The Ellogon Pattern Engine: Context-free Grammars over Annotations

Georgios Petasis
Software and Knowledge Engineering Laboratory
Institute of Informatics and Telecommunications
National Centre for Scientific Research (N.C.S.R.) “Demokritos”
GR-153 10, P.O. BOX 60228, Aghia Paraskevi, Athens, Greece
petasis@iit.demokritos.gr

Abstract

This paper presents the pattern engine that is offered by the Ellogon language engineering platform. This pattern engine allows the application of context-free grammars over annotations, which are metadata generated during the processing of documents by natural language tools. In addition, grammar development is aided by a graphical grammar editor, giving grammar authors the capability to test and debug grammars.

Keywords: context-free grammar, grammar over annotations, LARL parser

1. Introduction

Developing pipelines of tools for many tasks of natural language processing, such as named-entity recognition or sentiment analysis, has never been an easy task. One research direction that tried to alleviate this difficulty, was research on infrastructures/platforms for natural language engineering: Within this context of research several platforms have been developed and made publicly available, usually distributed with open source libraries. Indeed, platforms like GATE[1] (Cunningham et al., 2013), Ellogon[2] (Petasis et al., 2002) or Apache UIMA[3] have made the development of natural language processing pipelines easier, by providing suitable infrastructure for many common tasks, including corpora management, models for the linguistic metadata, a modular environment where components interoperate, metadata visualisation tools, and toolboxes of ready-to-be-used natural language processing tools. The availability of ready-to-use tools, such as ANNIE (Cunningham et al., 2002), has increased the popularity of these platforms, especially through their use as “black boxes”, that perform tasks such as information extraction.

However, there are situations where the provided by a platform tools and pipelines are not enough: Either the performance is not enough for specific tasks, or the task at hand requires modifications, or even new tools. In such cases, components offered by a platform must be modified, or new components must be written for the platform, which requires developers to be first acquainted with the internal organisation and programming interfaces of the platform, and then to develop the required components. Typically, a component inside a platform operates on the text of the document to be processed, taking into consideration linguistic metadata that has been produced by other processing components, while the component is expected to enrich this linguistic metadata.

On the other hand, there are grammars which are playing an important role in many natural processing tasks, and their use is widespread in a much broader audience, including non-developers, such as linguists. Despite the fact that grammars are usually associated with sequences of characters and strings, it is feasible to apply them on “sequences” of any kind of objects, which may be far more complex than characters. The work presented in this paper tries to combine grammars with component development, by implementing a pattern engine that can apply context-free grammars on linguistic metadata, in the form of annotations of the Ellogon language engineering platform. This pattern engine can apply a grammar on the *annotation graph* of a document, which models the linguistic metadata/information kept for a document within Ellogon. The motivation behind this pattern engine is to allow users to perform transformations of the linguistic metadata without the need to develop a component in languages such as C/C++/Java/Tcl/Python/Perl, and instead construct a grammar that contains rules and actions, in a “neutral” language, that is not tied to a specific programming language. Obviously, the linguistic tasks that can be modelled through such an engine, clearly depend on the expressivity of both the rules, that must be matched on the annotation graph, and the actions, which define the operations that can be performed on the linguistic metadata. Thus, we have opted context-free grammars over regular grammars for the rules, while actions can be augmented with Tcl code, if the provided action types are found insufficient for a specific natural language processing task.

2. Related Work

The idea of developing linguistic processing components with the help of patterns over both text and linguistic annotations in not new. For example, the “Common Pattern Specification Language” (CPSL) (Appelt and Onyshkevych, 1998) has been specified in the late 90’s, in the context of the TIPSTER (Harman, 1992) project. Among the first platforms to offer such an infrastructure was GATE (Cunningham et al., 2013), with its JAPE engine (Cunningham et al., 2000). JAPE, or “Java Annotation Patterns Engine”, creates finite state transducers that operates over both text and linguistic metadata (annotations), based on regular expressions. Being a version of CPSL, it has been used for a wide range of applications, including

[1] http://gate.ac.uk/
[2] http://www.ellogon.org/
[3] http://uima.apache.org/

pattern-matching, semantic extraction, and many other operations in the context of the ANNIE (Cunningham et al., 2002) information extraction system, included in the GATE platform. TextMarker (Klügl et al., 2008; Kluegl et al., 2009) is another rule-based engine, aiming also at information extraction, which has been integrated into the Apache UIMA[4] platform, leading to Apache UIMA Ruta[5] (Rule-based Text Annotation). UIMA Ruta offers an "imperative rule language extended with scripting elements"[6]. A rule is composed of a sequence of rule elements and a rule element essentially consists of four parts: A matching condition, an optional quantifier, a list of conditions and a list of actions. In addition, a workbench is provided, which provides editing support (i.e. syntax checking), rule application debugging, and rule learning through the included "TextRuler" framework, which is able to induce rules and, therefore, enable semi-automatic development of rule-based components.
"Stanford TokensRegex"[7] is a framework included in Stanford CoreNLP, for defining regular expressions over text and tokens, and mapping matched text to semantic objects. Implemented as a set of stages, the TokensRegex pipeline reads extraction rules from a file and applies them sequentially over text and tokens, followed by "composite" rules which are repeatedly applied until no changes are detected. Then, at the last stage, "filtering" rules are applied to to discard any expressions that should not be matched. Unitex[8] is a corpus processing system, based on automata-oriented technology. It offers a pattern engine, allowing regular expression over tokens associated with simple morphological information, such as token shapes. Despite the fact that it is more limited that the aforementioned systems, it still allows regular expressions not only on text, but also on linguistic metadata. Finally, "graph expression" or "GExp"[9], according to its authors, is similar to GATE JAPE, and allows expressions to be applied on graphs. However, not much information is available for this pattern engine, with the exception of a few examples.
One common aspect of all these engines, is their bias towards regular languages: All approaches presented so far employ some form of either regular grammars or Perl-style regular expressions (which are slightly more expressive than regular grammars), and they tend to handle cases that need greater expressiveness with the cascade application of different grammars. The fact that no engine tries to implement a more expressive language, motivated us to explore context-free grammars, in order to evaluate their applicability on the task of component construction, and decide whether a more expressive formalism has more advantages over disadvantages when compared to regular grammar formalisms. Potential advantages include applicability in more complex tasks (that require higher level dependencies than those that can be handled by regular expressions), and more "elegant", "compact" and easier to handle grammars. On the other hand, context free grammars can be less efficient with respect to their application and more ambiguous, leading to multiple parses. However, the Ellogon's Pattern Engine is a fairly recent addition to the Ellogon platform, with limited usage so far, which does not allow us to draw any conclusions over the advantages of using context-free grammars over regular ones. To our knowledge, the only uses of the Ellogon's Pattern Engine are limited to *a*) a shallow syntactic parser that identifies phrases for Greek (included in Ellogon), and *b*) to a commercial sentiment analysis system for Greek (Petasis et al., 2014).

3. Ellogon's Data Model

The data model of Ellogon is based on the data model of the TIPSTER (Harman, 1992) project. Due to this, it shares some basic features with other TIPSTER-based infrastructures, but it also offers a large number of features that differentiate it from such infrastructures. The central element for storing data in Ellogon is the Collection. A collection is a finite set of Documents. An Ellogon document consists of textual data as well as linguistic information about the textual data. This linguistic information is stored in the form of attributes and annotations.
An attribute associates a specific type of information with a typed value. An annotation associates arbitrary information (in the form of attributes) with portions of textual data. Each such portion, named span, consists of two character offsets denoting the start and the end characters of the portion, as measured from the first character of some textual data. Annotations typically consist of four elements (figure 1):

- A numeric identifier. This identifier is unique for every annotation within a document and can be used to unambiguously identify the annotation.
- A type. Annotation types are textual values that are used to classify annotations into categories.
- A set of spans that denote the range of the annotated textual data.
- A set of attributes. These attributes usually encode the necessary linguistic information.

The vast majority of the linguistic metadata about a document is represented through annotations, which associate an arbitrary set of attributes with one or more segments in the document's text.

4. A Pattern Engine for Context-free Grammars

The pattern engine accepts as input a grammar, which is parsed and converted into instructions, that can be executed in the context of the Ellogon platform. There are several steps involved into this conversion process: initially the syntax of the grammar is verified, and subsequently converted into the Backus–Naur Form (BNF) (Wirth, 1977). Then, the BNF grammar is converted either to a Look-Ahead LR (LARL) (DeRemer, 1969) parser, or an Earley

[4] `http://uima.apache.org/`
[5] `http://uima.apache.org/ruta.html`
[6] `http://uima.apache.org/ruta.html`
[7] `http://nlp.stanford.edu/software/tokensregex.shtml`
[8] `http://www-igm.univ-mlv.fr/~unitex/`
[9] `http://code.google.com/p/graph-expression/`

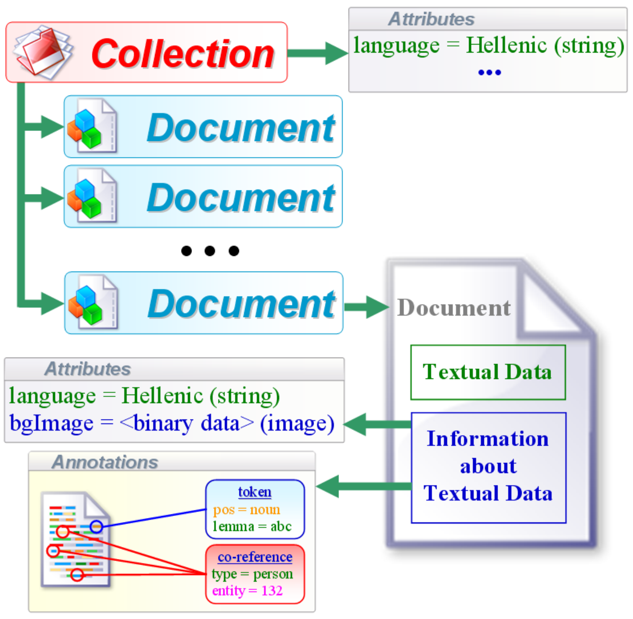


Figure 1: The data model of Ellogon.

parser (Earley, 1970), depending on the grammar. The resulting parser is then serialised into an object, which can later be loaded into Ellogon in order to process documents and their annotations.
The pattern engine will try to generate optimised parsers, and will silently perform some simple optimisations on the input grammar, in order to improve processing efficiency and detect some common problematic cases, which may cause efficiency problems, such as some types of recursion. In addition, it offers a simple development environment, where the grammar author can edit the grammar, see the various intermediate steps (such as the rules in BNF), apply the grammar on text and visualise the results of the matched actions. This development environment, with a simple grammar that adds an attribute to all verbs in a document, is shown in figure 2. Figures 3 and 4 show a slightly more complex grammar, which tries to detect some simple named-entities, and the entities that have been detected by applying this grammar on the sample text on the left part of the editor. Finally, figure 5 shows the grammar after its conversion to BNF. The editor, beyond being an editor and a test environment for a grammar, it additionally provides some simple debugging capabilities, in the sense that the user can apply the grammar "step-by-step", and examine how the generated parser matches each annotation found in the linguistic metadata. Grammars can use and mix any type of annotations, no matter what their level in the processing chain is.

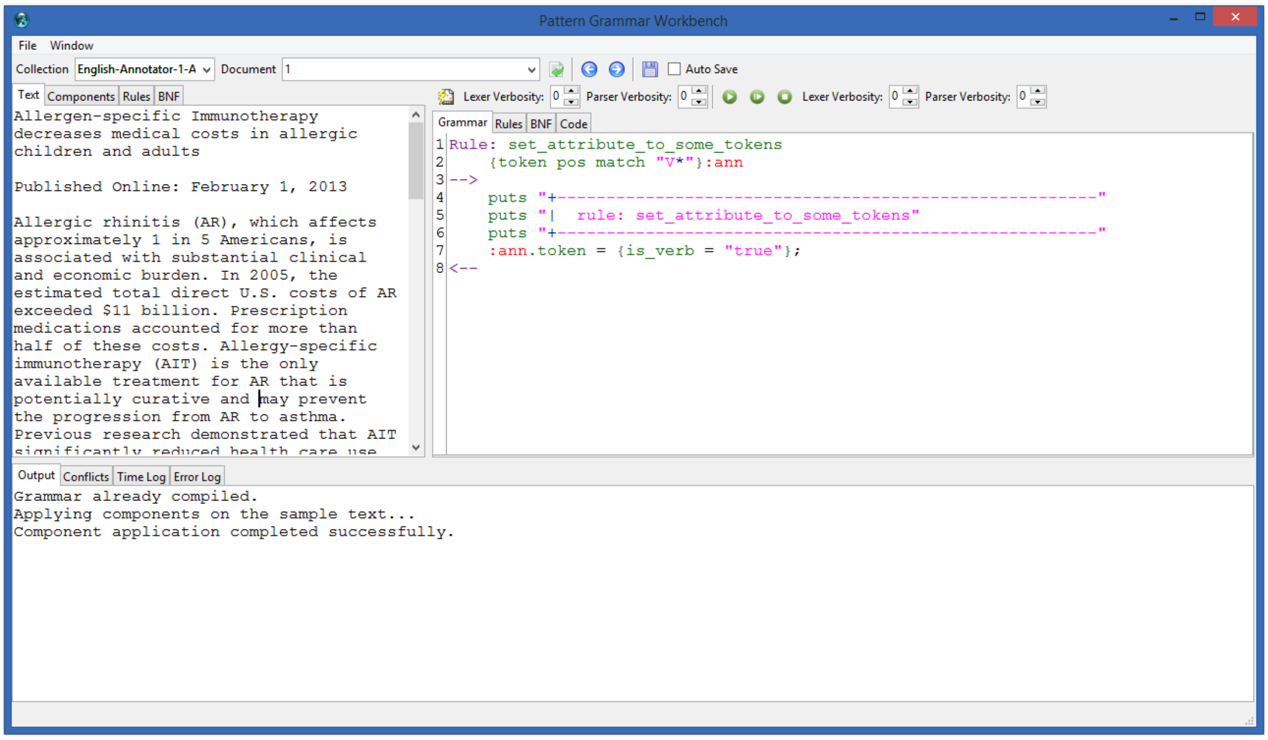
4.1. The Pattern Engine Grammar Language

The grammar of the Ellogon's pattern engine is loosely modelled after CPSL – Common Pattern Specification Language[10], upon which JAPE[11] is also based. JAPE is a similar pattern engine, which implements regular expressions over annotations within the GATE language engineering platform. Since both pattern engines share common ancestry, interoperability may be easier between these two engines than, for example, Apache Ruta, which uses a different grammar formalism. GATE Jape, having a history of more than ten years, provides a significant number of grammars (i.e. within the open source ANNIE system) that can allow the future comparison of the two engines, if these grammars get ported to the Ellogon's pattern engine. However, despite being based on a common ancestor and bearing some similarities, the two grammar languages are not identical, and grammars from one platform are not expected to operate on the other platform without modification. We expect that it will be easier to port grammars from JAPE to Ellogon's pattern engine than doing the opposite, which will be applicable only on Ellogon's grammars that are not context-free.
A grammar consists of a set of rules, each of which consists of a set of patterns (the left-hand-side (LHS) of the rule) and an action (the right-hand-side (RHS) of the rule). When the LHS of a rule is matched over the input, the RHS is executed and modifies the linguistic metadata, usually through the addition of new annotations or new attributes to existing annotations. Actions can refer to various parts of the pattern in the LHS by means of labels that are attached to pattern elements. Consider the following example:

```
Rule: set_attribute_to_some_tokens
    {token pos match "V*"}:ann
-->
    :ann.token = {is_verb = "true"};
<--
```

The LHS is the part preceding the symbol "-->" and the RHS is the part following it. The LHS specifies a pattern to be matched, while the RHS specifies what is to be done to the matched text. In this example, we have a rule entitled "set_attribute_to_some_tokens", which will match annotations of type "token" (generated by the tokeniser and representing words) that have an attribute named "pos" (representing the part of speech of the word), whose value matches the pattern "V*", which simply means that the value of the "pos" attribute starts with the letter "V" (which may well represent all verbs, with a suitable part-of-speech tag set). Once this rule has matched a sequence of input annotations, the entire sequence is tagged with a label by the rule, and in this case, the label is "ann". On the RHS, we refer to this matched input using the label given in the LHS; "ann". This action denotes that an annotation of type "token" that spans the matched input must be created (if it does not already exists of course), and an attribute named "is_verb" with the value "true" must be added. There are several operators that can be used in the LHS, including the operator "|" for specifying alternatives, parentheses for grouping elements, the operators "?", "*", "+" for denoting repetitions (optional, zero or more, one or more respectively), etc. In addition, each rule is associated with a name, specified after the "Rule:" keyword, which essentially is the name of the non-terminal symbol associated with the rule, and allows the embedding of a rule in the LHS of other

[10]http://www.ai.sri.com/~appelt/TextPro/
[11]http://gate.ac.uk/sale/tao/index.html#x1-2120008



Collection: English-Annotator-1-A Document: 1

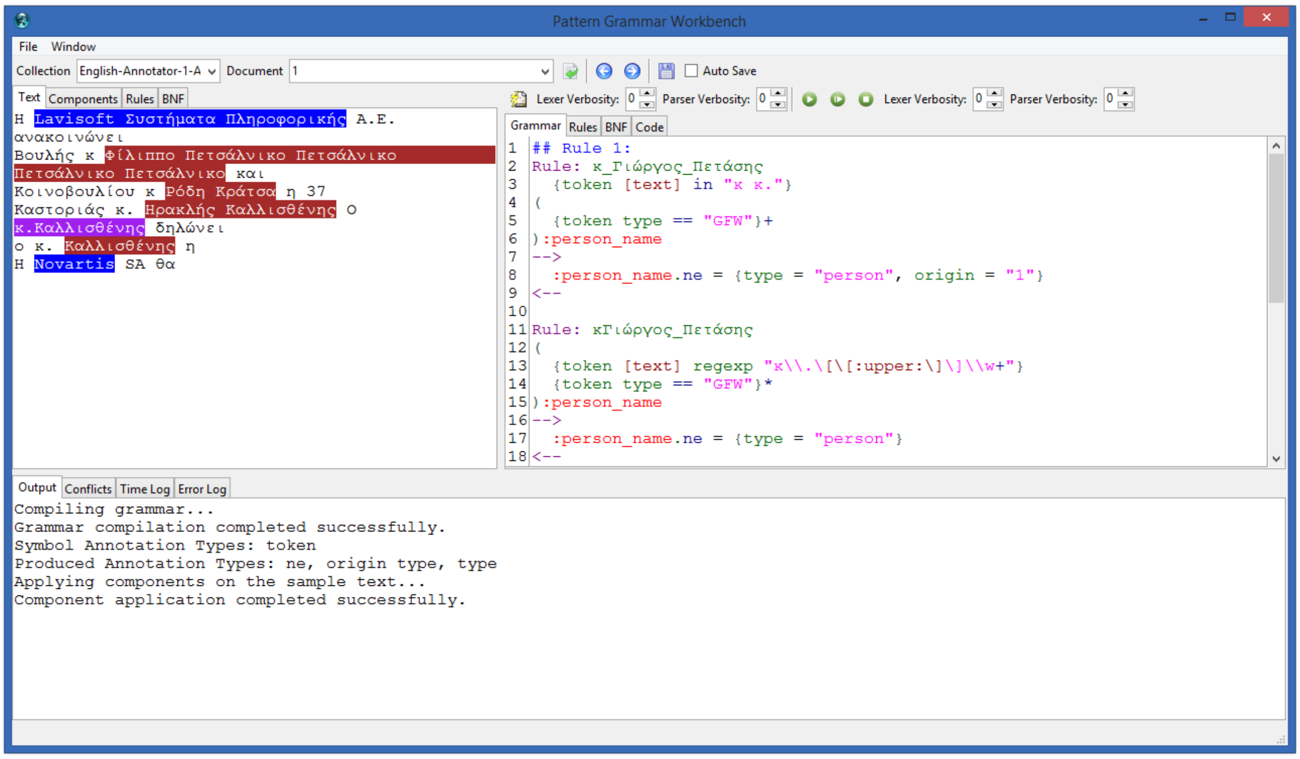
Lexer Verbosity: 0 Parser Verbosity: 0 Lexer Verbosity: 0 Parser Verbosity: 0

```
Allergen-specific Immunotherapy
decreases medical costs in allergic
children and adults

Published Online: February 1, 2013

Allergic rhinitis (AR), which affects
approximately 1 in 5 Americans, is
associated with substantial clinical
and economic burden. In 2005, the
estimated total direct U.S. costs of AR
exceeded $11 billion. Prescription
medications accounted for more than
half of these costs. Allergy-specific
immunotherapy (AIT) is the only
available treatment for AR that is
potentially curative and may prevent
the progression from AR to asthma.
Previous research demonstrated that AIT
significantly reduced health care use
```

```
Rule: set_attribute_to_some_tokens
    {token pos match "V*"}:ann
-->
    puts "+------------------------------------------------------"
    puts "|  rule: set_attribute_to_some_tokens"
    puts "+------------------------------------------------------"
    :ann.token = {is_verb = "true"};
<--
```

```
Grammar already compiled.
Applying components on the sample text...
Component application completed successfully.
```

Figure 2: The development environment of the pattern engine, with a simple grammar loaded.



Collection: English-Annotator-1-A Document: 1

Lexer Verbosity: 0 Parser Verbosity: 0 Lexer Verbosity: 0 Parser Verbosity: 0

```
Η Lavisoft Συστήματα Πληροφορικής Α.Ε.
ανακοινώνει
Βουλής κ Φίλιππο Πετσάλνικο Πετσάλνικο
Πετσάλνικο Πετσάλνικο και
Κοινοβουλίου κ Ρόδη Κράτσα η 37
Καστοριάς κ. Ηρακλής Καλλισθένης Ο
κ.Καλλισθένης δηλώνει
ο κ. Καλλισθένης η
Η Novartis SA θα
```

```
## Rule 1:
Rule: κ_Γιώργος_Πετάσης
  {token [text] in "κ κ."}
(
  {token type == "GFW"}+
):person_name
-->
  :person_name.ne = {type = "person", origin = "1"}
<--

Rule: κΓιώργος_Πετάσης
(
  {token [text] regexp "κ\\.\[\[:upper:\]\]\\w+"}
  {token type == "GFW"}*
):person_name
-->
  :person_name.ne = {type = "person"}
<--
```

```
Compiling grammar...
Grammar compilation completed successfully.
Symbol Annotation Types: token
Produced Annotation Types: ne, origin type, type
Applying components on the sample text...
Component application completed successfully.
```

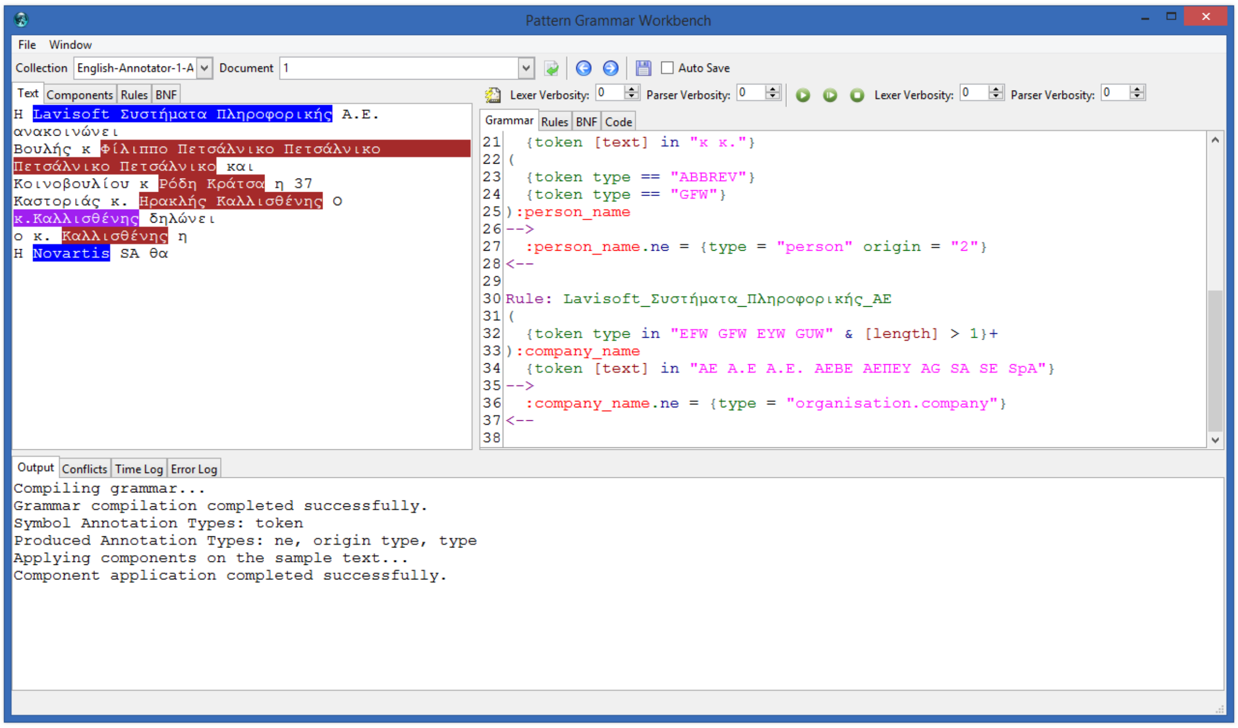
Figure 3: The development environment of the pattern engine, with a slightly more complex grammar loaded.

rules. More information about the grammar language can be found in the Ellogon developers manual[12].

[12]http://www.ellogon.org/index.php/download/documentation

5. Conclusions and Future Work

In this paper we present a pattern engine for applying context-free grammars over annotations, and not strings. Annotations represent linguistic metadata, which is usually the outcome of analysing text with natural language pro-

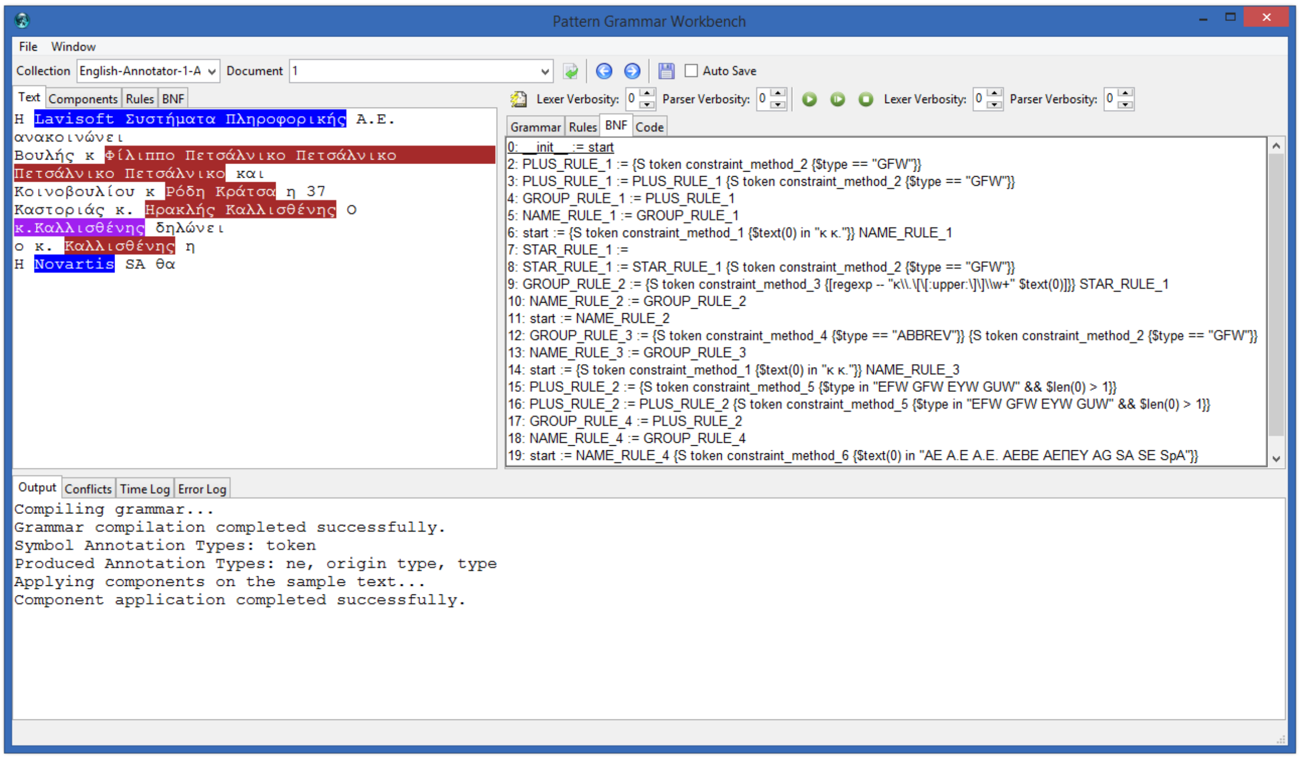


```
Η Lavisoft Συστήματα Πληροφορικής Α.Ε.
ανακοινώνει
Βουλής κ Φίλιππο Πετσάλνικο Πετσάλνικο
Πετσάλνικο Πετσάλνικο και
Κοινοβουλίου κ Ρόδη Κράτσα η 37
Καστοριάς κ. Ηρακλής Καλλισθένης Ο
κ.Καλλισθένης δηλώνει
ο κ. Καλλισθένης η
Η Novartis SA θα
```

```
  {token [text] in "κ κ."}
(
  {token type == "ABBREV"}
  {token type == "GFW"}
):person_name
-->
  :person_name.ne = {type = "person" origin = "2"}
<--

Rule: Lavisoft_Συστήματα_Πληροφορικής_AE
(
  {token type in "EFW GFW EYW GUW" & [length] > 1}+
):company_name
  {token [text] in "AE A.E A.E. AEBE AEΠEY AG SA SE SpA"}
-->
  :company_name.ne = {type = "organisation.company"}
<--
```

```
Compiling grammar...
Grammar compilation completed successfully.
Symbol Annotation Types: token
Produced Annotation Types: ne, origin type, type
Applying components on the sample text...
Component application completed successfully.
```

Figure 4: The development environment of the pattern engine, with a grammar applied on the sample text and the results of the actions visible.



```
Η Lavisoft Συστήματα Πληροφορικής Α.Ε.
ανακοινώνει
Βουλής κ Φίλιππο Πετσάλνικο Πετσάλνικο
Πετσάλνικο Πετσάλνικο και
Κοινοβουλίου κ Ρόδη Κράτσα η 37
Καστοριάς κ. Ηρακλής Καλλισθένης Ο
κ.Καλλισθένης δηλώνει
ο κ. Καλλισθένης η
Η Novartis SA θα
```

```
0: __init__ := start
2: PLUS_RULE_1 := {S token constraint_method_2 {$type == "GFW"}}
3: PLUS_RULE_1 := PLUS_RULE_1 {S token constraint_method_2 {$type == "GFW"}}
4: GROUP_RULE_1 := PLUS_RULE_1
5: NAME_RULE_1 := GROUP_RULE_1
6: start := {S token constraint_method_1 {$text(0) in "κ κ."}} NAME_RULE_1
7: STAR_RULE_1 :=
8: STAR_RULE_1 := STAR_RULE_1 {S token constraint_method_2 {$type == "GFW"}}
9: GROUP_RULE_2 := {S token constraint_method_3 {[regexp -- "κ\\.\[\[:upper:\]\]\\w+" $text(0)]}} STAR_RULE_1
10: NAME_RULE_2 := GROUP_RULE_2
11: start := NAME_RULE_2
12: GROUP_RULE_3 := {S token constraint_method_4 {$type == "ABBREV"}} {S token constraint_method_2 {$type == "GFW"}}
13: NAME_RULE_3 := GROUP_RULE_3
14: start := {S token constraint_method_1 {$text(0) in "κ κ."}} NAME_RULE_3
15: PLUS_RULE_2 := {S token constraint_method_5 {$type in "EFW GFW EYW GUW" && $len(0) > 1}}
16: PLUS_RULE_2 := PLUS_RULE_2 {S token constraint_method_5 {$type in "EFW GFW EYW GUW" && $len(0) > 1}}
17: GROUP_RULE_4 := PLUS_RULE_2
18: NAME_RULE_4 := GROUP_RULE_4
19: start := NAME_RULE_4 {S token constraint_method_6 {$text(0) in "AE A.E A.E. AEBE AEΠEY AG SA SE SpA"}}
```

```
Compiling grammar...
Grammar compilation completed successfully.
Symbol Annotation Types: token
Produced Annotation Types: ne, origin type, type
Applying components on the sample text...
Component application completed successfully.
```

Figure 5: The development environment of the pattern engine, showing the grammar converted into BNF.

cessing tools. The pattern engine is accompanied by a simple graphical grammar editor, which offers simple debugging facilities in order to facilitate grammar development. Having a pattern engine which can modify the linguistic metadata driven by a grammar and according to actions associated to grammar rules, can constitute a valuable tool for modelling a significant set of natural language processing tasks, such as named-entity recognition or sentiment analysis, without the need to develop an new component for the language engineering platform. The described pattern engine is a recent addition to the Ellogon platform, with limited usage so far, which does not allow us to draw any conclusions over the advantages of using context-free grammars over regular ones. As a result, future work will concentrate into writing grammars for this new engine, in order to evaluate the advantages and disadvantages of using a formalism with greater expressivity than regular patterns. Finally, the described pattern engine and the graphical grammar editor are distributed along with the Ellogon language engineering platform, freely available from `http://www.ellogon.org` under the LGPL version 3 open source license.

6. References

Appelt, D. E. and Onyshkevych, B. (1998). The common pattern specification language. In *Proceedings of a Workshop on Held at Baltimore, Maryland: October 13-15, 1998*, TIPSTER '98, pages 23–30, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cunningham, H., Maynard, D., and Tablan, V. (2000). Jape: a java annotation patterns engine. Technical report, University of Sheffield, Department of Computer Science.

Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.

Cunningham, H., Tablan, V., Roberts, A., and Bontcheva, K. (2013). Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS Comput Biol*, 9(2):e1002854, 02.

DeRemer, F. (1969). *Practical Translators for LR(k) Languages*. MAC-TR. Project Mac, Massachusetts Institute of Technology.

Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, February.

Harman, D. (1992). The darpa tipster project. *SIGIR Forum*, 26(2):26–28, October.

Kluegl, P., Atzmueller, M., and Puppe, F. (2009). Textmarker: A tool for rule-based information extraction. In Chiarcos, C., de Castilho, R. E., and Stede, M., editors, *Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop*, pages 233–240. Gunter Narr Verlag.

Klügl, P., Atzmüller, M., and Puppe, F. (2008). Integrating the rule-based ie component textmarker into uima. In Baumeister, J. and Atzmüller, M., editors, *LWA*, volume 448 of *Technical Report*, pages 73–77. Department of Computer Science, University of Würzburg, Germany.

Petasis, G., Karkaletsis, V., Paliouras, G., Androutsopoulos, I., and Spyropoulos, C. D. (2002). Ellogon: A New Text Engineering Platform. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, pages 72–78, Las Palmas, Canary Islands, Spain, May 29–31. European Language Resources Association.

Petasis, G., Spiliotopoulos, D., Tsirakis, N., and Tsantilas, P. (2014). Sentiment analysis for reputation management: Mining the greek web. In *A. Likas and K. Blekas and D. Kalles (Eds.): SETN 2014, LNCS 8445*, pages 327–340. Springer International Publishing Switzerland.

Wirth, N. (1977). What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11):822–823, November.