. (2005) *Learning to transform natural to formal languages.* Paper presented at the Proceedings of the 20th National Conference on Artificial Intelligence, Pittsburgh, PA. Retrieved April 10, 2008, from http://www.cs.utexas.edu/~ml/papers/transform-aaai-05.pdf

Little, G., & Miller, R. C. (2006). *Translating keyword commands into executable code.* Paper presented at the Proceedings of the 19th Annual ACM Symposium on User interface Software and Technology, Montreux, Switzerland. Retrieved March 22, 2008, from http://doi.acm.org/10.1145/1166253.1166275

Manaris, B. Z., Pritchard, J. W., & Dominick, W. D. (1994). Developing a natural language interface for the UNIX operating system. *SIGCHI Bulletin, 26*(2), 34-40. Retrieved April 10, 2008, from http://doi.acm.org/10.1145/198125.198137

Michos, S. E., Fakotakis, N., & Kokkinakis, G. (1996). Towards an adaptive natural language interface to command languages. *Natural Language Engineering, 2*(3), 191-209. Retrieved March 22, 2008, from http://portal.acm.org/citation.cfm?id=974680.974681

Norman, D. (2007). The next UI breakthrough: command lines. *Interactions, 14*(3), 44-45. Retrieved March 21, 2008, from http://doi.acm.org/10.1145/1242421.1242449

Raskin, A. (2008). The linguistic command line. *Interactions, 15*(1), 19-22. Retrieved March 21, 2008, from http://doi.acm.org/10.1145/1330526.1330535