LASSY WORK PACKAGE 5

SPECIFICATION OF A SEARCH AND EXTRACTION TOOL

Deliverable 5.2

Erik Tjong Kim Sang
Alfa-informatica
University of Groningen

LASSY WP 5.2: Specification of a Search and Extraction Tool

Erik Tjong Kim Sang
Alfa-informatica, University of Groningen

8th June 2009

1 Introduction

This document presents a specification of the search and extraction tools for syntactically annotated corpora, developed in the framework of the LASSY project. Work on these tools was started by Geert Kloosterman [Kloosterman, 2007]. This document presents a description of those tools, describes improvements of the tools demanded by the project and suggests how to achieve these improvements.

2 Available treebank tools

The two most important elements of the current treebank tools [Kloosterman, 2007] are dtview and dtsearch. The first allows inspection of syntactically annotated corpora encoded in XML. The second makes possible selection of parts of such corpora that match a particular query. Queries are specified in XPath, a query language for XML documents developed by the World Wide Web Consortium (W3C) [Clark and DeRose, 1999]. Queries can also be used in dtview for highlighting groups of nodes.

2.1 XML in the Lassy Treebank

The Lassy Treebank is a collection of Dutch texts in which each sentence has been enriched with syntactic annotation generated by the Alpino parser. The annotation is encoded in XML. A special document type definition (dtd) has been created to specify the XML tags that can occur in the treebank. The information about each sentence is encoded between `alpino_ds` (Alpino Dependency Structure) tags. The information consists of a tokenized version of the sentence (between `sentence` tags), the syntactic structure (inside a `node` environment) and an optional comment (inside a `comments` environment). Here is an example sentence showing the most important elements annotation structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_ds version="1.1">
  <node>
    <node word="test"/>
  </node>
  <sentence>test</sentence>
  <comments>
    <comment>some comment</comment>
  </comments>
</alpino_ds>
```

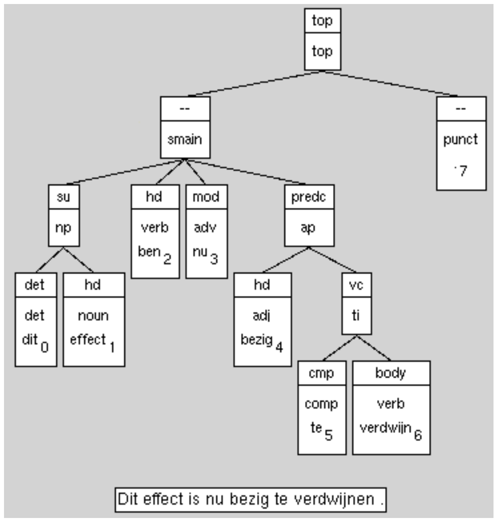
The XML tags in the annotation contain attributes, some of which have been shown in this example. Neither the `sentence` nor the `comments` tag may contain attributes. Much of the syntactic annotation is encoded in the attributes of the `node` tags. These are the most important `node` attributes that enclose a certain phrase in a sentence:

- begin: the sentence position of the start of the phrase
- cat: the syntactic category (for non-leaf nodes)
- end: the sentence position of the end of the phrase
- id: a unique id of this node
- index: a unique name for linking one node to another
- pos: the part-of-speech tag of the word in the node (for leaf nodes)
- rel: the syntactic relation of this node with its sister nodes
- root: the lemma contained by this node (for leaf nodes)
- word: the word contained by this node (for leaf nodes)

An example of an annotated sentence with these attributes can be found in Figure 1.

2.2 XPath

In the treebank tools, the query language used used for selecting annotated phrases is XPath [Clark and DeRose, 1999]. The language allows retrieval of sentences that contain certain words



```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_ds version="1.1">
  <node begin="0" cat="top" end="8" id="0" rel="top">
    <node begin="0" cat="smain" end="7" id="1" rel="--">
      <node begin="0" cat="np" end="2" id="2" rel="su">
        <node begin="0" end="1" id="3" pos="det" rel="det" root="dit" word="Dit"/>
        <node begin="1" end="2" id="4" pos="noun" rel="hd" root="effect" word="effect"/>
      </node>
      <node begin="2" end="3" id="5" pos="verb" rel="hd" root="ben" word="is"/>
      <node begin="3" end="4" id="6" pos="adv" rel="mod" root="nu" word="nu"/>
      <node begin="4" cat="ap" end="7" id="7" rel="predc">
        <node begin="4" end="5" id="8" pos="adj" rel="hd" root="bezig" word="bezig"/>
        <node begin="5" cat="ti" end="7" id="9" rel="vc">
          <node begin="5" end="6" id="10" pos="comp" rel="cmp" root="te" word="te"/>
          <node begin="6" end="7" id="11" pos="verb" rel="body"
root="verdwijn" word="verdwijnen"/>
        </node>
      </node>
    </node>
    <node begin="7" end="8" id="12" pos="punct" rel="--" root="." word="."/>
  </node>
  <sentence>Dit effect is nu bezig te verdwijnen .</sentence>
</alpino_ds>
```

Figure 1: The syntax tree and the XML structure corresponding to the syntactically analyzed version of the sentence *Dit effect is nu bezig te verdwijnen.* (*This effect is currently disappearing.*).

or syntactic relations. Here are a few examples of syntactic queries with the associated XPath expressions:

- Find all sentences that contain the word *geen*:
  XPath: `//node[@word="geen"]`
  Result example: *Er zijn Belfast* [*geen*] *toekomstmogelijkheden meer .*
- Find all sentences that contain a word with lemma *word*:
  XPath: `//node[@root="word"]`
  Result example: *Er* [*werd*] *geen autopsie verricht .*
- Find all sentences that start with an adverb
  XPath: `//node[@begin="0" and @pos="adv"]`
  Result example: [*Zo*] *niet de Nederlanders .*

(Square brackets in the result examples mark matching phrases)

In the Lassy treebank, almost all annotation has been encoded in `<node>` tags. This simplifies the XPath expressions that can be used for accessing the data. The syntactic information that in encoded in the expressions consists of required attribute values or required relations between node tags.

XPath expressions for accessing information in the Lassy Treebank will look for a node or a sequence of nodes somewhere inside an annotation structure. The corresponding XPath expression will start with `//node[` followed by a list of attribute requirements which can be separated by `and` or `or` operators, like in the examples shown above.

Child and ancestor relations can be specified in an XPath expression by adding an extra node statement to the first, separated by `/` (child relation) or `//` (ancestor relation). Here are two examples:

- Find all sentences that contain a noun with lemma *effect* and a subject parent
  XPath: `//node[@rel="su"]/node[@pos="noun" and @root="effect"]`
  Result example: *Dit [effect] is nu bezig te verdwijnen .*
- Find all sentences that contain preposition *aan* somewhere in an object
  XPath: `//node[@rel="obj1"]//node[@root="aan"]`
  Result example: *het lichaam heeft minder behoefte* [*aan*] *voedingsstoffen .*

More examples of XPath expressions can be found in Kloosterman (2007) and in the next section.

2.3 Using XPath in dtview and dtsearch

The treebank tools dtsearch and dtview allow access to the XML annotation produced by the Alpino parser via XPath queries. Here is an example call of dtsearch:

```
$ dtsearch -s '//node[@word="geen"]' *.xml
```

The command searches in the files `*.xml` in the current directory and shows all sentences that contain the word *geen*. The command line option `-s` specifies the desired output format: tokenized sentences with square brackets around matching phrases (like in the examples in the previous section). There are several alternative output modes, for example `-c`, matching constituents only, and `-v`, dtview's graphical mode [Kloosterman, 2007].

Dtview has the same functionality as dtsearch without the text-format output modes. Additionally dtview allows examining files without having to specify a search expression.

2.4 Other treebank tools

The 2007 version of the treebank tools contains some additional useful tools apart from dtview and dtsearch. First, there is an editor for the treebank annotations: dttred. It is based on the tred tree editor by Petr Pajas [Pajas, 2008]. There are also two more search and display tools with a similar functionality like two basic command line commands: dtgrep (functions like grep) and dtget (works like cat).

The data format of the Lassy corpus files is quite verbose. The compression software act (Alpino Corpus Tool) offers compression rates of more than 90% for Lassy treebank files. A compressed collection of *.xml files can be created with the following command:

```
$ act --create --recursive --targetdir=newdir dirWithFiles
```

The command will create two files in the new directory: directory.data.dz and directory.index. These only contain information extracted from XML files stored in the ancestors of the directory. Non-XML files will be ignored. The search and display tools dtsearch and dtview also accept compressed data files (both *.data.dz and *.index) as input.

3 Additional search and exploration requirements

Tjong Kim Sang (2009) describes three case studies in corpus linguistics and shows that dtsearch is insufficient for implementing the complex queries that were required in these studies. In this section check the different problems put forward by that study and present suggestions for future extensions of the available treebank tools.

3.1 Query problems reported in [Tjong Kim Sang, 2009]

Tjong Kim Sang (2009) presented three different syntactic queries originating from studies that used the Lassy corpora:

1. find all noun phrase pairs linked by three or fewer nodes
2. find verb objects that consist of noun phrase conjunctions
3. find pairs of verbs and their objects

Of these three, the only one that can be implemented in XPath in a straightforward way is the third. This means that two of the queries are hard or perhaps even impossible to perform with dtsearch. We examine the problems.

Selecting sentences containing two or more noun phrases is not very difficult: `count(//node[@cat="np"]) > 1`. However, finding out whether the syntactic path between the two nodes contains three or fewer nodes, is not possible. First, we would need to identify each individual noun phrase and then check the paths between each noun phrase pair. This would require assigning names to each of the noun phrases or having variables which could take them as values. Neither of these are possible in the current version of dtsearch.

An XPath expression which comes close to executing the second query, is the following:

```
//node[@pos="verb" and @rel="hd"]/../node[@rel="obj1" and @cat="conj" and
count(node[@cat="np"]) > 1 and node[@root="en" or @root="of"]]
```

The expression looks for verbs that are the head of their parent phrase and additionally contain a sister node that is a direct object as well as a conjunction and contains more than one noun phrase daughter as well as a daughter with the lemma *en* (*and*) or *of* (*or*).

This XPath expression works fine for retrieving conjunction phrases from trees. However, the Lassy annotations contain graphs rather than trees. Some of the information in the annotation might be linked from different places by index attributes. This happens sometimes with direct objects. The XPath expression is unable to handle all of these cases.

An additional problem is that the case study which used this query [Van Noord, 2009] was not looking for conjunction phrases but for pairs of verbs and noun phrases that were direct objects. Rather than generating np_1 *and* np_2, the query should produce something like *verb* – np_1 and *verb* – np_2. In order to be able to do this, we will again need to assign names to each of the noun phrases or have variables which could take them as values, something which is not possible in the current version of dtsearch.

3.2 What needs to be added to dtsearch?

The study of Tjong Kim Sang (2009) suggests that the most important omission from XPath as used in dtsearch, is the lack of variables for storing intermediate query results. Additionally, it would be useful to have commands for manipulating query execution, like conditional command execution and command iteration. However, this requires converting XPath to a full programming language which is outside the scope of our research project.

The solution is to use a programming environment which is already available and is already being used for querying the Lassy treebank: XQuery [Brundage, 2004, Boag et al., 2007]. It allows implementing all of the three syntactic queries mentioned in the previous section [Tjong Kim Sang, 2009].

Tjong Kim Sang (2009) indicated that a disadvantage of XQuery is that it is too difficult for novice users of the treebank tools. We agree with this and propose to make possible treebank access both by XPath and XQuery. Novice users will start by learning XPath and use that for creating treebank queries. Users that have programming experience will be able to formulate more complex queries with XQuery.

3.3 Requirements for the LASSY toolkit

Based on our recent work with the LASSY corpora [Tjong Kim Sang, 2009, Tjong Kim Sang and Hofmann, 2009], we define the following required tasks to be performed by the LASSY toolkit for accessing these corpora:

- offer graphical browsing facilities for annotated trees
  currently available via dtview; no changes required
- offer search and inspection facilities to novice users (via XPath)
  currently available via dtsearch; no changes required
- offer flexible and unrestricted access to annotated trees to experienced users (via XQuery)

XQuery access to the data is already possible but needs to be made easier, for example by providing a library of frequently used corpus access functions

For simplicity, the programs dtview and dtsearch could be combined to a single program that offers both selection functions and display functions.

An important additional requirement is that all searching and browsing tools need to be able to access the annotated data that was compressed with `act`. This is already the case for dtview and dtsearch. The toolkit must include an XQuery module that makes it possible to interact with the compressed data in the same way as with noncompressed XML data.

The most important work that needs to done, is making corpus access via XQuery easier. This involves choosing an XQuery processing engine and providing an easy to use interface from XQuery to the corpora, both to those in XML format and those that have been compressed with `act`. In our own work, we have used the Saxon engine [Kay, 2008] for processing XQuery expressions and this has worked out fine. Interface software should allow easy data access, for example by asking the user to specify an XQuery extraction script as well the input data files and next process the data and collect the output.

Easier corpus access via XQuery also involves providing a library of data-specific access functions to the user. The library needs to include at least the following functions:

- retrieving the lexical yield of a node
  a very common task in corpus analysis
- retrieving the lexical head of a phrase
  heads often have relation attribute `hd` but often they are annotated differently
- identifying noun phrases
  noun phrases have not always been marked up as such because of inherited annotation requirements
- resolving index nodes
  index nodes are pointers to other nodes

Apart from these basic functions there are other less commonly used functions which could be interesting for the XQuery library, for example providing access to (combinations of) names and providing access to pairs of phrases that are related.

4 Specification of the LASSY Toolkit

This section contains a specification of the required functionality of the LASSY toolkit. It deals with the data formats that the toolkit should be able to handle, the display facilities it should offer for syntactic trees and the search and extraction tasks it should be able to perform.

4.1 Data formats

All tools in the toolkit should be able to read two input formats:

1. XML data formatted according to the treebank dtd (Alpino Dependency Structures DTD, version 1.1 2005-11-25 14:43:27, file path location $ALPINO_HOME/Treebank/alpino_ds.dtd)
2. Compressed directories with files, each of which contains an annotated sentence in the previous file format. Each directory with associated files is compressed with dictzip, a compression algorithm which uses gzip and generates an additional index file which allows random access to the compressed data without having to decompress the whole file

Software for accessing these two LASSY data formats is already available (dtget).

4.2 Display requirements

1. The LASSY toolkit should enable graphical display of annotated sentences with the option of highlighting selected phrases and subtrees
2. Syntactic annotation may be displayed as trees rather than graphs. References by index attributes may be mentioned by name only to avoid display complexities
3. Display facilities should be available for all three major operating systems (Linux, Windows and Mac)

Software for displaying annotated sentences is already available (dtview). The chosen implementation method, Tcl/Tk, allows platform-independent access. The main tests have been performed at Linux systems with X-windows. Functionality testing at other platforms remains to be done.

4.3 Search and extraction tasks

1. The LASSY toolkit should enable search and selection of subtrees of sentences in an annotated corpus by XPath expressions

2. It should be possible to obtain the output of such search tasks in different formats:
   (a) matching lexical items
   (b) all lexical items with matching lexical items enclosed in brackets
   (c) full sentence annotation with attributes marking the selected nodes
3. The LASSY toolkit should enable search and selection of subtrees of sentences in an annotated corpus by XQuery programs
4. No output requirements are necessary for XQuery programs as these allow the creators to select any required output format
5. The LASSY toolkit should offer a standard library of useful XQuery functions which include at least:
   (a) retrieving the lexical yield of a node
   (b) retrieving the head of a node
   (c) retrieving all noun phrases of a sentence
   (d) retrieving the contents of nodes referred to by index attributes
6. Both XPath and XQuery access needs to be possible at the three major operating system platforms

The LASSY toolkit already contains software for XPath access (dtsearch). Note that the output of full annotations with selections involves adding an extra attribute (selected) to the XML dtd. Easy access to XQuery access needs to be provided. Some library functions are already available. The three user studies will offer ideas for other useful library functions.

5 Concluding remarks

We have presented a general description of the format of the Lassy treebanks as well as the available treebank tools for accessing the annotated data. The main query tool dtsearch is currently unable to support some interesting queries that were necessary in previous studies with the treebanks. We suggest to improve the strength of the toolkit by supporting XQuery access to the corpora. Next we specified the requirements of the toolkit. Much of the requirements are already met by the current version of the toolkit. The most important tasks for future work are:

- testing display facilities at Windows and Mac platforms
- enabling easier access to corpus data by XQuery programs
- offering a standard library for XQuery access

These are the tasks that we will focus on in the next phase of the project. The three user studies will offer useful insights for these three tasks.

References

[Boag et al., 2007] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., and Siméon, J. (2007). *XQuery 1.0: An XML Query Language.* World Wide Web Consortium (W3C). http://www.w3.org/TR/xquery/.

[Brundage, 2004] Brundage, M. (2004). *XQuery – The XML Query Language.* Addison-Wesley.

[Clark and DeRose, 1999] Clark, J. and DeRose, S. (1999). *XML Path Language (XPath).* World Wide Web Consortium (W3C). http://www.w3.org/TR/xpath.

[Kay, 2008] Kay, M. (2008). Saxon 9.1.

[Kloosterman, 2007] Kloosterman, G. (2007). *An overview of the Alpino Treebank tools.* Alfa-informatica, University of Groningen. http://www.let.rug.nl/vannoord/alp/Alpino/Treebank-Tools.html.

[Pajas, 2008] Pajas, P. (2008). *Tree Editor TrEd.* Charles University in Prague.

[Tjong Kim Sang, 2009] Tjong Kim Sang, E. (2009). *LASSY WP 5.1: Feasibility Study Search / Extraction Tools.* LASSY project report, University of Groningen.

[Tjong Kim Sang and Hofmann, 2009] Tjong Kim Sang, E. and Hofmann, K. (2009). Lexical patterns or dependency patterns: Which is better for hypernym extraction? In *Proceedings of CoNLL-2009.* Boulder, CO, USA.

[Van Noord, 2009] Van Noord, G. (2009). Self-trained bilexical preferences to improve disambiguation accuracy. In *Selected Papers from the IWPT 2007, CONLL 2007 and IWPT 2005.* Springer.