Controlled Language - Issues in Checkers' Design

Uus Knops
LANT nv/sa Belgium

Abstract

The present paper deals with several recurrent issues in the design and implementation of controlled language checkers. It is based on market analysis and on LANT's experience in building and customizing controlled language checkers.

1 Introduction

A controlled language (CL) refers to a standard in which the grammar, the style and the vocabulary are more limited than in the normal use of the language. A CL is generally adopted in order to increase the readability and translatability for non-native speakers. It is not entirely clear whether a CL increases the readability by native speakers of the language. The increase in translatability usually refers to machine-aided processes, although translatability by humans is normally implied as well.

The value of a CL for readability and translatability is now generally recognized for the restricted application in the domain of technical documentation. Also, the value of a checker to support the introduction of CL has become widely accepted. More particularly, CL checkers are believed to be supportive in the following respects:

- They reduce training time and costs for authors while allowing for computer-based training
- They support authors in adjusting their texts to the CL standard
- They support editors in validating documentation against the CL standard
- They improve translation quality in case of auto-correction.

Despite its generally accepted value, a CL and its support environment, is only one of the factors in the entire documentation development chain: document management, terminology management, CL and CL checking, translation memory, machine translation and other mechanisms must all combine for an optimization of the complete documentation process.

2 LANT®MASTER™

LANT®MASTER™, the CL checker developed and customized by LANT, is a client-server application in which the checking server is conceived as a special language pair (English to Controlled English) within the LANT machine translation (MT) environment. The checker's input is an English sentence, which is first analyzed by the same English grammar as is used for translation into other languages. Conformance checking itself is effected during the transfer and generation phase.

The checker's output is a dual object: the input sentence annotated with diagnosis labels for all detected rule violations, and an auto-correction string with proposed revisions for most diagnosed violations. The restricted vocabularies are implemented as an English-Controlled English transfer lexicon. Conformant terms have a target language form identical to the source language form. Non-conformant terms are mapped into conformant alternatives.

The client enables authors to import and export a file, to submit it to the checker, to edit it, to generate an error report, and to re-check individual sentences (interactive checking). It also provides users with the functionality to look up grammar rules and vocabulary, to overrule errors and to file reports to the CL administrator.

By now, LANT has implemented a whole library of controlled English rules, with each rule being tagged with the appropriate customer label. Up till now, all rules and vocabularies have been specified and implemented for English. A fair amount of the CL rules could be easily extended to other languages, such as German, French, Spanish and Dutch, but LANT has not yet implemented controlled grammars and lexicons for languages other than English.

LANT's library of controlled English rules is extensive enough to serve different application needs. In some applications the conformance checker is primarily conceived as an authoring tool for native and non-native writers in a monolingual English documentation development environment (e.g., AECMA SE). In other applications the emphasis lies more on translation. Here, the checker is primarily used as a pre-editing tool for translation either by humans or by machine (e.g., CASL, Means and Godden 1996).

3 Deep vs. Shallow Analysis

One of the main issues in CL checkers' design is the question whether checking should be based on a deep or shallow analysis of the language involved. This question cannot be easily answered in a general way, because the required depth of analysis varies greatly according to the rules.

Consider the CL rule pertaining to the restrictive length of sentences. Here a word-counting mechanism will do, and a full parse seems to be overkill. It should be noted, though, that even with regard to this simple rule the diagnostic results of a full parse may differ from those of a simple word-counting mechanism, in a sentence such as:

E.g. The PCM selects the most appropriate ignition timing settings from within the PCM's programming.

A full-parse approach performs a complete syntactic analysis of the sentence, before it identifies and localizes CL rule deviations. The approach is usually adopted by industries that already have a syntactic parser for a language and start to look for extensions in new application fields. The full-parse approach requires that the input to the checker is a complete and grammatical sentence. As a consequence, a full-parse checker may allow for less interactivity and robustness compared to a checker with a flat formalism. The main arguments for adopting a full-parse approach are the following (see also Schmidt-Wigger, 1998):

- Reuse of linguistic resources
- Accuracy in error diagnosis, both in terms of recall (false negatives) and precision (false positives)
- Extension towards corrector functionality
- Consistency between checker and MT results

The reuse of an already existing parser for a language allows for keeping checker development costs low, while ensuring high-quality results in diagnosis. If a checker is to be developed from scratch, the heuristic approach based on string-matching mechanisms will entail lower development costs, inevitably coupled with lower accuracy levels. Consider the CL rule, stating that *"in order to"* should be used instead of *"to"* to introduce a purpose clause. In order to disambiguate correctly between the subclause conjunction *"to",* the infinitive marker *"to",* the preposition *"to"* and the verbal adjunct *"to"*, a complete syntactic analysis of the sentence is needed. E.g.,

+ The program offers the possibility *to revise the text.*
+ The program allows the user *to revise the text.*
- The user selects the 'batch' or 'interactive' mode *to revise the text.*
+ The user selects the 'batch' or 'interactive' mode *in order to revise the text.*

In addition, only a full parse allows for the controlled generation of a correction string. Although this corrector's functionality is a controversial issue in CL (see section 5), it is clear that auto-correction, if offered, should aim at eliminating all errors and that it must not introduce new errors.

Consider the CL rule pertaining to the restrictive use of passives. It is possible for a tool based on shallow linguistics to recognize passives, but it is more difficult for such a tool to rephrase a passive sentence into an active and to propose this as a revision to the human editor. In a full-parse approach rephrasing the sentence is not a problem, provided the logical subject is expressed. E.g.,

- The program *can be started* by the administrator from the main screen.
+ The administrator *can start* the program from the main screen.

A further advantage in re-using an existing parser lies in the fact that a complete consistency between checker results and MT results can be obtained. Such a consistency is particularly important in applications where the conformance checker is conceived as a facilitating step to MT.

4 Parser-driven vs. Rule-driven Checkers

Within the full-parse approach a further distinction can be made between parser-driven and rule-driven checkers. Parser-driven checkers intervene in the authoring process, if and only if the parser is hindered in forming a complete representation of the underlying sentence. In fact, no full automatic analysis is performed. Instead, the author is interactively restricted to the structures allowed and covered by the grammar, or he is asked to solve structural ambiguities. Parser-driven checkers will by their very nature never produce auto-corrections. They simply ask questions and expect unambiguous answers from the users. They act more as text generation tools than as checkers.

Rule-driven checkers start from a set of rules that need to be adhered to by the authors. This is usually a set that can be learned, that is structured and makes some sense to the authors. The rule set can serve readability issues as well as translatability issues. The checker is more or less independent of the translation tools. It can be machine translation, but the underlying translation engine need not necessarily be the same as the checking engine. In addition, translation memories and human translation will profit from rule-driven checkers as well. The machine translation process will not necessarily yield a complete and perfect translation, however.

The disadvantages of parser-driven over rule-driven checkers are the following:

- Lack of user friendliness
- Lack of corrector functionality
- Lack of support functionality for text comprehension by humans
- Complete dependency between checking and translation

In view of the above disadvantages, pure parser-driven checkers are rare. Most full-parse checkers combine both approaches. Siemens' Checker for Siemens-Dokumentationsdeutsch (SDD) for instance, addresses besides the actual violations of some 10 SDD rules the more prominent problem of textual ambiguity by helping the author to solve it interactively (Schachtl 1996). LANT's checker, though basically rule-driven, warns the user in case of a parse-error or a phrasal analysis. These warnings serve a dual goal. First, they draw the user's attention to the fact that the diagnosis may be less reliable than in the case of a successful parse. Second, they signal the fact that a subsequent translation might not be complete.

The effectiveness of a rule-driven checker is for a great deal dependent on whether its rule set has been defined with computational tractability in mind. The difference between machine-oriented and human-oriented CL's (Huijsen 1998) is of relevance here. A machine-oriented CL typically defines implicitly the rules about structures that are allowed, and explicitly the rules about structures that are forbidden. Moreover, the explicit rules are more precise and specific than in a human-oriented CL. In a human-oriented CL the rules tend to be positive, vaguer and less formal. It is also much harder to determine the effects of their use. E.g.,

> "Present new and complex information slowly" (AECMA SE)
> "Make your instructions as specific as possible" (AECMA SE)
> "Take care with the logic of *and* and or" (Pym 1990)

5 Correction vs. Auto-correction

The question whether CL checker programs should attempt to automate correction is another controversial issue. In addition to flagging errors, most checkers make suggestions for error correction, but only few actually perform some amount of auto-correction. The opponents' arguments against auto-correction are that the control and correction procedure as well as the responsibility for the text should remain with the author (Schmidt-Wigger 1998, Wojcik 1998).

Still, there is no question that any automation in error correction is likely to be welcomed by authors, provided that there are few false alarms, that the auto-correction does not introduce new rule violations, and that authors can retain the feeling that they are in charge of their documents. The latter can be facilitated by presenting the auto-correction as a proposed revision, and by ensuring that authors always have the possibility to overrule errors that are diagnosed by the system.

Analogously to the situation for analysis, auto-correction follows a heuristic approach or an MT approach. The heuristic approach uses pattern substitution methods to correct rule violations. The MT approach applies full computational transfer and generation grammars in order to "translate" the unrestricted language into CL. The latter approach is adopted by LANT, even if its checker cannot correct every error. The auto-correction, offered as a proposed revision to the author, can still contain errors that need manual revision. Some very obvious examples of such errors are:

- lexical errors due to a term not being known to the system
- lexical errors for which selection of the most appropriate alternative requires human judgement
- sentence-length errors
- passive errors in contexts where the logical subject is not expressed
- errors due to ellipsis, such as the omission of the direct object(s) in a sentence
- parenthesis errors where parentheses are used for explanatory text

In order to increase a checker's performance with regard to its proposed auto-corrections, the integration of a checking memory should be considered as a further promising alternative. A checking memory is comparable to a translation memory in that it stores rewrites that have already been validated by humans. In addition, one might expect that the hit rate of the system is increased by the lexical and syntactic standardization, involved in CL. Hence, time and money can be saved by reducing repetitive human revision tasks to an absolute minimum.

Again, the function of a checking memory should be defined as an aid to the author, presenting matches to the author as proposed revisions, which are expected to be more useful than the proposed auto-correction produced by the checking engine. It is obvious that a checking memory is a particularly useful approach for systems that have no corrector's functionality at all. Storing text-type information with the original sentences (e.g. that a given phrase is a title) will render the results more accurate with regard to the matching algorithm. Some CL rules are text-type specific, and should a sentence occur in a certain text-type, the human revision, proposed by the checking memory, may not be appropriate for a different text-type.

6 Grammar and Style vs. Terminology

Customers normally focus on rule issues. They come to LANT with a set of rules, with the wish to have these rules implemented, or with the wish to have the set of AECMA-SE rules adapted to their own writing practices. Also training courses in CL focus on syntactic and stylistic rule matters.

In practice however, the biggest workload is involved in the development and maintenance of the lexicon. Two issues are at stake, here. First, there is the General Vocabulary. which has been defined in AECMA as a rather well-defined and finite set of approximately 2.000 conformant and 1.000 non-conformant words. An organization that wants or needs to define its own General Vocabulary will have to go through the process of defining its own General Vocabulary almost from scratch. Usually, the underlying norms tend to be less restrictive than in the AECMA case.

Second, apart from the General Vocabulary, there is the issue of the domain-specific terminology. In AECMA they are called Technical Terms and Manufacturing Processes. The big issue here is to adhere to the principle of a one-to-one correspondence between word forms and concepts, to disallow synonyms, homonyms, orthographic and morphological variants, and to define clear and sound criteria for controlled terminology. Once this is done, all terms need to be coded into the CL-system and, in case of machine translation, corresponding transfer and target terms need to be defined and implemented.

In our view, it is important that a terminology management system is used that is able to store information for human lookup as well as automatic processing. In addition, we believe that the terminologist should play a very central role in the CL business case. Although terminological input may be provided by designers, manufacturers, implementers, authors, editors, translators, and so on, the creation, standardization, coding, storing and maintenance of technical terms should be done at a central place, and disseminated from this place to the company and the outside world. This is one of the main reasons why in CL design a client-server approach is preferable to the standalone approach.

Unfortunately, we have learned that there is a tendency to underestimate the importance of terminology standardization. The dichotomy between grammar and terminology is not so much an issue of controversy than one of awareness. Terminology standardization immediately affects business processes such as knowledge management and communication within the company. It also immediately affects the consistency and readability of the documentation, and the workload involved in translation. The more standardized a source terminology, the more consistency in translations, and the lesser the work load involved in creating and maintaining target terminology systems.

Defining and implementing a controlled terminology is a huge task, usually underestimated in workload. It is moreover usually forgotten that this will be an ongoing task, because terminology always changes and is never complete.

7 CL Checkers vs. Grammar and Spelling Checkers

Introducing a controlled language is a much more radical change in a company's business process than the check step that is normally added to verify the quality of a source text before submitting it for automatic translation. Any source text must be grammatically correct, lexically covered by the machine-translation system, and it must follow certain style guidelines (e.g., with regard to average sentence length) in order to achieve an understandable translation from the system.

Introducing a CL not only increases the check step, but also shifts it to the authors, and adds it to the authoring process as an extra activity. From the author's point of view the writing task becomes more complex, and in his eyes the distinctions between different types of errors, such as spelling, lexicon, grammar and style errors may be irrelevant in terms of their effect on readability and translatability. Consequently, he will ask for a tool that combines orthographic, grammatical and stylistic checking and integrates with his normal authoring environment. This requirement is not fully compatible with a high-quality, full-parse CL checker, because the latter expects linguistic input that is correct in orthographic and grammatical respect (see section 3).

A full-parse checker will signal orthographic and grammatical errors as anomalies, but the real error type will normally not be diagnosed, and the results on real CL deviations will tend to be less reliable. A checker and corrector functionality for ungrammatical input may be added to a full-parse checker by enlarging the scope of the grammar and relaxing certain conditions on existing rules. However, when the search space for analysis is enlarged, the number of possible interpretations and the chances for misinterpretations increase as well. Therefore, the accuracy of a grammar checker will never reach that of a style checker.

The best way to deal with this problem is to combine different approaches and tools in a text-oriented workbench, analogous to the translator's workbench. Flat formalisms will do for spelling and grammar tools. A more principled approach will yield better results for style checking tools. A checking memory will enhance the quality of the proposed corrections for both formalisms. Authors can personalize their environments by switching on and off tools and rules, as they proceed with their re-writing task.

8 English vs. Other Languages

Most CL checkers deal with English. Many reasons can be given for this. First of all, English is the most influential language in the world (Weber 1997), not in terms of native speakers, but in terms of native and non-native users. English is also the language that is most often used for technical documentation and the language most often translated. Accidentally, English is also a language that is rather difficult to parse due to its heavy homography and ambiguity. This makes it particularly well suited for CL, although controlled variants of other languages, such as French (Barthe, e.a. 1999), German (Schachtl 1996, Schmidt-Wigger 1998), Swedish (Almqvist and Sågvall Hein 1996), Japanese (Shirai e.a. 1998) and Chinese (Zhang and Shiwen 1998) have been reported in the literature.

Also, English is very often used as a language for relay translation. Technical documents are originally written in a non-English source language, and after being translated into English, the English translation is used as the source for translation into other target languages. This raises the question as to whether an English CL checker can be successfully put into use in environments where English is not the native language of the originator of the documentation.

In such a situation, an organization might decide that all source documents are written in controlled English and then translated into other languages, including the native language. This requires an even more radical change than the introduction of CL, and it is feasible only when near-native competence of English is available. An alternative option is that the source documentation is written in the native language, translated into controlled English and translated from the latter into other languages. Here, it should be kept in mind that translating an uncontrolled version into controlled English implies more reformulation and rework of text than is necessary with an uncontrolled target language (cf. Barthe e.a. 1999). Thus, translators have to work in very close co-operation with the authors, should this option be considered.

References

AECMA, Simplified English. Document PSC-85-16598. Issue 1, Revision 1, 1998.

Almqvist, I. and A. Sågvall Hem (1996). "Defining Scania Swedish – A Controlled Language for Truck Maintenance". In CLAW 96. Proceedings of the First International Workshop on Controlled Language Applications, Leuven 1996. 159-165.

Barthe, K. e.a. (1999). GIFAS Rationalized French: A Controlled Language for Aerospace Documentation in French. Technical Communication. Second Quarter 1999, 220-229.

Huijsen, W. (1998). "Controlled Language - An Introduction". In CLAW 98, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh 1998, 1-15.

Knops, U. and Depoortere. B. (1998). "Controlled Language and Machine Translation". In CLAW 98, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh 1998. 42-50.

Means, L. and Godden, K. (1996). "The Controlled Automotive Service Language (CASL) Project". In CLAW 96, Proceedings of the First International Workshop on Controlled Language Applications, Leuven 1996, 106-114.

Pym, P.J. (1990). "Pre-Editing and the Use of Simplified Writing for MT: An Engineer's Experience of Operating an MT System". In P. Mayorcas (ed.). Translating and the Computer 10: The Translation Environment 10 Years On. ASLIB 1990, 80-95.

Schachtl, S. (1996). "Requirements for Controlled German in Industrial Applications". In CLAW 96, Proceedings of the First International Workshop on Controlled Language Applications, Leuven 1996, 143-149.

Schmidt-Wigger, A. (1998). "Grammar and Style Checking for German". In CLAW 98, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh 1998, 76-85.

Shirai, S., S. Ikehara, A. Yokoo and Y. Ooyama (1998). "Automatic Rewriting Method for Internal Expressions in Japanese to English MT and Its Effects". In CLAW 98, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh 1998, 62-75.

Weber, George (1997). "Top Languages". In Language Today, December 1997, 12-18.

Wojcik, R. (1998). "AECMA Simplified English and Controlled Language Checkers"" In ELRA Newsletter, 1998, 10-11.

Zhang, W. and Y. Shiwen (1998). "Construction of a Controlled Chinese Lexicon". In CLAW 98, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh 1998,