N. E. FUCHS, U. GESKE (EDS.) PROCEEDINGS OF THE POSTER SESSION AT JICSLP'96, BONN, GERMANY, SEPTEMBER, 1996, GMD STUDIEN NR. 296, 1996

Attempto Controlled English (ACE)

A Seemingly Informal Bridgehead in Formal Territory

Extended Abstract

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

Attempto Controlled English (ACE) – a subset of English with a restricted grammar and a domain-specific vocabulary – allows domain specialists to interactively formulate requirements specifications in domain concepts. ACE can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage. ACE has a principled structure: simple sentences are combined by constructors (e.g. negation, if-then, and-lists, or-lists) to powerful composite sentences while certain forms of anaphora and ellipsis render the language concise and natural. We have developed the Attempto system that unambiguously translates complete ACE specifications into discourse representation structures – a structured form of first-order predicate logic – and optionally into Prolog. Translated specification texts are incrementally added to a knowledge base. This knowledge base can be used to answer queries in ACE about the specification, and it can be executed for simulation, prototyping and validation of the specification. Tools like a paraphraser, a lexical editor, a spelling checker, and a metainterpreter for query answering and execution complement Attempto. Using Attempto we have successfully processed the non-trivial specification of an automated teller machine.

Views of Formal Specifications

While we believe that specifications should be formal to support verification and executable to aid prototyping and validation, we have to realise that domain specialists normally express requirements and specifications in domain-specific concepts using informal notations, and furthermore that domain specialists may not be familiar with formal specification methods. Thus there are two problems: a semantic gap between domain concepts and the concepts of formal methods, and the lack of acceptability of formal methods by domain specialists.

We propose to solve both problems by introducing graphical and textual views of formal specifications, and by mapping views to formal specifications, and vice versa. Mapping a view to its associated formal specification assigns a formal semantics to the view, while the inverse mapping paraphrases a formal specification in the language of the view. Thus views may seem to be informal, but are in reality formal and have the semantics of their associated formal specification.

The semantically equivalent representations bridge the semantic gap between the different conceptual worlds of the domain specialist and the software developer, and help to make formal methods acceptable.

Attempto Controlled English (ACE)

We introduce Attempto Controlled English as a textual view of a formal specification in a logic language. ACE is a computer processable subset of English specifically designed to write requirements specifications. ACE combines the familiarity of natural language with the rigor of formal specification languages and can be unambiguously translated into an executable logic language. The vocabulary of ACE comprises the usual function word classes (determiners, conjunctions, prepositions etc.) and an application-specific subset of content word classes (nouns, verbs, adjectives, adverbs). Currently, the syntax of ACE allows users to specify systems that can be modelled by finite-state machines, but the syntax can be easily extended to express concurrency.

A specification is an ACE text consisting of paragraphs. A paragraph is the basic unit of translation and consists of sentences that can be anaphorically interrelated. Sentences come as simple sentences, composite sentences, and interrogative sentences. Simple sentences have the form *subject + predicator (+ complement + adjunct)*, and express a true state of affairs. Composite sentences are built from simpler sentences with the help of constructors: coordination (*and*, *or*, *either-or*), subordination (*if-then*, *who/which/that*), and negation (*not*). Interrogative sentences allow users to pose *yes/no* questions and *wh*-questions.

Sentences can contain subject and object modifying relative sentences, anaphoric references (e.g. personal pronouns), coordination between equal constituents (e.g. *and*, *or*), ellipsis as reduction of coordination, noun phrase negation (*no X*), verb phrase negation (*does not*, *is not*), and synonyms and abbreviations.

Here is a short specification example derived from the two page ACE specification of an automated teller machine:

```
The customer enters a card and a numeric personal code.
If it is not valid then SM rejects the card.
```

The example employs composite sentences built with the constructors *and*, *if-then* and *not*, ellipsis, compound nouns (*personal code*), anaphoric references via the pronoun *it* and the definite noun phrase *the card*, and abbreviations (*SM* standing for SimpleMat).

ACE Principles

To make ACE easily learned and remembered, it is based on a small number of syntactic and semantic principles. Furthermore, ambiguity is handled in a principled way.

Syntactic Principles

- nouns are always used with a determiner

  ```
  The customer enters a card.
  ```

- coordination is only possible between equal constituents

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

- prepositional adjuncts relate to the verb

  ```
  The customer { enters a card with a code }.
  ```

- relative sentences relate to the right-most noun phrase

  ```
  The customer enters { a card that has ... }.
  ```

- anaphoric references are resolved on the basis of genus, numerus, and recency

  ```
  The machine checks the personal code.
  If it [personal code] is valid ...
  ```

- there is only syntactic ellipsis

  ```
  The customer enters a card and [] a code.
  If the code is valid then ... If not [the code is not valid] then ...
  ```

- verbs are only used in indicative mood and active voice, only in simple present tense, and only in third person singular or plural

Semantic Principles

- verbs denote events and states
- adverbial phrases qualify the event denoted by the verb

  ```
  The customer enters the card slowly into the slot.
  ```

- textual order of verbs determines the temporal order of the associated events

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

- modal verbs (*may*, *can*, *must* etc.) and intensional verbs (*believe*, *suggest etc.*) are not allowed
- modal adverbs (*possibly*, *probably* etc.) are not available
- (pn | the cn) + copula + (pn | the cn) denotes identity

```
John is the manager.
```

- np + copula + (a n1) denotes an attribute

```
John is a customer.
```

- negated noun phrase

```
No customer enters a card.
--- scope of negation ---
```

- negated verb phrase

```
The customer does not have a personal code.
                       - scope of negation -
```

Disambiguation

- some ambiguous constructs are not available

```
John and Mary enter a card.
```

- unambiguous alternatives with scope markers (*together*, *each*) can be used instead

```
John and Mary enter a card together.
John and Mary enter a card each.
```

- not all ambiguous constructs can be eliminated since this would render ACE unnatural, but the deterministic parsing on the basis of syntactic principles induces unique interpretations which are reflected to the user

```
Input:  The customer enters a card with a code.
Output: The customer { enters a card with a code }.
```

- the user either accepts the interpretation, or rephrases the input to change the interpretation, e.g.

```
The customer enters a card that has a code.
The customer enters a card and a code.
```

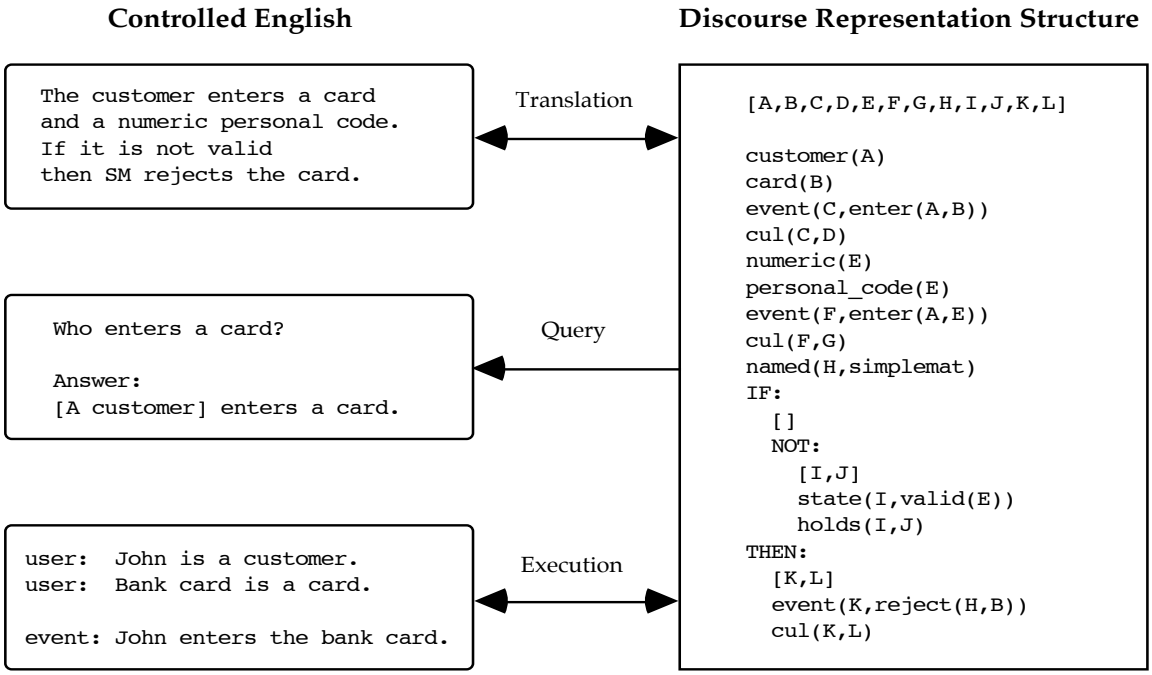
Using the Attempto System

The Attempto system accepts specifications in ACE and translates them paragraph by paragraph into Discourse Representation Structures – a structured form of first-order predicate logic. In a further step, the specification can be translated into Prolog. Translated specifications can be queried in ACE, and can be executed for prototyping and validation.

The following figure shows the translation of the example specification and the generated discourse representation structure which consists of a list of referents [A, B, C, ...] – standing for the objects of discourse – and conditions for these referents.

The figure also shows the sample query *Who enters a card?* and the generated answer *[A customer] enters a card.*

Finally, the figure gives an excerpt of the execution trace of the example specification. The contextual input (*John*, *bank card* etc.) is provided by the user while the actions associated with events are built into the execution environment.



Background References

N. E. Fuchs, R. Schwitter, Attempto Controlled English (ACE), CLAW 96, First International Workshop on Controlled Language Applications, University of Leuven, Belgium, March 1996

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

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

R. Nelken, N. Francez, Automatic Translation of Natural Language System Specifications into Temporal Logic, Technical Report, Computer Science Department, The Technion, Haifa, Israel, 1995

R. Schwitter, N. E. Fuchs, Attempto – From Specifications in Controlled Natural Language towards Executable Specifications, GI EMISA Workshop, Natürlichsprachlicher Entwurf von Informations-systemen, Tutzing, Germany, May 1996