Attempto Controlled English (ACE) for Software Specifications

Rolf Schwitter
Norbert E. Fuchs
Uta Schwertel

Department of Computer Science
University of Zurich

{schwitter, fuchs, uschwert}@ifi.unizh.ch
http://www.ifi.unizh.ch/~fuchs/
http://www.ifi.unizh.ch/~schwitter/

Motivation

Specifications

- specifications state the properties or constraints which a software system must satisfy to solve a problem
- in which notation should specifications be expressed?

Assumptions

- specifications should be formal to support verification
- specifications should be executable to aid prototyping and validation

Reality

- domain specialists normally express requirements and specifications in domain-specific concepts using informal notations
- domain specialists may not be familiar with formal specification methods

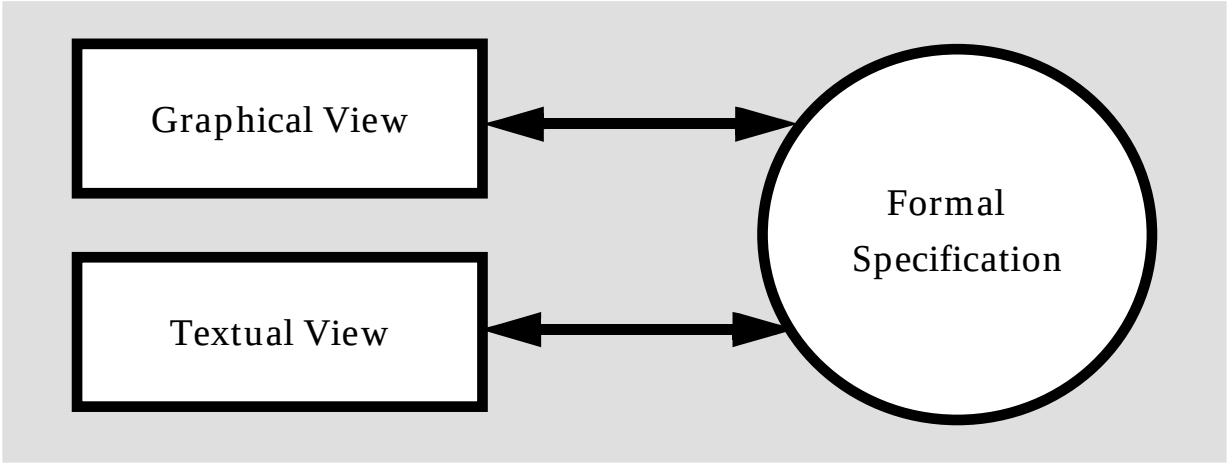
Problems

- semantic gap between domain concepts and concepts of formal methods
- acceptability of formal methods by domain specialists

Bridging the Conceptual Gap

Solution for both problems

- graphical and textual views of formal specifications



- bi-directional mapping between view and formal specification
- mapping of view to formal specification – in a logic language – assigns a formal semantics to the view
- views seem to be informal, but are in fact formal and have the semantics of their associated formal specification
- semantically equivalent representations bridge the gap between the different conceptual worlds of the domain specialist and the software developer

ACE as a Textual View

Natural language as textual view

- long tradition
- ambiguity, vagueness and incompleteness of full natural language impede unique mapping to formal specification
- controlled natural language solves deficiencies

Attempto Controlled English (ACE) has three components

- a vocabulary with predefined function words (e.g. *she*, *the*, *and*, *if*, *not*) and application-specific content words (e.g. *customer*, *enter*, *valid*, *manually*)
- a restricted grammar of English
- a small set of principles that show users how to construct and to interpret ACE texts

ACE has desired properties

- unambiguously translatable into an executable logic language
- combines familiarity of natural language with rigor of formal specification languages

The Language ACE

- a specification is an ACE text consisting of paragraphs
- paragraphs contain one or more sentences that can be anaphorically interrelated
- each paragraph is translated unambiguously into one Discourse Representation Structure – a structured variant of first order predicate logic – and optionally into Prolog
- sentences

  simple sentences
  composite sentences
  interrogative sentences
- simple sentences

  subject + verb + complement { + adjunct }
- composite sentences built from simpler sentences with the help of constructors

  coordination (*and*, *or*, *either-or*)
  subordination (*if-then*, *who* / *which* / *that*)
  negation (*not*, *no*)
  quantification (*each*, *every*, *for every*, *there is a*)
- interrogative sentences

  yes / *no*-questions
  wh-questions

Characteristics of ACE

- sentences can contain
  - subject and object modifying relative sentences
  - syntactic ellipsis as reduction of coordination
  - anaphoric references, e.g. personal pronouns
  - coordination between equal constituents, e.g. *and*, *or*
  - noun phrase negation, *no X*
  - verb phrase negation, *does not*, *is not*
  - synonyms and abbreviations
- verbs
  - denote events and states
  - only used in indicative mood and active voice
  - only simple present tense
  - only third person singular and plural
  - no modal verbs
- specification example in ACE

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

- example employs
  - composite sentences built with `and`, `if-then` and `not`
  - compound nouns, e.g. `personal code`
  - anaphoric references, e.g. `it → a numeric personal code` and `the card → a card`
  - syntactic ellipsis, e.g. `… and [enters] a numeric personal code.`
  - abbreviations (`SM` standing for `Simplemat`)

Feedback for the User

Paraphrase in ACE

- shows all substitutions and interpretations

  Input:

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

  Paraphrase:

  ```
  The customer enters a card and [enters] a numeric per-
  sonal code.
  If [the numeric personal code] is not valid then
  [Simplemat] rejects [the card].
  ```

- shows parsing principles

  Input:

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

  Paraphrase:

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

- paraphrase is a valid ACE text after removal of parentheses

Using the Attempto System

The Discourse Representation Structure can be

- used to answer queries in ACE about the specification
- executed for simulation, prototyping, and validation of the specification

Execution of the specification asks for

- definition of side effects for events (query the engineer)
- unknown information about the situation (query the user)

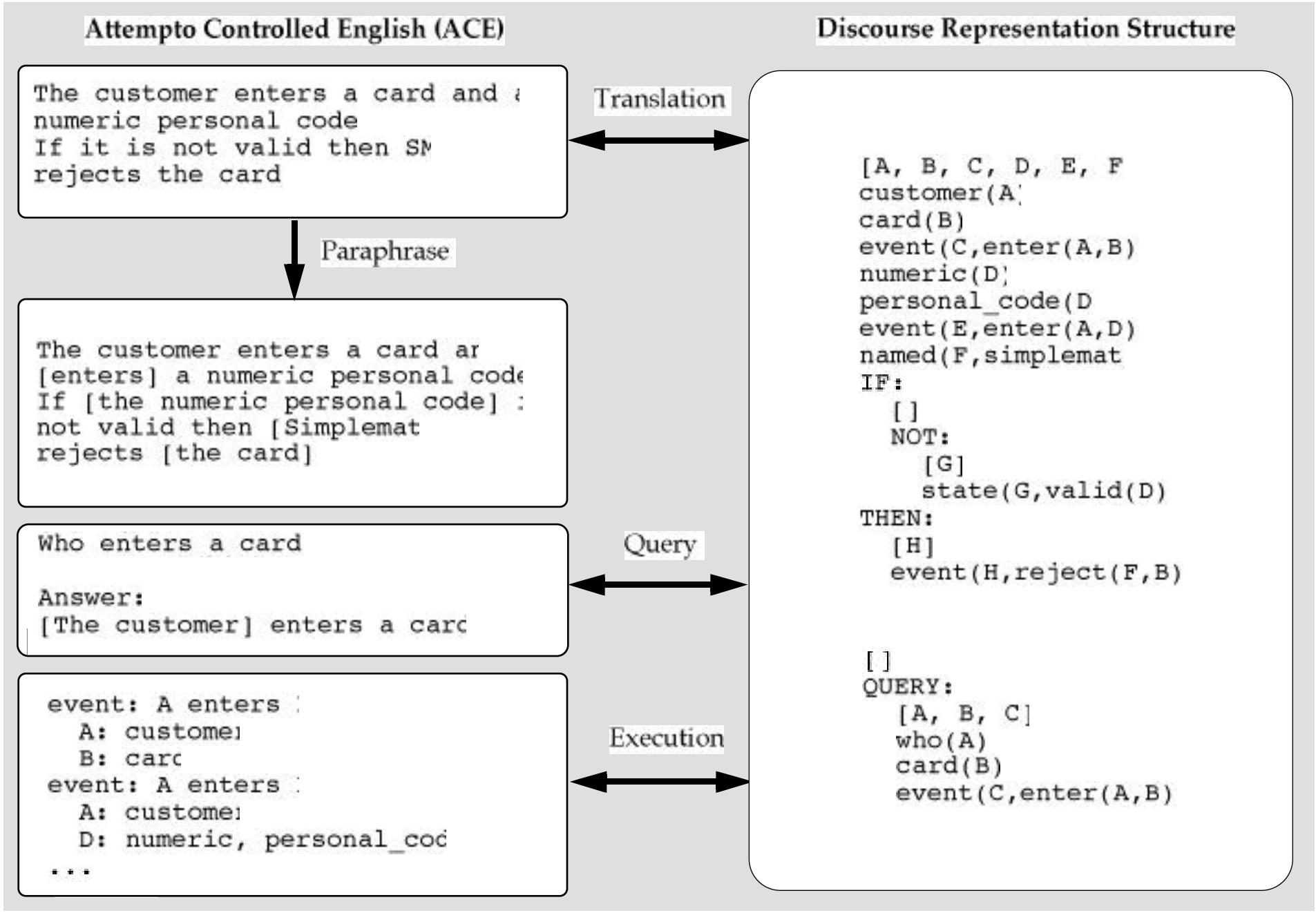
References

N. E. Fuchs, U. Schwertel, R. Schwitter, Attempto Controlled English (ACE), Language Manual, Version 2.0, Institut für Informatik, Universität Zürich, 1998

N. E. Fuchs, U. Schwertel, R. Schwitter, Attempto Controlled English — Not Just Another Logic Specification Language, LOPSTR'98, Manchester (to appear)

U. Schwertel, R. Schwitter, N. E. Fuchs, Attempto Controlled English, How to Specify Things with Words, submitted to 22. Jahrestagung Künstliche Intelligenz (KI-98), Bremen, 1998

R. Schwitter, Kontrolliertes Englisch für Anforderungspezifikationen, Dissertation, Universität Zürich, 1998


Attempto Controlled English (ACE)
Discourse Representation Structure
The customer enters a card and
numeric personal code
If it is not valid then SM
rejects the card
Translation
Paraphrase
The customer enters a card ar
[enters] a numeric personal cod
If [the numeric personal code]
not valid then [Simplemat
rejects [the card]
Who enters a card
Answer:
[The customer] enters a carc
Query
event: A enters
A: custome
B: carc
event: A enters
A: custome
D: numeric, personal_coc
...
Execution
[A, B, C, D, E, F
customer(A
card(B)
event(C,enter(A,B)
numeric(D
personal_code(D
event(E,enter(A,D)
named(F,simplemat
IF:
[]
NOT:
[G]
state(G,valid(D)
THEN:
[H]
event(H,reject(F,B)
[]
QUERY:
[A, B, C]
who(A)
card(B)
event(C,enter(A,B)