ECOLE: A Look-ahead Editor for a Controlled Language

Rolf Schwitter, Anna Ljungberg, David Hood
Centre for Language Technology
Macquarie University
Sydney, NSW 2109, Australia
{schwitt|anna|dhood}@ics.mq.edu.au

Abstract

This paper presents ECOLE, a look-ahead text editor that supports authors writing seemingly informal specifications in PENG, a computer-processable controlled natural language. ECOLE communicates via a socket interface with the controlled language processor of the PENG system. After each word form entered the look-ahead editor displays appropriate look-ahead categories. These syntactic hints tell the author what kind of word or syntactic structure can follow the current input string and reduce thereby the cognitive burden to learn and remember the controlled language. While the author types the text word by word and adds unknown content words on the fly to the lexicon, a discourse representation structure and a paraphrase is built up dynamically for the text in a completely compositional manner. The arising specification can be checked automatically for consistency and informativity with the help of third-party reasoning services.

1 Introduction

Controlled natural languages are - more or less - well-defined subsets of natural languages that have been restricted with respect to their grammar and lexicon. Grammatical restrictions result in less complex and less ambiguous texts. Lexical restrictions reduce the size of the vocabulary and the meaning of the lexical entries for a particular application domain. In general, these restrictions improve the readability and the processability of a document. However, different tasks need different types of controlled languages. For example, controlled languages for technical documentation require good readability (AECMA, 2001), while controlled languages for machine translation focus on processability and therefore have other restrictions (Mitamura and Nyberg, 2001). Controlled natural languages not only make life a lot easier with technical documents and machine translation, but they can also improve the whole knowledge acquisition process for any kind of intelligent system (Sowa, 2002).

It is well known that writing documents in a controlled natural language can be a slow and painful process, since it is hard to write documents that have to comply with the rules of a controlled language (Goyvaerts, 1996; Huijsen, 1998). One of the deciding factors for the acceptability of a controlled language is the availability of tools that check the compliance with the definition of a controlled language and support the writing process (Wojcik and Hoard, 1996). Without such tools that help the author to stick to the defined vocabulary and to the grammar rules, it is nearly impossible to produce informative and consistent documents.

To guarantee the painless and efficient use of our controlled natural language PENG (Processable ENGlish) (Schwitter, 2002), we have designed and implemented ECOLE, a look-ahead text editor that helps authors to write precise and seemingly informal specifications in controlled natural language. ECOLE guides the writing process and guarantees well-formed syntactic structures that can be translated deterministically into first-order logic via discourse representation structures (Kamp

and Reyle, 1993). The semantic interpretation of a specification is built up incrementally during parsing and the systems' interpretation is displayed as a paraphrase in controlled language.

Authors can write specifications in PENG without having to learn and remember the language since ECOLE handles all the approved structures and enforces the restrictions placed upon the language as specified by the controlled grammar and the lexicon.

Strictly speaking the arising specification is a logical theory that can be checked for its consistency and informativity with the help of third-party reasoning services. Apart from checking a specification for these acceptability constraints, an author can also query a PENG specification in order to find possible answers to questions in the text.

The remainder of this paper is organized as follows: In Section 2, we briefly introduce the controlled natural language PENG and discuss the most important lexical and grammatical restrictions of the language. In Section 3, we present ECOLE, the look-ahead text editor from an author's point of view and explain its functionality. In Section 4, we take a look behind the curtain and explain how ECOLE is integrated into the PENG system and how the text editor interacts with the controlled language processor and the reasoning services (theorem prover and model builder). Finally, in Section 5, we summarize the advantages of the presented approach.

2 PENG (Processable ENGlish)

Similiar to Attempto Controlled English (Schwitter, 1998; Fuchs et al., 1999), PENG is a computer-processable controlled natural language specifically designed to write precise specifications (Schwitter, 2002). PENG consists of a strict subset of standard English. The restrictions of the language are defined with the help of a controlled lexicon and a controlled grammar, and are enforced by ECOLE, the look-ahead editor. In the following section we will illustrate the coverage of the language with examples taken from the *Dreadsbury Mansion Mystery*, a logical puzzle that is usually used in the literature to test the capacity of automatic theorem provers (Pelletier, 1986).

2.1 Controlled Lexicon

The lexicon of PENG consists of predefined function words, a set of illegal words (especially intensional words), and user-defined content words. Function words such as

- determiners: *each, a, the, no, who, …*
- prepositions: *in, on, between, …*
- copula: *is, are*
- negation: *does not, is not, …*
- relative pronouns: *who, that, which, …*
- coordinators: *and, or*
- subordinators: *if, before, after, while, …*
- constructors: *there is/are, for every …*

build the structural scaffolding of the controlled language. User-defined content words such as

- nouns: *person, Agatha, butler, …*
- verbs: *lives, kills, is, …*
- adjectives: *rich …*
- adverbs: *slowly…*

can be added or modified by the author during the writing process with the help of a lexical editor that is part of the text editor ECOLE. Thus, by adding content words, the author creates his own application specific lexicon. In addition, the author can define synonyms for content words and acronyms or abbreviations for nouns.

2.2 Controlled Grammar

The controlled grammar defines the structure of simple PENG sentences and states how simple sentences can be joined into complex sentences by a set of coordinators and subordinators. Simple sentences are:

- *Agatha hates a person.*
- *A person lives in Dreadsbury Mansion.*
- *A person A hates a person B.*
- *Agatha is not the butler.*
- *No person hates every person.*

Complex PENG sentences are composed of simpler sentences with the help of coordinators and subordinators:

- *Agatha, the butler, and Charles each live in Dreadsbury Mansion.*
- *If a person lives in Dreadsbury Mansion then that person is Agatha or the butler or Charles.*
- *If a person is not the butler then Agatha hates that person.*

The grammar of PENG also specifies that simple sentences have a linear temporal order by default and that sentences can be anaphorically interrelated in a well-defined way to build coherent textual structures.

2.3 An Example Specification

The *Dreadsbury Mansion Mystery* is usually translated first by hand from unrestricted English into a formal language and the consequence of the puzzle is then proven by a theorem prover. Using PENG, the manual translation into a formal notation becomes unnecessary, since PENG specifications can be unambiguously translated into first-order logic via discourse representation structures. The following paragraph is the *Dreadsbury Mansion Mystery* in PENG. It becomes obvious that it would be hard to write this text in PENG without intelligent writing assistance.

1. *A person who lives in Dreadsbury Mansion kills Agatha.*
2. *Agatha, the butler, and Charles each live in Dreadsbury Mansion.*
3. *If a person lives in Dreadsbury Mansion then that person is Agatha or the butler or Charles.*
4. *If a person A kills a person B then the person A hates the person B.*
5. *If a person A kills a person B then the person A is not richer than the person B.*
6. *If Agatha hates a person then Charles does not hate that person.*
7. *If a person is not the butler then Agatha hates that person.*
8. *If a person is not richer than Agatha then the butler hates that person.*
9. *If Agatha hates a person then the butler hates that person.*
10. *No person hates every person.*
11. *Agatha is not the butler.*
12. Who kills Agatha?

Sentence *1* is a complex sentence that consists of a simple PENG sentence *A person kills Agatha* and an embedded relative clause *who lives in Dreadsbury Mansion*. Prepositional modifiers such as *in Dreadsbury Mansion* always relate to the closest preceding verb phrase in PENG. Sentence *2* contains a plural noun phrase *Agatha, the butler, and Charles* together with the floated quantifier *each* as a specific keyword. This keyword triggers a distributive reading of the plural noun phrase in PENG. Sentence *3* contains an anaphoric reference *that person → a person* and a verb phrase coordination *is Agatha or the butler or Charles* with a copula that has been elided in two conjuncts. Sentences *4* and *5* contain two so-called dynamic names *A* and *B* that make the anaphoric reference explicit on the surface level. Additionally, sentence *5* contains a negated verb phrase *is not richer than ...* that subordinates a comparative construction. PENG distinguishes between verb phrase negation such as in sentences *5*, *6*, *7* and *8* and noun phrase negation such as in sentence *10*. The scope of the negation extends by default to the end of a simple sentence. Sentence *10* consists of two quantifiers *no* and *every*. The relative scope of a quantifier corresponds to its surface position in PENG. Constructors such as *for every* and *there is a* allow quantified noun phrases to move to the sentence initial position to enforce an alternative reading - if necessary. With questions such as in sentence *12*, authors can examine the content of a PENG specification and find the not so obvious answer automatically that Agatha killed herself.

3 ECOLE: User Interface

ECOLE, the look-ahead editor for PENG consists of two parts: a text field where the author develops the specification and a response field where system messages are displayed. Figure 1 gives an overview of the options from which the author can choose.

The most important option of ECOLE is the look-ahead functionality. If this option is selected, then the author is given a list of choices of how to continue the sentence after each word form entered. These syntactic constraints ensure that the document remains unambiguous and precise.

For example, when the author starts typing the sentence *A person lives in Dreadsbury Mansion*,

ECOLE displays the following look-ahead categories as subscripts in angle brackets:

A *[adjective | common noun]*
A person *[verb | negation | relative sentence]*
A person lives *['.' | prepositional phrase | adverb]*

As this example shows, the editor makes use of graphical means to display these syntactic hints as a help to the author. Note that the author needs only minimal linguistic knowledge to choose from these restrictions. Each look-ahead category is implemented as a hyperlink. If something is unclear, then the author can click on one of the displayed categories to obtain more information.

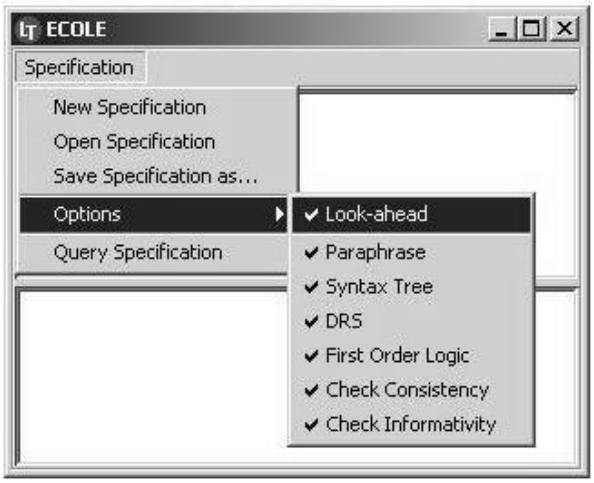

Figure 1: Functionality of ECOLE

The editor also handles compound nouns such as *Dreadsbury Mansion*. When the first noun *Dreadsbury* has been entered, then the editor displays the second part of the compound noun *Mansion* and all other suitable look-ahead categories.

The look-ahead categories are generated while the text is written using the information produced by the chart parser of the controlled language processor.

ECOLE comes with an integrated spelling checker and a lexical editor. If a content word is unknown and not misspelled, then the lexical editor pops up and allows the author to add the word to the lexicon. As soon as the word is available, the processing is resumed. If the author selects the corresponding options beforehand, then the system checks the text for its consistency and informativity after each new sentence. Whenever a new sentence violates these acceptability constraints, then the author gets immediate feedback.

Another option of ECOLE is the paraphrase that informs the author how the system interpreted the input. Below follows an example of an input sentence with a paraphrase generated by the controlled language processor:

Input:

Agatha is not the butler.

Paraphrase:

Agatha is not [identical to] the butler.

The paraphrase thus makes it clear to the author that the copula (*is*) followed by a definite noun phrase (*the butler*) is interpreted as identity. If the copula had been followed by an indefinite noun phrase such as (*a lady*), then the system would interpret this as a property and introduce a state in the semantic representation instead of an identity.

PENG allows only well-defined forms of anaphoric references (definite descriptions and names, but no personal pronouns). The paraphrase informs the author how anaphoric references are resolved during parsing:

Input:

If a rich person lives in Dreadsbury Mansion then that person is Agatha.

Paraphrase:

If a rich person lives in Dreadsbury Mansion then {that rich person} is [identical to] Agatha.

An anaphoric expression is always replaced by the complete antecedent and the string is put within curly brackets. In PENG an anaphoric expression refers to the most recent accessible noun phrase that is suitable in terms of agreement, gender, and type, with respect to the nominal head and the pre- and postmodifiers.

As another option of ECOLE the discourse representation structure (DRS) can be displayed, which may be of interest to anyone wishing to see how the semantics of the processed information is

represented. Normally the author does not have to worry about the underlying semantic representation since he can simply rely on the paraphrase produced by the controlled language processor. However, the underlying representation may be used, for example, to teach students logic and computational semantics.

In general, a DRS captures the information in a multi-sentence discourse and forms a logical theory that shows the relations between the entities, the states, and the events in the application domain. A DRS is represented as a term of the form `drs(U,CON)`, where `U` is a list of discourse referents and `CON` is a list of conditions for these discourse referents. The discourse referents are quantified variables that stand for entities in the specified application domain, while the conditions constitute constraints that these discourse referents must fulfil to make the DRS true. For the following input the controlled language processor generates a DRS and sends it to ECOLE (see also Figure 2 for another example).

Input:

A person who lives in Dreadsbury Mansion kills Agatha.

DRS:

```
[A,B,C,D,E,F]
person(A)
event(B,live(A))
location(C,in(B,D))
named(D,dreadsbury_mansion)
event(E,kill(A,F))
named(F,agatha)
```

The first part of the DRS consists of six discourse referents in square brackets. The discourse referents `A`, `D`, and `F` stand for individuals and have been derived from the noun phrases. The discourse referent `B` and `D` stand for events and have been derived from the verbs. The discourse referent `C` reifies a location and has been derived from the prepositional phrase. These discourse referents are used in the second part of the DRS, which consists of conditions for the discourse referents.

When a noun phrase is found to be anaphoric during parsing (such as the noun phrase *the person* in the example below), then the anaphoric reference is directly resolved and not represented in the DRS.

Input:

A person who lives in Dreadsbury Mansion kills Agatha. The person has a knife.

DRS:

```
[A,B,C,D,E,F,G,H]
person(A)
event(B,live(A))
location(C,in(B,D))
named(D,dreadsbury_mansion)
event(E,kill(A,F))
named(F,agatha)
state(G,have(F,H)
knife(H)
```

Paraphrase:

A person who lives in Dreadsbury Mansion kills Agatha. {The person } has a knife.

As we will see in the next section, such discourse representation structures are first translated into equivalent first-order formulas before they can be processed by off-the-shelf reasoning services.

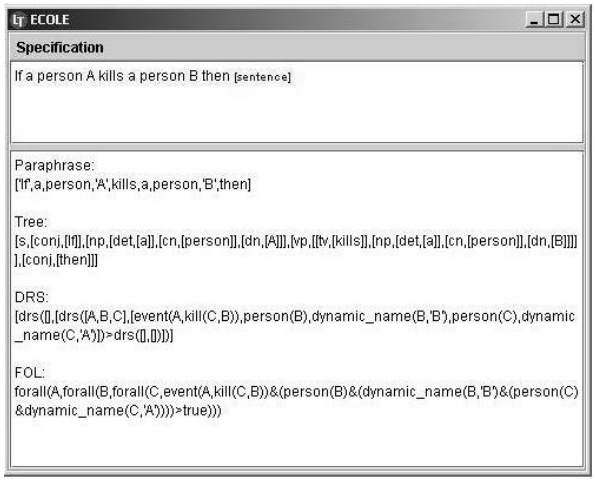

Figure 2: ECOLE in Action

The task of the reasoning services is to check the consistency and informativity of a specification as it is being built up.

4 How ECOLE Works

The top-level architecture of the PENG system consists of five main components: ECOLE, the look-ahead text editor, a controlled language (CL) processor, a server, a theorem prover, and a model builder (Figure 3).

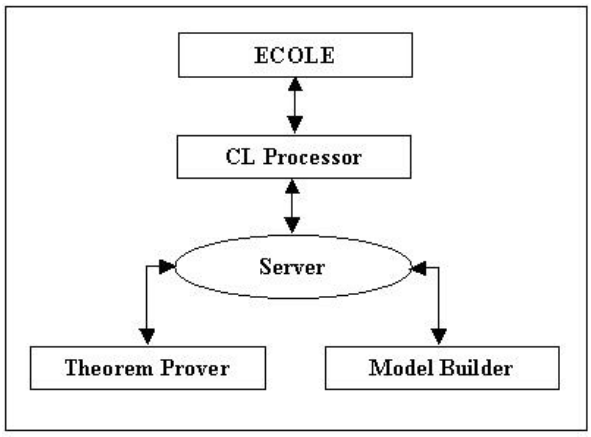

Figure 3: Architecture of PENG

ECOLE communicates with the CL processor via a socket interface. The CL processor is running as a client and is connected via a server with a theorem prover and a model builder. The theorem prover and the model builder are both running separate client processes. The server implements a blackboard on which the CL processor writes a (partial) specification for which the theorem prover searches a proof and the model builder looks for a countermodel. These two reasoning services are used to check whether a specification is consistent and informative, and to answer questions.

4.1 The CL Processor

When the author types a word form into ECOLE, then the current (partial) sentence is sent to the chart parser. The chart parser processes the input in the context of the previous sentences using a unification-based grammar. The grammar is written in a format similar to a definite clause grammar but the chart parser is implemented as a meta-interpreter that reads the grammar as data. This approach allows us to generate the look-ahead categories, resolve anaphoric references and produce the semantic representation and the paraphrase during parsing.

Here comes a simplified example of a grammar rule that is used by the chart parser:

```
n2(Agr,Index,Quant,Drs,Scope,
   ParaIn-ParaOut,[n2,T1,T2],
   Gap-Gap,Ana)
   --->
    det(Agr,Index,Quant,Drs,
        Rest,Scope,ParaIn-
        Para,T1),
    n1( cat:cn,Agr,Index,Quant,
       Rest,Para-ParaOut,T2,
       Gap-Gap,Ana).
```

The chart parser processes such grammar rules top-down and produces edges according to the rules of chart parsing (Gazdar and Mellish, 1989). The edges have the following general form:

edge(START,END,HEAD,BODY)

Such edges simply tell us, what categories of a grammar rule (*HEAD* $\rightarrow$ *BODY*) can span the substring of words found between the *START* point and the *END* point. We can distinguish two types of edges: active and inactive edges. An active edge is a hypothesis about a structure and an inactive edge is a result. For example, if the author types the determiner *the* into the editor, then the chart parser produces the following edges (simplified here):

```
edge(0,1,[det],[])
edge(0,0,[s],[n2,v2])
edge(0,0,[n2],[det,a2,n1])
edge(0,1,[n2],[a2,n1])
edge(1,1,[a2],[a1])
edge(1,1,[a1],[a0])
edge(0,0,[n2],[det,n1])
edge(0,1,[n2],[n1])
edge(1,1,[n1],[n0])
```

The first edge at the beginning of the chart is an inactive edge which contains an empty list `[]`. It represents a confirmed hypothesis and shows that a determiner has been parsed successfully between the nodes `0` and `1`. All other edges are active. That means that the chart is maintaining hypotheses about other structures that might follow.

The look-ahead categories are generated in the following way: during chart initialization the length `L` of the input string is calculated and as soon as active edges are added to the chart that end

at L then the leftmost category on the right hand side of a rule is collected. This process results in one or more look-ahead trees from which the lexical categories can be easily extracted as leafs. In our case chart parsing is incremental. If another word is added to the input string, chart parsing is resumed and additional look-ahead categories are collected for active edges at $L+1$.

The look-ahead categories are then sent back to ECOLE together with a paraphrase, a (partial) syntax tree, the updated discourse representation structure, and a first-order representation of the current input. As we will see in the next section, the first-order representation is not generated directly by the chart parser but derived from the discourse representation structure.

4.2 Reasoning Services

Standard reasoning services are not able to process discourse representation structures directly. Therefore, we translate the discourse representation structure into a set of first-order formulas with the help of an efficient compiler that behaves linearly on the size of the input (Blackburn and Bos, 1999). These first-order formulas build *a logical theory* that can be investigated by a theorem prover and a model builder. We are especially interested to check whether a theory is consistent and informative after new information has been added to that theory. For example, if the author writes:

Agatha is a woman who lives in Dreadsbury Mansion.

and later accidentally adds the information

Agatha does not reside in Dreadsbury.

then the consistency of the theory is violated.

As we will see below, the PENG system can detect such inconsistencies provided that *live* and *reside* are stored as synonyms in the lexicon and that *Dreadsbury* is known as an abbreviation of *Dreadsbury Mansion*.

In a similar way, if the author writes

Agatha lives in Dreadsbury Mansion.

and later adds the information

Agatha resides in Dreadsbury.

then the informativity constraint is violated since the second sentence does not add any new information. Here we would end up with a theory that contains redundant information.

To detect the inconsistency of a theory Φ, we can use a theorem prover and give it the negation of the theory $\neg\Phi$. If a proof is found for the negated theory, then the original theory is inconsistent (or unsatisfiable). To detect the consistency of a theory Φ, we can use a model builder. A model builder is a program that takes a theory and tries to build a model M for that theory. This is done with an interpretation function I that systematically maps predicates and constants of the language to members of a domain D. A theory Φ is consistent (or satisfiable) if the model builder can find at least one model M that satisfies all the formulas in the theory. In general, model builders are only able to construct finite models and require a parameter that constrains the domain size of the model.

Theorem prover and model builder can complement each other (Bos, 2001a; Bos, 2001b; Fuchs and Schwertel, 2002). If a theory is unsatisfiable, then the theorem prover will find a proof while the model builder has to do an expensive search that possibly does not terminate. If the theory is satisfiable for a finite domain, then the model builder will find a model while the theorem prover has to do an expensive search that possibly does not end. Finally, if the problem is satisfiable for an infinite domain, then the theorem prover will never be able to find a proof and the model builder will never succeed to find a model. To deal with this unpleasant case, the theorem prover and the model builder have to stop searching for a solution after a specific runtime.

Apart from helping each other out and checking for inconsistency and satisfiability, the theorem prover and the model builder can also be used to check a theory for its informativity and to construct answers to questions.

Testing whether a piece of information Ψ is new and informative with respect to its previous context Φ can be done by giving the theorem prover $\Phi \rightarrow \Psi$. If it finds a proof, then Ψ is *not* informative. The model builder can do a similar test, provided that we give it $\Phi \wedge \Psi$ and then $\Phi \wedge \neg\Psi$; if the model builder finds a model M in both cases, then Ψ is informative.

A variation of the basic proof procedure can be used to answer questions formulated in PENG.

During a proof variables are bound to values by substitutions. These bindings can be interpreted as a question answering process.

Interestingly, the structures generated by the model builder can also be used for the question answering process. Since the model builder constructs flat structures with no explicit quantification or boolean operators, answers to questions can be easily extracted from these structures (Blackburn and Bos, 1999).

4.3 The Theorem Prover in Action

The PENG system uses OTTER (McCune, 1995), a resolution style theorem prover for first-order logic with equality. The theorem prover client of PENG accepts a theory via ECOLE and the CL processor and transforms the formulas into first-order equivalent OTTER-syntax.

For example for the simple input

Agatha is a human.
Every human is a mortal.

we arrive at the OTTER input file below. The flag name `auto` stands for OTTER's autonomous mode, `max_seconds` for the maximal search time, and `prolog_style_variables` for the format of the variables that start here with `A` through `H`.

```
set(auto).
assign(max_seconds,3).
set(prolog_style_variables).
formula_list(usable).

(exists A (exists B (exists C
  (-(exists G (exists H
     (state(G) & be(G,C,H) &
      eq(C,H) &  mortal(H)))) &
  ((all D (human(D) → (exists E
    (exists F (state(E) &
      be(E,D,F) & eq(D,F) &
     mortal(F)))))) &
  (state(A) & be(A,C,B) &
   eq(C,B) & human(B) &
   eq(C,agatha)))))).

end_of_list.
```

This means that the client calls OTTER in the autonomous mode placing a time limit on the search. In the autonomous mode OTTER decides on inference rules and strategies. OTTER operates on clauses and therefore translates the first-order input immediately into clauses. OTTER uses a number of inference rules (binary resolution, hyperresolution, UR-resolution, and binary paramodulation). These inference rules take a small set of clauses and infer a clause. If the inferred clause is new and useful, then it is stored and used by OTTER for subsequent inferences. When OTTER stops running, it returns with an exit code that gives the reason for termination. For our simple example above, OTTER confirms that it found a proof.

In the case of a *wh*-question such as

Who is a human?

an answer literal of the form `$ans(C)` is automatically added to the query:

```
(-(exists G (exists H
  (state(G) & be(G,C,H) &
   eq(C,H) & mortal(H)))) &
   $ans(C) & …
```

Answer literals make it possible to record instantiations for variables in input clauses during a search for refutation. For example, OTTER produces the following skolemized terms during a proof:

```
eq($c1,$c2).
human($c2).
eq($c1,agatha).
…
$ans($c1)
```

This makes question answering possible and allows us to display the answer as a string in controlled natural language.

4.4 The Model Builder in Action

The PENG system uses MACE (McCune, 2001), a model builder for first-order logic with equality to search for finite models. The model-builder client of PENG accepts a theory from the CL processor and transforms the formulas into OTTER-syntax. OTTER and MACE accept nearly the same input but since MACE attempts to find minimal models and does not distinguish between constants, we need to add extra constraints to the

theory to guarantee that constants in the input are assigned unique elements of the domain. This is not a big problem since these constraints can be generated automatically and be specified in a special list in the input file. The client then calls MACE using that input file with a given finite domain size for the search. MACE transforms the first-order input into an equivalent propositional problem in conjunctive normal form. The propositional problem is then given to a propositional decision procedure. If the decision procedure finds a model that satisfies the set of propositional clauses, then the model is transformed into a first-order model of the original problem. MACE can print models in an easily parsable form that can be read by most Prolog systems. For the two example sentences:

Agatha is a human.
Every human is a mortal.

MACE builds the following kind of model (pretty printed):

```
f(1,human,[d2])
f(1,state,[d1])
f(3,be,[(d2,d2)])
f(1,mortal,[d2])
f(0,c3,d2)
f(0,c1,d2)
f(0,c2,d2)
f(0,agatha,d2)
```

The answers to the questions

Is Agatha a mortal?
Who is a mortal?

can now be extracted from these flat structures and an answer string can be generated in controlled natural language.

5 Conclusions

In this paper we presented ECOLE, a sophisticated look-ahead editor and discussed how this editor is embedded and used in the PENG system. ECOLE guides the author during the writing process. For each word form entered, ECOLE displays look-ahead categories and indicates what syntactic category can follow next. Writing PENG puts no big demands on the author when it comes to learning and remembering the rules of the controlled language as they are efficiently taken care of by the look-ahead editor. The use of the look-ahead categories guarantees well-formed expressions and provides the necessary structural basis for the semantic interpretation of the controlled language in a completely compositional manner. While the author is writing a sentence, a paraphrase is dynamically generated in PENG that explains how the system interprets the current input. PENG texts are deterministically translated into first-order logic via discourse representation structures and can be automatically checked for consistency and informativity with the help of off-the-shelf reasoning services.

Such a computer-processable controlled natural language that is automatically translatable into a formal language has an immense potential and can lead to practical solutions in various application domains:

- Software engineering is one of the first application domains that can benefit from a controlled natural language. Using a controlled language will make it possible to write unambiguous and precise software specifications and to develop taxonomies of domain concepts in a familiar and intuitive notation. Beyond that, it will become possible to check the resulting specification automatically for its consistency and to derive a formal specification automatically.

- The Semantic Web is another application domain that might profit from a controlled natural language. For example, a controlled natural language might be used to model human-readable Web structures and to exploit the underlying reasoning capabilities for the management of information. Instead of struggling with RDF or Notation3, non-specialists could work with a layer of a controlled language that is equivalent to a version of description logic.

These are only two obvious examples of possible application domains. Other domains are business process modeling, database modeling, and legal reasoning.

Besides that we are planning to use the PENG system for teaching students logic and concepts in language technology.

6 Acknowledgments

This research was kindly supported by Macquarie University's New Staff Grant (MUNS 9601/0078). We would like to thank Mitko Razboynkov for developing the first version of the look-ahead editor. We would also like to express our thanks to Sabine Geldof, Marc Tilbrook, and the anonymous reviewer for their useful and constructive comments and suggestions on earlier versions of this paper.

References

AECMA. 2001. The European Association of Aerospace Industries. *AECMA Simplified English, AECMA Document PSC-85-16598*. A Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language. Issue 1, Revision 2, 15 January.

J. Blackburn and J. Bos. 1999. *Representation and Inference for Natural Language*. A First Course in Computational Semantics. Volume II. Working with Discourse Representation Structures. Draft at http://www.comsem.org.

J. Bos. 2001a. DORIS 2001: Underspecification, Resolution and Inference for Discourse Representation Structures. In: Blackburn and Kohlhase (eds): *ICoS-3. Inference in Computational Semantics*. Workshop Proceedings, Siena, Italy, June.

J. Bos. 2001b. Model Building for Natural Language Understanding. Draft at http://www.cogsci.ed.ac.uk/~jbos/.

N. E. Fuchs, U. Schwertel, and R. Schwitter. 1999. Attempto Controlled English - Not Just Another Logic Specification Language. *Lecture Notes in Computer Science 1559, Springer*.

N. E. Fuchs and U. Schwertel. 2002. Reasoning in Attempto Controlled English. *Technical Report*. IFI, University of Zurich, July.

Gerald Gazdar and Chris Mellish. 1989. *Natural Language Processing in PROLOG*. An Introduction to Computational Linguistics. Addison-Wesley, Wokingham, England.

P. Goyvaerts. 1996. Controlled English, Curse or Blessing? – A User's Perspective. *Proceedings of the First International Workshop on Controlled Language Applications*, 26-27 March 1996, Leuven.

W. O. Huijsen. 1998. Controlled Language – An Introduction, Proceedings of the Second International Workshop on Controlled Language Applications. 21-22 May 1998, Pittsburgh, Pennsylvania.

H. Kamp and U. Reyle. 1993. *From Discourse to Logic*. Kluwer, Dordrecht.

T. Mitamura and E. Nyberg. 2001. Automatic Rewriting for Controlled Language Translation. *Proceedings of the NLPRS 2001 Workshop on Automatic Paraphrasing: Theory and Application.*

W. W. McCune. 1995. *OTTER 3.0 Reference Manual and Guide*, Argonne National Laboratory, ANL-94/6, Revision A, August.

W. W. McCune. 2001. *Mace 2.0 Reference Manual and Guide*, Technical Memorandum ANL/MCS-TM-249, Argonne National Laboratory.

F. J. Pelletier. 1986. Seventy-five Problems for Testing Automatic Theorem Provers, *Journal of Automated Reasoning 2*, pp. 191-216.

J. F. Sowa. 2002. Architectures for intelligent systems, *IBM Systems Journal*, Vol. 41, No. 2, pp. 331-349.

R. Schwitter. 1998. Kontrolliertes Englisch für Anforderungsspezifikationen. *Dissertation*, Institut für Informatik, Universität Zürich.

R. Schwitter. 2002. English as a Formal Specification Language. *Proceedings of the Thirteenth International Workshop on Database and Expert Systems Applications (DEXA 2002)*, Aix-en-Provence, France, pp. 228-232.

R. H. Wojcik and J. E. Hoard. 1996. Controlled Languages in Industry. In: R. A. Cole (ed). *Controlled Languages in Industry, Survey of the State of the Art in Human Language Technology*.