Natural Language Syntax Description using Generative Dependency Grammar

Ştefan Diaconescu

Abstract— **The paper presents a practical solution to describe natural language syntax. This solution is based on a Generative Dependency Grammar (GDG). A theoretical definition of these grammars and some of their proprieties is given. GDG are implemented in a declarative computer language GRAALAN (Grammar Abstract Language). The paper shortly present the features of GRAALAN and, after that, a more detailed implementation of natural language syntax description is given. GRAALAN offers for natural language syntactic description some strong features that respond to the following requests: a compact description, the possibility to express the syntax and the agreement and to specify the errors met in a text. The description has also the feature of reversibility. The paper presents some conclusions concerning the using of GRAALAN to describe the syntax (among others natural language features).**

Index Terms—**Dependency grammar, natural language syntax.**

I. Introduction

THE approach of different linguistic chapters in a unified manner was realized so far in many complex systems like EUROTRA [1], EAGLES [2], ROSETTA [20]. These large projects did not produce many successful implementations, but they are very important at least from theoretical point of view. One of the major drawbacks (among others) was the lack of unity among different linguistic chapters approach. Paradoxically, this lack of unity has grown for the worse due to the (successful) standardization effort of the different linguistic chapter representation, because the extremely useful approach of each individual section was not sufficiently correlated with the approach of other linguistic sections [11]. The language GRAALAN (Grammar Abstract Language) that will be very shortly presented in section II of this paper try to integrate many chapters of linguistic description and among these, the syntactic description of a natural language.

A lot of language models and language grammar types were proposed trying to solve the natural language description problem. There are three of the linguistic models that seem to be more successful and used in some applications [18]: TAG – Tree Adjoining Grammar [16], HPSG – Head-Driven Phrase Structure Grammar [17] and LFG – Lexical Functional Grammar [19].

During the last years another idea was more and more analyzed and studied: the dependency. Actually, it is quite an old idea – usually [21] is used as reference but the dependency idea in the grammar is millennial – but new valences and strength became attractive. The present paper is based on some researches that try to make a connection between two directions that seemed to be almost irreconcilable till now: the generative approach and the dependency approach. We present how this connection was done and implemented in the syntactic section of a GRAALAN (section II) in order to find a more adequate language model that could be used in natural language processing and that have the potential to produce many and better applications.

Some theoretical notions that are used to build GRAALAN are presented in the section III: DT - Dependency Trees, AVT - Attribute Value Trees, GDG - Generative Dependency Grammar and GDGF - Generative Dependency Grammar with Features. In section IV, it is presented how GRAALAN is used to describe the syntax under the form of rule sequence that indicates: the syntactic elements, the dependencies between these elements and the agreement between these elements. Finally (section V) some conclusions and the stage of current implementations are presented.

Manuscript received July 10, 2008. Manuscript accepted for publication October 20, 2008.

This work was supported in part by SOFTWIN SRL, Bucharest, Romania.

Stefan Diaconescu is with the SOFTWIN SRL, Bucharest, Romania (e-mail: sdiaconescu@softwin.ro).

II. GRAALAN: Grammar Abstract Language

GRAALAN is a language (in fact, a meta-language) that allows the description of a natural language and a correspondence between two natural languages. It contains some features that can be used to describe different natural language chapters (sections):

a) Alphabet Section defines the codes of the signs used to represent and describe the natural language. In this section the following information can be put: phonetic alphabet description (using, for example IPA – International Phonetic Alphabet [13]), normal alphabet and special characters (using. for example, UNICODE [14]), groups of characters (diphthongs or triphthongs, etc.) that contain the correspondences between some sequences of normal alphabet and phonetic alphabet, alphabetic classes (vowel class, consonant class, etc.). This section can describe also some special notation systems like those used by Japanese or Chinese languages.

b) Lexicon Section defines morphemes (roots, prefixes, suffixes, prefixoids, suffixoids, etc.), words (lemmas, some inflected forms of a word that accompanies the lemmas in an ordinary dictionary, for example, plural form of a noun), wordforms (some inflected form of another word that usually appears in a dictionary), multiword expression (MWE are groups of words represented as a DT - Dependency Tree), morphologic analytic structures, some typical syntactic structures (taken from the syntactic description), etc. For each lexicon entry some information belonging to the following types are present: semantic information (gloss, synonyms, antonyms, paronyms, hipernyms, hyponyms, connotations, homonyms, meronyms, etc.), etymology (original language, original form, transliteration of the original form), syllabification (euphonic, phonetic and morphologic), morphology (inflection situation, inflection rule identification, and segmentation), etc.

c) Syllabification Rules Section defines the syllabification rules for: euphonic syllabification (when the word is written with the normal or special alphabet), phonetic syllabification (when the word is written with the phonetic alphabet), morphologic syllabification (that respects the morphologic structure of the word). The elements of a word "separated" by syllabification (or not) are: the normal alphabet characters, groups (diphthongs, triphthongs, etc.) described in Alphabet Section (phonetic groups), some special characters, other constitutive elements (morphemes) described in Lexicon Section (morphologic groups).

d) Morphology Section defines morphologic categories and values. It is in fact an AVT - Attribute Value Tree (see section III.B), where attribute nodes are morphologic categories and value nodes are morphologic category values. Some information is attached with each type of node. For example, information attached to the attribute note is: the category name, the abbreviation of the category name, the indication if the category is inflected or not, (eventually) the name of a procedural program. Information attached to the attribute values are: the category value name, the abbreviation of the category value name, indication if it belongs to a lemma (or not), indication if it belongs to a lexicon entry (or not), (eventually) the name of a procedural program.

e) Inflection Rules Section defines the rules that can be used to generate the inflected forms. Lemma (from the lexicon) indicates a Compound rule. A compound rule is a list of basic rules. A basic rule contains an AVT where each leaf has one or more associated elementary inflection rules. An elementary inflection rule contains: a condition (logical expression) that indicates when the transformation sequence must be used, a transformation sequence (insert, delete, replace words or characters) acting on normal alphabet, a transformation sequence (insert, delete, replace words or characters) acting on phonetic alphabet form, an AVT for analytic forms, relations in a DT (dependency tree, see section III.A) for analytic forms.

f) Inflection Forms Section defines the inflected forms of the language. It contains an entry for an inflected form. An entry contains: the inflected form written using the normal alphabet, the inflected form written using the phonetic alphabet, the reference of the word in the lexicon whose inflected form is the current entry, the characterizing of the inflection situation (i.e., an AVT with lexical categories and lexical categories values), how the inflected form is syllabified in different situations: euphonic, phonetic, morphologic and at the end of the line (hyphenation).

g) Syntax Section defines the syntax rules (this section will be detailed in the following sections of the paper).

h) Bilingual Correspondences Section defines the correspondences between two languages for MWE (Multi Word Expression) correspondences [7] (it contains transformation rules based on dependency tree form of MWE, where nodes can be invariable elements, partial variable elements, total variable elements), word correspondences (particular cases of the MWE correspondences where both MWEs have only one word), syntactic structure correspondences (a particular case of MWE correspondences where the nodes can be non-terminals), morphologic analytic structure correspondences (a particular case of MWE where the correspondences is established between analytic inflection forms), morphologic sub-tree correspondences (a particular case of MWE too, that expresses the correspondences between a source morphologic sub-tree and a target morphologic sub-tree).

III. Graalan Theoretical Background

A. Dependency Tree

A generative dependency tree [3] is a 6-tuple $DT = \{N, T, P, A, SR, CR\}$ where:

- N - is the set of non-terminals n: $n(i_1, i_2, \ldots)$, $i_j \in SR$
- T - is the set of the terminals t: $t(i_1, i_2, \ldots)$, $i_j \in SR$
- P - is the set of pseudo-terminals p: $p(i_1, i_2, \ldots)$, $i_j \in SR$
- A - is the set of procedural actions a: $a(i_1, i_2, \ldots)$, $i_j \in SR$
- SR - is the set of subordinate relations sr: $sr(i_1)$, $i_1 \in N \cup T \cup \mathrm{P} \cup A \cup CR$
- CR - is the set of the coordinate relations cr: $cr(f_1, f_2, \ldots / s_1, s_2, \ldots)$, $f_i \in N \cup T \cup P \cup A \cup CR$, $s_i \in \mathrm{SR}$ ($f_1, f_2, \ldots$ are named fixed entry and $s_1, s_2, \ldots$ are named supplementary entry).

The non-terminals N are syntactic categories that can be described having a name and a structure.

The terminals T are words that can be found in the lexicon or can be obtained by applying some flexional rules on words from the lexicon.

The pseudo-terminals P are non-terminals that contain only terminals. When we will describe a dependency tree or a grammar we will not cover all the words from the lexicon because in this case the number of rules from the grammar can be too big. So, we can say that some non-terminals that we name pseudo-terminals (for example, some nouns or some verbs) will never be described in the grammar, but they are found in the lexicon.

The procedural actions (or "actions") A are the set of the routines that can be used to represent a certain portion of the text that we analyze. For example, a number represented like a sequence of digits or a mathematical formula or even an image with a certain significance that appear in a text can be “replaced” in grammars or dependency trees by a certain procedural action.

The subordinate relations *SR* are relations between a governor and a subordinate from the point of view of syntactic role in a phrase (for example the relation between a verb and a complement).

The coordinate relation *CR* are relations between two or many (but usually two) syntactic parts of a phrase, for example, the relation between *“read”* and *“write”* in the phrase *“I read and write.”*. The coordinated elements are represented by the fixed entries. A coordinate relation can also be a governor for the elements that came eventually on its supplementary inputs (that means that the set of coordinated elements form a governor for the elements that come on the supplementary inputs).

A dependency tree can be represented using the graphical symbols from Fig. 1.

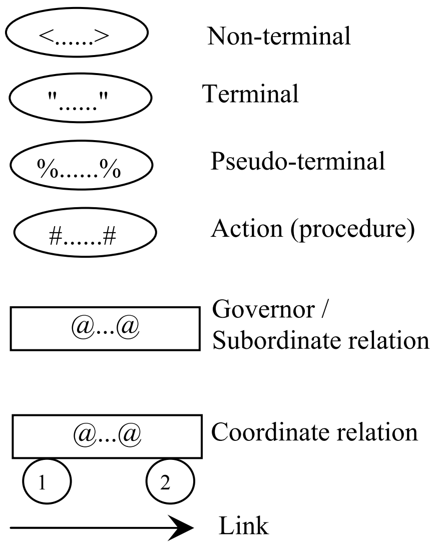

Fig. 1. Graphical DT symbols.

TABLE I.
LINKS IN A DEPENDENCY TREE

Link source	Link target				
	NTPA	GR	CR (supp. entry)	CR (fixed entry)	None
NTPA	▨	1	▨	2	7
GR	3	▨	6	▨	▨
CR	▨	4	▨	5	8
None	9	▨	10	▨	▨

The conditions respected by the links in a dependency tree are represented in Table I (there are 10 allowed situations) where we noted:

- NTPA: Non terminal (*N*) or Terminal (*T*) or Pseudo terminal (*P*) or Action (*A*).
- GR: governor/subordinate relation;
- CR: coordinates relation.

In a graphical representation:

- NTPA has maximum one input and maximum one output;
- GR has one input and one output;
- CR has maximum one output, zero, one or many supplementary input and a fixed number of fixed entry (we will consider only two fixed entry).

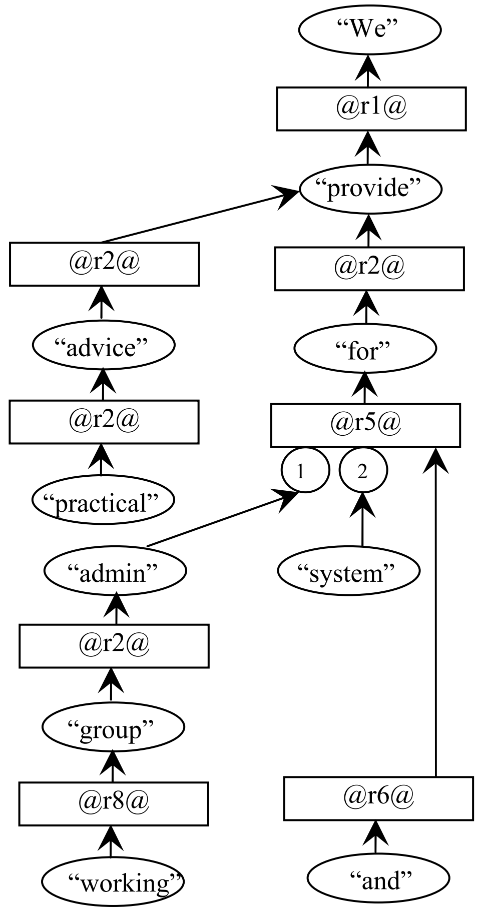

Fig. 2. Example of dependency tree.

We will consider also that the dependency tree is connected. As we can see, in this case, only one NTPA or one coordinate relation can have not output. This NTPA or coordinate relation will be named *head*.

The dependency trees will be used to define generative dependency grammar (see section III.C) and generative dependency grammar with features (see section III.D).

B. Attribute Value Tree

An attribute value tree (AVT) [4] [9] is used to describe morphologic or syntactic structures. It is in fact a list of attributes, each attribute having one or many values and each attribute value having associated one or many attributes. It can be defined as follows, using EBNF - Extended Backus-Naur Form from [22] without capital letter / lower case letter regular expression distinction:

[1] avt ::= ('{' S? label ':' S? attribute+ S? '}') | ('{' S? label S? '}') | ('{' S? attribute+ '}') | (attribute+)

[2] attribute ::= '[' S? <attribute content> S? ']'
[3] <attribute content> ::= (label ':' S? featureContent) | featureContent | label
[4] featureContent ::= attributeName S? '=' S? attributeValueList
[5] attributeValueList ::= attributeValueElement (S? ',' S? attributeValueElement)*
[6] attributeValueElement ::= attributeValueName (S? avt)*
[7] attributeValueName ::= label (S label)*
[8] label ::= labelChar (label)*
[9] labelChar ::= '_' | '-' | '.' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
[10] S ::= (#x20 | #x9 | #xD | #xA)+

Here *S* is any sequences of space, new line, carriage return or line feed characters.

We can see that in the description of an AVT we can use some labels that define some sub trees: labels for attributes lists (rule **[1]**) and labels for attribute content (rule **[3]**). These labels can be used in others parts of the tree and in this manner the tree is represented more compact.

A more formal definition of an AVT is given in [9]. The AVTs have different useful properties like: paths in the AVT, EC (Exclusive Combinations) in the AVT, equivalence, well formed AVT, ordering, intersection, difference, union, factoring, normalization, unifiability, unification. Among these properties, unifiability and the unification are the most important. They are used in the generation process for generative dependency grammar with features (see section III.D).

C. *Generative Dependency Grammar*

A generative dependency grammar is an 8-tuple $GDG = \{N, T, P, A, SR, CR, nt_0, R\}$ where:

- *N, T, P, A, SR, CR* are defined like in section III.A.
- nt_0 - belongs to N and is named root symbol.
- *R* - is the set of numbered rules of the form *(i)* $n_i ::=(p_i, q_i)$, $n_i \in N$, p_i is a sequence of elements from $N \cup T \cup P \cup A$, q_i is a dependency tree having nodes from p_i and oriented links (relations) from $SR \cup CR$.

In a GDG we can make generation that will build in the same time surface texts and dependency trees.

We give in the following an example of a grammar that can generate a phrase like:

"*We provide practical advice for system and working group administrators.*"

(1) <phrase> ::= (<nominal group> <verbal group>, <nominal group>(r1 (<verbal group>())))
(2) <nominal group> ::= ("we", "we"())
(3) <verbal group> ::= (<verb> <complement> <complement'>, <verb>(r2(<complement>()), r3(<complement'>())))
(4) <complement> ::= (<attribute> <noun>, <noun>(r4(<attribute>())))
(5) <complement> ::= ("for" <coordination>, "for"(<coordination>()))
(6) <coordination> ::= (<member> "and" <member'>, r5(<member>(), <member'>() / r6("and"())))
(7) <member> ::= (<noun>, <noun>())
(8) <member> ::= (<attribute> <noun>, <noun>(r7(<attribute>())))
(9) <attribute> ::= (<attribute> <noun>,<noun>(r8(<attribute>())))
(10) <attribute> ::= (<noun>, <noun>())
(11) <attribute> ::= (<adjective>, <adjective>())
(12) <attribute> ::= ("practical", "practical"())
(13) <verb> ::= ("provide", "provide"())
(14) <noun> ::= ("advice", "advice"())
(15) <noun> ::= ("system", "system"())
(16) <noun> ::= ("administrator", "administrator"())
(17) <noun> ::= ("group", "group" ())
(18) <adjective> ::= ("working", "working"())

Using this grammar we can generate the surface text as follows:

(19)(1, 2) <phrase> ::= ("we" <verbal group>, "we"(r1(<verbal group>( ))))
(20)(19,3) <phrase> ::= ("we" <verb> <complement> <complement'>, "we"(r1(<verb> (r2(<complement>( )), r3 (<complement'>( )))))
(21)(20,13) <phrase> ::= ("we" "provide" <complement> <complement'>, "we"(r1("provide"(r2(<complement>( )), r3(<complement'>( )))))
(22)(21,4) <phrase> ::= ("we" "provide" <attribute> <noun> <complement'>, "we"(r1("provide"(r2(<noun>(r4(<attribute>( )))), r3(<complement'>( )))))
(23)(22,5) <phrase> ::= ("we" "provide" <attribute> <noun> "for" <coordination>, "we"(r1("provide"(r2(<noun>(r4(<attribute>( )))), r3("for"(<coordination>( ))))))
(24)(23,12) <phrase> ::= ("we" "provide" "practical <noun> "for" <coordination>, "we"(r1("provide"(r2(<noun>(r4("practical"( )))), r3("for"(<coordination> ( ))))))
(25)(24,14) <phrase> ::= ("we" "provide" "practical" "advice" "for" <coordination>, "we"(r1("provide"(r2("advice"(r4("practical"( )))), r3("for"(<coordination> ( ))))))
(26)(25,6) <phrase> ::= ("we" "provide" "practical" "advice" "for" <member> "and" <member'>, "we"(r1("provide"(r2("advice"(r4("practical"( )))), r3("for"(r5 (<member>( ), <member'>( ) / r6("and"( ))))))))
(27)(26,7) <phrase> ::= ("we" "provide" "practical" "advice" "for" <noun> "and" <member'>, "we"(r1("provide"(r2("advice"(r4("practical"( )))), r3("for"(r5 (<noun>( ),

<member'>( ) / r6("and" ( ))))))))
(28)(27,15) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <member'>,
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5 ("system"( ),
<member'>( ) / r6("and"( ))))))))
(29)(28,8) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <attribute> <noun>,
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"( ),
<noun>(r7(<attribute>( ))) / r6("and"( ))))))))
(30)(29,16) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <attribute> "administrator",
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"(),
"administrator"(r7(<attribute>( ))) / r6("and"( ))))))))
(31)(29,9) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <attribute> <noun> "administrator",
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"( ),
"administrator"(r7(<noun>(r8(<attribute>( ))))) / r6("and"( ))))))))
(32)(31,17) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <attribute> "group" "administrator",
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"( ),
"administrator"(r7("group"(r8(<attribute>( ))))) / r6("and"( ))))))))
(33)(32,11) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" <adjective> "group" "administrator",
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"( ),
"administrator"(r7("group"(r8(<adjective>( ))))) / r6("and"( ))))))))
(34)(33,18) <phrase> ::= ("we" "provide" "practical" "advice" "for" "system" "and" "working" "group" "administrator",
"we"(r1("provide"(r2("advice"(r4("practical"( )))),
r3("for"(r5("system"( ),
"administrator"(r7("group"(r8("working"( ))))) / r6("and"( ))))))))

The final production we obtained contains in the left side of the right side the surface text and in the right side of the right side the dependency tree (represented in Fig. 2).

The GDG allows obtaining a structure from an unstructured text. This structure can be used in different purposes, for example in translation process, in defining correspondences between two languages [7].

D. General Dependency Grammar with Features

A GDG with feature structure is a GDG where each *ntpa* can have associated an AVT. The AVT associated with the non-terminal from the left side of the rules have always only indexed attributes.

<u>Example</u>

Let us have the next phrase in Romanian language: "*Ploile* (the rains) *văratice* (of summer) *sunt* (are) *călduțe* (lukewarm)" that means "*The summer rains are lukewarm*". We will not use all the grammatical categories involved in the analysis of this phrase but only few as an illustration.

Usually, the phrase to be analyzed is first of all annotated i.e. each word will have attached his lemma and a particular AVT (that have only one value for each attribute). Each word can have many interpretations. For example "*sunt*" can represent the third person plural (*are*) or the first person singular (*am*). Though, for the sake of simplicity, we will consider only one interpretation for each word.

The annotated phrase will be:

"Ploile" *ploaia* [class = noun] [gender = feminine] [number = plural] "văratice" *văratic* [class = adjective] [gender = feminine] [number = plural] "sunt" (a) *fi* [class = verb] [person: III] [number = plural] [mode = indicative] [voice = active] [time = present] "călduțe" *călduț* [class = adjective] [gender = feminine] [number = plural]

We marked the lemmas using italics.

A GDG with features that can generate this phrase can be as follows:

(1) <phrase> ::= (<nominal group> [gender = masculine, feminine, neuter] [number = singular, plural] [person = I, II, III] <compound nominal predicate> [gender = masculine, feminine, neuter] [number = singular, plural] [person = I, II, III], <nominal group>(@r_1@(<compound nominal predicate> ())))
(2) <nominal group> [gender = masculine, feminine, neuter] [number = singular, plural] [person = I, II, III] ::= (%noun% [class = noun] [gender = masculine, feminine, neuter] [number = singular, plural] %adjective% [class = adjective] [gender = masculine, feminine, neuter] [number = singular, plural], %noun%(@r_2@(%adjective% ())))
(3) <compound nominal predicate>[gender = masculine, feminine, neuter] [number = singular, plural] [person = I, II, III] ::= (%verb% [class = verb] [gender = masculine, feminine, neuter] [number = singular, plural] [mode = indicative] [voice = active] [time = present, future, imperfect past] %adjective% [class = adjective] [gender = masculine, feminine, neuter] [number = singular, plural], %verb%(@r_3@(%adjective% ())))

As we can see, we used pseudo terminals for nouns, verbs, adjectives, so this grammar can generate a set of phrases.

IV. Natural Language Syntax Description in GRAALAN

A. General Structure

The description of the syntax in GRAALAN [8] [10] will use GDG and AVT presented in section III. The language where we are describing the syntax must respect the following conditions:

a) Syntax: The description language will allow the description in a detailed and compact form of the manner to combine words in phrases respecting the rules of a natural language grammar.

b) Dependencies: We accept here that the dependency aspects are reduced to the mode different parts of a phrase are in relation one another (coordinate and governor/subordinate relations).

c) Agreement: By agreement [5] we will understand the mode different part of speech "match" one another when they are in certain dependency relations from the point of view of the values of different morphologic or syntactic categories.

d) Errors: The natural language syntax description must allow indicating the errors (at least the most frequent ones) that can be found in phrases. The bad built phrases must be recognized (in a certain measure) and marked as being incorrect.

e) Reversibility: By reversibility we will understand the property of the description language to be used to convert a source (surface) text in a deep structure (the dependency tree, in our case) and to convert the deep structure into the surface text.

We will give here an informal definition of natural language syntax description in GRAALAN. A more detailed definition of natural language syntax description in GRAALAN is given in the next sections.

A GRAALAN syntax description is a sequence of labeled rules. A rule has two parts: the left part and the right part. The left part of a rule contains a non terminal and an AVT. The AVT contains syntactic / morphologic categories with their values. The right part of a rule contains one or many Alternants. An alternant is formed by a set of subsections: the syntactic subsection, the dependency subsection and the agreement subsection.

a) The syntactic subsection is a sequence of (eventually labeled) one or many NTPAs. Each NTPA can have associated information about how this NTPA is linked with others NTPA by certain relations from the dependency subsection (these relations are indicated by their labels in the dependency subsection). There are three lists concerning the relations: coordinated list (CL), subordinated list (SL) and government list (GL).

Each NTPA can have associated an AVT describing syntactic / morphologic categories with their values.

b) The dependency subsection contains the description of the relations between the NTPA from the syntactic subsection (referred by their labels). There are two types of relations:

The subordinate relation SR is a relation between two N, T, P, A, or a coordinate relation CR. One of the two elements is considered to be the governor (that governs by SR) and the other the subordinate (that is governed by SR).

The coordinate relation CR is a relation between (usually) two N, T, P, A (that are said to be coordinated by CR), and eventually one or many SR (by which the CR is considered to be a governor for others N, T, P, A, or CR).

c) *The agreement subsection* contains a list of agreement rules. An agreement rule is a conditional expression expressed between the categories values of the NTPAs from the syntactic subsection. It can indicate some actions like error messages or how the analyze will be continued after an agreement error is found.

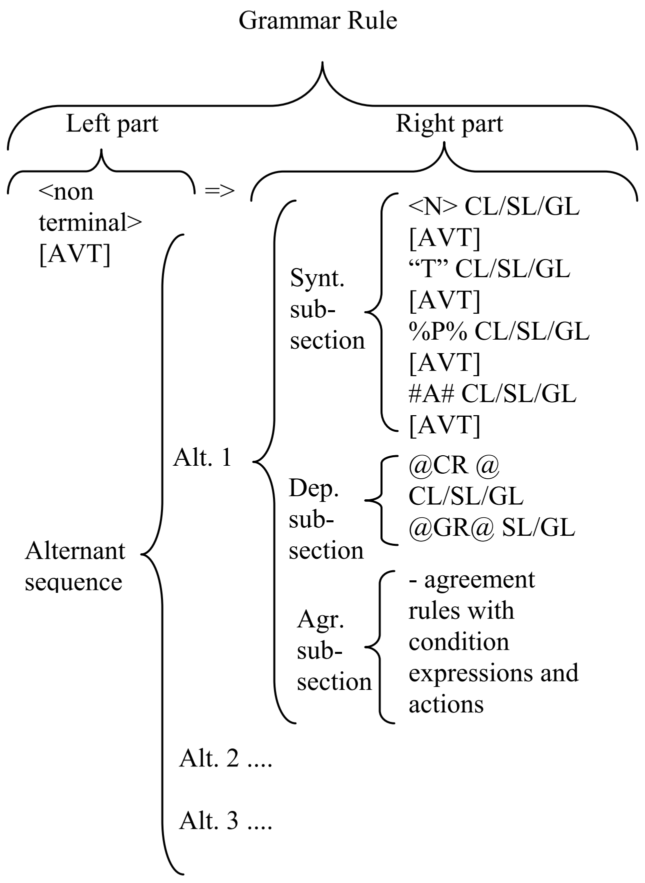

Fig. 3. Syntax rule elements

B. Graalan Syntactic Section Header

The syntactic section of a GRAALAN description is a sequence of rules preceded by a header. The description of the header in EBNF is the following:

[1] syntaxSection ::= 'Section' S 'syntax' S sectionHeader syntax S 'end' S 'of' S 'section'
[2] sectionHeader ::= (sourceLanguage, exploitationLanguage, sourceDirection, exploitationDirection)

A header is formed by elements that refer the source language and exploitation language.

[3] S ::= (#x20 | #x9 | #xD | #xA)+

Here *S* is any sequences of spaces, new line, carriage return or line feed characters.

[4] sourceLanguage ::= 'Source' S 'language' S language S
[5] exploitationLanguage ::= 'Exploitation' S 'language' S language S
[6] sourceDirection ::= 'Source' S 'direction' S ('left' | 'right')

[7] exploitationDirection ::= 'Exploitation' S 'direction' S ('left' |'right')
[8] language ::= ('RUM' | 'FRA' | 'FRA' | 'SPA' | 'RUS, ...')

We understand by *direction* the mode to scan the source text: *right* - the scan is done from left to right; *left* - the scan is done from right to left. The language is indicated according to [15].

[9] syntax ::= (rule S)+
[10] rule ::= 'Rule' S label ':' S '<' S? name S? '>' S? attribute* S? '::=' S? (('Alternant' S label ':' S? alternantContent)+ | ('Alternant' S alternantContent))

The alternant labels must be unique in the rule.

[11] name ::= label (S label)*
[12] label ::= labelChar (label)*
[13] labelChar ::= '_' | '-' | '.' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
[14] alternantContent ::= (syntacticSubsection ((dependencySubsection agreementSubsection?) | agreementSubsection)?)

An alternant can contain the three types of subsection (syntactic, dependency and agreement) in different combinations.

C. Syntactic Subsection

The syntactic subsection of a GRAALAN syntax rule contains information about a set of NTPAs that must be found in the analyzed / generated source text, in the corresponding sequence. The description of the syntactic subsection in EBNF is the following:

[15] syntacticSubsection ::= 'Syntax' S ((notRelationedNTPA S tpaRelationalList*)+) | ((notRelationedNTPA S nRelationalList*)+)
[16] notRelationedNTPA ::= (label ':' S?)? (('<' S? name S? '>') | ('"' terminal '"') | ('%' S? name S? '%') | ('#' S? label S? '#')) S attribute* (S ntpaBehaviour)*
[17] terminal ::= char (terminal)*
[18] char ::= &label; | &code;|...

Here *char* can be &label; or &code; or any character defined in GRAALAN Alphabet section (not described in this paper).

[19] code ::= '#x' hexaString
[20] hexaString ::= hexaChar (hexaString)*
[21] hexaChar ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
[22] ntpaBehaviour ::= ntpaBehaviourElement (S ntpaBehaviourElement)*
[23] ntpaBehaviourElement ::= '!' | (('OK' | 'KO') S? '=' S? ('OK' | 'KO')) | ('Message' S? = S? label) | ('Context' S? = S? '(' S? label (S label)* S? ')'))

Each element from *ntpaBehaviourElement* that characterises the NTPA behavior can appear only once in an *ntpaBehaviour*.

An NTPA that is head can have associated a feature named “cut” and indicated by “!”. This means that the head loses the characteristic of head. The corresponding NTPA will not have relations with other NTPAs. The grammar must be written in order to obtain a connected dependency tree. One alternant will have only one head. When we use another rule to substitute a non terminal in the current alternant, the different links of the substituted non-terminal will be applied to the head of the used rule.

The others elements that appear in *ntpaBehaviourElement* form the error sequence. In such a sequence the *Message* is mandatory. The error message itself is indicated by a *label* that defines the error text in another GRAALAN section (the GRAALAN Message Section that is not presented in this paper).

The condition ('OK' | 'KO') S? '=' S? ('OK' | 'KO') indicates the error triggering. The left of “=” indicates how NTPA treatment is terminated and the right side of “=” indicates the error condition, see Table II.

TABLE II
ERROR TRIGGERING FOR A NTPA

How NTPA treatment is terminated	Error condition	Continue as it was not an error
OK	*OK*	Yes (i)
OK	*KO*	No
KO	*KO*	Yes(ii)
KO	*OK*	No

The two cases when the treatment is continued as it were not an error has the following significance:

i) We found an error but this error is described as it was correct.

<u>*Example*</u>
(A simplified description.)

<imperative phrase>::=
<vocative nominal group> "," <imperative verbal group> "!"|
<vocative nominal group> <imperative verbal group >"!"

The first alternant corresponds to a phrase of the form (in Romanian):
"Domnilor, vorbiți mai încet !" (Gentlemen, shut up!)
(with a comma after the vocative).
The second alternant corresponds to a phrase of the form (in Romanian):
"Domnilor vorbiți mai încet!" (Gentlemen shut up!)
(without a comma after the vocative, therefore incorrect).
Though we have an error, the sense remains clear.

During the analysis of this incorrect phrase, the second alternant will return an OK value after the terminal "!". If we attach in the second alternant an error condition we will can indicated the error apparition:

```
Rule R1: <imperative phrase>::=
    Alternant A1:
        Syntax
                Label1:<vocative nominal group>
                        Coordinate Label5(1)
                Label2: ","!
                Label3: <imperative verbal group>
                        Coordinate Label5(2)
                Label4: "!"!
        Dependencies
                Label5: @vocative relation@(2)
    Alternant A2:
        Syntax
                Label1: <vocative nominal group>
                        Coordinate Label4(1)
                Label2: <imperative verbal group>
                                OK = OK Message =
                        ErrorMessage
                        Coordinate Label4(2)
                Label3: "!"!
        Dependencies
                Label4: @vocative relation@(2)
```

ii) There are situations when an NTPA must return OK but, if it returns KO, we have an error that must be identified as it is. Let us suppose a grammar fragment that must recognize phrases formed by a verbal group or by two verbal groups linked by "and".
(A simplified description.)

```
<phrase>::= <nominal group> <verbal group>"."|
    <nominal group><verbal group> "and"<verbal group>"."
```

In such a grammar, after an "and" was detected, a <verbal group> must be found. We can attach to the non terminal <verbal group> that is written after "and" an error sequence that must be triggered to KO return by this non terminal (giving an error message). A phrase like "*He came and.*" must produce this error message

The (simplified) description could be, for example:

```
Rule R1: <phrase>::=
    Alternant A1: <nominal group> <verbal group> "."
    Alternant A2: <nominal group> <verbal group> "and"
                <verbal group> KO = OK Message =
        Error101
                "."
```

A more detailed description:

```
Rule R1 <phrase>::=
    Alternant A1:
        Syntax
                Label1: <nominal group> Governor Label4
                Label2: <verbal group> Subordinate Label4
                Label3: "."!
        Dependencies
                Label4: @nominal / verbal relation@
    Alternant A2:
        Syntax
                Label1: <nominal group> Governor Label7
                Label2: <verbal group> Coordinate
        Label6(1)
                Label3: "and"!
                Label4: <verbal group>
                    KO = OK Message = Error101
                    Coordinate Label6(2)
                Label5 "."!
        Dependencies
                Label6: @"and" coordination@(2)
                        Subordinate Label7
                Label7: @nominal / verbal relation@
```

[24] tpaRelationalList ::= ('Governor' S labelList) | ('Subordinate' S labelList) | ('Coordinate' label S? '(' S? ('1' | '2' S?) ')')
[25] nRelationalList ::= ('Governor' S labelList) | ('Subordinate' S labelList) | ('Coordinate' label S? '(' S? ('1' | '2' S?) ')') | ('External' labelList)
[26] labelList ::= label (S label)*

A NTPA with relations is an NTPA that is linked with other NTPAs by relations (governor, subordinate, coordinate).

The *Subordinate* list of an NTPA can contain only the label of a *Subordinate* relation from the *Dependency* subsection.

The *Coordinate* list of an NTPA can contain only one label of a coordinate relation (that will be referred on a fixed entry) from *Dependency* subsection.

The *Governor* list of an NTPA can contain one or many labels of subordinate relations from the *Dependency* subsection.

For the same NTPA: the *Subordinate* list and *Governor* list can coexist, the *Governor* list and the *Coordinate* list can coexist, and the *Subordinate* list and the *Coordinate* list are exclusive.

The *External* list of can appear only for a nonterminal. It refers relations from *Dependency* subsection that appear wit the attribute *Definition* or *Reference* [6]. We must respect the discipline that, in the process of applying the rules (the generation of a new rule from two rules) always the external "definition" must appear before the corresponding external "reference". An external definition can appear only in a relational list of a non terminal, because only a non terminal can carry them further to someone that needs it.

D. Dependency Subsection

The dependency subsection of an alternant contains information about the relations that are defined between the NTPAs from the syntactic subsection of the alternant. The description of the dependency subsection in EBNF is the following:

[27] dependencySubsection ::= 'Dependencies' S (coordinateRelation | subordinateRelation)+
[28] coordinateRelation ::= label ':' S? ((('Reference' S) | ('Definition' ' S ')) ? '@' S? name S? '@' S? '(' S? '2' S? ')' (S? '!')? (('Subordinate' S label)? | ('Coordinate' S label S? '(' S? ('1' | '2') S? ')')?) ('Governor' (S label) | (S label S? '(' S? ('1' | '2') S? ')')+)?

A coordinate relation label must be unique among the alternant labels.

The *'Reference'* key word indicates (if present) that the current coordinate relation is not defined in the current rule but it is referred from another rule.

The *'Definition'* key word indicates (if present) that the current coordinate relation is defined in the current rule and it will be referred from other rules.

A coordinate relation can be followed by cut (“!”) because a coordinate relation can be head and using cut this feature will be discarded.

The *Subordinate* list of a coordinate relation can contain only one label of a subordinate relation from *Dependency* subsection.

The *Coordinate* list of a coordinate relation can contain only one label of a coordinate relation from *Dependency* subsection (referred on a fixed entry).

The *Governor* list of a coordinate relation can contain one or more labels of governor / subordinate relation from *Dependency* subsection. The current coordinate relation can be:

- governor for other NTPAs or coordinate relations (using a governor / subordinate relation indicated by a label not followed by a number in parenthesis); this means that the links from these governor / subordinate relations will come on the current coordinate relation on supplementary inputs);
- “governor” for other NTPAs or coordinate relations (using a label followed by a number of a fixed entry); this means that the links from these NTPAs or coordinate relations will appear on the corresponding fixed entry of the current coordinate relation.

For the same coordinate relation: the *Subordinate* list and the *Coordinate* list are exclusive and each of them can coexist with *Governor* list.

[29] subordinateRelation ::= label ':' S? ((('Reference' S) | ('Definition' ' S ')) ? '@' S? name S? '@' (('Subordinate' S label)? , ('Governor' S label)?)

A subordinate relation label must be unique among the alternant labels.

A subordinate relation has always one entry; so, we do not need to specify the number of entries. In fact, the presence of the entry number in coordinate relation indicates the fact that it is a coordinate relation.

The 'Reference' and 'Definition' key words have the same significance as for coordinateRelation.

The *Subordinate* list of a subordinate relation can contain only one label of a coordinate relation from the *Dependency* subsection (where it will go on a supplementary input) or of an NTPA from the *Syntax* subsection.

The *Governor* list of a subordinate relation contains only one label of an NTPA from the *Syntax* subsection or of a coordinate relation from the *Dependency* subsection.

Observation 1: Because a link has two extremities, it can be indicated by the any of its ends or by both. There are many ways to indicate a link according to its type. It is advisable to make the grammar description in such a way that a link appear only once (to the element from where the link leaves or to the element where the link arrives). See TABLE III where we use the notations:

GR = Governor/Subordinate Relation
CR = Coordinate relation
SL = Subordinate List
GL = Governor List
CL = Coordinate List

TABLE III
INDICATION OF LINKS

Link type	Link source (label A)	Link target (label B)	How the link is indicated	
			1	2
1	NTPA	GR	B in SL of A	A in GL of B.
2	NTPA	CR (on fixed entry)	B in CL of A (with fixed entry number of B)	A in GL of B (with fixed entry of B)
3	GR	NTPA	B in SL of A	A in GL of B
4	GR	CR (on supp. entry)	B in SL of A	A in GL of B
5	CR	GR	B in SL of A	A in GL of B
6	CR	CR (on fixed entry	B in CL of A (with fixed entry number of B)	A in GL of B (with fixed entry number of B)

Observation 2: In the Table IV it is indicated that we can put in different relational lists function of the element that the list belongs to.

<u>*Example*</u>

```
… ::= …<non terminal 1>
                    {Sequence:
                      (gender = masculine, feminine, neutral)
                      (number = singular, plural)}
          <non terminal 2> {Sequence}
          <non terminal 3> {Sequence}
```

TABLE IV
THE RELATIONAL LIST CONTENT

List type	List owner	List content
GL	NTPA	The labels of one or many governor / subordinate relations that have outputs going to the current NTPA.
	GR	The label of a coordinate relation or of an NTPA that have outputs going on the input of the current governor / subordinate relation.
	CR	The labels of one or many governor / subordinate relation (that have outputs going to the supplementary input of the current coordinate relation) and / or labels of some NTPAs or other coordinate relation that have outputs going on fixed entries of the current coordinate relation. In this case, the corresponding number of fixed entry is indicated too.
SL	NTPA	The label of only one governor / subordinate relation that have an input where the output of the current NTPA goes.
	GR	The label of a coordinate relation (where the output of the current governor / subordinate relation will go on a supplementary input) or of an NTPA (that have an entry where the output of the current governor / subordinate relation will go).
	CR	The label of a governor / subordinate relation that has an input where the output of the current coordinate relation will go.
CL	NTPA	The label of only one coordinating relation (where the output of the current NTPA will go on a fixed entry). In this case the number of the fixed entry is also indicated.
	CR	The label of only one coordinating relation (where the output of the current coordinate relation will go on a fixed entry). In this case the number of the fixed entry is also indicated.

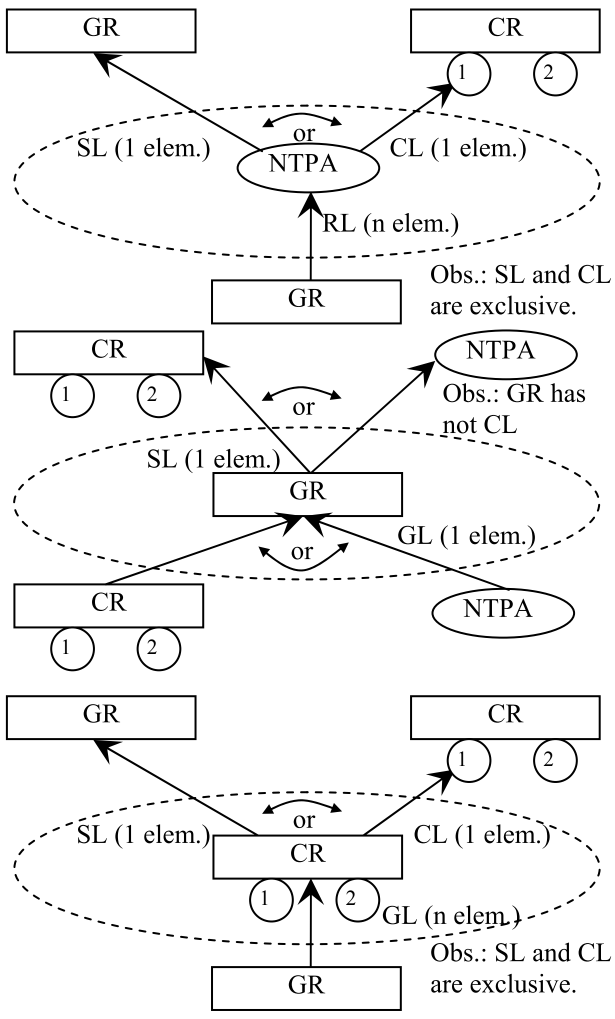

Fig. 4. The content of *Governor*, *Subordinate* and *Coordinate* lists (GL, SL, CL)

[30] attribute ::= notIndexedAttribute | indexedAttribute | '{' S? label ':' S? attribute+ S? '}' | '{' S? attribute+ S? '}' | '{' S? label S? '}'

In this representation, the *label* of an *attribute* sequence allows to compact the rule. If the same attribute sequence appears many times in a rule (in the left side or in the alternants from the right side), then the first apparition of the sequence can be labeled and the following apparitions can be indicated only by using this label. A label of an attribute sequence must be unique in the current rule.

[31] notIndexedAttribute ::= '(' (((S? label ':' S?)? attributeContent) | label) ')'
[32] indexedAttribute ::= '[' (((S? label ':' S?)? attributeContent) | label) ']'
[33] attributeContent ::= category S? '=' S? categoryValue (S? ',' S? categoryValue)*

In this representation, the *label* of an *attributeContent* allows to compact the rule. If the same attribute appears many times in a rule (in the left side or in the alternants from the right side), then its first apparition can be labeled and the following apparitions can be indicated only by using this label. A label of an attribute must be unique in the current rule.

Example

```
… ::= …<non terminal 1>
                (Gender1: gender = masculine, feminine,
        neutral)
                (Number1: number = singular, plural)
        <non terminal 2>
                (Gender2: gender = masculine, feminine)
                (Number2: number = singular)
        <non terminal 3>
                (Gender1) (Number1)
        <non terminal 4>
                (Gender2) (Number2)
```

If an *indexedAttribute* contains a *label* then this label will play the role of an index. If the *indexedAttribute* do not have a *label*, then the *category* from *attributeContent* will play the index role. In any cases, *categoryValue* from *attributeContent* represent all the values that the index can take.

Example

Let us have a set of rules of the form:

```
<complex subjective group>
        [person = I, II, III]
        [number = sg, pl]
        [gender = m, f, n]
        ::=
        Alternant A1:
                <unitary subjective group>
                [person = I, II, III]
                [number = sg, pl]
                [gender = m, f, n]
        Alternant A2:
                <logical subjective group>
                [person = I, II, III]
                [number = sg, pl]
                [gender = m, f, n]
        Alternant A3:
                <distributive subjective group>
                [person = I, II, III]
                [number = sg, pl]
                [gender = m, f, n]
        Alternant A4:
                <correlative subjective group>
                [person = I, II, III]
                [number = sg, pl]
                [gender = m, f, n
```

Considering the combinations for person, number and gender, this rule represents in fact 18 rules.

Example

The same thing can be written more compact as follows:

```
<complex subjective group>
        [e1: persoana = I, II, III]
        [e2: number = sg, pl]
        [e3: gen = m, f, n]
        ::=
        Alternant A1:
                <unitary subjective group> [e1][e2][e3]
        Alternant A2:
                <logical subjective group>[e1][e2][e3]
        Alternant A3:
                <distributive subjective group>[e1][e2][e3]
        Alternant A4:
                <correlative subjective group>[e1][e2][e3]
```

Example

A more important using of the indexing is when the same category must serve as index in many ways in the same alternant.

```
<non terminal1>
        [e1: attribute1 = value11, value12, value13]
        [e2: attribute2 = value21, value22]
        [e3: attribute3 = value31, value32, value33]
        ::=
        Alternant A1:
                <non terminal2>[e1][e2][e3]
                <non terminal3>
                [e4: attribute1 = value11, value12, value13]
                [e5: attribute2 = value21, value22, value23]
                [e6: attribute3 = value31, value32, value33]
                <non terminal4>[e4][e5][e6]
```

In this example, the attributes *e1: attribute1*, *e2: attribute2*, *e3: attribute3* are considered as indexes different from *e4: attribute1*, *e5: attribute2*, *e6: attribute3* (i.e., for example, *<non terminal1>* and *<non terminal2>* must have the same value for *attribute1*, *<non terminal3>* and *<non terminal4>* must have the same value for *attribute1* but the values for *attribute1* can be different in *<non terminal1>* and in *<non terminal3>*, etc.)

[34] categoryValue ::= name S attribute*

We can see that a *categoryValue* can be followed by an *attribute* sequence. In this way, a branching in an attribute value tree is represented. A sequence of “category = value” that pass by such branching points represents a path in the AVT. The syntax must be written in such a way that a path do not have many apparition of the same category.

E. Agreement Subsection

The agreement subsection of an alternant describes the conditions that must be respected by the morphologic / syntactic categories of the NTPA from the syntactic subsection. The description of the agreement subsection in EBNF is the following:

[35] agreementSubsection ::= 'Agreement' agreementRule+
[36] agreementRule ::= 'if' S? '(' S? conditionExpression S? ')' S? alternatives+ ((S? 'else' S? '(' S? agreementRule S? ')'

S?) | (S? 'else' S? agreementRule S?) | (S? 'else' S? '(' S? actionList S? ')'))?

[37] alternatives ::= ('true' S? '(' S? expression S? ')') | ('false' S? '(' S? expression S? ')') | ('not' S? 'applicable' S? '(' S? expression S? ')') | ('not' S? 'determinated' 'S? (' S? expression ')')

[38] expression ::= actionList | agreementRule

A *conditionExpression* can have one of the four truth values. We will use a tetravalent logic that has the following truth values: *TRUE*, *FALSE*, *NOT APPLICABLE*, *NOT DETERMINATE*. Therefore, after *'if' S? '(' S? conditionExpression S? ')' S?* we must have a list of maximum four alternatives and these alternatives must be different. If some alternatives are missing, they can globally be treated using *else*.

<u>*Example*</u>

Let us have the following sequence:

if (conditionExpression)
true(expression 1)
not applicable(expression 2)
else(expression 3)

Such an expression is read: "if *conditionExpression* is true then execute *expression1* and if *conditionExpression* is *not applicable* then execute *expression2* otherwise (i.e. *conditionExpression* is *false* or *not determinated*) then execute *expression3*".

[39] conditionExpression ::= ('(' S? conditionExpression S? ')' S? logicalOperator S? conditionExpression) | (S? '~' S? '(' S? conditionExpression S? ')' S? logicalOperator S? conditionExpression) | ('(' S? conditionExpression S? ')') | ('~' S? '(' S? conditionExpression S? ')') | (simpleExpression S? logicalOperator S? simpleExpression) | simpleExpression

In order to formulate the logical value of the *conditionExpression* we can use *logicalOperators* (including the negation "~"), parenthesis and operands that are *simpleExpression*.

[40] logicalOperator ::= 'and' | 'or'

[41] simpleExpression ::= ({operand} S? '+ S? {operand} S? '<-' S? {operand}) | ({operand} S? '<-' S? {operand}) | ({operand} s? '->' S? {operand}S? '+' S? {operand}) | ({operand} S? '->' S? {operand})

[42] operand ::= label attribute+

An operand indicates an NTPA that is involved in the agreement. The agreement is usually expressed between a governor and a subordinate. Let us have the example: *"Not only the rain but also the wind corrode the cliffs."* Between "*Not only the rains but also the winds"* as multiple subject and *"corrode"* must be described an agreement. (We can also describe a *sort of* agreement also between *"not only"* and *"but also"* as two parts of a correlation.)

If we have a governor / subordinate relation, then the operator representing the governor will be at left of *"<-"* or at right of *"->"* (the arrow looks at the governor).

An expression of the form *operand*$_1$ + *operand*$_2$ <- *operand*$_3$ or *operand*$_3$ ->*operand*$_1$ + *operand*$_2$ is read: "if *operand*$_1$ has some features a$_1$ (attributes and values: a certain gender, a certain number, etc.) and *operand*$_2$ has certain features a$_2$ then the *operand*$_3$ must have certain features a$_3$.

An expression *operand*$_1$ <- *operand*$_2$ or *operand*$_2$ -> *operand*$_1$ is read: "if *operand*$_1$ has certain features a$_1$ then *operand*$_2$ must have certain features a$_2$".

The *operand* contains a *label* of an NTPA (from the syntactic subsection of the current alternant) involved in the agreement and an *attribute* under the form of an AVT.

An AVT (indicated by *attribute*) of an operand must be unifiable with the AVT associated to the corresponding NTPA (indicated by *label*).

A *simpleExpression* is TRUE when all its operands have AVTs unifiable with the corresponding AVT from the syntactic subsection.

A *simpleExpression* is FALSE when the AVT corresponding to the operands represented the governor is unifiable with the corresponding AVT of NTPA from syntactic subsection and those representing the subordinate are not.

A *simpleExpression* is NOT APPLICABLE if at least one of the operands representing the governor has a not unifiable AVT.

(A value of NOT DETERMINED can appear only by the evaluation of the *conditionExpression* containing simple expressions.)

<u>*Example*</u>

Let us have two non terminals that appear in syntactic subsection of an alternant:

Label1: <non terminal1>	(a = av1, av2)
	(b = bv1, bv2, bv3)
	(c = cv1, cv2, cv3)
Label2: <non terminal2>	(d = dv1, dv2)
	(e = ev1, ev2, ev3)
	(f = vf1, vf2, vf3)

Let us have the simple expression of the form:

Label1(a = av1, av2)(b = bv2, bv3) -> Label2(e =ev1, ev2)(f = vf2)

During the syntactic analysis, after the current alternant was analyzed and recognized in source text, <non terminal1> and <non terminal2> will have only some of the above attributes/values.

The operand Label1(a = av1, av2)(b = bv2, bv3) will be unifiable with <non terminal1> when this one will have after the syntactic analysis:

- the category a with the values av1 or av2;
- the category b with the values bv1 or bv3.

The operand Label2(e =ev1, ev2)(f = vf2) will be unifiable with the <non terminal2> when this one will have after the syntactic analysis:

- the category e with the values ev1 or ev2;
- the category f with the value vf2.

Example

Let us have two non terminals that appear in syntactic subsection of an alternant:

```
Label1:   <elementary nominal group>
          (negation = affirmative, negative)
          (person = I, II)
          (number = singular)
          (gender = masculine, feminine)
Label2:   <verbal group>
          (negation = affirmative, negative)
          (person = I, II)
          (number = singular)
          (gender = masculine, feminine)
```

Let us have the expression of the form:

```
Label1(person = I) <- Label2(person = I)
or
Label1(persoana = II) <- Label2(person = II)
```

This expression will be *TRUE* when the non terminal with the labels Label1 and Label2 will have the same person (I or II).

Using the indexed representation of the attributes, the expression can be written more compact:

```
Label1[person = I, II] <- Label2[person = I, II]
```

[43] actionList ::= action (S? ',' S? actionList)*
[44] action ::= ('Message' S? '=' S? label) | ('OK' | 'KO') |
('Context' S? = S? '(' S? label (S label)* S? ')')

The *Message* is an error message indicated by a label in another GRAALAN section not described in this paper (where messages in different languages can be found). This message will be displayed during the syntactic analysis of a source text.

The mode OK | KO indicated how the current NTPA situation must be treated:

- KO: negative;
- OK: positive.

Observation 3: An agreement rule between different NTPAs of an alternant make sense only if all these NTPAs have associated attributes with many values in the syntactic subsection of the alternant. If the NTPAs have not attributes or they have attributes but all the attributes have only one value then the agreement problem is solved by the syntactic description itself and we do not need an agreement rule.

The message *Context* is represented by the labels of certain NTPA or relations that can be used in debugging process.

Example

Let us take an agreement expression of the form:

```
if (Label1(person = I) -> Label2(person = I))
     true ( OK )
     else ( Message = Label3, OK)
```

It will be read: “If the NTPA with the label Label1 from the syntactic subsection has the person I and the NTPA with the label Label2 from syntactic subsection has the person I then continue the syntactic analysis, otherwise display the error message with the label Label3 (and that can be for example defined in Message section of GRAALAN language like: "Person number agreement error") and continue after that the syntactic analysis as it was not an error.”

V. CONCLUSION

We presented a method to describe the syntax of a natural language. The description mode is part of a more general language GRAALAN that allows the description of a natural language or the correspondences between two natural languages. We consider that this kind of description is quite general and fit for almost any natural language.

In order to use the description method, some tools are needed [11]. Among these tools, a very important one is GRAALAN compiler. This compiler analyzes the description GRAALAN text and converts it to XML. The XML information will form an LKB (Linguistic Knowledge Base). The knowledge form LKB can afterwards be used to build different linguistic applications: morphologic analyzer, grammar checker, inflection application, indexing / searching application, lemmatizer, speller, hyphenating application, different kinds of lexicons and dictionaries, different kinds of machine translation applications (human assisted machine translation, computer assisted machine translation, automatic machine translation), etc.

Some tools for GRAALAN are already developed (GRAALAN Macro processor, GRAALAN Compiler, Inflection Forms Tool that allows an automatic / interactive generation of the inflected forms). Some tools are currently in design / implementation stage (Lexicon Tool that allows an automatic / interactive lexicon creation, LINK that checks the coherence of an entire LKB). Some Romanian linguistic knowledge bases are already defined (Alphabet Section, Morphologic Configurator Section, Syllabification Section, Inflection Rules Section), some are partially developed (Lexicon Section, Syntax Section), some will be soon developed (Inflection Forms Section).

We hope that the system built on GRAALAN will be an important tool to elaborate some unified and very general linguistic applications.

REFERENCES

[1] H. Alshawi, D. J. Arnold, R. Backofen, D. M. Carter, J. Lindop, K. Netter, S. G. Pulman, J. Tsujii, H. Uszkoreit. EurotraET6/1: Rule

Formalism and Virtual Machine Study. Final Report. Commission of the European Communities, 1991.

[2] R. Backofen et al. EAGLES Formalism Working Group Final Report. Expert Advisory Group on Language Engineering Standards, 1996.

[3] S. Diaconescu. Natural Language Understanding Using Generative Dependency Grammar. In: M. Bramer, A. Preece and F. Coenen (Eds), *Twenty second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, pp.439-452, Cambridge UK, Springer, 2002.

[4] S. Diaconescu. Morphological Categorization Attribute Value Trees and XML. In: M. A. Klopotek, S. T. Wierzchon, K. Trojanowski (Eds), *Intelligent Information Processing and Web Mining*, Proceedings of the International IIS: IIPWM'03 Conference, pp. 131-138, Zakopane, Poland, Springer, 2003.

[5] S. Diaconescu. Natural Language Agreement Description for Reversible Grammars. In: T. D. Gedeon, L. C. C. Fung (Eds.), *Advances in Artificial Intelligence, 16th Australian Conference on AI*, pp. 161-172, Perth, Australia, Springer, 2003.

[6] S. Diaconescu. Natural Language Processing Using Generative Indirect Dependency Grammar. In: M. A. Klopotek, S. T. Wierzchon, K. Trojanowski (Eds), *Intelligent Information Processing and Web Mining*, Proceedings of the International IIS, IIPWM'04 Conference, pp. 414-418, Zakopane, Poland, Springer, 2004.

[7] S. Diaconescu. Multiword Expression Translation Using Generative Dependency Grammar. In: J. L. Vicedo, P. Martinez-Barco, R. Muñoz, M. S. Noeda (Eds.), *Advances in Natural Language Processing*, Proceedings of 4th International Conference, ESTAL 2004, pp. 243-254, Alicante, Spain, Springer, 2004.

[8] S. Diaconescu. GRAALAN – Grammar Abstract Language Basics. In: J. M. Jun, B. M. Bae, K. Y. Lee (Eds) *GESTS International Transaction on Computer Science and Engineering*, Vol.10, No.1: Sunjin Publishing Co., 2005.

[9] S. Diaconescu. Some Properties of the Attribute Value Trees Used for Linguistic Knowledge Representation. In: *2nd Indian International Conference on Artificial Intelligence (IICAI-05)*, Pune, INDIA, 2005.

[10] S. Diaconescu. Creation of the linguistic resources using a specialised language. (Crearea resurselor lingvistice cu ajutorul unui limbaj specializat), In C. Forăscu, D. Tufiş, D. Cristea (Eds.), *Workshop on Linguistic resources and tools for Romanian Language Processing*, pp. 39-44, Iassi, Romania, Editura Universităţii A. I. Cuza, 2006.

[11] S. Diaconescu. Complex Natural Language Processing System Architecture. In: Corneliu Burileanu, Horia-Nicolai Teodorescu (Eds.), *Advances in Spoken Language Technology*, pp. 228-240, Bucharest, Romania: The Publishing House of the Romanian Academy, 2007.

[12] EAGLES Formalism Working Group Final Report, Version of september 1996.

[13] IPA International Phonetic Association. Handbook of the International Phonetic Association, A Guide to the Use of the International Phonetic Alphabet. Cambridge, UK: Cambridge University Press, 2005.

[14] ISO/IEC 10646. Information technology -- Universal Multiple-Octet Coded Character Set (UCS). Geneva, International Organization for Standardization, 1992.

[15] ISO 639 (E). Code for the representation of names of languages. Geneva, International Organization for Standardization, 1998.

[16] A. Joshi, L. Levi, L. Takabashi. Tree Adjunct Grammars. *Journal of the Computer and System Sciences*, 1975.

[17] C. Pollard, I. Sag. Head-Driven Phrase Structure Grammar. Stanford: CSLI & Chicago: U Chicago Press, 1994.

[18] S. Kahane. Grammaire d'Unification Sens-Texte. Vers un modèle mathématique articulé de la langue. Document de synthèse, Paris, France: Univ. Paris 7, 2002.

[19] R. Kaplan, J. Bresnan. Lexical Functional Grammar. A Formal System for Grammatical Representation. In: J. Bresnan (ed), *The Mental Representation of Grammatical Relations*: Massachusetts USA: MIT Press, 1982

[20] J. Landsbergen. Isomorphic grammars and their use in the ROSETTA translation system. In: *Machine Translation Today: The State of the Art*, Edinburgh UK: Edinburgh University Press, 1987.

[21] L. Tesnière. Éléments de syntaxe structurelle, Paris France: Klincksieck, 1959.

[22] W3C. Extensible Markup Language (XML) 1.0, Recommendation. 10-Feb-98, pp. 24-25, 1998.