Applying Attribute Grammars to Teach Linguistic Rules*

Patrícia Amorim Barros[1], Maria João Varanda Pereira[2], and Pedro Rangel Henriques[3]

1 **Departamento de Informática/Centro Algoritmi, Universidade do Minho, Braga, Portugal**
bpatrcia@gmail.com
2 **Departamento de Informática e Comunicações, IPB/Centro Algoritmi, Bragança, Portugal**
mjoao@ipb.pt
3 **Departamento de Informática/Centro Algoritmi, Universidade do Minho, Braga, Portugal**
pedrorangelhenriques@gmail.com

Abstract

An attribute grammar is a very well known formalism to describe computer languages but it can also be successfully used to describe linguistic phenomena. Since natural languages can also be expressed in grammars it is natural to describe rules using the same formalism. Linguistic teachers of the University Complutense of Madrid started using attribute grammars but they lack a tool that helps them to specify linguistic rules in a friendly and natural way. Therefore we propose a domain specific language (NLSdsl) carefully designed for non-programmers that will be implemented on an AnTLR based system.

1998 ACM Subject Classification D.3.4 Processors

Keywords and phrases Attribute Grammars, DSL, Linguistics

Digital Object Identifier 10.4230/OASIcs.SLATE.2017.1

1 Introduction

Attribute Grammars were first developed with the intent of describing the semantics of context-free languages by Donald Knuth [5, 6]. The difference between an attribute grammar and a context-free grammar is basically that the attribute grammar provides context sensitivity using attributes and assigning them values,that are calculated using evaluation rules [3].

Even though when they first appeared their main purpose was to specify programming languages and to be used in compiler development [11], currently, attribute grammars have several types of applications due to their adaptability. They can be used to define languages, generate compilers, design and specify algorithms, etc. Linguistics is also an application of attribute grammars.

Attribute grammars may be used to specify the way sentences can be formed in a natural language. There are several rules in every idiom that define the way sentences can be correctly formed (e.g. number agreement between adjectives of one termination and nouns, etc.). Many rules that exist in natural languages can be specified with an attribute grammar. Resorting

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.

to some of the Attribute Grammar features it is possible to verify semantic and syntactic correction of any sentence [4]. Given a sentence, it is possible to determine if it is written correctly, and if not, where the errors occurred.

The idea of using attribute grammars to specify syntax and semantics of natural languages has been somewhat practiced. However the technique requires knowledge on programming to code the evaluation rules of the attribute grammar [3], as will be discussed in Section 2. Previous work falls short in making the tools available to the people who are most interested in using them: linguists, who in most of the cases don't have the knowledge necessary to program.

So, the work described in this paper is focused on a new pedagogical tool for linguistic students. The construction of a friendly interface based on a simple domain specific language allows the students to use attribute grammars without programming background. In order to understand linguistic rules, the students usually construct tree based structures and they use those structures to understand the sentence parts and the relations between those parts. In that sense, if they have a computational version of these specifications the restrictions they specify can be automatically verified. Moreover with an appropriate framework it would be very useful to visualize the tree and the attribute evaluation. Although this, as was said, the specification of attribute grammars in processing language tools is not an easy task for people without programming experience. In this paper, we propose an AnTLR based framework (see description in Section 7) that includes a new friendly interface, a new domain specific language to specify linguistic rules (introduced in Section 5 and illustrated in Section 6), an automatic way to verify the correction of the sentences, a set of useful visualizations that will help the students to understand the computational version of their linguistic rules. For that, linguistic exercises (described in Section 3, and implemented in AnTLR in Section 4) were analysed and an appropriate DSL was created to cope with them. The new DSL is called NLSdsl and an AnTLR based framework translate NLSdsl programs into AnTLR specifications. After that other facilities will be implemented: friendly interface and generated visualizations. For this work, several case studies drawn by teachers from University Complutense de Madrid and University of Minho were used. At the end, groups of linguistics students of these institutions will test the proposed approach and tool.

2 Related work

This section focus on describing and explaining tools and approaches that already use attribute grammars in teaching contexts.

Context free grammars (CFG) have been proposed and used by linguists since Chomsky's proposal to fundamentally describe the syntax of natural languages and there are tools to construct and visualize syntactic trees [2]. But CFGs are not used for syntactic or syntactic-semantic analysis because of the complexity of specifying the constraints of natural languages (like concordances) and semantic calculations. For this reason, the computational linguists need grammars that incorporate attributes to the terminal and non-terminal symbols and use some mechanism for checking constraints. Use a more complete formalism based on attribute grammars has proved to be much more effective from the didactic point of view than starting from more traditional formalisms of Linguistics [10, 13]. So, attribute grammars come from the language processing field and are not known or used by linguists despite their effectiveness in representing and conducting analysis guided by syntactic-driven analysis. Therefore, there are no effective tools that allow linguists to write and test grammars at a sufficiently high level of abstraction.

PAG is the tool currently being used by the linguistic students at Universidade Complutense de Madrid. PAG has the same objective as this work: make attribute grammars easier for linguists to specify. To achieve that goal they created a *Prolog* based framework in which linguists can specify their grammars in a specific language, directly execute their specifications and see the results in a decorated parse tree [14].

Linguistic students have deep knowledge of natural language so they produce good formal specifications but have serious difficulties in translating their knowledge to computer models, since they lack Computer Science skills. Hence the need to create a tool that facilitates this process. To use PAG, in addiction to writing the specification for the language, the user has to type some information in a user interface: the sentence he wants to analyze (it can be uploaded or written) and the values of the inherited attributes. Then this information is processed and the program makes all the decorated parse trees possible (there may be more than one) and notifies the user of all the errors (if any) that occurred. Students find the decorated parse trees very useful to understand ambiguity of sentences, since it allows them to see a table that shows attribute values for each node. Also, each entry of this table links to the corresponding semantic equation used to calculate the attribute. Notwithstanding that this tool solves some of the problems this group of students and teachers faced, we feel that we can improve this solution by making the specification of the rules even simpler and a much user friendlier appealing user interface including animated visualizations.

Outside the field of linguistics, three tools (that will be presented in the next paragraphs) were developed all of them aiming at simplify and help the attribute grammars teaching-learning process.

EvDebugger is a software tool created on 2014 based on attribute grammars for language specification with the purpose of helping Computer Science students with compiler construction [12]. Students usually need to design and develop language processors as their final project and they usually face difficulties defining the correct semantic equations. This happens because they have a hard time understanding the dependencies among attributes, in particular their evaluation order. *EvDebugger* helps them by providing a visual debugger that displays a syntax tree view and a table with the values of the attributes being computed. In addition to the debugger, this tool also offers a Grammar Manager and a Grammar Editor to facilitate the proccess of creating an attribute grammar to the students.

Hafix et al. [3] proposed a useful system to modelling natural-language phenomena which allows language processors to be created as executable attribute grammars. This is a system build in a purely-functional language (Haskell) where developers can specify semantic and syntactic descriptions of natural languages using attribute grammars. This is achieved by using a top-down analysis method that allows the production of a compact representation for ambiguous parses and the computation of meanings of sentences, using semantics. The language in which the specifications can be written is really simple and similar to AnTLR.

Another tool, called *VisualLISA*, allows attribute grammars to be written in a visual way (dragging and dropping shapes). Since it makes attribute grammars simpler to specify can also be used by linguistic students but in a visual manner, requiring less mental effort. *VisualLISA*[1] (*A Visual Programming Environment for Attribute Grammars*) was created in 2010 by Pedro Oliveira [9]. Users create drawings to specify their attribute grammars and *VisualLISA* allows them to verify at any time the structural and semantic correctness of the specification. The language of *VisualLISA* consists on a set of icons that can be conjugated. Nevertheless, *VisualLISA* doesn't fit properly enough the needs of our target users. Linguists

[1] `http://www4.di.uminho.pt/~gepl/VisualLISA/`

are not familiar with terms like terminal, non-terminal, computation rule, etc., therefore they may not be able to understand the meaning of all the icons used to design the attribute grammar.

To close the section, we will reference another tool, *CONSTRUCTOR* [1], that uses attribute grammars to help construct geometrical figures using natural language because it demonstrates how attribute grammars can be used in different fields. This tool allows users to input instructions for the construction of geometrical figures (in natural language) and using attribute grammars transforms those directives into actual geometric figures.

All of the tools demonstrate how attribute grammars can be used to help specify every kind of things: since geometrical figures to linguistic phenomena. Furthermore they demonstrate ways to make the specification of attribute grammars simpler.

3 Case Studies, description

In order to get familiar with the type of problems that linguists usually deal and need to solve and specify, some examples of exercises were obtained from the computational linguistic courses at UCM and also at UM. In concrete, the case studies 1,2 and 3 are based in the final project of the students Petra Horáková & Juan Pedro Cabanilles Gomar, Laura Canedo Caravaca & Lara de Santos Tabares and Alfredo Polves Luelmo of the Degree in Linguistics and Applied Languages of UCM. Those examples were carefully analysed and the solution was written in AnTLR. In this section we can find the explanation of these exercises.

The first case study to be implemented consisted in the syntactic analysis of prepositional phrases and their components: the preposition and the noun phrase. The grammar should be able to verify number and gender concordance between the components of the noun phrase (determinant, noun and adjective). In order to make this verification it was necessary to specify two attributes for the words to be used in the input sentences: number and gender. For example, the Portuguese sentence “O amiga dos irmã” would throw an error because it has multiple concordance errors, both in gender and number: The determinant “O” is masculine and singular, so it requires the subject to have the same attributes. That doesn’t happen because the noun “amiga” is feminine. The same happens with the determinant “dos” which is masculine and plural and doesn’t agree with the noun “irmã,” that is feminine and singular. For better understanding, the AnTLR code for this example will be presented and explained in the next section.

The second case study was related to the German language: in German each noun, pronoun and article has four cases: nominative, genitive, dative and accusative. These cases affect the way a sentence must be structured, and the types of verbs (movement or position) that can be used in it. This exercise required us to verify if a sentence was correctly structured taking into account if the case of the sentence agreed with the strucutre and with the type of the verb. For example, the German sentence “Er legt der Teller auf den Tisch” is wrong while the senctence “Er legt den Teller auf den Tisch” is right even though they both translate to the same result: “He puts the plate on the table.” The reason why the first sentence is incorrect is because the article “den” must be used only with accusative sentences and this sentence is nominative.

The third case study consisted in a syllabic divider: starting from a word already divided in syllables the exercise was to determine if the division was well made, and if not, determine why. For example, the input phrase “v-en-to” is not correctly divided by syllables and the correct division would be “ven-to.” In order to verify this it is necessary to analyze the type (vogal and consoant) and subtype (strong, weak. . .) of every letter of the word, taking also into account its position in the word.

The fourth exercise was also related to noun and gender concordance, in this case for Portuguese sentences. In Portuguese, the noun determines the number and gender of the determinant and adjectives that are related to it. This kind of concordance in Portuguese can be hard to verify, specially when the components that need to concur are far apart in the sentences or when we need to deal with anaphora. For example, the sentence "Um especialista em fibras óticas ótimo" and the sentence "Um especialista em fibras óticas ótima"" are both equally correct. The challenge was to figure out with which one of the nouns the adjective must agree and verify the correctness of the sentence.

The fifth case study consisted in more semantic aspects of the Portuguese language: some verbs require nouns and complements of a certain type (animated or in-animated). This exercise required that we verified if the sentences respected those kind of constraints. For example, the sentence "O Carlos assusta a sinceridade" is incorrect because the verb "assustar" requires an animated complement, and it is not the case of the noun "sinceridade," while the sentence "A sinceridade assusta o Carlos" is correct, because "Carlos" is an animated noun. For better understanding, the AnTLR code for this example will be presented and explained in the next section.

The sixth and last case study was related to a Portuguese phenomenon called anaphora. An example of an anaphora can be found in the following sentence: "Eu gosto de **me** lavar com água quente." In this sentence, the word "de" is an anaphora. Anaphora must be connected to a local antecedent that is the subject of the sentence, in this case, "Eu." That means that for example the sentence "Eu gosto de **se** lavar com água quente" is incorrect, because the anaphora "se" is used with third person subjects, and "Eu" is in the first person.

The exercise requires us to verify if the anaphora in a sentence is correctly formed which means that agrees in noun and gender with the subject.

4 Case Studies, AnTLR implementation

For lack of space, in this section we only describe the resolution of the first and fifth case studies (introduced in the previous section) using AnTLR notation and technology.

Case Study 1

The first example consists in analysing the syntax of *prepositional phrases* and their components: the `preposition` and the `noun` phrase. The grammar should be able to verify *number* and *gender* concordance between the components of the noun phrase (determinant, noun and adjective).

In order to make this verification it was necessary to specify two attributes (exemplified in the listing below) for the *words* (the lexicon members) that can be used in the input sentences: `number` and `gender`.

```
adjective returns[String gender, String number]
   : 'pequeno'  { $gender = "m"; $number = "s"; }
   | 'pequena'  { $gender = "f"; $number = "s"; }
   | 'pequenos' { $gender = "m"; $number = "p"; }
   | 'pequenas' { $gender = "f"; $number = "p"; }
   | 'maior'    { $gender = "i"; $number = "s"; }
   ;
```

Adding semantic actions (specially conditional statements) to the syntactic rules of the AnTLR context free grammar, it is possible to declare that the `gender` and `number` of the *determinant* and the *adjective* must concur with the `number` and `gender` of the *noun*.

```
sn : detPos noun prepositionalPhrase?
{
  if($detPos.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detPos.text
     +","+$noun.text+")"); } }
| detPos noun adjective prepositionalPhrase?
{
  if($detPos.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detPos.text
     +","+$noun.text+")"); }
  if($noun.number != $adjective.number) { System.out.println("ERROR No
      number agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); }
  if(($noun.gender != $adjective.gender) && ($noun.gender != "i") && (
     $adjective.gender != "i")) { System.out.println("ERROR No gender
     agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); } }
| detArt noun prepositionalPhrase?
{
  if($detArt.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+")"); }
  if(($detArt.gender != $noun.gender)  && ($detArt.gender != "i") && (
     $noun.gender != "i")) { System.out.println("ERROR No gender
     agreement between determinant and noun! ("+$detArt.text+","+$noun
     .text+")"); } }
| detArt noun adjective prepositionalPhrase?
{
  if($detArt.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+") "); }
  if($detArt.gender != $noun.gender) { System.out.println("ERROR No
     gender agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+") "); }
  if(($noun.number != $adjective.number)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No number
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+") "); }
  if(($noun.gender != $adjective.gender)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No gender
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+") "); } }
| noun prepositionalPhrase?
| noun adjective prepositionalPhrase?
{
  if($noun.number != $adjective.number) { System.out.println("ERROR No
      number agreement between noun and adjectiveetivo! ("+$noun.text
     +","+$adjective.text+")"); }
  if(($noun.gender != $adjective.gender)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No gender
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); }
}
;
```

Case Study 5

The second example consists in analysing the *type of subjects and complements that each verb accepts* in a sentence. The grammar should be able to verify if the subject and complement of a sentence agree with the verb, in a semantic way.

In order to make this verification it was necessary to specify only one attribute: the `type`. The next listing shows the AnTLR productions that assign an initial value the `type` attribute of the *verbs* and *nouns* (lexicon elements) that can be used in the input sentences.

```
verb returns[String type]
        : 'teme'     { $type = "subjAnimated";   }
        | 'assusta'  { $type = "complAnimated";  }
        | 'sucedeu'  { $type = "subjInanimated"; }
        | 'durou'    { $type = "complTemporal";  }
        ;

noun returns [String type]
        : 'Maria'       { $type = "animated";   }
        | 'sinceridade' { $type = "inanimated"; }
        | 'Carlos'      { $type = "animated";   }
        | 'homem'       { $type = "animated";   }
        | 'acidente'    { $type = "inanimated"; }
        | 'animal'      { $type = "animated";   }
        | 'reuniao'     { $type = "inanimated"; }
        | 'horas'       { $type = "temporal";   }
        ;
```

Now the approach is similar to the one followed to solve the previous case study. Again, adding semantic actions (specially conditional statements) to the syntactic rules of the AnTLR context free grammar, it is possible to declare that the `type` of the *subject* or *complement (direct or indirect)* must agree with the `type` of the *verb*: for example, if the verb type is `subjAnimated` it means that the subject of every sentence in which that verb appears must be of type '`animated`'.

```
predicate returns [String typeVerb, String typeComplDir, String
    typeComplInd,String verbTxt, String complementDirTxt, String
    complementIndTxt]
  : verb complementDir {
    $typeVerb          = $verb.type;
    $typeComplDir      = $complementDir.type;
    $verbTxt           = $verb.text;
    $complementDirTxt = $complementDir.text; }
  | verb complementInd {
    $typeVerb          = $verb.type;
    $typeComplInd      = $complementInd.type;
    $verbTxt           = $verb.text;
    $complementIndTxt = $complementInd.text; }
  | verb complementDir complementInd {
    $typeVerb          = $verb.type;
    $typeComplDir      = $complementDir.type;
    $typeComplInd      = $complementInd.type;
    $verbTxt           = $verb.text;
    $complementDirTxt = $complementDir.text;
    $complementIndTxt = $complementInd.text; }
  ;
```

```
sentence: subject predicate '.' {
  switch($predicate.typeVerb) {
    case("subjAnimated") :
      if($subject.type == "inanimated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an animated subject !");
      break;
    case ("complAnimated"):
      if($predicate.typeComplDir == "inanimated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an animated complement !");
      break;
    case ("subjInanimated"):
      if($subject.type == "animated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an inanimated subject!");
      break;
    case ("complTemporal"):
      if(($predicate.typeComplDir != "complTemporal") && ($predicate.
          typeComplDir != null))
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires a temporal complement!");
      if(($predicate.typeComplInd != "complTemporal") && ($predicate.
          typeComplInd != null))
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires a temporal complement!");
      break;
  }
};
```

5 NLSdsl

A Domain Specific Language (DSL) is a language designed to describe a specific domain [7, 8]. In this section a new Domain Specific Language for linguistic rules specification will be introduced. Its main purpose is to turn easier the rules specification avoiding the complexity of the AnTLR code for non-programmer users.

The first step needed to specify our Domain Specific Language was to define what would be the main characteristics of the language:

- It needed to be intuitive for non-programmers;
- It needed to be as close as possible to natural language;
- It couldn't have many programming elements (such as keys, semicolons, etc.);
- It should follow the functional notation instead of the object oriented (it is more intuitive for non-programmers to understand *function(argument)* than *function.argument*).

Taking these items into account it was possible to make the first sketch of the new DSL, NLSdsl, defined and explained below – our proposal to leverage the use of attribute grammars by Linguists.

```
grammar NLSdsl;
   specification : (grammarRule)+;
```

The first rule of the DSL says that the specification should be formed by at least one grammar rule. Then, a grammar rule defines the components of a sentence (one or more symbols), a possible set of calculations (`evalRule`) and a possible set of conditions (ifthen statements).

```
grammarRule: ntSymb':'(symbol('?')?)+ (evalRule)* (condition)*;
```

The symbols can be terminals or non-terminals in the sense that can be used to define other rules or not. A terminal symbol must be written using uppercase letters.

```
symbol : ntSymb | tSymb;
ntSymb : PAL;
tSymb  : PALCAPIT;
```

`evalRule` is where the value of the attributes is evaluated, when necessary.

```
evalRule : '->' attName '=' expr ('&' attName '=' expr)*;
```

The evaluation rule must start with the character '->' and must be formed by an attribute name followed by the '=' sign and an expression that determines the value the attribute will take. One or more attributes can be defined. The expression for now may only be a word, a number or an attribute, because in all the examples we tested there was no need for more complex expressions.

```
expr  : attrib | PAL | NUM;
```

An attribute is the way to obtain the value of a symbol's attribute. It is formed by the attribute name followed by the symbol between brackets.

```
attrib : attName '(' symbol ')';
```

Returning to the first rule of the grammar, the conditions refer to rules that must be verified and if they turn out to be false an error message must be printed.

```
condition : '=>' logicExp errorMsg;
```

A condition must start with the character '=>' and then have a logic expression and an error message. The error message should only be printed if the logic expression evaluates false.

A logic expression is composed of one or more relations connected by Boolean operators.

```
logicExp : rel (opBOOL rel)*;
opBOOL   : 'AND' | 'OR';
rel      : attrib opREL expr;
```

A relation is formed by an attribute followed by a relational operator and an expression. The error message must be written between two exclamation marks and can contain various elements (maybe strings, attribute names, or symbol names) aggregated by the concatenation operator '+'.

```
errorMsg : '!' elem ('+' elem)* '!';
elem     : STR | attrib | symbol;
```

This is useful to print attribute values or symbols embedded in the error message to make clearer what is wrong in the input sentence.

6 Revisiting the Case Studies of Section 4 with NLSdsl

For a better understanding of the practical application of the DSL we proposed in the previous section, we revisit in this section the two case studies discussed in Section 4, and show excerpts of what their specification in NLSdsl would look like. We also show the textual output that our tool (see Section 7) would produce when executed with an input sentence, and the decorated parse tree it would generate.

Case Study 1

Recalling that the first case study presented is concerned with *noun and gender concordance* in a prepositional sentence, part of the NLSdsl specification for this linguistic phenomenon is shown in the listing below.

```
prepositionalPhrases : (prepositionalPhrase '.')*

prepositionalPhrase : Preposition nominalPhrase
prepositionalPhrase : Contraction nominalPhrase

nominalPhrase : noun prepositionalPhrase?;
nominalPhrase : noun adjective prepositionalPhrase
  => number(noun) == number(adjective)
      ! ERROR, no number agreement between noun a adjective !

  => gender(noun) == gender(adjective)
      ! ERROR, no gender agreement between noun e adjective !

nominalPhrase : positionDeterminant noun prepositionalPhrase?
  => number(positionDeterminant) == number(noun)
      ! ERROR, no number agreement between noun e determinant !

nominalPhrase : positionDeterminant noun adjective
    prepositionalPhrase?
  => number(positionDeterminant) == number(noun)
          ! ERROR, no number agreement between noun e determinant !

  => number(noun) == number(adjective)
          ! ERROR, no number agreement between noun a adjective !

  => gender(noun) == gender(adjective)
          ! ERROR, no gender agreement between noun e adjective !

adjective: PEQUENO  -> gender = masculine & number = singular
adjective: PEQUENA  -> gender = feminine  & number = singular
adjective: PEQUENOS -> gender = masculine & number = plural
adjective: PEQUENAS -> gender = feminine  & number = plural
adjective: MAIOR    -> gender = undefined & number = singular
```

The output of our tool when invoked with the input `"Para teu irmãs pequeno."` (sentence one) would be the following:

```
ERROR No number agreement between determinant and noun!(teu,irmas)
ERROR No number agreement between noun and adjective!(irmas,pequeno)
ERROR No gender agreement between noun and adjective!(irmas,pequeno)
```

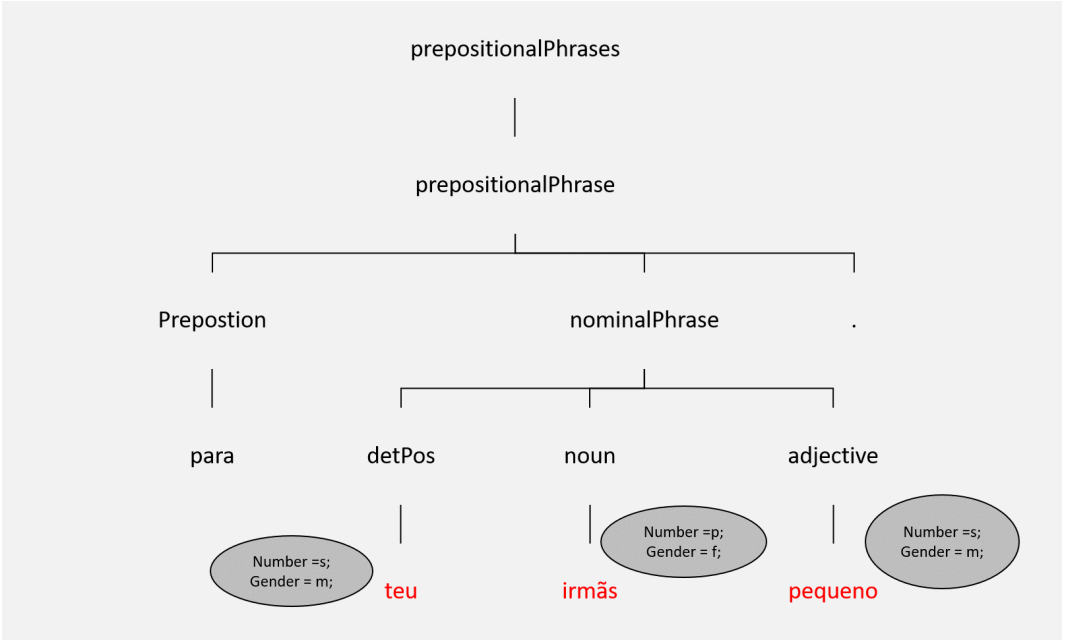

Figure 1 Attributed Parse Tree generated by our tool for case study one, sentence one.

And the parsing tree that complementary would be generated by our tool, decorated with attribute values in each node and colored (in red) to enhance the nodes where errors where detected, is shown in Figure 1.

Case Study 5

As previously said, the fifth case study is related to *the type of verbs and the type of complements and subjects they require*. So, part of the NLSdsl specification for this linguistic phenomenon is shown in the listing below.

```
sentence : subject predicate '.'
 => typeVerb(predicate) == "subjAnimated" && type(subject) == "
    animated"
        ! ERROR, The verb requires an animated subject !

 => typeVerb(predicate) == "complAnimated" && typeComplDir(predicate)
    == "animated"
        ! ERROR, The verb requires an animated complement !

 => typeVerb(predicate) == "subjInanimated" && type(subject) == "
    inanimated"
        ! ERROR, The verb requires an inanimated subject !

 => typeVerb(predicate) == "complTemporal" && typeComplDir(predicate)
    == "temporal"
        ! ERROR, The verb requires a temporal complement !

 => typeVerb(predicate) == "complTemporal" && typeComplInd(predicate)
    == "temporal"
        ! ERROR, The verb requires a temporal complement !
```

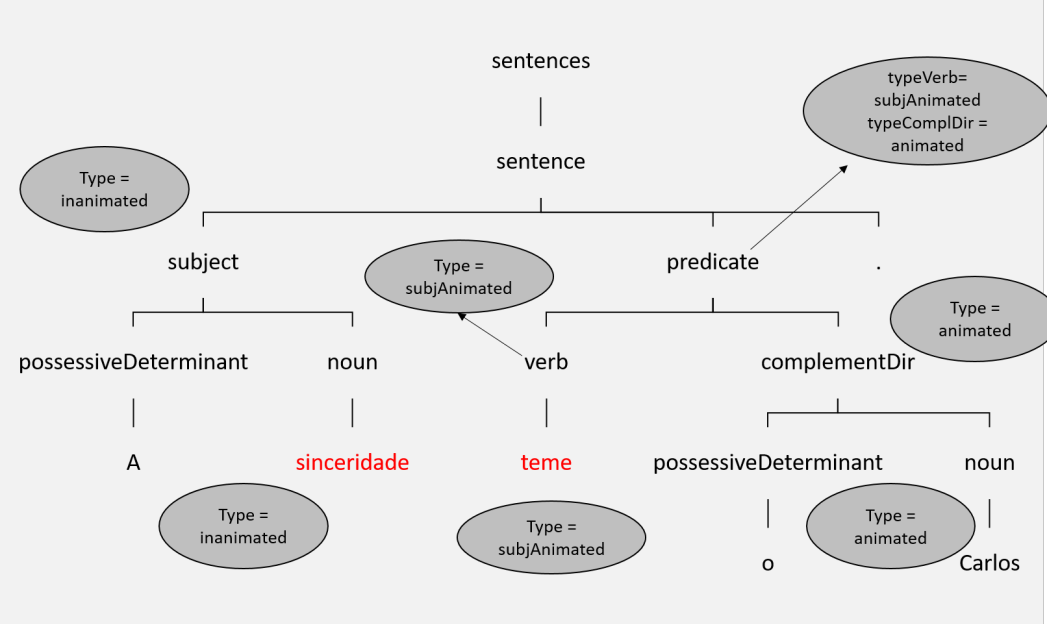

Figure 2 Attributed Parse tree generated by our tool for case study five, sentence two.

The output of our tool when invoked with the input `A sinceridade teme o Carlos.` (sentence two) would be the following:

```
ERROR! The verb 'teme' requires an animated subject !
```

The parsing tree generated, with attribute values decorating its nodes and red color to enhance the nodes where errors where detected, is presented in Figure 2.

As we can see in Figure 2, there is an error in the sentence because the word `sinceridade` does not comply with the requirement of the verb `teme` of being of type `animated`.

7 Our tool: description and architecture

This section presents our tool, based on the description of the system architecture. Figure 3 represents the architecture of our system.

The central component is a translator that takes a Domain Specific Language specification and transforms it into the equivalent AnTLR specification.

This translator is generated by AnTLR using the NLSdsl grammar (described in Section 5). This process is straight-forward and does not deserve a more detailed explanation here. Actually, the core of this translation process is the set of rules that map NLSdsl constructors into the AnTLR attributed productions, but this contribution (under work) raise up directly from the handwritten examples (like the ones presented in Section 4).

After that process (transformation of an NLSdsl specification into the AnTLR grammar) is completed, a compiler is automatically constructed by AnTLR to cope with sentences written in L Language. Then the user can input a sentence to test and this new compiler produces the sentence verification assessment (an Ok or an error message) and a tree based visualization to show, more clearly, the structure of the input sentence locating and enhancing the error occurrences in the appropriate tree nodes. Some examples were presented in Section 6.

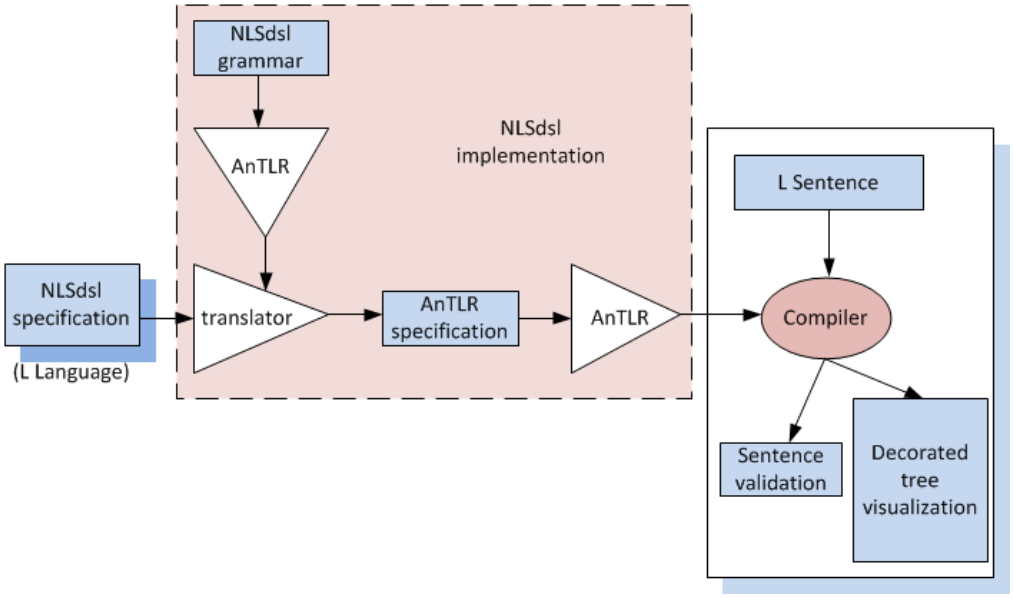

Figure 3 System Architecture.

The use of the NLSdsl to describe a linguistic rule and the subsequent validation process of concrete sentences allows the linguistics' students to better understand those rules and the errors that occur when they are not obeyed. In our opinion the decorated tree visualization (also under development) will be crucial to aid in the understanding of those rules and their verification process.

8 Conclusion

Even though the objective of the master's thesis project, underlying the work here discussed, is to develop the entire system described in this document, in this paper we focused on the three contributes below:

- the specification in AnTLR of several natural language case studies in order to understand how these linguistic phenomena can be formally described, as explained in Section 4;
- the creation of NLSdsllanguage specifically designed to be intuitive for non-programmers but still powerful enough to express the linguistic phenomena required, thoroughly explained in Section 5;
- the proposal of a system architecture, described in Section 7, that aggregates all of the components to develop.

As future work the development of the NLSdsl to AnTLR translator will be finished and tested for sentence validation.Then, after implementing the visualization module, a set of experiments in classroom will be conducted in order to assess the usability and efectiveness of new tool. These experiments will take place at the beginning of the next school year at UCM.

Acknowledgements. We are deeply indebt to Ana Fernandez-Pampillon and Jose Luis Sierra, from Universidade Complutense de Madrid, for introducing us to this topic, challenging us to cooperate in the linguistic project, and also for all for the discussion sustained and all the material/examples provided.

References

1 Zoltán Alexin, József Dombi, Károly Fábricz, and Tibor Gyimóthy. CONSTRUCTOR: A natural language interface based on attribute grammars. *Acta Cybernetica*, 9(3):247–255, 1990.

2 Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

3 Rahmatullah Hafiz and Richard A. Frost. Modular natural language processing using declarative attribute grammars. In Ildar Batyrshin and Grigori Sidorov, editors, *Advances in Artificial Intelligence: MICAI 2011*, pages 291–304. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-25324-9_25.

4 Petra Horáková and Juan Pedro Cabanilles Gomar. La concordancia nominal de género en las oraciones atributivas del español: una descripción formal con gramáticas de atributos. *Entrepalavras*, 4(1):118–136, 2014.

5 Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

6 Donald E. Knuth. The genesis of attribute grammars. In *Proceedings of the international conference on Attribute grammars and their applications*, pages 1–12. Springer-Verlag, 1990.

7 Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91, 2016.

8 Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

9 Nuno Oliveira, Maria João Varanda Pereira, Pedro Rangel Henriques, Daniela da Cruz, and Bastian Cramer. VisualLISA: A visual environment to develop attribute grammars. *Computer Science an Information Systems (Special issue on Advances in Languages, Related Technologies and Applications)*, 7(2):266–289, May 2010.

10 Fernando. Pereira and David H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

11 Daniel Ravan. *A graphical structure-editor that generates code for attribute grammar systems*. PhD thesis, University of Windsor, 1995.

12 Daniel Rodríguez-Cerezo, Pedro Rangel Henriques, and José-Luis Sierra. Attribute grammars made easier: EvDebugger a visual debugger for attribute grammars. In *International Symposium on Computers in Education (SIIE)*, pages 23–28. IEEE, 2014.

13 Stuart Merrill Shieber. *An introduction to unification-based approaches to grammar*. CSLI Publications, Stanford, California, 1986.

14 José-Luis Sierra, Ana María Fernández-Pampillon, and Alfredo Fernández-Valmayor. An environment for supporting active learning in courses on language processing. *SIGCSE Bull.*, 40(3):128–132, June 2008.