Developing a Black Box Specification in Controlled English

Rolf Schwitter
Centre for Language Technology
Macquarie University
Sydney, NSW 2109, Australia
schwitt@ics.mq.edu.au

Abstract

This paper presents a controlled natural language (PENG - Processable ENGlish) and suggests a dialog-driven method for developing a Black Box specification from behavioral requirements written in that controlled language. Such a Black Box specification can be developed in an interactive and systematic way through the process of sequence enumeration. The strength of sequence enumeration is that it requires the domain specialist to consider all possible types of interaction with the future system. The process of sequence enumeration results in a complete and consistent specification that is traceable to its requirements and gives the domain specialists and software engineers a clear picture of what to do in all circumstances of use.

1 Introduction

It can be very cumbersome to describe the total behavior of a complex software system because of the vast number of possible uses of the future system. Neither is it clear what kind of language should be used to describe the behavioral requirements so that a specification can be derived in a systematic way from these requirements (Jackson, 1995; Fuchs et al., 1998).

Natural languages are one possibility: They are most familiar to domain specialists and do not need to be learned. Domain specialists can express their needs in terms that are well established in the application domain. However, requirements written in full natural language are inherently ambiguous, sometimes vague and therefore very difficult to process automatically.

Formal languages are another possibility: They are the preferred choice of software engineers since formal languages are precise, unambiguous and machine-processable. But formal languages are difficult to learn and to understand especially for domain specialists since requirements are then very often written in an opaque manner that abstracts away all considerations of a particular application domain.

It seems that we are trapped between the disadvantages of full natural language and the disadvantages of formal languages. But there is a way out of this quandary: controlled natural languages.

A controlled natural language is a subset of a natural language that has been restricted with respect to its grammar and its lexicon. Grammatical restrictions result in less complex and less ambiguous sentences. Lexical restrictions reduce the size of the vocabulary and the meaning of the lexical entries for a particular application domain. These restrictions make the requirements easier to read and to understand for humans and easier to process for machines (Pulman, 1996; Schwitter, 1998; Fuchs et al., 1998; Grover et al., 2000).

We have designed and implemented PENG, a computer-processable controlled natural language (Schwitter, 2002). PENG covers a well-defined subset of standard English and is precisely defined by a controlled grammar and a controlled lexicon. The controlled language contains domain specific content words and predefined function words. The domain specialist does not need to learn and to remember the restrictions of the controlled language. A look-ahead text editor displays and enforces the restrictions while the requirements are written. Specifications in PENG are deterministically parsed and translated into discourse representation structures (Kamp and Reyle, 1993) with the help of a unification-based parser (Covington et al., 1988; Schwitter, 2002).

Once a set of requirements has been described in PENG, we need a method that helps us to discover the completeness and consistency of our description. One way to check for potential inconsistency in a text is the use of an automated theorem prover. Such a theorem prover can be used in combination with a model builder that generates first-order models and thereby reveals consistency of the text (Bos, 2001; Schwitter, 2002). Although very promising, this approach has not yet been applied to larger texts and scalability with respect to inference is an open issue.

In this paper we are going to investigate an alternative (but less powerful) technique that is known as sequence enumeration (Mills, 1988; Prowell, 1999). Sequence enumeration is a theoretically sound and highly practical approach to describe the external behavior of a system in such a way that the result is traceable to the requirements. This approach considers all permutations of input stimuli to a system and maps them to a response given a prior history. When a sequence of stimuli is noted as equivalent to a previously enumerated sequence by an observer then the current sequence can be reduced to the previous sequence. This enumeration process forms a reduction system that must always end otherwise the future program may never terminate.

Since requirements are automatically translated into discourse representation structures, requirements that are found not to be complete or consistent with the expected behavior of the system can be interactively refined during the sequence enumeration process. The strength of sequence enumeration is that this technique requires the domain specialist to consider obscure behavioral sequences that are usually overlooked and not described in the specification.

The remainder of this paper is organized as follows: In Section 2, we introduce the language PENG and discuss briefly the most important grammatical and lexical constraints of this controlled natural language. In Section 3, we show how domain specialists can write down requirements using a look-ahead editor and how this editor constrains the admissible syntactic structures and checks for approved words. In Section 4, we discuss how PENG sentences are processed and translated into discourse representation structures. In Section 5, we describe how the suggested sequence enumeration module (SEQENU) of PENG works using as an example a soda machine. Finally, in Section 6, we summarize the advantages of the presented approach.

2 PENG - Basic Concepts

PENG is a computer-processable controlled language specifically constructed to write specifications and use cases. PENG is made up of a strict subset of standard English. The restrictions of the language are defined with the help a controlled grammar and a controlled lexicon.

We can give here only a short overview of PENG. For a detailed description of the language please refer to (Schwitter, 2002).

2.1 Controlled Lexicon

The lexicon of PENG consists of predefined function words (*determiners*, *conjunctions*, *prepositions*), a set of illegal words (especially intensional words), and user-defined content words (*nouns*, *verbs*, *adjectives*, *adverbs*). The content words are incrementally added or modified by the domain specialist during the specification process with the help of a lexical editor - a software tool that guides the input of new words. Thus, by adding content words, authors create their own application specific lexicon. In addition, authors can define synonyms, acronyms, and abbreviations for nouns using the lexical editor.

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 coordinators and subordinators. Verbs can only be used in the simple present tense, the active voice, the indicative mood, and the third person singular (or plural) and denote events or states. The grammar also specifies that simple sentences have a strictly linear temporal order and that they can be anaphorically interrelated in a well-defined way to build coherent textual structures.

In the context of a soda machine simple PENG sentences might look as follows:

> *The soda machine accepts dollars.*
>
> *The user enters a dollar.*
>
> *Every user enters a dollar.*
>
> *No user enters a dollar.*
>
> *The user does not enter a dollar.*
>
> *The user enters three dollars one by one.*

Simple sentences can be combined by coordinators (*and*, *or*) and subordinators (*if*, *after*, *before* and *while*):

> *If the operator starts up the soda machine and the user enters three dollars and selects the Soda button then the machine produces a soda.*
>
> *After the operator starts up the soda machine, the machine turns on the Power light.*

In PENG only definite noun phrases and proper nouns can be used anaphorically while personal pronouns are not allowed since their resolution is difficult to control. Here is an example of an admissible anaphoric structure in PENG:

> *If the user enters three dollars one by one then the soda machine produces a soda. The machine accepts dollars only.*

In the above sentence the definite noun phrase *the machine* has the noun phrase *the soda machine* as antecedent. In PENG the result of the anaphora resolution process is indicated by a paraphrase. After processing, the second sentence is displayed as

> {*The soda machine*} *accepts dollars only.*

As the example shows, the system replaces the anaphoric expression by the complete antecedent and mirrors the interpretation.

3 Writing PENG with ECOLE

In contrast to Attempto Controlled English (Schwitter, 1998; Fuchs et al., 1999), the author does not need to know the grammar rules of the controlled language explicitly. PENG uses ECOLE, a look-ahead editor that indicates after each word form entered what kind of syntactic construction the author can use next. In this way the author is guided and the cognitive burden to learn and remember the grammar rules of the controlled language disappears. From a broader theoretical perspective, this look-ahead technique does not only generate and guarantee well-formed expressions but also provides the necessary structural basis for the semantic interpretation of the controlled language in a completely compositional manner.

The look-ahead editor uses the grammatical rules of a definite clause grammar (DCG) and the syntactic categories derived from these rules. Currently we are experimenting with different techniques to derive these look-ahead categories. The simplest but most tedious way is to multiply the DCG rules out and then to add the appropriate look-ahead categories to the grammar rules by hand. A more sophisticated solution is to translate the initial DCG rules into the target format via term expansion while the grammar is consulted or compiled.

When the domain specialist starts typing the sentence *The user enters a dollar*, ECOLE displays the following restrictions as subscripts in angle brackets.

> *The* $_{[\ adjective\ |\ noun\ ]}$
>
> *The user* $_{[\ relative\ clause\ |\ verb\ |\ negation\ ]}$
>
> *The user enters* $_{[\ determiner\ |\ name\ ]}$ ...

This type of functionality is available for programming languages in many modern software development environments. Note that the author needs only minimal linguistic knowledge to resolve these restrictions. The look-ahead categories can be easily implemented as hypertext links in order to provide additional information and examples that explain the meaning of the categories. Experienced PENG users may want to switch off the look-ahead feature and rely on occasional error messages if something goes wrong.

4 Processing PENG Sentences

Specifications written in PENG can be automatically translated into discourse representation structures (DRSs), the representations used in discourse representation theory (Kamp and Reyle, 1993). DRSs make it possible to encode information contained in a multi-sentence discourse and to deal with phenomena such as anaphoric references and presuppositions. Each part of a PENG sentence contributes some logical conditions to the DRS using the preceding textual information as context. We represent a DRS 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. To support our dialog-driven method that we are going to use later in the sequence enumeration process, it is convenient to add a pointer to each condition Con_i in the (unresolved) DRS that refers to the sentence and the surface string from which condition has been derived. For example, the sentence

The user enters three dollars.

can be represented as

```
[A,B,C]

user(A)-[1,[2]]

event(C,enter(A,B))-[1,[3]]

cardinality(B,3)-[1,[4]]

dollar(B)-[1,[5]]
```

whereas *A*, *B*, *C* are discourse referents for the conditions and the integers (*2*, *3*, *4*, *5*) in the list are the back pointers to the surface forms of the first sentence (*1*).

5 An Example Black Box Specification

As an example, we will now consider a simple soda machine and start from an incomplete set of requirements R_i written in PENG that describe the functional behavior of that soda machine. In the following we will focus on the logic of the application rather than on interface issues. However, it is important to keep in mind that the user interacts with the system in controlled language and that she is guided by the look-ahead editor.

R_1: *If the user enters three dollars then the soda machine produces a soda. The machine accepts dollars only.*

R_2: *If the user selects the Change Return button then the machine returns the change that is in the input tray.*

R_3: *If the machine detects that no soda is available then the machine turns on the Sold Out light.*

R_4: *After the operator starts up the soda machine, the machine turns on the Power light.*

The above description provides a partial model of the reactive behavior of the planned soda machine and looks similar to a set of production rules with supplementary constraints (e.g. *The machine accepts dollars only*). Once such a set of requirements is available in PENG, we can perform a sequence enumeration to check whether these requirements are consistent and complete with respect to every possible observed stimulus to the soda machine.

For this purpose let us first represent the soda machine as a black box and list the set of all observable stimuli:

S_1: *The operator starts up the soda machine.*

S_2: *The user enters a dollar.*

S_3: *The user selects the Soda button.*

S_4: *The user selects the Change Return button.*

S_5: *The soda machine detects that no soda is available.*

Once we have collected these stimuli, we have to describe the set of all observable responses (= external actions) that the black box is emitting:

A_1: *The soda machine turns on the Power light.*

A_2: *The machine produces a soda.*

A_3: *The soda machine returns the change.*

A_4: *The machine turns on the Sold Out light.*

The sequence enumeration process starts by evaluating a single stimulus S_i in form of a hypothetical question Q_i to the soda machine and determining the appropriate response for that question. On the first level enumeration (**Level 1**) the domain specialist (= user) answers each question that is generated by SEQENU, the sequence enumeration module of PENG, separately.

With each question Q_i the most suitable requirement R_i is traced and displayed. Since Q_i and the entire set of requirements R_1-R_4 is translated into a discourse representation structure, the most suitable requirement can be found by calculating the overlap of the conditions for the stimulus S_i in Q_i that matches best with the conditions of the available requirements.

If the user detects that the most suitable requirement is not complete or consistent with the expected observations in the environment then the requirement can be modified and the result is ascribed to the stimulus under investigation.

Level 1

System

Q_1: *What happens if the operator starts up the soda machine?*

R_4: *After the operator starts up the soda machine, the soda machine turns on the Power light.*

User

A_1: *The soda machine turns on the Power light.*

System

E_1: *Is there a shorter equivalent sequence?*

User

J_1: *No.*

Here the domain specialist assigns the response (= external action) A_1 to the hypothetical question Q_1 since she can trace the response back to the requirement R_4. After that, the SEQENU module asks the user if there exists a shorter equivalent sequence for the stimulus S_1: *The operator starts up the soda machine* that appears in Q_1. Obviously, the initial stimulus cannot be replaced by a shorter stimulus (sequence) and therefore the user's judgment is *No* in J_1.

In the case of all other stimuli (S_2-S_5) on the first enumeration level, the domain specialist does not assign a response. For example, if the stimulus S_3 occurs in Q_3, then the response A_0 is no (*Nothing*) and the SEQENU module detects that asking for a shorter equivalent sequence is self-contradictory and therefore illegal.

System

Q_3: *What happens if the user selects the Soda button?*

R_0: *none*

User

A_0: *Nothing.*

System

E_3: *A shorter equivalent sequence is illegal.*

The reason for this is that it is not possible for the user of the soda machine to select the Soda button successfully before the operator starts up the soda machine.

The result from performing the first level enumeration is that the operator must start up the soda machine before anything else can be done. The SEQENU module must remember that this event occurred so that the stimulus S_1 can be used on the second level of enumeration. In general, the second level of enumeration uses all stimulus sequences that were not illegal or did not have a shorter equivalent sequence on the previous enumeration level than the first stimulus in the sequence.

This is only the case for the stimulus S_1. This stimulus is combined with each of the possible stimuli S_1-S_5 to the soda machine and a new set of hypothetical questions is automatically generated for the second level of enumeration (**Level 2**).

Level 2

System

Q_6: *What happens if the operator starts up the soda machine and the operator starts up the soda machine?*

R$_4$: *After the operator starts up the soda machine, the machine turns on the Power light.*

User

A$_0$: *Nothing.*

System

E$_6$: *Is there a shorter equivalent sequence?*

User

J$_6$: *The operator starts up the soda machine.*

The first hypothetical question on the second level of enumeration seems to be obscure at first glance. However, if we want to specify the complete behavior of the soda machine, then we have to include such uses into our description. The observer recognizes here that if the soda machine receives a stimulus S_1 followed by another stimulus S_1, then the soda machine reacts as if only a single stimulus S_1 was applied. Since there is no new external response for that sequence, the observer sets the stimulus sequence S_1 S_1 to the shorter stimulus in J_6.

In the next step on the second level the SEQENU module asks what happens if the stimulus S_1 is followed by the stimulus S_2: *The user enters a dollar*.

System

Q$_7$: *What happens if the operator starts up the soda machine and the user enters a dollar?*

R$_1$: *If the user enters three dollars then the soda machine produces a soda. The soda machine accepts dollars only.*

User

A$_0$: *Nothing.*

System

E$_7$: *Is there a shorter equivalent sequence?*

User

J$_7$: *No.*

The two events in Q_7 do not produce an external action of the soda machine (the dollar coin is in the input tray and no action has been triggered) nor does a shorter equivalent sequence exist.

In the next step the user expresses that the sequence S_1 followed by the stimulus S_3: *The user selects the Soda button* produces no response and is equivalent to the sequence S_1 in terms of behavior. Here no valid requirement for Q_8 exists and therefore the user adds a description of the behavior for this situation in D_8. Such derived requirements make explicit behavior which was omitted from the initial requirements.

System

Q$_8$: *What happens if the operator starts up the soda machine and the user selects the Soda button?*

R$_0$: *none*

User

A$_0$: *Nothing.*

D$_8$: *If the user selects the Soda button before the user enters the correct change then the soda machine does not respond.*

System

E$_8$: *Is there a shorter equivalent sequence?*

User

J$_8$: *The operator starts up the soda machine.*

In the next step the sequence S_1 followed by S_4: *The user selects the Change Return button* produces a clearly observable response A_3 and a shorter equivalent sequence is possible.

System

Q$_9$: *What happens if the operator starts up the soda machine and the user selects the Change Return button?*

R$_2$: *If the user selects the Change Return button then the machine returns the change that is in the input tray.*

User

A_3: *The machine returns the change.*

System

E_9: *Is there a shorter equivalent sequence?*

User

J_9: *The operator starts up the soda machine.*

In the last step of the second enumeration level the user provides the response A_4 and adds the derived requirement D_{10} but no shorter equivalent sequence can be found.

System

Q_{10}: *What happens if the operator starts up the soda machine and the machine detects that no soda is available?*

R_3: *If the machine detects that no soda is available then the machine turns on the Sold Out light.*

User

A_4: *The machine turns on the Sold Out light.*

D_{10}: *If the machine detects that no soda is available before the user selects the Soda button, then the machine turns on the Sold Out light.*

System

E_{10}: *Is there a shorter equivalent sequence?*

User

J_{10}: *No.*

Each sequence that does not have a sequence that is illegal or does not have a shorter equivalent sequence on the second level must be remembered by the SEQENU module. For this purpose, the stimuli in the hypothetical questions Q_7 and Q_{10} must be remembered and will be used in the next enumeration level.

This process continues until all observable stimuli are eventually mapped to a shorter equivalent sequence. The enumeration process must always end or the program to be built will not terminate.

In our case the process finally terminates in step 28 on the fifth level (**Level 5**). The stimulus sequence S_1 S_2 S_2 S_2 S_3 will produce the required response A_2: *The machine produces a soda.*

Level 5

System

Q_{28}: *What happens if the operator starts up the soda machine and the user enters a dollar and the user enters a dollar and the user enters a dollar and the user selects the Soda button?*

R_1: *If the user enters three dollars then the soda machine produces a soda. The machine accepts dollars only.*

User

A_2: *The machine produces a soda.*

System

E_{28}: *Is there a shorter equivalent sequence?*

User

J_{28}: *The operator starts up the soda machine.*

Once the enumeration is complete, it is possible to derive a finite state machine that represents the behavior of the soda machine. The number of states in the finite state machine can be found by counting the enumeration sequences that are not illegal or do not have equivalent sequences. The following five sequences occur in the enumeration process and meet this condition ($Level_{Step}$). Starting from these canonical sequences, state data can be defined that encapsulate the behavior described in these sequences.

1_1: *The operator starts up the soda machine.*

{START_UP=TRUE, NO_SODA=FALSE, COIN_COUNT=0}

2_7: *The operator starts up the soda machine. The user enters a dollar.*

{START_UP=TRUE, NO_SODA=FALSE, COIN_COUNT=1}

2_{10}: *The operator starts up the soda machine. The soda machine detects that no soda is available.*

{START_UP=TRUE, NO_SODA=TRUE, COIN_COUNT=0}

3_{12}: *The operator starts up the soda machine. The user enters a dollar. The user enters a dollar.*

{START_UP=TRUE, NO_SODA=FALSE, COIN_COUNT=2}

4_{22}: *The operator starts up the soda machine. The user enters a dollar. The user enters a dollar. The user enters a dollar.*

{START_UP=TRUE, NO_SODA=FALSE, COIN_COUNT=3}

The technique of sequence enumeration can be scaled to larger systems using abstractions for stimuli and responses (Prowell, 1999). We can achieve such abstractions in a natural way on the level of the controlled language by introducing for example plural noun phrases as long as the order of events is preserved (e.g. *The user enters three dollars one by one*).

6 Conclusion

In this paper we presented PENG, a controlled natural language that can be used to write down unambiguous requirements, and we combined this language with a dialog-driven method for developing a black box specification. The sequence-based enumeration technique that drives the dialog offers a systematic method to discover a black box specification from behavioral requirements. The enumeration process considered every possible combination of stimuli to the soda machine and the requirement trace ensured that we covered all possible requirements and - if necessary - allowed us to add omitted requirements on the fly. Because we have considered every combination of stimuli only once and did not map a stimuli combination to more than one possible output, we can guarantee consistency. The big advantage of the presented approach is that the domain specialist can validate the correctness of the mapping from stimuli to responses against requirements written in a familiar notation.

Acknowledgments

This research was partially supported by Macquarie University's New Staff Grant (MUNS 9601/0078). I would like to thank Anna Ljungberg for creating and refining the grammar of PENG and for many interesting discussions, Mitko Razboynkov for developing the first version of the look-ahead editor, and David Hood for integrating the grammar with the look-ahead editor. Finally, I would like to thank three anonymous reviewers for their valuable suggestions.

References

J. Bos. 2001. 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.

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

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.

C. Grover, A. Holt, E. Klein, and M. Moens. 2000. Designing a controlled language for interactive model checking. *Proceedings of the Third International Workshop on Controlled Language Applications*. 29-30 April 2000, Seattle, pp. 29–30.

M. Jackson. 1995. *Software Requirements and Specifications, a lexicon of practice, principles and prejudices*. Addison-Wesley, Wokingham.

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

H. D. Mills. 1988. Stepwise Refinement and Verification in Box-structured Systems. *IEEE Computer*, Number 21, Volume 6, pp. 23–36, June.

S. J. Prowell. 1999. Developing Black Box Specifications Through Sequence Enumeration. *Proceedings of the Harlan Mills Colloquium*, IEEE, May.

S. G. Pulman. 1996. Controlled Language for Knowledge Representation. *Proceedings of the First International Workshop on Controlled Language Applications*, Katholieke Universiteit Leuven, Belgium, pp. 233–242.

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.