Controlled Natural Languages for Interface Agents

Tamás Mészáros, Tadeusz Dobrowiecki

Budapest University of Technology and Economics

Objective

Creating a easy-to-use framework for building natural language-based human-agent interfaces, where
– users can solve complex tasks using easy-to-understand controlled natural languages,
– programmers can create natural language interfaces without deep knowledge in NL processing and understanding.

Background

Computerized systems become more and more complex – their interfaces put greater cognitive load on the users.

Interface agents can play a significant role to help humans in using computerized systems. It is commonly required that such agents should employ **natural language communication** that **“relocates” the interface complexity to the language level**. Building a natural language processing system is, however, a very complex and difficult task for agent programmers.

Restrictions on the user interface (like using template- or menu-based communication) could significantly ease the implementation but they constrain considerably the expressiveness, the flexibility and the applicability.
Building free-form natural language communication with domain restrictions is also possible but it requires the application of very complex language tools (parsing, disambiguation, understanding the user's intention) and a detailed knowledge base.

Proposed approach

Authors propose to **design an interface agent around a controlled natural language**, which provides a good trade-off between interface restrictions and the complexity of general natural language understanding.

Controlled natural languages

Controlled natural language resembles the ordinary natural languages but it has a strict (and restricted) set of language rules, vocabulary and unambiguous meaning. These restrictions allow the successful processing of a controlled natural language by avoiding disambiguation and uncertain grammar rules.

In order to overcome the shortcomings of these languages (limited flexibility, difficult maintenance) authors propose special interfaces and **automatic language generation based on the conceptual model of the interface**.

Controlled interfaces

In order to coerce the user to follow the rules of the controlled language the interface agent monitors and corrects the user's input The agent also provides help to the user on how to use the restricted language by suggesting possible language constructs, words, or expressions.

Proposed workflow for creating a controlled language interface

Step 1. Conceptual modeling using Conceptual Graphs and GraphML

Artificial languages are not easy to create and maintain. In order to overcome this problem, authors propose a modeling framework that makes the design of a controlled natural language relatively easy. Programmers without any knowledge of natural language processing techniques should be able to create and maintain controlled language user interfaces. Language grammars and vocabularies should be automatically generated from the interface model.

A visual modeling framework is provided for this purpose that is based on Conceptual Graphs (CG).

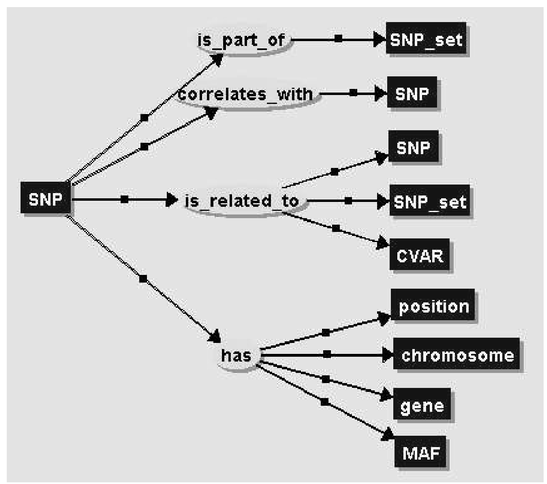
CGs are capable of representing the interface concepts and relations in an easy to understand way.

They have graphical editors and the CG format is an ISO standard.

CGs can also be extended to describe two other levels of the same model: the application (data) binding and the natural language.

In order to store these extensions in the model we applied the GraphML standard. (CG formats do not allow such extensions.)

GraphML also makes possible the utilization of general purpose graph editors and visualization tools.



Part of a concept model in CharGer (a CG editor)

Step 2. Binding the model to the application

The conceptual graph model is extended to describe the application (data) bindings.

At this level concepts defined in the model can specify relations to the data stored in the application. These relations can be used during language generation to extend the vocabulary and during the operation of the interface to process the user's input.

The extensions are stored inside GraphML `data` elements that are extensible according to the GraphML standard.

In their simplest form these bindings could link interface concepts to data types in the application (e.g. the type of an SNP is string).

These application bindings make it possible to define instantiation rules using data sources (e.g. gene names could be determined from a database).

The last node uses agent communication to query the available clinical variables (CVAR).

Several other types of application bindings could be described this way.

```
<node id="101">
  <desc>SNP</desc>
  <data>
     <type>xs:string</type>
  </data>
</node>

<node id="102">
  <desc>GENE</desc>
  <data>
     <source type="sql">
        SELECT gene_name FROM gene_table;
     </source>
  </data>
</node>

<node id="103">
  <desc>CVAR</desc>
  <data>
     <source type="acl">
        ... ACLMessage(QUERY, A2, "...") ...
     </source>
  </data>
</node>
```

Part of an extended graphML representation

Step 3. Generating the controlled natural language

From the conceptual model the controlled language can be generated in two steps:
– firstly, relations (and concepts) form the basis of the language, and the grammar,
– secondly, the vocabulary is constructed from concept names (language level) and their application bindings.

The type of generated language rules also depends on the nature of the application, e.g. different generation patterns are applied to query-type applications and to command-type applications.

For example (following the model above) in a query-like application:

From the “is_part_of” relation the algorithm generates language rules like “which SNP is part of an SNP_SET ...”. In addition to the query structures it also generates language rules for filtering
e.g. “... an SNP which is part of an SNP set ...”.

These rules will contain a basic vocabulary derived from the model.

This vocabulary can be extended using application bindings. Application data types can provide additional symbols for the language (e.g. numbers, enumerated data types, etc).

Bindings to data sources can extend the vocabulary by defining replacement patterns for nonterminal symbols (e.g. the GENE symbol could be replaced by any gene name determined by a database query).

```
<?xml version="1.0" encoding="utf8" ?>
<topic idname="genome_db">
<name lang="en">Genome Database Query</name>
<grammar type="context free" lang="en">
<rule symbol="QS"><alt>QW1 REL1</alt></rule>
<rule symbol="QW1"><alt>which</alt></rule>
<rule symbol="REL1">
   <alt>SNP IS_PART_OF SNP_SET</alt>
   <alt>SNP CORRELATES_WITH SNP</alt>
   <alt>SNP IS_RELATED_TO ALT1</alt>
</rule>
<rule symbol="SNP"><alt>snip</alt></rule>
<rule symbol="IS_PART_OF">
   <alt>is part of a</alt>
</rule>
<rule symbol="SNP_SET"><alt></alt>snip set</rule>
<rule symbol="IS_RELATED_TO">
   <alt>is related to a</alt>
</rule>
<rule symbol="GENE"
      alts="SELECT gene_name FROM gene_table;">
</rule>
<rule symbol="MAF">
   <alt>minor allele frequency {[1-9][0-9]*}</alt>
</rule>
...
</topic>
```

Part of a grammar showing symbols and vocabulary generation

Step 4. Interface assembly

The figure shows the general architecture of the controlled interface and an example Web implementation.
The operation is divided into two phases: controlling the user's input and parsing the complete input.

Considering this architecture, the complexity of the required components, and the user interface requirements, the controlled interface could be implemented as a standalone or as a client-server application as well.
The Interface Agent is deployed at client's side, the Proposer could run on client's or server's side (depending on the available resources, and the other components typically reside on a server which processes the user's input.

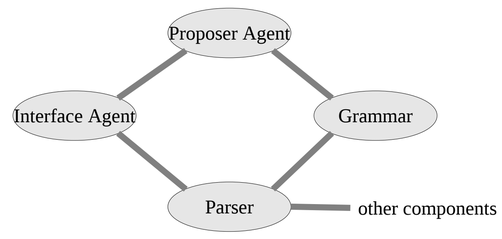
The Interface Agent is responsible for continuously monitoring the user's input and adjusting it to the rules of the controlled language. Several input methods could ensure this (e.g. predictive text input).

The Proposer Agent is helping the interface agent by providing text continuations, new menu items, etc. Its operation depends on the selected input method.

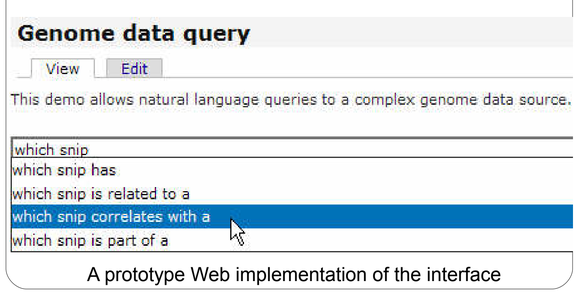
The Grammar module serves grammar rules and vocabulary to other components. It should be implemented in a way that it supports both the Parser and the Proposer (the later in real-time).

The Parser processes the complete input and provides a parse tree (e.g. for query processing).

The figure to the right demonstrates a Web-based interface using predictive text input and displaying a list of possible query continuations.



General architecture of the interface



A prototype Web implementation of the interface

Prototype implementations

Prototypes were built for information access applications and for controlled text authoring.

Querying information systems is a difficult task for humans because complex relations and query structures are hard to formulate using conventional user interfaces. Users with little technical experience are not able to use these interfaces. We are experimenting with new, controlled language interfaces in linguistic and biological applications where users are overwhelmed by the amount of available information and the required query structures.

The third prototype was built to help users in creating curriculum vitae documents in a controlled (standardized) way. Writing a Europass CV or European Mobility Document in an appropriate language requires the knowledge of the desirable format and the available competencies (and their proper translations). An interface agent could help in forming the sentences and selecting the right words for describing the user's competencies in a given language.

Future perspectives

Current CG tools do not allow the required application and language extensions. A **graphical framework** is being developed for modeling the conceptual, language and application binding levels. This framework also allows the generation and customization of language rules.

Generating the conceptual model of the interface from an application model (e.g. agent models, database models, etc.) using automated **model transformation techniques** could further ease the interface programming.

Mobile and embedded systems with limited display capabilities can also benefit from using controlled natural language interfaces. A prototype interface for the Android platform is under development to experiment with such systems.

Summary

Authors proposed an approach to effectively utilize controlled natural languages in human-agent interfaces.
The approach is based on the idea of using automatically generated application-specific controlled natural languages in restricted user interfaces.

This approach allows human-agent communication in (controlled) natural languages, and it facilitates the automatic generation and maintenance of the applied controlled language which eases the implementation of such interfaces even for programmers without deep knowledge in natural language processing.

More information & online demonstrations: http://project.mit.bme.hu/clif/

Contact

Tamás Mészáros
assistant professor

Department of Measurement and Information Systems, BUTE
1117 Budapest, Magyar tudósok körútja 2. Bldg. I. Room E437
Phone: +36 1 463-2899, +36 1 463-2057 Fax: +36 1 463-4112
email: meszaros@mit.bme.hu