Preprocessing and Tokenisation Standards in DELPH-IN Tools

Benjamin Waldron[*], **Ann Copestake**[*], **Ulrich Schäfer**[†], **Bernd Kiefer**[†]

[*]University of Cambridge Computer Laboratory
15 JJ Thomson Avenue
Cambridge CB3 0FD, UK
{bmw20, aac10}@cl.cam.ac.uk

[†]Language Technology Lab
German Research Center for Artificial Intelligence (DFKI) GmbH
Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany
{ulrich.schaefer, bernd.kiefer}@dfki.de

Abstract

We discuss preprocessing and tokenisation standards within DELPH-IN, a large scale open-source collaboration providing multiple independent multilingual shallow and deep processors. We discuss (i) a component-specific XML interface format which has been used for some time to interface preprocessor results to the PET parser, and (ii) our implementation of a more generic XML interface format influenced heavily by the (ISO working draft) Morphosyntactic Annotation Framework (MAF). Our generic format encapsulates the information which may be passed from the preprocessing stage to a parser: it uses standoff-annotation, a lattice for the representation of structural ambiguity, intra-annotation dependencies and allows for highly structured annotation content. This work builds on the existing Heart of Gold middleware system, and previous work on Robust Minimal Recursion Semantics (RMRS) as part of an inter-component interface. We give examples of usage with a number of the DELPH-IN processing components and deep grammars.

1. Introduction

The standard assumption about natural language parsing is that input data must be initially divided into chunks suitable for analysis (generally sentences) with each chunk further split into tokens. The tokenization step is generally defined to return a sequence of strings. There are several complications to this simple story when we consider processing real documents. Data which we wish to parse comes in many formats, possibly textual (e.g., raw text, HTML, SGML or XML markup, PDF), but possibly audio or video. Markup on the original input which is irrelevant for sentence parsing may nevertheless be useful for further processing (e.g., paragraph boundaries) and thus should be preserved so that it is available alongside the information extracted from parsing. Furthermore, it may be impossible in general for tokenization to produce a unique correct output without access to information usually associated with deeper processing. Finally, the assumption that tokens are strings does not allow for preprocessing which returns further information (e.g., named entity recognition, POS tagging or morphological analysis). Thus a general text interface should:

1. take into account document markup (e.g., via markup-aware tokenizer rules which provide the possibility to strip and recover markup as needed)
2. provide grounding in the text document under consideration (e.g., via standoff pointers)
3. provide an adequate representation for token ambiguity (e.g., a lattice)
4. provide a representation of adequate power for description of token content (e.g., feature structures).

Obviously it is also desirable to adopt commonly accepted standards such as Unicode and XML. Ideally, it should be possible to use the tokenization component in generation as well as for analysis, although we will not discuss that further here.

In this paper, we present a preprocessing and tokenization standoff-annotation standard which has been designed for use in DELPH-IN[1], a loose international collaboration of researchers developing open-source software components for language processing. DELPH-IN components include deep parsers (currently the LKB (Copestake, 2002) and PET (Callmeier, 2000)) which run deep grammars for various natural languages (e.g., the English Resource Grammar (Flickinger, 2002), JACY[2], NorSource[3]) and a range of tools for shallower processing. In previous work associated with DELPH-IN, in the Deep Thought project, the Heart of Gold system (Callmeier et al., 2004) for the integration of shallow and deep linguistic processors was developed along with the Pet Input Chart (PIC) XML interface format. Our current work extends this to provide a more general approach.

2. Standoff Annotation

Linguistic annotation may be generated automatically (the situation we are interested in here) or manually. Annotations may be inline or standoff (Thompson and McKelvie, 1997). Inline annotation is simpler and provides a convenient and straightforward way to annotate many documents. It is particularly suitable to unambiguous manual annotation. However, in the case of XML documents, it

[1]http://www.delph-in.net/
[2]http://wiki.delph-in.net/moin/JacyTop
[3]http://www.ling.hf.ntnu.no/forskning/norsource/

presupposes that the annotated output can be treated as tree-structured, which may be inconvenient or impossible. In contrast, in standoff annotation, each layer of annotations is kept in a separate space; the original data is left unaltered and the standoff document contains structures which are grounded via pointers into the original document. In such a framework, complex annotations with non-tree structure are possible, and a clear distinction exists between markup found in the original document and annotations in the separate standoff document. If parsing is treated as providing standoff annotation, document markup remains available to discourse-level processing.

Standoff annotations have been used successfully in other work: e.g., NITE (Carletta et al., 2003); GATE (Cunningham et al., 2002); WHITEBOARD (Frank et al., 2003); TEI (Sperberg-McQueen and Burnard, 2004). The Deep Thought project, which used and developed various DELPH-IN technology, experimented with a limited form of standoff annotation in the form of character position information encoded as part of RMRS (Copestake, 2003).

The appropriate pointer scheme for standoff annotation depends on the data format of the original input. For example, annotations over an audio file may use frame (or time) offsets; annotations over text might use character pointers into the source. In this paper, our primary focus is standoff annotation over XML documents — we assume that other text formats will mapped to XML in a prior stage. In the next section, we discuss XML standoff pointers in more detail.

2.1. Choice Of Standoff Pointer Implementation

A variety of approaches to standoff pointers in XML have been investigated and have different strengths and weaknesses: XML ID tags are coarse-grained, requiring that the source document contain markup elements around everything of interest (and crucially they cannot overlap); byte offsets are fine-grained but not robust to changes in the character encoding; character offsets are simple to use, although not robust to changes elsewhere in the document; XPointer[4] is very powerful, but its full machinery unnecessary for our purposes.

We take a hybrid approach to choice of standoff pointer: existing non-XML-aware processing components can often easily be converted to produce character pointers, but for XML-aware components, it is easier to work with a pointer scheme aware of the XML tree structure. A mapping between the two pointer schemes provides the necessary interconversion.

An XML text document may be considered at a number of levels: (i) at one level, the document consists of a sequence of bytes, to be interpreted with respect to some coding scheme (specified in the XML header; e.g., `encoding='ISO-8859-1'`); (ii) at a higher level, an XML document is considered a sequence of Unicode characters; (iii) at an even higher level, the document is a collection of abstract nodes forming a tree structure (in which certain distinctions in the input, such as sequences of whitespace, have been collapsed). Non-XML-aware processing components consider an input document as a sequence of bytes or characters. XML-aware processing components work with the XML tree.

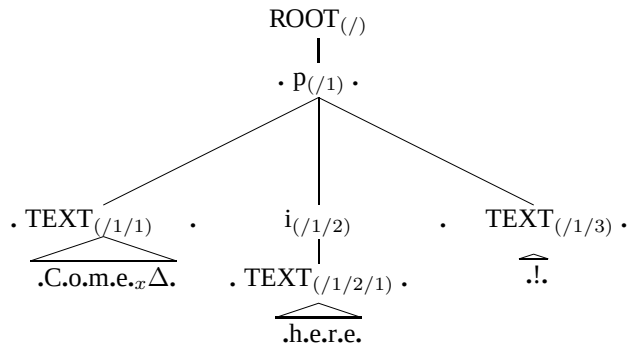
[4] http://www.w3.org/TR/WD-xptr

Raw Text: "`<p>Come <i>here</i>!</p>`"

Unicode character points:

.<.p.>.C.o.m.e.Δ.<.i.>.h.e.r .e .< ./ .i .> .! .< ./ .p .> .

Figure 1: character pointers (points shown as '.' and numbered from the left, starting from 0)



xpoint at x is: "`/1/1.4`"

Figure 2: xpoint-based pointers (points shown as '.')

We use '*char*acter' pointers refer to points between the sequence of Unicode characters (see fig. 1); '*xpoint*-based' pointers refer to points in the abstract XML tree (see fig. 2). Our xpoint-based pointers are derived from the xpoint scheme detailed in the XPointer specification. Any point may be uniquely referenced by specifying an XML node (via XPath) paired with an offset (Unicode character offset inside a text node; XML node offset otherwise). Points within XML attribute values can also be specified in this way. XPath allows multiple expressions to refer to the same node in a particular document; by appropriate choice of XPath expression, it is possible to trade simplicity of pointer scheme against robustness. For example, suppose the `<p>` element in fig. 2 contained attribute `id='p23'`. Point *x* could be specified by one of: "`/1/1.4`" or "`/p/text()[1].4`" or "`//[@id='p23']/1.4`" (in order of robustness to changes to the raw document).

3. Interface formats within DELPH-IN

Firstly we outline the PIC XML interface format which has been in use for some time within DELPH-IN. PIC provides an XML input format tailored to the needs of the PET deep parser. It utilises: (i) standoff pointers (character counts in the source document); (ii) two types of annotation (word, named entity); (iii) an implicit lattice; (iv) for the annotation content, a variety of XML elements/attributes chosen to mesh with the data structures of the deep processor. The PIC format is generated using XSLT mappings from other formats. PIC documents reference grammar-specific types.

Secondly we describe a more generic XML interface format (SMAF) which was derived from MAF (Clement and de la Clergerie, 2005). This is intended to be independent of the parser and grammar. SMAF utilises: (i) choice of standoff annotation scheme; (ii) a number of annotation types

for specific purposes; (iii) a lattice; (iv) intra-annotation dependencies; (v) a range of representations for the content of individual annotations. The mapping from annotation contents into grammar data structures is specified in grammar-specific configuration files. SMAF is designed for the situation where multiple processors are to be flexibly integrated, as, for instance, in the SciBorg project[5]. It allows for levels of interface other than tokenization and morphology, although these will not be discussed further here.

3.1. PIC implementation

The PIC (PET XML Input Chart) format has been used with the deep HPSG parser PET within the Heart of Gold system (Callmeier et al., 2004; Schäfer, 2005). Heart of Gold is a middleware for the integration of deep and shallow natural language processors on the basis of XML standoff annotation that can be flexibly configured and easily accessed by applications.

Heart of Gold uses XSLT (Clark, 1999) for combining and integrating XML markup. The general idea has been presented and motivated in (Schäfer, 2003). XSLT can be used to convert between the various XML formats, and to combine and query standoff annotations. In particular, XSLT stylesheets may also be used to resolve conflicts resulting from multi-dimensional markup, choose among alternative readings, follow standoff links, or decide which markup source to give higher preference.

The PIC format contains the original input text to be parsed, the character span information of the words, optional part-of-speech information (e.g., to allow guessing of the type of unknown words) with weights in case of multiple readings, and a construct for combining simple items to complex ones, e.g., multi-words named entities or chunks. Furthermore, types and, more generally, typed feature structures can be 'injected' via PIC directly into the deep parsers' chart by specifying feature paths with associated values.

In the following, we give an example (for sentence *'When will LREC 2006 take place?'*) of two preprocessing components, a PoS tagger and a named entity recognition (NER) component. Their output triggers generic entries of words unknown to the HPSG lexicon to increase robustness. Both preprocessing components independently produce standoff markup from the input text. The markup is transformed and combined into a single PIC markup document which is then passed to the deep HPSG parser running the English Resource Grammar.

Similar setups as for English have also been implemented for German, modern Greek and Japanese preprocessing components and HPSG grammars in the Heart of Gold.

SProUT (Drożdżyński et al., 2004), the NER component used for these languages, produces typed feature structure markup (Lee et al., 2004) (Figure 3), and transforms it to the PET input chart format using XSLT stylesheets that are automatically generated offline from the SProUT output feature structure specifications (Schäfer, 2005).

The various PoS taggers either directly produce the PIC format or use small, manually developed transformation stylesheets. Figure 6 shows the PET input chart generated

[5] http://www.sciborg.org.uk/

ne-event	
CSTART	"10"
CEND	"18"
VARIANT	**top**
SURFACE	"LREC 2006"
PREPOSITIONS	**list**
EVENTNAME	"LREC 2006"
NE-CONCEPT	*Active_Conference*
NE-OBJID	"obj_89404"
NE-ABBID	"LREC 2006"

Figure 3: SProUT named entity recognition output.

by combining PoS tagger output (<w> elements) and NER (in element <ne>).

Finer-grained information (e.g., gained from integrating ontology information in the NER resources as described in (Schäfer, 2006)) that is not relevant for robust HPSG parsing, can be by-passed as an RMRS structure. The RMRS (Figure 4) is generated from the SProUT output using another automatically generated stylesheet, and can be combined with the deep RMRS (Figure 5) in a post-parsing transformation.

3.2. SMAF implementation

A SMAF document describes a segment (generally, sentence) of the raw input document packaged in a manner suitable for input to a parser. Relevant annotation levels for such a description include simple token strings, part-of-speech tags over simple tokens, complex named-entity tokens and morphological descriptions of tokens (when these are provided by a component external to the parser). The format borrows from MAF and PIC. It incorporates RMRS-XML as a possible annotation content.

The following properties are global to a SMAF document:

- *document* points to the original document (optionally, the raw data may be embedded in the *text* property);
- optionally, the standoff pointer *addressing* scheme may be specified (defaults to *char*acter pointers);
- a collection of OLAC-compatible metadata (including an *identifier*);
- a global span (standoff pointers *cfrom* and *cto*);
- a *lattice* containing the annotations themselves as *edge*s (with special *init*ial and *final* nodes).

The following properties are applicable to an *edge* annotation:

- an *id*entifier;
- a *type* (one of *token*, *pos*, *namedEntity*, *morph*ology);
- a span (standoff pointers *cfrom* and *cto*);

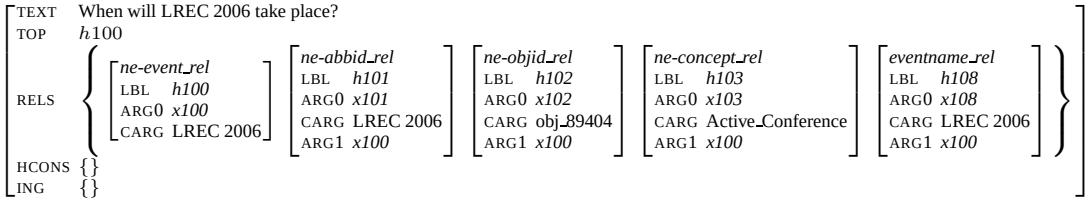


Figure 4: RMRS generated by SProUT with fine-grained named entity information.

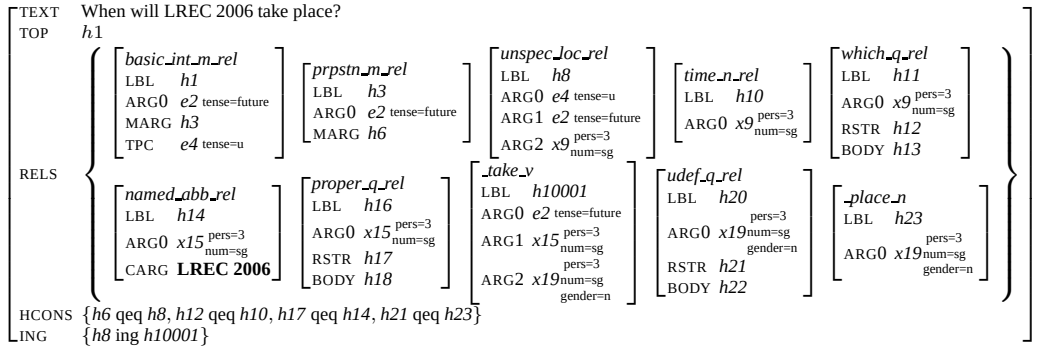


Figure 5: Deep RMRS generated by PET with named entity from external NER component.

- *deps*, a set of edge *id*s on which the annotation has a dependency;
- plus the annotation 'content' (below).

Annotation content consists of a combination of the following elements:

- simple *slot* elements, each consisting of a *name* part (such as *surface*, *weight*, *tagset*, *tag*) paired with a value string;
- complex feature structure (*fs*) elements: these may be typed, and are compatible with the TEI/ISO FSR standard used by MAF;
- complex RMRS description.

Fig. 7 provides a sample SMAF XML document.

A SMAF document may be generated natively by a processing component (such as the DELPH-IN tokenizer), via a wrapper around an external component (e.g., around the ChaSen morphological analyser (Asahara and Matsumoto, 2000) for Japanese), via an XSLT stylesheet from an alternative XML format such as PIC, or by picking relevant annotations from a larger standoff annotation document.

A deep processing component (such as the LKB or PET) must map an input SMAF document into internal component- (and grammar-)specific data structures. The input format is designed with this in mind. The content (*slot*s, *fs*'s, *rmrs*'s) requires component- and grammar-specific mappings (in general determined by the edge *type*); these are specified in simple configuration files. For example, the *tag* slot on an edge of type *pos* (part-of-speech) will be mapped to a grammar-specific type dependent on the *tagset* used. A sample SMAF configuration file entry is shown below:

- `pos.[tag='NN1', tagset='CLAWS7'] => $genericname`

SMAF represents structural ambiguity in tokenization by means of a lattice. The lattice is required to allow for complex forms of ambiguity. For instance, a comma might be normal punctuation or part of the name of a chemical compound. While some disambiguation can be handled by the tokenizer, other cases could require a lexicon and processing of the surrounding context. Rather than complicate the tokenizer, we prefer to delay the decision so that the resources of deeper processing can be brought to bear. This approach is also useful in handling clitics such as the possessive 's' in Norwegian.

SMAF allows three types of annotation content. Simple *slots* hold simple unstructured values, such as a part-of-speech tag with respect to some tagset. Complex structured content may be represented as a (typed) feature structure or as an RMRS.

Users of the DELPH-IN machinery have a number of choices when choosing components for the preprocessing and deep processing modules. The first step in processing a document is segmenting the document into segment spans (generally, sentence units) fit for parsing, and tokenization of text within such spans. This provides us with the necessary data to create a simple SMAF document containing only edges of type *token*. In addition, further components such as a POS tagger or named-entity recogniser (or morphosyntactic analysis external to the deep parser) may provide additional edges. The DELPH-IN tokenizer (regex-based, character-point standoff-aware, markup-aware, and

incorporating a light-weight named entity component) provides a simple preprocessor sufficient for many contexts. It produces SMAF documents natively. Other components have been written from scratch, or adapted (the RASP (Briscoe and Carroll, 2002) sentence splitter, the ChaSen preprocessor for Japanese) to produce this XML interface format. Components previously adapted to produce PIC within the Heart of Gold system are converted via a PIC2SMAF XSLT stylesheet as an interim solution.
Crucially, grounding in the source document (via standoff pointers) is preserved throughout processing and hence the link between system output and the processed text is never broken. These standoff pointers are preserved in the RMRS semantic analyses output by the deep parsers.

4. Summary

This paper has described the use in DELPH-IN of a standoff-annotation interface format between the level of preprocessing (consisting of an obligatory tokenizer module plus optional additional modules) and the deep parser module (consisting of a deep grammar for a particular natural language running on a DELPH-IN deep processor) of the DELPH-IN tools. The role of interface format is played by SMAF, a fairly general XML serialization loosely based on the MAF ISO working draft. Configuration files define the mapping from this format to the specific data structures required by particular deep processor implementations and the type systems of particular grammars.

5. Acknowledgements

We would like to thank the reviewers for valuable comments. We wish to thank Dan Flickinger, Stephan Oepen and other colleagues within the DELPH-IN collaboration for many informative discussions. This work was partly funded by a grant from Boeing to Cambridge University, partly by EPSRC project EP/C010035/1, and partly by a grant from the German Federal Ministry of Education and Research (FKZ 01IWC02).

6. References

M. Asahara and Y. Matsumoto. 2000. Extended Models and Tools for High-performance Part-of-Speech Tagger. In *Proceedings of COLING 2000*, Saarbrücken, Germany.

E. Briscoe and J. Carroll. 2002. Robust accurate statistical annotation of general text. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas, Gran Canaria.

U. Callmeier, A. Eisele, U. Schäfer, and M. Siegel. 2004. The DeepThought core architecture framework. In *Proceedings of LREC-2004*, Lisbon, Portugal.

U. Callmeier. 2000. PET – A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(1):99–108.

J. Carletta, J. Kilgour, T. O'Donnell, S. Evert, and H. Voormann. 2003. The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets. In *Proceedings of 3rd Workshop on NLP and XML, NLPXML-2003*.

J. Clark, 1999. *XSL Transformations (XSLT)*. World Wide Web Consortium, http://w3c.org/TR/xslt.

L. Clement and E. de la Clergerie. 2005. MAF: a morphosyntactic annotation framework. In *Proceedings of the 2nd Language and Technology Conference*, Poznan, Poland.

A. Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford.

A. Copestake. 2003. Report on the Design of RMRS. Technical Report D1.1a, University of Cambridge, UK.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia.

W. Drożdżyński, H.-U. Krieger, J. Piskorski, U. Schäfer, and F. Xu. 2004. Shallow processing with unification and typed feature structures — foundations and applications. *Künstliche Intelligenz*, 2004(1):17–23.

D. Flickinger. 2002. On building a more efficient grammar by exploiting types. In D. Flickinger, S. Oepen, H. Uszkoreit, and J. Tsujii, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*. CSLI Publications.

A. Frank, M. Becker, B. Crysmann, B. Kiefer, and U. Schäfer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of ACL-2003*, Sapporo, Japan.

K. Lee, L. Burnard, L. Romary, E. de la Clergerie, U. Schäfer, T. Declerck, S. Bauman, H. Bunt, L. Clément, T. Erjavec, A. Roussanaly, and C. Roux. 2004. Towards an international standard on feature structure representation (2). In *Proceedings of the LREC-2004 workshop on A Registry of Linguistic Data Categories within an Integrated Language Resources Repository Area*, Lisbon, Portugal.

U. Schäfer. 2003. WHAT: An XSLT-based infrastructure for the integration of natural language processing components. In *Proceedings of the HLT-NAACL Workshop on the Software Engineering and Architecture of Language Technology Systems*, Edmonton, Canada.

U. Schäfer, 2005. *Heart of Gold – an XML-based middleware for the integration of deep and shallow natural language processing components, User and Developer Documentation*. DFKI Language Technology Lab, Saarbrücken, Germany. http://heartofgold.dfki.de/doc/heartofgolddoc.pdf.

U. Schäfer. 2006. OntoNERdIE—mapping and linking ontologies to named entity recognition and information extraction resources. In *Proceedings of LREC-2006*, Genoa, Italy.

C. Sperberg-McQueen and L. Burnard, 2004. *TEI P4: Guidelines for Electronic Text Encoding and Interchange*. http://www.tei-c.org/P4X/.

H. Thompson and D. McKelvie. 1997. Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML-EU-1997*.

```
<?xml version='1.0'?>
<!DOCTYPE pet-input-chart [
 <!ELEMENT pet-input-chart ( w | ne )* >
 <!ELEMENT w ( surface, path*, pos*, typeinfo* ) >
 <!ATTLIST w       id ID      #REQUIRED
               cstart NMTOKEN #REQUIRED
                 cend NMTOKEN #REQUIRED
                 prio CDATA   #IMPLIED
             constant (yes | no) "no" >
 <!ELEMENT surface ( #PCDATA ) >
 <!ELEMENT path EMPTY >
 <!ATTLIST path     num NMTOKEN #REQUIRED >
 <!ELEMENT typeinfo ( stem, infl*, fsmod* ) >
 <!ATTLIST typeinfo  id ID    #REQUIRED
                   prio CDATA #IMPLIED
               baseform (yes | no) "yes" >
 <!ELEMENT stem ( #PCDATA ) >
 <!ELEMENT infl   EMPTY >
 <!ATTLIST infl   name  CDATA #REQUIRED >
 <!ELEMENT fsmod  EMPTY >
 <!ATTLIST fsmod  path  CDATA #REQUIRED
                  value CDATA #REQUIRED >
 <!ELEMENT pos  EMPTY >
 <!ATTLIST pos  tag     CDATA #REQUIRED
                prio    CDATA #IMPLIED >
 <!ELEMENT ne  ( ref+, pos*, typeinfo+ )  >
 <!ATTLIST ne        id ID    #REQUIRED
                   prio CDATA #IMPLIED >
 <!ELEMENT ref  EMPTY >
 <!ATTLIST ref      dtr IDREF #REQUIRED >
 ]>
<pet-input-chart>
<w id="TNT0" cstart="0" cend="3">
  <surface>When</surface>
  <pos tag="WRB" prio="1.000000e+00"/>
</w>
<w id="TNT1" cstart="5" cend="8">
  <surface>will</surface>
  <pos tag="MD" prio="1.000000e+00"/>
</w>
<w id="TNT2" cstart="10" cend="13">
  <surface>LREC</surface>
  <pos tag="NNP" prio="1.000000e+00"/>
</w>
<w id="TNT3" cstart="15" cend="18">
  <surface>2006</surface>
  <pos tag="CD" prio="1.000000e+00"/>
  <typeinfo id="TYI3" baseform="no">
    <stem>$generic_number</stem>
  </typeinfo>
</w>
<w id="TNT4" cstart="20" cend="23">
  <surface>take</surface>
  <pos tag="VB" prio="8.349828e-01"/>
  <pos tag="NN" prio="8.672786e-02"/>
  <pos tag="VBP" prio="7.828936e-02"/>
</w>
<w id="TNT5" cstart="25" cend="29">
  <surface>place</surface>
  <pos tag="NN" prio="1.000000e+00"/>
</w>
<w id="TNT6" cstart="30" cend="30" constant="yes">
  <surface>?</surface>
  <pos tag="?" prio="1.0"/>
</w>
<ne id="SPR1" prio="1.0">
  <ref dtr="TNT2"/>
  <ref dtr="TNT3"/>
  <surface>LREC 2006</surface>
  <pos tag="PN" prio="1.0"/>
  <typeinfo id="TIN2" baseform="no">
    <stem>$generic_name</stem>
  </typeinfo>
</ne>
</pet-input-chart>
```

Figure 6: PET input chart generated from PoS tagger and named entity recognition component output.

```
<smaf document='/data/doc01.txt'>
  <olac:olac
   xmlns:olac='http://www.language-archives.org/OLAC/1.0/'
   xmlns:dc='http://purl.org/dc/elements/1.1/'>
   <dc:creator>SMAF 0.0</dc:creator>
   <created>16:14:31 3/05/2006 (UTC)</created>
   <dc:identifier>s4</dc:identifier>
   <posTagset>CLAWS7</posTagset>
  </olac:olac>
  <lattice init='vo' final='v7' cfrom='0' cto='31'>
   ...
   <edge type='token' id='t43' cfrom='10' cto='14'
         source='v2' target='v3'>
    <slot name='surface'>LREC</slot>
   </edge>
   <edge type='token' id='t44' cfrom='15' cto='19'
         source='v3' target='v4'>
    <slot name='surface'>2006</slot>
   </edge>
   <edge type='token' id='t45' cfrom='20' cto='24'
         source='v4' target='v5'>
    <slot name='surface'>take</slot>
   </edge>
   ...
   <edge type='namedEntity' id='n10' deps='t43 t44'
         cfrom='10' cto='19'>
    <fs type='ne-event'>
     <f name='surface'>LREC 2006</f>
     <f name='eventname'>LREC 2006</f>
     <f name='ne-concept'>Active_Conference</f>
     <f name='ne-objid'>obj_89404</f>
     <f name='ne-abbid'>LREC 2006</f>
    </fs>
   </edge>
   ...
   <edge type='pos' id='p63' deps='t43'>
    <slot name='tag'>NNP</slot>
   </edge>
   <edge type='pos' id='p64' deps='t44'>
    <slot name='tag'>CD</slot>
   </edge>
   <edge type='pos' id='p65' deps='t45'>
    <slot name='weight'>8.349828e-01</slot>
    <slot name='tag'>VB</slot>
   </edge>
   <edge type='pos' id='p66' deps='t45'>
    <slot name='weight'>8.672786e-02</slot>
    <slot name='tag'>NN</slot>
   </edge>
   <edge type='pos' id='p67' deps='t45'>
    <slot name='weight'>7.828936e-02</slot>
    <slot name='tag'>VBP</slot>
   </edge>
   ...
 </lattice>
</smaf>
```

Figure 7: SMAF document