Multilingual Generation of Controlled Languages

Richard Power, Donia Scott
IT Research Institute
University of Brighton
Brighton BN2 3GJ, UK
First.Last@itri.bton.ac.uk

Anthony Hartley
Centre for Translation Studies
University of Leeds
Leeds LS2 9JT, UK
a.hartley@leeds.ac.uk

Abstract

We describe techniques based on natural language generation which allow a user to author a document in controlled language for multiple natural languages. The author is expected to be an expert in the application domain but not in the controlled language or in more than one of the supported natural languages. Because the system can produce multiple expressions of the same input in multiple languages, the author can choose among alternative expressions satisfying the constraints of the controlled language. Because the system offers only legitimate choices of wording, correction is unnecessary. Consequently, acceptance of error reports and corrections by trained authors are non-issues.

1 Introduction

A 'Controlled Language' (CL) is a subset of a natural language that is specifically designed for writing clear technical documentation in a particular domain: for instance, AECMA (1996) is a subset of English used for writing maintenance manuals for aircraft. The subset is defined partly through a restricted vocabulary, and partly through rules of composition (e.g., avoid the progressive tense; keep to only one topic per paragraph). Adherence to these rules brings two advantages: first, the text is easier to understand, especially for non-native readers; secondly, the text can be translated more easily into another language, so that Machine Translation will produce more reliable results.

Practical applications of CLs have revealed several problems, in particular the difficulty of adhering to the rules during writing. Various tools have been developed to support authors, for instance by warning them when a rule has been violated (Adriaens, 1996); however, since these tools are unable to suggest permissible alternatives, authors may waste a lot of time searching for a formulation that the system will accept. A related problem is that the rules are often imprecise or incomplete. In AECMA, for example, the rules are essentially of three kinds: specific syntactic constraints (*avoid progressive tense; manufacturing process words must be verbs*), general stylistic precepts (*sentences in procedures should not exceed 20 words; sentences in descriptions should not exceed 25 words*) or semantic precepts (*e.g., keep to one topic per sentence; write only one instruction per sentence*). The syntactic constraints may be fairly precise individually, but overall they do not provide full control since they can be contradictory, and so still require human judgement (Hartley and Paris, 2001). The stylistic and semantic precepts rely necessarily on judgement, and therefore do not lend themselves to automatic checking (Lieske et al., 2002). Even more problematic are the negative reaction of authors to false error reports and the fact that the correction of non-compliant sentences often requires the writer to approach the revision more globally (Wojcik and Holmback, 1996).

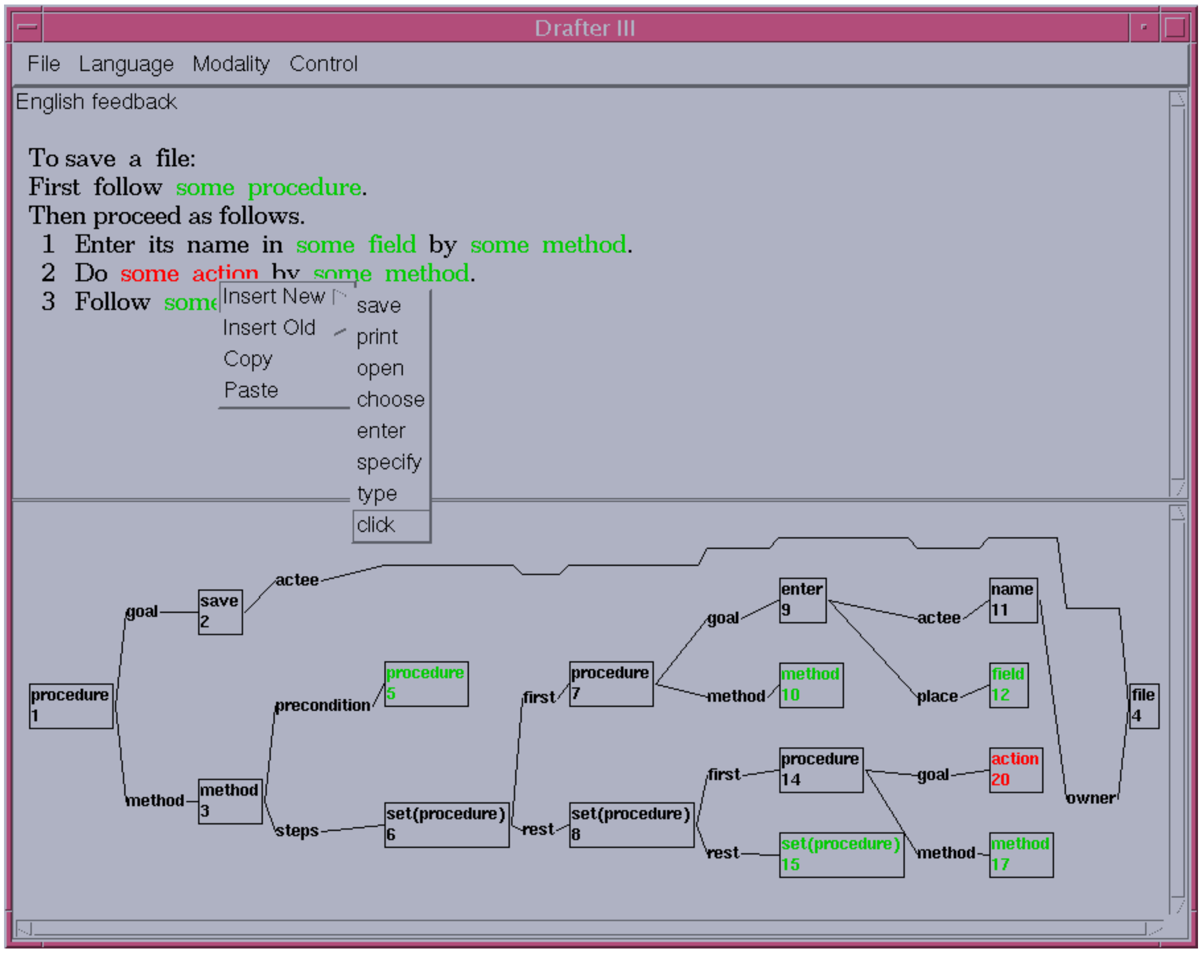
We suggest in this paper an approach to these problems based on the technology of Natural Language Generation (NLG). The radical feature of this approach is that the author is no longer allowed to compose a document by typing in text. All text is generated by the program; the author's contribution is to make a series of choices from a list of options. To implement such an approach, two technical problems must be solved. First, the CL must be defined formally: in place of the sometimes vague precepts and prohibitions illustrated above, we must enumerate precisely not only the vocabulary but also the syntactic patterns that are allowed; moreover, these permissible syntactic constructions must be related in a context-sensitive manner to the section(s) of a document in which they may appear, such as procedures, descriptions, cautions and warnings (Lalaude et al., 1998). Secondly, an interface must be provided through which an author can navigate the options allowed by the CL, thus constructing (and perhaps later modifying) the sentences of the text. This might at first seem impossible, but we will describe later a program that demonstrates how such an interface might work.

If a document can be authored in this way, several benefits accrue. First, as Danlos et al. (2000) have pointed out, text produced by an author cannot but adhere to the restricted vocabulary and the structural constraints of the CL — the program is literally incapable of generating a text that does not! Perhaps less obviously, the editing operations can be defined not on the surface text, but on an underlying structural representation. The author may have no formal knowledge of syntactic patterns like 'restrictive relative clause' or syntactic features like 'perfective', but by selecting from listed options he/she is actually (perhaps unknowingly) building up a syntactic representation which the program then realises as a formatted string of words. This enormously simplifies the task of translating the text to another language: analysis is no longer needed, since transfer rules can be applied directly to the syntactic representations that the author has built up as a by-product of the choices made. A final benefit is that the design of the CL can (indeed must) be precisely specified. A precise formulation containing hundreds of syntactic rules would be far too complicated to serve as a guide for human authors, but an NLG program can take such a grammar in its stride.

Taking a step further, we are now working on an extension of this idea in which editing operations are defined at a *semantic* rather than a *syntactic* level. The crucial observation here is that a CL actually controls *the meanings that may be expressed* as well as the manner in which they are expressed. If no word for 'tractor' is provided in the approved vocabulary, the author cannot write on the subject of tractors. Most NLG systems encode meaning through formulas in some kind of predicate logic, where the predicates are taken from a formally defined 'ontology'. In systems that generate in several languages (e.g., Drafter (Paris et al., 1995; Power and Scott, 1998), AGILE (Hartley et al., 2001), PILLS (Bouayad-Agha et al., 2002), MTA (Brun et al., 2000)), this underlying semantic representation is language-neutral, an interlingua for the natural languages that are supported. As we will show in section 2, from the author's point of view, editing a semantic representation will seem exactly the same as editing a syntactic one — in either case, the options are presented in the interface through generated texts. However, the semantic-based system would bring several further advantages, of which the most obvious is that machine translation would no longer be needed: the production of multilingual documentation would become entirely a problem of generation.

2 Example of authoring

We address straight away what must seem the major obstacle to our approach: how can an author compose a document by select-



To save a file:
First follow some procedure.
Then proceed as follows.
1 Enter its name in some field by some method.
2 Do some action by some method.
3 Follow some

Figure 1: Example of WYSIWYM editing

ing options rather than by typing characters? Surely the number of options will be impracticably large? Two solutions have been tried. In the NLMenu system (Tennant et al., 1983), a sentence is built word by word, from left to right; at every stage, the system computes all possible continuations that are allowed by the grammar and lexicon. In WYSIWYM editing (Power and Scott, 1998), a semantic knowledge base is created by expanding a feedback text which serves primarily to describe the current state of the knowledge base and available ways of extending it. Figure 1 shows a snapshot of a user using WYSIWYM to author a set of software instructions. The top pane shows the feedback text, with the user selecting expansion options; the bottom pane shows the knowledge base that has been constructed so far from the user's choices.

This method is far easier to implement efficiently; we also think it is more intuitive, although this has not yet been demonstrated through an evaluation study. The top-down method relies on the use of general labels called 'anchors' to mark the points at which a pattern is incomplete. For instance, in building the sentence 'the patient takes the medicine', the author would first select the pattern '[someone] takes [something]', only subsequently choosing specific options ('the patient', 'the medicine') to replace the anchors '[someone]' and '[something]'. The number of options at every decision point is kept within manageable bounds thanks to selection restrictions which limit the options displayed to those concepts that can legitimately instantiate the anchors.

As a running example, we will show how

a sentence is constructed in a variant of the PILLS system (Bouayad-Agha et al., 2002), which generates instructions in the pharmaceutical domain, including patient information leaflets (PILs) in English, French and German. We will assume that the author is working in English on the first sentence, which will warn the patient that before starting to take the medicine, he/she should read the leaflet carefully. Half way through editing the sentence in our example, the feedback text might read as follows:

> Before you start to take **your medicine**, [something is the case].

A feedback text has special features that allow editing by direct manipulation. By exploring with the cursor, the author will find that some spans are mouse-sensitive, and can be selected with a click. Once selected, a span is displayed in a distinctive colour (blue rather than black); in the paper, instead, we use bold face – thus in the example, the span currently selected is 'your medicine'. Spans that can be selected in this way are called 'editing units'. Editing units are typically nested, so that the author may select the whole as well as the parts. In the example, the whole sentence may be selected, by clicking around the word 'Before'; alternatively, the clause 'you start to take your medicine' may be selected by clicking around 'start to take'; the other editing units are 'you', 'your medicine', 'your', and '[something is the case]'. Generally, editing units correspond to syntactic clauses or phrases, and are selected by clicking on the head word (e.g., the verb for a sentence, or the noun for a noun phrase). For larger units like paragraphs or sections it is convenient to provide some scaffolding – for instance, a label or glyph at the start of a paragraph through which the whole paragraph can be selected.

Having selected an editing unit, the author can do various things with it. If the unit is an anchor such as '[something is the case]', it can be replaced by a specific pattern based around a head word (e.g., a verb). If the unit is already complete, or partially complete, it can be cut (in which case it returns to an anchor), or it can be modified in various ways by choosing from a list of potential replacements. These operations can be compared with their counterparts in drawing editors like MacDraw and Xfig. Choosing a pattern at an anchor is similar to choosing a basic shape like a line, or a circle, or a rectangle. Having chosen a pattern, you can tweak it in order to change the default settings for colour, size, and so forth; similarly, a basic clause pattern with defaults like TENSE=`present`, POLARITY=`positive`, can be gradually reshaped so that it becomes a negative statement in the past tense, with perhaps an adverbial modifier added for good measure.

To show how these operations are presented, let us trace through the process of completing our sample sentence. The author should first click on the anchor '[something is the case]', in order to select it as the current editing unit. This has two results: first, as already mentioned, the selected unit is highlighted (bold face); second, a list of replacement options appears in a second pane:

> Before you start to take your medicine, [**something is the case**].

```
[someone] asks [someone] [something]
[something] attacks [something]
...
[someone] provides [something]
[someone] reaches [something]
[someone] reads [something]
[someone] remembers [that something is the case]
[someone] removes [something] from [something]
[someone] starts [to do something]
...
```

Each replacement is a clause pattern based on one of the verbs in the restricted vocabulary (note by the way that we can restrict patterns as well as verbs, thus perhaps allowing 'starts [to do something]' but not 'starts [doing something]'). Of course there may be hundreds or even thousands of verb patterns; to prevent tedious scrolling, the author may type a few letters of the verb into a text field, so limiting the displayed options to ones containing this substring. At this point, the program could also respond helpfully if the author types in a verb that is *not* approved in the restricted vocabulary: thus on typing in 'commence' or 'initiate', assuming these are prohibited words, he/she might obtain the licensed pattern based on 'start'.

Having possibly narrowed down the options in this way, the author next selects the desired replacement pattern by a mouse click. In response, the program adds this pattern to the underlying syntactic structure, then regenerates the feedback text and the replacement options:

> Before you start to take your medicine, **[someone] reads [something]**.

```
SENTENCE TYPE
 [something] is read by [someone]
 does [someone] read [something]?
 read [something]

NEGATIVE
 [someone] does not read [something]

TENSE
 [someone] has read [something]
 [someone] read [something]
 [someone] will read [something]

MODAL
 [someone] can read [something]
 [someone] should read [something]

MODIFIER
 [someone] reads [something] [somehow]
```

The replacement options now correspond to the various ways in which a geometrical shape (e.g., a line or a rectangle) can be tweaked in a drawing editor. Each proposed replacement is the result of varying the current pattern *on one dimension at a time*. Just as you cannot change the size and colour of a rectangle through a single operation, so you cannot choose an option that yields both imperative sentence-type and an adverbial modifier – in this way, the number of options shown at any point remains manageable. Also, here as in a drawing editor, these modifications can be made in any order (i.e., imperative before modifier, or modifier before imperative); in either case, the resulting clause in the feedback text will be 'read [something] [somehow]'. It now remains to add the noun phrase and the adverb. Again, this can be done in any order, but we will assume that first the author selects the anchor '[something]':

> Before you start to take your medicine, read **[something]** [somehow].

```
an answer
a book
a compendium
a data-sheet  a document
...
a leaflet
a prescription
...
```

After selecting 'a leaflet', the author again gets a list of replacements through which the basic pattern can be tweaked:

> Before you start to take your medicine, read **a leaflet** [somehow].

```
DETERMINER
the leaflet
this leaflet
...

MODIFIER
a [some-kind-of] leaflet
a leaflet [of some kind]

RELATIVE
a leaflet [such that something is the case]

SEQUENCE
a leaflet and [something]
a leaflet or [something]
```

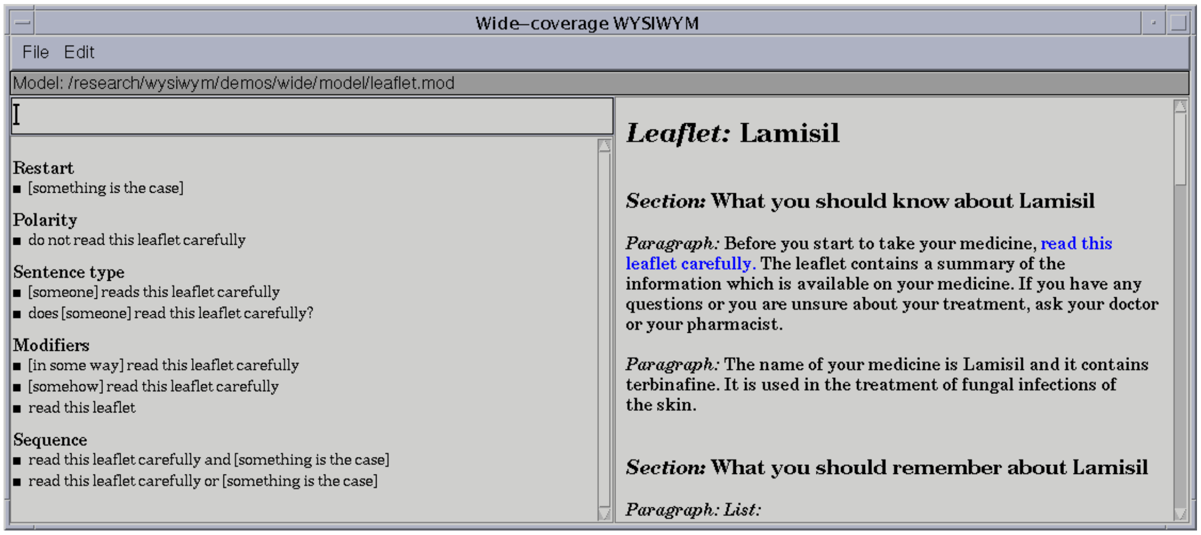
After choosing the replacement 'this leaflet', the sentence can be completed by selecting the anchor '[somehow]' and choosing the adverb 'carefully' (which requires no tweaking):

> Before you start to take your medicine, read this leaflet **carefully**.

Figure 2 shows a snapshot from the an application that uses this editing method; it includes the sentence that we have just constructed.

3 Computing replacements

From the author's point of view, editing consists of replacing a span in the current feedback text by a phrase from the list of replacement options. However, this is strictly speaking an illusion: what actually happens is that an underlying structure is changed, and the surface text then regenerated. This is true also for drawing editors. A rectangle is stored not as a pixel pattern but as a model, with parameters for size, colour, and so forth; this model might remain the same even though the surface presentation changes – for instance,


Wide–coverage WYSIWYM

File Edit

Model: /research/wysiwym/demos/wide/model/leaflet.mod

Restart
■ [something is the case]

Polarity
■ do not read this leaflet carefully

Sentence type
■ [someone] reads this leaflet carefully
■ does [someone] read this leaflet carefully?

Modifiers
■ [in some way] read this leaflet carefully
■ [somehow] read this leaflet carefully
■ read this leaflet

Sequence
■ read this leaflet carefully and [something is the case]
■ read this leaflet carefully or [something is the case]

Leaflet: Lamisil

Section: What you should know about Lamisil

Paragraph: Before you start to take your medicine, read this leaflet carefully. The leaflet contains a summary of the information which is available on your medicine. If you have any questions or you are unsure about your treatment, ask your doctor or your pharmacist.

Paragraph: The name of your medicine is Lamisil and it contains terbinafine. It is used in the treatment of fungal infections of the skin.

Section: What you should remember about Lamisil

Paragraph: List:

Figure 2: Editing a patient information leaflet

because the rectangle is covered by another shape. Both in a drawing editor and WYSIWYM editing, the ways in which a pattern may be modified depend on parameters in the underlying model.

For the program described in the previous section, the underlying model was a deep-syntactic structure with parameters similar to those in the RealPro system (Lavoie and Rambow, 1997). With some simplifications, here is the model for the completed clause 'read the leaflet', presented in a feature-structure notation, with alternative values for some features shown in brackets:

```
sentence
VERB read
SENTYPE imperative (declarative, ...)
VOICE active (passive)
MODAL none (can, might, would, ...)
TENSE none (present, past)
PERFECTIVE none (perfective)
POLARITY positive (negative)
ARG0 none (anchor [someone])
ARG1 nominal
     NOUN leaflet
     DETERMINER the
     NUMBER singular
     PREMOD none
     POSTMOD none
     RELATIVE none
ARG2 none
PREMOD none (anchor [somehow])
MIDMOD none (anchor [somehow])
POSTMOD none (anchor [somehow])
```

If the unit currently selected is the whole sentence, replacements are computed by varying the features SENTYPE, VOICE, MODAL, etc., one at a time. Thus for SENTYPE, the alternative values might be `interrogative` and `declarative`, yielding this group of options:

does [someone] read the leaflet?
[someone] reads the leaflet

Note that sometimes a change in one feature requires a change in other features: here, present tense and an anchor for ARG0 have been introduced (these features are not used in an imperative).

4 Editing a semantic representation

Building up sentences in this way is somewhat tedious: a sentence that one would normally write in perhaps 30 seconds requires perhaps two or three minutes. The comparison with systems such as TRANSTYPE (Langlais et al., 2002), which offer the author a ranked list of possible completions as she/he types, is even less flattering.

In compensation, the author can be confident that the sentence conforms to a very strictly defined controlled language – there will be no need to fish around for alternative formulations because the first effort has been rejected. Whether this outweighs the extra construction time is unclear. Again, the same result could be achieved with the TRANSTYPE approach, provided the possible completions were drawn from an authoring memory that was itself controlled language compliant.

However, the balance of advantages changes completely if the editing tool allows the re-use of the underlying semantic model for the automatic generation of the same content in styles appropriate to different sections of the documentation and/or for the automatic generation of versions in other languages (Hartley et al., 2001). Two minutes spent on a ten-word sentence might be regarded as a worthwhile investment of effort if it yields versions (say) in all the languages of the European community.

To obtain this extra benefit, the program described above must be extended so that instead of editing a syntactic model (specific to English or some other natural language), the author edits a *semantic* model. Everything else remains the same, including what is shown on the screen. In fact, from the author's viewpoint, the system will seem almost the same, the only differences being slight changes in the organisation of the options, and the new facility of getting versions in other languages.

In developing a semantic representation, we have emulated Bateman (1993), among others, in seeking a classification scheme that maps as closely as possible to linguistic structure. For logical purposes (i.e., for use in a reasoning system), a representation should address issues like quantifier scope and deixis; at a surface-semantic level, we can ignore these problems, leaving variables unscoped, and allowing modifiers (like *therefore*) that make an implicit reference to some previous proposition. However, a surface-semantic representation must progress beyond features like SENTENCE-TYPE, TENSE, and MODAL. For example, an Italian rendering of 'read the leaflet' will probably use the infinitive instead of the imperative to express the command; moreover, Italian has no modal verbs like 'can' and 'would', but has instead many more tenses (e.g., imperfect, conditional, subjunctive, as well as present and past).

Designing the semantic representation is work in progress, but we will give as an example a possible representation for 'read the leaflet' (or its Italian counterpart 'leggere l'opuscolo'):

```
event
TYPE read
FORCE command
CONFIDENCE medium
TIME present
ACTOR unspecified
ACTEE object
      TYPE leaflet
      REFERENCE definite
      NUMBER singular
      MODIFIERS none
MODIFIERS none
```

An important feature here is FORCE, with possible values like `assert`, `deny`, `query`, `command`, and `forbid`. The FORCE value will affect several syntactic variables, including sentence type, modal, and polarity. The syntactic realisation of FORCE often depends on the CONFIDENCE level: thus for low confidence we might obtain 'you might consider reading the leaflet', or for high confidence 'you must absolutely read the leaflet'.

5 Conclusion

We have illustrated a method through which a document can be authored using a direct-manipulation interface. The drawback is that composing the text will take longer, but a good implementation would yield several compensating advantages:

- The documents produced with the authoring tool would always conform to a precisely specified CL.
- Using semantic (as opposed to syntactic) authoring, versions in multiple natural languages could be obtained using language generation only – no interpretation or transfer would be needed.
- Since the program has a structural description of the document, it would be easier to add facilities for document reprocessing (e.g., generating a summary, or changing the linguistic style in a context-sensitive manner).

A potential problem is that these benefits may not be appreciated by the technical author (or whoever encodes the content) — they accrue only further down the line.

More generally, we have argued that the lexical and structural restrictions in a CL can be seen as controlling *content* as well as linguistic realisation. This means that a family of CLs, perhaps in several natural languages, might share the same ontology (i.e., the same rules for constructing semantic expressions). One could even imagine several CLs in the *same* natural language expressing the same information in a manner suited to different readers – for instance, medical information could be presented differently to doctors, patients, and pharmacists.

The technology required in order to develop this kind of authoring tool is well advanced. In several traditions of research in NLG, ontologies and grammars with fairly wide coverage have been developed. A major difficulty has been the sheer complexity of the relationship between meaning and linguistic form: even a simple 10-word sentence can be paraphrased in thousands of ways. However, in applications in which the target language is a CL, it is much easier to find a principled and efficient path through this maze of possibilities — by reducing potential realisations to a small manageable set, we mirror exactly the constraints that a CL seeks to impose.

Finally, it is worth noting that the authoring tool would be far more efficient if coupled with a system that could extract information (albeit unreliably) from free text. Suppose that an interpretation system takes as input an existing text (perhaps a 'legacy document' of the user), and does its best to render the meaning in the semantic formalism used by the authoring tool. The resulting model, which may contain interpretation errors, can now be viewed through a feedback text generated by the authoring tool, so that any mistakes can be corrected through normal editing. Some interpretation capability might also speed up the process of sentence composition: for instance, instead of laboriously building up the sentence 'read this leaflet' through a series of choices, the author might be allowed simply to type it into a text field, whereupon the program will present its interpretations (possibly mutliple) as replacement options. If there are no interpretations (the input was incomprehensible), the author will have to take the long road; if there are multiple interpretations (the input was ambiguous), the author can choose the correct one. Of course, the program will express its interpretations in the approved wording of the CL, not in the wording entered by the user, so different interpretations will be presented through distinct replacement options.

References

Geert Adriaens. 1996. SECC: Using text structure information to improve checker quality and coverage. In *Proceedings of the First International Workshop on Controlled Language Applications*, pages 226–132. CCL, Leuven.

AECMA. 1996. AECMA Simplified English: A guide for the preparation of aircraft maintenance documentation in the International Aerospace Maintenance Language. AECMA, Brussels.

John A. Bateman. 1993. Ontology construction and natural language. In *Proceedings of the International Workshop on Formal Ontology*, pages 83–93, Padova, Italy. LABSEB-CNR.

Nadjet Bouayad-Agha, Richard Power, Donia Scott, and Anja Belz. 2002. PILLS: Multilingual generation of medical information documents with overlapping content. In *Proceedings of the Third International Conference on Language Resoures and Evaluation (LREC 2002)*, pages 2111–2114, Las Palmas.

C. Brun, M. Dymetman, and V. Lux. 2000. Document structure and multilingual authoring. In *Proceedings of First International Natural Language Generation Conference (INLG 2000)*, pages 24–31. Mitzpe Ramon, Israel.

Laurence Danlos, Guy Lapalme, and Veronika Lux. 2000. Generating a controlled language. In *Proceedings of the First International Conference on Natural Language Generation (INLG 2000)*, pages 141–147. Mitzpe Ramon, Israel.

Anthony Hartley and Cecile Paris. 2001. Translation, controlled languages, generation. In E. Steiner and C. Yallop, editors, *Exploring*

Translation and Multilingual Text production, pages 307–325. Mouton de Gruyter.

Anthony Hartley, Donia Scott, John Bateman, and Danail Dochev. 2001. AGILE: A system for multilingual generation of technical instructions. In *Proceedings of 8th Machine Translation Summit (MT Summit VIII)*, pages 145–150. Santiago de Compostela, Spain.

Myriam Lalaude, Veronika Lux, and Sylvie Regnier-Prost. 1998. Modular controlled language design. In *Proceedings of the Second International Workshop on Controlled Language Applications*. LTI, Carnegie Mellon University.

Philippe Langlais, Marie Loranger, and Guy Lapalme. 2002. Translators at work with TRANSTYPE: resource and evaluation. In *Proceedings of the Third International Conference on Language Resoures and Evaluation (LREC 2002)*, pages 2128–2134. Las Palmas de Gran Canaria, Spain.

B. Lavoie and O. Rambow. 1997. RealPro: A fast, portable sentence realizer. In *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC.

Christian Lieske, Christine Thielen, Melanie Wells, and Andrew Bredenkamp. 2002. Controlled authoring at SAP. In *Proceedings of Translating and the Computer*. ASLIB 2002, London.

Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott. 1995. A support tool for writing multilingual instructions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1398–1404, Montreal, Canada.

R. Power and D. Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, pages 1053–1059, Montreal, Canada.

H. R. Tennant, K. M. Ross, and C. W. Thompson. 1983. Usable natural language interface through menu-based natural language understanding. In *CHI'83 Proceedings*. Computer Human Interactions.

Richard Wojcik and Heather Holmback. 1996. Getting a controlled language off the ground at Boeing. In *Proceedings of the First International Workshop on Controlled Language Applications*, pages 22–31. CCL, Leuven.