A text model and a formalism for expressing text-level linguistic knowledge

Couto, Javier jcouto@fing.edu.uy
Crispino, Gustavo crispino@fing.edu.uy
Grassi, Mariela mgrassi@seciu.edu.uy
Skorodynski, Mónica mskorodynski@bps.gub.uy
Gustavo Crispino
Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Julio Herrera y Reissig 565, Montevideo, Uruguay
CP 11300
Fax: (598) 2 711 0469
Telephone: (598) 2 711 4244 Ext. 110

Abstract

At the moment, most applications and products in Natural Language Processing (NLP) have been considering the sentence as the unit of processing. In order to reflect in NLP applications the progress in researches that go beyond the sentence, it is necessary to be provided with text models and formalisms. In this paper we introduce a text model and a formalism independent of any implementation destined to express linguistic knowledge that leans on the structure of a text. We have developed an implementation of the formalism in Java. This implementation was used to build the Semantic Labeling Module in the Contexto Platform. This platform supports different applications of the Contextual Exploration Method.

1. Introduction

At the moment, most applications and products in Natural Language Processing (NLP) have been considering the sentence as the unit of processing. In a more general way, traditional linguistic was limited to the study of the sentences. This fact explains why various phenomena remained unstudied until recently. Several phenomena justify the extension of traditional linguistics far beyond the sentence.

The language uses a series of terms, which have the function of connecting propositions. To study them, we must go beyond the framework of the proposition. The anaphors, the cataphors and the ellipsis oblige us to look backward or forward to find the missing element.

Beyond the limits of traditional linguistic studies, Adam in [Adam 90] makes several observations that show the interest of a textual linguistic. And some of this textual relations has been also studied by M.K Halliday and R. Hasan [Halliday et al 76].

For M. Charolles [Charolles 88] a text is a sequence of propositions, but there are some linguistic elements, that he calls "textual organizers", which play the role of indicating the relations between these different propositions.

In order to reflect in NLP applications the progress in textual linguistic researches, it is necessary to be provided with text models and formalisms.

There are works that take in consideration text structure like [Chali et al 96] and [DOM 98] but with different goals than us. In [Chali et al 96] the authors propose a text model that is constrained by and needed to study the impact of text structure on comprehension and memorization processes.
The Document Object Model [DOM 98] is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The goal of the DOM specification is to define a programmatic interface for XML and HTML.

In this paper we introduce a text model and a formalism independent of any implementation destined to express linguistic knowledge that leans on the structure of a text.
We have developed an implementation of the formalism in Java. This implementation was used to build the semantic labeling module in the Contexto platform. This platform supports different applications of the Contextual Exploration method [Desclés et al 97].

This article is organized as follows. In Section 2 we introduce the text model. In Section 3 we describe the formalism. The application of our

implementation to the Contextual Exploration method is presented in Section 4. We report our research conclusions in Section 5.

2. Text Model

To define a text model, we take into account its different components and the way they are related to each other.

2.1 Definition of Text

In our first approach to define texts, we make certain simplifications. Some text elements are not considered (i.e. chapters, lists of elements, footnotes, tables and figures). The elements considered in our analysis are sections, paragraphs, sentences and words[1].

We see a text as a hierarchic organization. Therefore, a text is composed of a sequence of paragraphs and sections. A section is composed of a sequence of paragraphs and subsections. A paragraph is composed of a sequence of sentences and a sentence is composed of a sequence of words. Due to the existence of subsections, we can define nesting levels between sections, where a section differs from another one of different level, from a conceptual point of view, only in the level and nothing else.

We want to remark the notion of order that exists between elements of the same hierarchic level. We must also consider the dynamic nature of text. The definition that we present, given the restrictions of simplicity, is well applied to certain texts (i.e. papers, press articles) but may be insufficient for other types of text.

2.2 Representation of a Hierarchy

In order to represent a hierarchic structure we were based upon the notion of n-arity dynamic tree.

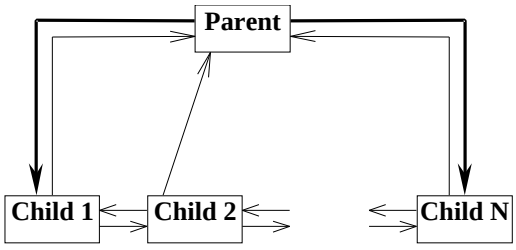

Figure 2.1 n-arity tree diagram

In this representation:

i. A node knows his first and last child.

ii. All the nodes know his parent.

iii. All the nodes know their next and previous sibling.

Based on this representation, we can move from an element of the tree to another one using only the information of its hierarchic relationship.

2.3 The Textual Hierarchy

Our vision of the text as a hierarchic structure and its representation allow us to browse it applying the knowledge of the hierarchic relationship between the different elements. As an example, given a paragraph, we are able to know its sentences (they are its children). In addition, we are able to know the words that compose it (they are the union of the children of all its children). We can know the following paragraph or the previous one within the same hierarchic level, or the paragraphs that are to a distance D before or after it (they are its siblings). Following the example, we are able to know the section that contains it directly (its parent), as well as the section of level 1 that contains it (its parent or ancestor).

3. Formalism to Represent Texts

So far we have presented a text definition. We have seen how the text is organized and we have represented it in a certain way. In this section, we will define a notation and a set of operators and operations associated that will serve to refer to text elements more precisely.

3.1 Notation

Our effort is oriented to find an unambiguous notation able to offer as much information as possible. Our proposition is: in the right place we have the text element (Sentence, Paragraph, Section, etc.) from which we start. Then, in the very left place we have the operator that references the text element at which we arrive. And from right to left, we have the path that goes from the first element to the last one, according to the hierarchy of the text. This path is formed by the successive application of different operators.

[1] We understand word in a wide sense, so it includes complex words, abbreviations, etc.

Op. ← Op. ← · · · · · ← Op. ← Text Element

We want to remark three things about the notation:

i. When we deal with an expression in this notation, we must remember that it refers to a particular text element, although the expression is a path. The path solves the reference, so we can know with which text element we deal.

ii. In order to obtain a correct reference, the path must be valid if we follow the hierarchy of the text.

iii. Finally, to refer a valid text element, the element of the right, the initial one, must have a correct reference, since all the others are relative to it.

3.2 Operators

Before presenting the operators, we state some definitions that stand from the tree concept and from the text definition previously presented.

Text element: a text element can be the text itself, a section, a paragraph, a sentence or a word.
Element Type: the definition of type of element is very intuitive: in the case of a text we will say that its type is "TEXT", in the case of a section that it is "SECTION", and so on.
Sibling: two text elements are siblings if they are of the same type and they have the same parent.
Relative Position: we define the relative position of a text element as the position that it has in the sequence of its siblings.
Absolute Position: we define the absolute position of a text element as the position that it has in the sequence formed by all the text elements of the same type.
Level: we define the level of a section like the nesting level that it has, being 1 the topmost level.

The operators we define for the hierarchy, classified according to the type of text element that they return, are:

Text:

ParentText → Text. It takes a section or a paragraph, and returns a reference to the text that contains it. The section must be of maximum level and the paragraph a direct child of the text.

Section:

ParentSec (*n*: integer Default = 1) → Section. It takes a section or a paragraph and returns a reference to the ancestor that is placed n levels up in the hierarchy.

ParentSecLevel (lvl: integer) → Section. It takes a section or a paragraph and returns a reference to the parent or ancestor section that has level *lvl*.

PrvSec (*n*: integer Default = 1) → Section. It takes a section and returns a reference to the sibling section that is n positions before.

NextSec (*n*: integer Default = 1) → Section. It takes a section and returns a reference to the sibling section that is n positions after.

FstSubSec → Section. It takes a section or text and returns a reference to the first subsection or the first section of topmost level, respectively.

LastSubSec → Section. It takes a section or text and returns a reference to the last subsection or the last section of topmost level, respectively.

SubSecNumber (*n*: integer) → Section. It takes a section or text and returns a reference to the subsection or section of topmost level that has relative position *n*, respectively.

The definitions of the following operators are analogous to the previous ones.

Paragraph:

ParentPar → Paragraph
PrvPar (***n***: integer Default = 1) → Paragraph
NextPar (***n***: integer Default = 1) → Paragraph
FstPar → Paragraph
LastPar → Paragraph
ParNumber (***n***: integer) → Paragraph

Sentence:

ParentSent → Sentence
PrvSent (***n***: integer Default = 1) → Sentence
NextSent (***n***: integer Default = 1) → Sentence
FstSent → Sentence
LastSent → Sentence
SentNumber (***n***: integer) → Sentence

Word:

PrvWord (***n***: integer Default = 1) → Word
NextWord (***n***: integer Default = 1) → Word

FstWord → Word
LastWord → Word
WordNumber (***n***: integer) → Word
WordValue → string

Examples:

The previous sentence to the previous one of the word *W*:

PrvSent (2) ← *ParentSent* ←*W*

The section of level 1 to which the sentence *S* belongs:

ParentSecLevel (1) ←*ParentPar* ←*S*

The section to which the word *W* belongs:

ParentSec ←*ParentPar* ←*ParentSent* ←*W*

The next sentence to the first sentence in paragraph *P*:

NextSent ← *FstSent* ← *P*

The last paragraph in the section of level 2 that is the parent of the word *W*:

LastPar ←*ParentSecLevel (2)* ←*ParentPar* ←*ParentSent* ←*W*

3.3 Operations

While the operators allow us to move within the hierarchy, the operations allow us to know certain properties of its elements.

The notation for the operations is similar to the one of the functions of a programming language Pascal-like. They can have a set of parameters, each one of them with an associated type and they can return a value of certain type.

The operations that we define on elements of the hierarchy are the following ones, classified according to the data type they return:

Boolean

IsFstRel (*E*: TextElement) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the relative position of the element is the first one, false in other case.

IsFstAbs (*E*: TextElement) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the absolute position of the element is the first one, false in other case.

IsLastRel (*E*: TextElement) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the relative position of the element is the last one, false in other case.

IsLastAbs (*E*: TextElement) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the absolute position of the element is the last one, false in other case.

IsRelNumber (*E*: TextElement, *n*: integer) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the relative position of the element is *n*, false in other case.

IsAbsNumber (*E*: TextElement, *n*: integer) → Boolean. It is applied to elements of type: section, paragraph, sentence and word. It returns true if the absolute position of the element is n, false in other case.

HasLevel (*S*: Section, *n*: integer) → Boolean. It is applied to elements of type section. It returns true if the nesting level of the section is n, false in other case.

Integer

RelPos (*E*: TextElement) → Integer. It is applied to elements of type: section, paragraph, sentence and word. It returns the relative position that the element has in the text.

AbsPos (*E*: TextElement) → Integer. It is applied to elements of type: section, paragraph, sentence and word. It returns the absolute position that the element has in the text.

Amount (*T*: ElementType, *E*: TextElement, Depth: Boolean = False) → Integer. It is applied to elements of type: section, paragraph and sentence. It returns the amount of elements of type *T* that has the text element *E*. The types of element can be "SECTION", "PARAGRAPH", "SENTENCE" and "WORD".

It is possible to obtain a relatively broad assembly of combinations, depending on the type of the requested element and its relation with the type of the element in where it will be looked for. As an example, with this operation we can know, of a section (type of element where the search will become), the amount of

subsections, paragraphs, sentences and words that it contains.

The *Depth* parameter is useful when we want to count elements across several hierarchic levels. For example, suppose that we want to know the amount of words that a paragraph has. In this case, if we did not set the *Depth* parameter as true, then the result will be 0 because a word does not belong directly to a paragraph. But if we set the *Depth* parameter as true, this operation will return the amount of words that the paragraph has (the sumatoria of words that has each sentence in the paragraph).

Level (*S*: Section) → Integer. It is applied to elements of type section. It returns the nesting level that it has.

Note: *we assume relativity by default. So the operations IsFstRel, IsLastRel, IsRelNumber and RelPos can be referenced with their alias IsFst, IsLast, IsNumber and Pos, respectively.*

Examples:

Check if the paragraph to which the word *W* belongs is the first relative:

IsFst (ParentPar ←ParentSent ←W)

Check if the section of level 1 to which the word *W* belongs is the last absolute:

IsLastAbs (ParentSecLevel (1) ←ParentPar ←ParentSent ←W)

Check if the sentence parent of the word *W* is the third relative:

IsNumber (ParentSent ←W, 3)

Check if the section to which the sentence *S* belongs has level 2:

HasLevel (ParentSec ←ParentPar ←S)

Return the relative position of the sentence parent of the word *W*:

Pos (ParentSent ←W)

Return the amount of sentences that the paragraph to which the word *W* belongs contains:

Amount ("SENTENCE", ParentPar ←ParentSent ←W)

Return the amount of words that the section to which the sentence *S* belongs contains:

Amount ("WORD", ParentSec ←ParentPar ←S, True)

Return the level of the section parent of paragraph *P*.

Level (ParentSec ←P)

4. Application to the Contextual Exploration Method

The formalism presented in section 3 was implemented in Java. This implementation was used to write the semantic labeling module in a software platform for the Contextual Exploration method (the Contexto Platform[2]).

4.1. The Contextual Exploration Method

Several textual processing tasks, such as knowledge extraction or automatic summarizing, may be solved by analyzing exclusively linguistic units in the text, given that their linguistic context is taken into account. The Contextual Exploration method [Desclés 96] takes into account all the information about textual tokens (such as contextual word meaning, word location in the sentence, sentence or paragraph location in the text, structuration level of text, etc.) to assign semantic labels to sentences.

4.2. The Contexto Platform

Contexto is a software platform able to organize and exploit linguistic knowledge. In response to calls of specialized agents, an engine of contextual exploration starts, for one or more specific tasks, the process of recognition of the indicators and the indexes present in a textual segment. This process is carried out by a linguistic knowledge management system. This system furnishes to the exploration contextual engine a set of rules that could be potentially triggered. These rules were written using our implementation of the formalism presented in chapter 3.[3]

[2] Developed in the Lalic team of CAMS (UM 17 CNRS, EHESS, Paris IV)

[3] This work was developed in the frame of the program ECOS for the accomplishment of joint

A contextual exploration rule is associated to one or more indicators and belongs to a certain task. It has one or several conditions and one action. Each condition has associated one or more textual segments that determine one or more search spaces. The search spaces determine where the engine must look for the index specified in the conditions. Each condition expresses a restriction on the index (it is present or not in the specified search space) or on the location of a textual element (i.e. phrase, paragraph, section, etc.) in which is placed the indicator that triggered the rule. If a rule is successful (if their conditions are all satisfied) it activates its action part, which consists on the attribution of a semantic label to the textual segment.

5. Conclusions

In this work we have presented a text model and a clear and concise formalism that is independent of any implementation. In spite of that we developed an implementation in Java.

Our claim is that these tools represent a contribution to the development of applications that take into account researches about a textual linguistic.

We had verified, in the process of building an informatic platform for the Contextual Exploration method, the flexibility of the model and the formalism.

As a future work, we propose our text model to deal with other textual elements such as foot notes, lists of elements, tables, figures, etc.

References

[Adam 90] Adam Jean-Michel.(1990) *Éléments de linguistique textuelle*. Mardaga, Liège

[Charolles 88] Charolles, Michel (1988) *Les plans d'organisation textuelle; période, chaînes, portées et séquences*. Pratiques, n° 57, pp 3-13

[Chali et al 96] Chali, Yllias; Pascual, Elsa; Virbel, Jacques. (1996) *Text Structure Modeling and Language Comprehension Processes*. Joint Conference ALLC-ACH'96, Bergen, Norvège, 25-29 juin 1996. Actes 46-49

[Desclés 96] Desclés, Jean-Pierre. (1996) *Systèmes d'exploration contextuelle*. Actes du colloque sur le Calcul du sens et contexte. Université de Caen.

[Desclés et al 97] Desclès, Jean-Pierre; Cartier, Emmanuel; Jackiewicz, Agata; Minel, Jean-Luc. (1997) *Textual Processing and Contextual Exploration Method*. CONTEXT'97, Rio de Janeiro, Brasil.

[DOM] Document Object Model (DOM) Level 1 Specification Version 1.0
W3C Recommendation 1 October, 1998
http://www.wz.org/TR/1998/REL-DOM-Level-1-19981001/

[Halliday et al 76] Halliday, M.K; Hasan R. (1976) *Cohesion in English*. London: Longman

---

projects of scientific investigation between Uruguay and France. The collaboration was carried out between the *Lalic* team of *CAMS* and the group *TALN* of the *Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República (Uruguay)*