Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 31 (2014) 1005 – 1011

Procedia Computer Science

2nd International Conference on Information Technology and Quantitative Management, ITQM 2014

The Design of a Natural Language Interface for File System Operations on the Basis of a Structured Meanings Model

Alexander A. Razorenov*, Vladimir A. Fomichov

Department of Innovations and Business in the Sphere of Informational Technologies, Faculty of Business Informatics, National Research University Higher School of Economics, Kirpichnaya str. 33, 105187 Moscow, Russia

Abstract

Due to evolutionary increase of the software complexity, graphical user interfaces have been becoming more and more complex, and often this process encounters the restricted possibilities of reflecting graphics on a display. This paper describes an original approach to designing natural language interfaces for the action-based applications as the complements and alternatives to traditional graphical interfaces. The principal peculiarity of this approach is that it is based on a formal model of natural language structured meanings. The first part of the paper outlines the main properties of the class of SK-languages (standard knowledge languages) which is used for building semantic representations of user commands. The class of SK-languages can be employed for representing the meanings of input texts from practically interesting sublanguages of English, French, German, Russian and other natural languages. The theory of K-representations (knowledge representations), introducing the class of SK-languages, was developed by V.A. Fomichov and described in his monograph published by Springer in 2010. The second part presents an application of this theory to the design of a natural language interface to an action-based software system, it is called NLC-1 (Natural Language Commander – Version 1).

Keywords: natural language processing, natural language interface, action-based applications, theory of K-representations, structured meaning, SK-languages, semantic representation

* Corresponding author. Tel.: +7-916-705-2940
E-mail address: razorenov@mail.ru.

1. Introduction

During last 15 – 20 years, the standard of a user interface for the interaction with applied software de-facto has been graphical user interface. Due to evolutionary increase of the software complexity, graphical user interfaces have been becoming more and more complex, and often this process encounters the restricted possibilities of reflecting graphics on a display. This situation has lead to the following unpleasant consequences:

- A user is unaware of a part of the software possibilities, because he/she doesn't find them in the endless menus, nested interface's elements, and in the opening (one after another) windows.
- The user spends a considerable time for the search or the access to the required element (in case he/she remembers the location of the required element of an interface: a button, a menu item, a checkbox, etc.).

The first kind of problems could be demonstrated by modern office software: document editors, spreadsheets, etc. Despite of design improvements like the Ribbon in the Microsoft Office, most part of features are not available for casual user.

The navigation problems may be seen in large EPR systems, for example, Oracle e-Business Suite or SAP ERP. These problems are a characteristic of Graphical User Interface (GUI). For example, the search of files in the folders or the search of the programs in the numerous menus of the system often becomes a tedious process. Modern operation systems contain tools which could help to solve this problem in some cases.

Another approach is software tools that define a required action and its parameters during the process of constructing natural language sentences. For example, Microsoft Outlook uses a method of the kind to define the rules of message processing. The Business Rules for Oracle SOA Suite is an example within the Enterprise software field. Also there is at least one development tool with the same features. It is MyProLang[2]. But this kind of interfaces are not Natural Language Interfaces.

The analysis shows that a perspective way out of this complex situation seems to be the design of interfaces enabling the user of a complex software system to interact with it with the help of a restricted natural language (in other terms, by means of a controlled natural language) – English, French, Russian, etc. Here natural language (NL) interfaces (NLI) for the action-based applications (NLIABA) are considered as the compliments and alternatives to traditional graphical interfaces.

2. State of the arts

The field of *Natural Language Interfaces (NL-interfaces) to Action-Based Applications* is not so explored as *Natural Language Interfaces to Databases*, but some theoretical studies do exist, see, e.g.,[3].

One of the first commercial applications with NLI was Lotus HAL – a NL-interface for Lotus 1-2-3 spreadsheet application. According to[12], Lotus HAL was more effective in average but maximum user performance was reached with native Lotus 1-2-3 interface.

Also there are different applications using NL-interfaces to access their functionality or to access functionality of other applications. For example:

- Siri[14] – a personal assistant software for iPhone 4S/5;
- *Sobesednik HD*[15] – a Siri's analog for Russian;
- *Ubiquty*[16] – an extention to Firefox web browser;
- *GNOME Do*[11] – an application for GNOME desktop environment;
- *Braina Project*[10] – a software for working with computer;
- *NLUI Server*[13] – a multilingual application server for NL-interface Scripts supports many languages, including English, Russian, French, German, Japan and many others.

These applications support simple "one-action" commands: open file, create folder, copy a file to a folder, answering "how many time". But the scope of user commands is much more. It contains complex instructions like “Copy all music and text files created this year from folder A to folder B, archive them and send to e-mail address someperson@example.com”. It is impossible to support such complex instructions using simple algorithms like pattern matching. A complex theory of NL-interfaces to action-based applications is required to support this kind of user commands.

3. Some principal properties of SK-Languages

The analysis shows that the broadest prospects for developing the theory of designing NL-interfaces to action-based applications are open by the theory of K-representations (knowledge representations). This theory, developed by V.A. Fomichov, introduces, in particular, a new class of formal languages called SK-languages (standard knowledge languages). The definition of the class of SK-languages is a part of a mathematical model developed in[4] and describing ten partial operations on conceptual structures. This model provides the forms for representing structured meanings of natural language sentences and arbitrarily complex discourses pertaining to mass spheres of professional activity (technology, medicine, economics, sport, etc.). Let us consider some important properties of SK-languages, determined in[4] and discussed in[5,6].

The collection of basic symbols of each SK-language includes two disjoint sets:

- *X* – a primary informational universe – a set of basic semantic items;
- *V* – a set of the variables.

A primary informational universe includes several disjoint sets:

- *St* – the set of sorts (the designations of most general notions), its items can be used for associating the expressions of SK-languages with their semantic characteristics (they are called types);
- *F* – the set of the functional symbols;
- *R* – the set of the relational symbols.

There are some required symbols in every primary informational universe, in particular, $\equiv$, $\neg$, $\wedge$, $\vee$, $\forall$, $\exists$. These symbols are used for building K-representations of natural language sentences and discourses. The term “K-representations” is used for denoting semantic representations of texts being the expressions of SK-languages.

The K-representations of natural language texts are constructed from the items of the primary informational universe, the variables, and several service symbols by means of an iterative process of applying the rules of building well-formed formulas P[0], P[1], …, P[10]. The initial set of simplest formulas is determined by a special formal object called a conceptual basis and playing the role of the simplest knowledge base [4]. The language determined by the considered conceptual basis *B* and the rules P[0], P[1], …, P[10] is denoted as *Ls(B)* and is called the standard knowledge language (SK-language) generated by the basis *B*.

Let’s consider a short introduction to the rules of constructing formal representations of structured meanings P[0], P[1], …, P[10].

The rule P[0] provides an initial stock of formulas. For example, if the string *file* is an element of a certain primary informational universe *X*(*B*), then *file* is a formula of *Ls(B)*.

The rule P[1] can be used to join intentional quantifiers to the concepts and to produce the formulas like

*certain file, certain computer * (Manufacturer, IBM), all computer * (Manufacturer, IBM).*

The rule P[2] can be used to construct the formulas like *f(t1,…tn)*, where *f* is a functional symbol, and *t1, ..., tn* are the well-formed formulas of *Ls(B)*. For example, *Size(certain file)* is a well-formed formula of a certain SK-language *Ls(B)*. The rule P[3] can be used to construct the formulas with identity symbol *(a* $\equiv$ *b)*. For example, *(Size(certain file)* $\equiv$ *n12), (Duration(x18))* $\equiv$ *9/minute).*

The rule P[4] can be used to construct the formulas like r(t1, …, tn), where r is a relational symbol, and *t1, ..., tn* are the well-formed formulas of *Ls(B)*. For example, *Less(Size1(certain file), 1024/byte).*

The rule P[5] allows us to mark K-representations by some variable from the set of variables. For example, if a part of a K-representation looks like *certain file * (Extension, ".rtf") : v1,* then we can refer to the expression *certain file * (Extension, ".rtf")* in another part of a K-representation, using *v1*.

The rule P[6] provides the possibility to construct K-representations in the form ¬*Formula*, for example ¬ *file*.

The rule P[7] allow us to use conjunction and disjunction in K-formulas: *(file ⋁ folder)*, *(file ⋀ folder)*.

The rule P[8] can be used to create compound designations of notions in the form

*concept * (r[1], value[1])…(r[N], value[N]),*

where *concept* is an element of a primary informational universe denoting a notion, r[1],…, r[N] are the names of functions or relations, and the value[1],…,value[N] are well-formed formulas. This rule allows us to construct the formula *file * (Size,0)(Extension, "txt")* which is a K-representation of text "*a file with zero size and extension "txt"*".

The set of the rules described above provides the possibility to construct a K-representation of the natural language text "Empty text file is a file with zero size and extension "txt"". This K-representation is as follows:

*(file * (Property, empty) ≡ file * (Size1, 0)(Extension, "txt"))* .

The rule P[9] allows us to use the quantifiers ∀ and ∃ like in first order logic. The rule P[10] enables us to build the representations of ordered tuples as the formulas of the form *<a[1],..., a[N]>,* where *a[1],..., a[N]* are some well-formed formulas. This rule can be used to construct K-representations like the following:

<Place, certain backup-drive>, <Time, Midnight>, <Frequency, Everyday>.

These tuples could be used to construct representations of complex verb constructions. For example:

*Delete(<Object, all file * (Size, 0)(Extension, "txt")>, <Time, Midnight>, <Frequency, Everyday>).*

Besides, this rule allows us to construct more long tuples, for example, for 3D coordinates: *<1,15,10.5>*.

SK-languages are strongly typed. Every item of primary informational universe is typed by a certain sort or more complex type. The variables are also typed. The type of a K-representation is defined by types of its parts and construction rule.

Every complex type belongs to one of the follows classes:

- The types of the form *t * u *... * v,* where *t, u,..., v* are sorts. This class of types is used for typization of objects with different properties. For example: *disk-drive* can be typed as *informational-object* and as *physical-object*, so its compound type is *informational-object * physical-object* in a certain conceptual basis.
- The types of the form *{t}* are used for the groups of objects of the type *t*. For example, the type of the formula *all disk-drive* could be *{information-object * physical-object}*.
- The types of the form *(t , u,...,v)* allow us to define the types of *n*-tuples constructed with the help of the P[10] rule.
- The types of the form ↑*t* are used as the characteristics of notions (or concepts). The type *t* can be either a sort or a complex type. For example, the type of every sort *s* is ↑*s*.

The association of semantic items (primary and compound) with the types provides the possibility to prevent the construction of semantically incorrect K-representations such as (*file ⋀ 1024),* where a designation of a notion is mixed with a designation of number.

4. Natural language interface design and implementation

The Natural Language Commander – Version One (NLC-1) is a file manager with a NL-interface actively developed now for the studies and experiments in the field of NL-interfaces to action-based applications. Its algorithm of semantic-syntactic analysis of input texts *SemSynt2* is an expansion and modification of the multilingual algorithm of semantic-syntactic analysis *SemSynt1* stated in Chapters 8, 9, and 10 of the monograph 4. The main goal of the project is to develop a comprehensive theory of NL-interfaces to action-based applications (NLIABA) and to design a useful software system for human - computer interaction. The application NLC-1 transforms an input text into its possible K-representation for user instruction and then transforms it *into a unified form*.

This unified form is also a K-representation where the designations of complex concepts like music, video, etc. are replaced with basic concepts and additional qualifications. For example, the term “documents” is interpreted as the files with extensions “doc”, “docx”, “odt”. The content of the next step is that the application transforms this unified K-representation into a script of the operation system shell; now it’s Bourne-again shell (bash) both for Unix-like OS and for Windows. The final step is the execution of this script.

Example. Let’s look how NLC-1 processed the following user instruction: “Move documents from “Project” folder to the folder with the name “Docs” or “Documents” on backup drive if their size is less than 2 GB”. This instruction is transformed by the algorithm *SemSynt2* into the following K-representation:

*If-then(Less(SizeOf(all Document * (Place, certain folder1 * (Name, "Project")):o1), 2/GB), Command(#Operator#, #Executor#, #now#, movement1 * (Source,o1)(Destination, certain folder1 * (Name, ("Docs" ∨ "Documents")))(Place, certain backup-drive) : o2))).*

Now if the knowledge base of NLC-1 contains the K-representations

*(Document ≡ File * (Extention, ("doc" ∨ "docx" ∨ "odt"))),*

*(backup-drive ≡ drive1 * (Name, "F"))*

and the knowledge management system includes the rule

(x, x ≡ y ├ y)

then the system NLC-1 transforms the constructed K-representation of the user instruction into a unified K-representation

*If-then(Less(SizeOf(all file * (Extention, ("doc" ∨ "docx" ∨ "odt"))(Place, certain folder1 *(Name, "Project")): o1), 2/GB), Command(#Operator#, #Executor#, #now#, Copying1 * (Source, o1)(Destination1, certain folder1 * (Name, ("Docs" ∨ "Documents")))(Place, certain drive1 * (Name, "F") : o2))).*

Then the result shell script is as follows:

```
if [ $(du -cb "Project/*.doc" "Project/*.docx" "Project/*.odt" | grep total|sed -e "s/\s.*$//g") -le
1000000000 ]; then mv "Project/*.doc" "Project/*.docx" "Project/*.odt" $(ls /f/|grep -iE
"^Docs$|^Documents$" | head -n1); fi
```

Written in Haskell programming language[7,8,9], the system NLC-1 is a flexible and scalable application. It may be configured by the researchers for different domains and underlying shells.

5. Conclusion

According to some studies (see, in particular,[1,12]), natural language interfaces are not effective for simple applications. But they could be effective for more complex software such as enterprise software or operation system at whole. The method described in this paper can simplify human-computer interaction. Due to the use of SK-languages for constructing semantic representations (or text meaning representations) of the input texts, the proposed algorithm can operate with complex user commands. That is why the elaborated method provides the possibility to create natural language interfaces to complex software. This method allows us to develop natural language interfaces for action-based applications with support of user commands from simple to long complex instructions.

References

1. Androutsopoulos I, Ritchie G., and P. Thanisch. Natural Language Interfaces to Databases. arXiv:cmp-lg/9503016
2. Bassil Y., Barbar A. MyProLang – My Programming Language: A Template-Driven Automatic Natural Programming Language. arXiv:1204.0221
3. Chong S., Pucella R. A Framework for Creating Natural Language User Interfaces for Action-Based Applications, Proceedings of the Third International AMAST Workshop on Algebraic Methods in Language Processing, TWLT Report 21, pp. 83-98, 2003 (arXiv:cs/0412065)
4. Fomichov V.A. Semantics-Oriented Natural Language Processing: Mathematical Models and Algorithms. IFSR International Series on Systems Science and Engineering, Vol. 27. Springer: New York, Dordrecht, Heidelberg, London, 2010.-354 p.
5. Fomichov V.A. Theory of K-calculuses as a Powerful and Flexible Mathematical Framework for Building Ontologies and Designing Natural Language Processing Systems // Troels Andreasen, Amihai Motro, Henning Christiansen, Henrik Legind Larsen (Eds.), Flexible Query Answering Systems. 5th Intern. Conference, FQAS 2002, Copenhagen, Denmark, Oct. 27 - 29, 2002. Proceedings; LNAI 2522 (Lecture Notes in Artificial Intelligence, Vol. 2522), Springer: Berlin, Heidelberg, New York, Barcelona, Hong Kong, London, Milan, Paris, Tokyo, 2002. P. 183-196.
6. Fomichov, V.A. The Advantages of Using SK–languages for Designing Semantic-Syntactic Analyzers of Recommender Systems // Kurt J. Engemann and George E. Lasker (Eds.) Advances in Decision Technology and Intelligent Information Systems, Vol. XIV. The International Institute for Advanced Studies in Systems Research and Cybernetics (IIAS), Tecumseh, Ontario, Canada, 2013, p. 37-41.
7. Marlow, Simon. Parallel and Concurrent Programming in Haskell, O'Reilly, August 2013, English, 322 pages, ISBN-10: 1449335942, ISBN-13: 978-1449335946
8. O'Sullivan B., Stewart D., and Goerzen J.: Real World Haskell, Paperback: O'Reilly, November 2008, English, 700 pages, ISBN-10: 0596514980, ISBN-13: 978-0596514983
9. Peyton Jones, Simon: Haskell 98 language and libraries: the Revised Report, Cambridge University Press, 2003, 272 pages, ISBN 0521826144
10. Braina Project Home Page: http://www.brainasoft.com/braina/#overview (retrieved 2014-02-10)
11. GNOME Do Home Page: http://do.cooperteam.net/ (retrieved 2014-02-10)
12. Is it easier to learn how to use a computer program if you communicate with it using English-language type commands?: http://onlinestatbook.com/case_studies_rvls/interface/index.html (retrieved: 2014-02-15).
13. NLUI Server Home Page: http://www.linguasys.net/web_production/server-item/NLUI%20Server (retrieved 2014-02-09)

14.Siri Homepage: http://www.apple.com/ios/siri/ (retrieved: 2014-02-10)

15.Sobesednik HD Home Page: http://sobesednik.idevelops.com/ (retrieved 2014-02-10)

16.Ubiquity Home Page: https://wiki.mozilla.org/Labs/Ubiquity (retrieved 2014-02-10)