Features of an Error Correction Memory to Enhance Technical Texts Authoring in LELIE

Patrick Saint Dizier

Features of an Error Correction Memory to Enhance Technical Texts Authoring in LELIE

Patrick SAINT-DIZIER*

ABSTRACT

In this paper, we investigate the notion of error correction memory applied to technical texts. The main purpose is to introduce flexibility and context sensitivity in the detection and the correction of errors related to Constrained Natural Language (CNL) principles. This is realized by enhancing error detection paired with relatively generic correction patterns and contextual correction recommendations. Patterns are induced from previous corrections made by technical writers for a given type of text. The impact of such an error correction memory is also investigated from the point of view of the technical writer's cognitive activity. The notion of error correction memory is developed within the framework of the LELIE project an experiment is carried out on the case of fuzzy lexical items and negation, which are both major problems in technical writing. Language processing and knowledge representation aspects are developed together with evaluation directions.

Keywords:
error correction memory, controlled natural languages, natural language processing, logic programming

1. Introduction

1.1. Technical documents and authoring guidelines

Technical documents form a linguistic genre with specific linguistic constraints in terms of lexical realization, syntax, typography and overall document organization, including business or domain dependent aspects. Technical documents cover a large variety of types of documents: procedures, equipment and product manuals, various notices such as security notices, regulations of various types (security, management), requirements and specifications. These documents are designed to be easy to read and as efficient and unambiguous as possible for their users and readers. They must leave little space for personal interpretations. For that purpose, they tend to follow relatively strict controlled natural language (CNL hereafter) principles concerning both their form and contents. These principles are described in documents called authoring guidelines. There are general purpose guidelines, among which various norms in e.g. aeronautics, chemistry, or guidelines proper to a company. When these latter are not coherent with the former, they tend to be preferred to the general guidelines. There

* IRIT-CNRS, Toulouse, France (stdizier@irit.fr)

are several guidelines but no general consensus on what they should contain. Finally, depending on the industrial domain, the traditions of a company, the required security level, and the target user(s), major differences in the writing and overall document quality are observed.

Guidelines may be purely textual, i.e. their application is manually controlled by technical writers. Guidelines may also be implemented via templates also called boilerplates that authors must use. These are primarily designed for unexperienced authors, in particular for producing simple texts and requirements.

Guidelines are extremely useful to produce texts which are easy to interpret by users: technical texts are oriented towards action, However, guidelines are often felt to be too rigid, they are often felt to lack the flexibility and the context sensitivity that technical writers need in a number of situations. For example, uses of fuzzy lexical items may be acceptable in certain contexts (*progressively close the tap*) since 'close' is a punctual action in this context. Next, a strict observation of CNL principles may lead to very complex formulations: correcting some errors may be counterintuitive and complex from a cognitive point of view. For example, negation may be permitted when there is no simple alternative (*do not threw in the sewer*) since enumerating the possibilities may be long and context dependent. Similarly, complex sentences may be left as they are when their reformulation into several sentences makes the understanding even more difficult, e.g. with the use of various forms of references. Finally, besides their lack of flexibility, authoring principles and guidelines, in the everyday life of technical writers, are often only partly observed, for several reasons in including workload, authoring habits and the large number of more or less consistent revisions made by several actors on a given text. As a result, and in spite of several proof readings made by technical writers and validators, it turns out that most technical texts still contain major authoring problems that must be improved.

1.2. The LELIE project

These considerations motivated the development of the LELIE project (Barcellini et al., 2012), (Saint-Dizier, 2014), which is a system that detects several types of errors in technical documents, whatever their authoring and revision stages are. Lelie produces **alerts** related to these errors on terms, expressions or constructions that need various forms of improvements. By error, we mean the non-observation of an authoring rule in the guidelines that is applicable to the situation, for example the use of passives, modals or negation which must be avoided in instructions. LELIE also allows to specify business constraints such as controls on style and the use of business terms. The errors detected by LELIE are typical errors of technical texts (e.g. Table 1), they are not errors in ordinary language. To overcome difficulties such as the lack of flexibility and context sensitivity in Lelie (error detection is rigid and realized on a strict word occurrence basis), our aim is to pair Lelie with a mechanism that introduces flexibility and context sensitivity in error detection and correction, using techniques that complement Lelie's rule-based approach. We show in this paper that this can be realized via an error correction memory.

Error detection in LELIE depends on the discourse structure: for example modals are the norm in requirements (Grady, 2006) but not in instructions. Titles allow deverbals which are not frequently

admitted in instructions or warnings. LELIE is parameterized and offers several levels of alerts depending on the a priori error severity. LELIE and the experiments reported below have been developed on the logic-based <TextCoop> platform (Saint-Dizier, 2012). Lelie is fully implemented in Prolog; its kernel is freely available for French and English. The output of LELIE is the original text with annotations.

Table 1 below shows some major errors found by LELIE, statistics have been realized on 300 pages of proofread technical documents from companies A, B and C (kept anonymous). The results presented in this table show that there are still many errors of various types that need to be fixed and therefore space for tools that help writers to improve their texts. Error rates are given for 30 pages which corresponds to an average size technical document in our corpus.

Table 1. Errors found in technical texts for companies A, B and C

error type	number of errors for 30 pages	A	B	C
fuzzy lexical items	66	44	89	49
deverbals	29	24	14	42
modals in instructions	5	0	12	1
light verb constructions	2	2	2	3
pronouns with unclear reference	22	4	48	2
negation	52	8	109	9
complex discourse structures	43	12	65	50
complex coordinations	19	30	10	17
heavy N+ N or noun complements	46	58	62	15
passives	34	16	72	4
future tense	2	2	4	1
sentences too complex	108	16	221	24
irregular enumerative construction rate	average	low	high	average
incorrect references to sections or s	13	33	22	2

These results show that there is an average of about one error every 3 to 4 lines of text. The alerts produced by the LELIE system have been found useful by most technical writers that tested the system. The main reactions and comments of technical writers indicate that:

- false positives (about 25 to 30% of the alerts) must be filtered out. This is essentially due to the lack of context sensitivity of error detection rules, e.g. progressively in *progressively close the pipe* is judged not to be fuzzy ('close' is almost punctual), whereas *progressively decrease the air speed* is fuzzy since 'decrease' is durative.
- errors which are not detected (about 8 to 10%) should be limited as much as possible to guarantee a good performance level. Non detection originates from incomplete lexical resources or ambiguities between business terms and ordinary language expressions. For example, a fuzzy lexical item inside a business term must not trigger an alert.

- severity levels must be finely tuned so that technical writers can organize their revisions, starting e.g. by the most severe errors. More generally, an error detection system must be very flexible w.r.t. the writer's practices.
- help must be provided to technical writers under the form of correction recommendations, whenever possible.
- corrections made by technical writers should be memorized so that they can benefit others and, in the long term, allow homogeneous corrections over a whole team of authors and be re-used as a tutoring system for novices.

The present paper aims at specifying, developing and testing several facets of an **error correction memory** system that would, after a period of observation of technical writers making corrections from the LELIE alerts, develop flexibility and context sensitivity in error detection and correction. This includes the two following operations:

1. memorize errors which are not or almost never corrected so that they are no longer displayed in texts in the future (false positives) and
2. memorize corrections realized by writers from which typical correction recommendations can be induced.

1.3. Pairing LELIE with an error correction memory

An error correction memory is a necessary complement to Lelie to produce technical documents with an efficient and contextually accurate controlled natural language level. Indeed, systems such as LELIE, are rule-based. Rules can detect the main situations. Filters can be paired with these rules, however, these cannot be multiplied indefinitely. A hybrid approach that pairs a generic rule-based system with a learning procedure that observes in an accurate manner the technical writer's correction activity to adjust errors seems to be relevant and operational.

In this paper, we develop our analysis and the features of an error correction memory. Our approach is based on a two level organization:

- the development of relatively generic **correction patterns**, which correspond to a common correction practice of a number of technical writers. These are stable over a domain, a company or a type of text. These patterns are induced from the general behavior of technical writers when they make corrections. They may therefore contain underspecified fields.
- the development of accurate **contextual correction recommendations**, based on previously memorized and analyzed corrections made on a small set of closely related terms and situations in context. These are paired with the generic correction patterns, they suggest values for the underspecified Recommendations allow to add more precise data to the generic patterns so that they are more relevant in precise situations.

For example a fuzzy adverb such as *about* in 'about 5 minutes' may possibly be replaced by

a pattern such as: 'between X and Y minutes', but the boundaries of this interval, X and Y, depend on the context. Their instantiations will be realized via recommendations.

Generic patterns as well as recommendations are induced from a collection of correction situations which have been previously observed. However, correction divergences between technical writers often arise; therefore, a strict automatic learning process is not totally satisfactory. The objective is rather to propose to a team of technical writers several possible corrections (via simple generalizations on coherent subsets of corrections) and to let them decide on the best solution, via discussion, mediation, or via a decision made by an administrator. For errors with a straightforward correction, general correction patterns may be proposed a priori, and possibly tuned by a validator.

This paper is organized as follows. Section 2 develops related work aspects, and somewhat elaborates the differences between our approach and existing projects. Section 3 is twofold: it first discusses the impact of such a system on the technical writer's cognitive activity and then develops the experiments that were made and the resulting formalism. Section 4 and 5 develop two major typical cases of CNL which are quite different: correcting fuzzy lexical items, which is essentially of a lexical nature, and the correction of negation and negative expressions, which is much more grammatical and involve access to knowledge. Our approach is applicable to other types of errors such as passives, deverbals, modals, light verb constructions, complex sentences, etc. Section 6 develops some aspects of the implementation and the various facets of an evaluation. Section 7 concludes with perspectives.

2. Related Work

The approach of an error correction memory that (1) includes flexibility and context sensitivity in the detection of errors and that (2) helps technical writers by providing them with error corrections validated and made homogeneous over a whole team of technical writers, via discussion and mediation, seems to be new to the best of our knowledge. It is a crucial tool, paired with a rule-based system such as LELIE, for improving the language quality of technical texts following CNL principles, which are felt to be too rigid. This tool is useful since correcting errors in texts is indeed a time consuming, painful and costly task. It should also make corrections more homogeneous over large texts and should avoid to introduce new errors when making corrections.

This notion of error correction memory originates from the notion of translation memory, it is however substantially different in its principles and implementation. An in-depth analysis of memory-based language processing is developed in (Daelemans et al., 2005) and implemented in the TiMBL software. These investigations develop several forms of statistical means to produce generalizations in syntax, semantics and morphology. They also warn against excessive forms of generalizations. (Buchholz, 2002) develops an insightful memory-based analysis on how grammatical constructions can be induced from samples. Memory-based systems are also used to resolve ambiguities, using notions such as analogies (Schriever et al. 1989). Finally, memory-based techniques are used in programming language support systems to help programmers to resolve frequent errors.

Guidelines for writing documents following controlled languages have been elaborated in various sectors by user consortium and companies, resulting in a large diversity of specifications (e.g. AECMA,

ASD-STE, SLANG, Attempto simplified English, see also (Wyner et al., 2010) for some implementations). Norms tend to emerge, such as SBVR for business rules and OMG-INCOSE for requirements (Hull et al., 2011). The reader is invited to consult a detailed synthesis and classification of CNL principles and projects given in (Kuhn, 2014). Furthermore, investigations on grammars for CNL is investigated in (Kuhn, 2013). More specialized analysis for critical systems are given in e.g. (Tommila et al., 2013). Concrete examples and training for writing in CNL are summarized in e.g. (Alred, 2012), (Umwalla, 2004), (O'Brian, 2003) and (Weiss, 2000).

A number of systems have been developed in the past years to help technical writers to produce documents that follow CNL guidelines. These systems include facilities to introduce domain considerations via a user interface. This allows, for example, an author to indicate preferred terms, fuzzy lexical items which can be allowed, or specific lexical data that must be avoided, buzz words, etc. This view is developed in e.g. in (Ganier et al., 2007), where the need of a flexible approach to CNL is advocated, from an ergonomics and conceptual point of view. A number of these systems are briefly presented below, and their features w.r.t. the present work are outlined.

ACE (Fuchs et al., 2008, 2012), stands for Attempto Controlled English. This system was initially designed to control software specifications, and has been used more recently in the semantic web. ACE contains a powerful parser, with a large lexicon of more than 100,000 entries, which produces Discourse Representation Structures (DRS). CNL principles are introduced via an automatic and unambiguous translation into first-order logic. The most notable features of ACE in terms of analysis include complex noun phrases, plurals, anaphoric references, subordinated clauses, modality, and questions. ACE is paired with RACE, a reasoning component that carries out consistency checking, proving and query answering. ACE has an editor that allows users to specify data and make reference to previous texts. It is applied to English, it does not propose corrections when errors are detected, but examples are provided to help the technical writer.

PENG (Processable English (White et al., 2009)) is a computer-processable controlled natural language system designed for writing unambiguous and precise specifications. It covers a strict subset of standard English and is precisely defined by a controlled grammar and a controlled lexicon. An intelligent authoring tool indicates the restrictions while the specification of the grammar is written, which makes the system flexible and adaptable to several contexts. The controlled lexicon consists of domain-specific content words and predefined function words that can be defined by the author on the fly. Texts written in PENG can be deterministically parsed and translated into discourse representation structures and also into first-order predicate logic for theorem proving. During parsing of a PENG specification, several operations are performed: anaphoric references are resolved, ellipsis are reconstructed, synonyms, acronyms, and abbreviations are checked and replaced, a discourse representation structure is constructed, etc., and a paraphrase is generated that shows how the machine interpreted the input. PENG has a power globally comparable to ACE, with some flexibility and some facilities for users, including the treatment of exceptional cases. It however does not have any correction facilities.

RUBRIC, a Flexible Tool for Automated Checking of Conformance to Requirement Boilerplates, (Chetan et al., 2013) is dedicated to requirement control. It is based on a kind of grammar that describes the structure of the various boilerplates the users want to use, based on Rupps Boilerplates

syntax. The parsing allows the recognition of ill-formed structures. Besides boilerplate recognition, this system has rather simpler controls on CNL constraints (called Natural Language Best Practices Checking) such as absence of negation, no passives, conjunctions, controls on fuzzy and vague terms, etc. which are potentially problematic constructs in requirements. This tool does not propose any correction help. The RAT-RQA system, developed by the Reusecompany for authoring requirements has similar properties. It is developed for a number of languages and is connected to the IBM Doors requirement management system. Finally, let us mention RABBIT, designed for developing control natural language for authoring ontologies in a form understandable to people, in opposition to e.g. OWL.

3. Main Features of an Error Correction Memory System

In this section, the features and the impact of an error correction memory are analyzed w.r.t. the language performance and cognitive activity of technical writers: how this system affects and improves their activity and what could be the real benefits when it is operational and accepted. Then we develop the experiments that lead to the system: the corpus and the memorization of corrections. Finally, we develop more formal aspects concerning the formalism of correction rules, their associated contexts of use, and the way error correction rules are induced from previously made corrections.

3.1. Technical writer behavior analysis

The first step in the definition of an error correction memory is to observe how technical writers work from a linguistic and cognitive point of view with the alerts produced by LELIE. Alerts are inserted in the original document in various ways, depending on the authoring tool that is used. For example, alerts are displayed as specific comments in MS Word. In Editorial suites such as Scenari, which has been connected to Lelie, alerts appear as text bubbles or paper pads with links to explanations and definitions. Corrections can therefore be made on the original document.

In our experimentation, the following questions, crucial for the design of a controlled natural language system, have been considered:

- What do technical writers think of the relevance of each alert?
- What are the strategies deployed by technical writers to make corrections?
- Over large documents, how do they manage to produce stable and homogeneous corrections?
- How much of the text fragment where the error occurs is modified? Does the modification affect the sentence content?
- How difficult is a correction and what resources are required (e.g. external documentation, asking someone else)?
- How many corrections have effectively been done? How many are left pending and why?

Two small groups of technical writers, all native speakers, respectively composed of 3 and 4 persons with different ages (from novices of about 23 years old to senior writers of more than 20 years experience), technical qualification (from novice in the area to expert) and authoring experience, have been observed in the areas of energy and transportation. These writers are full-time working in technical writing and have received a theoretical and practical training in the area, their level is a master in technical writing. Our main observations are the following:

1. Most technical writers have first a quick glance at the set of alerts produced by LELIE on the text to improve, to estimate the workload and how they are going to proceed. They can ask LELIE to only produce certain types of alerts, and then proceed gradually,
2. About half of them made corrections in the order they appear, starting from the beginning of the document, while others proceeded by type of alert, in general correcting the ones they feel are the most crucial first, or those they can fix quickly and easily,
3. Given an alert:
   (a) they make the correction (in about 60% of the cases), possibly add a comment or a justification.
   (b) they make the correction, but realize that the problem is larger than indicated by the alert and make additional corrections in the text segment around the alert,
   (c) they make the correction but at the same time they make another type of error, for example, when simplifying a complex sentence, they introduce a pronoun with an unclear antecedent,
   (d) they understand the alert, but they cannot make the correction because they do not know how to make it or too much time consuming and they leave it for later (about 15% of the cases),
   (e) in this case, they often look in the text for similar errors, in order to get ideas on how to make the correction, or they directly ask for help to a more experienced colleague,
   (f) they understand the alert, but consider that it is a minor error and decide to leave the text unchanged,
   (g) they analyze the alert as inappropriate or they do not understand it: they leave the text unchanged (about 20% of the cases for these two latter situations).
4. in terms of corrections, in a large majority of cases, technical writers make minimal corrections, in order not to alter too much the original text and to avoid making new errors,
5. they have no predefined and stable correction strategy, due to the large variety of errors with their context,
6. in terms of external documentation, technical writers consult previously written texts or technical documents such as product manuals. They also consult, for basic language aspects, synonym or antonym dictionaries (paper or internet). This means that some corrections may take a very long time, and, besides their crucial character, the induced cost is a parameter that technical writers consider,
7. Finally asking help to more experienced writers is frequent and represents an important workload for these latter writers, up to 50% of their time.

This analysis shows that only about 60% of the alerts are processed, while about 15% are left pending. It also shows that correcting an error is, by large, not only a language problem, but that business aspects are involved. These results show some useful features that an error correction memory should have:

- Corrections must take into account the error context for a higher accuracy,
- Corrections must result from a consensus among technical writers via an administrator or via mediation, following principles given in e.g. (Boule et al., 2005),
- These corrections are then proposed in future correction tasks in similar situations.

Corrections are made directly accessible to technical writers: a lot of time is saved and corrections become homogeneous over the various documents of the company. Corrections reflect a certain know-how of the authoring habits and guidelines of a company: they can be used to train novices.

3.2. Construction of the corpus

The goal is to evaluate the nature and form of corrections: what kind of prototypical forms emerge, their dependence to the utterance context and the type of lexical and grammatical resources which are needed to model and implement an error correction memory. For that purpose, we built a corpus of technical texts at different production stages and submitted it to LELIE.

Our experiments are based on a corpus of technical texts coming from seven companies, kept anonymous at their request. Our corpus contains about 120 pages extracted from 27 documents. The main features considered to validate our corpus are:

- texts corresponding to various professional activities: product design, maintenance, production launch, specifications, regulations and requirements,
- texts following various kinds of CNL recommendations, and business style and format guidelines imposed by companies,
- texts coming from various industrial areas: finance, telecommunications, transportation, energy, computer science, and chemistry.

Texts are in French or English in almost equal proportions. Technical writers were then asked to make corrections in these texts for as many alerts as possible.

3.3. Memorizing technical writers' corrections

An observation on how technical writers proceed was then carried out from our corpus. The tests we made do not include any temporal or planning consideration (how much time it takes to make a correction, or how they organize the corrections) or any consideration concerning the means and the strategies used by technical writers. At this stage, we simply examine the correction results, which are stored in a database. At the moment, since no specific interface has been designed, the initial and the corrected texts are compared once all the corrections have been made. The only exception are requirements since the Doors requirement management system keeps tracks of all modifications made by authors. The global process for memorizing corrections is the following in the context of fuzzy lexical items and negation:

- for a new fuzzy lexical item or negation that originates an alert, create a newentry in the database, include its category and a priori severity level for fuzzy terms. The severity level has two origins: the a priori severity level given in CNL recommendations, and the opinion of senior technical writers involved in our experiments. This severity level can be updated during the learning process. Severity mainly relies on the ambiguities which are generated or the difficulty to realize the task (e.g. close smoothly: what does this means and how to carry out this action?).
- for each alert concerning this item, include the whole sentence in which it appears in its database entry and the correction made by the technical writer. Indicate who made the correction (several writers often work on similar texts). Alerts as well as corrections are tagged. Changes are basically characterized by the use of new words or a different word order.
- If the initial sentence has not been corrected then it is memorized and no correction is entered.

The database is realized in Prolog as follows:

```
alert ( [ type of alert ], [ tagged term(s) ], [ lexical category ],
[ severity level (1 to 3) ],
[ [ sentence with alert, sentence after correction with tags,
ID of writer ], ....]).
```

For example:

```
alert ( [ fuzzy ], [ progressively ], [ adverb ], [3],
[ [ [ <fuzzy>, progressively, </fuzzy>, heat, the, probe ],
     [ [ <revised>, heat, the, probe, progressively,
               in, 5, seconds, </revised>]], [John] ] .... ]
```

The determination of the scope of the error (the word it applies to) and its context (defined in sections 3.4 and 3.5) is realized during the induction process (section 3.6) via lexical look-up or the use of a local grammar. This allows the induction parameters on the form of the scope and the context to be explored in a later stage without affecting the database.

3.4. Denition of Error Contexts

Let us now de ne the parameters for the taking into account of the context sensitivity of an alert so that corrections can be made flexible and parameterized depending on the context of the alert.

In our first experiment, an **Error Context** is a set of words which appears either before or after the alert (i.e. the fuzzy lexical item or the negation). In the case of fuzzy lexical items and negation, a context is composed of nouns or noun compounds (frequent in technical texts) N_i, adjectives A_k and action verbs V_j. Our strategy is to first explore the simple case of the use of a number of such terms to unambiguously characterize a context, independently of the alert category. In our experiment, the context is composed of (1) a main or head word (or expression) which is the word to which the fuzzy lexical item or the negation applies (e.g. 'fire alarms' in 'minimize alarms',

or 'close' in 'do not close the door while the system is in operation') and (2) additional words that appear either before or after the main one. The closest words in terms of word distance are considered. In case of ambiguity, the words in the same clause are preferred.

This approach has the advantage of not including any complex syntactic consideration. To evaluate the number of additional words which are needed in the context besides the head word, we constructed 42 contexts from 3 different texts composed of 2, 3, 4 and 5 additional words (including technical compound terms, e.g. 'trim management', which count each as a single word). We then asked technical writers to indicate from what number of additional words each context was stable, i.e. adding a new words does not change what it basically means or refers to. Over our small sample, results are the following:

Table 2. Size of contexts

number of additional words	stability from previous set
3	83%
4	92%
5	94%

From these observations, a context of 4 additional words (if these can be found in the utterance) in addition to the main word is adopted. This is probably a little bit vague, but sufficient for our present aim. This number of words is implemented as a parameter so that it can be revised in the future or depending on the type of document.

3.5. General form of patterns and correction rules

As indicated in 1.3, the definition of correction rules is based on a two level approach:

(1) the development of relatively generic **correction rules**, that reflect correction practices for a domain, a company or a type of text. These rules may be induced from the technical writer correction activity or may be defined a priori when they are linguistically or cognitively straightforward. They often contain underspecified fields.

(2) the development of accurate **contextual correction recommendations**, based on previously memorized and analyzed corrections made on a small set of closely related terms and situations in context. These recommendations correspond to the underspecified fields of the generic correction rules. They add flexibility and context sensitivity to the rules.

Correction rules are based on patterns that identify structures via unification. The first pattern identifies the error while the second one, based on the first proposes a correction:
[error pattern] → [correction pattern].

These patterns are based on the syntax of Dislog (Discourse in Logic, (Saint-Dizier, 2012)), a logic-based language that runs on the TextCoop platform. Patterns include specific variables that represent contextual recommendations. Besides its logic-based character, Dislog extends the expressive power of regular expression, in particular via (1) the introduction of feature structures which may

be typed, (2) negation on symbols which must not occur in the expression, (3) reasoning procedures and calls to knowledge for various controls and computations and (4) the construction of representations or results (e.g. corrections). These properties are necessary for the development of patterns and correction rules.

Patterns are finite ordered sequences of the following elements (external form):

- **terminal symbols** that represent words, expressions or punctuations, They are written in small letters,
- **preterminal symbols** are symbols which are derived directly into terminal elements. These are used to capture various forms of low level generalizations. Symbols can be associated with a type feature structure that encodes a variety of aspects of those symbols, from morphology to semantics. These symbols start by a capital letter.
- **non-terminal symbols** which can also be associated with type feature structures. These symbols encode 'local' syntactic constructions such as NPs, temporal expressions or domain specific constructs. These symbols start by a capital letter.
- **recommendation variables** which are represented by symbols such as: X, Y, Z, to differentiate them from non-terminal symbols,
- **optionality and iterativity marks** over non-terminal and preterminal symbols, as in regular expressions,
- **gaps** which are symbols that stand for a finite sequence of words of no present interest for the rule. These must be skipped. A gap can appear only between terminal, preterminal or non-terminal symbols.
- **contextual variables** standing for underspecified fields in the correction pattern, which will be instantiated by contextual recommendations.
- a few **functions** related to the domain ontology or other types of knowledge to produce corrected terms.

Pattern samples are:
[Neg VP before VP] where Neg is a preterminal element that stands for 'do not' or 'never' 'before' is a terminal element and VP is a verb phrase,
[in order not to V NP] similarly, 'in order not to' is a terminal element and VP and NP are non-terminal elements,
[Verb antonym (Adjective)] where antonym is a function, Verb and Adjective are pre-terminal elements,
[more than X Noun] where X is a variable representing a recommendation.

A correction rule is a rewrite rule that, given an error identified by a pattern (the error pattern), rewrites the text segment into a correct construction (the correction pattern), represented also by a pattern partly based on material from the error pattern. Such a correction rule used for fuzzy manner adverbs is e.g.:
[progressively VP (durative)] → [progressively VP (durative) in X (time)],
e.g. *progressively heat the probe X37 → progressively heat the probe X37 in 10 minutes.*

X (time) (variable of type time, 10 minutes, in the example) is suggested by the correction recommendation, the adverb is present in order to keep the manner facet which is not fuzzy, since it

is the temporal dimension that is fuzzy in this expression.

A contextual correction rule is a correction rule associated with a context. This allows the specification of a precise recommendation or possibly set of recommendations:

error pattern → correction pattern [[head term], [terms of Context]
[Values for contextual variables]],
e.g.: [progressively VP (durative)] → [progressively VP (durative) in X (time)] [[heat],
[probe X37, Airbus A320, pitot tube, icing conditions] [X (time) = 9 sec]].

In this example a recommendation of 9 seconds is provided for the time needed for 'heating' (head term, type durative) 'the probe X37 of the pitot tube of an airbus A320 in icing conditions' (context of 4 relevant terms). More examples are developed below in the sections dedicated to fuzzy lexical items and to negation.

3.6. The pattern and correction rules induction method

The induction method appropriate for our objective combines formal forms of generalizations with interactions with technical writers. We view it as a tool for technical writers so that they can develop correction rules in a simpler and more reliable way. This tool induces generic correction proposals from corrections already made in similar contexts. Of interest is the identification of those parameters needed by writers to adjust the tool to the way they conceive the correction process.

From a formal point of view, the induction mechanism is based on unification theory, in particular on the notion of least upper bound (lub) (Lloyd, 2013). In unification theory, lubs are in general computed from a type lattice following the formal model developed in e.g. (Pfenning, 1992). A relatively similar approach is called Order-Sorted Unification, which is often paired with forms of type inference (Baader et al., 1998). In our case, linguistic structures and terminologies are considered instead of a strict type lattice, however, lexical entries features and terminologies can be interpreted as typed constructions.

Let us now develop different steps of the induction mechanism. In patterns and contexts let us consider the following elements:

- error pattern: a structure Err composed of the word (or expression) W that triggers the error (e.g. a fuzzy term) and the term it immediately applies to, its restricted scope, called Scope, e.g. in *progressively heat...*, W = 'progressively' and Scope = 'heat' (and not the whole VP). This restricted scope can appear either before or after W.
- correction pattern: a structure Corr that corresponds to the correction. Err, W as well as Scope contents may be affected,
- context: a structure Cont composed of a list of words as defined in section 3.4. Scope of the error pattern is shared with the context.

The induction mechanism is organized around the idea that the correction pattern is the reference or the invariant, whereas generalizations can be made on the other structures: W, Scope and Cont to reach more abstract contextual error correction rules. Steps 1a,b,c are designed to produce a consistent set of corrections that has no duplicates. Steps 2a,b,c develop the three dimensions of

generalizations. Step 3 includes two generalizations.

The generalization steps are the following:

- **Step 1a**: *grouping similar error contextual corrections*. The task consists in constructing, from the error correction database, the set Err_1 of the different contextual error corrections where duplicates are eliminated and replaced by a frequency measure.
- **Step 1b**: *detecting correction inconsistencies*. The next task is to detect in the set Err_1 situations where W, Scope and Cont are similar but Corr is different. This means that a given error in the same context gets different corrections. This kind of inconsistency must be resolved by means of different strategies which can be parameterized: (a) a decision is made by an administrator or via mediation, (b) the most frequent correction Corr is kept, or (c) it is possible to generalize over the two corrections when there are minor differences between them. The result is the set Err2 of consistent contextual error corrections. Each error in a given context gets a unique correction.
- **Step 1c**: *observation of correctly realized cases*. Whenever possible, it may be of interest to confirm the choice of a correction via a search in the current texts of a similar situation (same pattern and context) that has been correctly written (no alert).
- **Step 2a**: *generalizations on Scope*, with W and Cont unchanged. The goal is to construct the set of all the occurrences of Scope with W and Cont fixed.

  Two strategies are possible which can be a parameter:

  (a) a generalization over all contextual error corrections in Err_2, as a result, Scope becomes the set (possibly organized) of all the constructions found where W and Cont are similar. Scope is defined as follows:

  $\{ Scope \mid (W\ Scope \rightarrow Corr,\ Cont) \}$, or

  (b) a set of generalizations based on a partition $P(T)$ defined from the semantic types $T = \{t_1, ..., t_n\}$ mentioned in the lexicon of the head words of all the candidates for Scope, where: $P(T) = \{ T_1, ...T_j ,..., T_k \},\ k < n$.

  The result is a set of sets S_{Ti} of contextual error corrections from the elements in the partition T_i. This set is composed of elements of the form:

  $\{ Scope\,(T_i) \mid (W\ Scope\,(T_i) \rightarrow Corr,\ Cont) \wedge T_i \in \{ t_i, ..., t_{i+m}\}\}$.

  The partition of the types t_i is complete and unambiguous since, due to step 1b, there are no duplicates and inconsistencies in the set of contextual error corrections.

- **Step 2b**: *generalizations on context Cont* (except Scope), with W, Scope and Corr unchanged. Generalizations on a context Cont must be carried out with care. In our experiment, we propose the following generalization parameters: morphological variants (plural vs singular, verb tenses) and terminological variants (immediate sisters and parents in the domain ontology, except on named entities which are not abbreviations). Going beyond these generalizations (e.g. skipping a word) seems to lead to overgeneralizations. Note that the word order in Cont is not relevant. In terms of generalization, a single rule is produced which accommodates all the variants $Cont_i$ of Cont. The set CONT of contexts $Cont_i$ is defined by:

$CONT = \{ Cont_i \mid (W\ Scope \rightarrow Corr,\ Cont_i) \}$
Then a single contextual correction rule is produced which has the following skeleton:
$(W\ Scope \rightarrow Corr,\ Cont_i) \wedge Cont_i \in CONT \}$.

- **Step 2c**: *generalizations on the error word W*, keeping the others unchanged. The goal is to generalize over terms which have a relatively close semantics and which behave similarly in terms of corrections, e.g. *a few, a small number of, a limited amount of*. This form of generalization is in general limited since the lexicon of technical terms is rather restricted. In terms of generalization, the set W of words W_i is defined by:
$W = \{ W_i \mid (W_i\ Scope \rightarrow Corr,\ Cont) \}$
Then a single contextual correction rule is produced which has the following skeleton:
$(W_i\ Scope \rightarrow Corr,\ Cont) \wedge W_i \in W\}$.

- **Step 3**: *generalization over two factors*. This step is more exploratory at the moment. The pair we are exploring is (Scope, Cont) which is of a similar nature and Scope is included in fact into Cont with a special status. We think that going beyond this level of generalization leads to overgeneralizations.

3.7. Overall organization of the system

The overall organization of the system is rather simple, it is basically an extension to Lelie connected to the text editor used by the company at stake. The main components of the error correction system include resources and processing modules which are specific to this task.

The main resources are:

- lexicons of e.g. fuzzy lexical items. These are categorized by means of lexical categories (e.g. adverb, adjective, etc.), severity level, semantic features (e.g. for adverbs: manner, temporal, for verbs: WerbNet classes)). Additional, domain specific, features can be added to allow more accurate and relevant forms of generalizations. This allows the definition of the types required for the induction step presented in the previous section,
- local grammars, e.g. to identify a construction that is a negation or an adverbial phrase where the head term is fuzzy,
- the contextual error corrections database.

The main processing components are:

- the contextual error correction rule induction mechanism, with its connection to the error database,
- the module that interacts with technical writers to propose generalizations in a readable manner,
- the modules that connect Lelie (rules and resources) with the induction mechanism.

All these modules are written in Prolog SWI, using the kernel syntax so that they can be used in various implementation contexts.

4. The case of fuzzy lexical items

4.1. Linguistic aspects of fuzzy terms

A first experimentation of the principles developed above is devoted to the case of fuzzy lexical items which is a major type of error, very representative of the use of an error correction memory. Roughly, a fuzzy lexical item denotes a concept whose meaning, interpretation, or boundaries can vary considerably according to context, readers or conditions, instead of being fixed once and for all. A fuzzy lexical item must be contrasted with underspecified expressions, which involve different forms of corrections. For example, a verb such as *damaged* in *the mother card risks to be damaged* is not fuzzy but underspecified because the importance and the nature of the damage is unknown similarly for *heat the probe to reach 500 degrees* because the means to heat the probe are not given but are in fact required to realize the action. In terms of corrections, an underspecified expression often requires a complement or an adjunct to be added, e.g. *using a voltage of 50 Volts*. This adjunct is highly dependent on the domain and on the user knowledge.

There are several categories of fuzzy lexical items which involve different correction strategies. They include a number of adverbs (manner, temporal, location, and modal adverbs), adjectives (*adapted, appropriate*) determiners (*some, a few*), prepositions (*near, around*), a few verbs (*minimize, increase*) and nouns. These categories are not homogeneous in terms of fuzziness, e.g. fuzzy determiners and fuzzy prepositions are always fuzzy whereas e.g. fuzzy adverbs may be fuzzy only in certain contexts. The degree of fuzziness is also quite different from one term to another in a category.

The context in which a fuzzy lexical item is uttered may have an influence on its severity level. For example 'progressively' used in a short action (*progressively close the water pipe*) or used in an action that has a substantial length (*progressively heat the probe till 300 degrees Celsius are reached*) may entail different severity levels because the application of 'progressively' may be more difficult to realize in the second case. In the case of this adverb, it is not the manner but the underlying temporal dimension that is fuzzy. Finally, some usages of fuzzy lexical items are allowed. This is the case of business terms that contain fuzzy lexical items which should not trigger any alert. For example, *low visibility landing procedure* in aeronautics corresponds to a precise notion, therefore 'low' must not trigger an alert in this case. The equivalent, non-business expression *landing procedure with low visibility* should probably originate an alert on 'low', but there is no consensus among technical writers.

In average, 2 to 4 fuzzy lexical items are found per page in our corpus. On a small experiment with two technical writers from the 'B' company, considering 120 alerts concerning fuzzy lexical items in different contexts, 36 have been judged not to be errors (rate: 30%). Among the other 84 errors, only 62 have been corrected. The remaining 22 have been judged problematic and very difficult to correct. It took between 2 and 15 minutes to correct each of the 62 errors, with an average of about 8 minutes per error. Correcting fuzzy lexical items indeed often requires domain expertise.

4.2. A lexicon of fuzzy lexical items

In the Lelie system, a lexicon has been implemented that contains the most common fuzzy lexical items and fuzzy expressions found in our corpus (about 450 terms). Since some items are a priori more fuzzy than others, they have been categorized by means of semantic features and a mark, between 1 and 3 (3 being the worse case) has been assigned a priori to each subcategory. This mark is however not fixed, it may evolve depending on technical writers' behavior. This mark has been defined according to the same methodology as developed in the above section, and with the same group of technical writers. This ensures a certain homogeneity in the judgements.

Such a lexicon is necessary so that the fuzzy term identification process can start. For illustrative purposes, Table 3 below gives figures about some types of entries of our lexicon for English.

Table 3. Main fuzzy lexical classes

category	number of entries	a priori severity level
manner adverbs	130	2 to 3
temporal and location adverbs	107	in general 2
determiners	24	3
prepositions	31	2 to 3
verbs and modals	73	1 to 2
adjectives	87	in general 1

4.3. Some typical error correction memory scenarios for fuzzy lexical items

Error correction memory scenarios include the following main situations, that we have observed in various texts of our corpus:

1. A fuzzy lexical item that is **not corrected** over several similar cases, within a certain word context or in general, no longer originates an alert. A threshold of 3 to 5 alerts without any corrections in the same context before this decision can be validated seems appropriate, but may depend on the alert frequency. The corresponding fuzzy lexical item in the LELIE lexicon then becomes inactive for that context, e.g. in *to minimize fire alarms*, 'minimize' describes a general behavior, not something very specific, it is therefore no longer displayed as an alert. Same situation for 'easy' in *a location that allows easy viewing during inspection*.
2. (2a) A fuzzy lexical item that is **replaced or complemented by a value, a set of values or an interval**, may originate, via generalizations, the development of correction patterns that require e.g. values or intervals. This is the most frequent case. For example, from examples such as:
   progressively close the pipe → progressively close the pipe in 30 seconds.
   Progressively heat the probe → heat the probe progressively over a 2 to 4 minutes period.
   The power must be reduced progressively to 65% to reach 180 knots → reduce the power to 65% with a reduction of 10% every 30 seconds to reach 180 knots.

   A correction pattern could be the association of 'progressively' (to keep the manner and its continuous character) with a time interval, possibly complex, as in the second example.

3. A fuzzy lexical item that is simply **erased** in a certain context (probably because it is judged to be useless, of little relevance or redundant) originates a correction recommendation that specifies that it may not be necessary in that context. For example: *any new special conditions → any new conditions; proc. 690 used as a basic reference applicable to airborne → proc. 690 used as a reference*.... In these contexts, 'special' and 'basic' are fuzzy, but they have been judged not to be crucial, therefore they must be erased. These specific situations in context must be memorized to avoid alert production in similar cases.
4. A fuzzy lexical item may be **replaced by another term or expression in context that is not fuzzy**, e.g. *aircraft used in normal operation → aircraft used with side winds below 35 kts and outside air temperature below 50 Celsius*, in that case the revision of 'normal' in context is memorized and then proposed in similar situations. This type of correction is also typical of underspecified terms.
5. Finally a fuzzy lexical item may involve a **complete rewriting** of the sentence in which it occurs. This is a difficult case which must be avoided whenever possible because it often involves alterations of the utterance meaning.

A rough frequency indication for each of these situations is given below, based on 52 different fuzzy lexical items with 332 observed situations:

Table 4. Distribution of correction situations

case nb.	number of cases	rate (%)
1	60	18
2	154	46
3	44	13
4	46	14
5	28	9

It is important to note that, in a given domain, errors are very recurrent, they concern a small number of fuzzy terms, but with a large diversity of contexts. For example, a text of about 50 pages long contain between 5 to 8 fuzzy manner adverbs, which is very small and allows an accurate control in context.

4.4. A few typical error correction patterns for fuzzy lexical items

Correction patterns have been categorized considering (1) the syntactic category of the fuzzy item and (2) the correction samples collected in the database. Organized by syntactic category, here are a few relevant and illustrative types of patterns which have been induced:

- **fuzzy determiners**: their fuzzy character is not easy to circumvent, a solution is the specification, via a recommendation, of an upper or a lower boundary (X) or an interval, major patterns are: [a few Noun] → [less than X Noun].,

[most Noun] → [more than X Noun].
Besides patterns, which are generic, the context may induce a correction recommendation for the value of X: depending on Noun and its usage (context) a value or a few close values for X can be suggested, e.g. '12' in *take-off a few knots above V1 → take-off less than 12 knots above V1*, with Context = [main term: knots, additional: take-off, above V1].
In some cases, fuzzy determiners are not corrected because their severity level is really low or they are difficult to correct, as in e.g. 'few' in *wait for a few seconds that the light becomes stable*.

- **temporal adverbs**, combined with a VP with an action verb, such as *frequently, regularly*, are made more precise in a number of cases by the specification of a temporal value with an adequate quanti er, e.g.:
[regularly VP (action)] → [VP (action) every X (time)].,
and syntactic variants, where X (time) is a variable of type time that is instantiated on the basis of the context or the VP. For example: *regularly control the steam pressure → control the steam pressure every 10 minutes*.
The regular character of the control is fully integrated into the time period. An adverb such as progressively concerns both manner and time. It is associated with a temporal interval when it modifies a durative verb:
[progressively Verb (durative)] → [progressively Verb (durative) in X (time)].,
e.g. *progressively heat the probe in 10 minutes*. X(time) is suggested by the correction recommendation level, the adverb is left in order to keep the manner facet which is not fuzzy.

- **manner adverbs**, such as *carefully*, do not have any direct measurable interpretation. The strategy used by technical writers is:
  (1) to keep the adverb and produce a warning that describes the reasons of the care if there is a risk, or
  (2) to replace the adverb by an explanation on how to make the action in more detail, via a kind of 'zoom in', or
  (3) to simply skip the adverb in case it is not crucial.

  For example:
  [carefully VP (action)] → [carefully VP (action) Warning].,
  e.g. *carefully plug-in the mother card → carefully plug-in the mother card otherwise you may damage its connectors*. With the warning, the nature of what to care about is less vague.

- **prepositions** such as *near, next to, around, about* are often corrected using a strategy quite close to fuzzy determiner correction by the specification of a value or an interval of values that depends on the context. A pattern is for example:
[near Noun (location)] → [less than X (distance) from Noun (location)].,
where X(distance) depends on the context, e.g. *park near the gate → park less than 100 meters*

from the gate. The variable X(distance) is the correction recommendation, making the pattern more precise.

- **adjectives** such as *acceptable, convenient, specific* as in *a specific procedure, a convenient programming language* can only be corrected via a short paraphrase of what the fuzzy adjective means. For example, *convenient* may be paraphrased by *that has debugging tools*. Such paraphrases can be suggested to technical writers from the corrections already observed and stored in the error correction database.

At the moment, 27 non-overlapping patterns have been induced from the corpus. Error correction recommendations are more difficult to stabilize because contexts may be very diverse and related to complex business aspects. At the moment, (1) either a precise recommendation has emerged or has been found in non fuzzy text counterparts and has been validated or (2) the system simply keeps track of all the corrections made and displays them by decreasing frequency. In any case, the correction decision is always the responsibility of the technical writer.

5. The Case of Negation

5.1. Linguistic aspects of negation

Negation is a complex phenomenon, both from a semantic and pragmatic point of view (Horn, 2001), with cognitive aspects related to e.g. presupposition and implicature, whose control is important in technical documents to avoid any misconceptions. Negation is linguistically realized in technical texts in several ways: first by the use of the adverbs *not, never, no longer* etc. but also by the use of terms with a strong negative dimension: verbs (e.g. avoid and forbid verb classes), quantifiers (no, none), prepositions (without), prefixes (un-) or suffixes (-less), etc. A number of forms of negative expressions must not be corrected. For example, expressions describing system states must not be altered, e.g. *non available, invalid, failed, degraded*, etc.

Although most of the above categories are important aspects of negation, CNL mainly deals with the adverbial forms of negation (not, never, etc.). In most CNL recommendations, negation must be avoided. This rule is felt to be too rigid by technical writers. Indeed in warnings and in some types of requirements negation is appropriate, clear and unambiguous: *never close the door when the system is in operation otherwise*..... Negation may also be acceptable in conditional expressions, goal or causal expressions (acting e.g. as warning supports): *in order not to damage its connectors*. The LELIE system, paired with TextCoop allows the recognition of the main discourse structures found in technical texts (Saint-Dizier, 2014), therefore, it is possible to select those structures where negation must or must not originate an alert.

5.2. Some typical error correction memory scenarios for negation

Negation is much more difficult to correct than fuzzy lexical items. On the same corpus, our observations show that only about 50% of the alerts on negation produced by Lelie are judged to be relevant by technical writers, therefore there is about 50% of noise, which is very high. This is however not surprising because technical writers tend to say that an alert is not relevant when they do not know how to correct it without altering too much the text or producing a correction that is worse than the error. Among the 50% which are relevant, about 35% can be easily corrected while the remaining 15% must be corrected but their correction is difficult from a language or knowledge point of view.

In contrast with fuzzy terms, negation gets corrections which may be quite different in French and in English. Corrections may also be quite subtle and require a good command of the language. English glosses are given for French examples. Considering technical writers corrections, error correction memory scenarios include the following main situations:

1. **negation not corrected**: this situation is comparable to fuzzy lexical items, but with a higher frequency (about 40 to 50%). A threshold of 3 to 5 alerts without any corrections in the same context and with the same negative term is required to validate the fact that the expression with negation will no longer be displayed as an alert.
2. **useless uses of negation**: in a number of instructions, negation in French is simply used to make the statement stronger, it is a form of insistence. Negation can then be skipped, without any major alteration of the instruction contents:
   avant qu'elle ne soit requise → avant qu'elle soit requise (before it is required, 'ne' is the negation in this utterance).
3. **reference to an antonym**: in a majority of situations, corrections are realized by skipping the negation and using an antonym term for the term or expression the negation has scope over:
   the use of hydrogen is not forbidden → is allowed.
   The APU is no longer required → The APU is optional / useless.
   Here again, negation introduces a certain persuasion or emphasis that is lost with the antonym, which is more neutral, as shown in (Cruse, 1986).
   For example the combination of the negation 'not' with a negative verb 'forbidden' has a much stronger impact than just the neutral term 'allowed'. The next problem, illustrated by the second example, is the difficulty to identify the 'best' antonym term. Here 'optional' and 'useless' are candidates, where 'useless' is stronger but with a negative orientation. 'Optional' is in general preferred because it is more neutral.
4. **logical re-organization of the utterance** is another strategy we have observed. In this case, this may require the subtle use of a negatively oriented term, but this is judged to be softer than a real negation. This can be illustrated by:
   this does not require any borication → is made without any borication which is cognitively lighter to understand. Similarly for:
   in order not to risk to converge → in order to avoid to converge.

5. **structural re-organization** of a group of instructions: is a frequently encountered alert. It mainly concerns temporally or causally organized instructions which are not given in an optimal order: *do not unplug before stopping the machine → stop the machine. unplug it.*

5.3. Some error correction patterns for negation

Errors concerning negation are more difficult to generalize because they involve more aspects of syntax than fuzzy terms which are essentially lexical. Here are a few correction patterns induced from the technical writers' activity, these have been induced form our corpus.

- **useless forms of negation in French**: these often occur with a temporal expression such as 'avant que (before that)' followed by an NP followed by a negation and a verb in the subjunctive form:
  [avant que NP ne Ver b (subjunctive)] → [avant que NP Ver b (subjunctive)].
  Five such patterns have been produced for French. None has been observed for English.

- cases where a negative expression is replaced by its (or one of its) **antonym expression** is realized by several patterns depending on the lexical elements found before that expression. Patterns for French and English are for example:
  (French) [ne Verb pas Adjective / Past-participle] →
  [Verb antonym (Adjective / Past-participle)].
  (English) [Aux not Adjective / Past-participle] →
  [Aux antonym(Adjective / Past-participle)].,
  e.g.: *is not closed → is opened.*
  (French, English) [ne pas Verb (use) NP] → [Verb (use) hyperonym (NP) other than NP].
  do not use hydrogen → use a gas other than hydrogen:
  this pattern includes verbs of the 'use' family, where the main nouns of the NP has an hyperonym. This pattern has several variants for concrete actions with tools or products that must be used only in specic contexts.
  (French) with a condition: [ne Verb pas COND] → [Verb inverse (COND)].
  The English pattern is quite similar, e.g.
  does not increment if the pressure is lower than 3 bars → increments if the pressure is greater than 3 bars.
  In this example, the scope of the correction is quite large since it includes the rewriting of a condition.

- **logical reorganization** or event reorganization originate a few typical and quite generic patterns, among which:
  (French) [ne pas VP avant VP'] → [VP seulement apres VP']. or [VP'. Puis VP].
  (English) [do not/never VP before VP'] → [VP only after VP'] or [VP'. Then VP].
  For example *never unplug before the machine has been stopped. → stop the machine. then*

unplug it.

In this correction, the strength conveyed by 'never' is lost, however, marks such as 'rst ... and then' can be used to clearly mark the sequence or actions.

(French, English) [in order not to V NP/S] → [in order to antonym (V) NP/S] as in e.g.: *in order not to risk to converge → in order to avoid to converge.*

In this case 'avoid' has a negative connotation but it is felt to be more clear than the direct negation 'do not'.

A logical reorganization can entail new alerts, therefore it is important to have a correct analysis of induced patterns. For example, a frequent correction of:

all the X are not Adjective into *only some X are Adjective* introduces the fuzzy term 'some'.

At the moment 16 patterns have been identified, but this task is ongoing. In the case of negation, recommendations mainly concern functions that produce antonyms or hyperonyms. In general these are not unique and are largely contextual as outlined in the previous section. Recommendations mainly concern the choice of an antonym or an hyperonym that is the most relevant.

6. Implementation and Premises of an Evaluation

The main modules of this error correction memory system have been implemented in Prolog and connected to Lelie: the database, the induction algorithm and the automatic production of correction patterns with recommendations. These modules must now be integrated into each company's authoring system to be operational. Companies use a large diversity of editorial environments where the integration of the modules we implemented is possible via some substantial interface work. Systems like Doors (for requirement authoring and management) or Scenari (editorial suite) keep track of all changes including corrections made on texts, these traces can be re-used to construct the error correction database (section 3.3). The internal representations of most editorial systems are XML structures: it is therefore a priori possible to generate correction patterns and to use the man-machine interfaces of the editorial system at stake to generate readable correction patterns. These technical matters are out of the scope of this paper.

The integration of the error correction memory system into a company environment and information system is ongoing: it is a long and very technical task. At this stage, we can simply have a global evaluation of the improvement of the system in terms of noise reduction, carried out on a test corpus of 50 pages, from two companies. These evaluations are carried out with the collaboration of the two groups of 3 and 4 technical writers mentioned at the beginning of this article. Since they have very different skills, this allows us to have a number of useful reactions of different types.

The results are still preliminary and are more an estimation and indicative than final ones:

- for **fuzzy lexical items**, the initial noise of about 30% has been reduced to about 10%, therefore 20% of the noise has been eliminated. Most of the remaining 10% are due to situations not

found during the pattern constructions, or to fuzzy lexical items or expressions not identified previously.

- for **negation**, among the 50% judged to be real errors by technical writers, the patterns defined so far are relevant and used to correct 40%, therefore, only 10% are still problematic. Among the 50% judged not to be errors, about 25% are no longer displayed (they have been memorized not to be alerts), while the other 25% are still displayed. With a large set of observations, this level can be decreased. Some of these can, in the end, be analyzed as errors.

From a more global perspective, the evaluation of an error correction memory must be realized in a company's context from a functional point of view. The main areas we think are essential to be considered and evaluated are:

- **language processing**: What are the categories and the syntactic and semantic features necessary to induce the correction patterns (e.g. adverb categorization for fuzzy lexical items, verb semantic classes, e.g. (Croce et ali. 2012)) ? What are the limits of those categorizations to induce correction patterns? How much are they domain independent and how much needs to be tuned when an application is deployed in an industrial environment?
- **knowledge representation**: what are the resources which are needed and how accurate they have to be ? For example, a number of antonyms and synonyms must be encoded in the domain terminology to deal with negation. Antonyms may be ambiguous, context dependent and difficult to encode; they need some form of typing.
- **stability of correction patterns**: it is of much interest to analyze, on the long term, how correction patterns and recommendations may evolve, in particular, when larger sets of data are available or when new visions of document authoring are discussed among the technical writers.
- **adequacy and coherence of correction patterns**: when a large number of correction patterns are induced, in particular low-level patterns, it is crucial to identify overlaps and contradictions between them. A kind of pattern editor could be defined that would check coherence and overlaps, possibly taking into account preferences among patterns.
- **pattern selection**: our system suggests several patterns of various levels of abstraction. It is then the role of technical writers to decide which ones are the most appropriate and should be kept. This operation can be done either via discussion and mediation involving all the technical writers or via an administrator who is alone responsible of the decisions. In both cases it is of much interest to evaluate the difficulty of the task and if additional tools are necessary to help making the adequate decisions.
- **usability**: once the above points are realized and evaluated, it is also useful to evaluate how much of these patterns are effectively used, how, how much time is saved and what is the gain in reliability. It is clear that our system should reflect as much as possible the technical writer activity, so that its use is as straightforward as possible.
- **writer interface**: an evaluation of how the user interface adapts itself to the technical writer and the task is a major issue. For example, some authors prefer to display all the errors simultaneously whereas others prefer to proceed by type of error. Some authors also prefer to proceed paragraph or section by section whereas others prefer to run the system on a much larger

text portion. These considerations obviously depend on the kind of document, e.g. a large procedure of 200 pages versus a list of requirements can be processed differently.

- **new features of technical authoring**: our error correction memory involves a certain evolution of the technical writer activity. We feel it is important to characterize this evolution in terms of e.g.: new ways of working on documents, new types of tasks, new responsibilities, new forms of document validation, new forms of life-cycles for documents. Similarly, since our system is a kind of memory of the authoring traditions of a company, it can also be evaluated w.r.t. its completeness and its role as a tutor for novices.

Except for the two first items, which are slightly more standard in evaluation technology, the other items involve major elaborations and require the development of dedicated protocols, once the system is operational in a company. This is obviously necessary to make sure the system is used on a large scale, but also to collect remarks in order to improve it and extend it to other areas of technical writing.

7. Perspectives

In this paper, we have explored the notion of error correction memory, which, paired with the LELIE system that detects specific errors of technical writing, allows a flexible and context sensitive detection and correction of errors. Correction scenarios are based on an architecture that develops an error correction memory based on (1) generic correction patterns and (2) contextual correction recommendations for elements in those patterns which are more contextual. Both levels are acquired from the observation of already realized corrections and correct texts.

This approach is quite new, it needs an in-depth evaluation in terms of linguistic adequacy and usability for technical writers. It is however still in an early research stage: evaluation is designed to develop improvement directions rather than to give definitive performances of the approach.

Besides negation and fuzzy lexical items, we are exploring additional facets of an error correction memory for the other major types of errors such as passives, future forms, deverbals, N+ N constructions (Garnier 2011), misplaced discourse structures, and complex sentences. Integration into a real company information system is ongoing and would permit a more suitable evaluation of the service and of its evolution.

References

Alred, G. J., Charles T. B., & Walter, E. O. (2012). *Handbook of Technical Writing*. St Martin's Press, New York.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., & Gnaga, R. (2013). Automatic Checking of Conformance to Requirement Boilerplates via Text Chunking: An Industrial Case Study, 7th ACM/ IEEE International Symposium on Empirical Software Engineering and Measurement

(ESEM 2013). Baltimore, MD, USA.

Baader, F., & Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press.

Barcellini, F., Albert, C., & Saint-Dizier, P. (2012). Risk Analysis and P revention: LELIE, a Tool dedicated to P rocedure and Requirement Authoring, LREC, Istanbul.

Boulle, L., & Mesic, M. (2005). Mediation: Principles Processes Practice, Australia. Butterworths.

Buchholz, S. (2002). Memory-based grammatical relation finding, PhD, Tilburg.

Croce, D., Moschitti, A., Basili, R., & Palmer, M. (2012). Verb Classification using Distributional Similarity in Syntactic and Semantic Structures, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, ACL-2012, Jeju Island, Korea.

Cruse, A. (1986). *lexical semantics.* Cambridge university Press.

Daelemans, W., & van Der Bosch, A. (2005). Memory-Based Language Processing, Cambridge.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, Reasoning Web, Fourth International Summer School 2008, Lecture Notes in Computer Science 5224(pp. 104-124). Springer.

Fellbaum, C. (1998). *WordNet An Electronic Lexical Database*. The MIT Press.

Fuchs, N. E. (2012). First-Order Reasoning for Attempto Controlled English, In Proceedings of the Second International Workshop on Controlled Natural Language (CNL 2010), Springer.

Ganier, F., & Barcenilla, J. (2007). Considering users and the way they use procedural texts: some prerequisites for the design of appropriate documents. In D. Alamargot, P. Terrier and J. -M. Cellier (Eds), Improving the production and understanding of written documents in the work-place, Elsevier Publishers.

Garnier, M. (2011). Correcting errors in N+N structures in the production of French users of English, EuroCall, Nottingham.

Grady, J. O. (2006). *System Requirements Analysis*. Academic Press, USA.

Horn, L. (2001). A natural history of negation, D. Hume series, University of Chicago Press.

Hull, E., Jackson, K., & Dick, J. (2011). Requirements Engineering, Springer.

Kuhn, T. (2013). A Principled Approach to Grammars for Controlled Natural Languages and Predictive Editors. *Journal of Logic, Language and Information*, 22(1).

Kuhn, T. (2014). A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1).

Llyod, J. W. (2013). *Foundations of Logic Programming, Springer Verlag*, 3rd edition.

O'Brien, S. (2003). Controlling Controlled English. An Analysis of Several Controlled Language Rule Sets. Dublin City University report.

Pfenning, F. (1992). Types in Logic Programming, MIT Press.

Saint-Dizier, P. (2012). Processing Natural Language Arguments with the <TextCoop> Platform. *Journal of Argumentation and Computation*, 3(2).

Saint-Dizier, P. (2014). *Challenges of Discourse Processing: the case of technical documents*. Cambridge Scholars, UK.

Schriver, K. A. (1989). Evaluating text quality: The continuum from text-focused to reader-focused methods. *IEEE Transactions on Professional Communication*, 32, 238-255.

Unwalla, M. (2004). AECMA Simplified English. Available at: <http://www.techscribe.co.uk/ ta/ aecma-simplified-english.pdf> Retrieved 2015.06.25.
Van der Linden K. (1993). Speaking of Actions: choosing Rhetorical Status and Grammatical Form in Instructional Text Generation. PhD, Univ. of Colorado, USA.
Weiss E. H. (2000). Writing remedies. Practical exercises for technical writing. Oryx Press.
White, C., & Schwitter, R. (2009). An Update on P ENG Light. In: Pizzato, L., Schwitter, R. (eds.) Proceedings of ALTA 2009, Sydney, Australia, 80-88.
Wyner, A., et ali. (2010). On Controlled Natural Languages: Properties and Prospects.

• About the authors:

Patrick SAINT-DIZIER
E-mail: stdizier@irit.fr

Patrick SAINT-DIZIER, PhD, is a senior researcher in Computational Linguistics and Artificial Intelligence at CNRS, IRIT, Toulouse, France. He is specialized in discourse and semantic analysis. He has developed several national and European projects dedicated to logic programming, argumentation and technical text analysis. He is the author of more than 80 conference and journal articles and of 11 books. Besides foundational research, he has a long practice and experience of research and development activities.

His research merges foundational aspects with empirical studies and prototype development.A few generic ideas guide my team's work:

(1) elaborating a formalism and tools for discourse analysis: TextCoop based on logic, that includes reasoning capabilities (Dislog and the TextCoop platform). Free software available, see link below
(2) Linguistics and conceptual semantics for language processing: linguistics of argumentation, linguistics of operational texts (procedures, requirements), linguistics of opinion expression, lexical semantics of predicative forms.
(3) Text Clinic: LELIE, improving the quality and contents of technical texts: application to business error correction and style enhancement in procedures and in requirements. Application to risks analysis and prevention in industrial documents: The LELIE project. Free software available, see link below.
(4) Argumentation mining in various contexts and domains.
(5) Argumentation, Discourse and Rhetoric in Non-verbal forms of communication: application to western tonal music analysis and modelling.