# A Class Library for the Integration of NLP Tools:

# Definition and implementation of an Abstract Data Type Collection for the manipulation of SGML documents in a context of stand-off linguistic annotation

**X. Artola, A. Díaz de Ilarraza, N. Ezeiza, K. Gojenola, G. Hernández, 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 a program library conceived and implemented to represent and manipulate the information exchanged in the process of integration of NLP tools. It is currently used to integrate the tools developed for Basque processing during the last ten years at our research group. In our opinion, the program library is general enough to be used in similar processes of integration of NLP tools or in the design of new applications built on them. The program library constitutes a class library that provides the programmer with the elements s/he needs when manipulating SGML documents in a context of stand-off linguistic annotation, where linguistic analyses obtained at different phases (morphology, lemmatization, processing of multiword lexical units, surface syntax, and so on) are represented by well-defined typed features structures. Due to the complexity of the information to be exchanged among the different tools, feature structures (FS) are used to represent it. Feature structures provide us with a well-formalized basis for the exchange of linguistic information among the different text analysis tools. Feature structures are coded in SGML following the TEI's DTD for Fs, and Feature-System Declarations (FSD) have been thoroughly specified. So, TEI-P3 conformant feature structures constitute the representation schema for the different documents that convey the information from one linguistic tool to the next in the language processing chain. The tools integrated so far are a lexical database, a tokenizer, a wide-coverage morphosyntactic analyzer, a general purpose tagger/lemmatizer and a shallow syntactic parser. The type of information contained in the documents exchanged among these tools has been analyzed and characterized using a set of Abstract Data Types.

## 1. Introduction

In this paper we present a program library conceived and implemented to represent and manipulate the information exchanged in the process of integration of NLP tools. It is currently being used to integrate the tools developed for Basque processing during the last ten years at our research group. In our opinion, the program library is general enough to be used in similar processes of integration of NLP tools or in the design of new applications built on them, given that feature structures are used to represent linguistic information.

The program library constitutes a class library that provides the programmer with the elements s/he needs when manipulating SGML documents in a context of stand-off linguistic annotation, where linguistic analyses obtained at different phases (morphology, lemmatization, processing of multiword lexical units, surface syntax, and so on) are represented by well-defined typed feature structures.

Due to the complexity of the information to be exchanged among the different tools, feature structures (FS) are used to represent it. Feature structures provide us with a well-formalized basis for the exchange of linguistic information among the different text analysis tools. Feature structures are coded in SGML following the TEI's DTD for FSs, and Feature-System Declarations (FSD) have been thoroughly specified. So, TEI-P3 conformant feature structures constitute the representation schema for the different documents that convey the information from one linguistic tool to the next in the language processing chain.

The tools integrated so far are:

1. *EDBL*, a lexical database, which at the moment contains more than 80,000 entries (Aldezabal *et al.*, 2001).
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 functions (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).
5. A shallow syntactic analyzer (Aduriz et al., 1998b) that identifies noun phrases and verbal chains.

The figure 1 shows the linguistic tools integrated and the information flow among them.

An overview of the I/O stream format between programs is presented in the next section. Section 3 explains by means of an example the use of feature structures to interchange complex linguistic information and gives some details of our representation. Section 4 describes the library of programs as a repository of Abstract Data Types on Feature Structures and other entities related to the SGML documents to be manipulated. The final section presents some conclusions.

## 2. An I/O stream format between programs

The figure shows 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—, *EusLem*, and the shallow syntactic analyzer 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.

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. 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.

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, 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[1]. The treatment of MWLUs is finally completed by the MWLUs' morphosyntactic treatment module.

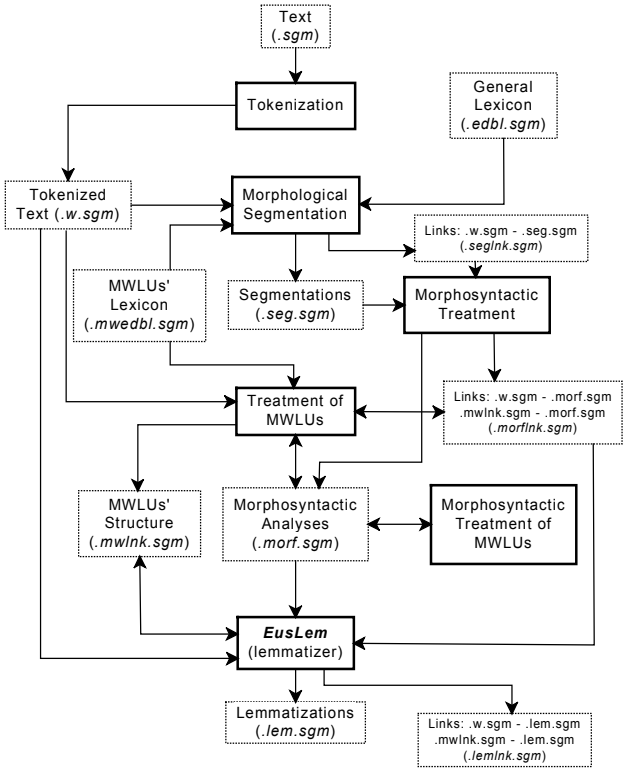


Figure 1. Detailed information flow between the analysis tools.

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.

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[2]. 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. Using feature structures to interchange complex linguistic information

[1] 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.

[2] 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.

We decided to use feature structures to represent the information to be interchanged among the linguistic tools. The use of feature structures quickly spread to other domains within linguistics since Jakobson (1949) first used them 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-System Declarations (FSD) that have been thoroughly described for all the inputs/outputs in the tool pipeline.

The example (Figure 2) 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 instance, see the `"lemma"` and `"morpheme"` FS types in Figure 3).

As a last example (Figure 3), 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*).

## 3.1. Our representation

A key issue in software development in NLP tasks is the definition of a framework for linguistic knowledge representation. Such a framework has to satisfy needs entailed by the different tools and has to be general enough (Basili et *al.*, 1998). It is not trivial to adopt a formalism to represent this information. Different approaches have been considered for this task. Some of them as ALEP (Simkins, 1994), Advanced Language Engineering Platform, can be considered the first integrating environment for NLP design. All the components (linguistic information, processing modules and resources) are homogeneously described using ALEP User Language (AUL) based on a DAG formalism. Others, like GATE (Cunningham, *et al*., 97) and Calypso (Zajac, 1998) represent textual information by using the notion of textual annotation firstly introduced in the TIPSTER project (Grisham., 1997). In other solutions, the linguistic information is added in the form of markup, SGML/XML, like in LT-NSL (Thompson *et al*. 1996) and Sissa (Lavelli *et al*. 2001). XML is acquiring more and more relevance in this area, as the solutions using it are becoming very popular, e.g. XCES (Ide *et al*. 2000), an XML-based encoding standard for linguistic corpora; LT-XML, the XML version of LT-NSL; ATLAS (Bird *et al*. 2000), an architecture for linguistic annotation. In our case the representation is modeled as a collection of abstract data types that have been implemented as classes in C++, using LT-NSL functions. As a result, we have built a library of programs designed in the process of the integration of linguistic tools, developed following the TEI P3 guidelines. These Abstract Data Types constitute a library of programs that follow the TEI P3 guidelines and are used to integrate the different linguistic tools mentioned earlier.

Within a framework of stand-off linguistic annotation, the output of each of the analysis tools may be seen as composed of several documents that, in our most complex case, constitute a five-document set. Looking at the characteristics of the information to be manipulated, different groups of documents have been identified. We define abstract data types to represent each information type. The abstract data types specify the possible values and their operations (their behavior). Next we will show the information types identified:

1. Text elements found in the input: the list of lexical instances or single-word tokens issued from the tokenizer. They are represented by the SGML <w> element (with its correspondent W class). W-class objects are groups together in a WL class.
2. Description of the structure of multiword lexical units: the collection of "multiword tokens" identified in the input. The MWStruct class represents the constituents of a multiword units. MWStructL represents the list of MWStruct objects.
3. Analysis collection: A library of the analyses (FSs) corresponding to the tokens in the given input text through the different analysis phases. Several classes have been defined here: FS (feature structure class), FL (list of features of a feature structure), F (feature class), FVL (the list of values of a feature), FValue (the value of a feature), and so on. A list of <fs> elements is represented by the class FSL. Figure 5 w shows the set of classes (abstract data-types) defined in this group. This information is found in the following files: *.seg.sgm, .morf.sgm and .lem.sgm.*
4. Links from the text elements to their corresponding analysis or analyses. The list of all links will be a list of <link> elements, identified in our system by the LinkL class. The files

[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.

```
<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 2. Multiple segmentations for *softwaregileek*.

*.seglnk.sgm, .morflnk.sgm, .lemlnk.sgm* contain the different links created as a result of the output of some of the tools described earlier.

5. Documents: collections of text elements —single and multiword—, analyses, and links. We differentiate four different document types at the moment : A document containing a list of text elements (*WSGMDoc)*, a document containing a list of analysis (*ASGMDoc*), a document containing a list of links (*LnkSGMDoc*)and a document containing a list of multiword unit (*MWSGMDoc*).
   All of them contain, apart of their specific information, data about the characteristics of the document such as date of creation, author, sources, relation with other documents etc.

```
<!-- 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 3. Disambiguated output of the lemmatizer.

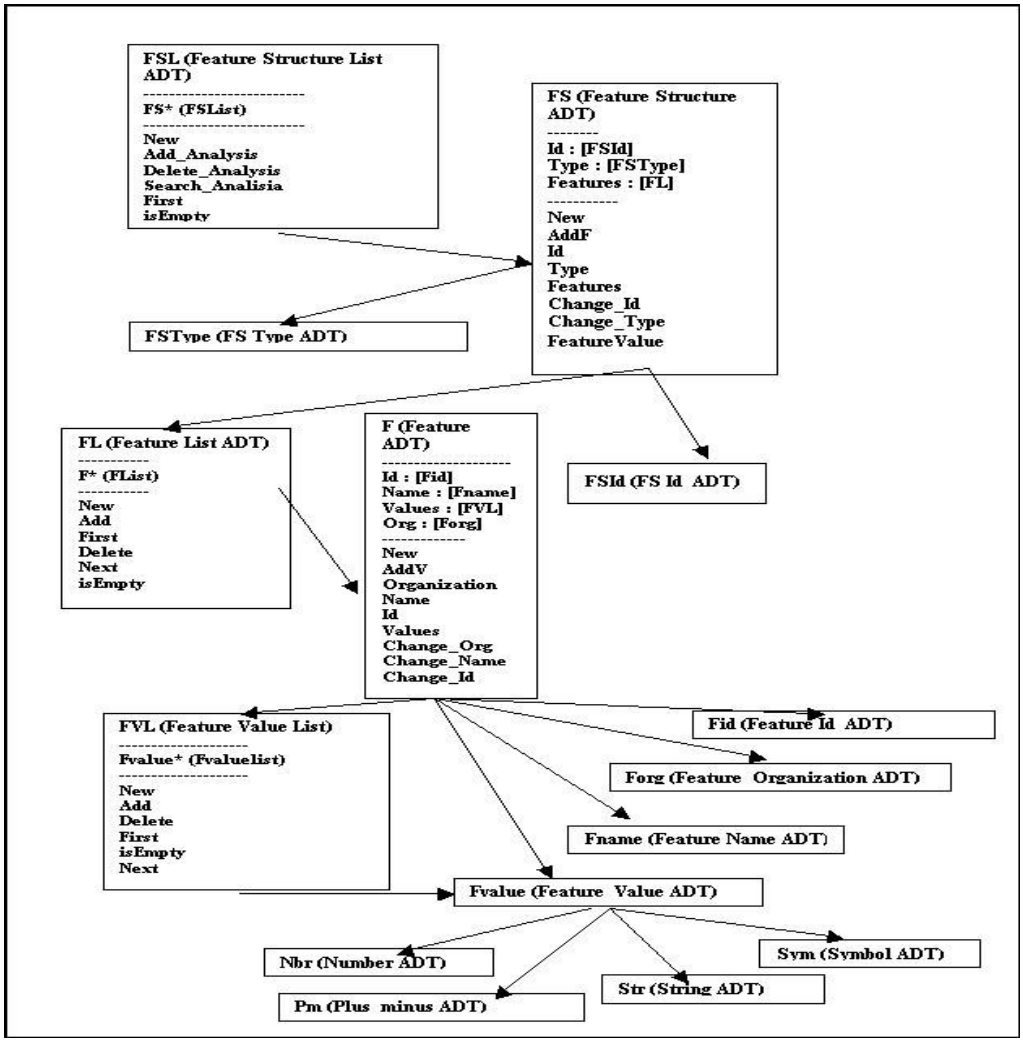


Figure 4. Set of classes (abstract data-types) defined in the analysis collection.

## 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 and related classes 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 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 5 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 the figure, 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 to 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).

```
FS Abstract Data   1
FS::
 Id: FSId
 Type:
 Features:

Operations
FEATURE_STRUCTURE ([Id: FSId]; Type: FSType; [Feature_List:
FStruct: FS
 pre
 post Id (FStruct) = Id & Type (FStruct) =
      (Features        = [] or Features (FStruct) =
ADD_F (FStruct1: FS; F1: F) FStruct2:
 pre
 post Features (FStruct2) = Features       • F1
ID (FStruct: FS) Id:
 pre
 post Id = Id
TYPE (FStruct: FS) Type:
 pre
 post Type = Type  truct)
FEATURES (FStruct: FS) Feature_List:
 pre
 post   Feature_List = Features
ID_MODIFY (FStruct1: FS; Id: FSId) FStruct2:
 pre
 post Id = Id
FEATURE_VALUE (FStruct: FS; N: FName) V:
 pre
 post (exists I in indseatures (FStruct) | Name (Features
(I)) = N &
       (V = Value (Features (FStruct) (I))) or V =
COPY (FStruct1: FS) FStruct2:
 pre
 post  FStruct2 =
EQUAL (FStruct1: FS; FStruct2: FS) B:
 pre
 post B = (FStruct2 = Struct1)
```

Figure 5. Formal specification of the `FS` Abstract Data Type.

## 5. Conclusion and future work

A central issue in the software development in NLP is the definition of a framework for linguistic knowledge representation. Such representation has to satisfy different needs and has to be general enough.

*Programming applied to Natural Language Processing held as part of EUROLAN'01 Summer School*. Iasi (Romania).

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.

Simkins N.K., 1994. An Open Architecture for Language Engineering. In *First Language Engineering Convention*. Paris (France)

Thompson H., McKelvie D., Finch S., 1996. The Normalised SGML Library LT NSL version 1.4.6. *Technical Report, Language Technology Group, University of Edinburgh*. http://www.ltg.ed.ac.uk/software/lt nsl.html

Zajac, R., 1998. Reuse and Integration of NLP Components in the Calypso Architecture. In *Workshop on Distributing and Accessing Linguistic Resources* . Granada, (Spain).