A Run-Time Shared Repository for NLP Tools

João Graça[1], Nuno J. Mamede[1], and João D. Pereira[2]
Spoken Language Systems Lab
Rua Alves Redol 9, 1000-029 Lisboa, Portugal

[1] L^{2}F – INESC-ID Lisboa/IST
{joao.graca,nuno.mamede}@l2f.inesc-id.pt,
[2] Software Eng. Group – INESC-ID Lisboa
joao@inesc-id.id

Abstract. Natural Language systems are usually composed by several tools executed in a pipelined manner. This type of processing presents some problems caused by loss of information between tools. This paper presents a solution to this problem in the form of a shared repository for natural language processing. First, related architectures are described along with the reasons why they were not chosen as reference architectures. Then we present our solution in the form of an architectural model and the corresponding API.
Resumo. Os sistemas de língua natural são normalmente compostos por diferentes ferramentas cuja execução se processa sequencialmente. Este tipo de processamento apresenta problemas devido à perda de informação entre ferramentas. Este artigo apresenta uma solução para este problema que consiste num repositório partilhado de língua natural. Primeiro, apresentam-se arquitecturas relacionadas e o motivo pelo qual estas não foram consideradas como arquitecturas de referência. De seguida, apresenta-se a nossa solução sob a forma de um modelo arquitectural e a respectiva API.
Keywords. Run-Time Repository,Linguistic Annotation, Blackboard.

1 Introduction

A Natural Language system is composed by several tools, each performing a specific task. Executing the system corresponds to executing the various tools in a pipelined manner (see figure 1).

This kind of pipeline architecture presents the following problems:

- Information may be lost between executions of different tools: when PAsMo splits the contraction "do" into "de" + "o", the information that the original word in the text was "do" is lost to the following tools in the system.
- Tools must ensure that information that they consider irrelevant is passed to the output. In the example, the syntactic parser SuSAna must keep the word stems for future tools although it has no need for them.

To avoid these problems, our solution consists of having a shared repository which is the source of input/output of all tools.

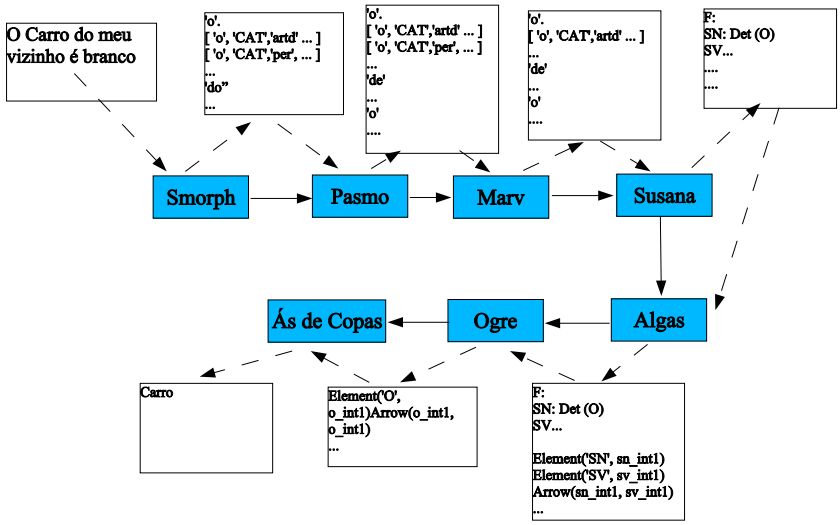

Fig. 1. Existing system chart: Smorph [1] is a morphology processor, PAsMo [10] is a ruled-based rewriter, MARv [11] is a morphological disambiguator, SuSAna [2] is a syntactic analyzer, Algas [5], Ogre [5], and AsdeCopas [6] are semantics domain tools.

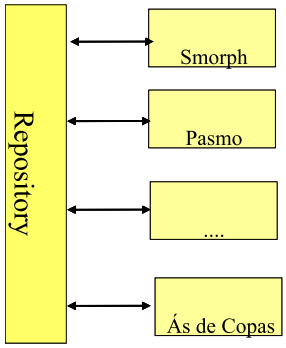

Fig. 2. Distributed repository

Each tool reads the required information, and adds new information to the repository, as in a blackboard architecture. The repository must have the following properties in order to solve the presented problems:

- There is no information lost;
- Tools use only the information they require.

The repository also has the following properties:

- Tools use a generic API for input/output: this way, the tool's developer avoids dealing with input/output decisions and is able to concentrate on the algorithmic part;
- The repository provides a broad coverage for the data used by linguistics tools;
- The repository does not associate any semantics with the data it contains. Each tool registers itself in the repository and uses the repository's API to read/write its data;
- Each tool can access all the repository data, as if all the data was present in memory, through the API. This avoids reading several output files into structures inside the application.

2 Related Work

To deal with this problem directly or indirectly, some researchers define an architecture for representing linguistic information. The following architectures will be analyzed: The annotation graph architecture [8], Atlas [9], Emu System [7], Natural Language toolkit [4] and Whiteboard [3].

The annotation graphs architecture [8] is a model for linguistic annotations of time-series data. It presents the following problems when used as a reference architecture: first, it limits the kind of signal that can be used because the signal must be related with time. Since it is an architecture for building annotation tools, not to build a shared repository, it lacks the concept of application (described in section 3.1) that is required in our repository. It also has some limitations concerning the expressive power of the linguist annotation model, and it has no obvious way to represent ambiguous annotations for the same region of the source signal. An ambiguous annotation appears, for example, in the text segmentation process where a compound word can be segmented either as a compound word or as several individual words (see figure 3).

The Atlas [9] architecture is a general purpose annotation framework. Its starting point is the annotation graphs model. It presents several generalizations to the annotation graphs models, namely any kind of signal can be used. Nevertheless, the other problems mentioned above remain.

The Emu speech database system [7] provides a flexible set of interfaces for developing and extracting data from speech databases. An Emu speech database consists of a collection of sampled data files, each of which has one or more associated label files. Emu is not adequate because it requires that a database

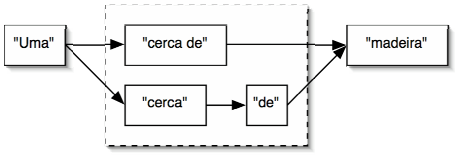

Fig. 3. Segmentation ambiguity

template be defined and one of the objectives of our repository is that it has no information about the data it contains, being a completely dynamic system.

The Natural Language Toolkit [4] is a suite of open source program modules, tutorials and exercises, covering symbolic and statistical natural language processing. These modules work with structures called tokens which correspond to units of text containing a set of properties. The token structure works as a blackboard because no information is removed from it during the processing phases carried out by the various tasks. Nevertheless, it lacks the concept of shared repository with different tools adding a level. Also, the token abstraction is unable to represent some ambiguity problems, as explained above.

The WhiteBoard [3] architecture is an architecture based on the concept of annotated text. The different tools enrich an XML-encoded text with layers of meta information that are also represented in XML. Each component can exploit or disregard previously assigned annotations: one of the reasons that lead us to discard this architecture is that it requires the source signal to be a text. One objective of our repository is that it can accept any kind of source signal.

3 Proposed Solution

The shared repository consists of a server that accepts connections from different tools. Each tool can consult the information present in the repository and add the information it produces. The repository avoids information loss between different tools and allows developers to concern themselves only with the algorithmic part of the tool.

3.1 Model

The repository represents linguistic data according to the following model (see figure 4):

- **Repository** - This is the main concept: keeps the data produced by the different tools. It contains the source signal which is the data that will be analyzed by the tools.
- **Application** - Corresponds to the output of the execution of a tool. It is identifiable by a unique id, the name and type of the tool that originated it, and a time stamp. The same tool may be executed several times and each

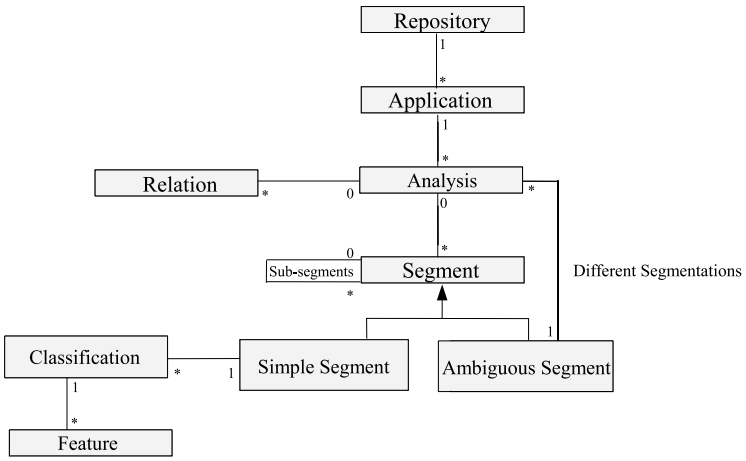

Fig. 4. Model

output corresponds to a different application, these different applications will have different ids and different time stamps.

– **Analysis** - Corresponds to an analysis performed by a tool, like a syntactic tree created by a syntactic parser. The execution of a syntactic parser generates an application that will contain several analysis, one for each syntactic tree produced. An analysis is identifiable by an unique id inside the repository.
– **Segment** - Corresponds to a part of the source signal that will be classified. For example, a word inside a text (see figure 5). An analysis can contain

Fig. 5. Word segment over a text signal

a set of segments corresponding to the source signal that was analyzed. Segments are aligned with the source signal. Each is further divided into simple segments and ambiguous segments. It can contain sub-segments used for defining trees (see figure 6). It has a unique id inside the repository.
– **Simple Segment** - Corresponds to a segment that can be classified.
– **Ambiguous Segment** - Corresponds to a graph which represents ambiguity inside a segment. The ambiguous segment contains a set of analysis containing the graph paths (see figure 7, representing the ambiguity presented in figure 3)

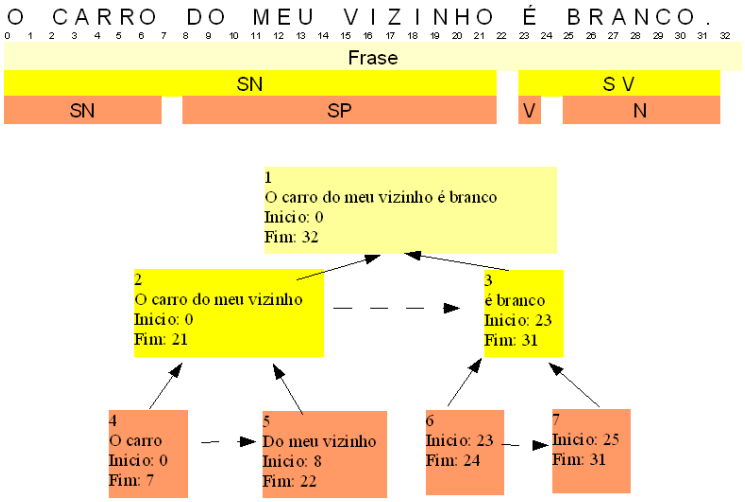

Fig. 6. Subsegments representing a syntactic tree

- **Relation** - Represents an association between segments. An analysis may contain a set of relations, each relation can associate segments from different applications present in the repository: those segments are identifiable by their id.
- **Classification** – Corresponds to the classification given to a simple segment. A segment can contain different classifications in order to represent the classification ambiguity, for example, the part of speech of a given word.
- **Feature** - A feature is an attribute-value pair corresponding to some characteristic of the segment. A classification is composed by a set of features. A feature attribute does not have any semantic meaning.

3.2 API

The repository works as a server that accepts connections. Each tool sends a message to the repository in order to register itself. The tool receives a new application instance where it will add the information it produces and receives a read-only repository instance in order to navigate through the existing information. When the tool finishes its processing phase, it sends a message to the repository to save the new application object and end its interaction with the repository.

The data inside the repository is persistent. From the tool's point of view, all the repository data is reachable through the repository object that holds the root of the information graph. From the repository's point of view the data may be saved in a persistent store and will be accessed when required.

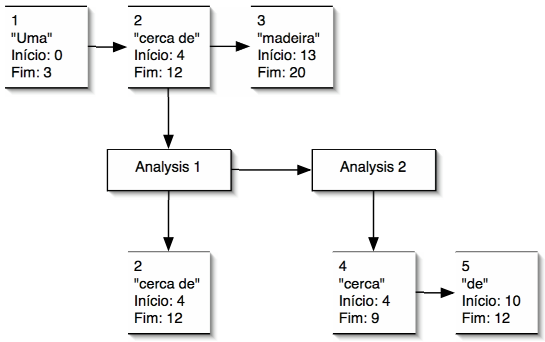

Fig. 7. Representation of a tokenization ambiguity

The tool can access the repository data in several ways. It can look for a given object by its id, like when selecting a given application, analysis or segment. It can look for all applications produced by a given tool, or for all applications of a given type. For example, a part of speech disambiguation tool may look for all applications generated by morphologic analyzers and disambiguate the part of speech based on the different applications.

The API also has iterators over each domain element: it is possible to iterate over the application of the repository, over the segments of an analysis, etc.

When iterating over a list of segments that contains ambiguous segments, two options are available: one consists of receiving several iterators, each corresponding to a possible path of the graph representing the ambiguity. Another possibility consists of passing to the tool the ambiguous segment object. It is the tools responsibility to deal with the ambiguous segment.

4 Future work

Concerning the model, the main improvement consists of segments that contain references to a source signal that is present in the repository. The source signal will be removed from the repository and each segments will reference other segments or groups of segments. The source signal will correspond to another level inside the repository composed of segments with no semantic meaning. This abstraction of the source signal is necessary to make the repository more dynamic and to allow different source signals over the repository's life cycle. One tool can use a speech wave as the source signal while another tool can use the text recognized from that speech wave as the source signal.

Although the repository API's is sufficiently general for all applications, allowing access to all repository data, some specific APIs will be build on top of the base API. These specific APIs will be defined for some well known tool classes like, for example, morphologic analyzers. These APIs will work as syntactic sugar for the tool builder, using concepts present in the tool that will be

mapped on to the model. When building a morphologic analyzer, the specific API will have functions like these:

- Next Word - That will return the next word in the text, corresponding to iterate over the segments;
- add part of speech - Adds a classification to a segment corresponding to its part of speech

The objective is that in the future people tend to use and develop specific APIs for their tools in order to turn their code more readable.

Currently the API is being implemented and the tools present in the laboratory are being integrated with the repository, so we still do not have any practical results.

References

1. Salah Aït-Mokhtar. *L'analyse Présyntaxique en une seule étape.* PhD thesis, Université Blaise Pascal, 1998.
2. Fernando Batista and Nuno Mamede. SuSAna: Módulo multifuncional de análise sintáctica de superfície. In Julio Gonzalo, Anselmo Pe nas, and Antonio Ferrández", editors, *Proc. Multilingual Information Access and Natural Language Processing Workshop*, pages 29–37, Sevilla, Spain, November 2002. IBERAMIA 2002.
3. Bernd Kiefer Hans-Ulrich Krieger Stefan Müller Günter Neumann Jakub Piskorski Ulrich Schäfer Melanie Siegel Hans Uszkoreit Berthold Crysmann, Anette Frank and Feiyu Xu. An integrated architecture for shallow and deep processing. Technical report, DFKI GmbH. http://www.dfki.de/ feiyu/wb-acl02.pdf.
4. Steven Bird and Edward Loper. Nltk:the natural language toolkit. Technical report, Department of Computer Science and Software Engineering University of Melbourne Victoria 3010, Australia and Department of Computer and Information Science University of Pennsylvania Philadelphia PA 19104-6389, USA. http://www.ldc.upenn.edu/sb/home/papers/nltk.pdf.
5. Luísa Coheur, Nuno Mamede, and Gabriel G. Bès. From a surface analysis to a dependency structure. In *Workshop on Recent Advances in Dependency Grammar (Coling 2004)*, Genebra, Suiça, 2004.
6. Luísa Coheur, Nuno Mamede, and Gabriel G. Bès. A multi-use incremental syntax-semantic interface. In *Estal – España for natural Language Processing*, Alicante, Espanha, Outubro 2004. Springer-Verlag.
7. The emu speech. http://emu.sourceforge.net/.
8. HaeJoong Lee Kazuaki Maeda, Xiaoyi Ma and Steven Bird. *The Annotation Graphs Toolkit (Version 1.0): Application Developer's Manual.* Linguistic Data Consortium, University of Pennsylvania.
9. Cristophe Laprun, Jonathan Fiscus, Jonh Garofolo, and Sylvain Pajot. Recent improvements to the atlas architecture. Technical report, National Intitute of Standards and Technology, 99. http://www.nist.gov/speech/atlas/download/hlt2002-atlas.pdf.
10. J. L. Paulo. Aquisição de termos automática. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, 2002. (este documento).
11. R. Ribeiro, L. Oliveira, and I. Trancoso. Morphossyntactic Disambiguation for TTS Systems. In *Proc. of the 3rd Intl. Conf. on Language Resources and Evaluation*, volume V, pages 1427–1431. ELRA, 2002. ISBN 2951740808.