From Detection/Correction to Computer Aided Writing

Damien GENTHIAL, Jacques COURTIN
Laboratoire de génie informatique - Imag Campus - BP53X
F-38041 GRENOBLE CEDEX - France
Phone: (33) 76 51 48 78
FAX: (33) 76 44 66 75
E-Mail: genthial@imag.fr
courtin@imag.fr

RÉSUMÉ

La plupart des textes actuels sont produits sous forme électronique à l'aide de systèmes informatiques qui fournissent des facilités de manipulation de chaînes mais aussi des outils linguistiques : correcteur d'orthographe, dictionnaire voire vérificateur grammatical. Nous pensons qu'un système d'aide à la rédaction doit être conçu comme un environnement complet pour la production, la maintenance, l'édition et la communication des textes. Ceci suppose par exemple l'utilisation d'un gestionnaire d'idées et de dictionnaires pour la production, d'un éditeur de textes et de vérificateurs linguistiques pour la maintenance, d'un traitement de textes pour l'édition et d'une forme normalisée pour la communication.

A la suite de nos travaux sur la détection et la correction des erreurs, nous proposons une architecture logicielle capable d'intégrer de manière uniforme nos outils linguistiques (analyse et génération morphologique, techniques de correction lexicale, analyse et vérification syntaxique) ainsi que des outils de traitement de texte, d'édition et d'exportation de documents. Ces outils sont conçus comme des modules spécialisés disposés autour d'une structure de données unique qui constitue la représentation interne du texte. Cette structure est un treillis multi-dimensionnel qui traduit la linéarité mais aussi la structure et les ambiguïtés du texte. Elle est complétée par un lexique basé sur des structures de traits typées qui contiennent les informations morphologiques, syntaxiques et sémantiques associées aux mots.

La distribution de la compétence globale du système dans des modules spécialisés facilite sa maintenance et, surtout, permet le partage des compétences locales entre les modules, ce qui est très important pour les modules linguistiques (le vérificateur syntaxique, par exemple, requiert presque tous les autres modules linguistiques : morphologie, phonétique, syntaxe).

ABSTRACT

Most texts nowadays are produced in an electronic form by the use of systems which provide text processing facilities but also linguistic facilities such as spelling checkers, on-line lexicons and even syntactic checkers. We think that a computer-aided writing system must be designed as a complete environment for the production, maintenance, edition and communication of texts. This implies for example the use of an ideas manager and on-line lexicons for production, a text editor and linguistic verifiers for maintenance, a text processor for edition and a standardized form for communication.

Following our work on detection and correction of errors, we propose an architecture of a system able to integrate in a uniform way our linguistic tools (morphological parsing and generation, lexical correction techniques, syntactic parser and verifier) as well as tools for text processing and document editing and exporting. Tools are designed as specialized modules disposed around a unique data structure, which is the internal representation of the text. This structure is a multi-dimensional lattice, coding the linearity but also the structure and the ambiguities of the text. It is completed by a lexicon based on typed feature structures encoding morphological, syntactic and semantic information on words.

The distribution of the competence of the system in specialized modules permits an easier maintenance of the system itself but, moreover, allows competence sharing among the modules, which is very important for the linguistic ones (for example the syntactic verifier needs to use almost every linguistic module: morphology, phonetic, syntax).

1. Introduction

In their life-cycle from creation to publishing, all texts nowadays take an electronic form. Most of them are directly produced in this form and take the paper form only for publishing. Thus a lot of services can be provided to the writer who uses a computer to produce his texts. This idea is not new but, following our work on detection and correction of errors, we think it must be investigated more deeply than it has been.

We first introduce what we mean by computer aided writing. We then propose an architecture for a computer aided writing environment and quickly describe its modules. We outline one of its main characteristics (limited data structures), and finally justify the second one (distribution of services) in the light of our work on detection and correction of errors.

2. Computer Aided Writing (CAW)

A computer system for a writer is basically a personal computer which runs a text processor; the power increase of personal computers has been followed by the growth of services provided to the user. Some of these services aim to increase the writers productivity but most of them aim to obtaining a better quality of produced documents. We will distinguish here between two categories of services: presentation services and production services. The first ones concern the way the paper form of the text looks: justification, formating, multi-column... They are very powerful in modern systems, especially if you add to your text processor a graphic processor and a page maker, but they have little to do with linguistics and so we will not discuss them here.

The second ones concern the text itself, in its content and in its form. The best known and most achieved service in this category is the spelling checker, which can be found in every modern text processor. Recently, other services have emerged:

- on-line lexicons with synonym and antonym links;
- idea managers which help the user to build the plan of his document;
- syntactic checkers in the spirit of the IBM system CRITIQUE [6].

In most cases, these new services are add-ons to an existing text processor and CAW systems are stacks of tools, lacking the coherence of an integrated approach.

Our idea is that CAW must be thought of as a goal in itself and our aim is to build an environment for the production, maintenance, edition and communication of texts. Such a system will be based on a coherent set of software tools reflecting the state of the art in string manipulation and linguistic treatment. At a first glance, the system should include classic and well-known tools such as those cited above and more sophisticated tools like:

- morphological analysis and generation, which can for example be used for lemmatization of words or groups of words. The idea here is to use these lemmatized groups as keys to access external knowledge bases or document bases [9].
- syntactico-semantic analysis and generation to allow operations like: changing the tense of a paragraph, changing the modality of a sentence, help in detecting ambiguous phrases and in disambiguation by proposing paraphrases. There is also the possibility of generating a definition of a word on the basis of its formal description in the lexicon.

- lexical and syntactic checkers, which must also be able to propose corrections, by the use of all the linguistic knowledge included in the system.
- structural manipulations of the text in the spirit of idea managers but also some verifications on the structure by the use of a grammar of the text, which depends on the type of document created. For example, a software documentation will include a user manual and a reference manual, the user manual will include an installation chapter, a tutorial introduction chapter,....
- interface with the outside world: that includes of course the production of a paper form of the text but also, at least as important as the former, the production of the text in some standardized form (for example the form recommended by the TEI [8]) which can travel on networks and be legible by most software. This form can also be used to store the text in databases or to pass it on to other software. A very interesting type of software could be an automatic translator, so that a text could be created in one language and published in one or more other languages.

Such a system is a long term objective and we will see in the next section an architecture which makes possible a short term full implementation, while being open for future extensions.

3. Architecture of a CAW environment

Figure 1 describes the architecture of the CAW system under development in our team. Its caracteristics are the use of a minimal number of data structures and a distributed architecture. We will here quickly describe the role of each module, leaving for the next two sections the discussion about data structures and architectural choices.

The proposed system is primarily built for French but every module has been designed to be as general as possible, and is completely configurable, so that it can be used for other languages.

Each module is viewed as a server which is able to provide some service. Following our work on detection and correction of errors, many modules are dedicated to this sort of task.
Given an incorrect word, the *similarity key* module is able to produce a list of correct words which are possible corrections of the incorrect one. It is well-suited for typographic errors.
The *phonetic graphic transducer* plays the same role by using the phonetic invariant of words. It is well-suited for spelling errors.
The *morphological module* can also be used for lexical correction [3] but its main purpose is to produce an input for the *syntactico-semantic parser*, which is in charge of building a decorated structure of the sentences of the text. The parser we use is a dependency-tree transducer designed as a robust parser [4, 5].
The *syntactic checker* is in charge of verifying agreement rules in sentences [7].
The *multi-purpose lexicon* contains all lexical information and furnishes access tools (see next section).

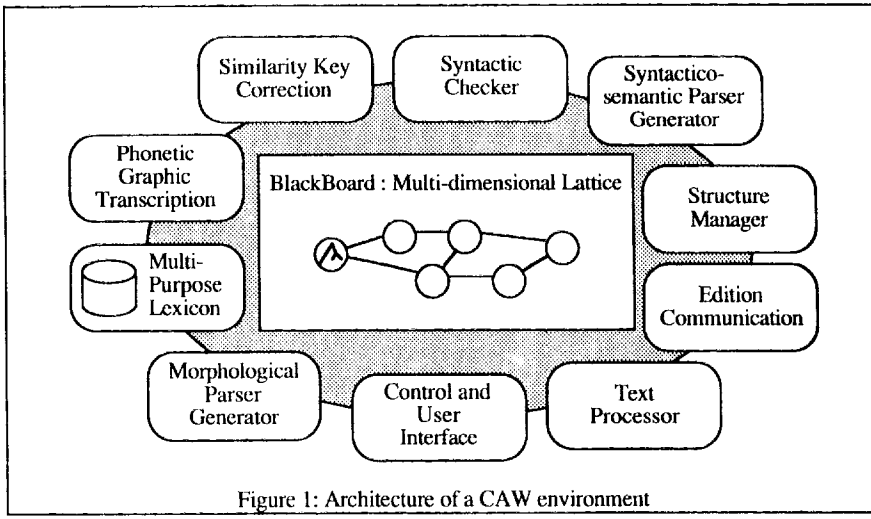


Figure 1: Architecture of a CAW environment

The *text processor* provides string manipulations while the *edition communication module* gives a paper or communicable form of the text.
The *structure manager* is in charge of global manipulations on the surface structure of the text (chapter, sections,...) and of the much more difficult task of verifying the internal coherence (there is an introduction, a development, a conclusion,...).
Finally, *the control and user interface* module assumes the synchronisation and communication between modules and the transmission of user orders.

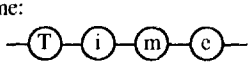
The correctors, the syntactic checker, the morphological parser and generator, the syntactico-semantic parser are all operational on micro-computers. At the moment, the lexicon is a roots and endings dictionary (35,000 entries, generating 250,000 forms) with only morphological information on words, but its extension is under development.

4. Data Structures

4.1. Blackboard

A main caracteristic of our system is the use of an internal representation of the text in the form of a multi-dimensional lattice (inspired by [2]) which play the role of a blackboard for all the modules.

Each node of the lattice bears information on a piece of text, and we propose that they all have the same structure: each node bears a tree (sometimes limited to the root) and each node of the tree bears a typed feature structure (a Ψ-term, see §4.2). We can imagine that the lattice is initiated by the flow of characters which come from the text processor, thus the word "Time" will become:



For performance problems, it seems more reasonable to initiate the lattice with the lexical units resulting from the morphological parsing of the text. With the sequence of characters "Time flies...", we will obtain the bottom four nodes of the figure 2 lattice.

We can see two dimensions of the lattice on this example: a sequential dimension ("time" is the first word and is followed by the second word "flies"), and an ambiguity dimension (both words have two possible interpretations).

A third dimension appears when the syntactic parser starts its work. It produces new lattice nodes which bear dependency trees. With the lattice above, the syntactic parser will add the two top nodes (figure 2).

Every module can read or write in this lattice; for example, the corrections produced by lexical correctors can be added as multiple interpretations of a word.

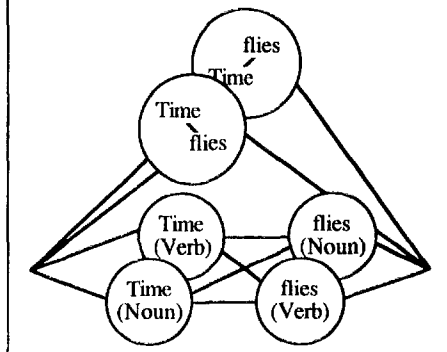


Figure 2: Example of a lattice

4.2. Lexicon

We think it is very important, for the coherence of the knowledge embedded in the system, that all lexical information be contained in a unique dictionary. Multiple access and adapted software tools will extract and present the information to the user in different forms, for example the natural language form of a formal entry may be computed by the syntactic generator.

To represent knowledge associated with words, we have chosen typed-feature structures called Ψ-terms [1]. With these structures, basic concepts are ordered in a hierarchy which can be extended to whole structures. Thus we can determine if a Ψ-term is less than another and the unification of two Ψ-terms is the biggest Ψ-term which is less than both unified ones. In other words, the unification of two terms is the most general term which synthesizes the properties of both unified ones. This caracteristic is very interesting for the implementation of paradigms: a paradigm is the representative of a class of words and contains the information which describes the behaviour of a word. We distinguish three types of paradigms: morphological, syntactic and semantic.

Morphological paradigms bear the category of the word and a few linguistic variables such as gender and number. *Syntactic paradigms* contain information about the function of the word within its context. The aim is to code sub-categorization of words, and it is very important for verbs but also for nouns and some

adjectives. A *semantic paradigm* is the semantic concept associated with the word or the logical structure in the case of predicate words.

Examples of paradigms:

LU stand for Lexical Unit, NP for Nominal Phrase and P for Proposition.

baby: morphological

```
LU(cat => cnoun;
   gender => (masuculin ; feminine);
   number => singular)
```

baby: syntactic

```
LU(syn => NP)
```

baby: semantic

```
LU(sem => HUMAN)
```

choose: morphological

```
LU(cat => verb)
```

choose: syntactic

```
LU(syn => P(subject =>
              NP(sem => ANIMATE);
            object =>
              NP(sem => OBJECT))
```

choose: semantic

```
LU(sem => CHOOSE(agent => ANIMATE;
                 choice => OBJECT))
```

For a verb like *rain*, we can be more precise in the syntactic paradigm:

rain: syntactic

```
LU(syn => P(subject =>
              NP(cat => pers_pronoun;
                 person => 3;
                 number => singular;
                 lex => "it"))
```

Each entry in the lexicon contains a key, which is used to access the entry, and a reference to a paradigm of each type. In order to allow information sharing between Ψ-terms, we add to the entry an optional list of equational constraints. For example, for *choose*, we have : `syn.subject.sem = sem.agent` and `syn.object.sem = sem.choice` saying that usually the subject of the verb is its agent and the object is the choice. The result of morphological parsing of a form is the unification of the three paradigms of each lexicon entry used. For example, for the form *chooses*, we use the root *choose* and the ending *s* (which add the features *person* and *number* to the paradigms of the verb) thus we obtain:

```
LU(cat => verb;
   person => 3;
   number => singular;
   syn => P(subject =>
              NP(sem => @S:ANIMATE);
            object =>
              NP(sem => @O:OBJECT);
   sem => CHOOSE(agent => @S;
                 choice => @O))
```

where the notation @X is used to write reference links (equational constraints).

The idea behind paradigms is to allow a great factorization of knowledge: it is obvious for morphological paradigms (in the actual dictionary, we have only 400 paradigms for 250,000 forms) and for syntactic paradigms (the number of possible sub-categorizations for verbs is far less than the number of verbs). It is less obvious for semantic paradigms, especially if you want a very fine description of a word: in this case, there is almost a paradigm for each word.

So the lexicon is essentially built around three Ψ-term bases, one for each set of paradigms. The bases are accessed by the roots and endings dictionary used by morphological tools (parser and generator), and we can easily add synonym and antonym links to this dictionary. The key-form correspondence table, required by the similarity key correction technique cannot easily be embedded in this lexicon structure, but we propose to append it to the lexicon so that any module requiring lexical information must use the multi-purpose lexicon module. This constraint is imposed in view of coherence: each time a root is added to the main dictionary, all key-form pairs obtainable from this root must be added to the table.

5. Distribution

Each module in our system must be viewed as a server which responds to requests sent by any other module. Such an architecture has the classical advantages of modular structures: you can add or remove a module very easily, you can modify a module in a transparent manner as long as you do not change its interface, ...

But this structure has another advantage which is very important in the context of linguistic treatments: the linguistic competence of each module can be exploited by the others. We will use two examples to illustrate our purpose.

First, in detection and correction of lexical errors, we have implemented classical tools (similarity key and phonetic). Then we decided to implement syntactic checking, so we needed the services of a morphological parser. We added to the system (a prototype called DECOR) our morphological tools, and the availability of these tools gave the idea of using them for detection and correction, so we implemented a third technique of correction : morphological generation.

Example of correction using morphological generation :

foots, although incorrect, may be parsed as *foot* + *s*, and the root *foot*, plus the variables (*plural*) associated with the *s*, when passed on to the morphological generator, give the correct form *feet*.

As a second example, consider the problem of proposing correction for agreement errors: when an error occurs, it means that at least **two** words do not agree so there are at least two possible corrections depending on which of the two words you choose to correct. The solution for the system is to propose both corrections to the user and let him choose one. Even this simple method requires linguistic service: a morphological generator is necessary to produce each correction.

But we think that in most cases the good correction can be choosen automatically, according to criterions[1] such as those considered by [10]:

- number of errors in a group: *little cat are funny pets* must be corrected *little cats are funny pets* rather than *little cat is funny pet*;
- it is better to correct in a way which does not modify the phonetic of the phrase. We give here a French example[2]: *Les chiens dressées...* will be corrected *Les chiens dressés...* rather than *Les chiennes dressées...*
- one can give priority to the head of the phrase: *cat which are...* becomes *cat which is...*;
- writer laziness: a writer sometimes omit an *s* where one is necessary, but rarely add one where it is not.

Such criterions are sometimes contradictory and we propose to use an evaluation method which gives a relative weight to each criterion so that each possible correction has a probability of being correct. The user is asked for a choice only in cases where both corrections have equivalent probability.

But, whatever strategy is implemented, it needs the cooperation of various linguistic modules in order to perform the evaluation: phonetic transducer, morphological parser and generator, and our architecture permits the use of the available ones.

Finally, beyond linguistic justifications, one can find computational justifications: each module of the system can work in parallel with the others and they can even work on different computers, putting the distribution at a physical level.

[1]Note that these criterions are pertinent for French, where there are a lot of agreement rules (between noun, adjectives and determiner, between subject and verb,...)

[2]An similar english example might be *The skis slides* wich is corrected *The ski slides* rather than *The skis slide*.

6. Conclusion

As sophisticated linguistic treatments are expensive in time and space, we think it is very important, that a CAW system should integrate all treatments and knowledge in a uniform way. It makes it easier to take advantage of the whole knowledge in each service involved in order to provide very powerful services. This power of the services is a mean to compensate, for a potential user, the lack of ergonomy due to poor performance: a system which can build the multi-dimensional lattice in real-time does not seem a realistic goal for the near future.

As a typical application for our system, we think of the production of the technical documentation of an industrial product: as there are for example software engineering environments, we propose linguistic engineering ones. In such a context it is possible to add structure services, more powerful services at the semantic level and interface with other software such as an automatic translator.

References

[1] Aït Kaci (H.), *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures*. Ph.D. Thesis - Computer and Information Science, Univ. of Pennsylvania, Philadelphia, USA, 1984

[2] Boitet (C.), *Representation and computation of units of translation for Machine Interpretation of spoken texts*. GETA & ATR Tech. Report TR-I-0035, August 88

[3] Cohard (B.), *Logiciel de détection et de correction des erreurs lexicales*. Thèse CNAM, Grenoble, Mars 88

[4] Genthial (D.), Courtin (J.), Kowarski (I.), *Contribution of a Category Hierarchy to the Robustness of Syntactic Parsing*. 13th CoLing, Helsinki, Finland, August 1990, Vol. 2, pp 139-144

[5] Genthial (D.), *Contribution à la construction d'un système robuste d'analyse du français*. Thèse de l'université Joseph Fourier, Grenoble I, Janvier 1991

[6] Richardson (S.D.), *Enhanced Text Critiquing using a Natural Language Parser: the CRITIQUE System*. IBM Research Report RC 11332, Yorktown Heights, USA, 1985

[7] Strube de Lima (V.L.), *Contribution à l'étude du traitement des erreurs au niveau lexico-syntaxique dans un texte écrit en français*. Thèse de l'Université Joseph Fourier, Grenoble I, Mars 1990

[8] TEI (Text Encoding Initiative), *Guidelines for the Encoding and Interchange of Machine Readable Texts*. Computer Center MC135, University of Illinois at Chicago and Oxford University Computing Services.

[9] Tomasino (I.), *ODILE : Un Outil d'Intégration Extensible de Dictionnaires et de Lemmatiseurs*. Thèse CNAM, Grenoble, Déc. 90.

[10] Véronis (J.), *Morphosyntactic correction in natural language interfaces*. 12th CoLing, Budapest, Hungary, August 1988, pp 708-713