Tools and resources for Tree Adjoining Grammars

François Barthélemy,
CEDRIC – CNAM,
92 Rue St Martin
FR-75141 Paris Cedex 03
barthe@cnam.fr

Pierre Boullier, Philippe Deschamp
Linda Kaouane, **Abdelaziz Khajour**
Éric Villemonte de la Clergerie
ATOLL - INRIA,
Domaine de Voluceau - BP 105
FR-78153 Le Chesnay Cedex
Eric.De_La_Clergerie@inria.fr

Abstract

This paper presents a workbench for Tree Adjoining Grammars that we are currently developing. This workbench includes several tools and resources based on the markup language XML, used as a convenient language to format and exchange linguistic resources.

1 Introduction

Our primary concern lies in the development of efficient parsers for various grammatical formalisms of interest for Natural Language Processing. Tree Adjoining Grammars [TAG] is one of these formalisms, important from a linguistic point of view but also because it is possible to design efficient parsers.

However, during our work on TAG, we were confronted with a lack of standardization of grammars, especially when dealing with wide coverage grammars. The XTAG System[1] (The XTAG Research Group, 1995) provides an implicit standard, but it is not very readable and lacks explicit specifications. The various grammars we studied presented many variations. Moreover, we also noted many problems of consistencies in most of them.

Following others, amongst whom LT XML[2] and especially (Bonhomme and Lopez, 2000), we considered that the markup language XML[3] would be a good choice to represent TAG, especially with the possibility of providing an explicit and logical specification via a DTD. Being textual, resources in XML can be read by humans and easily exchanged and maintained. Finally, there exists more and more supports to handle XML resources. We have also found that XML is a convenient language to store linguistic results, such as the shared derivation forests output by our TAG parsers.

The paper starts with a brief introduction to TAGs. Section 3 presents the different XML encodings that we have designed for the representation of grammars and derivation forests. Section 4 presents several different maintenance tools we are developing to handle grammars and derivation forests. Section 5 presents servers used to access different kind of informations. Interfaces for these servers are presented in Section 6.

2 Tree Adjoining Grammars

The TAG formalism (Joshi, 1987) is particularly suitable to describe many linguistic phenomena. A TAG is given by a set of *elementary trees* partitioned into *initial trees* and *auxiliary trees*. Internal nodes are labeled by non-terminals and leaves by non-terminal or terminals. Each auxiliary tree β has a distinguished leaf, called its *foot* and labeled by a non-terminal, the same as the root node of β.

Two operations may be used to derive trees from elementary trees. The first one, called *substitution*, replaces a leaf node labeled by a non-terminal A by an initial tree α whose root is also labeled by A. The second operation, called *ad-*

[1] http://www.cis.upenn.edu/~xtag/
[2] http://www.ltg.ed.ac.uk/
[3] http://www.w3c.org/XML/

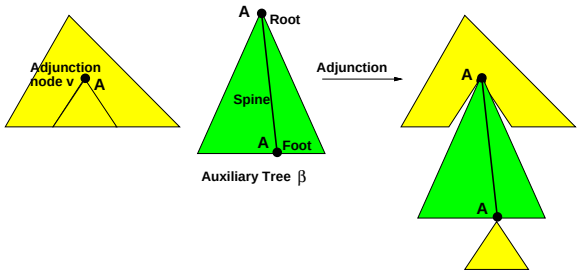

Figure 1: Adjunction

junction, is illustrated by Figure 1. An auxiliary tree β whose root is labeled by A may be adjoined at any node ν labeled by A. The subtree rooted at ν is grafted to the foot of β.

Feature TAGs extend TAGs by attaching to nodes a pair of first-order terms represented by *Feature Structures* [FS] and called *top* and *bottom* arguments. These feature structures may be used, for instance, to handle agreement or enforce semantic restrictions.

Lexicalized (Feature) TAGs assumes that each elementary tree has at least one lexical node labeled by a terminal. However, explicit lexicalized grammars would be huge, with one or more elementary trees for each entry in the lexicon. The choice made by the XTAG group and by all the designers of wide coverage TAGs is to factorize the grammars and gives enough information to lexicalize parts of the grammars when needed. Morphological entries (or inflected forms) reference one or more lemma entries, which, in turn, refer to families of tree schema. A tree schema is an elementary tree with a distinguished leaf called *anchor node* that is to be replaced by a morphological entry. Each reference may be completed by additional constraints.

For instance, extracted from a small French grammar, Figure 2 shows the basic elements (morphological entry `donne`, lemma `\DONNER\`, and tree schema `tn1pn2`) used to build the tree `tn1pn2(donne)` corresponding to the syntactic pattern (1) and illustrated by sentence (2). The lemma part states that the subject NP_0 and the prepositional complement NP_2 must both be human and that NP_2 is introduced by the preposition *à* (co-anchoring). In the tree `tn1pn2`, the substitution nodes are marked with $\downarrow$ and the anchor node with $<>$.

(1) *quelqu'un* **donne** *quelque chose* **à** *quelqu'un*
somebody **gives** *something* **to** *somebody*

(2) Yves **donne** un joli livre **à** Sabine
Yves **gives** a nice book **to** Sabine

```
donne: \DONNER,\ V
       {mode=,indnum=sing}
\DONNER,\V: tn1pn2[p_2=à]
       {NP_0.t:restr=+,hum
        NP_2.t:restr=+hum}
```

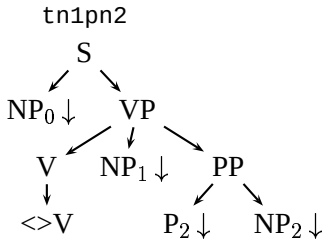

Figure 2: Tree schema

3 XML Encoding

3.1 Representing grammars

We have designed a DTD[4] that clearly specifies the relations between the various components of a grammar. For instance, the following DTD fragment states that a morphological entry is characterized by a field `lex` and includes zero or more description entries (for documentation) and at least one reference to a lemma (`lemmaref`). Similarly, an element `lemmaref` is characterized by the fields `name` and `cat`, and may be completed by a FS argument (`fs`).

```
<!ELEMENT  morph  (desc*,lemmaref+)>
<!ATTLIST  morph   lex   CDATA   #REQUIRED>
<!ELEMENT  lemmaref  (fs?)>
<!ATTLIST  lemmaref   name   CDATA   #REQUIRED
                      cat    CDATA   #REQUIRED>
```

Following the DTD, the various elements described in Figure 2 may be represented by the (tiny) following XML fragment, omitting the FS specification on nodes for sake of space and clarity.

```
<tag axiom="s">
  <morph lex="donne">
        <lemmaref cat="v" name="*DONNER*">
            <fs>
                <f name="mode">
                    <val>ind</val>
                    <val>subj</val>
```

[4] `http://atoll.inria.fr/~clerger/tag.dtd,xml`

```
                </f>
                <f name="num">
                    <val>sing</val>
                </f>
            </fs>
        </lemmaref>
  </morph>
  <lemma cat="v" name="*DONNER*">
        <anchor tree_id="family[@name=tn1pn2]">
            <coanchor node_id="p_2">
                <lex>à</lex>
            </coanchor>
            <equation node_id="np_0" type="top">
                <fs>
                    <f name="restr">
                       <val>plushum</val>
                    </f>
                </fs>
            </equation>
            <equation node_id="np_2" type="top">
                <fs>
                    <f name="restr">
                        <val>plushum</val>
                    </f>
                </fs>
            </equation>
        </anchor>
   </lemma>
 <family name="tn1pn2">
  <tree name="tn1pn2">
   <node cat="s" adj="yes" type="std">
    <node cat="np" id="np_0" type="subst" />
    <node cat="vp" adj="yes" type="std">
     <node cat="v" adj="yes" type="anchor" />
     <node cat="np" type="subst" />
     <node cat="pp" adj="yes" type="std">
      <node cat="p" id="p_2" type="subst"/>
       <node cat="np" id="np_2" type="subst"/>
     </node>
    </node>
   </node>
  </tree>
 </family>
</tag>
```

Currently, we have encoded a small French grammar (50 tree schemata, 117 lemmas and 345 morphological entries) and an English grammar (456 tree schemata, 333 lemmas and 507 morphological entries). We are processing some other larger grammars (for both English and French).

3.2 Encoding derivations

A (deterministic) TAG parser may return either the result of the analysis as a parse tree, or the steps of the derivation as a derivation tree. These two alternatives are illustrated for sentence (3) by Figures 3 and 4 (with Figure 5 showing the elementary lexicalized trees). A derivation tree indicates which operation (substitution or adjunction of some tree) has taken place on which node of which tree, for instance the adjunction of tree a(joli) at node labeled N. It is worth noting that the parse tree may be retrieved from the derivation tree, which motivates our interest in derivation trees.

(3) Yves donne un joli livre à Sabine
Yves gives a nice book to Sabine

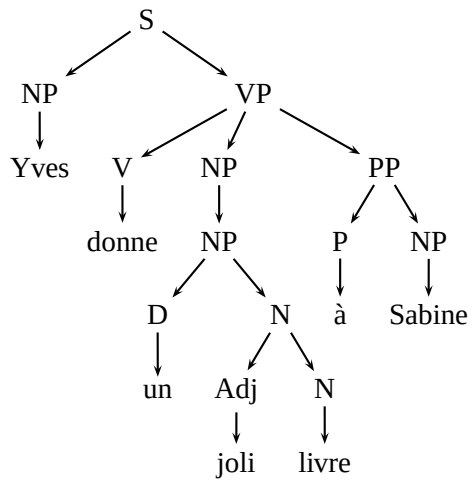

Figure 3: Parse Tree

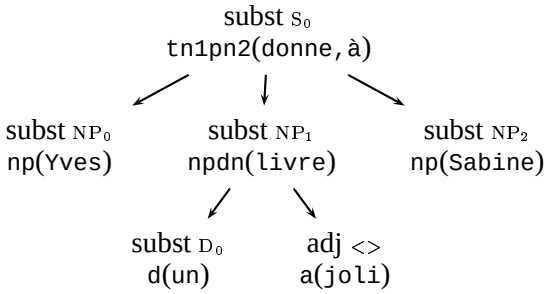

Figure 4: Derivation Tree

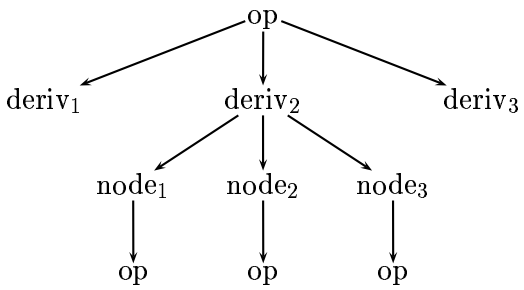

Figure 6: Organization of a derivation forest

In case of ambiguity (frequent in NLP), several or even an unbounded number of derivation trees may actually be compacted into a *shared derivation forest*, equivalent to a Context-Free Grammar (Lang, 1991). This remark has guided the design

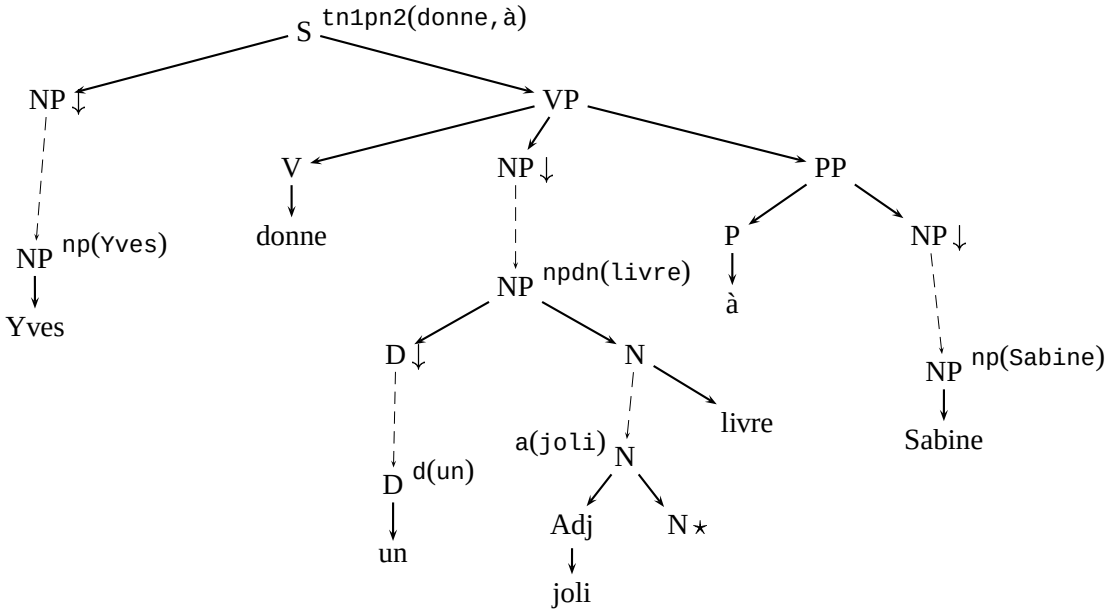

Figure 5: Elementary trees

of a DTD[5] to encode shared derivation forests. This DTD introduces the primary elements `op`, `deriv`, and `node` as well as an element `opref` used to reference elements `op`. The logical organization of these elements is sketched in Figure 6. More precisely:

op , identified by its attribute `id`, denotes either an operation of substitution or of adjunction (attribute `type`) on some syntactic category (attribute `cat`) for some span of the input string (attribute `span`). Sub-elements of `op` may also be present to specify the feature values associated to the operation.

deriv details a possible derivation for some operation, based on some lexicalized tree given by a tree schema (attribute `tree`) and an anchor (`anchor`).

node specifies which operation `op` has been performed on some node (attribute `node_id`) of an elementary tree during a derivation.

A derivation tree may be expressed in a nested way using only elements `op`, `deriv`, and `node`. A shared forest will require the use of `opref` to denote multiple occurrences of a same operation.

[5] `http://atoll.inria.fr/~clerger/forest.dtd,xml`

The above derivation tree may be represented by the following XML fragment (omitting information about the feature structures).

```
<forest parser="Light DyALog">
  <sentence> Yves donne un joli livre à Sabine
  </sentence>
  <op cat="s" span="0 7" id="1" type="subst">
    <deriv tree="tn1pn2" anchor="donne">
      <node id="p_2"><opref ref="5" /></node>
      <node id="np_0"><opref ref="2" /></node>
      <node id="1"><opref ref="4" /></node>
      <node id="np_2"><opref ref="6" /></node>
    </deriv>
  </op>
  <op cat="np" span="0 1" id="2" type="subst">
    <deriv tree="np" anchor="Yves" />
  </op>
  <op cat="np" span="2 5" id="4" type="subst">
    <deriv tree="npdn" anchor="livre">
      <node id="n_"><opref ref="10" /></node>
      <node id="0"><opref ref="8" /></node>
    </deriv>
  </op>
  <op cat="p" span="5 6" id="5" type="subst">
    <deriv tree="p" anchor="à" />
  </op>
  <op cat="np" span="6 7" id="6" type="subst">
    <deriv tree="np" anchor="Sabine" />
  </op>
  <op cat="d" span="2 3" id="8" type="subst">
    <deriv tree="d" anchor="un" />
  </op>
  <op cat="n" span="3 5 4 5" id="10" type="adj">
    <deriv tree="an" anchor="joli" />
  </op>
</forest>
```

4 Maintenance tools

4.1 For the grammars

The XML encoding of grammars is convenient for maintenance and exchange. However, it does

not correspond to the input formats expected by the two parser compilers we develop. One of them (DyALog) expects a prolog-like representation of the grammars (Alonso Pardo et al., 2000) while the second one expects Range Concatenation Grammars [RCG] (Boullier, 2000).

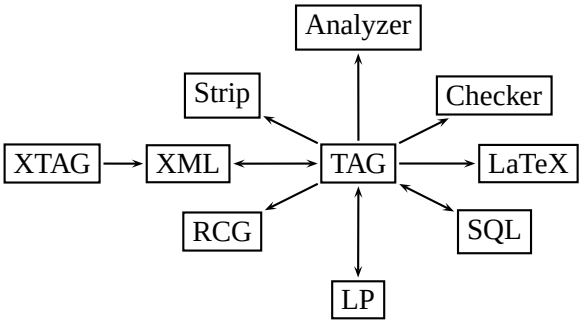

Figure 7: Maintenance Tools for the grammars

Therefore, we have developed in Perl a set of maintenance modules, for these conversions and for other tasks (Figure 7). The central module **TAG** implements an object-oriented view of the logical structure specified by the Grammar DTD. The other modules add new methods to the classes introduced by **TAG**.

Besides the conversion modules **LP** and **RCG**, we also have a read/write **XML** module. Module **Checker** is used to check (partially) the coherence of the grammar and to produce some statistics. Module **Analyzer** extracts information needed for the compilation by the DyALog system. Module **Strip** deletes all information relative to feature structures from the grammar. Module **SQL** may be used to store to and load from a SQL database.

Our choice of Perl has been motivated by the availability from archive sites of many Perl modules to handle XML resources or database access. Moreover, the development of a Perl module is fast (for a prototype), generally only a few hours. For instance, we have realized a prototype module **LaTeX**, useful to build the documentation of a grammar. We are also thinking of an **HTML** module to build online versions.

4.2 For the derivation forests

Similarly, we have also developed a set of modules to handle derivation forests (Fig. 8) with a

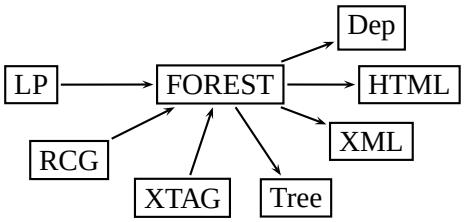

Figure 8: Maintenance Tools for the forests

central module **FOREST** and conversion modules. Modules **LP**, **RCG**, and **XTAG** read the output formats of the derivation forests produced by our parsers and by the TAG parser[6]. The forests can then be emitted in XML format (module **XML**), in HTML format (module **HTML**), as trees (module **Tree**) or as dependency graphs (module **Dep**).

Other modules should be added in the future, such as **SQL** module to read to and to write from a database used as a derivation tree-bank, a **Strip** module to remove features, or different filtering modules to extract subsets of a forest.

5 Servers

Exploiting some of these modules, but also other components developed in Java, we are installing several servers to access different kinds of information (parsers, grammars, forests) in uniform and distributed ways.

5.1 A server of parsers

We are exploring several ways to build efficient parsers for TAGs (Éric Villemonte de la Clergerie and Alonso Pardo, 1998; Alonso Pardo et al., 2000; Éric Villemonte de la Clergerie, 2001; Boullier, 2000; Barthélemy et al., 2001; Barthélemy et al., 2000), which leads us to maintain a growing set of parsers. Moreover, we wish to be able to compare the output produced by these parsers to check soundness, completeness and level of sharing in the derivation forests. To achieve these objectives, we provide a uniform setting by installing a simple *server of parsers*, written in Perl. Once connected to this server, one

[6] `http://www.cis.upenn.edu/~xtag/`

selects a parser and sends a sentence to parse; the server returns the shared derivation forest in raw, HTML, XML, Tree or Dep formats.

A WEB front-end[7] may be used to connect to this server. Figures 9 and 10 show two views of a derivation forest built using the server.

Another WEB front-end[8] allows the direct submission to the server of a derivation forest in one of the 3 recognized input formats (LP, RCG, XTAG). Submission in XML format should be added soon.

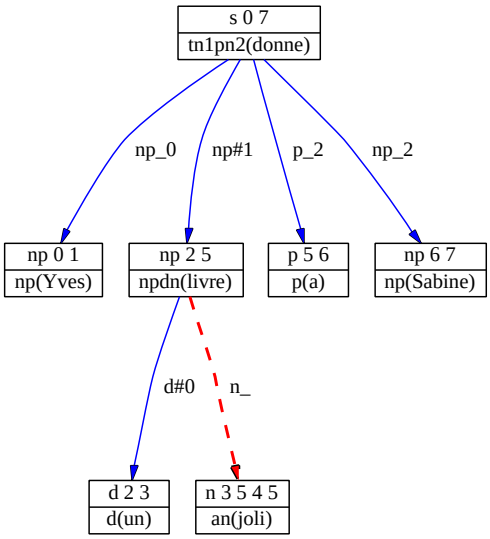

Figure 9: Derivation tree

5.2 A server of grammars

Because of the size of wide-coverage grammars, we believe that working with grammars stored in files is no longer a valid option (except for exchanging them). Databases should be used instead where bits of information can be efficiently retrieved. Moreover, modern database managers provide server fonctionalities and security mechanisms.

Around these ideas, we are currently developing in Java a server of grammars.

First the grammars are loaded into a SQL database (namely MySQL[9]). It should be noted that the structure of the database reflects the XML structure for grammars and not directly the structure of the grammars. This means that the loading phase may be performed for (almost) any kind of XML documents.

The second main component of the server is a small query language used to fetch information from the database while hiding, for non specialists, the complexity of languages SQL, XQL or XPath. We have chosen an object oriented notation which, once again, reflects the structure of the TAG DTD and which is also close to path equations familiar to computational linguists in, for instance, HPSG. We have several *types* such as `family`, `morph`, `tree` or `node` corresponding to the different kinds of elements of the DTD. For each type, several *methods* are available. For example, the following query returns the name and the Database Id (DBId) of all trees belonging to family `tn1pn2`. A second kind of requests takes a DBId n and returns the full XML fragment associated to the XML element whose index in the database is n.

```
var
  tree t;
select
  t.name; t;
where
  t.family.name' = 'tn1pn2
end
```

The grammar server works as a Java *servlet*, integrated in a HTTP server Apache using JServ[10]. It may be accessed using URL with parameters (as done for CGI scripts). The server decodes the parameters and transforms the query into a SQL query send to the database corresponding to the selected grammar. The result is either a table encoded in XML format or an XML fragment of the grammar. A small WEB interface[11] is available to display the results in a navigator (by transforming them into HTML) but it is also possible to get the results as an XML file. Tools can therefore query the server by sending a URL and getting back the results in XML.

The server should be soon completed for edition tasks. Full deletion of an element and of its descendants may be achieved using DBIds. Addition can be achieved by sending an XML fragment and a DBId (stating where to attach the XML fragment).

[7] `http://medoc.inria.fr/pub-cgi-bin/parser.cgi`
[8] `http://medoc.inria.fr/pub-cgi-bin/forest.cgi`
[9] `http://www.mysql.com/`
[10] `http://www.apache.org`
[11] `http://tequila.inria.fr/`

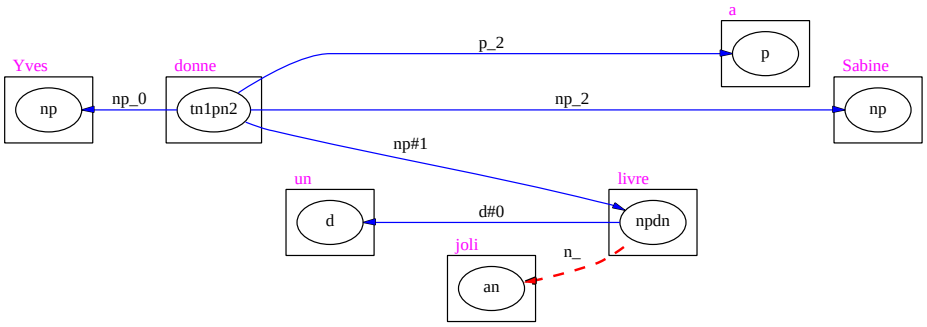

Figure 10: Dependency view

5.3 A server of derivation forests

The development of a server of derivation forests has just started, along the same lines we followed for the server of grammars. Such a server will be an alternative to treebanks. Two main functionalities are planned for this server. First the possibility to add a derivation forest to the underlying database (in order to build corpora of derivations) and second, ways to access these derivations through a simple but powerful query language.

6 Interfaces

We have already mentioned WEB interfaces to the parser server and the grammar server. Besides these interfaces, we (Kaouane, 2000) have also modified and enriched a Java interface developed by Patrice Lopez (Lopez, 2000). The new version can import grammars and derivation forests that follow our XML DTD. It can also use the server of parsers, sending the sentence to parse and receiving the shared derivation forest in XML format.

The derivations are extracted from the derivation forests and, for each derivation, the tool displays both the derivation tree and the corresponding parse tree (see Figure 11). It is also possible to follow step by step a derivation by moving forward or backward. We found this functionality useful to understand or explain to students the workouts of TAG.

The viewer may also be used to browse the different components of a grammar (trees and lexicons), therefore helping in its maintenance.

This tool already exploits the parser server, but we also plan to extend it to exploit the grammar server (for browsing the grammars or displaying a derived tree) and the forest server (for accessing derivation forests).

Conclusion

The experiments we have performed have strengthened our opinion that XML is really adequate to maintain and exchange linguistic resources and that XML allows us to quickly develop tools to handle these resources. Most of the components presented in this paper have been developed over a short period of time, and, while still preliminary, are fully usable. We believe that XML and these tools gives us solid foundations to further develop a complete environment to handle TAGs, based on many simple and easy-to-maintain tools (instead of having a monolithic system). We also think that the availability of such tools or resources may prove useful for linguists (to develop grammars with browsing and maintenance tools), for parser developers (to browse grammars and derivations), and for students in computational linguistics (to understand Tree Adjoining Grammars).

The tools and resources that we develop are freely available. Tools are based on a modular architecture with a specification given by the DTD and we hope that new components will be added by other people.

References

Miguel Alonso Pardo, Djamé Seddah, and Éric Villemonte de la Clergerie. 2000. Practical aspects in compiling tabular TAG parsers. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 27–32, Université Paris 7, Jussieu, Paris, France, May.

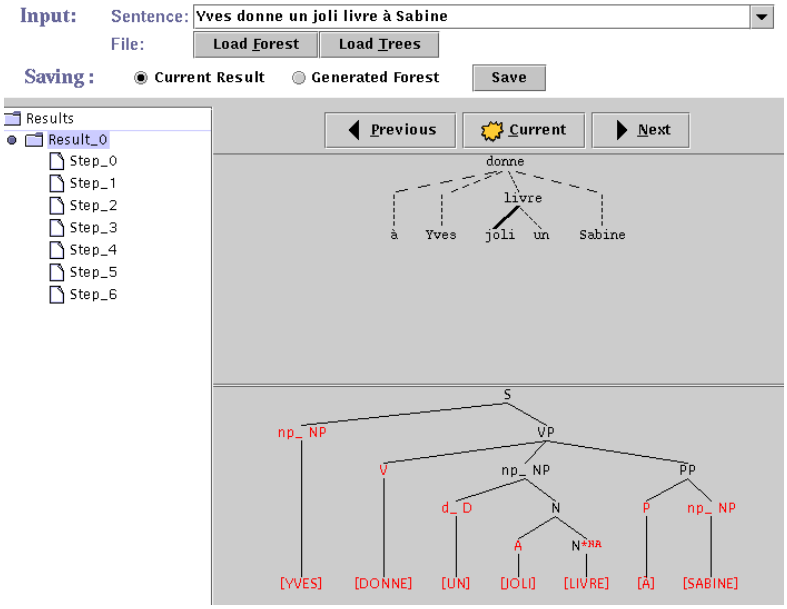

Figure 11: Screen capture of the Java derivation viewer

F. Barthélemy, P. Boullier, Ph. Deschamp, and É. Villemonte de la Clergerie. 2001. Guided parsing of range concatenation languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, University of Toulouse, France, July. to be published.

Patrick Bonhomme and Patrice Lopez. 2000. TagML: XML encoding of resources for lexicalized tree adjoining grammars. In *Proceedings of LREC 2000*, Athens.

Pierre Boullier. 2000. Range concatenation grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February. see also *Rapport de recherche n° 3342*, online at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA, France, January 1998, 41 pages.

Aravind K. Joshi. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia.

Linda Kaouane. 2000. Adaptation et utilisation d'un environnement graphique pour les TAG au dessus du système DyALog. Mémoire de DEA, Université d'Orléans.

Bernard Lang. 1991. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current issues in Parsing Technology*, chapter 11. Kluwer Academic Publishers. Also appeared in the Proceedings of International Workshop on Parsing Technologies – IWPT89.

Patrice Lopez. 2000. LTAG workbench: A general framework for LTAG. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, May.

Éric Villemonte de la Clergerie and Miguel Alonso Pardo. 1998. A tabular interpretation of a class of 2-stack automata. In *Proceedings of ACL/COLING'98*, August. online at `ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/SD2SA.ps.gz`.

Éric Villemonte de la Clergerie. 2001. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, June. to be published.

The XTAG Research Group. 1995. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA, USA, March.