ESPRIT CompulogNet Area Meeting on Program Development, Utrecht, Netherlands, September 1995

Attempto

Specifications in Controlled Natural Language

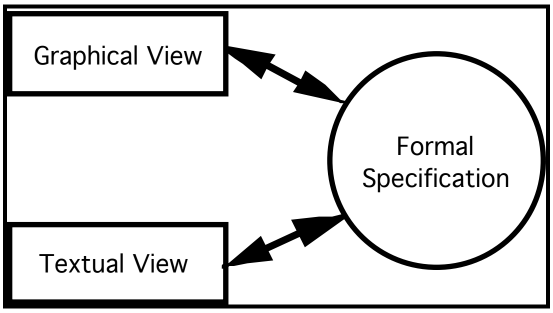
Extended Abstract in 10 Pictures

Norbert E. Fuchs, Rolf Schwitter
Department of Computer Science, University of Zurich
{fuchs, schwitter}@ifi.unizh.ch

Writing specifications for computer programs is not easy since one has to take into account the disparate conceptual worlds of the application domain and of software development. To bridge this conceptual gap we propose controlled natural language as a declarative and application-specific specification language. Controlled natural language is a subset of natural language that can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage by non-specialists. Specifications in controlled natural language are automatically translated into Prolog clauses, hence become formal and executable. The translation uses a definite clause grammar (DCG) enhanced by feature structures. Inter-text references of the specification, e.g. anaphora, are resolved with the help of discourse representation theory (DRT). The generated Prolog clauses are added to a knowledge base. We have implemented the prototypical specification system Attempto that successfully processes the specification of a simple automated teller machine.

Views

To bridge the conceptual gap between an application domain and the formal specification of a problem of that domain we introduce graphical and textual views as application-oriented specifications.



An automatic mapping between a view and its associated formal specification assigns a formal semantics to the view. Though views give the impression of being informal and having no intrinsic meaning, they are formal and have the semantics of their associated formal specification.

Controlled Natural Language

Controlled natural language – a subset of natural language with restricted grammar and an application-specific vocabulary – can serve as a view for a formal specification in a logic language.

A specification is a multi-sentential text consisting of

- **simple declarative sentences of the form subject – predicate – object**
- ***if-then* sentences**
- ***yes/no* queries, *wh*-queries**

The specification texts can contain

- **anaphoric references**
- **relative clauses, both subject and object modifying**
- **comparative clauses like *bigger than*, *smaller than* and *equal to***
- **elliptical compound phrases like *and*-lists, *or*-lists**
- **negation like *does not, is not* and *has not***

Example Specification: SimpleMat

The following is a small excerpt of the controlled natural language specification of a simple automated teller machine called SimpleMat.

```
The customer enters a card and a personal code.

If the personal code is not valid then SimpleMat rejects the card.
```

The specification text uses

- **declarative and *if-then* sentences**
- **ellipsis**
- **anaphoric reference**
- **negation**
- **compound nouns, e.g.** `personal code`

Parsing

The specification text is parsed by a top-down parser using a Definite Clause Grammar enhanced by feature structures.

The following graph shows one of the s-nodes of the syntax tree.

```
                              s
         |--------------------|--------------------|
         s1                                        s1
|--------|--------|                         |------|------|
cj                s                         adv           s
|     |-----------|-----------|             |     |-------|-------|
|     np                      vp            |     np              vp
|   |-|--|              |-----|-|----|      |     |          |----|-----|
|   det  n1             copula  neg  ap     |     pn         tv         np
|   |    |              |       |    |      |     |          |        |-|--|
|   |    nn             |       |    a      |     |          |        det  n1
|   |    |              |       |    |      |     |          |        |    |
|   |    |              |       |    |      |     |          |        |    cn
|   |    |              |       |    |      |     |          |        |    |
if  the   personal_code  is         not  valid   then   simplemat  rejects   the
card.
```

Semantic Representation

The specification text is translated into a discourse representation structure which contains discourse referents representing the objects of the discourse, and conditions for these discourse referents.

```
[A, B, C, D]
customer(A)
card(B)
enter(A, B)
personal_code(C)
enter(A, C)
named(D, simplemat)
IF:
  []
  personal_code(C)
  NOT:
    []
    valid(C)
THEN:
  []
  card(B)
  reject(D, B)
```

Translation into Prolog

Finally, the discourse representation structure is translated into Prolog clauses which are asserted to a knowledge base.

```
customer(1).
card(2).
enter(1, 2).
personal_code(3).
enter(1, 3).
named(4, simplemat).
reject(4, 2) :-
  personal_code(3),
  neg(valid(3)).
```

Discourse referents – which are existentially quantified variables - are replaced by Skolem constants and functions.

Implications with disjunctive consequences are replaced by sets of Prolog clauses, one for each disjunct.

Information for the User

A paraphrased text – displaying all substitutions and interpretations – explains how the system interpreted the user's input.

```
the customer enters a card and the customer [same object] enters
[same predicator] a personal_code. if the personal_code [same
object] is not valid then simplemat rejects the card [same
object].
```

The system informs the user about the processing time, and about spelling and parsing errors, e.g. if `this` and `checked` are unknown

```
The customer enters a card. This card is checked for validity.
```

After parsing the first sentence successfully the system replies

```
Unparsable Sentence:     this card is checked for validity.
Unknown words:           this      checked
```

Further Features of the Attempto System

Questions (*yes/no* and *wh*-queries) can be used to interrogate the knowledge base. Questions are translated into Prolog queries, and answered by logical inference.

Formal specifications can – at least partially – be retranslated into their equivalent controlled natural language text.

Formal specifications in the form of discourse representation structures, or equivalent Prolog clauses, can be executed, and thus serve as a prototype of the specified system.

A lexical editor – exhibiting interfaces for non-experts and for experts – allows users to modify and to extend the lexicon while the systems parses the specification text.

A spelling checker allows users to determine whether all words of a specification text are known to the system.

References

M. A. Covington, D. Nute, N. Schmitz, D. Goodman, From English to Prolog via Discourse Representation Theory, Research Report 01-0024, Artificial Intelligence Programs, University of Georgia, 1988

N. E. Fuchs, R. Schwitter, Specifying Logic Programs in Controlled Natural Language, CLNLP 95, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 3-5, 1995

Y. Ishihara, H. Seki, T. Kasami, A Translation Method from Natural Language Specifications into Formal Specifications Using Contextual Dependencies, in: Proceedings of IEEE International Symposium on Requirements Engineering, 4-6 Jan. 1993, San Diego, IEEE Computer Society Press, pp. 232 - 239, 1992

H. Kamp, U. Reyle, From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer Academic Publishers, Dordrecht, 1993

B. Macias, S. Pulman, Natural Language Processing for Requirements Specifications, in: F. Redmill, T. Anderson (eds.), Safety-Critical Systems, Current Issues, Techniques and Standards, Chapman & Hall, pp. 67-89, 1993