

# Dialogue Interface for Programming in Prolog

*Adriana Jergová*

*Faculty of Informatics, Masaryk University,  
Botanická 68a, 602 00 Brno, Czech Republic  
e-mail: ada@fi.muni.cz*

## **Motivation**

In the last few years spoken interfaces are often used in the telephone-based querying systems, voice-controlled systems, etc. For the visually impaired users or users with other disabilities they can bring new opportunities to overcome some of the difficulties resulted from their handicap. At Laboratory of Acoustics and Phonetics at Masaryk University we are developing programming environments for various languages that facilitate the task of programming for blind programmers as the conventional programming environments are extremely inconvenient for them [4, 5].

Here we present a dialogue conception of spoken programming environment for Prolog language that is designed for visually impaired programmers. It addresses specific issues that arise in process of programming by means of spoken interface without possibility of visual checking. Advantages and problems related to sequential input, structured input, and semantic input are compared.

Employment of spoken interface in the applications where graphical interface is serviceable needs special techniques and strategies to be adopted. In the case of programming by means of spoken interface the communication between the user and the system involves several interfering areas and levels—input of text (source code), speech informational feedback providing the source code overview, system commands, help (explication, guidance), and metacommunication (clarifications, repairs).

In this paper we concentrate our attention to the first two mentioned areas and we suggest the representation of the treated text (programs in Prolog) which allow us to exploit advantages of the spoken input and to compensate impossibility of visual checking. The main goals are

- elimination of syntactic errors,
- reduction of semantic errors and support for their search and correction, and
- support for good orientation in the program source.

## **Sequential Syntactic Input**

A straightforward way of entering program source could be providing a user with possibility to dictate the text in sequential manner "word by word" or "character by character" imitating thus classical keyboard input. This approach without additional analysis does not support good orientation in text and programs created in this way would be very error prone. On the other side, this method can be applicable in some cases, for example when the text is completely prepared and does not require modification or consists of only short sequences without nesting, sequences with atypical or without any structure. This mode of input may be employed also for unexperienced single-session users, etc.

## **Structured Syntactic Input**

More natural and safer way would be entering of semantic units of reasonable granularity having precise syntactic form constructed automatically. To determine the size and the structure of input units as well as their hierarchy we must take into account

- grammatical structure of program sentences – in order to preserve their original structure and to provide correct syntax of the final output text

- semantic sectioning of the program – in order to allow convenient organization of the program and to facilitate orientation in the entered text for the user
- human capabilities – in order to minimize memory load of the user.

### **Language Constructs and Program Units**

In order to create a particular language construct we need to specify its name and content, its exact written form will be produced automatically. Support for manipulation with predicates, clauses.

### **Semantic Input**

Although compound term syntax is the same (with exceptions like prefix, infix, or postfix form for operators, list notation, etc.) on its various positions in the language constructs, treating terms differentially (i.e. using term's context information in naming of processed particles) supports better orientation in the text. Another helpful strategy is the employment of meaning of the term elements. This can be realized implicitly via intelligent variable and atom naming. This is to be done by the user, but the system can support and exploit proper naming and thus facilitate orientation in term during its creation, reading, or modification as well as searching and checking of interpredicate relations, reducing thus the risk of semantic errors. Meaning can be introduced also explicitly by using *field names* for the term arguments in the style of ECL<sup>1</sup>PS<sup>o</sup> [2] library `structure.pl` or by means of predefined sentence patterns.

At the present stage of the system development, the meaning of term arguments must be provided by the user itself and therefore it is always a trade-off in efficiency of input and efficiency of later processing. Providing additional information is recommended mainly for terms representing predicates or often used data. It is subject of the future research, how to detect arguments and approximate their meaning automatically from input sentences.

### **Conclusion**

### **Acknowledgement**

The research has been partially supported by Czech Ministry of Education under the Grant VS97028 and by Grant Agency of the Czech Republic under the Grant 201/99/1248.

### **References**

- [1] Bernsen, N.O., Dybkjoer, H., and Dybkjoer, L.: What should your speech system say?, IEEE COMPUTER, 30(12) 1997, pp. 25-30
- [2] ECL<sup>1</sup>PS<sup>o</sup> User Manual, Release 4.0, ECRC 1998
- [3] Kopeček, I.: Modeling of the Information Retrieval Dialogue Systems; Proceedings of the Workshop on Text, Speech and Dialogue - TSD'99, pp. 302-308
- [4] Kopeček, I.: Dialog Based Programming; Proceedings of the First Workshop on Text, Speech and Dialogue - TSD'98, 1998, pp. 407-412.
- [5] Kopeček, I., Jergová, A.: Programming and Visually Impaired People; in Proceedings of ICCHP'98, Wien-Budapest, September 1998, pp. 365-372.
- [6] Mooney, R.: Learning for Semantic Interpretation: Scaling Up Without Dumbing Down, In Proceedings of LEARNING LANGUAGE IN LOGIC (LLL) WORKSHOP, 1999
- [7] McRoy, S.W., Ali, S.S., Restificar, A., and Channarukul, S.: Building Intelligent Dialog Systems, to appear in Intelligence, 1(10), ACM Spring 1999
- [8] Waern, A.: What is an Intelligent Interface? <http://www.sics.se/~annika/papers/initint.html>