

**EXLEPSE: AN ECLIPSE-BASED, EASY-TO-USE
EDITOR FOR COMPUTATIONAL LFG GRAMMARS**

Roman Rädle, Michael Zöllner and Sebastian Sulger
Universität Konstanz

Proceedings of the LFG11 Conference

Miriam Butt and Tracy Holloway King (Editors)

2011

CSLI Publications

<http://csli-publications.stanford.edu/>

Abstract

We present eXLEpse, an easy-to-use editor for creating computational grammars based on the Lexical-Functional Grammar (LFG) formalism (Dalrymple, 2001). The editor is implemented as a plugin for the open-source program development platform Eclipse. eXLEpse provides functionality for editing computational LFG grammars and an interface to the XLE grammar development platform (Crouch et al., 2011). The editor can replace Emacs as an editor and provides a graphically enriched user interface as an alternative to the shell-based interface of the XLE platform. It is available free of charge on the internet.

1 Introduction and Motivation

The Eclipse plugin eXLEpse¹ provides functionality for editing computational LFG grammars and an interface to the XLE grammar development platform. The primary goal of eXLEpse was to develop an easy-to-use editor for computational grammars with a simple yet powerful interface to XLE. The editor can replace Emacs² as an editor and provides an alternative to the shell-based interaction with the XLE platform.

For novices in XLE grammar development, it can be quite hard to get used to Emacs and the XLE command prompt. Also, the grammar syntax required by XLE can sometimes be confusing (e.g., nested templates). eXLEpse addresses these problems, providing a graphically enriched user interface via the Eclipse platform³. eXLEpse connects to the XLE binaries and the X11 windows platform to parse text and display parse results. Furthermore, various error support functions and advanced syntax highlighting enable novice users to concentrate solely on the grammar development process without painfully learning the details of Emacs and its concepts. Additionally, Eclipse offers support for a diversity of version control systems (VCS) such as the version management software Subversion via the Subclipse⁴ plugin. Developers can make use of all the Subversion features through this plugin, without having to leave eXLEpse. This is useful especially for large grammar projects.

We emphasize that eXLEpse and its concepts constitute work in progress. In particular, we do not, as of yet, view eXLEpse as a Swiss army knife of grammar development, but rather as an evolutionary development based on previously existing XLE editors towards a homogenous LFG design environment. The eXLEpse plugin is available as public and free software under the terms of the Eclipse Public License (EPL). The plugin is distributed together with Eclipse and the Subclipse

[†]We thank the LFG11 audience and the members of the ParGram community for their constructive criticism and feedback.

¹<http://www.exlepse.org>

²<http://www.gnu.org/s/emacs/>

³<http://www.eclipse.org>

⁴<http://subclipse.tigris.org/>

plugin in a single zipped package for easy installation; packages for the most common operating systems are available.

In Section 2, we review some of the issues that arise in current approaches to LFG grammar development, introducing the features of the eXLEpse editor step by step in subsections. The paper concludes with a brief summary and perspectives for future work.

2 eXLEpse – an XLE Perspective for Eclipse

The motivation for the development of the eXLEpse editor arose from registering usability problems in user interfaces currently employed in LFG grammar development with XLE. This section describes some of these problems; novices in grammar development taking XLE courses at the Universität Konstanz have reported that part of their difficulties with XLE have to do with the usability obstacles reported here. The eXLEpse project therefore focuses on problems experienced by XLE and Emacs novices, looking for accessible techniques to start learning about LFG grammar development.

LFG grammar development with XLE usually takes place using Emacs in combination with the Unix command shell. The Emacs plugin `lfg-mode.el`, written by Mary Dalrymple, is distributed together with XLE for easier LFG grammar development.⁵ It loads commands that cause Emacs to activate syntax highlighting, indentation, and other visual aids that help in LFG grammar development. Moreover, `lfg-mode.el` provides Emacs with special commands that are loaded into the Emacs menus. The commands are used to load (1) load files that are relevant to the grammar being opened, (2) start and restart XLE shells, and (3) let you browse rules, templates and lexical entries.

The views and tools of eXLEpse are described in the following sections and compared to the standard grammar development process using Emacs in combination with `lfg-mode.el`.

2.1 Managing Windows and Editors

In the non-eXLEpse architecture, the actual XLE process used for parsing and generating sentences has to be opened in a separate Emacs buffer. Therefore, the user's desktop environment during grammar development may look as in Figure 1, with different grammar files opened in multiple Emacs buffers and a separate buffer for the XLE process. For professional grammar developers or computer scientists, this may not be a big issue, but for novices looking to learn about grammar development in an easy way, this can be very cumbersome.

Note that the user may open several files in a single Emacs window pane. The user can then cycle through the files using special Emacs-specific shortcuts. Note

⁵For the remainder of the paper, we use the term Emacs, referring to instances of Emacs running with the `lfg-mode.el` plugin.

further that a single Emacs pane may be split (vertically or horizontally) in two or more parts. The user can open and edit different files in the different parts of the pane. They may also display, e.g., a grammar file in one part of the pane and the XLE buffer for parsing text in another part of the pane. These functionalities are accessible using special keyboard commands or the drop-down menus. Thus, it is in fact not necessary to open files in separate Emacs windows; however, to command these functionalities, it is necessary to know a) that Emacs offers them and b) the menus and keyboard shortcuts that activate them.

In eXLEpse, these functionalities are provided in a more intuitive way. See Figure 2 for an overview of the eXLEpse perspective. The files of a grammar folder may be opened by double-clicking them in the project explorer on the left. If a file is already open, the newly opened file will be displayed in a new tab. The user may create additional editor windows (with their separate tabs) by dragging files to the edge of the central editing area of eXLEpse; see Figure 3. Files may be distributed across editor windows and tabs within editors using drag-and-drop.

2.2 Different Keyboard Shortcuts

The default keyboard shortcuts of Emacs do not conform to current conventions. By consequence, the keyboard shortcuts for opening, creating, or saving files in Emacs differ from the shortcuts used throughout well-known operating systems. Hence, a user who is already familiar with conventional operating systems or standard applications (e.g. word processor, text editor) has to learn Emacs shortcuts from scratch. For instance, in Emacs M-w (for “wipe”) is set as a shortcut to cut selected regions whereas most WIMP⁶ applications use Ctrl+X (Cmd+X on Mac OS). This complicates the fluent grammar development, as the novice user needs to shift focus away from grammar development to look up unknown shortcuts. In contrast, all shortcuts of eXLEpse are in accordance with current editor standards set across operating systems.

2.3 XLE Programming

To implement LFG, XLE uses a syntax that is distinct from programming or scripting languages (e.g. Java, C#/WPF) and thus a beginner needs to learn XLE syntax from scratch. In combination with the sparse error messages, this can lead to user frustration. As an example, consider the delimiter symbol that separates the entries within a grammar (e.g. rules, lexical entries) — in XLE, the delimiter symbol is the dot, which is the smallest visible lexical symbol and therefore hard to find on the screen. This gives rise to errors in grammar development as the result of a missing dot or other delimiters (e.g., semicolons). If an error occurs, XLE prints error messages to the standard command line output and the grammar developer has to distinguish regular output and error messages. However the details of these error

⁶Windows Icons Menus Pointers



Figure 1: Possible user desktop during grammar development with XLE and Emacs

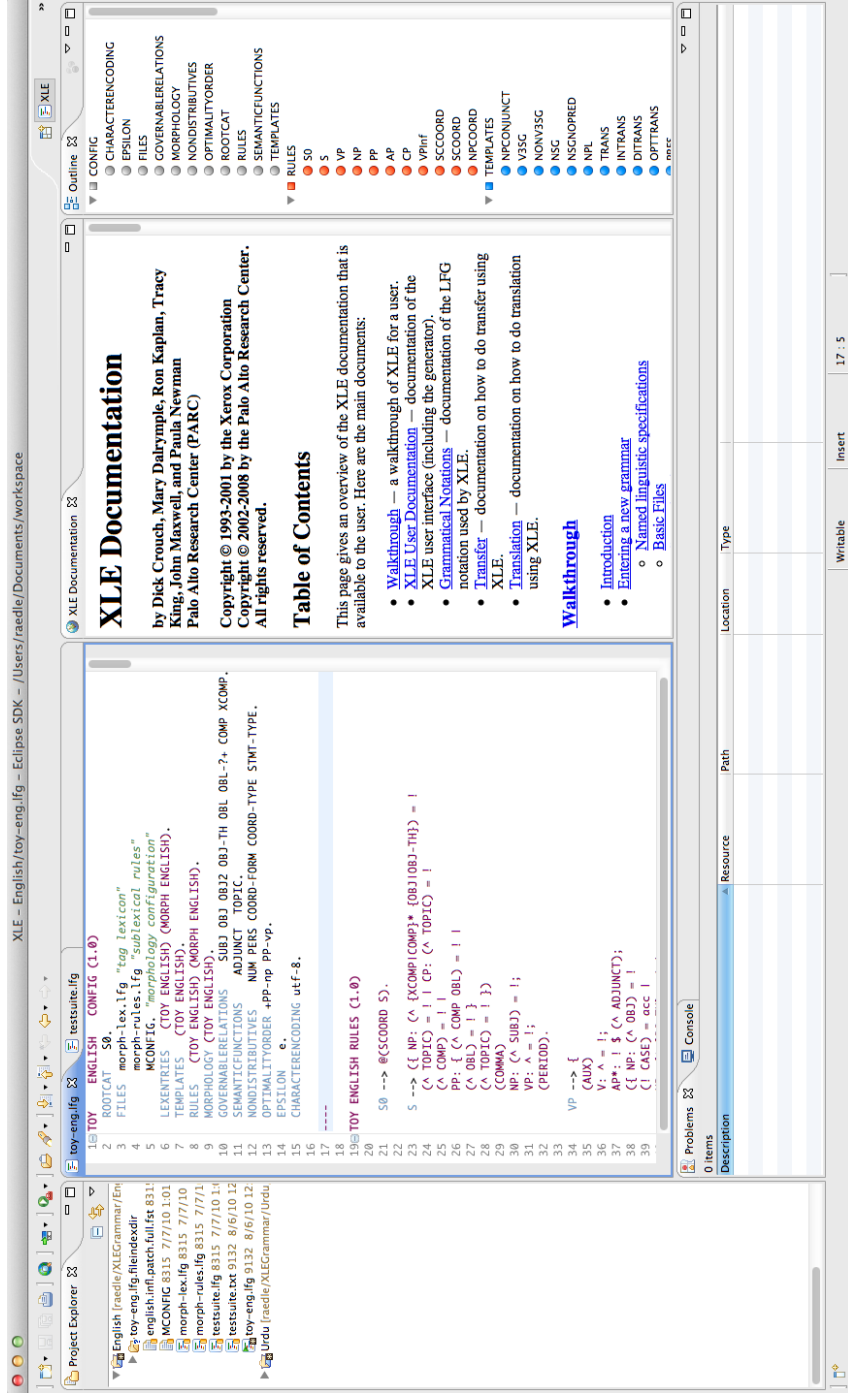


Figure 2: The eXLE Eclipse XLE perspective consists of a project explorer (left), an XLE editor (middle-left), an XLE documentation browser (middle-right), a grammar outline (right), console input and output and a problems view (both at the bottom).

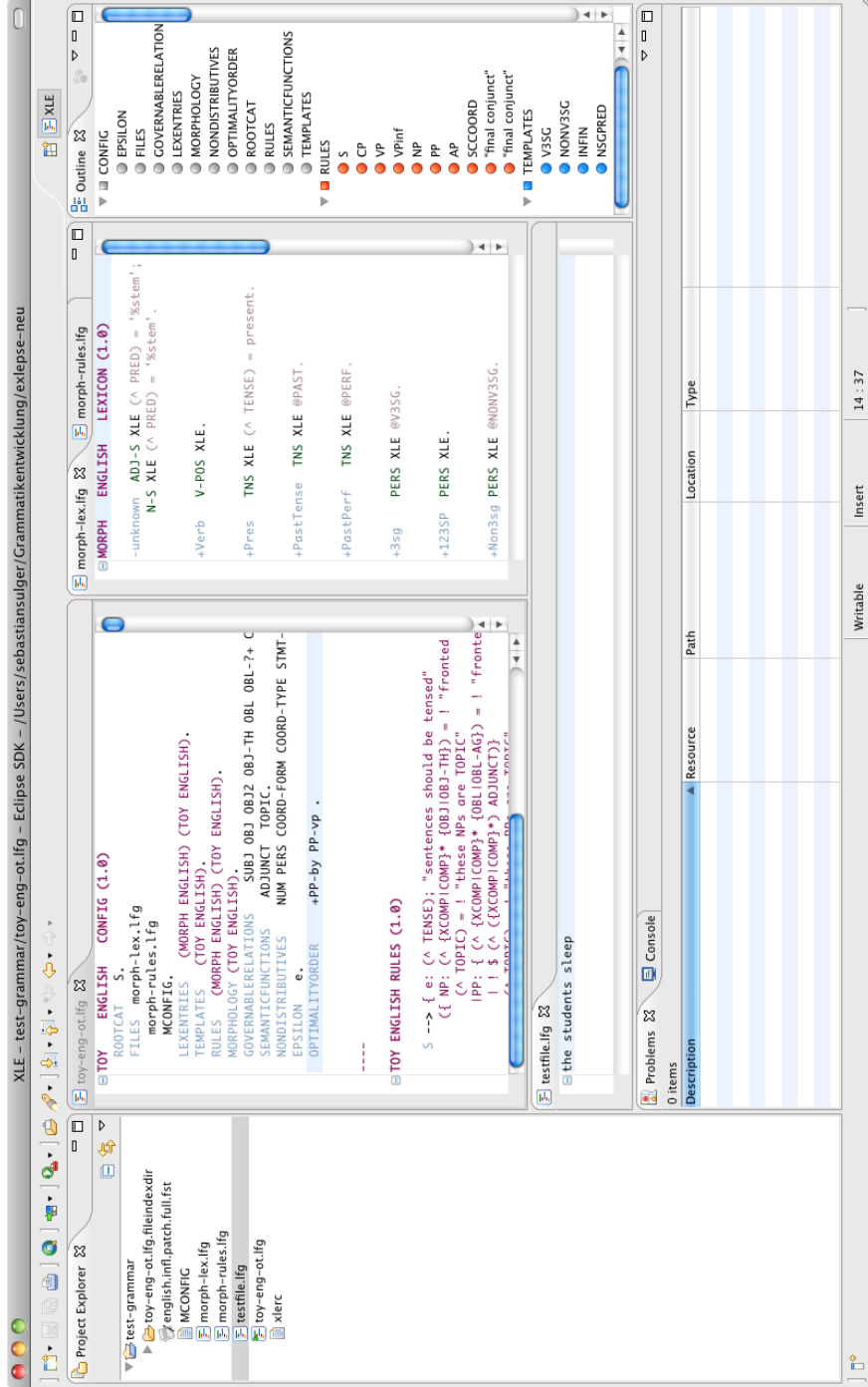


Figure 3: The xLEpse XLE perspective allows for editing distributed across editor windows and tabs using drag-and-drop. In the above screenshot, toy-eng-ot.lfg is displayed in the editor to the left, while the testsuite testfile.lfg is displayed in the bottom editor, and morph-lex.lfg and morph-rules.lfg are displayed together in the editor to the right, each file in its own tab.

messages are problematic because of the sparse clues that are given to identify the error in the grammar.

In contrast to Emacs, eXLEPse evaluates grammars automatically after saving a grammar file. The error message and a line number is shown in the problems view if an error occurs during evaluation (see Figure 4). The grammar developer is provided with rapid, incremental feedback that allows them to make fewer errors and complete grammar development in less time (Shneiderman, 1983). In addition, new error detection can be added if future XLE releases introduce further error types. The eXLEPse preference pane provides an input dialog to add additional problem types based on regular expressions. One of the five different problems that are currently recognized is the template invocation error, an example of which is given in (1).

- (1) Template invocation error near line 242, column 105 in file /Users/xle/Documents/toy-eng.lfg: The invocation of template NPL has 1 argument, but the definition has 0 parameters at line 131 in /Users/xle/Documents/toy-eng.lfg.

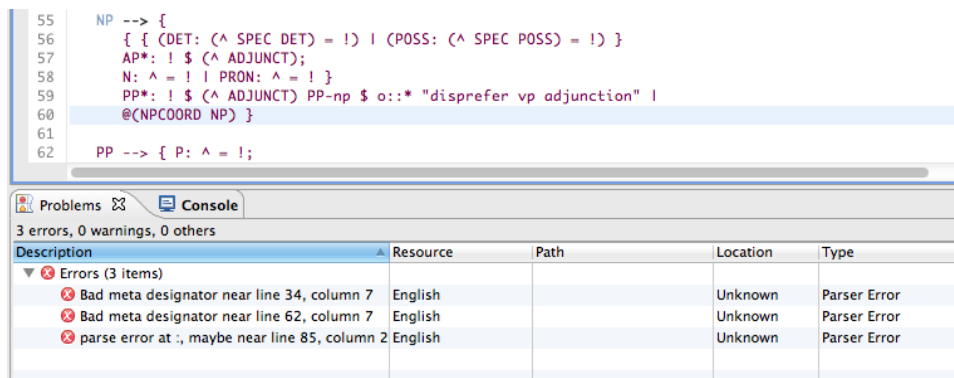


Figure 4: The problems view in eXLEPse displays errors of a currently opened grammar file and highlights error position. Error positions are extracted from XLE console output.

2.4 Syntax Highlighting

The Emacs editor highlights different aspects of an LFG file such as rules, templates, or comments. Some aspects, however, are not highlighted, for instance the different parts of a configuration section (see Figure 6(a)). The eXLEPse editor highlights the previously mentioned parts and moreover offers support for code completion within the configuration section (see Figure 6(b)). When pressing the shortcut Ctrl+Space the code completion popup opens and displays suitable configuration templates. The programmer can select a desired template to auto-complete the input. Also, the colors used by eXLEPse to highlight the grammar code can

be changed by the user using the eXLEpse preference pane to match their personal preferences (see Figure 5).

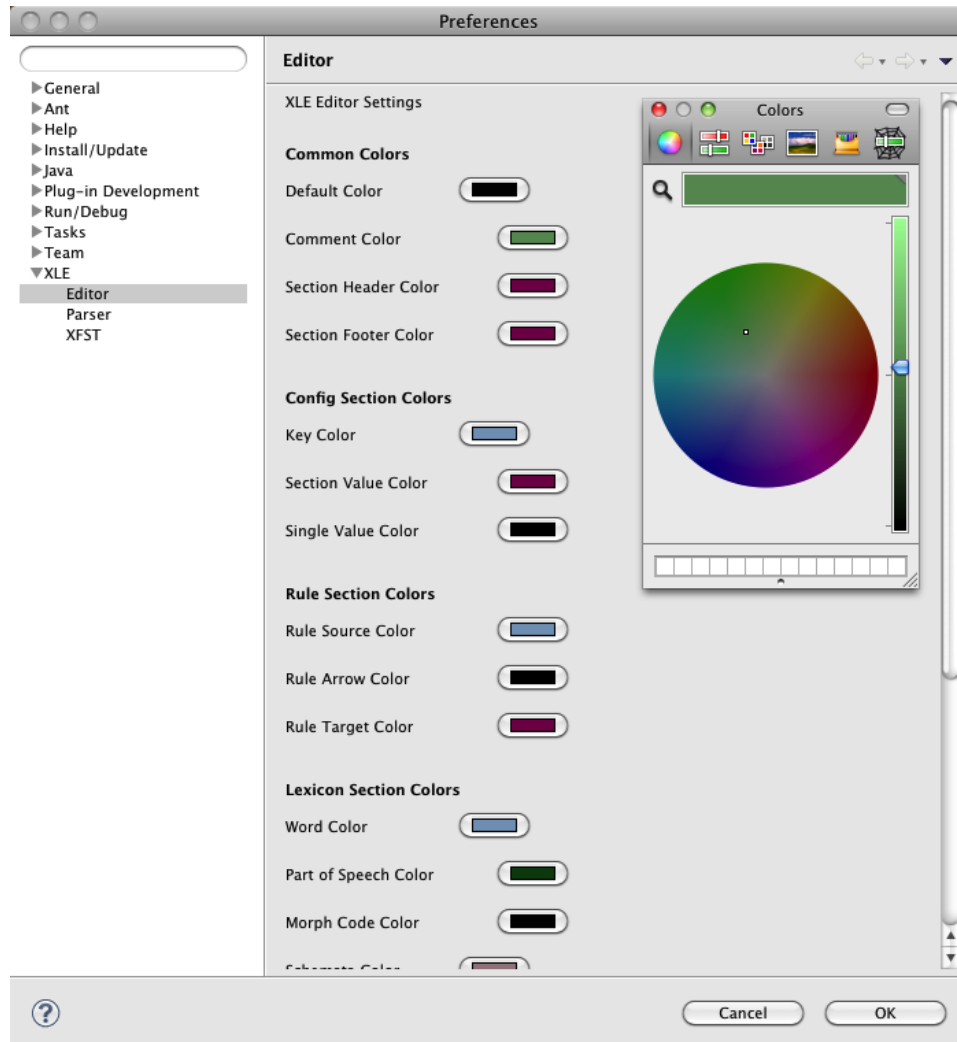


Figure 5: The colors used for the syntax highlighting in eXLEpse can be changed using the eXLEpse preference pane.

2.5 Grammars as Projects

Large grammars often consist of several files (e.g. morphological rules or test suites). Therefore, eXLEpse offers the possibility of assigning files of a grammar to a logical construct, so called projects (see Figure 7). Multiple projects are further organized in workspaces. On the one hand, this reduces complexity by chunking different grammars into smaller units. On the other hand it provides cen-

```

File Edit Options Buffers Tools LFG Help
TOY ENGLISH CONFIG (1.0)
ROOTCAT S0.
FILES morph-lex.lfg "tag lexicon"
morph-rules.lfg "sublexical rules"
MCONFIG. "morphology configuration"
LEXENTRIES (TOY ENGLISH) (MORPH ENGLISH).
TEMPLATES (TOY ENGLISH).
RULES (TOY ENGLISH) (MORPH ENGLISH).
MORPHOLOGY (TOY ENGLISH).
GOVERNABLERELATIONS SUBJ OBJ OBJ2 OBJ-TH OBL OBL-?+ COMP XCOMP.
SEMANTICFUNCTIONS ADJUNCT TOPIC.
NONDISTRIBUTIVES NUM PERS COORD-FORM COORD-TYPE STMT-TYPE.
OPTIMALITYORDER +pp-np pp-vp.
EPSILON e.
CHARACTERENCODING utf-8.
-----
TOY ENGLISH RULES (1.0)
S0 --> e(SCOORD S).
S --> (( NP: (^ (XCOMP|COMP)* (OBJ|OBJ-TH)) = !
(^ TOPIC) = ! | CP: (^ TOPIC) = !
(^ COMP) = ! |
PP: { (^ COMP OBL) = ! |
(^ OBL) = ! }
(^ TOPIC) = ! }
(COMMA)
NP: (^ SUBJ) = !;
VP: ^ = !;
(PERIOD) .
VP --> {
(AUX)

```

(a) Standard Emacs view with XLE syntax highlighting.

```

*toy-eng.lfg [X] ENGLISH CONFIG (1.0)
1 TOY ENGLISH CONFIG (1.0)
2 ROOTCAT S0.
3 FILES morph-lex.lfg "tag lexicon"
4 morph-rules.lfg "sublexical rules"
5 MCONFIG. "morphology configuration"
6 LEXENTRIES (TOY ENGLISH) (MORPH ENGLISH).
7 TEMPLATES (TOY ENGLISH).
8 RULES (TOY ENGLISH) (MORPH ENGLISH).
9 MORPHOLOGY (TOY ENGLISH).
10 GOVERNABLERELATIONS SUBJ OBJ OBJ2 OBJ-TH OBL OBL-?+ COMP XCOMP.
11 SEMANTICFUNCTIONS ADJUNCT TOPIC.
12 NONDISTRIBUTIVES NUM PERS COORD-FORM COORD-TYPE STMT-TYPE.
13 OPTIMALITYORDER +pp-np pp-vp.
14 EPSILON e.
15 CHARACTERENCODING utf-8.
16
17 BASECONFIGFILE
18 ENCRYPTFILES
19 EXTERNALATTRIBUTES
20 TOFEATURES
21 GENOPTIMALITYORDER
22 GRAMMARVERSION
23 LEXENTRIES
24 MORPHACCESSPATH
25 OTHERFILES
26 PARAMETERS
27 PP: { (^ COMP OBL) = !
(^ OBL) = ! }
28 (^ TOPIC) = ! }
29 (COMMA)
30 NP: (^ SUBJ) = !;
31 VP: ^ = !;
32

```

(b) eXLEpse XLE editor with improved syntax highlighting and code completion.

Figure 6: The two different XLE editors – Emacs and eXLEpse – in a side-by-side comparison.

tralized access to grammars so that fast and frequent switches between projects is enabled. The grammar developer can display several grammar files of different projects simultaneously and within a single eXLEpse instance just by pointing at double-clicking the files. Files may also be shared across grammar projects, e.g. by putting them in the top-level directory of the workspace that contains the grammar projects.⁷

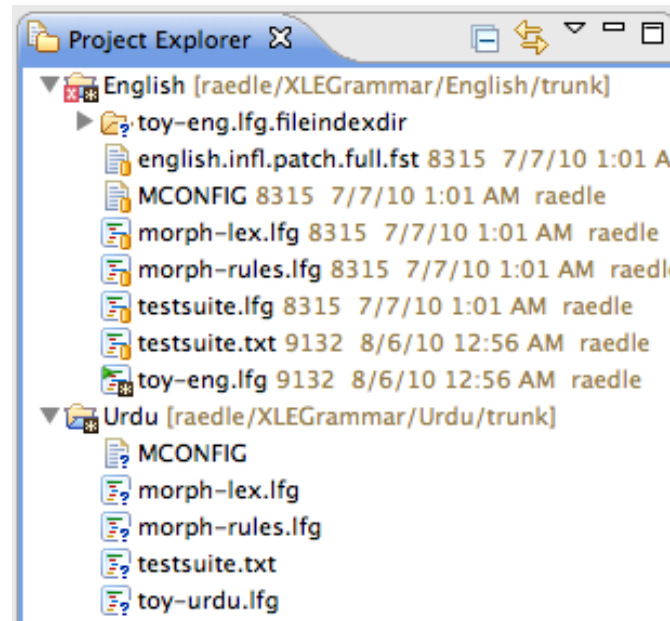


Figure 7: XLE grammars consisting of multiple files can be grouped into logical units, so called projects.

2.6 The Outline View

In Emacs, rules, templates and lexical entries contained in a grammar file may be browsed using the LFG drop-down menu *Rules, templates, lexicon menus*. Selecting the desired rule, template or lexical entry causes Emacs to jump to the respective place in the grammar file.

In eXLEpse, the outline view summarizes the contents of a currently opened grammar file (see Figure 8). Hence, contents are grouped into config, rules, templates, and lexical entries. Therefore users can rapidly overview a grammar and highlight the corresponding grammar fragment by selecting an item of the outline view. This functionality relies on the concept of brushing and linking (Buja et al.,

⁷The common templates and common features files of the ParGram project are examples of grammar files that are commonly shared across grammars (Butt et al., 2003). In eXLEpse, these can be put in the workspace directory for easy access.

1991). In contrast to Emacs, the outline feature in eXLEpse is provided to the grammar developer without the need to navigate through the application menu by selecting rules, lexical entries or templates from nested drop-down menus. The outline view is provided to the user immediately after opening the file.

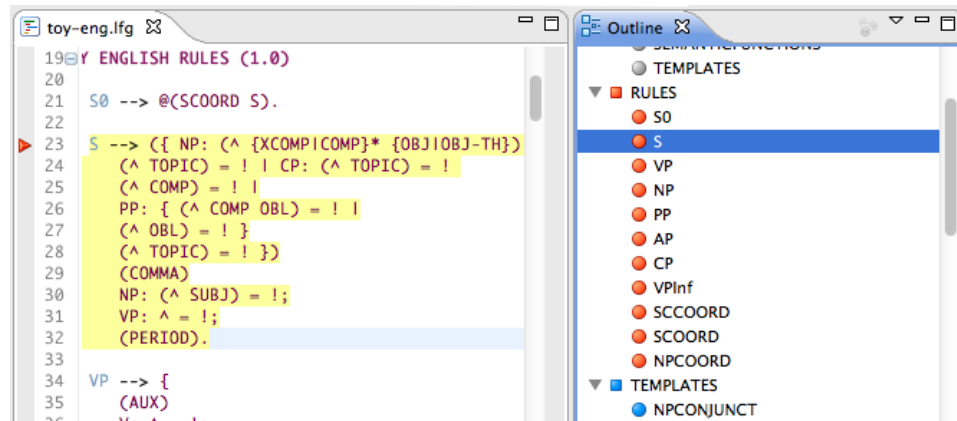


Figure 8: This view outlines config, rules, templates, and lexical entries and thus provides a quick access to the contents of a grammar file.

2.7 Sentence Parsing

When using Emacs, text parsing with XLE is executed in a separate buffer. If the user does not use split buffers as described in Section 2.1, they need to arrange buffer windows in order to perceive application states (see Figure 1). Although rearrangement of buffer windows is possible, the user has to layout windows manually. Whenever the grammar developer wants to parse text, they are forced to switch to the proper buffer, which could be hidden by other windows. Often, users will have an XLE buffer running somewhere in the background of an Emacs pane. They then have to select that pane, and cycle through all buffers opened in that pane to find the XLE buffer.

Moreover, when changes are made to rules within a grammar, the currently running XLE process has to be restarted for the changes to take effect.⁸ While XLE produces a warning if there are non-lexicon changes without a restart of XLE, this can lead to user frustration, as novice users may forget to restart XLE and ignore the warning. Emacs buffers displaying grammar files include commands from the LFG drop-down menu to either start a new XLE process, or start an XLE process in an XLE buffer or switch to an existing one (among other commands, see Figure 9(a)). Emacs buffers that display a running XLE process include commands from

⁸Note that it is not necessary to restart XLE if only the lexicon is affected by changes (i.e., additions to the lexicon or changes in lexical entries), since the lexicon sections of a grammar are re-indexed by XLE at parse-time.

the XLE drop-down menu to either restart XLE (alternatively, using the XLE-specific shortcut `Ctrl+c+Ctrl+f`) or start a new XLE process in another window (among other commands, see Figure 9(b)). That is, there is no possibility to directly restart XLE from a grammar buffer and switch to the XLE buffer at the same time.

Another problem with sentence parsing in the non-eXLEpse architecture is connected to the XLE command `create-parser`. When a new XLE process is started, the main grammar file containing the rules etc. is not loaded automatically, unless there is an `xlerc` configuration file for the grammar. If the command `create-parser grammar-file` is put in the `xlerc` file, the XLE process will load the specified grammar when starting up. In many cases, the `xlerc` file will only contain that single command.⁹

In eXLEpse, none of these issues arise. Because of the integrated design, there is no need to switch between windows. The user can specify which of the files of a project is the main grammar file containing the configuration section; that file will receive a small green arrow next to the file name (see Figure 7). When the user attempts to parse a sentence, eXLEpse automatically calls `create-parser` on that specified file, effectively eliminating the need for an `xlerc` file in most cases. `create-parser` is called again automatically when a user saves a grammar file, so that it is not necessary to restart XLE manually.

Current approaches that aim at parsing sentences with XLE require at least basic experience with a command line tool (e.g., shell). Therefore, grammar developers have to learn how to use the shell in advance. However, eXLEpse hides this complexity and provides simple access to the parser actions through a toolbar or a context menu (see Figure 10). The grammar developer can either input text manually (see Figure 10(a)) or choose to parse a pre-selected text (see Figure 10(b)).

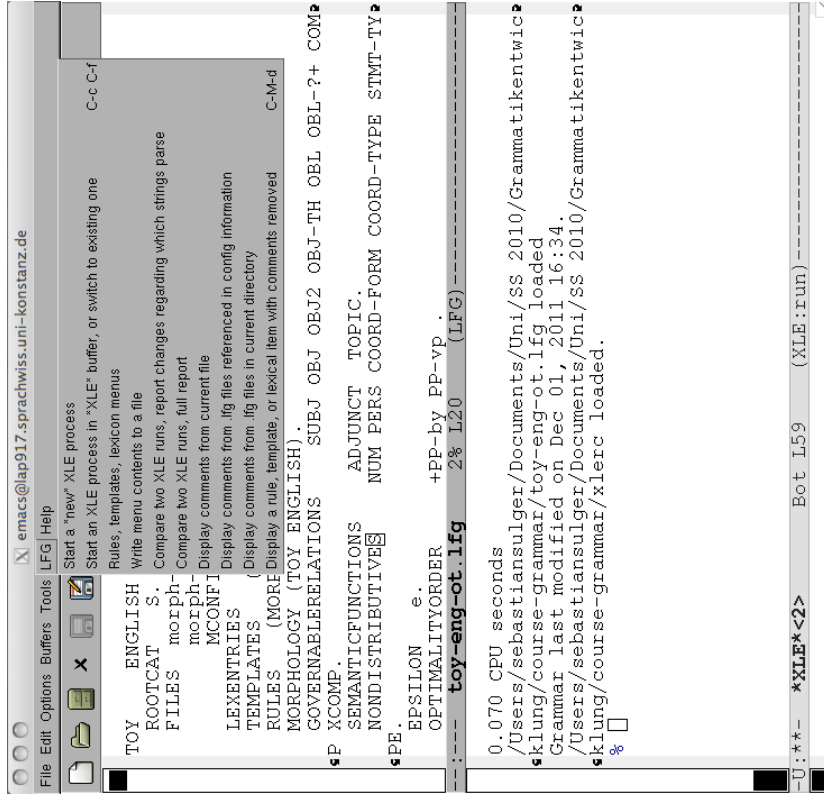
2.8 Console Input

Only basic XLE commands (i.e., parsing sentences, parsing parts of a testsuite) are implemented in the current version of eXLEpse via icons and context menus (see Figure 10), although we plan to integrate more commands in the future (see Section 3). In order to provide eXLEpse with fully-fledged XLE support, a console has been included (see Figure 11). The console constitutes a command line interface to XLE. It allows text input and enables a grammar developer to give arbitrary commands to the XLE process. Any XLE command that is otherwise not accessible may be issued to XLE using the console from within eXLEpse.

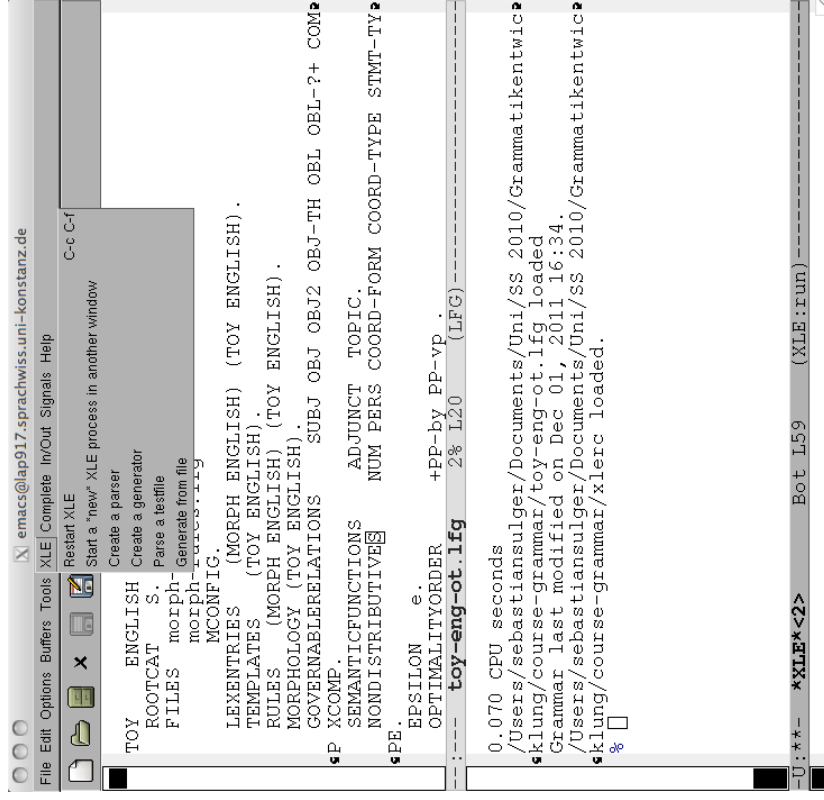
2.9 XLE Documentation

For programming tasks it is very important to have documents at hand that describe either the programming syntax or the application programming interface (API).

⁹Note that `xlerc` files may contain useful shortcut commands, ranging from manipulating OT marks to running complicated testsuite commands to customizing the XLE display windows. For the novice, however, these commands will not be applicable in the majority of cases.

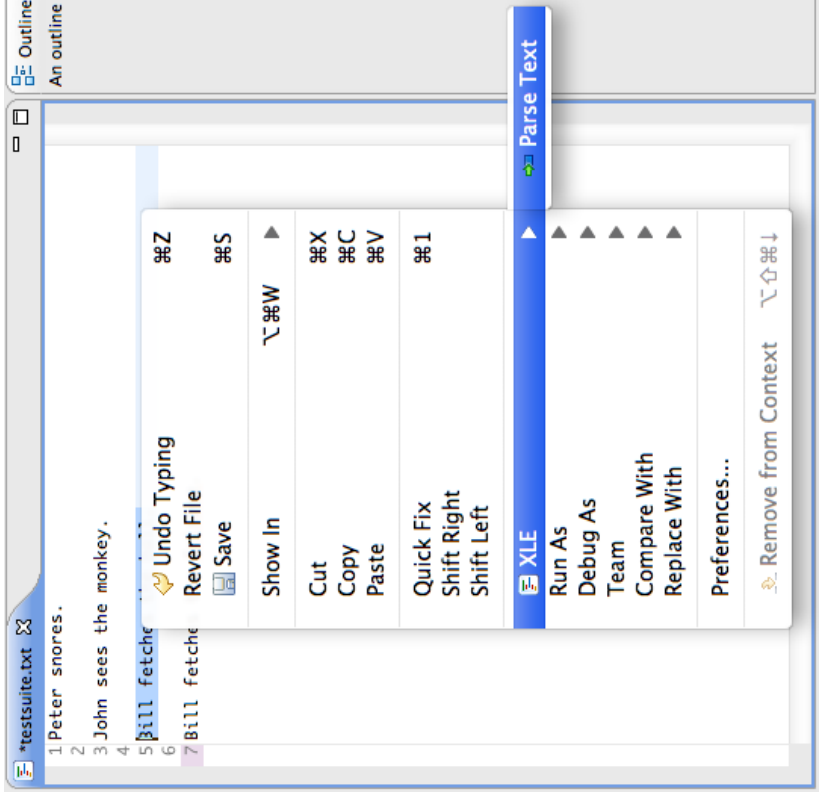


(a) Emacs editor with LFG drop-down menu, accessible from buffers with grammar files.

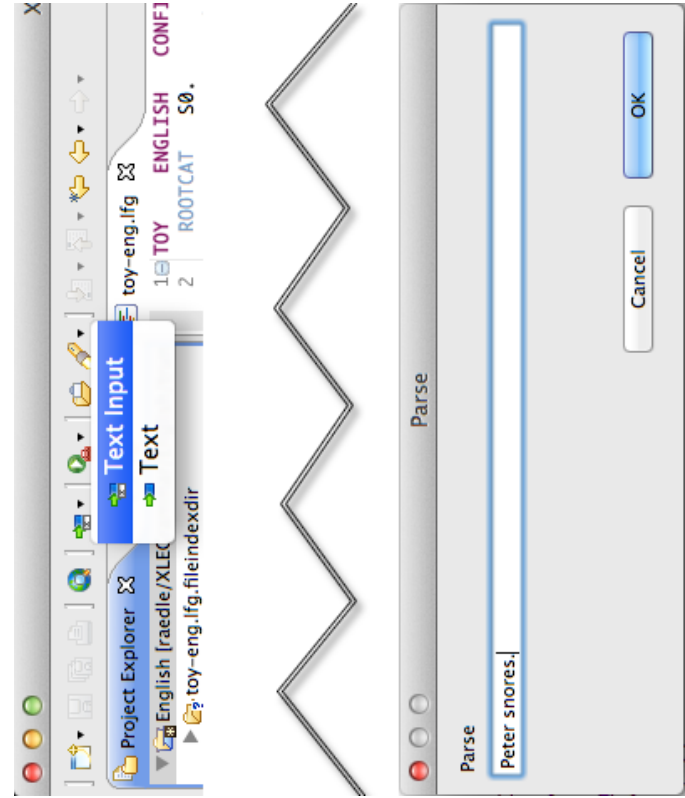


(b) Emacs editor with XLE drop-down menu, accessible from XLE buffers.

Figure 9: The two different drop-down menus to interact with XLE from within Emacs.

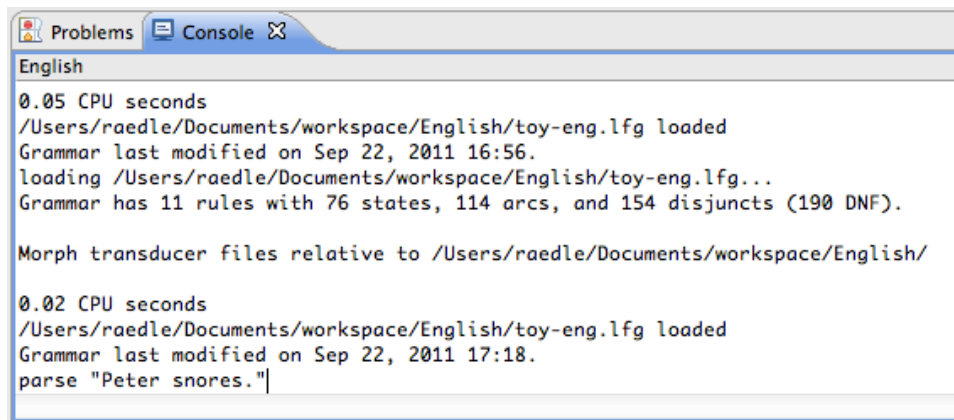


(b) Parse an arbitrary selected text with help of the context menu.



(a) Input a text using a dialog.

Figure 10: The two different options to parse text with eXLEpse. Option (a) allows manual input of text whereas option (b) enables parsing of a given, pre-selected text.



```
English
0.05 CPU seconds
/Users/raedle/Documents/workspace/English/toy-eng.lfg loaded
Grammar last modified on Sep 22, 2011 16:56.
loading /Users/raedle/Documents/workspace/English/toy-eng.lfg...
Grammar has 11 rules with 76 states, 114 arcs, and 154 disjuncts (190 DNF).

Morph transducer files relative to /Users/raedle/Documents/workspace/English/

0.02 CPU seconds
/Users/raedle/Documents/workspace/English/toy-eng.lfg loaded
Grammar last modified on Sep 22, 2011 17:18.
parse "Peter snores."|
```

Figure 11: eXLEPse’s console view allows input of arbitrary not yet graphically supported XLE commands.

In the Emacs interface to XLE, users can enter the command `documentation`, which launches a web browser and displays the XLE documentation.

In eXLEPse, the XLE documentation (Crouch et al., 2011) is accessible directly in the eXLEPse window; an external browser application is not needed. Moreover, the documentation window integrates seamlessly into the eXLEPse perspective and can be placed next to the XLE editor in order to program and to look up a definition or examples simultaneously (see Figure 12).

3 Conclusion and Future Work

We have presented eXLEPse, an easy-to-use editor plugin for developing computational LFG grammars. eXLEPse supersedes both shell-based parsing and command input as well as the Emacs editor. It represents a complete development platform that seamlessly integrates into operating systems with help of the Eclipse platform. eXLEPse uses the XLE binaries to parse sentences and displays parser results to the user by communicating with the X11 window system.

Future work includes the following improvements to eXLEPse. We plan to integrate the `parse-testfile` command in the editor, providing a button for file input similar to the parse button. Also, we intend to include code reformatting functionalities (indentation etc.) similar to the `Esc+q` command in Emacs, which provides a reliable way to render grammars more readable. Moreover, we plan to change the outline pane to refer not only to the currently opened file, but to the whole grammar project. This way, the user can work with a more complete overview of rules, templates and lexical entries included in a grammar. Also, popup menus can be integrated and open up when a specific template or rule is selected by a user. The popup shows where that template or rule is called in the

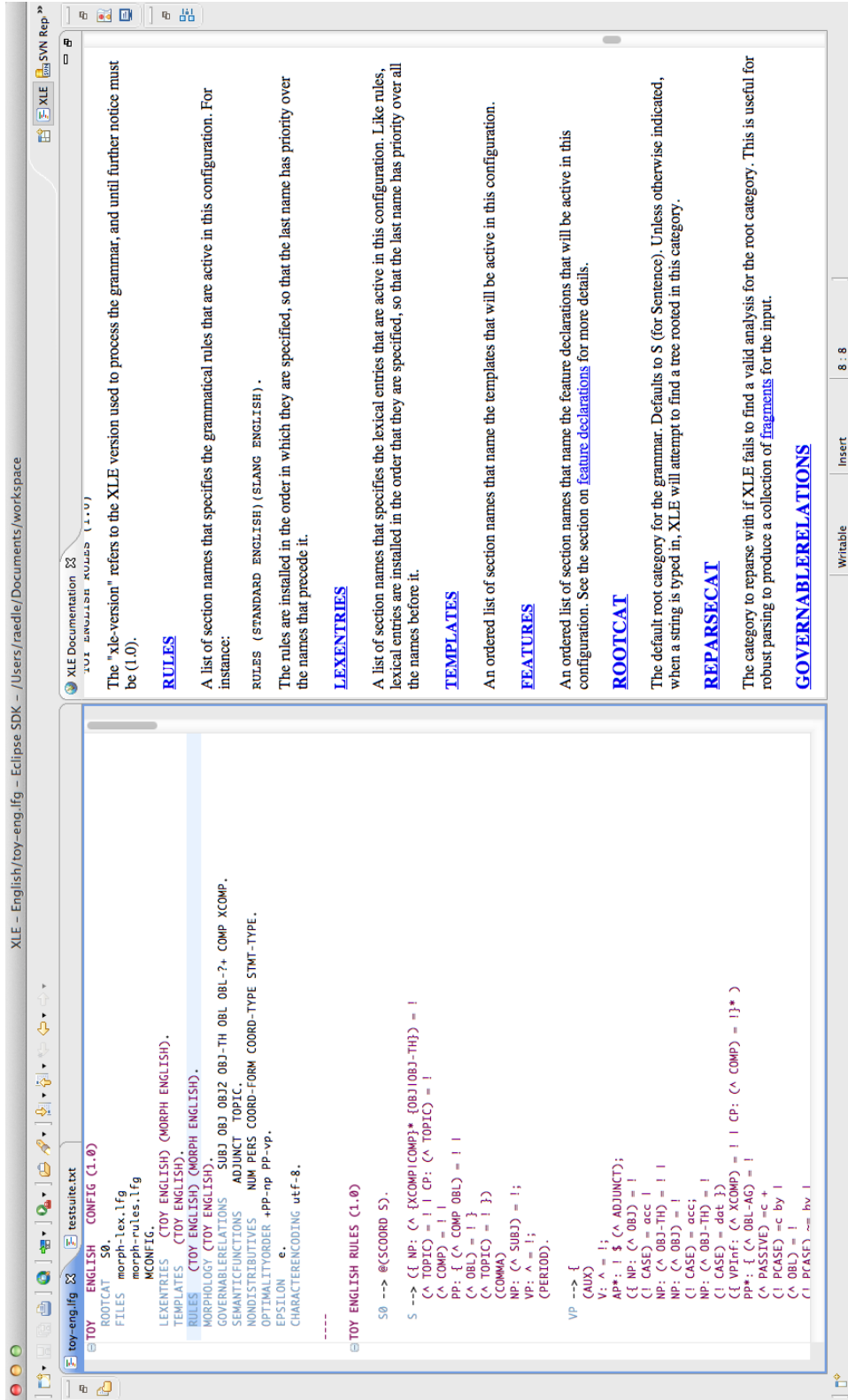


Figure 12: A documentation view is integrated in eXLEpse.

grammar. Incorrect template calls (e.g., template calls with the wrong number of arguments) could be detected more easily in this manner. Functionalities for creating and editing finite-state morphologies (Beesley and Karttunen, 2003) from within eXLEpse are currently being investigated. Finally, support for generating documentation from commented XLE grammars or additional XML documentation files as suggested by Dipper (2003) could be integrated straightforwardly into eXLEpse.

References

- Beesley, Kenneth and Karttunen, Lauri. 2003. *Finite State Morphology*. Stanford, CA: CSLI Publications.
- Buja, Andreas, McDonald, John Alan, Michalak, John and Stuetzle, Werner. 1991. Interactive data visualization using focusing and linking. In *Proceedings of the 2nd conference on Visualization '91, VIS '91*, pages 156–163, Los Alamitos, CA, USA: IEEE Computer Society Press.
- Butt, Miriam, Forst, Martin, King, Tracy Holloway and Kuhn, Jonas. 2003. The Feature Space in Parallel Grammar Writing. In *Proceedings of the ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*, pages 9–16.
- Crouch, Dick, Dalrymple, Mary, Kaplan, Ronald M., King, Tracy Holloway, Maxwell III, John T. and Newman, Paula. 2011. *XLE Documentation*. Palo Alto Research Center.
- Dalrymple, Mary. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press.
- Dipper, Stefanie. 2003. *Implementing and Documenting Large-Scale Grammars*. Ph.D.thesis, University of Stuttgart.
- Shneiderman, B. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 57–69.