A Framework for Integrating Natural Language 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/IST
joao@inesc-id.id

Abstract. Natural Language processing (NLP) systems are typically characterized by a pipeline architecture in which several independently developed NLP tools, connected as a chain of filters, apply successive transformations to the data that flows through the system. Hence when integrating such tools, one may face problems that lead to information losses, such as: (i) tools discard information from their input which will be required by other tools further along the pipeline; (ii) each tool has its own input/output format.
This work proposes a solution that solves these problems. We offer a framework for NLP systems. The systems built using this framework use a client server architecture, in which the server acts as a blackboard where all tools add/consult data. Data is kept in the server under a conceptual model independent of the client tools, thus allowing the representation of a broad range of linguistic information.
The tools interact with the server through a generic API which allows the creation of new data and the navigation through all the existing data. Moreover, we provide libraries implemented in several programming language that abstract the connection and communication protocol details between the tools and the server, and provide several levels of functionality that simplify server use.

1 Introduction

Natural Language processing (NLP) systems are typically characterized by a pipeline architecture, in which several NLP tools connected as a chain of filters apply successive transformations to the data that flows through the system. Usually, each tool is independently developed by a different person whose focus is on his/her own problem rather than on the future integration of the tool in a broader system. Hence when integrating such tools, several problems arise, which are mainly related to the following: (i) how the tools communicate with each other, (ii) what kind of information flows between the several tools (may cause information lost).

At the Spoken Language Systems Lab (L^2F), where this work was developed, several NLP systems have already been created. Every time tools were integrated to compose a system most of the detected problems were concerned with the information flow between those tools, which led to the loss of information. These problems are: (i) usually, the output of a tool consists just of the data it has acted upon and it does not contain all the input data. Sometimes this raises a problem if the discarded data is also required by a tool appearing at a later stage of the pipeline; (ii) each tool has its own input/output format so conversions between data formats may be needed when a tool consumes data produced by another one. Moreover, this conversion may not be possible if the descriptive power of each format is distinct; (iii) the formats used by different tools do not establish relations between the input/output data. These relations are useful for aligning information produced at different levels and to avoid the repetition of common data across them.

The proposed solution is a framework using a client server architecture instead of a pipelined architecture. In our solution, the server acts as a blackboard where all NLP tools (clients) add/consult data. The server maintains cross-relations between the existing layers of data. The data is kept in the repository under a conceptual model independent of the client tools. This conceptual model allows the representation of a broad range of linguistic information. The tools interact with the repository through a generic remote API that permits the creation of new data and the navigation through all the existing data. Moreover, this work provides libraries implemented in several programming languages that abstract the connection and communication protocol details between NLP tools and the server, and provide several levels of functionality that simplify the integration of NLP tools.

2 Solution Requirements

We define some requirements that a framework should fulfil in order to solve the problems we detected. These requirements concern the expressive power of the conceptual model and the functionalities offered to the tools. The model must be able to represent linguistic phenomena deemed of interest, and several types of primary data sources (text, speech). The requirements for the conceptual model include:

– Keeping all information produced by an NLP tool on the same layer;
– Representing segmentation ambiguity;
– Representing trees of linguistic elements;
– Representing relational information between linguistic elements;
– Representing classification ambiguity;
– Representing relations between information from different layers (cross-relations).

The requirements defined for the conceptual model were compared against the requirements that are being defined by ISOTC37/SC4 (Terminology and other language resources) to define a standard for linguistic annotation [6]. We found them to be very similar, which strengthened our conviction that any model used to represent linguistic information should follow these requirements.

Finally, we identified the following requirements concerning the functionality that must be supported by the framework:

– It must allow the selection of data based on the identification of each layer;
– It must allow parallel processing of data kept in the server;
– It must guarantee that all data in the repository is kept persistently;
– It must allow interaction with tools written in any programming language.

3 Related Work

We analysed several architectures whose goal was to simplify the creation or integration of NLP tools, towards their usage in NLP systems, namely: the Emdros text database system [9], a text database engine for analysis, and retrieval of analyzed or annotated text; the Natural Language Toolkit [8], a suite of libraries, and programs for symbolic, and statistical natural language processing; the Gate architecture [3], a general architecture for text engineering that promotes the integration of NLP tools by composing them into a pipes and filters architecture; and the Festival speech synthesis system [10], a general framework for building speech synthesis systems.

We also compared some works from the linguistic annotation field, whose focus is on the definition of a logical level for annotation independent of the annotations' physical format. This logical level should be able to represent the most common types of linguistic annotations to promote reuse of annotated corpora. The conceptual model we required to represent the input/output of an NLP tool can be seen as this logical level. In this field we compared two works: the Annotation Graphs Toolkit (AGTK) [7] that is an implementation of the Annotation Graphs formalism [2], the most cited work in this area; and the ATLAS architecture [1], a generalization of the Annotation Graphs formalism to allow the use of multidimensional signals.

The AGTK and the ATLAS architectures do not allow the separation of information into layers. In these architectures, to avoid the loss of information, each tool has to load all previous annotations, and then save them together with its results. This strategy has several drawbacks: first, each tool must know how to manage data which may be unrelated with the tool itself. Second, each tool may have to load and parse extra data upon its initialization and consequently save extra data when terminating. Finally, it is difficult for a tool to handle data from several tools at the same time, because it must merge the common data from the input tools. Moreover, the adoption of the Annotation Graphs model is not possible, mainly because it does not allow the representation of relational information, nor the representation of cross-relations between several data layers. The extensions performed by the ATLAS architecture provide a better representation for conceptually different linguistic phenomena, such as hierarchic trees, and ambiguous segmentations. Furthermore, ATLAS allows the use of every type of data sources. But, even so, this model still presents the same problems as the Annotation Graphs formalism.

The Emdros framework has a representation model that is too restrictive for our objectives. For example, it restricts the media type to text. In Emdros it is not possible to properly represent some types of linguistic phenomena, such as classification ambiguity, or relations between elements.

The NLTK restricts development to the Python programming language, and relies on the Python interpreter to work. Moreover, its underlying conceptual model is not able to fulfill all our requirements, for instance, the representation of ambiguities, such as, classification ambiguity.

The GATE framework presents the same problems as the AGTK formalism, concerning its conceptual model. Moreover, it limits its utilization to the Java programming language. GATE promotes the integration of NLP tools into a pipes and filters architecture, which as we mentioned in the introduction, has some problems that led to the development of this work.

Finally, the Festival framework has a model that does not allow the fulfilment of all our requirements, namely: i) its data source must be text, ii) it cannot represent all types of ambiguities defined in the requirements, iii) it does not allow the concurrent execution of different tools.

4 Proposal

Our proposal consists of a client server architecture. The clients are NLP tools while the server consists of a centralized repository of linguistic information and data sources represented under a conceptual model. Each NLP tool can interact with the server in two ways:

– By using a remote interface independent of the tool's programming language;
– By using a module in its programming language that abstracts the communication and protocol details between the client and the server, and offers an implementation of the conceptual model. The use of the server is simpler using the client library than using the remote API, but the use of the client library requires an implementation of the client library for each programming language used.

4.1 Conceptual Model

The conceptual model is able to represent and relate various types of linguistic information produced by several NLP tools. Besides representing the different linguist phenomena, the model simplifies the use of linguistic information.

The entities composing the conceptual model, described in Fig.1 are:

– *Repository*: a centralized linguistic repository that stores the output of several NLP tools, and organizes that information into layers. Each layer is univocally identified inside the repository. There are two types of layers: *SignalData* and *Analysis*;

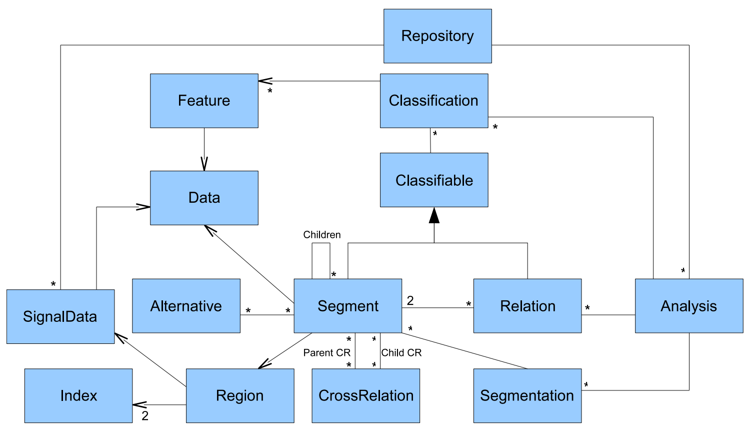

Fig. 1. Conceptual Model class diagram.

- *Data*: data type abstraction used by the *Repository*, e.g. *String*;
- *SignalData*: an abstraction of a raw data source, such as a text;
- *Index*: a point in a *SignalData*;
- *Region*: defines a region in a *SignalData*, using a pair of *indexes*;
- *Analysis*: linguistic information other than *SignalData* elements produced by an NLP tool. An *Analysis* may be open or closed indicating whether the tool has already finished the addition of data into the *Repository*. An *Analysis* can only be changed if it is open. The *Analysis* is responsible for the creation of *Segmentations*, *Relations* and *Classifications*;
- *Segment*: a linguistic element, e.g., a word. A *Segment* may contain two Data elements: the original data and the derived data. The original data corresponds to a linguistic element identified in a *SignalData*, while the derived data corresponds to a possible transformation performed over the original data. A *Segment* may be ambiguous, meaning that it has a set of *Alternative* elements for the linguistic element that it identifies. It may be hierarchic, meaning that it has a *Parent Segment* and *Child Segments*. A *Segment* has a set of disjunct *Classification* elements, where each *Classification* assigns a set of characteristics to the *Segment*. It also has a set of Relations, that establish links between two Segments from the same Analysis. A *Segment* may belong to a set of *CrossRelations*, which are used to establish structural relations between *Segments* from different *Analyses*;
- *Segmentation*: a set of sequentially ordered *Segments*, e.g., the words in a sentence. The *Segmentation* is responsible for the creation of *Segments*;
- *Relation*: a link between two segments. For example, the relation between the subject and the verb in a phrase;
- *Classification*: a set of characteristics of a *Segment* or *Relation*. For instance, the morphological features of a word;

- *Alternative*: a set of *Segments* representing different alternatives for an ambiguous linguistic element;
- *Cross-Relation*: a structural relation between *Segments* of distinct *Analyses*.

Figure 2 shows how *CrossRelations* can be used to align different *SignalData* elements in the translation of the sentence *"The red rose is pretty"* from English to Portuguese, *"A rosa vermelha é bonita"*. The *Repository* contains four layers. The first corresponds to the original *SignalData* containing the English text, where the possible *Indexes* are represented as integers under each character, and the possible *Regions* identifying the words are represented inside a box. The second layer is an *Analysis* containing the segmentation of that text into words: it contains one *Segment* for each word, that uses a *Region* (indicated by an arrow from the *Segment* to the *Region* in the *SignalData*) to refer to the word's text. The *Segmentation* contains those *Segments* in accordance with the word's order in the text. The third layer is a *SignalData* containing the translation of the English text from the first layer, and the fourth layer is an *Analysis* produced by an English to Portuguese translator, which creates a *Segmentation* where each *Segment* represents a Portuguese word from the third layer. The representation of the third and fourth layers is the same as the explained for the first two layers. The alignment between the two texts is achieved by adding *CrossRelations* between *Segments* (dotted arrows) from the corresponding *Analyses*.

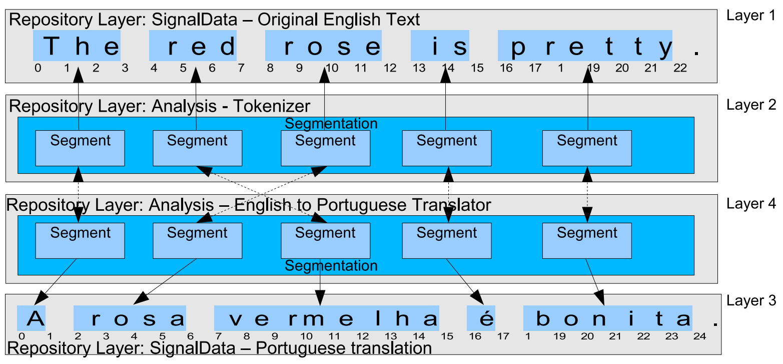
Repository Layer: SignalData – Original English Text
Layer 1
The red rose is pretty .
Repository Layer: Analysis - Tokenizer
Layer 2
Segmentation
Segment
Repository Layer: Analysis – English to Portuguese Translator
Layer 4
Segmentation
Layer 3
A rosa vermelha é bonita .
Repository Layer: SignalData – Portuguese translation

Fig. 2. Data alignment example.

4.2 Server Architecture

The server architecture consists of a shared data style. This architecture has the advantages of allowing clients to be added without the server knowledge, and of allowing the integration of the data produced by all the tools. The linguistic information, described using the conceptual model, is managed by the server.

The server is organized in three layers: the data layer, the service layer, and the remote interface layer. Each layer can only use the adjacent layers through their interfaces. The layered approach promotes portability, and maintainability since the role of each layer is well identified, and the implementation of each layer can be changed without affecting the other layers.

The Data Layer contains the logic of the application, and uses the conceptual model to represent the linguistic information. Besides representing all the linguistic information, the Data Layer is also responsible for guaranteing the persistence of all linguistic information stored in the server.

The Service Layer [5] defines the server's boundary by providing a set of methods and coordinates the server's response to each method. It is used by the Remote Interface Layer, which handles the specific protocol details, and encapsulates the Data Layer. The Service Layer is responsible for hiding the details regarding the representation of the domain elements of the Data Layer. It transforms the domain elements into Data Transfer Objects (DTO) which will be passed to the client. A DTO [5] is an object with no semantics, and is used to pass information between the client and the server. Each DTO can hold two kinds of information from domain objects: i) identification information used to access domain objects; ii) read-only information from domain objects that may be required by the client.

The Service Layer is also responsible for providing methods that allow the creation of iteration facilities on the client side. Moreover, and since the Service Layer is a single entry point into the server,it is an ideal place to perform logging and authentication actions.

The Service Layer together with the DTOs works as a Remote Facade [5] thus diminishing the number of remote calls required for certain operations.

The Remote Interface Layer provides the methods that are available to client tools according to a selected protocol. It communicates with the Service Layer, and is responsible for serializing the DTOs provided by the Service Layer into their external representation, which will be sent across the connection. It is also responsible for assembling the DTOs back, and pass them to the Services Layer.

4.3 Client Library Architecture

The utilization of the client library allows an NLP tool to abstract from details concerning the communication with the server, and the data exchange protocol. It also provides some high level interfaces that may simplify the integration of NLP tools. The client library uses a layered architecture, each layer is described in the rest of this subsection.

The Client Stub Layer is responsible for communicating with the server under the chosen protocol, through the server's Remote Interface layer. All the other layers of the client library depend and use this layer. This way the other layers are independent of the specific communication protocol that is being used.

The Conceptual Model Layer implements the conceptual model. It allows NLP tools to use the conceptual model as their object model, thus simplifying their creation. Since the concepts used by NLP tools are usually similar, by using the conceptual model we desire to avoid the definition of an equivalent one every time a new tool is created. In addition, by using only the interfaces provided by the Conceptual Model layer its concrete implementation can be changed without changing the tool. This way, an NLP tool can be used as a stand-alone tool or as a client tool connected to the shared repository just by changing the implementation of the Conceptual Model Layer.

The Conceptual Model layer elements are proxies for the elements of the Conceptual Model in the server. The methods performed on those elements are delegated into the corresponding elements in the server.

The repository can be used concurrently by several NLP tools, so it is possible that a tool consumes information that is being produced by another tool at the same time. If the consumer is faster than the producer and depletes the data that is being produced, the consumer may finish its processing due to a lack of data before what was expected. To avoid this situation the iterators on the client side have a blocking behaviour. The method `hasNext()` only returns false when the Analysis that contains the data that is being iterated is closed and there are no more elements to iterate. However, this policy can result in a deadlock to the consumer tool if the producer tool ends abruptly without closing its Analysis. So, we introduced a time limit for which a client method can be blocked in the method `hasNext()`.

The Extra Layers Layer provides extensibility to the client library. It represents new layers that can be added on top of previous ones, enabling the creation of domain specific layers, which may simplify the creation of new NLP tools. For example, a part-of-speech tagger could use a layer that provides the concepts of word, phrase and text, with methods such as `nextWord()`, and `addGender(Word w)`. The use of Extra Layers can also provide semantic meaning to the linguistic information kept in the Repository for a given NLP system.

5 Results and Future work

We implemented our framework in *Java* (around 57 classes) using the XML-RPC protocol provided by the APACHE XML-RPC package. This protocol was chosen because of its simplicity and because it does not impose any restrictions on the programming language used by client tools. Each layer of the server defines a set of interfaces that are used by the other layers. We have implemented a set of specialized classes in the Data Layer to handle text input signals (*TextSignalData*,*StringData*,*TextIndex*). The Service Layer is implemented by two classes: one to implement the methods from its interface, and the other responsible for assembling DTOs from domain objects. The Remote Interface layer is implemented by a class registered in the XML-RPC server as a handler class.

We implemented a client library in *Java* (around 40 classes) using the XML-RPC protocol. The client library Extra Layers layer contains four specific classes to handle text that provide more meaningful methods to NLP tools developers.

We also developed an NLP system to verify the feasibility of our solution. The system is composed of several tools executed sequentially, where each tool uses information produced by previously executed tools. These tools mimic the behaviour of real NLP tools in terms of the input/output data requirements. We have defined the following tools: a text data source creator; a word identification tool, that splits contractions and identifies compound terms, a part-of-speech tagger, a sentence boundary identifier, a syntactic parser, a pos-syntactic parser, and an English-to-Portuguese translator, which keeps both the source text and the translated text aligned.

These tools were implemented using the conceptual model as their object model, avoiding some issues which are usually the developer's responsibility, as the definition of an object model, and the definition of an IO interface. Moreover, each tool uses cross-relations to navigate through the different layers. For example, the translator accesses the original English text segments, and using their cross-relations, accesses their part-of-speech tags disambiguated by the syntactic parser. Using that information it generates the corresponding translation.

This implementation fulfils all the requirements defined, allowing the representation of all types of linguistic phenomena, and complies with the requirements being defined by the ISO committee.

5.1 Future Work

We plan to implement more types of primary data sources, e.g. audio files, and thus enable the use by other NLP systems.

The development of some works in our laboratory, such as, character identification in stories, anaphora resolution, semantic analysis, were suffering from the problems identified in this work. We expect that the use of our framework, that allows them to access all the information produced by NLP tools, and to navigate through related information using cross-relations, simplifies their creation.

Another problem regarding the integration of NLP tools consists in the tags used by each tool to classify linguistic elements. Even if the data structure between two tools is the same, if they use different tag sets to classify the linguistic elements, they will be unable to communicate. This problem was addressed in [4]. Since our framework established a single entry point for the assignment of classifications, a conversion between tagsets could be performed to guarantee that inside the repository all tags were represented in the same way. Each tool would indicate which tag set it requires, and receive the information with the proper tags.

We wish to promote this framework as an annotation framework, and to do so, a graphical interface has to be developed to allow the edition of its data. Moreover, some converter modules have to be defined to allow the use of data annotated in other formalisms, for instance, the *Annotation Graphs* formalism, in which, large corpora have already been annotated, and are publicly available.

As for issues that require future research, our framework does not answer two important questions regarding the integration of NLP tools. First, what data must each tool fetch from the repository. For this problem we intend to integrate the repository with a workflow mechanism and a browser. This will enable the creation of NLP systems, by simply selecting from a browser the tools the system should have. The browser will deal with the selection of the data for each tool. Another question is how does an NLP tool interpret the data kept in the repository. Our conceptual model is capable of representing all linguistic information. However, the same linguistic information may be represented in different ways according to its use. For example, if the phonetic transcription of a text is going to be the target of several NLP tools it should be represented as a new data source. On the contrary, the transcription of each word can be represented as a word's attribute if it is not going to be heavily used. In this work we assume that each tool knows the exact representation of the data. This approach might be too restrictive in terms of extensibility. Some research work should be done in this field, namely in the definition of a meta language, that allows each tool to define its data pre-requirements and pos-requirements, and a way to match this information automatically.

Acknowledgements

This paper was partially supported by project POSI/PLP/41319/2001.

References

1. S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. Atlas: A flexible and extensible architecture for linguistic annotation, 2000.
2. Steven Bird and Mark Liberman. A formal framework for linguistic annotation. Technical Report MS-CIS-99-01, Philadelphia, Pennsylvania, 1999.
3. K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Eng.*, 10, 2004.
4. David Manuel Martins de Matos. *Construção de Sistemas de Geração Automática de Língua Natural.* PhD thesis, IST - UTL, July 2005.
5. Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, November 2002.
6. N. Ide, L. Romary, and E. de la. International standard for a linguistic annotation framework, 2003.
7. 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, January 2002.
8. Edward Loper and Steven Bird. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028, 2002.
9. Ulrik Petersen. Emdros - a text database engine for analyzed or annotated text. In *Colling*, 2004.
10. P. Taylor, A. Black, and R. Caley. The architecture of the the festival speech synthesis system, 1998.