A Standoff Annotation Interface between DELPH-IN Components

Benjamin Waldron and **Ann Copestake**
University of Cambridge Computer Laboratory
JJ Thomson Avenue
Cambridge CB3 0FD, UK
{bmw20,aac10}@cl.cam.ac.uk

Abstract

We present a standoff annotation framework for the integration of NLP components, currently implemented in the context of the DELPH-IN tools[1]. This provides a flexible standoff pointer scheme suitable for various types of data, a lattice encodes structural ambiguity, intra-annotation relationships are encoded, and annotations are decorated with structured content. We provide an XML serialization for intercomponent communication.

1 Background

An NLP system aims to map linguistic data to a description at some suitable level of representation. To achieve this various component processes must perform complex tasks. Increasingly these individual processes are performed by distinct software components in cooperation. The expressiveness of communication between such components hence becomes an issue. For example: we may wish to preserve a linkage from a final semantic analysis to the raw data input; we may wish to pass ambiguity to some later stage where it is more appropriately resolved; we may wish to represent the dependence of certain analyses on other analyses; and we require sufficient expressiveness for the content of individual analyses (henceforth 'annotations'). This work addresses these issues.

Annotations are often associated with a document inline. This provides a convenient and straightforward method of annotating many documents, but suffers from well-known drawbacks. We adopt standoff annotation as an alternative. Here annotations live in a separate standoff annotation document, and are anchored in the raw data via standoff pointers.

[1]http://wiki.delph-in.net

2 The DELPH-IN collaboration

DELPH-IN is a loose international collaboration of researchers developing open-source software components for language processing. These components include deep parsers, deep grammars for various natural languages, and tools for shallower processing. The HOG system (Callmeier et al., 2004) for the integration of shallow and deep linguistic processors (using a pipeline making use of XML plus XSLT transformations to pass data between processors) was developed during the Deep Thought project, as was a standard for the integration of semantic analyses produced by diverse components: RMRS (Copestake, 2003) allows underspecification of semantic analyses in such a way that the analysis produced by a shallow component may be considered an underspecification of a fuller semantic analysis produced by a deeper component. Other work (Waldron et al., 2006) has provided a representation of partial analyses at the level of tokenization/morphology – using a modification of MAF (Clement and de la Clergerie, 2005). Current work within the SciBorg project[2] is investigating more fine-grained integration of shallow and deep processors.

3 Standoff Annotation Framework (SAF)

Our standoff annotation framework borrows heavily from the MAF proposal. The key components of our framework are (i) grounding in primary linguistic data via flexible standoff pointers, (ii) dec-

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

```
<?xml version='1.0' encoding='UTF8'?>
<!DOCTYPE saf SYSTEM 'saf.dtd'>
<saf addressing='char'>
 <olac:olac xmlns:olac='http://www.language-archives.org/OLAC/1.0/' xml
ns='http://purl.org/dc/elements/1.1/' xmlns:xsi='http://www.w3.org/2001
/XMLSchema-instance' xsi:schemaLocation='http://www.language-archives.o
rg/OLAC/1.0/ http://www.language-archives.org/OLAC/1.0/olac.xsd'>
  <creator>LKB x-preprocessor</creator>
  <created>18:11:31 1/31/2006 (UTC)</created>
 </olac:olac>
 <fsm init='v0' final='v9'>
  <state id='v0'/>...<state id='v9'/>
  <annot type='token' id='t1' from='0' to='6' value='Gutten' source='
v0' target='v1'/>
  ...
  <annot type='token' id='t11' from='30' to='31' value='.' source='v8
' target='v9'/>
 </fsm>
</saf>
```

Figure 1: SAF XML (containing token lattice)

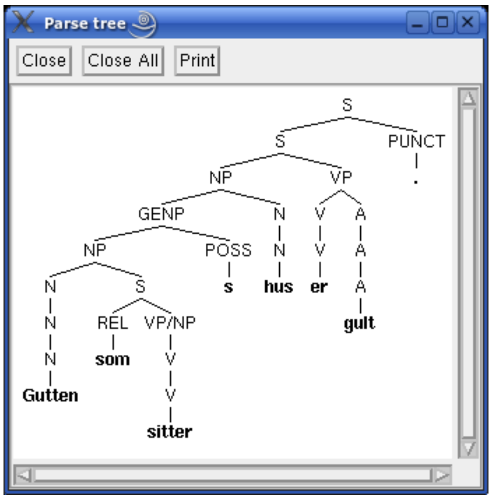

Figure 2: Parse tree (LKB)

oration of individual annotations with structured content, (iii) representation of structural ambiguity via a lattice of annotations and (iv) a structure of intra-annotation dependencies. In each case we have generalized heavily in order to apply the SAF framework to a wide domain. The basic unit of the SAF framework is the annotation. An annotation possesses properties as outlined below.

Each annotation describes a given span in the raw linguistic data. This span is specified by *from* and *to* standoff pointers. In order to cope with different possible data formats (e.g. audio files, XML

```
<saf document='/home/bmw20/b110865
b.xml' addressing='xpoint'>
<annot type='sentence' id='s93' fr
om='/1/3/54.3' to='/1/3/58.89' val
ue='The results of this study are
depicted in Table 2&lt;p/&gt;'/>
</saf>
```

Figure 3: A 'sentence' annotation

text, pdf files) we make the pointer scheme a property of each individual SAF object. So annotations with respect to an audio file may use frame offsets, whilst for an XML text file we may use character (or more sophisticated xpoint-based) pointers. When processing XML text files, we have found it easiest to work with a hybrid approach to the standoff pointer scheme. Existing non-XML-aware processing components can often be easily adapted to produce (Unicode) character pointers; for XML-aware components it is easier to work with XML-aware pointing schemes – here we use an extension of the xpoint scheme described in the XPath specification[3]. A mapping between these two sets of points provides interconversion sufficient for our needs.

[3]For example: */1/3.2* specifies the second point in the third element of the first element of the root node, and an extension allows text nodes in non-elements to be referenced also.

```
<fs type="ne-organisation">
  <f name="OrgName">National Aerona
utics and Space Administration</f>
  <f name="OrgType">institution</f>
</fs>
```

Figure 4: Named entity FSR content

Each annotation possesses a *content*, providing a (structured) description of the linguistic data covered by the annotation. E.g. the *content* of an annotation describing a token may be the text of the token itself (see fig. 1); the *content* of an annotation describing a named entity may be a feature structure describing properties of the entity (see fig. 4); the *content* of an annotation describing the semantics of a sentence may be an RMRS description (see fig. 6). In most cases we describe this content via a simple text string, or a feature structure following the TEI/ISO specification[4]. But in some cases other representations are more appropriate (such cases are signalled by the *type* property on annotations). The *content* will generally contain meta-information in addition to the pure content itself. The precise specification for the *content* of different annotation types is a current thrust of development.

Each annotation lives in a global *lattice*. Use of a *lattice* (consisting of a set of nodes – including a special start node and end node – and a set of edges each with a source node and a target node) allows us to handle the ambiguity seen in linguistic analyses of natural languages. E.g. an automatic speech recognition system may output a word lattice, and a lattice representation can be very useful in other contexts where we do not wish to collapse the space of alternative hypotheses too early.

Fig. 2 shows a Norwegian sentence[5] for which the token lattice is very useful. Here the possessive *s* clitic may attach to any word, but unlike in English no apostrophe is used. Hence it not feasible for the tokenizer to resolve this ambiguity in tokenisation. The token lattice (produced by a regex-based SAF-aware preprocessor) provides an elegant solution to this problem: between nodes 2 and 4 (and nodes 4 and 6) the lattice provides alternative paths.[6] The parser is able to resolve the ambiguity with lexical and syntactic knowledge unavailable to the preprocessor component. See fig. 5 for a simple representation of the token lattice, and fig. 1 for the equivalent SAF XML.

[4]http://www.tei-c.org/release/doc/tei-p5-doc/html/FS.html

[5]Translation: *The boy who is sitting's house is yellow*.

[6]The sentence also exhibits the same phenomena for the final period – it could form part of an abbreviation.

```
0-1 [1] Gutten <0 c 6>
1-2 [2] som <7 c 10>
2-3 [3] sitter <11 c 17>
2-4 [5] sitters <11 c 18>
3-4 [4] s <16 c 18>
4-5 [6] hu <19 c 21>
4-6 [8] hus <19 c 22>
5-6 [7] s <20 c 22>
6-7 [9] er <23 c 25>
7-8 [10] gult <26 c 30>
7-9 [12] gult. <26 c 31>
8-9 [11] . <30 c 31>
```

Figure 5: Token lattice with character-point standoff pointers

Each annotation also lives in a hierarchy of annotation dependencies built over the *lattice*. E.g. sentence splitting may be the lowest level; then from each sentence we obtain a set (lattice) of tokens; for individual tokens (or each set of tokens on a partial path through the lattice) we may obtain an analysis from a named-entity component. A parser may build on top of this, producing perhaps a semantic analysis for certain paths in the lattice. Each such level consists of a set of annotations each of which may be said to build on a set of lower annotations. This is encoded by means of a *depends on* property on each annotation. The annotation in fig. 6 exhibits the use of the *depends on* property to mark its dependency on the annotation shown in fig. 3.

A number of well-formedness constrains apply to SAF objects. For example, the ordering of standoff pointers must be consistent with the ordering of annotation elements through all paths in the lattice. Sets of annotations related (directly or indirectly) via the *depends on* property must lie on a single path through the lattice.

4 XML Serialization

Our SAF XML serialization is provided both for inter-component communication and for persistent storage. XML provides a clean standards-based framework in which to serialize our SAF objects. Our serialization was heavily influenced by the MAF XML serialization.

```
<annot type='rmrs' deps='s93'>
 <label vid='1'/>
 <ep cfrom='18476' cto='18526'>
  <gpred>prpstn_m_rel</gpred>
  <label vid='1'/>
  <var sort='e' vid='2'
       tense='present'/>
 </ep>
 ...
 <rarg>
  <rargname>MARG</rargname>
  <label vid='1'/>
  <var sort='h' vid='3'/>
 </rarg>
</annot>
```

Figure 6: An annotation with RMRS content

The SAF XML serialization is contained within the top `saf` XML element. Here the pointer addressing scheme used (e.g. `char` for character point offsets, `xpoint` for our xpoint-based scheme), and the location of the primary data are specified as attributes. This element may contain an optional `olac` element[7] to specify metadata (e.g. creator) and a single `fsm` element holds the rest of the object (as shorthand we also allow a sequence of the `annot` elements defined below in place of the `fsm`). The `fsm` element consists of a number of `state` elements (with attribute `id`) declaring the available lattice nodes, followed by `annot` annotation definitions.

Each annotation (`annot`) element possesses the following attributes: `from` and `to` give standoff pointers into the primary data, encoded according to the scheme specified by the `saf` element's `addressing` attribute; `source` and `target` each give a `state` id (absent if the annotations are listed sequentially outside of an `fsr` element); `deps` is a set of idrefs; `value` is shorthand for a string-valued content; `type` is shorthand for a particular type of annotation content. The annotation content, if not a `value` string, is represented using the TEI/ISO FSR XML format or the appropriate XML format corresponding to the annotation `type`.

5 Summary

We are in the process of SAF-enabling a number of the DELPH-IN processing components. A SAF-aware sentence splitter produces SAF XML describing the span of each sentence, from which a SAF-aware (and XML-aware) preprocessor/tokeniser maps raw sentence text into a SAF XML token lattice (with some additional annotation to describe tokens such as digit sequences). External preprocessor components (such as a morphological analyser for Japanese) may also be manipulated in order to provide SAF input to the parser. SAF is integrated into the parser of the LKB grammar development environment (Copestake, 2002) and can also be used with the PET runtime parser (Callmeier, 2000). The MAF XML format (compatible with SAF) is also integrated into the HOG system, and we hope to generalize this to the full SAF framework.

6 Acknowledgements

The authors wish to thank Bernd Kiefer, Ulrich Schaefer, 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 the NorSource Grammar Project[8] at NTNU, and partly by EPSRC project EP/C010035/1.

References

Ulrich Callmeier, Andreas Eisele, Ulrich Schaefer, and Melanie Siegel. 2004. The DeepThought Core Architecture Framework. In *Proceedings of the 4th International Conference on Language Resources and Evaluation, LREC'04*, Lisbon, Portugal.

Ulrich Callmeier. 2000. PET. A Platform for Experimentation with Efficient HPSG Processing Techniques. *Journal of Natural Language Engineering*, 6(1):99–108.

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

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

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

Benjamin Waldron, Ann Copestake, Ulrich Schaefer, and Bernd Kiefer. 2006. Preprocessing and Tokenisation Standards in DELPH-IN Tools. In *Proceedings of the 5th International Conference On Language Resources and Evaluation, LREC'06*, Genoa, Italy.

[7]http://www.language-archives.org/OLAC/metadata.html

[8]http://www.ling.hf.ntnu.no/forskning/norsource/