Linguistic parsing of lists in structured documents

Salah Aït-Mokhtar **Veronika Lux**
Xerox Research Centre Europe
6, chemin de Maupertuis
38 240 Meylan, France
{ait,lux}@xrce.xerox.com

Éva Bánik*
Linguistics Department
University of Pennsylvania
Philadelphia, USA
ebanik@babel.ling.upenn.edu

Abstract

This paper shows how taking document structure into account helps to improve the performance of linguistic parsing. We restrict our study to one specific structure in a single markup language: lists in HTML documents. First we establish a typology of lists based on a corpus study. Then, after describing a transformation process that creates documents with uniform list markup, we show how the list tags can be incorporated into a parsing system, and how they enhance performance at every level of parsing.

1 Introduction

Due to the availability of huge collections of electronic documents and the rising needs to process them automatically, a lot of recent research on parsing has considered robustness as a key issue. There exist now many robust parsers (Grefenstette and Tapanainen, 1996; Abney, 1996; Collins, 1996; Aït-Mokhtar and Chanod, 1997; Tapanainen and Järvinen, 1997; Srinivas and Joshi, 1999; Aït-Mokhtar et al., 2002) capable of handling free texts, i.e., capable of producing (at least partially) correct analyses for real-world input texts. However, the accuracy of such parsers, that is, the quality of their linguistic analyses, is not homogeneous over different types of texts. For instance, (Gala Pavia, 2001) shows how the linguistic accuracy of two existing robust parsers significantly varies depending on corpus types (technical, newspaper articles, etc.)

The variations are not only due to differences in grammatical style. Documents may also contain specific (non-linguistic) layout structures, which are intended to attract attention and make the text more informative and readable. Examples of layout structures are headings, lists, tables, frames, etc. Such structures are so intensively used in web pages for instance that continuous regular text sometimes becomes a rare "commodity".

Even though those specific structures do not prevent a robust parser from processing the input text, they nevertheless deteriorate the accuracy of its output. There are two reasons for this misbehavior: first, layout structures contradict the traditional assumption that the parser's input unit is *a sentence*, which is usually defined as a segment of text containing a (usually finite) main clause ending in a dot. Second, layout structures cause some distorsions in the regular grammatical schemes and these distorsions are not predicted by the parser.

In this paper, we focus on the *list* structure and argue that the quality of robust parsing can benefit from the correct recognition and handling of lists in documents. In order to facilitate the maintainance and evolution of layout structure processing, we decided to define a simple and uniform XML annotation for lists. In the spirit of (Grover et al., 2002; Gala Pavia, 2003), the documents are first annotated and then linguistically processed.

*The work described here was done during an internship at XRCE.

Hence, we propose an architecture (see figure 1) based on two main processing steps:

- A document transformation process that takes the initial document and produces a document where the interesting structures (here, lists) are unambiguously marked according to a uniform XML annotation.
- A parsing step where, besides the regular grammar rules, specific rules are applied to handle the list structure, using lexical entries assigned to the list markup tags.

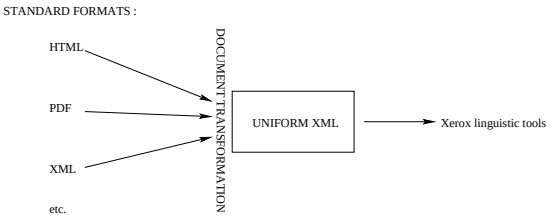

Figure 1: General architecture

The first step is meant to yield a uniform list annotation that will be handled by the parser. This step is necessary because lists are not annotated consistently across documents. In HTML documents for instance, several markup tags can be used for list annotation. In plain text, lists are indicated with punctuation symbols and special characters (semicolons, newlines and hyphens for instance) that have other possible functionalities. Therefore, transformation rules are written for each new type of document (that shows a particular list structure annotation) so that any document ends up annotated according to the uniform XML annotation. In the present study, we concentrate on HTML documents and show how they are transformed so that the list structures are marked up with the uniform annotation. Document transformation is performed using the Circus system (Vion-Dury et al., 2002).

The second step is the linguistic processing itself, where specific linguistic information is added for the proper handling of lists. It includes the assignment of lexical entries to the list markup tags and the addition of specialized parsing rules to the regular grammar. The specific rules only apply to list structures and have the suitable property of not disturbing the behaviour of the existing regular parsing rules. We use the Xerox linguistic tools, mainly the XIP (Xerox Incremental Parser, (Aït-Mokhtar et al., 2002)) parsing system. We show how handling lists can improve the XIP English parser at all levels: sentence segmentation, POS tagging, chunking and dependency parsing.

In the next sections, we detail a corpus-based study of lists and the two step analysis.

2 Linguistic characteristics of lists

We have chosen to focus on lists since they are an interesting example for illustrating that document structure and linguistic structure are interdependent. The lists that we study here consist of a list introduction and a set of list items. We are not concerned with sets of list items without an introduction[1].

Wwe collected a corpus documents that contain lists . It consists of users' guides and technical documents from different areas (fruit processing manual, isbn manual, career development manual, protocol description manual, equipment maintenance manual). Our corpus contains 81 513 words (without markup) and 410 lists.

In the next subsections, we give a brief description of lists, which seem to be rather special objects from a linguistic point of view ((Nunberg, 1990; Douglas and Hurst, 1996; Pascual and Virbel, 1996; Luc et al., 1999)).

From a syntactic point of view, lists are either made up of:

- a syntactically incomplete introduction followed by one or more items that are phrases of the type missing from the introduction, or
- a syntactically complete phrase or sentence followed by one or more items of the same syntactic category.

For linguistic processing the first question that emerges is: how to segment the text that is provided to the parser? Clearly, if the introduction and the list items are provided as distinct units,

[1]Note that lists where a set of items is followed by a conclusion seem exceptional (one case in our corpus) and thus are not considered here.

the parsing, in particular the extraction of syntactic dependencies, cannot be correct.

From a semantic point of view, the list structure conveys additional information: in particular, each item works as a "scope" delimiter. Hence, the list structure can be used to resolve syntactic ambiguities for which semantic knowledge would otherwise have been necessary. For instance in the following example :

> Session manager saves and restores :
> - the appearance and behavior settings
> - the window applications that were running

the list structure helps to correctly attach the relative clause "that were running" to the noun phrase "the window applications".

From a general point of view, one could even argue that lists occasionnally question the assumption that texts are sequences of sentences to be processed in a linear way.

> **The special options available to the user are:**
> **[H]istogram Screen**: This will go to the histogram screen described above.
> **[B]lank Screen**: This option will clear the screen making avida run marginally faster (since it will not be wasting much CPU time on the display)
> **[R]edraw Screen**: If the screen gets garbled, this will erase it and refresh all text which is supposed to appear.
> **[C]hoose New CPU**: This option will put avida in map mode with the cursor on the screen. Position the cursor over the CPU you would like to select, and press.

But this would lead us far beyond the scope of our work here.

Based on our corpus study we categorize lists depending on the type of syntactic and/or semantic dependencies between the introducer and the items[2].

2.1 Lists with syntactic dependencies between the introducer and the items

These are lists where the introduction is a syntactically unsaturated sentence, lacking an argument (object or other argument of a verb, subject, complement of a preposition, a complementizer or a relative pronoun (e.g. *for, that, where*), etc.).

[2]This criterion is determined by our objective, i.e. enhancing the syntactic analysis and particularly the extraction of syntactic dependencies.

> This section will help you to:
> - Discover the wide range of work available
> - Find out the specifics of a particular job
> - Detect work-related trends

List items can also be modifiers of the introduction.

> The ISBN must appear:
> - on the verso of the title page (copyright page)
> - on the lower section of the outside back cover
> - on the foot of the title page, if there is no space elsewhere
> ...

Conversely, in some lists, the introduction is a modifier of the items. This occurs often in the case of sequences of instructions where the introduction phrase expresses a goal or a condition, etc.

> To delete a custom property:
> - From the DocuShare Administration toolbar, click Object Properties.
> - Click the name of the object where you want to delete a custom property.

In all these cases, if we fed the parser the introduction and each item separately, some syntactic dependencies would be missed.

2.2 Lists with no syntactic dependencies between the introducer and the items

In the following cases, there are no syntactic dependencies between the introducer and the items. The introduction can be a nominal group similar to a title or it can be a complete sentence. From a syntactic point of view, the introduction and each item can be input to the parser separately without any loss in the quality of the analysis.

> Installation:
> - Slide the SOP into the telescope body.
> - Press the SOP up into the rail and install the screws.

However, from a semantic point of view, it can be useful to parse the introduction and list items as one unit since there seem to be dependencies to be extracted at this level as well (as underlined by (Gala Pavia, 2003)). Some lists where both the introduction and the items are syntactically complete phrases (or sentences) seem to follow typical lexical or syntactic patterns that could allow us to extract semantic dependencies. These are cases where the introduction contains a 'keyword' or an indicator phrase which acts as a 'placeholder':

The areas of the school building are [the following]:
- Structure
- Roofing
- Building exterior
- Building interior

Such indicators are phrases that contain numbers or the words *following, including, below, as follows*, etc. In some cases lists with an indicator (or placeholder) in the introduction can be transformed into a grammatical sentence by substituting the coordinated list items in the introduction (in the place of the indicator).

When the indicator phrase is a syntactic argument of a head in the introduction (e.g. a preposition or finite verb which is not the copula), the presence of the placeholder gives a good indication to extract a (semantic) relation between the list items and the introduction. In other words, for these particular cases there seems to be quite a direct "mapping" between a syntactic dependency (the one between the indicator and the head of the introduction) and a semantic dependency (the one between each list item and the introduction).

3 Document transformation

As illustrated in Figure 1, the document transformation process is part of a pre-processing step.

The output of the transformation is a "best suited" input for the parser. Our transformation aims at converting list structures expressed in various formats (e.g. HTML, PDF) into structures using a pivot format known by the parser (here, XML tags of a given vocabulary). This way, the linguistic parser itself does not need to include information about the many existing document formats.

To transform documents we need a formal specification of the transformation and a transformation engine. For example, if both the input and output of the transformation were XML documents, we could write a transformation in XSL-T and execute it with Xalan.

Since our goal in the long-term is to transform structured documents with various formats, we need a technical solution that is more general than XSL-T.

Circus (Vion-Dury et al., 2002; Vion-Dury, 2002), a programming language specialized in structure transformation and developped at XRCE, meets such needs[3]. Circus has been designed to address general structure transformation problems and is especially well-suited to the task of programming XML transformations. Its library includes dedicated components for XML and HTML processing.

In the XML world, Circus is positioned at an intermediate level of abstraction between XSLT and lower-level approaches such as DOM+Java (general purpose programming language) and should provide the right balance between expressiveness and ease of use.

In the work presented here, we focus on list structures and restrict ourselves to HTML. HTML is widely used on the internet (which facilitates the collection of an HTML corpus). Furthermore, HTML provides elements to mark lists but these are oriented to layout considerations; thus the transformation to our XML format for lists is feasable but not trivial.

In the next sections we give a brief description of lists in HTML based on the same corpus as the linguistic study. We then sketch the transformation that was implemented in Circus and provide some elements for a discussion on our two-step architecture for the linguistic processing of structured documents.

3.1 Markup characteristics of HTML lists

One first striking point is that the HTML understanding of lists is not the same as our understanding of lists from a linguistic point of view : lists in HTML are only sets of items and list introductions do not have a dedicated associated element.

The transformation from HTML lists to our XML list format will have to enrich the structure with information on list introductions.

Another striking characteristic of the corpus is the heterogeneity in the list markup. As allowed by the HTML specification, lists in our corpus are formatted in several ways :

- ordered lists and unordered lists are marked with <ol> and <ul> tags respectively, their elements with <li> tags

[3]For more information about Circus, see http://www.alphaave.world.xerox.com/

- definition lists are marked with <dl>, their elements with <dd> and <dt> tags
- other lists are marked with <menu> tags

Besides these "legal" possibilities, our corpus also includes other kinds of HTML markup that are sufficient to display a list presentation in a web-browser. In particular, combinations of
 and <p> tags are used very frequently.

The transformation from HTML lists to our XML format will have to standardize all these list markups.

3.2 Transformation of HTML documents with Circus

Document transformation is carried out by a Circus program that discards all HTML tags except <ul>, <ol> and <menu> tags, which are transformed into <list>. We also keep <li> tags. Inside lists, the <p> tags that are immmediate descendants of <list> (i.e. paragraphs that are sisters of or replace <li>), are transformed into <li> tags. The last sentence of the paragraph that precedes the list is marked with <listintro> tags. The 'last sentence' is the part of the paragraph that is:

- between the last two punctuation marks if the paragraph ends in a punctuation mark ('.', ':', '!', '?')
- the substring following the last punctuation mark if there's no paragraph-final punctuation
- the whole paragraph otherwise

Embedded lists (lists that are contained in list items) are marked up similarly. The final punctuation in list introductions is marked with <punct> or <colon> tags and finally, the first colon of a list item is marked as <colon>, except when the list item is itself an introduction, in which case it is marked up accordingly.

Figure 2 provides examples of the intput and output of the transformation process.

3.3 Discussion

Although our transformation seems straigthforward, interesting questions emerge concerning the division of labor between the transformation step and the parser. For instance, it seems that some processing can be performed either in the document transformation phase or during linguistic analysis. This is illustrated with the following example of list organizers:

> The add-on information may have the following formats:
> 1.five-digit bar code indicating the price with
> human-readable numbers above the bar code, or
> 2.five-digit bar code indicating that no price is present,
> with the human-readable numbers of 90000
> above the bar code.

In our current system, list organizers are grouped into a special chunk by the parser to distinguish them from other, linguistically meaningful constituents. In most cases however this grouping could also be performed by the document transformation engine e.g. by adding attributes to list items (e.g. <li attribute='organizer' value='1'>).

Handling list item organizers in the transformation process has the advantage that numbers and punctuation like '1.' would no longer need to be taken into account by the linguistic parser.

The question is then: how do we decide where it is best to do a particular processing ?

4 Linguistic processing

For linguistic parsing we use the Xerox Incremental Parser (XIP) (Aït-Mokhtar et al., 2002), a robust parser that produces text annotated with phrase chunks and dependency relations. Like many robust parsers, XIP performs linguistic analysis in several steps: tokenization and morphological analysis, part-of-speech (POS) tagging, chunking and syntactic dependency recognition.

The input text is first tokenized, i.e. segmented into tokens (wordforms, punctuation symbols, markup tags, etc.) and morphologically analyzed. The morphological analyzer might assign more than one POS tags to some tokens so the output is passed to an HMM-based disambiguator module which determines the most likely POS for each token given the context. After this stage the resulting list of tokens and POS tags is input to the XIP grammar rules. Characteristic mistakes in the output of the HMM tagger are corrected by hand-written *tagging rules*; words are grouped

```
INPUT:

 <html>
  <head>
 <title> </title></head>
<body>
<p><br>This is a paragraph. With a
<b>list</b><u>introduction</u>.
<i>This</i> is the introduction:
</p>
<ol TYPE="A">
<li><u>LIST ITEM ONE.</u></li>
<li><u>LIST ITEM TWO</u> followed
 by an embedded list.
 This is the intro :</li>
<ol>
 <li>Title: Some list item</li>
 <li>Another list item</li>
 <li>and this is the last list
   item: final list item</li>
</ol>
</ol>
This is outside of the list.
</body>
</html>
```

```
OUTPUT

This is a paragraph. With
a list introduction.

<listintro>This is the
introduction<punct>:</punct>
</listintro>
<list>
<li>LIST ITEM ONE.</li>
<li>LIST ITEM TWO followed
 by an embedded list.
 <listintro>This is the intro
 <punct>:</punct></listintro>
 </li>
 <list>
 <li>Title <colon>:</colon>
  Some list item </li>
 <li>Another list item</li>
 <li>and this is the last
 list item<colon>:</colon>
  final list item</li>
 </list>
</list>
This is outside of the list.
```

Figure 2: Input and output of the Circus transformation

into phrase chunks by *chunking rules* and finally, *dependency rules* are used to find syntactic relations between elements.

The XML tags introduced in the document transformation step are part of the parser input sequence just like words, although the lexical analysis gives them specific morphosyntactic readings. These readings consist of a main category (LIST_XML) and specific features that depend on the role of the markup tag in the list structure. For instance, the XML tag `<listintro>`, which indicates the beginning of a list introduction, is assigned the `intro_beg` feature, yielding the specific category `LIST_XML[intro_beg]`, as defined in the following lexical entry:

```
<listintro> = LIST_XML[intro_beg=+].
```

These morphosyntactic readings are used in all steps of parsing and help improve the performance of the parser in the analysis of lists. Among other effects, they allow for (a) a suitable segmentation of lists (as single input units for syntax), (b) the correction of some POS tagging errors in the context of lists, (c) a sound application of chunking rules over list items, and (d) the recognition of dependency relations between list introductions and list items.

4.1 Segmenting text into parsing units

The text is segmented into fragments (usually sentences) for the linguistic analysis using regular pattern rules. The patterns are based on specific punctuation marks (such as '.' or '!' or ':'). Lists often include such punctuation marks and thus are often segmented in an awkward way.

The XML markup allows for the writing of list-aware segmenting rules, which ensures that lists are handled as single units. The following segmenting rule handles list segmentation:

```
|LIST_XML[intro_beg],?*,LIST_XML[list_end]|
```

It defines any sequence starting with the XML list introduction tag (whose linguistic category is `LIST_XML[intro_beg]`) and ending with the XML list ending tag (`LIST_XML[list_end]`) as a single input unit to the parsing process.

4.2 POS tagging

The HMM tagger sometimes assigns POS incorrectly in cases where the immediate context considered is not informative enough to resolve ambiguity. To correct some of the mistakes, we use hand-written XIP disambiguation rules in which a larger context can be taken into account.

Based on the XML markup, additional disambiguisation rules can be written to correct the POS tagging errors in the particular context of lists. For example, *that* at the end of a list introduction is not a pronoun but most often a complementizer (CONJ). Hence, the following rule:

```
PRON[form:fthat,rel:~] %=
    CONJ |LIST_XML[intro_end]|.
```

corrects the POS error for the word *that*, when it occurs at the end of a list introduction (the category of which is `LIST_XML[intro_end]`). It replaces the pronoun reading (PRON) with a complementizer reading (CONJ).

4.3 Chunking

When list structures are not explicitly marked, chunking rules can apply across item boundaries (when the list items do not end with a strong punctuation mark) yielding awkward chunks. For instance, in the following list a single NP chunk was built including two nouns from two different list items.

> Such methods include:
> Canning
> Concentration

The XML tags now prevent NP chunking rules from applying across item boundaries.

4.4 Syntactic relations

In the absence of XML markup, the list introduction is not handled as a single input unit together with the list items because it is often ended with a strong punctuation mark (usually a colon). Therefore syntactic relations (e.g. verb-object relation, verb-modifier relation, etc.) between the introduction and the items are missed.

Using XML tags to mark up lists, they are considered as a single input unit and specific parsing rules can be written in order to extract syntactic relations. For the example mentioned above, the relations *OBJECT(include, canning)* and *OBJECT(include, concentration)* are extracted. The following is a specific dependency rule that extracts the object dependency between the main verb of the list introduction and the noun heads of the list items:

```
|INTRO{?*,SC{?*,FV#1},LIST_XML[intro_end]},
 LIST_XML,ITEM*,
 ITEM{BULLET,NP{?*,?#2[last]}}|
OBJ(#1,#2).
```

The regular tree pattern of the rule says that, whenever there is a finite verb (`FV#1`) in the main clause (`SC`) of the list introduction (`INTRO`), and a noun head (`#2`) in the main NP of any of the items of the list (`ITEM`), then there is an object dependency relation between the finite verb and the noun head (`OBJ(#1,#2)`).

Besides making it possible to capture the otherwise missed dependency relations, XML tags also allow for specific rules that take advantage of the "good properties" of the list structure (e.g. scope for the attachement of prepositional phrases is limited by the item boudaries).

5 Conclusion and future work

We have reported our first experiment on using document structure information to improve parsing results. Using the list structure in HTML documents as an example we have showed that defining a pivot XML format can help improve linguistic parsing. We have also pointed out certain technical issues concerning the definition of a general architecture, more specifically, different types of collaboration between an XML parser and a linguistic parser.

In the future we would like to extend the implementation of XIP rules that are specific to lists, adding rules for syntactic as well as semantic dependencies.

Another priority for future work is, of course, evaluation. We conducted some preliminary tests to evaluate list structure annotation and linguistic chunking within lists but these are clearly not sufficient for the evaluation of the quality and benefits of our system. We plan to evaluate the performance of the augmented parser on list structures (precision and recall of dependency relations), and see how it compares to the original parser (with no

list-specific features).

We can think of several directions in which this research could be extended in the future.

First of all, the list transformation process could be made more robust to handle "non standard" uses of HTML for marking up lists. The XML markup in the output of the transformation could also be enriched to include more information about document structure (e.g. a different font type used at the beginning of list items often reveals a title for the list item, which should be considered separately from the list item body and could be tagged explicitly). Finally, the correct processing of imbricated lists requires further studies.

More generally, the document transformation process should handle other structures than lists (still to be defined). More work is needed to complete the definition of a pivot XML format suited for linguistic analysis and to achieve the architecture sketched in the introduction of this paper.

Acknowledgements

We would like to thank all the people who gave valuable comments on a previous version of this paper: Jean-Pierre Chanod, Caroline Hagège, Aaron Kaplan, Aurélien Max, Ágnes Sándor and François Trouilleux.

References

Steven P Abney. 1996. Partial Parsing Via Finite-State Cascades. In *ESSLLI'96 Workshop on Robust Parsing*, Prague, Czech Republic.

Salah Aït-Mokhtar and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC, USA.

Salah Aït-Mokhtar, Jean-Pierre Chanod, and Roux Claude. 2002. Robustness beyond shallowness: incremental deep parsing. *Natural Language Engineering*, 8(2):121–144.

Michael J. Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, Santa Cruz, USA.

Shona Douglas and Matthew Hurst. 1996. Layout and language: Lists and tables in technical documents. In *Proceedings of ACL SIGPARSE Workshop on Punctuation in Computational Linguistics*, pages 19–24.

Núria Gala Pavia. 2001. A two-tier corpus-based approach to robust syntactic annotation of unrestricted corpora. *T.A.L*, 42(2):381–411.

Núria Gala Pavia. 2003. *Un modèle d'analyseur syntaxique robuste basé sur la modularité et la lexicalisation de ses grammaires*. Ph.D. thesis, Université Paris XI, March.

Gregory Grefenstette and Pasi Tapanainen. 1996. Light Parsing as Finite-State Filtering. In *Proceedings ECAI'96 workshop on "Extended finite state models of language"*, Budapest, Hungary.

Claire Grover, Ewan Klein, Alex Lascarides, and Maria Lapata. 2002. Xml-based nlp tools for analysing and annotating medical language. In *Proceedings of the Second International Workshop on NLP and XML (NLPXML-2002)*. Taipei.

Christophe Luc, Mustapha Mojahid, Jacques Virbel, Claudine Garcia-Debanc, and Marie-Paule Pery-Woodley. 1999. A linguistic approach to some parameters of layout: A study of enumerations. In *Using Layout for the Generation, Understanding or Retrieval of Documents, AAAI Fall Symposium*, pages 20–29. North Falmouth, Massachusetts, November.

Geoffrey Nunberg. 1990. *The linguistics of punctuation*. Center for the Study of Language and Information, Stanford.

Elsa Pascual and Jacques Virbel. 1996. Semantic and layout properties of text punctuation. In *Proceedings of the Association for Computational Linguistics Workshop on Punctuation*, pages 41–48, June. Santa Cruz, California.

Bangalore Srinivas and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 20(3):331–378.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC, USA.

Jean-Yves Vion-Dury, Veronika Lux, and Emmanuel Pietriga. 2002. Experimenting with the Circus language for XML modeling and transformation. In *Proceedings of the DocEng'2002 Symposium on Document Engineering*, Mac Lean, VA, USA.

Jean-Yves Vion-Dury. 2002. Circus reference manual.