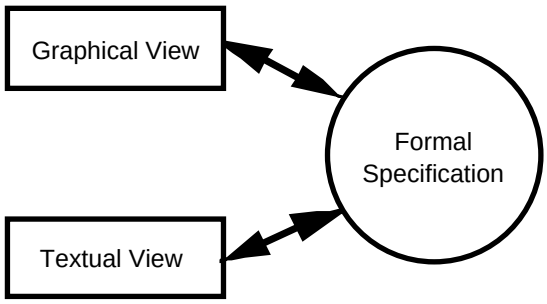
Attempto
Controlled Natural Language for Requirements Specifications

Norbert E. Fuchs, Rolf Schwitter
Department of Computer Science, University of Zurich
CH-8057 Zurich, Switzerland
{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.

1 Introduction: Views as Declarative Specifications

We develop formal specifications in logic languages, specifically first-order predicate logic and Prolog. To bridge the conceptual gap between application domains and formal specifications we introduce graphical and textual views of formal specifications as application-oriented, i.e. in the true sense declarative, specifications [Fuchs & Fromherz 94].



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. This dual-faced appearance of views reduces the conceptual gap, and eases the transition from informal to formal notations. Furthermore, if the formal specification is executable its execution can be observed on the level of the view. Thus validation and prototyping in concepts close to the application domain become possible.

In previous publications we presented graphical views [Fuchs & Fromherz 94], while this paper introduces controlled natural language as a textual view of a formal specification. The paper is structured as follows: in section 2 we define our version of controlled natural language; in section 3 we give an overview of the Attempto specification system; sections 4 and 5 describe the translation process from controlled natural language to Prolog; finally, in section 6 we conclude and outline further research.

2 Controlled Natural Language as a Textual View of a Logic Program

2.1 Controlled Natural Language

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

language. Our version of controlled natural language can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage. It has a principled structure: declarative sentences are combined by constructors (e.g. negation, if-then, and-lists, or-lists) to powerful phrases, while restricted forms of anaphora and ellipsis render the language concise and natural.

A specification in controlled natural language is a multisentential text consisting of

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

The specification text can contain

- anaphoric references, e.g. pronouns
- relative phrases, both subject and object modifying
- comparative phrases 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*

Controlled or simplified English has been used for technical documentation [Wojcik et al. 90], and as data base query language [Androutsopoulos 95]. Pulman and Rayner are suggesting a computer processable controlled language that could be used for various purposes ranging from structured documentation over access to information to the control of devices [Pulman & Rayner 94]. However, very few researchers have tried to employ controlled natural language for software specifications since this leads to additional syntactic and semantic constraints for the language especially if one requires the specifications to be executable [Ishihara et al. 92, Macias & Pulman 92, Fuchs & Schwitter 95].

Users seem to be able to construct sentences in controlled natural language, and to avoid constructions that fall outside the bounds of the language, particularly when the system gives feedback of the analysed sentences in a paraphrased form using the same controlled language [Capindale & Crawford 89].

2.2 Example Specification in Controlled Natural Language

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 numeric personal code.
If it is not valid then SM rejects the card.
```

The example specification text employs

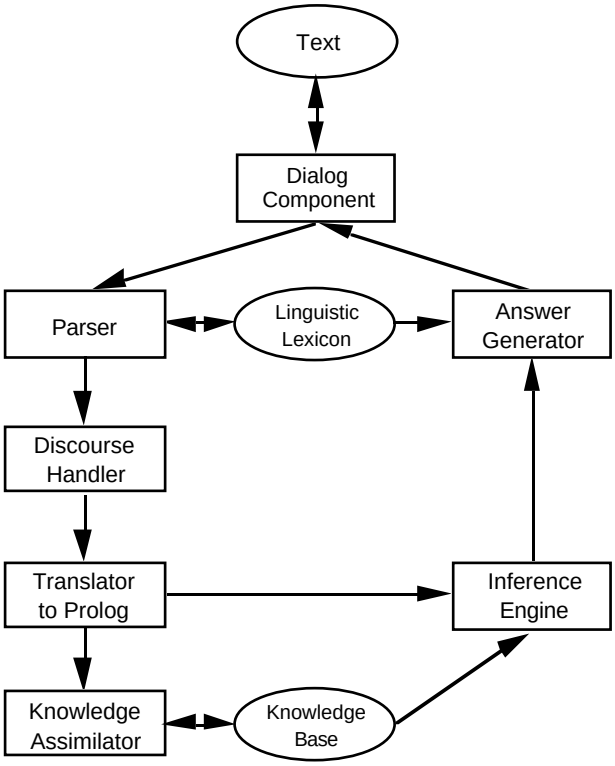
- declarative and *if-then* sentences
- elliptical compound phrases
- compound nouns, e.g. `personal code`
- anaphoric reference via the pronoun `it`
- negation
- abbreviations (`SM` standing for the name SimpleMat)

3 Overview of Attempto

3.1 Translation Components

The Attempto system translates specifications in controlled natural language into discourse representation structures, and then into Prolog. Here, we briefly describe Attempto's components.

The user enters specification text in controlled natural language that the *Dialog Component* forwards to the parser in tokenised form. Parsing errors and ambiguities to be resolved by the user are reported back by the dialog component. The user can also query the knowledge base in controlled natural language. The *Parser* uses a predefined Definite Clause Grammar enhanced by feature structures and a predefined linguistic lexicon to check sentences for syntactical correctness, and to generate syntax trees and sets of nested discourse representation structures. The *Linguistic Lexicon* contains an application-specific vocabulary. The lexicon can be modified by a lexical editor invokable from the dialog component. The *Discourse Handler* analyses and resolves inter-text references and updates the discourse representation structures generated by the parser. The *Translator* translates discourse representation structures into Prolog clauses.



These Prolog clauses are passed to the knowledge assimilator, or – in case of queries – to the inference engine. The *Knowledge Assimilator* adds new knowledge to the knowledge base. The *Inference Engine* answers user queries with the help of the knowledge base. In a preliminary version the inference engine is just the Prolog interpreter. The *Answer Generator* takes the answers of the inference engine, reformulates them in controlled natural language, and forwards them to the dialog component.

3.2 Lexical Editor and Spelling Checker

Specification texts are incrementally developed by domain specialists. Though Attempto's lexicon contains entries of the closed word classes, e.g. determiners, prepositions, and conjunctions, the entries for domain specific subsets of the open word classes, e.g. nouns and verbs, have to be added as needed for the specification text. A lexical editor – exhibiting interfaces for linguistic experts and non-experts – allows users to interactively modify and extend the lexicon while the system parses the specification text.

The expert interface represents lexical entries as complete feature structures and allows experts to freely modify any lexical entry. The interface for non-experts employs templates that help users with minimal linguistic understanding to enter information. Help texts and balloon help support both groups of users.

A spelling checker allows users to determine whether all words of a specification text are in the lexicon. This spelling checker is invoked automatically if (part of) a specification text cannot be parsed.

4 Parsing

The specification text is parsed by a top-down parser using a Definite Clause Grammar enhanced by feature structures in GULP notation [Covington 94]. The parser generates a syntax tree as syntactic representation, and concurrently a discourse representation structure as semantic representation.

In addition, the parser generates a paraphrase – displaying all substitutions and interpretations made – that explains how Attempto interpreted the user's input, e.g. our example specification.

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

The user can now decide to accept Attempto's interpretation, or to rephrase the input to achieve another interpretation. For ambiguous input Attempto always suggests one standard interpretation as default. It is up to the user to reformulate the input to achieve non-standard interpretations.

In addition, the parser informs the user about spelling and parsing errors. If the user enters

```
The customer enters a card. SimpleMat checks it.
```

the parser processes the first sentence successfully, then finds the unknown word `checks` which renders the second sentence unparsable, and replies

```
First unparsable sentence:   simplemat checks it.
Unknown word:                checks
```

With the help of the lexical editor the user can add the unknown word to the lexicon and immediately resubmit the input to the parser .

5 Semantic Translation

5.1 Contextual Semantic Translation

The specification text is translated into a discourse representation structure (DRS) which contains discourse referents representing the objects of the discourse, and conditions for these discourse referents [Kamp & Reyle 93].

Each sentence is translated in the context of the preceding sentences, yielding for our example the DRS

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

A DRS is a semantic representation of the input text, and is considered true if the input describes a section of reality.

Conditions of a DRS can be simple or complex, i.e. again a DRS. This can lead to nested DRSs. In our case, the topmost DRS contains an `IF-THEN` sub-DRS which itself contains a `NOT` sub-DRS.

Anaphoric references, e.g. the pronoun `it` of the second sentence referring to the compound noun `personal code` of the first sentence, are automatically resolved. The resolution algorithm picks the closest referent in a superordinate DRS that agrees in gender and number.

5.2 Translation into Prolog

Finally, the discourse representation structure is translated into Prolog clauses which are asserted as `fact/1` to the knowledge base.

```
fact(customer(0)).
fact(card(1)).
fact(enter(0, 1)).
fact(numeric(2)).
fact(personal_code(2)).
fact(enter(0, 2)).
fact(named(3, simplemat)).
fact((reject(3, 1):-neg(valid(2)))).
```

Discourse referents – being existentially quantified variables – are replaced by Skolem constants, or – if they are in the scope of a universal quantor – by Skolem functions.

`IF-THEN` DRSs with disjunctive consequences are represented by sets of Prolog clauses to avoid disjunctive clauses.

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

6 Conclusions and Further Research

The present prototypical implementation of Attempto proves that controlled natural language can be used for the non-trivial specification of an automated teller machine, and that the specification can be translated as coherent text into Prolog clauses. Much more work needs to be done, however.

6.1 Controlled Natural Language

Our current version of controlled English was derived in an attempt to represent typical constructs in natural language specifications in a structured and concise way. It seems that other researchers have chosen similar ad hoc approaches to define their versions of controlled natural languages. However, a more systematic definition of controlled English based on a small number of easily remembered principles has to be found.

6.2 Retranslation

To hide the internal representation of a formal specification it will be retranslated into controlled natural language when the user wants to examine or query the knowledge base. Formal specifications in the form of a DRS can – at least partially – be retranslated into their equivalent controlled natural language text since the grammar of the Attempto system is reversible. Another approach would use predefined translation schemata with variable parts. During the retranslation the lexicon is accessed to instantiate these variable parts.

6.3 Executing the Specification

The internal representation of a specification can be used for simulation or prototyping by executing it. In our example specification, this means executing/running the specification of the automated teller machine. However, the specification does not yet provide all the necessary information and needs to be enhanced in three ways. First, an order of events has to be established. In our approach based on discourse representation theory the order of events is to a great extent established when we introduce eventualities (events and states) into the processing of our controlled natural language. Second, many relations representing events are not only truth-functional, but also cause side-effects, e.g. I/O operations. The required side-effects can be defined by interface predicates that depend on the simulation environment. Third, the execution needs some situation specific information, or scaffolding. We can either provide the relevant facts in the knowledge base, or more conveniently, get the information by querying the user.

References

[Androutsopoulos 95] I. Androutsopoulos, G. D. Ritchie, P. Thanisch, Natural Language Interfaces to Databases – An Introduction, Journal of Natural Language Engineering, vol 1, no. 1, Cambridge University Press, 1995

[Capindale & Crawford 89] R. A. Capindale, R. G. Crawford, Using a natural language interface with casual users, International Journal Man-Machine Studies, 32, pp. 341-362, 1989

[Covington 94] M. A. Covington, GULP 3.1: An Extension of Prolog for Unification-Based Grammar, Report AI-1994-06, Artificial Intelligence Center, University of Georgia, 1994

[Fuchs & Fromherz 94] N. E. Fuchs, M. P. J. Fromherz, Transformational Development of Logic Programs from Executable Specifications, in C. Beckstein, U. Geske (eds.), Entwicklung, Test und Wartung deklarativer KI-Programme, GMD Studien Nr. 238, 1994

[Fuchs & Schwitter 95] N. E. Fuchs, R. Schwitter, Specifying Logic Programs in Controlled Natural Language, CLNLP 95, Workshop on Computational Logic for Natural Language Processing, Edinburgh, 1995

[Ishihara et al. 92] 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

[Kamp & Reyle 93] 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

[Macias & Pulman 92] 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

[Pulman & Rayner 94] S. Pulman, M. Rayner, Computer Processable Controlled Language, SRI International Cambridge Computer Science Research Centre, 1994

[Wojcik et al. 90] R. H. Wojcik, J. E. Hoard, K. C. Holzhauser, The Boeing Simplified English Checker, Proc. Internatl. Conf. Human Machine Interaction and Artificial Intelligence in Aeronautics and Space, Centre d'Etude et de Recherche de Toulouse, pp. 43-57, 1990