PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 09-01

Lavinia: a collaborative NLP platform

Cecilia Techera Diego Garat Guillermo Moncecchi

2009

Lavinia: a collaborative NLP platform
Techera, Cecilia; Garat,Diego; Monchecci, Guillermo
ISSN 0797-6410
Reporte Técnico RT 09-01
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, Febrero de 2009

Lavinia: a collaborative NLP platform

Cecilia Techera, Diego Garat, Guillermo Moncecchi

Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Herrera y Reissig 565
Montevideo
Uruguay
{*ctechera,dgarat,gmonce*}*@fing.edu.uy*

February 22, 2009

Abstract

In this article we present Lavinia, a UIMA-based, collaborative web platform for Natural Language Processing, were both NLP software developers and linguistic analysts can test, use and share different NLP components in a straightforward way. Lavinia allows users to execute UIMA components using a web browser: they can create and configure pipelines of tasks, and view their execution results, without installing any extra-software. We believe that this approach can help people with little computational or programming background to get closer to NLP tools, and NLP component developers to easily share their work.

Keywords: NLP, UIMA, collaborative platform, Web 2.0

1 Introduction

It is well known that Natural Language Processing (NLP) applications are usually implemented as a set of smaller tasks, executed as a pipeline: the output of a task is the input of the next one. In addition, many of these tasks are repeatedly found in every NLP application: sentence and word splitters, part-of-speech taggers, etc. An important part of the effort in the construction of an NLP application relies not in the implementation of the required modules –most of them can be found implemented as open source–, but in their integration, that is, making the output of one task compatible with the input of the next.

Being aware of this problem, the NLP community has been working on approaches to increase the re-usability of well known solutions by easing their integration. This included the definition of standards for the representation of the object of the analysis (text, voice, etc.), the analysis process outputs, and the configuration of the different tasks as a unique pipeline.

The TIPSTER architecture [7] –the result of an ARPA-sponsored program in the US– was a major step in this process: its referential approach, where information about the subject of analysis (for example, for written language, a source text) is stored in a separate database, has been the predominant approach in NLP platforms. It also defined the concept of *annotations*: attributed labels to represent the analysis results, that are associated with *spans* of the text. The work of Bird and Liberman for the Atlas Framework [2, 1] defined the formal concept of *annotation graph*, a representation of text analysis as annotations over time spans, and its generalization to n-dimensional cases, called *annotation sets*. Most modern platforms adhere to these specifications.

Many of the most important present NLP architectures, as for example GATE [4], UIMA [5] or LinguaStream [8], use these definitions as a basis for the representation of their subjects of analysis. They also define mechanisms for the integration of NLP modules developed in different languages and their execution in a pipeline. Using this architectures, the modules have standardized access to external resources (lexicons, n-gram files, etc.) and visualization of analysis results.

But faced to the process of developing a new application, the researcher should install and configure not only the selected architecture, but also the different tools required for each task of his desired pipeline. Our work tries to aid in this process: we implemented a *collaborative NLP platform* for text processing, based on UIMA compliant modules. The platform, called Lavinia, offers a web interface to manage the modules, create an application as a pipeline of tasks, execute the application over a collection of texts, and see the analysis results. The user is not compelled to install any extra software but a web browser.

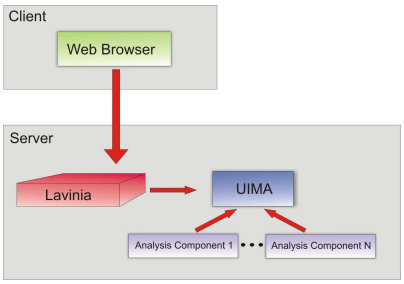

Figure 1: Lavinia architecture overview

2 Lavinia

We started our work with several objectives in mind:

- First, we wanted a general-purpose NLP platform to aid in the study of linguistic phenomena on text corpora, integrating our existing statistical and rule-based tools for the analysis of written texts.
- Second, the platform should act as a resource repository, allowing an easy incorporation of new or preexisting analysis modules, and their execution in a pipeline. After being added to the repository, each new module should be available immediately to every platform user.
- Third, we wanted an homogeneous and clear visualization of analysis results. This included the display of overlapping annotations, and the possibility to dynamically select the annotation set to be shown in a particular text.
- Finally, we wanted to put a focus on ease of use and minimal installation requirements, provided some of the modules users are not software experts.

After studying the state-of-the-art in NLP architectures, we decided that building a new NLP platform from scratch would not be necessary, provided that many solutions (as the ones mentioned above) already existed. Instead, we choose to focus on extending the existing tools capabilities by constructing a collaborative environment in which developers and users could share and use their NLP modules in a straightforward way.

In conclusion, our new platform, born as Lavinia, acts as a wrapper of an existing NLP platform, adding the new collaborative functionalities. The mission of Lavinia is to improve the user experience, being the latter a linguistic analyst or a component programmer. Lavinia core is open-source –the UIMA platform–, and, build upon it, our solution adds mechanisms to help on its use: it allows to dynamically generate pipelines with configurable components, to save and retrieve their configuration, to export the intermediate or final analysis results to a file, and, finally, to see the analysis results displayed accordingly to the user's choice. All of these functionalities are exposed as a Web application, with a web browser as the only requirement, for the final user.

A general scheme of the architecture of Lavinia is depicted in Figure 1; in the following sections we present the application in more detail.

2.1 UIMA, the processing core

The architecture we selected as Lavinia processing core was UIMA (Unstructured Information Management Architecture) [5], an open platform for the creation, integration and deployment of large scale unstructured information management (UIM) applications, originated at IBM, and now an incubating project at the Apache Software Foundation. Its main goal is to provide a common foundation for different developers worldwide to collaborate in the creation and exchange of natural language processing (NLP), information retrieval and machine learning solutions working on unstructured information. UIMA works on different kinds of information sources; as Lavinia is intended to work only on electronic written texts, in this section we will assume such kind of input.

UIMA makes focus on the analysis phase of information processing. There, unstructured information is analyzed, and structured information, representing analysis results, is created and associated with the input information in a predefined format. Every UIMA component (called *analysis engines*) must behave that way, independently of the specific task it does or the approach it takes. For example, a Part-Of-Speech tagger component associates POS information to words as a result of a more o less complex analysis of the source document. UIMA allows the aggregation of processing modules to create complex natural language processing applications, isolating the language processing algorithms from systems services such as component communication, information exchange, resource access, etc. (see figure 2)

For any NLP platform, the representation of documents and 'associated meta-data representing analysis results is a critical aspect. Following the TIPSTER guidelines, UIMA uses a referential approach, storing metadata in the form of separated *annotations*, which refer to (but are not part of) all or part of the original document [6]. This annotations associate typed *labels* to *spans* of text, identified by the start and end positions. Every information UIMA analysis engines shares are in this format, called Common Analysis System (CAS), and the architecture provides a container object which provides the API for their manipulation and modification. Labels are typed, with a user-defined type system, and take the form of *feature structures*, which each component makes accessible to

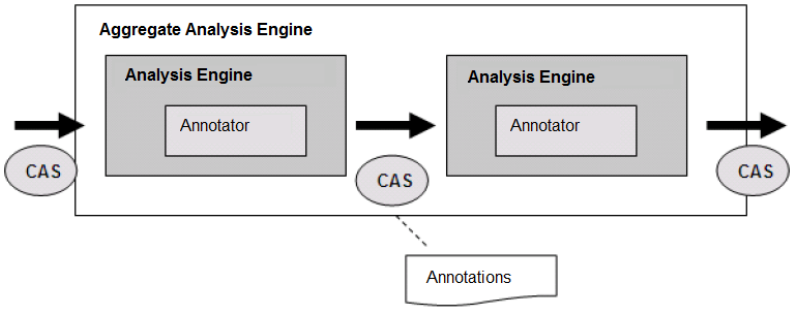

Figure 2: UIMA document processing

other components putting them on an *index*. CAS are also general information representation schema, and can hold any information relevant to a particular UIMA application.

The presented approach for natural language platforms (a common information representation and mechanisms for module composition and aggregation) is not new in language engineering. Architectures as GATE or Atlas also allow this type of organization. We selected UIMA because its data manipulation layer and type system are more appealing than GATE's. As we were thinking of an environment where the user could choose on demand which modules to use, and then which information to show, we found a clear and hierarchical representation of label types a must have for the system.

Summarizing, as a processing core, UIMA supports many aspects of Lavinia:

- Lavinia uses UIMA analysis engines as their processing components, and it uses UIMA infrastructure for grouping and deploying them.
- Every information Lavinia uses and generates is stored in a CAS.
- Lavinia component flow control is managed by UIMA.
- The type system that Lavinia uses for its parameters inherits from UIMA's *Annotation* type.

As a final remark, we want to state that, although it seems that Lavinia is highly coupled with UIMA, it is designed and implemented in a way that the processing core can be changed in the future without affecting the subsequent levels of the platform.

2.2 A front-end for better user experience

Having UIMA as a solid core for giving access to the meta-data representing analysis results, we decided to put the focus on the user experience.

As our first goal, we tried to build a solution that allowed any user, just by accessing a web page, to select different text analysis modules, build with them the desired analysis flow, and setting up the needed module parameters and resources. The platform would then verify the pipeline coherence: any component could be part of a flow if the output of the previous modules in the chain generate the meta-data needed for its own analysis.

For example, a component for a syntactic parser, which required a text tagged with part-of-speech tags, would necessarily appear after a POS-tagging module that generates them. Figure 3 shows a possible analysis flow, built on a tokenizer, a sentence splitter and a POS-tagger. Lavinia implements these controls using the type system provided by UIMA. Each UIMA component must specify what types it uses for the input and output. These types can be organized in a hierarchy and have attributes [6]. When a component is introduced into a pipe, Lavinia shows up its associated input and output types, and checks, at design time, if the pipeline is well-built.

Usually, NLP components are parameterizable; for instance, the Freeling POS tagger [3] can be configured to process texts for more than one

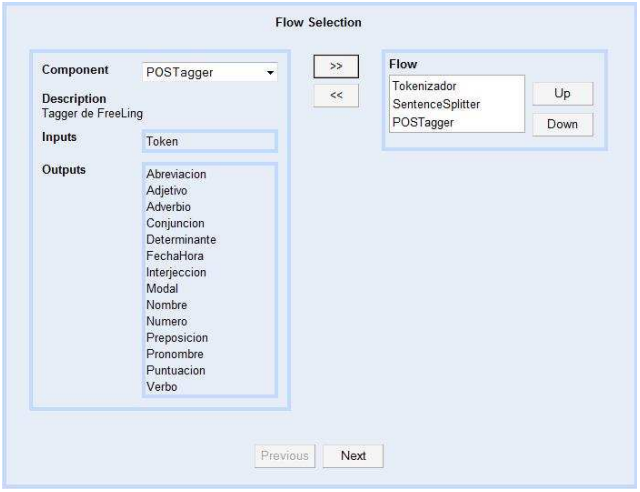

Figure 3: Flow Selection

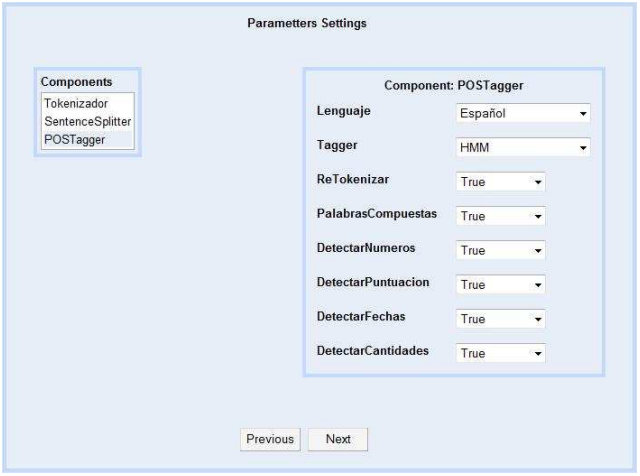

Figure 4: Parameter Configuration

language. Thus, components parameters are exposed by Lavinia to allow the user to customize the analysis pipe. As an example, figure 4 shows the Lavinia's window for configuring a POS-Tagger that will be used as a part of the analysis pipe.

From the component designer perspective, Lavinia allows the quick integration of new analysis modules: it is only necessary to build a UIMA-complaint module, upload it using the web interface, and declare its system types, the parameters to expose in the platform, and the required external resources (dictionaries, configuration files, etc.) For this moment on, the module will be available to all Lavinia users, who will be able to use it in their analysis pipes.

For the design of the web interface, we decided to put a strong emphasis in two characteristics of what is called the Web 2.0: the collaborative aspect of the implemented solution, and the use of dynamic web interfaces, based on asynchronous communications and light clients, using the Ajax technology. Because we believe this kind of interfaces helps in the usability by non specialized software people, choosing this technology was a fundamental decision in the construction of our platform. One pending point in our work is to measure how much this decision has effectively contributed to fulfill its intended goal.

In the next section, we present another aspect that we found very appealing of Lavinia: the graphical display of annotations in a uniform and dynamic way.

2.3 Showing the analysis results

As an architecture, UIMA does not provides mechanisms for visualizing the results obtained from text analysis. Their motto is "we give the mechanisms to generate and read the analysis results, and the application built up over UIMA shows it to the final user". Instead, Lavinia tries to be an application to collaborate and, above all, to be friendly to non programmer users. Thus, we decided to include a unified and clear view of the input text and the results of the analysis performed with it.

As it is mentioned in previous sections, the analysis results are annotations over the text associated with begin and end offsets of the marked span, and a series of attributes that can even refer to another annotations in the text. In a similar way as we did with flow construction, we worked with the following idea in mind: it is the user, and not the platform, the one who decides which and in what way the annotations will be displayed. In addition, we decided to allow the user to change this view without having to execute the analysis all over again.

Solving the problem of how to show the results of several kind of analysis in a uniform way –that is, a way that does not depend on the components involved in the analysis pipe– was not a simple task. Although several ways of displaying analysis results can be found in the literature –as far as we know–, neither of them can fulfill the requirements we imposed to our platform, that is:

- The user should have the possibility of choosing easily which linguistic phenomena to mark on the text.

```
For each annotation type ti, in priority order
   For each annotation aj of type ti
      For each cell between aj.start and aj.end
         If the cell has background color
            If the cell has border
               Assign border to aj.start and aj.end
            else
               Assign borders
         else
            Assign background color
       End
    End
End
```

Figure 5: Pseudo-code for the visualization algorithm

- Annotations should be clearly highlighted, and the different type of annotations should be recognizable at a first glance.
- The extra-information of a given annotation –for example, the lemma of a verb, its tense, etc.– should be in some way accesible.
- The overlapping annotations should be managed by the visualization module in a way that all of them are clearly displayed to the user.
- The platform should provide a way to see cross-references between annotations.

Therefore, we developed a mechanism for visualizing the annotations, which was deployed into Lavinia under the "visualization module" name. This mechanism is based on priority layers of annotations, in a similar way that traditional Geographic Information Systems do. Each type of annotation generated by an analysis pipe is viewed as a layer, and the user can chose its preference order, if it should be highlighted or not, and, if so, the associated background and font colors.

The visualization module displays the layers by their priority order, painting the marked span of text accordingly to the layer's colors. When an annotation overlaps a segment that is already painted –in other words, part of the annotation span is already painted because of the existence of an annotation with greater priority–, the visualization module only paints a box around the corresponding span, without filling it. Thus, if two annotations overlap, the one with greater priority will be displayed over the other one. In case there are already two overlapping spans –that is, the overlapping segment already has a background color, given by the first annotation, and a color border, given by a second one–, the third layer is only marked by a pair of brackets at the beginning and the end of the span. The pseudo-code of the painting algorithm is shown in figure 2.3.

One point to remark is that all of the layer attributes –font and background colors, its priority level, and if it should be displayed or not– can be modified dynamically and *after* doing the analysis; the display window will be modified accordingly without requiring to run the analysis

again. In this way, the user can quickly see the results in several different ways –for example, changing the order of the layer or making it visible or invisible– in order to see different linguistic phenomena. In addition, the user can click over any portion of the text, and Lavinia will show all the attributes associated with the annotations that include the clicked part of the text.

Figures 6 and 7 show how the global vision of a pipe output can be drastically modified: they show the same analysis results over the same input text, but displayed in two different ways. In both figures, at the left, it can be found the color configuration, which annotations should be displayed, and with which priority (upper annotations have higher priority than lower ones.)

Going back to the points stated as our goals, this solution does not resolve the problem of linked annotations, as it is, for example, the case of anaphora. Another unresolved problem is the displaying of hierarchy annotations, such as syntactic parser trees. We believe that both problems are related, and we expect to solve it for the next version of our platform.

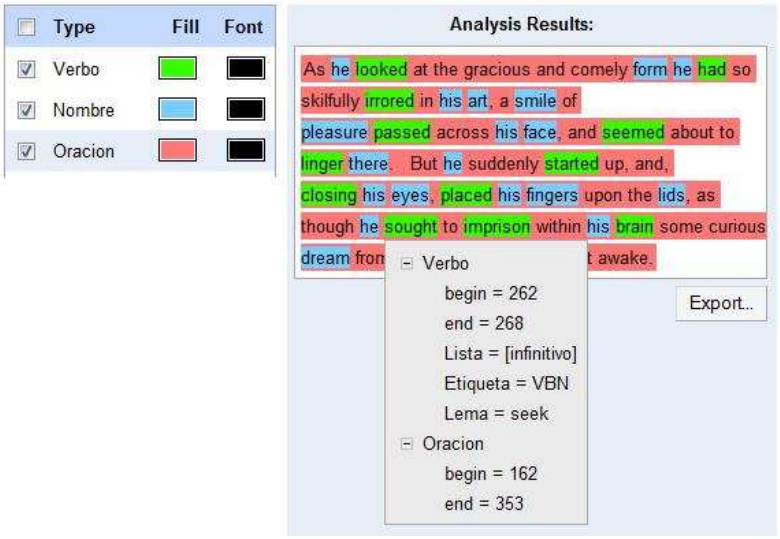

Figure 6: Analysis Results

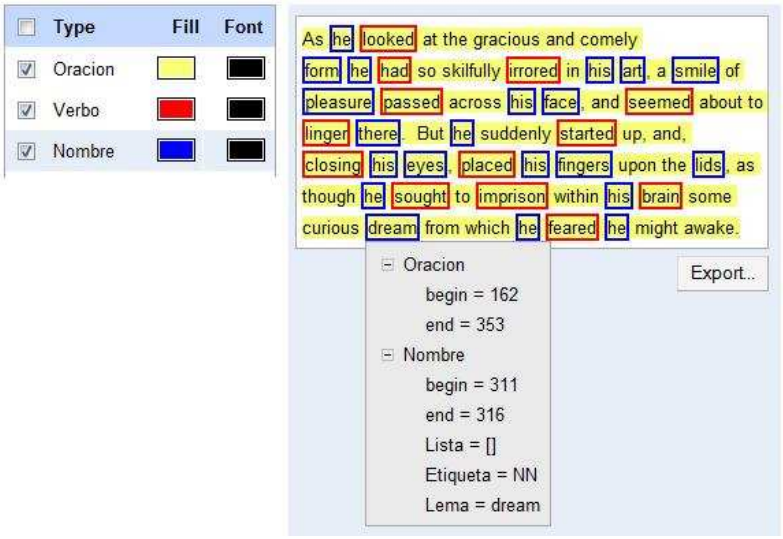

Figure 7: Analysis Results, another view

3 Conclusions

Lavinia is a collaborative NLP platform that allows to share NLP modules in a web based environment. The objective of Lavinia is to help people interested in natural language analysis to get closer to NLP tools, even if they do not have an extensive computational background.

In addition, Lavinia adds a new perspective for NLP developers: they no longer need to install and configure the platform or the tools they require; instead, they simple can play with the modules in a browser and test the results. Thus, the web platform can aid for prototyping a more complex application.

Lavinia also offers a place to publish new components implemented in a well-recognized standard (UIMA). The process is easy and, in the same way as with the analysis, all can be done through the web interface. Existing components can be adapted to this platform, by implementing a UIMA-proxy. This was the case of a rule-based engine, the contextual-rules analyzer [9], originally developed in Prolog, and encapsulated in a UIMA-analyzer, or the Freeling tools, developed in C++, that were adapted in the same way.

Lavinia started with a small set of modules to offer: a tokenizer, a part-of-speech tagger and a shallow parser (based on Freeling), and a text marker based in the Contextual Rules engine. Nevertheless, the platform has being successfully used for tasks such as semantic and orthographical disambiguation and time expression recognition, within a curse of natural language processing.

The expected next steps in the development of Lavinia are: (a) the development of new components to the platform for common NLP tasks, possibly based on already existing solutions adapted for the platform; (b) the addition of new visualization components and capabilities, to provide, for instance, ways of displaying hierarchy results and annotation links; (c) the addition of semantic search capabilities over analyzed text and (d) the ordered evaluation of our platform by a group of final users, and its improvement, in the constant effort for making it easier and more intuitive

References

[1] Bird, S., Day, D., Garofolo, J. S., Henderson, J., Laprun, C., and Liberman, M. Atlas: A flexible and extensible architecture for linguistic annotation. *CoRR cs.CL/0007022* (2000).

[2] Bird, S., and Liberman, M. A formal framework for linguistic annotation. Tech. Rep. MS-CIS-99-01, University of Pennsylvania, Philadelphia, Pennsylvania, 1999.

[3] Carreras, X., Chao, I., Padró, L., and Padró, M. FreeLing: An Open-Source Suite of Language Analyzers. In *4th International Conference on Language Resources and Evaluation (LREC'04)* (Lisbon, Portugal, 2004).

[4] Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. 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* (2002).

[5] Ferrucci, D., and Lally, A. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering* (2004).

[6] Götz, T., and Suhre, O. Design and implementation of the uima common analysis system. *IBM Syst. J. 43*, 3 (2004), 476–489.

[7] Grishman, R. Tipster text architecture design version 3.1. Tech. rep., New York University, Philadelphia, Pennsylvania, 1998.

[8] Widlöcher, A., and Bilhaut, F. La plate-forme LinguaStream: un environnement intégré pour l'expérimentation en TAL. In *TALN 2006* (Leuven, Belgique, avril 2006). Présentation orale et démonstration sans acte.

[9] Wonsever, D., and Minel, J.-L. Contextual rules for text analysis. In *CICLing '01: Proceedings of the Second International Conference on Computational Linguistics and Intelligent Text Processing* (London, UK, 2001), Springer-Verlag, pp. 509–523.