XTAG – A Graphical Workbench for Developing Tree-Adjoining Grammars*

Patrick Paroubek, Yves Schabes and Aravind K. Joshi**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104-6389 USA
pap/schabes/joshi@linc.cis.upenn.edu

Abstract

We describe a workbench (*XTAG*) for the development of tree-adjoining grammars and their parsers, and discuss some issues that arise in the design of the graphical interface.

Contrary to string rewriting grammars generating trees, the elementary objects manipulated by a tree-adjoining grammar are extended trees (i.e. trees of depth one or more) which capture syntactic information of lexical items. The unique characteristics of tree-adjoining grammars, its elementary objects found in the lexicon (extended trees) and the derivational history of derived trees (also a tree), require a specially crafted interface in which the perspective has shifted from a string-based to a tree-based system. *XTAG* provides such a graphical interface in which the elementary objects are trees (or tree sets) and not symbols (or strings).

The kernel of *XTAG* is a predictive left to right parser for unification-based tree-adjoining grammar [Schabes, 1991]. *XTAG* includes a graphical editor for trees, a graphical tree printer, utilities for manipulating and displaying feature structures for unification-based tree-adjoining grammar, facilities for keeping track of the derivational history of TAG trees combined with adjoining and substitution, a parser for unification based tree-adjoining grammars, utilities for defining grammars and lexicons for tree-adjoining grammars, a morphological recognizer for English (75 000 stems deriving 280 000 inflected forms) and a tree-adjoining grammar for English that covers a large range of linguistic phenomena.

Considerations of portability, efficiency, homogeneity and ease of maintenance, lead us to the use of Common Lisp without its object language addition and to the use of the *X* Window interface to Common Lisp (CLX) for the implementation of *XTAG*.

XTAG without the large morphological and syntactic lexicons is public domain software. The large morphological and syntactic lexicons can be obtained through an agreement with ACL's Data Collection Initiative.

XTAG runs under Common Lisp and *X* Window (CLX).

*This work was partially supported by NSF grants DCR-84-10413, ARO Grant DAAL03-87-0031, and DARPA Grant N0014-85-K0018.

**Visiting from the Laboratoire Informatique Théorique et Programmation, Institut Blaise Pascal, 4 place Jussieu, 75252 PARIS Cedex 05, France.

1 Introduction

Tree-adjoining grammar (TAG) [Joshi *et al.*, 1975; Joshi, 1985; Joshi, 1987] and its lexicalized variant [Schabes *et al.*, 1988; Schabes, 1990; Joshi and Schabes, 1991] are tree-rewriting systems in which the syntactic properties of words are encoded as tree structured-objects of extended size. TAG trees can be combined with adjoining and substitution to form new derived trees.[1]

Tree-adjoining grammar differs from more traditional tree-generating systems such as context-free grammar in two ways:

1. The objects combined in a tree-adjoining grammar (by adjoining and substitution) are trees and not strings. In this approach, the lexicon associates with a word the entire structure it selects (as shown in Figure 1) and not just a (non-terminal) symbol as in context-free grammars.
2. Unlike string-based systems such as context-free grammars, two objects are built when trees are combined: the resulting tree (the *derived tree*) and its derivational history (the *derivation tree*).[2]

These two unique characteristics of tree-adjoining grammars, the elementary objects found in the lexicon (extended trees) and the distinction between derived tree and its derivational history (also a tree), require a specially crafted interface in which the perspective must be shifted from a string-based to a tree-based system.

[1] We assume familiarity throughout the paper with the definition of TAGs. See the introduction by Joshi [1987] for an introduction to tree-adjoining grammar. We refer the reader to Joshi [1985], Joshi [1987], Kroch and Joshi [1985], Abeillé *et al.* [1990a], Abeillé [1988] and to Joshi and Schabes [1991] for more information on the linguistic characteristics of TAG such as its lexicalization and factoring recursion out of dependencies.

[2] The TAG derivation tree is the basis for semantic interpretation [Shieber and Schabes, 1990b], generation [Shieber and Schabes, 1991] and machine translation [Abeillé *et al.*, 1990b] since the information given in this data-structure is richer than the one found in the derived tree. Furthermore, it is at the level of the derivation tree that ambiguity must be defined.

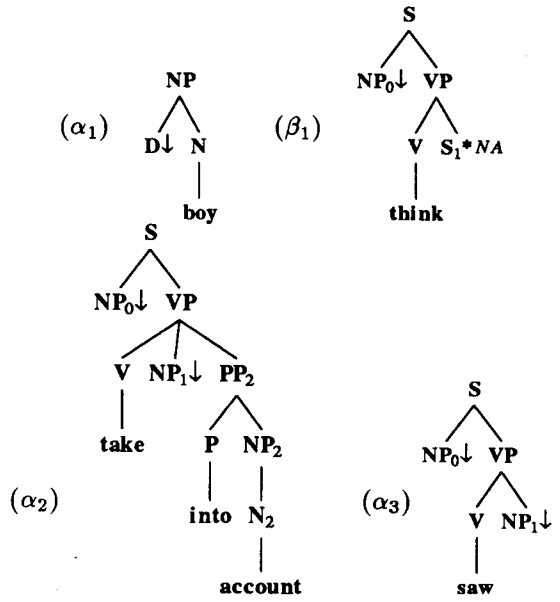

Figure 1: Elementary trees found in a tree-adjoining grammar lexicon

XTAG provides such a graphical interface in which the elementary objects are trees (or tree sets) and not symbols (or strings of symbols).

Skeletons of such workbenches have been previously realized on Symbolics machines [Schabes, 1989; Schifferer, 1988]. Although they provided some insights on the architectural design of a TAG workbench, they were never expanded to a full fledged natural language environment because of inherent limitations (such as their lack of portability).

XTAG runs under Common Lisp [Steele, 1990] and it uses the Common LISP X Interface (CLX) to access the graphical primitives defined by the *X11* protocol. *XTAG* is portable across machines and Common Lisp compilers.

The kernel of *XTAG* is a predictive left to right parser for unification-based tree-adjoining grammar [Schabes, 1991]. The system includes the following components and features:

- Graphical edition of trees. The graphical display of a tree is the only representation of a tree accessible to the user. Some of the operations that can be performed graphically on trees are:
  - Add and edit nodes.
  - Copy, paste, move or delete subtrees.
  - Combine two trees with adjunction or substitution. These operations keep track of the derivational history and update attributes stated in form of feature structures as defined in the framework of unification-based tree-adjoining grammar [Vijay-Shanker and Joshi, 1988].
  - View the derivational history of a derived tree and its components (elementary trees).
- A tree display module for efficient and aesthetic formatting of a tree based on a new tree display algorithm [Chalnick, 1989]. The algorithm is an improvement of the ones developed by Reingold and Tolford [1981] and, Lee [1987]. It guarantees in linear time that tress which are structural mirror images of on another are drawn such that their displays are reflections of one another while achieving minimum width of the tree.
- Capabilities for grouping trees into sets which can be linked to a file. This is particularly useful since lexicalized TAGs organize trees into tree-families which capture all variations of a predicative lexical item for a given subcategorization frame.
- Utilities for editing and processing equations for unification based tree-adjoining grammar [Vijay-Shanker and Joshi, 1988; Schabes, 1990].
- A predictive left to right parser for unification-based tree-adjoining grammar [Schabes, 1991].
- Utilities for defining a grammar (set of trees, set of tree families, set of lexicons) which the parser uses.
- Morphological lexicons for English [Karp *et al.*, 1992]
- A tree-adjoining grammar for English that covers a large range of linguistic phenomena.

2 *XTAG* Components

The communication with the user is centralized around the interface manager window (See Figure 2) which gives the user control over the different modules of *XTAG*.

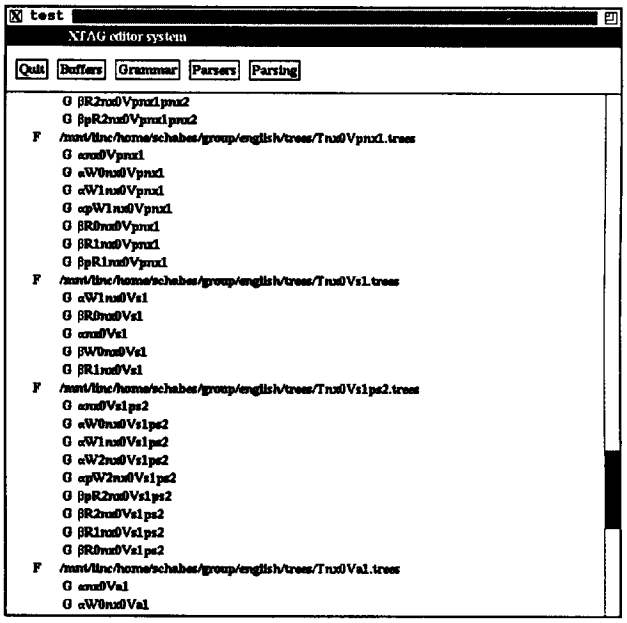

Figure 2: Manager Window.

This window displays the contents of the tree buffers currently loaded into the system. The different functions of *XTAG* are available by means of a series of pop-up menus associated to buttons, and by means of mouse actions performed on the mouse-sensitive items (such as the tree buffer names and the tree names).

A tree editor for a tree contained in one of the tree buffer contained in the window can be called up by clicking over its tree name. Each tree editor manages one tree and as many tree editors as needed can run concurrently.

For example, Figure 2 holds a set of files (such as Tnx0Vs1.trees)[3] which each contain trees (such as αnx0Vs1). When this tree is selected for editing, the window shown in Figure 3 is displayed. Files can be handled independently or in group, in which case they form a tree family (flag F next to a buffer name).

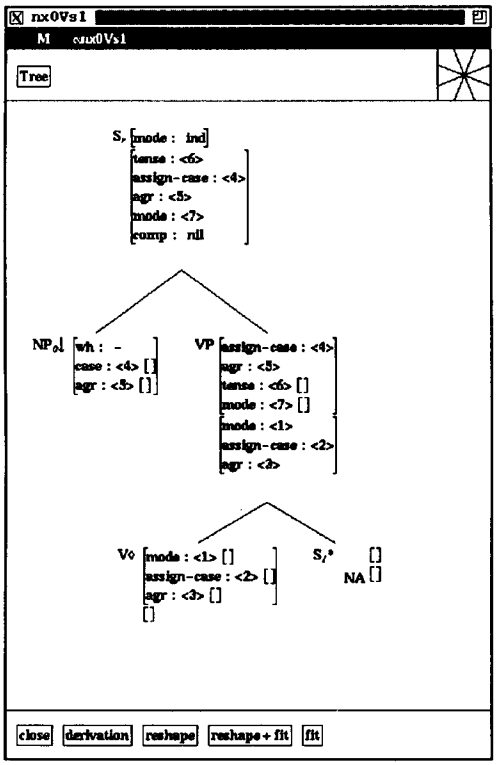

Figure 3: A tree-editing window for the tree αnx0Vs1.

All the editing and visualizing operations are performed through this window (see Figure 3). Some of them are:

- Add and edit nodes.
- Copy, paste, move or delete subtrees.
- Combine two trees with adjunction or substitution. These operations keep track of the derivational history and update attributes stated in form of feature structures as defined in the framework of unification-based tree-adjoining grammar [Vijay-Shanker and Joshi, 1988].
- View the derivational history of a derived tree and its components (elementary trees).
- Display and edit feature structures.
- Postscript printing of the tree.

[3]The particular conventions for the tree and family names reflect the structure of the trees and they can be ignored by the reader.

XTAG uses a centralized clipboard for all binary operations on trees (all operations are either unary or binary). These operations (such as paste, adjoin or substitute) are always performed between the tree contained in *XTAG*'s clipboard and the current tree. The contents of the clipboard can be displayed in a special view-only window.

The request to view the derivational history of a tree result of a combining operation triggers the opening of a view-only window which displays the associated derivation tree. Each node in a derivation tree is mouse-sensitively linked to an elementary tree.

Since the derivational history of a derived tree depends on the elementary trees which were used to build it, inconsistency in the information displayed to the user could arise if the user attempts to modify an elementary tree which is being used in a derivation. This problem is solved by ensuring that, whenever a modifying operation is requested, full consistency is maintained between all the views. For instance, editing a tree used in a derivation tree will break the link between those two. Thus consistency is maintained between the derived tree and the derivation tree.

Figure 4 shows an example of a derived tree (leftmost window) with its derivation tree window (middle window) and an elementary tree participating in its derivation (rightmost window).

As is shown in Figure 3, the tree display module handles the bracketed display of feature structures (in unification-based TAG, each node is associated two feature structures: top and bottom, see Vijay-Shanker and Joshi [1988] for more details). The tree formatting algorithm guarantees that trees that are structural mirror images of on another are drawn such that their displays are reflections of one another [Chalnick, 1989]. A unification module handles the updating of feature structures for TAG trees.

XTAG includes a predictive left to right parser for unification-based tree-adjoining grammar [Schabes, 1991]. The parser is integrated into *XTAG* and derivations are displayed by the interface as illustrated in Figure 4. The parser achieves an $O(G^2n^6)$-time worst case behavior, $O(G^2n^4)$-time for unambiguous grammars and linear time for a large class of grammars. The parser uses the following two-pass parsing strategy (originally defined for lexicalized grammars [Schabes *et al.*, 1988]) which improves its performance in practice [Schabes and Joshi, 1990]:

- In the first step the parser will select, the set of structures corresponding to each word in the sentence. Each structure can be considered as encoding a set of 'rules'.
- In the second step, the parser tries to see whether these structures can be combined to obtain a well-formed structure. In particular, it puts the structures corresponding to arguments into the structures corresponding to predicates, and adjoins, if needed, the auxiliary structures corresponding to adjuncts to what they select (or are selected) for. This step is performed with the help of a chart in the fashion of Earley-style parsing.

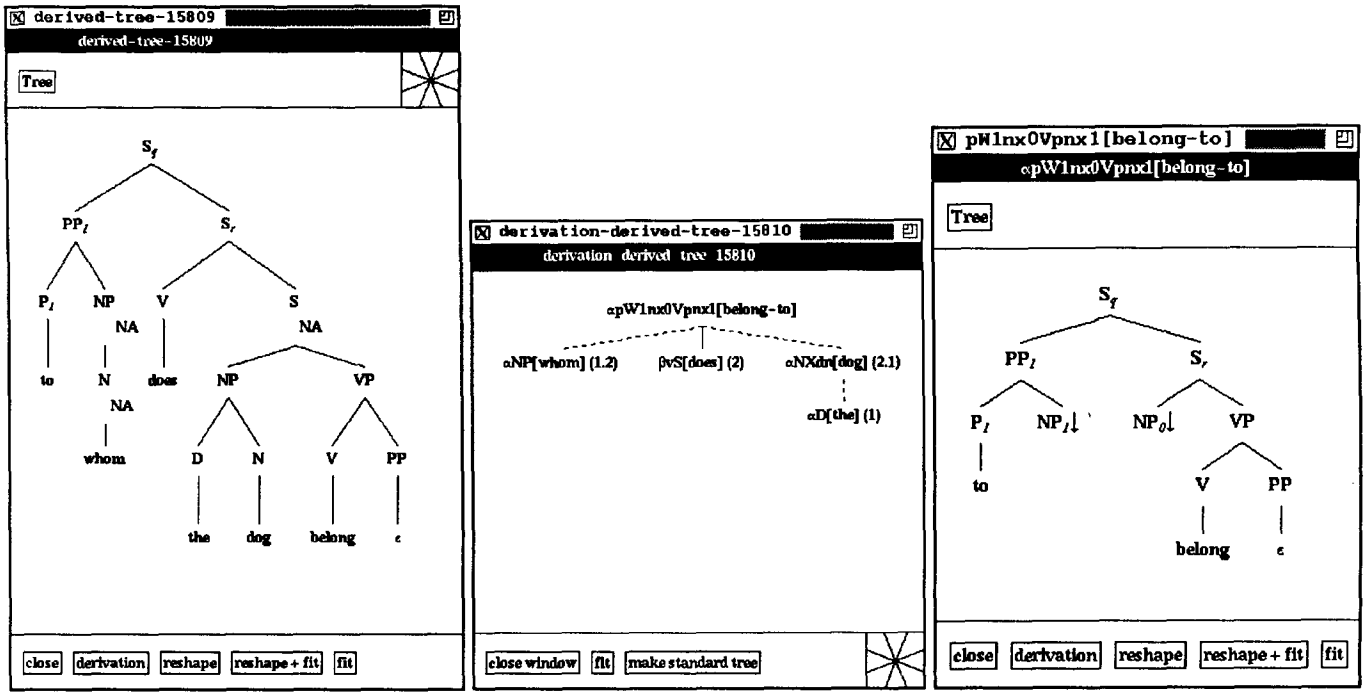

Figure 4: *left*, a derived tree, *middle*, its derivation, *right*, an elementary tree participating in the derivation.

The first step enables the parser to select a relevant subset of the entire grammar, since only the structures associated with the words in the input string are selected for the parser. The number of structures filtered during this pass depends on the nature of the input string and on characteristics of the grammar such as the number of structures, the number of lexical entries, the degree of lexical ambiguity, and the languages it defines. In the second step, since the structures selected during the first step encode the morphological value of their words (and therefore their position in the input string), the parser is able to use non-local bottom-up information to guide its search. The encoding of the value of the anchor of each structure constrains the way the structures can be combined. This information is particularly useful for a top-down component of the parser [Schabes and Joshi, 1990].

XTAG provides all the utilities required for designing a lexicalized TAG structured as in Schabes *et al.* [1988]. All the syntactic concepts of lexicalized TAG (such as the grouping of the trees in tree families which represents the possible variants on a basic subcategorization frame) are accessible through mouse-sensitive items. Also, all the operations required to build a grammar (such as load trees, define tree families, load syntactic and morphological lexicon) can be predefined with a macro-like language whose instructions can be loaded from a file (See Figure 5).

The grammar writer has also the option to manually test a derivation by simulating adjoining or substitution of trees that are associated with words defined in the lexicon.

The grammar consists of a morphological English analyzer and a syntactic lexicon, which is the domain of structural choice, subcategorization and selectional information. Lexical items are defined by the tree structure or the set of tree structures they select.

```
(defgrammar demo1
  (:start-symbol "S"
   :start-feature "<mode> = ind")
  (:tree-files   "lex"  "modifiers"
    (:type "trees"))
  (:family-files
    "Tnx0V" "Tnx0Va" "Tnx0Vnx1" "Tnx0Vdn1"
    "Tnx0Vnx1pnx2" "Tnx0Vpnx1" "Tnx0Vs1"
    (:type "trees"))
  (:lexicon-files  "lexicon" (:type "lex"))
  (:example-files "examples" (:type "ex")))
```

Figure 5: An example of instructions for loading and defining a grammar.

The morphological lexicons for English [Karp *et al.*, 1992] were built with PC-KIMMO's implementation of two-level morphology [Antworth, 1990] and with the 1979 edition of Collins English Dictionary. They comprise 75 000 stems deriving 280 000 inflected forms.

XTAG also comes with a tree-adjoining grammar for English [Abeillé *et al.*, 1990a] which covers a large range of linguistic phenomena.

The entries for lexical items of all types belong to the syntactic lexicon and are marked with features to constrain the form of their arguments. For example, a verb which takes a sentential argument uses features to constrain the form of the verb acceptable in the complement clause. An interesting consequence of TAG's extended

domain of locality is that features imposed by a clausal lexical item can be stated directly on the subject node as well as on the object node. These features need not be percolated through the VP node as in context-free formalisms.

When a word can have several structures, corresponding to different meanings, it is treated as several lexical items with different entries in the syntactic lexicon. Morphologically, such items can have the same category and the same entry in the morphological lexicon[4]. Examples of syntactic entries follow:[5]

```
INDEX:  cut
ENTRY:  NP0 cut into NP1
POS:    NP0 V P1 NP1
FS:     #inv-, #pass-
DEF:    make an incision in.

INDEX:  cut
ENTRY:  NP0 cut down (NP1)
POS:    NP0 V PL (NP1)
FS:     #pass+
DEF:    consume less; reduce.
EX:     "The city must cut its expenses down.'

INDEX:  accuse
ENTRY:  NP0 accuse NP1 (of NP2)
POS:    NP0 V NP1 (P2 NP2)
FS:     #inv1-, #dat-, #inv2-, #pass1+, #pass2-
DEF:    say that somebody is guilty (of).

INDEX:  face
ENTRY:  NP0 face away (from NP1)
POS:    NP0 V PL (P1 NP1)
FS:     #inv-, #pass-
DEF:    look in the opposite direction (from).
EX:     My house faces away from the ocean.
```

3 The choice of a graphical package: *X* Window and CLX

The choice of a graphical package was motivated by considerations of portability, efficiency, homogeneity and ease of maintenance. *XTAG* was built using Common Lisp and its *X* Window interface CLX.

We chose this rather low level approach to realize the interface as opposed to the use of a higher-level toolkit for graphic interface design because the rare tools available which were fulfilling our requirements for portability, homogeneity and ease of maintenance were still under development at the beginning of the design of *XTAG*.

The first package we considered was Express Window. It attracted our attention because it has precisely been created to run programs developed on the Symbolics machine in other Common Lisp environments. It is an implementation of most of the Flavors and graphic primitives of the Symbolics system respectively in terms of the Common Lisp Object System (CLOS) [Keene, 1988] primitives and CLX [Scheifler and Lamott, 1989]. We did not use it mainly because it was known to work only with the PCL version from Xerox Parc (we want to remain as compatible as possible between the different dialects of Common Lisp), and was not robust enough.

Although WINTERP has many interesting points for our purpose, we did not choose it because we wanted to have a complete and efficient (i.e. a compiler) Common Lisp implementation. WINTERP is an interpretive, interactive environment for rapid prototyping and application writing using the OSF Motif toolkit [Young, 1990]. It uses a mini-lisp interpreter (called XLISP; it is not available as a compiler) to glue together various C-implemented primitive operations (Xlib [Nye, 1988] and Xtk [Asente and Swick, 1990]) in a Smalltalk-like [Goldberg, 1983] object system (widgets are a first class type). WINTERP has no copyright restrictions.

Initially we were attracted by GARNET [Meyers *et al.*, 1990], mainly because it is advertised as look-and-feel independent and because it is implemented using only Common Lisp and CLX (but not CLOS, nor any existing X toolkit such as Xtk or Motif). The system is composed of two parts: (1) a toolkit offering objects (prototype instance model [Lieberman, 1986], constraints, (2) and an automatic run-time maintenance of properties of the graphic objects based on a semantic network. The different behavior of the interface components is specified by binding high level interactors objects to the graphic objects. An interface builder tool (Lapidary) allows the drawing of the graphic aspects of an interface. However we did not use GARNET because the version at the time of the design of *XTAG* was very large and slow, and still subject to changes of design. Furthermore, another reason for not choosing GARNET was the fact that Carnegie Mellon University retained the copyrights, slowing the distribution.

PICASSO [Schank *et al.*, 1990], a more recent package from Berkeley University, offers similar functionalities shared by other non Common Lisp based application frameworks like InterViews [Linton *et al.*, 1989], MacApp [Schmuker, 1986] and Smalltalk [Goldberg, 1983], but is freely distributed. It is an object-oriented system implemented in Common Lisp, CLX and CLOS. With each type of PICASSO object is associated a CLOS class, the instances of which have different graphic and interactive properties. The widgets implementing those properties are automatically created during the creation of a PICASSO object. Unlike the two previous systems, the PICASSO objects may be shared in an external database common to different applications (persistent classes) when this is enabled, PICASSO requires the use of the database management system INGRES [Charness and Rowe, 1989]. PICASSO was not available as a released package at the time the implementation of *XTAG* started.

4 Programming considerations

All the graphic objects of XTAG are defined as contacts and are implemented using only the structures of Common Lisp and their simple inheritance mechanism. Because of the relatively low computing cost associated with the contacts, we have been able to define every

[4]The lexical entries below have been simplified for the purpose of exposition.

[5]Syntactic lexicons can be stated in various forms. In the following examples, a tree family is associated with each string of part of speeches (POS).

graphic object of XTAG (whatever its complexity as a contact is) without having to resort to a different procedure oriented implementation for simpler objects as was done in InterViews with the painter objects [Linton *et al.*, 1989].

The programming difficulties we have encountered deal with re-synchronizing XTAG with the server during a transfer of control between contacts (the active contact is the one containing the cursor). These difficulties stem from the asynchronous nature of the communication protocol of X and from the large number of events mouse motion may generate when the cursor is moved over closely located windows. The fact that windows may be positioned anywhere and stacked in any order (overlapping windows) makes the handling of those transitions a non trivial task. A careful choice of the event-masks attached to the windows is by itself insufficient to solve the problem. To limit the number of queries made to the server, we use extensive caching of graphic properties. The structures implementing the contacts contain fields that duplicate server information. They are updated when the graphic properties of the object they describe are changed. We found this strategy to improve the performance noticeably. This feature can easily be turned off, in case a particular X-terminal or workstation would provide hardware support for caching.

While we put a lot of attention on issue of portability, we did not worry about look independence, limiting the user possibilities in this domain to geometric dimension parameterization and font selection by means of a configuration file and a few menus.

Our current implementation uses the twm window manager, but another window manager could also be used. We have found the need for multi-font string support for *XTAG* because the tree names and node labels require a mix of at least two or three fonts (ascii symbols and greek symbols such as α, β and ϵ, and a font for subscripts). We could have used a font which contains all the characters may use the same font as the normal differ from those only by their location to the writing line), but we preferred to define a multi-font composed of several existing fonts (which can be customized by the user) for portability purposes and to leave open the way for future extensions.

In order to be able to scroll over the trees when they are too big to be displayed in a window, every tree editor window is associated with an eight direction touch-pad (inspired from the mover of InterViews [Linton *et al.*, 1989]).

The nodes displayed in the window of a tree editor are not sensitive to the presence of the cursor, they react only to mouse button clicks. During earlier versions of *XTAG* we highlighted the visited node with a border, but this required too much overhead because of the numerous cursor motions over the tree window which occur during editing.

The text editing task we had to implement fall into two classes:

- short line editing requiring multi-fonts (e.g. edition of node names);
- text editing not requiring multi-fonts (e.g. multi-line comments, unification equations).

For the former, we implemented all the editing functions ourselves because they do not require much processing and multi-font support was unavailable. For the latter, we used system calls to an external editor (emacs in our case).

Concerning the programming task, we would have liked to have available tools to help us write an X application in Common Lisp at a level slightly higher than the one of the CLX interface without going up to the level of elaborate toolkits like GARNET or PICASSO which implies the use of a complex infra-structure, perhaps something like an incremental or graded toolkit.

Our next developments effort will be concerned with introducing parallelism in the interface (actors), adding new features like an undo mechanism (using the Item list data structure proposed by Dannenberg [1990]), and extending XTAG for handling meta-rules [Becker, 1990] and Synchronous TAGs [Shieber and Schabes, 1990b] which are used for the purpose of automatic translation [Abeillé *et al.*, 1990b] and semantic interpretation [Shieber and Schabes, 1990a; Shieber and Schabes, 1991].

5 Requirements for Running *XTAG*

Version 0.93 of *XTAG* is available as *pub/xtag0.93.tar.Z* by anonymous ftp to *linc.cis.upenn.edu* (130.91.6.8).[6] *XTAG* requires:

- A machine running UNIX and X11R4.[7]
- A Common Lisp compiler which supports the latest definition of Common Lisp [Steele, 1990]. Although *XTAG* should run under a variety of lisp compilers, it has only been tested with LUCID Common Lisp 4.0 and Allegro Common Lisp 4.0.1. A version running under KCL is currently under development.
- Common Lisp X Interface (CLX) version 4 or higher.[8]

XTAG has been tested on UNIX based machines (R4.4) (SPARC station 1, SPARC station SLC, HP BOBCATs series 9000 and SUN 4 also with NCD X-terminals) running X11R4 and CLX with Lucid Common Lisp (4.0) and Allegro Common Lisp (4.0.1).

6 Conclusion

We described a workbench for the development of tree-adjoining grammars and their parsers and discussed some issues that arise in the design of the graphic interface.

The unique characteristics of tree-adjoining grammars, its elementary objects found in the lexicon (extended trees) and the derivational history of derived

[6] Newer versions of the system are copied into the same directory as they become available.

[7] Previous releases of X will not work with *XTAG*. X11R4 is free software available from MIT.

[8] CLX is the Common Lisp equivalent to the Xlib package for C. It allows the lisp programmer to use the graphical primitives of Xlib within Common Lisp.

trees (also a tree), require a specially crafted interface in which the perspective is shifted from a string-based to a tree-based system. *XTAG* provides such a workbench.

The kernel of *XTAG* is a predictive left to right parser for unification-based tree-adjoining grammar [Schabes, 1991]. *XTAG* includes a graphical editor for trees, a graphical tree printer based on a new algorithm, utilities for manipulating and displaying feature structures for unification-based tree-adjoining grammar, facilities for keeping track of the derivational history of TAG trees combined with adjoining and substitution, a parser for unification based tree-adjoining grammars, utilities for defining grammars and lexicons for tree-adjoining grammars, a morphological recognizer for English (75 000 stems deriving 280 000 inflected forms) and a tree-adjoining grammar for English that covers a large range of linguistic phenomena.

XTAG without the large morphological and syntactic lexicons is public domain software. The large morphological and syntactic lexicons can be obtained through an agreement with ACL's Data Collection Initiative.

XTAG runs under Common Lisp and *X* Window (CLX).

Acknowledgments

Many people contributed to the development of *XTAG*.

We are especially grateful to Mark Liberman for his help with the extraction of morphological and syntactic information available in the data collected by the ACL's Data Collection Initiative.

We have benefitted from discussions with Evan Antworth, Lauri Karttunen, Anthony Kroch, Fernando Pereira, Stuart Shieber and Annie Zaenen.

Anne Abeille, Kathleen Bishop, Sharon Cote, Beatrice Daille, Jason Frank, Caroline Heycock, Beth Ann Hockey, Megan Moser, Sabine Petillon and Raffaella Zanuttini contributed to the design of the English and French TAG grammars.

The morphological lexicons for English were built by Daniel Karp.

Patrick Martin automated the acquisition of some of syntactic lexicons with the use of on-line dictionaries.

We would like to thank Jeff Aaronson, Tilman Becker, Mark Foster, Bob Frank, David Magerman, Philip Resnik, Steven Shapiro, Martin Zaidel and Ira Winston for their help and suggestions.

References

[Abeillé *et al.*, 1990a] Anne Abeillé, Kathleen M. Bishop, Sharon Cote, and Yves Schabes. A lexicalized tree adjoining grammar for English. Technical Report MS-CIS-90-24, Department of Computer and Information Science, University of Pennsylvania, 1990.

[Abeillé *et al.*, 1990b] Anne Abeillé, Yves Schabes, and Aravind K. Joshi. Using lexicalized tree adjoining grammars for machine translation. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics (COLING'90)*, Helsinki, 1990.

[Abeillé, 1988] Anne Abeillé. A lexicalized tree adjoining grammar for French: the general framework. Technical Report MS-CIS-88-64, University of Pennsylvania, 1988.

[Antworth, 1990] Evan L. Antworth. *PC-KIMMO: a two-level processor for morphological analysis.* Summer Institute of Linguistics, 1990.

[Asente and Swick, 1990] P. J. Asente and R. R. Swick. *X Window System Toolkit.* Digital Press, 1990.

[Becker, 1990] T. Becker. Meta-rules on tree adjoining grammars. In *Proceedings of the 1st International Workshop on Tree Adjoining Grammars*, Dagstuhl Castle, FRG, August 1990.

[Chalnick, 1989] Andrew Chalnick. Mirror image display of n-ary trees. Unpublished manuscript, University of Pennsylvania, 1989.

[Charness and Rowe, 1989] D. Charness and L. Rowe. CLING/SQL – Common Lisp to INGRES/SQL interface. Technical report, U.C. Berkeley, Computer Science Division, 1989.

[Dannenberg, 1990] R. B. Dannenberg. A structure for efficient update, incremental redislay and undo in graphical editors. *Graphical Editors, Software Practice and Experience*, 20(2), February 1990.

[Goldberg, 1983] A. Goldberg. *Smalltalk-80: The Interactive Programming Environment.* Addison Wesley, 1983.

[Joshi and Schabes, 1991] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees.* Elsevier, 1991. Forthcoming.

[Joshi *et al.*, 1975] Aravind K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1), 1975.

[Joshi, 1985] Aravind K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions—Tree Adjoining Grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing—Theoretical, Computational and Psychological Perspectives.* Cambridge University Press, New York, 1985.

[Joshi, 1987] Aravind K. Joshi. An Introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language.* John Benjamins, Amsterdam, 1987.

[Karp *et al.*, 1992] Daniel Karp, Yves Schabes, and Martin Zaidel. Wide coverage morphological lexicons for English. Submitted to the 14^{th} International Conference on Computational Linguistics (COLING'92), 1992.

[Keene, 1988] S. Keene. *Object-Oriented Programming in Common Lisp.* Addison-Wesley, 1988.

[Kroch and Joshi, 1985] Anthony Kroch and Aravind K. Joshi. Linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-18, Department

of Computer and Information Science, University of Pennsylvania, April 1985.

[Lee, 1987] Albert Lee. Performance oriented tree-display package. University of Pennsylvania Senior Thesis, 1987.

[Lieberman, 1986] H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In *ACM Conference on Object-Oriented Programming Systems Languages and Applications*, 1986.

[Linton *et al.*, 1989] M. A. Linton, J. M. Vlissides, and P. R. Calder. Composing user interfaces with interviews. *IEEE Computer*, February 1989.

[Meyers *et al.*, 1990] B. A. Meyers, D. Giuse, B. Dannenberg, and V. B. Zanden. The garnet toolkit reference manuals: Support for highly-interactive, graphical user interfaces in lisp. Technical Report CMU-CS-90-117, Carnegie Mellon University, 1990.

[Nye, 1988] A. Nye. *Xlib Progamming Manual (Vol. 1), Xlib Reference Manual (Vol. 2).* O'Reilly and Associates, 1988.

[Reingold and Tolford, 1981] E.M. Reingold and John S. Tolford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, SE-7:223–228, 1981.

[Schabes and Joshi, 1990] Yves Schabes and Aravind K. Joshi. Parsing with lexicalized tree adjoining grammar. In Masaru Tomita, editor, *Current Issues in Parsing Technologies*. Kluwer Accademic Publishers, 1990.

[Schabes *et al.*, 1988] Yves Schabes, Anne Abeillé, and Aravind K. Joshi. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, August 1988.

[Schabes, 1989] Yves Schabes. TAG system user manual for Symbolics machines, 1989.

[Schabes, 1990] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA, August 1990. Available as technical report (MS-CIS-90-48, LINC LAB179) from the Department of Computer Science.

[Schabes, 1991] Yves Schabes. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proceedings of the second International Workshop on Parsing Technologies*, Cancun, Mexico, February 1991.

[Schank *et al.*, 1990] P. Schank, J. Konstan, C. Liu, A. R. , S. Seitz, and B. Smith. Picasso reference manual. Technical report, University of California, Berkeley, 1990.

[Scheifler and Lamott, 1989] W. Scheifler, R. and O. Lamott. Clx - common lisp x interface. Technical report, Texas Instruments, 1989.

[Schifferer, 1988] Klaus Schifferer. TAGDevENV eine Werkbank für TAGs. Technical report, KI – Labor am Lehrstuhl für Informatik, Universitä des Saarlandes, June 1988.

[Schmuker, 1986] K. J. Schmuker. Mac-app: An application framework. *Byte*, August 1986.

[Shieber and Schabes, 1990a] Stuart Shieber and Yves Schabes. Generation and synchronous tree adjoining grammars. In *Proceedings of the fifth International Workshop on Natural Language Generation*, Pittsburgh, 1990.

[Shieber and Schabes, 1990b] Stuart Shieber and Yves Schabes. Synchronous tree adjoining grammars. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics (COLING'90)*, Helsinki, 1990.

[Shieber and Schabes, 1991] Stuart Shieber and Yves Schabes. Generation and synchronous tree adjoining grammars. *Computational Intelligence*, 4(7), 1991. forthcoming.

[Steele, 1990] Guy L. Jr. Steele, editor. *Common LISP–the Language.* Digital Press, second edition, 1990.

[Vijay-Shanker and Joshi, 1988] K. Vijay-Shanker and Aravind K. Joshi. Feature structure based tree adjoining grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, August 1988.

[Young, 1990] D. A. Young. *OSF/MOTIF Reference Guide.* Prentice, 1990.