A Proposal for the Integration of NLP Tools using SGML-tagged documents

X. Artola, A. Díaz de Ilarraza, N. Ezeiza, K. Gojenola, A. Maritxalar, A. Soroa

Faculty of Computer Science
University of the Basque Country (UPV/EHU)
649 p.k., 20080 Donostia (The Basque Country)
jiparzux@si.ehu.es

Abstract

In this paper we present the strategy used for an integration, in a common framework, of the NLP tools developed for Basque during the last ten years. The documents used as input and output of the different tools contain TEI-conformant feature structures (FS) coded in SGML. These FSs describe the linguistic information that is exchanged among the integrated analysis tools.
The tools integrated until now are a lexical database, a tokenizer, a wide-coverage morphosyntactic analyzer, and a general purpose tagger/lemmatizer. In the future we plan to integrate a shallow syntactic parser.
Due to the complexity of the information to be exchanged among the different tools, FSs are used to represent it. Feature structures are coded following the TEI's DTD for FSs, and Feature Structure Definition descriptions (FSD) have been thoroughly defined.
The use of SGML for encoding the I/O streams flowing between programs forces us to formally describe the mark-up, and provides software to check that these mark-up hold invariantly in an annotated corpus.
A library of Abstract Data Types representing the objects needed for the communication between the tools has been designed and implemented. It offers the necessary operations to get the information from an SGML document containing FSs, and to produce the corresponding output according to a well-defined FSD.

1. Introduction

In this paper we present a strategy followed for the integration of different NLP tools developed for Basque during the last ten years in the IXA research group[1].

SGML, the Standard Generalized Markup Language, provides us with a well-formalized basis for the exchange of linguistic information among the different text analysis tools. TEI-P3 conformant feature structures (Sperberg-McQueen & Burnard, 1994) constitute the representation schema for the different documents that convey the information from one linguistic tool to the next in the analysis chain. So, SGML-coded documents are used as input and output of the integrated tools.

SGML is a well-defined standard for representing structured documents. Its value consists of the fact that it closes off the option of a proliferation of ad-hoc notations and the associated software needed to read and write them. The most important reason for using SGML to encode the I/O streams between programs is that it forces us to formally describe the mark-up used, and that there exists software to check that these mark-up invariants hold in an annotated corpus.

The tools integrated so far are:

1. EDBL, a lexical database for Basque, which at the moment contains more than 75,000 entries (Aduriz *et al.*, 1998a).
2. A tokenizer that identifies tokens from the input text.
3. *Morpheus*, a wide-coverage morphosyntactic analyzer for Basque (Alegria *et al.*, 1996). It attaches to each input word form all its possible interpretations. The result is a set of possible morphosyntactic readings of a word in which each morpheme is associated with its corresponding features in the lexicon: category, subcategory, declension case, number, and definiteness, as well as its syntactic function (Karlsson *et al.*, 1995) and some semantic features. It is composed of several modules such as:
   - A segmentizer, which splits up a word into its constituent morphemes.
   - A morphosyntactic analyzer (Aduriz *et al.*, 2000), whose goal is to group the morphological information associated with each morpheme obtaining the morphological information of the word form considered as a unit. This is an important step in our analysis process due to the agglutinative character of Basque.
   - A recognizer of multiword lexical units (MWLUs), which performs the morphosyntactic analysis of multiword units present in the text (Aduriz *et al.*, 1996).
4. *EusLem*, a general-purpose tagger/lemmatizer. (Ezeiza *et al.*, 1998).

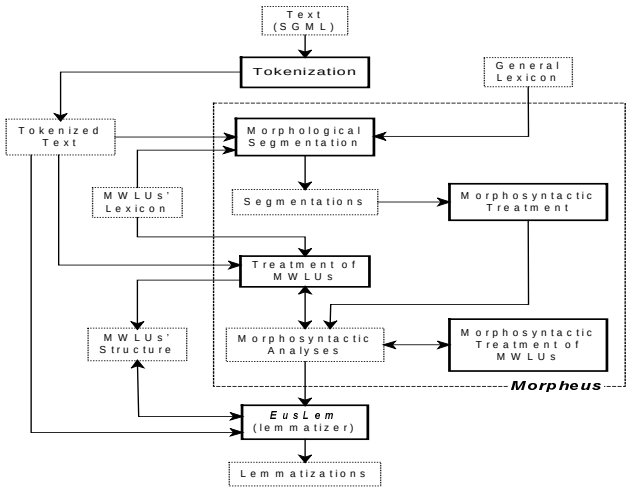

Figure 1. Information flow between the linguistic analysis tools.

[1] URL: http://ixa.si.ehu.es

In the future we plan to integrate other tools currently under development, such as a shallow syntactic analyzer, a lexical disambiguator, etc. (Aduriz *et al.*, 1998b).

The main goals of this methodology of integration are twofold. Firstly, to formally define the input and output formats for the whole linguistic analysis chain. Secondly, we would like to propose a formal and standardized framework for the exchange of information that will make effective the communication between the analysis tools.

The information flow between the different tools in the chain is shown in Figure 1. Having an SGML-tagged text as the input document, the tokenizer produces, as output, the list of tokens identified in the input text (*Tokenized Text* in the figure).

```
<p>   <!-------------- kartera, non-standard noun unit -------------------->
  <fs type="Nouns">
    <f name="Key">
      <fs type="Key">
        <f name="Entry"><str>kartera</str></f>
        <f name="Homograph-Id"><nbr value="0"></f>
      </fs>
    </f>
    <f name="POS"><sym value="NOUN"></f>
    <f name="Standardization-Status">
      <fs type="Non-Standard-FS">
        <f name="Non-Standard-Code"><sym value="LD_MA"</f>
        <f name="Variant"><minus></f>
        <f name="Corresponding-Standards" org="set">
          <fs type="Key">
            <f name="Entry"><str>diru-zorro</str></f>
            <f name="Homograph-Id"><nbr value="0"></f>
          </fs>
        ...
        </f>
      </fs>
    </f>
    <f name="Structure">
      <fs type="OneWordUnit-Features-FS">
        <f name="Morphotactics" org="set">
          <fs type="Morphotactics-FS">
            <f name="TWOL-Form"><str>karterA</str></f>
            <f name="Continuation-Class"><str>I</str></f>
            <f name="Lexicon"><str>et_izenak</str></f>
          </fs>
        </f>
      </fs>
    </f>
    <f name="Noun-Features">
      <fs type="Noun-Features-FS">
        <f name="SUBCAT"><sym value="COMMON"></f>
        <f name="ANIM"><minus></f>
        <f name="COUNT"><plus></f>
        <f name="MEAS"><plus></f>
        <f name="PLU"><minus></f>
      </fs>
    </f>
  </fs>
</p>  <!------------------------ kartera ---------------------------->
```

Figure 2. Output of the unit *kartera* (non-standard Basque for *wallet*) as exported from the lexical database.

The main source of lexical information in the system is the lexical database, which is considered as a permanent store developed and maintained by an independent process. The lexical information in the database needs to be extracted so as to compile it into *Morpheus* or any other program that might need it. Hence, the output of the lexical database is carried out in two steps, so distinguishing the exportation of the general lexicon (single-word units) and that of MWLUs (Figure 1)[2].

The segmentizer in *Morpheus*, taking as input the general lexicon exported from EDBL, performs the morphological segmentation of the input text, splitting up each token into its constituent morphemes. A more comprehensive morphosyntactic treatment of words follows the segmentation process and, finally, multiword lexical units are treated in two stages (*Treatment of MWLUs* and *Morphosyntactic Treatment of MWLUs*), taking as input in this case the MWLUs' lexicon.

Finally, the general-purpose lemmatizer, *EusLem*, refines the previous documents eliminating contextually incorrect analyses and reducing the ambiguity on single and multiword units.

2. Using feature structures to interchange complex linguistic information

Due to the complexity of the information to be represented we decided to use feature structures. The use of feature structures quickly spread to other domains within linguistics since Jakobson (1949) first used them

[2] As can be seen in Figure 1, the MWLUs' lexicon is already needed in the Morphological Segmentation process, in order to be able to recognize in the text MWLU's constituents that are not by themselves entries of the lexical database.

for the representation of phonemes. The ability of feature structures to serve as a general-purpose linguistic metalanguage led us to use them as the basis of our encoding.

The feature structures in the integrated system are coded following the TEI's DTD for FSs, and they fulfill the Feature Structure Definitions (FSD) that have been thoroughly described for all the inputs/outputs in the tool pipeline.

Here are two examples. The first one (Figure 2) corresponds to the non-standard noun entry *kartera* (Basque for *wallet*) exported from the lexical database into the general lexicon.

```
<tei.2>
...
  <p>
    <fs type="Segmentation"> <!-- first segmentation: softwaregile + ek -->
      <f name="Form"><str>softwaregileek</str></f>
      <f name="Lemma-Morphemes" org="list">
        <fs type="Lemma">
          <f name="TWOL"><str>softwaregile</str></f>
          <f name="Unit">
            <fs type="Key">
              <f name="Entry"><str>softwaregile</str></f>
              <f name="Homograph-Id"><nbr value="0"></f>
            </fs>
          </f>
          <f name="Features">
            <fs type="Feature-List">
              <f name="POS"><sym value="NOUN"></f>
              ...
              <f name="ROOT">
                <fs type="Key">
                  <f name="Entry"><str>software</str></f>
                  <f name="Homograph-Id"><nbr value="0"></f>
                </fs>
              </f>
              <f name="SUFL" org="list">
                <fs type="Key">
                  <f name="Entry"><str>gile</str></f>
                  <f name="Homograph-Id"><nbr value="1"></f>
                </fs>
        ...
        <fs type="Morpheme">
          <f name="TWOL"><str>ek</str></f>
          <f name="Unit">
            <fs type="Key">
              <f name="Entry"><str>ek</str></f>
              <f name="Homograph-Id"><nbr value="1"></f>
            </fs>
          </f>
          <f name="Features">
            <fs type="Feature-List">
              <f name="POS"><sym value="DEC"></f>
              <f name="CASE"><sym value="ERG"></f>
    ...
    </fs>  <!------------------ end of first segmentation   ------------------>
  </p>
  <p>
    <fs type="Segmentation"> <!-- second segmentation: software + gile + ek -->
    ...
    </fs>  <!---------------- end of second segmentation --------------------->
  </p>
...
</tei.2>
```

Figure 3. Multiple segmentations for *softwaregileek*.

The second example (Figure 3) represents a partial view of the output of the segmentizer for the derivative word form *softwaregileek* (Basque term for *software makers* in the ergative case). The word form *softwaregileek* can be split up in two different ways:

a) *softwaregile* + *ek*
b) *software* + *gile* + *ek*

The first one reflects the case in which *softwaregile* is analyzed as a lexicalized term (the information about the constituents of the word then comes from EDBL). As can be seen in the figure, in this case the two constituents of the word are represented by two `<fs>` elements: `<fs type="lemma">` and `<fs type="morpheme">`[3]. In the second case (not in the figure), the number of constituents is three: two parts of the lemma (the root and the lexical suffix) and one declension morpheme.

A linguistic analysis may consist of many different types of `<fs>` elements, each of which may group together different types of `<f>` elements. In order to distinguish among the different types of `<fs>` elements, a type attribute that specifies the FS type is provided (for

[3] An `<fs>` element represents a feature structure. It is composed by a set of features and their values, represented by `<f>` elements. The element `<Lemma>` is used to distinguish the lemma-constituent morphemes from the inflection morphemes, which are described by means of `<Morpheme>` elements.

instance, see the "lemma" and "morpheme" FS types in Figure 3).

As a last example (Figure 4), we show a partial view of the output of the lemmatizer for the same word form (*softwaregileek*), in which the resultant FS shows a much simpler structure, and where one of the interpretations has been removed by the morphological disambiguation process (part of *EusLem*).

```
<!-- output of EusLem (.lem.sgm): softwaregileek -->
<tei.2>
...
  <p>
    <fs id="IZE-ARR-1905" type="Lemmatization">
      <f name="Form"><str>softwaregileek</str></f>
      <f name="Lemma"><str>softwaregile</str></f>
      <f name="Morphological-Features">
        <fs type="TopLevel-Feature-List">
          <f name="POS"><sym value="NOUN"></f>
          <f name="SUBCAT"><sym value="COMMON"></f>
          <f name="ANIM"><plus></f>
          <f name="ROOT"><str>software</str></f>
          <f name="SUFL" org="list">
            <str>gile</str>
          </f>
          <f name="CASE"><sym value="ERG"></f>
          <f name="NUM"><sym value="P"></f>
          <f name="DET"><sym value="DET"></f>
          <f name="SYNTFL" org="list">
            <sym value="@SUBJ">
    ...
    </fs>
  </p>
...
</tei.2>
```

Figure 4. Disambiguated output of the lemmatizer.

3. An I/O stream format between programs

The information flow in Figure 1 is given in more detail in Figure 5. Shown there is the integration of the lexical database, the tokenizer, *Morpheus* —including the modules that perform the morphological segmentation, morphosyntactic treatment, treatment of MWLUs and morphosyntactic treatment of MWLUs—, and *EusLem*, emphasizing that the communication among the different processes is made by means of SGML documents (*.sgm* files). As in Figure 1, here also thick line-border rectangles are used to represent processes. The rest of this section is devoted to the description of these processes in sequence.

Having an SGML-tagged input text file (*.sgm*), the tokenizer takes this file and creates, as output, a *.w.sgm* file, which contains the list of the tokens recognized in the input text (see also Figure 6). The tokenized text (*.w.sgm*) is of great importance in the rest of the analysis process, in the sense that it intervenes as input for different processes.

After the tokenization process, the segmentizer takes as input the tokenized text and the general lexicon issued from the lexical database, and will produce two documents: a *.seg.sgm* file, which contains the different segmentation analyses (FSs describing the different morphemic segments found in each word token), and a *.seglnk.sgm* file containing the links between the tokens in the *.w.sgm* file and their corresponding analyses (one or more) in the *.seg.sgm* file (analogous to the *.lemlnk.sgm* file in Figure 6).

After that, the morphosyntactic treatment module included in *Morpheus* takes as input the output of the segmentation process to produce its result: the collection of morphosyntactic analyses (FSs) corresponding to the input text (*.morf.sgm*). The morphosyntactic treatment module processes the *.seglnk.sgm* file issued in the previous phase producing a *.morflnk.sgm* file that contains now the links between the tokens in the *.w.sgm* file and their corresponding analyses (one or more) in the *.morf.sgm* file. This file will be later enriched by the MWLUs' treatment module. This module, included also in *Morpheus,* performs the processing of multiword lexical units, and produces a *.mwlnk.sgm* document which describes, by means of a collection of <link> elements (see Figure 6), the structure of the MWLUs identified in the text. This module has obviously access to: (a) the *.morf.sgm* file, in order to be able to remove some single-word analysis FSs in the cases that MWLUs are unambiguously recognized, and (b) the *.morflnk.sgm* file, into which it will add the links between the *.mwlnk.sgm* file and the *.morf.sgm* file[4]. The treatment of MWLUs is finally completed by the MWLUs' morphosyntactic treatment module.

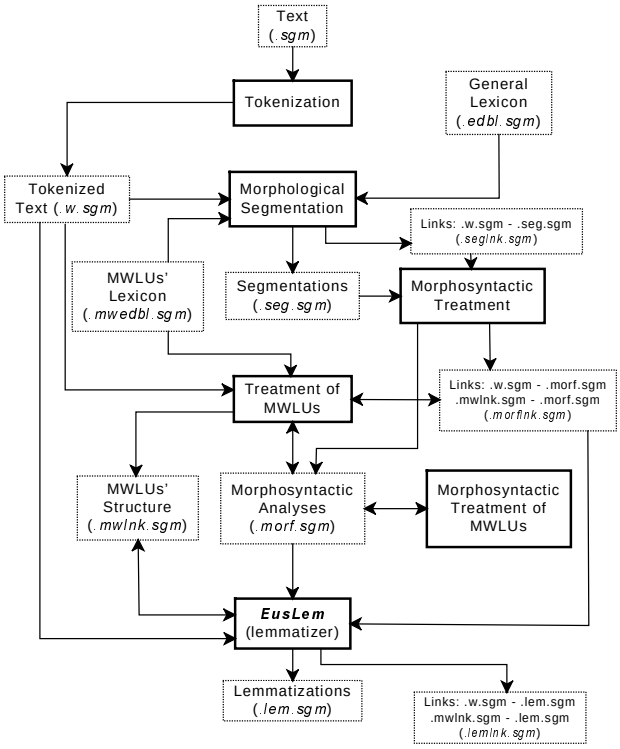

Figure 5. Detailed information flow between the analysis tools.

The file containing the morphosyntactic analysis FSs, the *.mwlnk.sgm* file, the *.morflnk.sgm* file, and the output of the tokenizer constitute the input of the lemmatizer.

[4] The links between the *.mwlnk.sgm* file and the *.morf.sgm* file represent the MWLU analyses found in the text. In this case, they do not link tokens with their analyses, but a MWLU's structure denoting links (in the *.mwlnk.sgm* file) with its corresponding analyses in the *.morf.sgm* file.

Figure 6. Output of the lemmatizer: a sample of the four-document set.

The lemmatizer produces two more files: *.lem.sgm* that contains the lemmatization FSs corresponding to the input text, and *.lemlnk.sgm* that stores the links between the tokens and their corresponding lemmatization analyses, plus, in the case of MWLUs, the links between the MWLU's formation denoting links (*.mwlnk.sgm*) and their corresponding lemmatization analyses[5]. It is also capable of updating the *.mwlnk.sgm* file if, due to the disambiguation performed, it has to remove some of the links previously included in it.

3.1. Details about the linguistic information

As has been shown in the preceding section, the output of each one of the analysis tools may be seen as composed of several documents that, in the most complex case, constitute a four-document set.

As an example, we show in Figure 6 the lemmatizer's output. It can be seen as composed of four SGML-coded documents:

- Text elements found in the input:
  1) The list of lexical instances or single-word tokens issued from the tokenizer (*.w.sgm*). This document is, in fact, part of the input of the lemmatizer, but it must be also seen as a part of its resulting set of documents in order to be able to, for example, display it in an integrated fashion.
  2) The document that describes the MWLU's structure (*.mwlnk.sgm*). It can be considered as an enrichment of the information contained in the list of tokens, in the sense that it constitutes the collection of "multiword tokens" identified in the input.

[5] In fact, the lemmatizer also gives some information about the syntactic functions corresponding to the word and multiword tokens recognized in the text. This information comes in part from the lexicon, and it is enriched in the lemmatization process by applying Constraint Grammar mapping rules. It is represented by means of two documents, a library of the different syntactic functions (*.sf.sgm*) and the corresponding link file (*.sflnk.sgm*) that attach, in this case, the token, the lemmatization identifier, and the syntactic function identifier. The purpose of this information is obviously to be used in the syntactic analysis of the sentence that is outside the scope of this paper; because of that, these documents are not represented in the figure.

- Analysis collection:
  3) The lemmatization analyses produced (*.lem.sgm*). It constitutes a library of the different analyses (FSs) found in the given input text.
- Links from the text elements to their corresponding analysis or analyses:
  4) Finally, the document *.lemlnk.sgm*, which contains the links between the single-word tokens in the 1st file and their corresponding analyses included in the 3rd, and the MWLU's in the 2nd and their analyses in the 3rd.

So, we speak about a four-document set system that gives, as pointed out in (Ide & Véronis, 1995), more independence and flexibility to the different processes, and greater facilities for their integration (see Figure 6).

It is interesting to note that, in the case of *EusLem* —as well as in that of *Morpheus*—, the collection of links between the text elements and their corresponding analysis FSs (*.lem.sgm* file) is divided into two different parts, grouped into two different `<linkgrp>` elements: (1) the links between single-word text elements and their analyses (`<linkgrp type='w-lem'>`), and (2) the links between the multiword text elements and their analyses (`<linkgrp type='mwlnk-lem'>`).

As has been mentioned above, the 3rd document can be considered as a library that stores the different analyses (FSs) found in the text. It is clear that, before adding a new analysis FS to this document, the lemmatizer's back-end will always check whether it has already been stored because of a previous occurrence of the same text element in the input. In this case, the lemmatizer will not add it again (it will just add a new link to the 4th file).

We are already building an environment for manual disambiguation of the output of the lemmatization process, which will facilitate the disambiguation task for the linguists, providing them with a graphical interface with hypertextual facilities, based on the four-document set and on the input text.

FS Abstract Data Type[6]

```
FS::
 Id: FSId
 Type: FSType
 Features: FList

Operations
FEATURE_STRUCTURE ([Id: FSId]; Type: FSType; [Feature_List: FList]) FStruct: FS
 pre
 post Id (FStruct) = Id &
      Type (FStruct) = Type &
      (Features (FStruct) = [] or Features (FStruct) = Feature_List)
ADD_F (FStruct1: FS; F1: F) FStruct2: FS
 pre
 post Features (FStruct2) = Features (FStruct1) • F1
ID (FStruct: FS) Id: FSId
 pre
 post Id = Id (FStruct)
TYPE (FStruct: FS) Type: FSType
 pre
 post Type = Type (FStruct)
FEATURES (FStruct: FS) Feature_List: FList
 pre
 post   Feature_List = Features (FStruct)
ID_MODIFY (FStruct1: FS; Id: FSId) FStruct2: FS
 pre
 post Id = Id (FStruct2)
FEATURE_VALUE (FStruct: FS; N: FName) V: [FValue]
 pre
 post (exists I in inds Features (FStruct) |
        Name (Features (FStruct) (I)) = N &
        (V = Value (Features (FStruct) (I))) or V = nil)
COPY (FStruct1: FS) FStruct2: FS
 pre
 post  FStruct2 =  FStruct1l
EQUAL (FStruct1: FS; FStruct2: FS) B: Boolean
 pre
 post B = (FStruct2 =  FStruct1)
```

Figure 7. Formal specification of the `FS` Abstract Data Type.

4. A repository of Abstract Data Types on Feature Structures

In this section we present the library designed in order to facilitate the work with the FSs describing the linguistic information in our integrated system. The different elements used in it have been characterized as Abstract Data Types (ADT). As is well known, the theory underlying ADTs gives the user a way to describe which kind of values belong to a particular type, and to

[6] `F`, `FName`, and `FValue` are ADTs specified in the same way.

determine precisely the set of operations that can be performed on them.

As a result of the analysis of the characteristics and structure of the different data used as input and output of the analysis tools, we have identified the different ADTs intervening, and we have consequently implemented several library modules to encapsulate them. The set of packages implemented provides internal representation and operations for the following types among others: `FS`, `FSD`, `Link`, `MWLink`, `Feature`, `Value`, `FSId`, `FSList`, `FList`, `LList`, and so on.

These packages offer the necessary operations the different tools need to perform their task when recognizing the input and producing their output.

These functions allow:

a) getting the necessary information from an SGML document containing tokens, links, multiword structure links or FSs;
b) producing with ease the corresponding output according to a well-defined SGML description.

In Figure 7 we show a partial view on the specifications of the `FS` (feature structure) ADT. Values of this type are represented by triples (`Id`, `Type` and `Features`). Each component of the triple is an attribute whose value belongs to another ADT. So, the `Id` component belongs to the `FSId` ADT, the `Type` component to the `FSType` ADT, and, finally, the `Features` component to the `FList` (features list) ADT. Each one of these ADTs has been specified elsewhere in the same way.

As can be seen in Figure 7, the operations defined in the `FS` ADT are the following:

- `FEATURE_STRUCTURE` (type's constructor): builds up an `FS` object given the type and, optionally, an identification and a feature list.
- `ADD_F`: adds a new feature to the feature structure.
- `ID`, `TYPE` and `FEATURES`: operations that give access to the feature structure attributes.
- `EQUAL`, `COPY` and so on: perform different actions on the feature structure. The first one examines two FSs saying whether they represent the same object; `COPY` will reproduce an FS object in another FS.

The ADTs' library has been implemented in C++, following an object-oriented methodology. For the implementation of the different operations we make use of the LT NSL system (Mckelvie *et al.,* 1997), a tool architecture for SGML-based processing of text corpora. The current release of the library works on Unix (Solaris 2.5).

5. Conclusion and future work

We have presented our methodology for the integration of linguistic tools, developed following the TEI P3 guidelines.

We have mentioned above the environment for manual disambiguation of the lemmatizer's output that is under development. We are also considering the feasibility of building general front- and back-end modules for the analysis tools, which will take as input the specific FSDs for each input/output. A schematic view of the integration of these general modules with a particular tool can be seen in Figure 8.

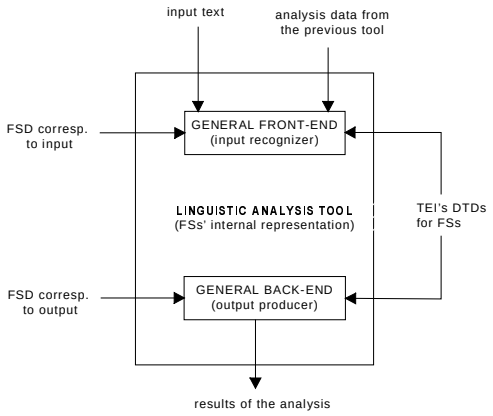

Figure 8. Schematic view of a linguistic analysis tool with its general front- end back-ends.

The use of SGML as an I/O stream format between programs has, in our opinion, the following advantages:

a) It is a well-defined standard for the representation of structured texts that provides a formal framework for the internal processing.
b) It provides widely recognized facilities for the exchange of data: given the DTD, it is easy to process any conformant document.
c) It forces us to formally define the input and the output of the tools used for the linguistic analysis of the text.
d) It facilitates the future integration of new tools into the analysis chain.
e) Pieces of software are available for checking the syntactic correctness of the documents, information retrieval, modifications, filtering, and so on. It makes it easy to generate the information in different formats (for processing, printing, screen-displaying, publishing in the web, or translating into other languages).
f) Finally, it allows us to store different analysis sets (segmentations, complete morphosyntactic analyses, lemmatization results, and so on) linked to a tokenized piece of text, in which any particular analysis FS will not have to be repeated.

6. Acknowledgements

We would like to thank Prof. Jon Patrick for his valuable comments on an earlier draft of this paper.

This work is being carried out in the project G19/99, supported by the University of the Basque Country.

7. References

Aduriz I., Aldezabal J.M., Artola X., Ezeiza N.,Urizar R., 1996. Multiword Lexical Units in EUSLEM: a lemmatiser-tagger for Basque. In *Proc. in Computational Lexicography (Complex'96)*, 1-8. Linguistics Institute, Hungarian Academy of Sciences. Budapest (Hungary).

Aduriz I., Agirre E., Aldezabal I., Alegria I., Ansa O., Arregi X., Arriola J.M., Artola X., Díaz de Ilarraza A., Ezeiza N., Gojenola K., Maritxalar A., Maritxalar M., Oronoz M., Sarasola K., Soroa A., Urizar R., Urkia M., 1998. A Framework for the Automatic Processing of Basque. In *Proc. of the First Int. Conf. on Language Resources and Evaluation.* Granada (Spain).

Aduriz I., Aldezabal I., Ansa O., Artola X., Díaz de Ilarraza A., Insausti J. M., 1998. EDBL: a Multi-Purposed Lexical Support for the Treatment of Basque. In *Proc. of the First Int. Conf. on Language Resources and Evaluation*, vol II, 821-826. Granada (Spain).

Aduriz I., Agirre E., Aldezabal I., Arregi X., Arriola J.M., Artola X., Gojenola K., Maritxalar A., Sarasola K., Urkia M., 2000. A Word-Level Morphosyntactic Grammar For Basque. In *Proc. of the Second Int. Conf. on Language Resources and Evaluation*. Athens (Greece).

Alegria I., Artola X., Sarasola K., Urkia M., 1996. Automatic morphological analysis of Basque. *Literary & Linguistic Computing*, 11, no. 4, 193-203.

Ezeiza N., Aduriz I., Alegria I., Arriola J.M., Urizar R., 1998. Combining Stochastic and Rule-Based Methods for Disambiguation in Agglutinative Languages. In *Proc. COLING-ACL'98*, 10-14. Montreal (Canada).

Ide N., Véronis J. (eds.), 1995. *Text Encoding Initiative. Background and Context.* Kluwer Academic Pub.

Jacobson R., 1949. The Identification of Phonemic Entities. *Travaux du Cercle Linguistique de Copenhague*, 5, 205-213.

Karlsson F., Voutilainen A., Heikkilä J., Anttila A., 1995. *Constraint Grammar: A Language-independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Mckelvie, D., Brew, C., Thompson, H., 1997. Using SGML as a basis for Data-Intensive NLP. In *Proc. ANLP'97*. Washington (USA).

Sperberg-McQueen C.M., Burnard L., 1994. *Guidelines for Electronic Text Encoding and Interchange*. TEI P3 Text Encoding Initiative.