Querying Dependency Treebanks in XML

Gosse Bouma*, **Geert Kloosterman**†

*Computational Linguistics
Groningen University
gosse@let.rug.nl

†Artificial Intelligence
Groningen University
geertk@ai.rug.nl

Abstract

The need for manual editing during construction of a treebank may impose constraints on the representation of dependency trees which are not optimal for linguistic exploration. Using XML-technology it is possible to maintain the treebank both in a form suitable for editing and in a form suitable for linguistic exploration. By choosing a compact representation, we can use XPath directly as query language. We argue that, given an explicit encoding of string positions, this direct encoding of dependency trees as XML-trees can represent discontinuous constituents in a way that supports queries involving both dependency and linear order.

1. Introduction

During development of a treebank, the emphasis is often on representing the treebank in a way which supports interaction with a parser environment (for automatically generating analyses) as well as with an editor (for correcting analyses manually). For users of a treebank, however, a representation which supports linguistic exploration is essential. It is not always clear that these two requirements can be satisfied by a single representation.

Below, we describe the Alpino treebank, a dependency treebank for Dutch which is being developed using a wide-coverage parser for Dutch and a graphical tool for displaying and editing linguistic data-structures. The initial motivation for developing the treebank was the need for evaluation material for the syntactic parser for Dutch, which is being developed in parallel with the treebank. For (automatic) evaluation, the internal format of the treebank is not very important. However, as the treebank grows in size, it becomes increasingly interesting to explore it interactively as well. Queries to the treebank may be motivated by linguistic interest (i.e. which verbs take inherently reflexive objects?) but can also be a tool for quality control (i.e. find all PP's where the head is not a preposition).

The XPath standard implements a powerful query language for XML documents, which can be used to formulate queries over the treebank. However, we found that both the complexity and size of the XML documents produced during the annotation process, makes these less suitable for querying. Implementation of a query expansion tool only partially solved this problem. Therefore, we transformed the annotated data into a more compact XML-representation, ideally suited for linguistic exploration. As the new format encodes dependency trees directly, it can be explored using XPath only. Using XPath has the advantage that existing technology can be used to query the treebank.

One of the attractions of dependency trees is the fact that dependents may span discontinuous parts of the input string. For a language like Dutch, with its crossing dependency word orders, this is especially important. Several possibilities exist for encoding discontinuous constituency in XML. In our proposal, the structure of the dependency tree is mapped directly onto an XML-tree with the same structure. The relation with word order in the input string is encoded by adding attributes for string positions. We show that this allows subtle queries concerning linear order and dependency to be stated.

In the next section, we briefly describe the grammar used to create dependency trees. Next, we introduce the Alpino treebank and the annotation process. In section 4, we present a transformation of the original treebank format into a format which supports linguistic exploration. In section 5, we discuss how we encode word order in dependency trees. We conclude with various examples queries illustrating the kind of information that can be extracted.

2. The Alpino Grammar

The Alpino grammar (Bouma et al., 2001) is a lexicalized grammar for Dutch in the tradition of constructionalist Head-driven Phrase Structure Grammar (Pollard and Sag, 1994).[1] The grammar currently contains over 270 rules, defined in terms of a general rule structures and principles. The grammar covers a substantial part of the syntactic constructions of Dutch (including main and subordinate clauses, (indirect) questions, imperatives, (free) relative clauses, a wide range of verbal and nominal complementation and modification patterns, verbal crossing-dependency constructions, extraposition, and coordination) as well as a wide variety of more idiosyncratic constructions (appositions, verb-particle constructions, PP's including a particle, NP's modified by an adverb, punctuation, etc.). The lexicon contains approximately 47,000 lemma's. Lemma's are associated with complicated attribute-value matrices, containing, for instance, subcategorization frames enriched with dependency relations. The lexicon was created to a

[1]Alpino is being developed as part of the NWO PIONIER project *Algorithms for Linguistic Processing*, `www.let.rug.nl/~vannoord/alp`

large extent by extracting information from two existing resources (Bouma, 2001), a version of Celex (Baayen et al., 1993) extended with valency information and Parole.[2] Currently, the lexicon contains definitions for 70 different verbal subcategorization types, various nominal types (nouns with various complementation patterns, proper names, pronouns, temporal nouns, deverbalized nouns), various adjectival subcategorization types, and distinct complementizer, determiner, and adverb types.

Linguistic analysis of a given input string on the basis of the grammar described above, proceeds in three steps. First, the lexical analysis phase assigns the most likely part of speech tags to the input. Lexical analysis recognizes multi-word lexical items, and uses several heuristics to decide on the most likely tags for unknown words. The POS-tagger consists of a HMM trained on material analyzed automatically by the grammar without POS-tagging (Prins and van Noord, 2001). Next, a parse forest is constructed, in which the various analyses (typically, several hundreds) of the input are represented in a compact and non-redundant manner. Parsing is robust, in that it will find the constituents spanning a maximal portion of the input in case no full parse is available (van Noord, 2001). The third step consists of the selection of the best parse from the parse forest. Here, we use a log-linear statistical model which computes the likelyhood of an analysis as the weighted sum of predefined properties of a parse. Relevant properties are defined by hand, and weights are estimated using a combination of supervised and unsupervised Maximum Entropy learning (Bouma et al., 2001). Supervised learning uses the dependency treebank described below.

We use the dependency treebank described below for evaluation of the parser in a manner similar to that described in Carroll et al. (1998). The system currently identifies dependency relations with an accuracy of around 80%.

3. The Alpino Dependency Treebank

To evaluate the coverage and disambiguation component of the system, a testbench containing syntactically annotated material is absolutely crucial. Furthermore, (supervised) training of a statistical disambiguation module requires syntactically annotated material. Given the current lack of such material for Dutch, we have started to annotate a corpus of newspaper text with dependency trees in parallel with the grammar development effort.

Dependency structures make explicit the dependency relations between constituents in a sentence. Each nonterminal node in a dependency structure consists of a head-daughter and a list of non-head daughters, whose dependency relation to the head is marked. An example is given in figure 1. Control relations are encoded by means of co-indexing (i.e. the subject of *hebben* is the dependent with index **1**). Note that a dependency structure does not necessarily reflect (surface) syntactic constituency. The dependent *haar nieuwe model gisteren aangekondigd*, for instance, does not correspond to a (surface) syntactic constituent.

Dependency trees provide a relatively theory-neutral level of annotation which is especially suited for annotating languages with a strong word order variation and discontinuous constituency, such as Czech, German, or Dutch (Hajicova et al., 1998; Skut et al., 1997) . Dependency relations also have been used successfully in statistical parsing (Collins, 1999). An important further reason for adopting dependency trees in our case is that ensures consistency with the guidelines of the Spoken Dutch Corpus (Oostdijk, 2000; Moortgat et al., 2000). This project will eventually provide a substantive amount of syntactically annotated spoken language, which we hope to use as further training and testing of the Alpino grammar.

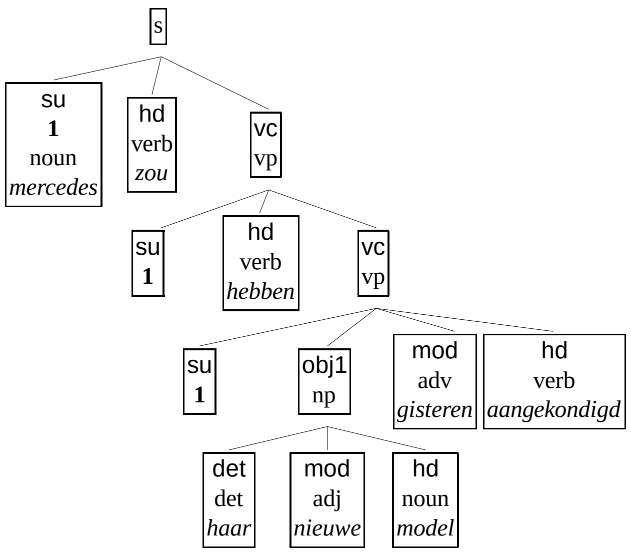

Figure 1: Dependency structure for *Mercedes zou haar nieuwe model gisteren hebben aangekondigd*, *Mercedes should her new model yesterday have announced*.

3.1. Construction of the Treebank

The annotation process typically starts by parsing a sentence with the Alpino grammar. This produces a (often large) number of possible analyses. The annotator picks the analysis which best matches the correct analysis. To facilitate selection of the best parse among a large number of possibilities, the grammar development environment HDRUG (van Noord and Bouma, 1997) has been extended with a number of auxiliary functions. First, a graphical tool allows the annotator to select or remove POS-tags suggested by lexical analysis. Second, the input string may be extended with brackets, which must be respected by grammatical analysis. Finally, a graphical tool based on the SRI TreeBanker (Carter, 1997) has been added which displays all remaining fragments of the input which are a source of ambiguity. By disambiguating these items (usually a much smaller number than the number of analyses), the annotator can quickly pick the most accurate parse.

If the parse selected by the annotator is fully correct, the dependency structure for that parse is stored as XML in the treebank. If the best parse produced by the grammar is not the correct parse as it should be included in the treebank, the dependency structure for this parse is sent to the Thistle

[2] `http://www.inl.nl/corp/parole.htm`

editor. LT Thistle (Calder, 2000)[3] is an editor and display engine for linguistic data-structures which allows corrections to be made easily. The result of the annotation rocess is stored as XML.

We have started to annotate various smaller fragments using the annotation tools described above. The largest fragment consists of 4,000 newspaper sentences (67,000 words) extracted from the Eindhoven corpus (Uit den Boogaart, 1975).[4]

4. Querying the treebank

As the treebank grows in size, it becomes increasingly interesting to query it for purposes other than evaluating the accuracy of a parser. Many aspects of the annotation-guidelines cannot be represented by a DTD. A DTD can ensure that trees have a given structure, and that categories, part of speech tags, and dependency relations are members of a fixed vocabulary, but cannot be used to prevent, say, that two dependents both have the *head* dependency relation, or that a noun is mistakingly tagged as a preposition. Querying the treebank finds many of such mistakes. The most obvious reason for querying the treebank, however, is the fact that it contains valuable linguistic information. In section 6, we present examples of linguistically motivated queries.

The XPath standard implements a powerful query language for XML documents, which can be used to formulate queries over the treebank. However, we found that both the complexity and size of the XML representation produced during the annotatin process makes it less suitable for querying. Implementation of a query expansion tool only partially solved this problem. Therefore, we transformed the data into a more compact XML representation, ideally suited for linguistic exploration. As the new format encodes dependency trees directly, it can be explored using XPath only.

4.1. XPath

As the treebank is encoded in XML, we can use XPath[5] to formulate queries over the treebank. XPath supports conjunction, disjunction, negation, and comparison of numeric values, and seems to have sufficient expressive power to support a range of linguistically relevant queries. The fact that the XML-encoding can directly reflect the dominance relations of the dependency trees further facilitates the formulation of queries.

Various tools support XPath and can be used to implement a query-tool. Currently, we are using a C-based tool implemented on top of the LibXML library[6]. We found that this solution provides a more efficient solution than some of the alternatives. Note, however, that LibXML processes the data as XML (i.e. as textual data) and does not use any kind of indexing scheme. For the medium size treebank we are working with, this proved to be feasible, especially after applying the transformation described below.

[3]`www.ltg.ed.ac.uk/software/thistle/`

[4]The current version of the treebank is available at `www.let.rug.nl:~vannoord/trees/`.

[5]`www.w3.org/TR/xpath`

[6]`www.xmlsoft.org/`

4.2. The encoding used for annotation

In Thistle, linguistic datatypes are defined both as hierarchically structured data-types and as graphical objects. A non-terminal node, for instance, can be defined as an object consisting of a category label, a dependency relation, an optional index, and a list of daughters, while at the same time it is defined as a tree whose top-node consists of a framed box whose elements are to be displayed in bold. Given a definition of a data-type, Thistle automatically generates a DTD for XML-documents encoding the hierarchical structure of the object. As access to the editor is essential during the annotation process, all material in the treebank is encoded initially using the DTD generated by Thistle.

The DTD encodes grammatical dominance relations by dominance in the corresponding XML document. For example, the phrase *de Westerse wereld* (*the Western world*), ranging from position 9 to 12 in the input, and functioning as a conjunct, is schematically represented as follows:

```
<tree>
  <mother>
     <deprel>cnj</deprel>
     <cat>np</cat>
  </mother>
  <daughters>
    <daughter>
      <deprel>det</deprel>
      <pos>det</pos>
      <word>de/[9,10]</word>
    </daughter>
    <daughter>
      <deprel>mod</deprel>
      <pos>adj</pos>
      <word>westers/[10,11]</word>
    </daughter>
    <daughter>
      <deprel>head</deprel>
      <pos>noun</pos>
      <word>wereld/[11,12]</word>
    </daughter>
  </daughters>
</tree>
```

A dependency tree consists of a mother node and a list of daughters, where phrasal nodes contain elements representing the dependency relation and the phrasal category, and leaves contain elements representing the dependency relation, part-of-speech, and a word string containg the root and pointers for the position of the word in the input string.

The actual representation contains several additional layers, which ensure compatibility with the editor. For instance, the word *wereld* alone gives rise to the code below:

```
<leaf_type>
  <tcategory_x_leaf>
    <tnode_type>
      <pos_x_tnode>
        <pos_type>
          <pos_x_pos>noun</pos_x_pos>
        </pos_type>
      </pos_x_tnode>
      <word_x_tnode>
        <word_type>
          <lex_x_word>
            wereld/[11,12]
          </lex_x_word>
        </word_type>
      </word_x_tnode>
    </tnode_type>
  </tcategory_x_leaf>
  <deprel_x_leaf>
    <deprel_type>
      <rel_x_deprel>hd</rel_x_deprel>
    </deprel_type>
 </deprel_x_leaf>
</leaf_type>
```

Although the encoding is complex, querying the treebank for linguistically relevant patterns can in principle be achieved using XPath directly. For instance, to search for main clauses which contain a modifier dependent whose part of speech is ADJ, on might use the following query:

```
//tree_type[
     ./mother_x_tree//cat_x_cat="smain"
 and ./mother_x_tree//rel_x_deprel="top"
 and ./daughters_x_tree/leaf_type[
         .//rel_x_deprel="mod"
     and .//pos_x_pos="adj"]
           ]
```

The XPath constructs for accessing child ('/') and descendant ('//') elements are used to query linguistically relevant mother and daughter relationships. Note that this is possible because the dependency tree is encoded as a tree in XML as well.

The disadvantage of using XPath on highly structured XML-documents is that queries tend to get complex quickly. The XML encoding of dependency-trees contains various layers of annotation which are not linguistically relevant. Therefore, queries are often verbose, and require intimate knowledge of the DTD.

To overcome this problem, we implemented a query expansion script on top of the XPath search tool. The query expansion language accepts queries such as:

```
[cat=smain,deprel=top,
  daughter([deprel=mod,pos=adj])]
```

and expands this into the XPath expression shown above.

4.3. A compact encoding

Although querying the treebank in the form used during the annotation process is possible in principle, we found that a number of disadvantages remained:

- As documents are large, evaluation of queries is slow.
- The query expansion language helps to formulate queries, but is less expressive than XPath.
- As linear order is only implicitly present in the strings representing a word, it is impossible to formulate queries involving linear precedence.

To overcome these problems, we designed a new DTD for dependency trees, optimally suited for efficient querying. We used XSLT and Pillow(Cabeza et al., 1996)[7] to transform the original treebank into the new format.

According to the new DTD, the phrase *de Westerse wereld* is represented as in figure 2. This encoding has several aspects worth noting:

- Intermediate layers are eliminated.
- Information is stored in attributes wherever possible. In particular, all XML elements in the old DTD which contained strings only are converted into attributes.
- The distinction between phrasal nodes and leaves is expressed by treating the latter as empty element tags of type NODE.
- Information about the (inflected) word form is added and information about linear order is made explicit.

Elimination of intermediate layers, and the use of attributes over elements led to a 90% reduction of the size of the treebank (from 70Mb to 7Mb). Note that this reduction is possible in spite of the fact that information (about string and head positions and word form) has been added or made explicit. The reduction in size also implies that queries are evaluates much faster. A comparison of equivalent queries on the original and compact encoding learns that query evaluation is about 5 times faster on the compact encoding.

As the new format is much simpler to grasp than the original encoding, queries can easily be formulated in XPath directly, and the need for a seperate query expansion layer is much less obvious. For instance, the following query corresponds to the XPath query used for illustration in the previous subsection:

```
//node[@cat="smain" and @rel="top"
    and ./node[@pos="adj"
            and @rel="mod"]]
```

This query has a complexity which is comparable to the query accepted by the query expansion method.

5. Dependency trees and linear precendence

Linguistic exploration of a syntactic treebank often involves search for patterns which are characterized both in terms of dependency and linear order. For instance, one might be interested in the question how often a direct object precedes an indirect object. Dependency trees primarily encode grammatical relations, and allow discontinuous parts of the input string to form a single dependent.

The direct encoding of dependency trees in XML presented above makes querying for dependency relations straightforward, but it is not clear whether this encoding also supports queries in which linear precedence plays a role. Note, for instance, that the order of elements in the XML document is typically not isomorphic with the order of words in the input string. Mengel and Lezius (2000)

[7]http://www.clip.dia.fi.upm.es/Software/

```
<node rel="cnj" cat="np" start="9" end="12" hd="12">
  <node rel="det" pos="det" start="9" end="10" hd="10" root="de" word="de"/>
  <node rel="mod" pos="adj" start="10" end="11" hd="11" root="westers" word="westerse"/>
  <node rel="hd" pos="noun" start="11" end="12" hd="12" root="wereld" word="wereld"/>
</node>
```

Figure 2: Compact encoding

argue against a direct tree based encoding of dependency trees, exactly because of the conflict between dependency and linear order. They opt for an approach where each word in the input is an element, which is linked to higher elements, such as phrases, by an indexing scheme. A similar solution is proposed in the XCES project (Ide et al., 2000). We believe that such solutions introduce a certain amount of needless complexity into the encoding, which can be an obstacle for querying the treebank with XPath directly.

The encoding of linear order in the XML documents produced during the annotation process is not optimal. Only the string position of words are encoded, and even this is only encoded as part of the string value of the WORD element. Not even the more advanced features of XPath suffice to search for specific linear order patterns in this case.

In the compact encoding, information about linear order has been made explicit. Not only the string positions of words are represented, but also the start and end positions of phrases. For discontinuous phrases, the leftmost word defines the start position, and the rightmost word defines the end position. It is not the case, therefore, that a phrase with start B and end E actually consists of all material between B and E. For instance, a direct object with an extraposed relative clause preceding and indirect object (*Kim showed the book to the students that they had to study for their exam*) has both a start preceding the indirect object and an end following the indirect object. Questions concerning linear order now become ambiguous: to find out whether A precedes B, one can ask about the order of the respective start or end positions, or one can ask whether the end of A precedes the start of B.

From a linguistic point of view, it seems useful to add another string position. If phrases are discontinuous, one often decides on linear order on the basis of the head. I.e. in the example above, it seems natural to decice that the direct object precedes the indirect object because the head of the direct object precedes that of the indirect object. Adding the location of the head to each phrase allows precedence queries to be formulated as queries concerning the relative positions of the respective heads.

6. Example Queries

We now present a number of examples which illustrate that the new encoding supports various types of linguistic queries.

Objects of prepositions are usually of category NP. However, other categories are not completely excluded. The query

```
//node[@cat="pp"]/node[@rel="obj1"]
```

finds the objects within PP's. In the Alpino dependency treebank, 98% (5,892 of 6,062) of these are regular NP's. The remainder is formed by relative clauses (*voor wie het werk goed kende, for who knew the work well*), PP's (*tot aan de waterkant, till on the waterfront*), adverbial pronouns (see below), and phrasal complements (*zonder dat het een cent kost, without that it a penny costs*).

The annotation guidelines distinguish between three possible dependency relations for PP's: complement, modifier, or 'locative or directional complement' (a more or less obligatory dependent containing a semantically meaningful preposition which is not fixed). Assigning the correct dependency relation is difficult, both for the computational parser and for human annotators. The following query finds the head of PP's introducing locative dependents:

```
//node[@rel="hd" and ../@cat="pp"
    and ../@rel="ld"]
```

Note that in this query, we are looking for a node with dependency relation *hd*, which is *dominated* by a PP with a *ld* dependency relation. Here, we exploit the fact that the mother node in the dependency tree corresponds with the immediately dominating element in the XML encoding as well.

Comparing the list of matching prepositions with a general frequency list reveals that about 6% of the PP's are locative dependents. The preposition *naar* (*to*, *towards*) typically introduces locative dependents (50% (74 out of 151) of its usage), whereas the most frequent preposition (i.e. *van, of*) does introduce a locative in only 1% (15 out of 1496) of the cases.

In PP's containing an impersonal pronoun like *er* (*there*), the pronoun always precedes the preposition. The two are usually written as a single word (*eraan, there-on*). A further peculiarity is that pronoun and preposition need not be adjacent (*In Delft wordt* **er** *nog* **over** *vergaderd In Delft, one still talks about it*). As traditional grammar classifies these impersonal pronouns as adverbs, they are encoded as such in the treebank as well. The following query finds such discontinuous phrases:

```
//node[@cat="pp" and
    ./node[@rel="obj1"
        and @pos="adv"]/@end <
     ./node[@rel="hd"]/@start ]
```

The corpus contains 110 discontinuous PP's containing an impersonal pronoun vs. almost 500 continuous pronoun-preposition combinations, realized as a single word, i.e. in almost 20% of the cases, the preposition + impersonal pronoun construction is discontinuous.

In Dutch subordinate clauses, past and passive participles may either precede or follow the auxiliary. The first option (OBJ PPART AUX, i.e. *dat Jan weinig wedstrijden* **gereden heeft** , *that Jan few races participated has*) gives rise to 'nesting' word order whereas the second (OBJ AUX PPART, i.e. *dat Jan weinig wedstrijden* **heeft gereden**) gives rise to so-called 'crossing dependency' word order. The following query matches all subordinate clauses with a past participle and crossing dependency word order:

```
//node[@cat="ssub" and
    ./node[@rel="hd"]/@hd <
        ./node[@cat="ppart"]/@hd ]
```

Note that we are looking for patterns where the embedded VP headed by the participle is discontinuous. The position of the head of the VP is used to determine whether the past participle precedes or follows the verb. By reversing the sense of the comparison, we find all nesting word orders. A count of the matching substrings reveals that nesting word order is found in 15% of the cases (45 out of 282). In non-finite clauses (i.e. in cases where the auxiliary is itself governed by a modal, almost 30% (43 out of 147) of the clauses has the participle preceding the auxiliary (i.e. PPART MODAL AUX, *dat een correctie* **toegepast** *moet* **worden**, *that a correction applied must be*). The other 70% has the auxiliary preceding the participle (i.e. MODAL AUX PPART, *dat een correctie moet* **worden toepgepast**). Finite modal verbs selecting an infinitival complement in principle allow nesting word order as well, however here we find that this occurs in only 2% of the cases (5 out of 269).

7. Conclusions

The XML encoding of linguistic databases can be verbose and complex, especially if the encoding is partially generated automatically and needs to support interaction with editing tools. We have presented a compact format for encoding dependency trees in XML which can be derived from a much more complex encoding by means of a transformation. The structure of the resulting XML document directly corresponds with the structure of the dependency tree. The new format supports a range of linguistic queries to be formulated in XPath directly. It makes information about word order explicit, which allows queries concerning dependency, linear order, and discontinuous constituency to be stated.

8. References

R. H. Baayen, R. Piepenbrock, and H. van Rijn. 1993. *The CELEX Lexical Database (CD-ROM)*. Linguistic Data Consortium, UPenn, Philadelphia, PA.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide-coverage computational analysis of Dutch. In *Computational Linguistics in The Netherlands 2000*. Rodopi, Amsterdam.

Gosse Bouma. 2001. Extracting dependency frames from existing lexical resources. In *Proceedings of the NAACL Workshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations*, Somerset, NJ. Association for Computational Linguistics.

D. Cabeza, M. Hermenegildo, and S. Varma. 1996. The pillow/ciao library for internet/www programming using computational logic systems. In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September.

Jo Calder. 2000. Thistle and interarbora. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 992–996, Saarbrücken.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: A survey and a new proposal. In *Proceedings of the LREC 1998*, pages 447–454, Granada, Spain.

David Carter. 1997. The TreeBanker: A tool for supervised training of parsed corpora. In *Proceedings of the ACL Workshop on Computational Environments For Grammar Development And Linguistic Engineering*, Madrid.

Michael Collins. 1999. *Head-driven Statistical Models for Natural Language Processing*. Ph.D. thesis, University of Pennsylvania.

E. Hajicova, J. Panevova, and P. Sgall. 1998. Language resources need annotations to make them really reusable: The Prague Dependency Treebank. In *Proceedings of LREC 1998*, pages 713–718, Granada, Spain.

N. Ide, P. Bonhomme, and L. Romary. 2000. XCES: An XML-based standard for linguistic corpora. In *Proceedings of LREC 2000*, pages 825–30, Athens, Greece.

Andreas Mengel and Wolfgang Lezius. 2000. An XML-based encoding format for syntactically annotated corpora. In *Proceedings of LREC 2000*, pages 121–126, Athens, Greece.

Michael Moortgat, Ineke Schuurman, and Ton van der Wouden. 2000. CGN syntactische annotatie. Internal Project Report Corpus Gesproken Nederlands, see http://lands.let.kun.nl/cgn.

Nelleke Oostdijk. 2000. The Spoken Dutch Corpus: Overview and first evaluation. In *Proceedings of LREC 2000*, pages 887–894.

Carl Pollard and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. Center for the Study of Language and Information Stanford.

Robbert Prins and Gertjan van Noord. 2001. Unsupervised pos-tagging improves parsing accuracy and parsing efficiency. In *IWPT 2001: International Workshop on Parsing Technologies*, Beijing China.

Wojciech Skut, Brigitte Krenn, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC.

P. C. Uit den Boogaart. 1975. *Woordfrequenties in geschreven en gesproken Nederlands*. Oosthoek, Scheltema & Holkema, Utrecht. Werkgroep Frequentie-onderzoek van het Nederlands.

Gertjan van Noord and Gosse Bouma. 1997. Hdrug. a flexible and extendable environment for natural language processing. In Dominique Estival, Alberto Lavelli, and Klaus Netter, editors, *Computational Environments for Grammar Development and Linguistic Engineering*, pages 91–98, Madrid.

Gertjan van Noord. 2001. Robust parsing of word graphs. In Jean-Claude Junqua and Gertjan van Noord, editors, *Robustness in Language and Speech Technology*. Kluwer Academic Publishers, Dordrecht.