The PracTEX Journal, 2007, No. 3
Article revision 2007/08/12

ACIDE: An Integrated Development Environment Configurable for LaTeX

Fernando Sáenz-Pérez

Website http://www.fdi.ucm.es/profesor/fernan
Address Facultad de Informática, Universidad Complutense de Madrid, Spain

Abstract This article introduces the configurable integrated development environment ACIDE, which is an ongoing development project currently in alpha status. It is cross-platform, open-source, and free, and will be distributed under GPL. Although targeted to any programming language environment, including compilers, interpreters, and database systems, in particular it is well-suited to tasks required for LaTeX and TeX document preparation systems. It manages projects, is useful in dealing with multifile documents, allows configurable menus, has a toolbar for executing commands and lexical tokens for syntax colouring, and even grammars to identify programming (syntactical) errors on the fly.

Keywords: Tools, Free, Open-source, Cross-platform, Editor, IDE, LaTeX, TeX

1 Introduction

As a system implementor of declarative languages (e.g., the constraint functional logic language Toy [16], and the Datalog Educational System DES [17]), I needed an Integrated Development Environment (IDE) for each of them. Because of the need to work with several languages, developing a configurable IDE was almost a necessity. Therefore, during the course "Computing Systems" (which belongs to a postgraduate study), I directed a team of students in developing such a system which we called A Configurable IDE (hence the acronym ACIDE). The benefits are not only for system implementors who need an IDE for their systems, but also for other users, such as database users, and more importantly for readers of this journal, who can use it as already configured for LaTeX and TeX (cf. the suggestions in Section 4).

The requirements of this project were several-fold:

Configurable The main objective of this system is to be as highly configurable as possible, keeping the configurations easy and portable by means of text files. This configuration includes menus, a toolbar for executing commands, lexical tokens for syntax colouring, and even grammars to identify programming errors on the fly.

Simple We looked for, first, a simple, intuitive GUI. Second, a system that can be easily configured for a requested programming system, both the language lexicon as well as its grammar. Third, it must be easily installed, even requiring no installer. In fact, only decompressing an archive and executing a file should be enough to start working with the system. And, fourth, to have a simple multifile editor for basic users, which can also be used by advanced users who need extra features (projects, language configurations, etc., see Sections 2 and 3)

Internationalized In order to have a wider audience, text files are used to define all the texts of the GUI, so that the system can be easily localized to different languages.

Open-source We do believe that the open-source alternative will produce better software and systems, and allow developers to contribute enhancements. This is fitting in a university context, so that students will have sources available to learn and practice with them. In an unusual step, the project's main application has been developed without using other software as a basis. While doing this would have allowed more rapid development, starting from scratch gave the students a unique opportunity, one of the few they will probably have in their careers. Nonetheless, another free and open-source project ANTLR [2] has been used for parsing.

A Cross-Platform System Despite the basic need to develop the same system for several platforms, many of us use several different operating systems and therefore want to have the same applications available on several platforms. As a consequence, the selected development language was Java, which is widely known

and works on any platform. Of course, this was not the only alternative, since one can use C#, for example. But for developing a cross-platform system that can be widely used and modified in an open-source project, Java was chosen.

Free One of our goals in the university is to share knowledge, and we make our work freely available to others. Further, providing free software allows for widespread usage. In addition, our university and several national projects support us as researchers and lecturers (see Section 6), so we do not need additional income from developing the software.

Before starting to develop this project, other related systems were analysed. On the one hand, there are multi-file editors such as JEdit [9], which is free, cross-platform, configurable and with many nice features. However, JEdit is targeted as a file editor, not as an IDE, and lacks features such as parsing and shell. Crimson Editor [5] is a Windows editor also configurable and allows writing commands to send to a shell in order to do code compilations and run executables. However, it also lacks parsing capability, and moreover it is not cross-platform.

On the other hand, there are several IDEs which can be categorised into specific or general purpose. Specific IDEs can also be configured in various ways. For instance, WinEdt [14] is a Windows editor highly targeted to LaTeX and TeX documents, and it is quite powerful for this document preparation system. However, it is neither cross-platform, nor free, nor open-source. Other free IDEs are LaTeX Editor [10], WinShell [15], and TexnicCenter [13] (all of these only for Windows), Texmaker [12] (free and open-source), LyX (a WYSIWYM document processor for *nix platforms), and several others. JBuilder [7] and JCreator [8] are examples of commercial software for programming IDEs that also may provide hints for detected functionalities requested by users. But maybe the best exponent of an open-source, free project is Eclipse [6], which is a development platform comprised of extensible frameworks, tools and runtime libraries for building, deploying and managing software, and it comes with many programming environments already configured. The main drawback is that it cannot be easily configured to deal with programming environments other than the ones it uses.

Therefore, having stated our original goals and reviewed related systems, we conclude that no system completely fits our requirements, and that our proposal might be of interest for other users. In particular, LaTeX/TeX users might find an

alternative to existing editors. Note, however, that our system is emerging and is in a very early development stage, but its future development will be guided by the aforementioned objectives as well as user feedback.

The next sections describe the system. First, Section 2 states the concrete features of the system as well as its main current limitations. Section 3 describes ACIDE from a user point-of-view. Section 4 summarises an instance of the system configured for LaTeX. In Section 5 some conclusions and future work are noted. Finally, acknowledgements are offered (Section 6).

2 Features and Limitations

This section first describes the inherent features of ACIDE, although not all of them or their functionalities are fully operational in the current development stage. In addition, some limitations of the system are noted.

2.1 Features

- **Multi-File Editor**. Many files can be opened and two views for the same file and window are possible. The usual copy, paste, undo, and redo operations are possible via contextual menus and application menus. File printing is also provided. Syntax, programming errors, and delimiter pairs (parentheses, square brackets, ...) are highlighted.
- **Configurable.** Menus, toolbar, syntax highlighting (which includes tokens, delimiters, and remarks), parsing (from an EBNF language description), compiler, shell, and GUI language (currently localised to English and Spanish). Tool variables are provided to configure the commands assigned to the toolbar. For instance, `$activeFile$`, which keeps the complete file name of the selected file in the tool.
- **Text-based Configuration.** This tool allows configuration of all of its parameters by the use of text files, which permits using the provided dialog boxes without the need to resort to manually encode the data in the plain text files, as well as editing these files with a text editor, and even generate them with applications. For instance, one could generate the lexicon of a language from its formal description (e.g., EBNF).

- **Project Management.** Logical views of projects arranged in folders, which contains files or other folders. A concrete file can be selected as compilable or as a main file for a compilation. Also, several files can be selected for compilation by specifying the file extension.
- **Logging.** For implementation purposes, a log has been added, which helps in developing the tool.

2.2 Limitations

- **Macros.** They are quite helpful for defining scripts inside the tool. Currently, this task has to be passed to the operating system. That is, we can define a command as a system call, which can be a batch or script file. Toolbar commands might use these macros.
- **Debugging.** An almost indispensable feature for programming, but with current limitations: the tool can only handle debugging an interpreter with input/output via the standard streams. It can be used with toolbar buttons, therefore easing the task. Think, for instance, of the debugging model of Prolog.
- **Menu Commands.** There is a prefixed set of menu commands which can be enabled or disabled, but it is not possible to define new menu commands in the same way they are defined for the toolbar.
- **Command History.** This is quite useful for the shell, and also provides an autocompletion feature.
- **Dictionary**. A feature which operates the opposite of syntax highlighting, in that unrecognised words are highlighted, and in addition suggestions are provided for fixing errors.
- **Editor Facilities.** Other features can be useful, such as shorthand methods for completing text fragments (e.g., type `ltx` and get `\LaTeX`), text autocompletion, case changing, word wrapping, patterns (e.g., LaTeX environments as `itemize`, therefore automatically filling `\begin{itemize}`, `\item`, and `\end{itemize}`), autoindent, font selection for editor windows and console, and many other features.

In addition to this short list, it is possible to find many more limitations. Here, the most important ones, in our view, have been listed.

3 Description

3.1 Technology

The implementation of the tool has been completely done using Java under Eclipse. Version control was kindly provided by Berlios [3]. Students prepared the documentation using Rational Rose for UML diagrams, MS Project for Gantt charts, and MS Word. It seems that our graduate students do not feel the need for more powerful text editing when working with LaTeX.

3.2 The Main GUI

Figure 1 shows the main GUI of ACIDE. It consists of three main panels. The left panel shows the organisation of the current project, the MDI windows in the right are the opened files which may belong to the project (files may be opened without assigning them to the project). Below, the shell panel is shown, which allows user interaction. The case shown is a command console of Windows XP. Both the shell and project panels can be hidden. Moreover, there is no need to work with projects if this flexibility is not needed; a regular user may use the system as is. The status of the GUI is remembered for the next time the tool is executed. If the tool opens a project, its status when it was last saved is restored.

The menu bar includes some common entries:

- File. For file-related operations (New, Open, Close, Close All for closing all opened files, Save, Save As, Save All, Print and Exit). Missing useful options: Recent Files.
- Edit. For clipboard-related operations, search, undoing changes, and go to line number. Missing useful options: Select All, Insert Comment, Remove Comment.
- Project. For project-related operations (New Project, Open Project, Close Project, Save Project, Save Project As, Add File to the current project, Remove File from the current project, New Folder in the project structure, Delete Folder from the

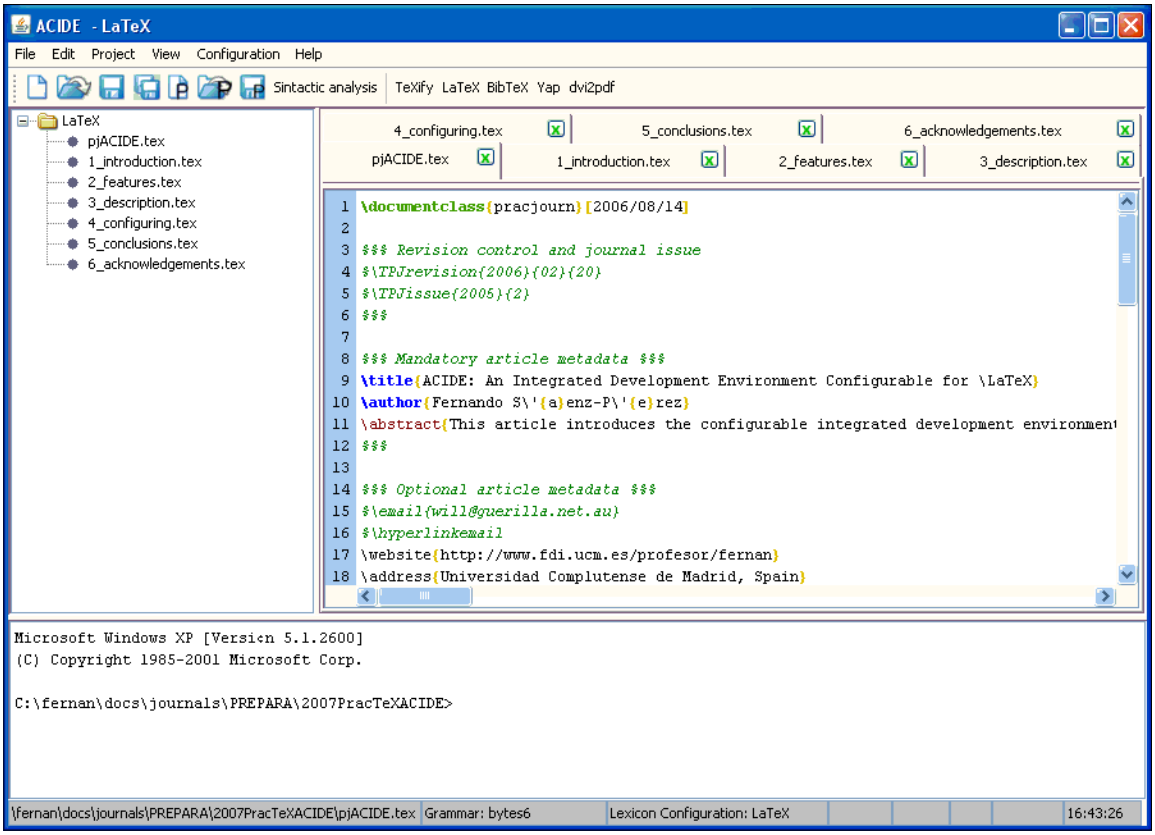

Figure 1: ACIDE Main GUI

project structure, Compile the specified files (see Section 3.5.3), Execute the result of the compilation, Set File as Compilable, Unset File, and Set as Main File). Missing useful options: Recent Projects.

– View. For showing/hiding project and shell panels, and displaying the log. Window arrangements are not possible up to now, but usual features are cascading and tiling windows both vertically and horizontally. Changing the order of the MDI windows (tabs) is not possible.

– Configuration. This entry allows to configure Lexicon (for syntax highlighting), Grammar (for parsing on-the-fly), Compiler (for compiling the project), Shell (the shell in the bottom panel), Language of the GUI, Menu entries that are displayed, and Toolbar for the commands, which can be displayed ei-

ther as icons or textual descriptions. Tooltips for toolbar commands can be configured.

- Help. This entry contains Show Help, and About ACIDE.

In addition, there is a fixed toolbar which includes common buttons for file and project-related basic operations: New, Open, Save, and Save All (this last one only for files). For development purposes, a button with label Syntactic analysis is included, and it is intended to test text parsing. Currently, a fixed grammar is included, but it will be specified by the user (see Section 3.5.2). Next to the fixed toolbar, there is the configurable toolbar (see Section 3.5.5).

Finally, the status bar gives information about some items: The complete path of the selected file, the selected grammar and lexicon, the line and column numbers, Caps Lock, Scroll Lock, Num Lock, and current time.

3.3 ACIDE System Variables

There are two system variables which are used for configuring commands in the toolbar, and also for configuring compiler options and the executable file:

- `$activeFile$`. Contains the complete file name of the current file (the one in the active MDI window).
- `$mainFile$`. Contains the complete file name of the main file (the one manually marked with Set as Main File).

In both cases, it is also useful to access only the name of the file or the extension, an issue to be addressed soon.

3.4 Projects

A project contains the whole status of a session, which is defined by all the possible configurations as well as the current display status. It consists of files arranged in folders (with any tree depth), all the configurations for the session (lexicon, grammar, compiler, shell, language, menu, and toolbar), main GUI arrangement (panel sizes, and opened files in the project), and file attributes. File attributes identify a file in a project as compilable and/or main. If a file is compilable, then the compiler configuration can be set to compile each of these files.

If a file is a main file (there is only one in the project), then it can be used in the compiler configuration or in the toolbar commands. For instance, a LaTeX project would compile the main file whether it is opened and active in the multifile panel or not. The filename of the main file can be accessed with the system variable `$mainFile$`.

The project structure shown in folders is a logical view which may coincide with the physical structure of the OS folders, but this is not needed. You can include in a given project a file belonging to another tree structure, therefore allowing to share files for different projects.

3.5 Configuration

Next, possible configurations are briefly described.

3.5.1 Lexicon

The lexicon for a programming or description language can be configured (as shown in Figure 2) for implementing a useful feature in text editors: syntax highlighting. Our tool allows defining the tokens and their associated format (colour, bold face and/or italics). Also, delimiters are needed to detect each token and these can also be declared. Line comments start with a given string; "%" in LaTeX, for instance. Also, a colour can be specified for comments.

3.5.2 Grammar

Both lexical categories and syntax rules can be defined to configure a grammar in EBNF (see Figure 3). Our tool allows the user to type or load those and create a parser, which is generated with the open-source, free project ANTLR [2] (ANother Tool for Language Recognition). With this tool the user will be able to activate parsing on-the-fly, a valuable aid in avoiding programming errors. This feature is still under development and it is expected to be completed by September 2007.

3.5.3 Compiler

A dialog box allows definition of the compiler to be used, its parameters, and the files to compile (see Figure 4). These files can be the files in the project that the

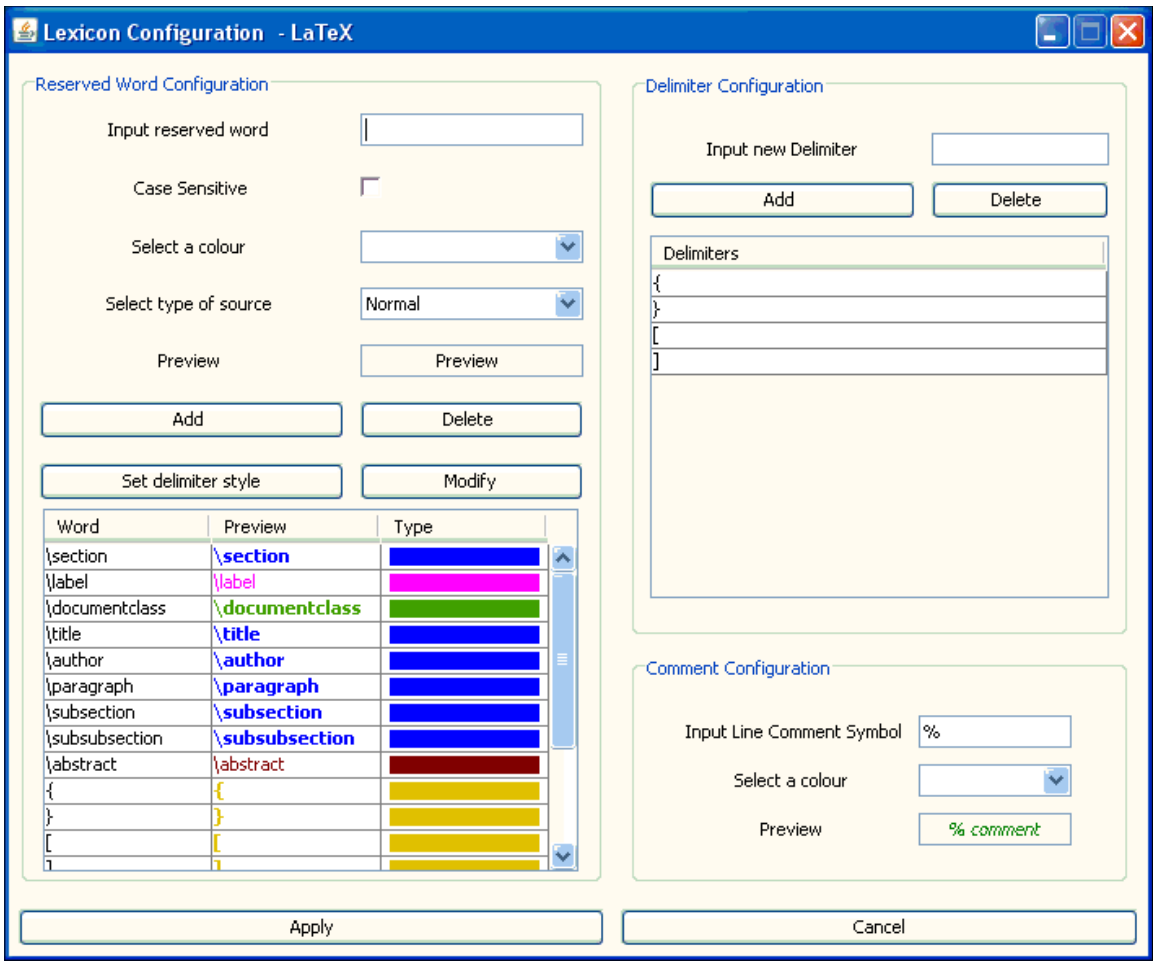

Figure 2: Lexicon Configuration

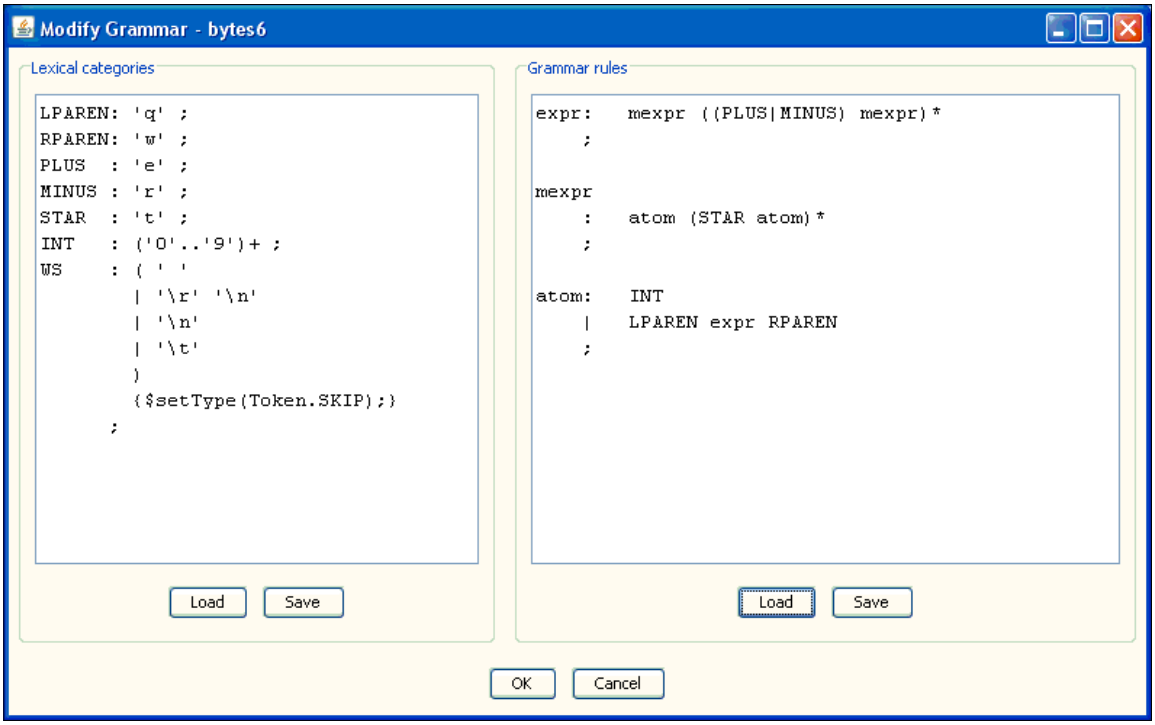

Figure 3: Grammar Configuration

user sets as compilable (a menu entry in the Project menu) or the files with a given extension (e.g., `.java`). Here, only the main file is compiled; there is no need to compile others. This dialog is useful for other programming environments in which several files must be compiled.

3.5.4 Executable

For programming projects that generate an executable, this dialog box allows the user to define the file's location as well as its arguments, if any (see Figure 5). The entry Execute in the menu Project will execute this file.

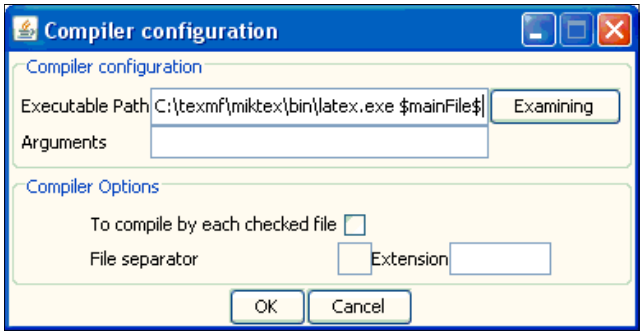

Figure 4: Compiler Configuration

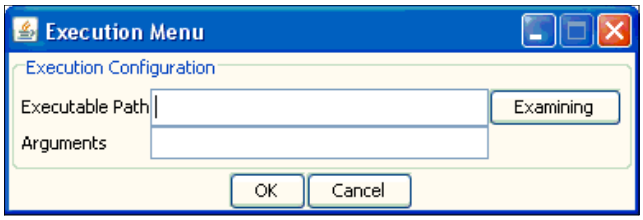

Figure 5: Executable Configuration

3.5.5 Toolbar

The toolbar can be configured by adding icons to the toolbar, which are associated with user-defined commands (see Figure 6). Currently, toolbar commands are sent to the console in the shell panel, but sending them as a separate process is also needed (a pending issue).

3.5.6 Menu

The menu bar can be configured by enabling or disabling predefined entries (see Figure 7). This configuration has to be extended to user-defined commands, as in the toolbar.

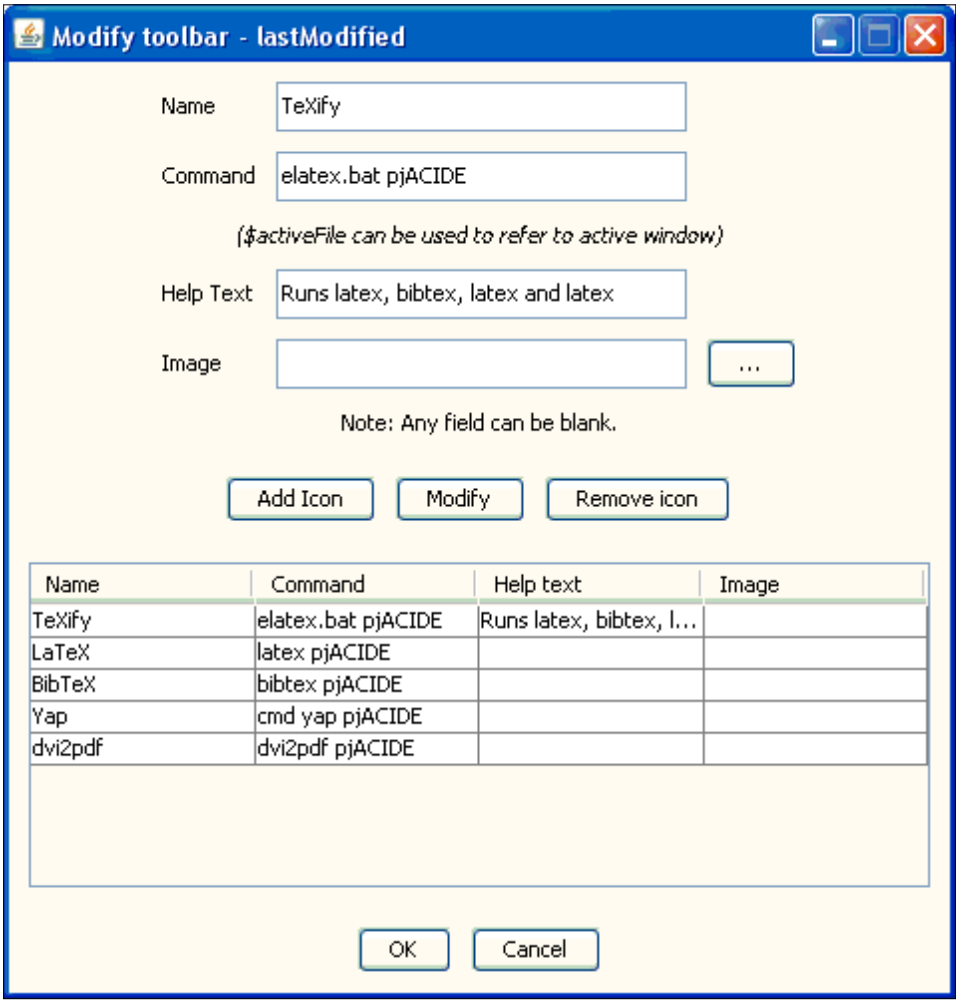

Figure 6: Toolbar Configuration

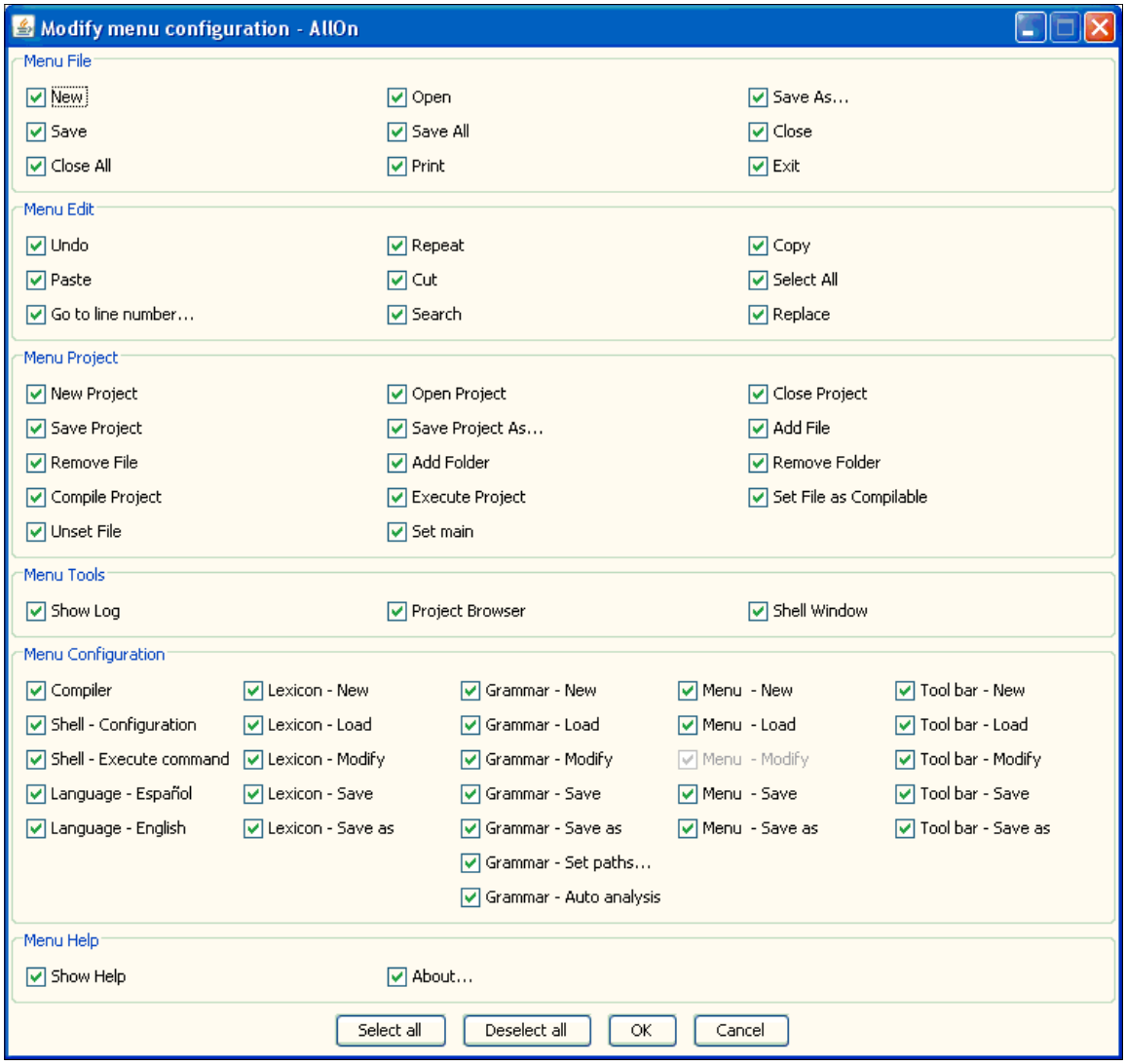

Figure 7: Menu Configuration

3.5.7 Shell

The shell can be configured with the dialog box seen in Figure 8. The check box is used for echoing the input command when the shell reads the input but does not output it.

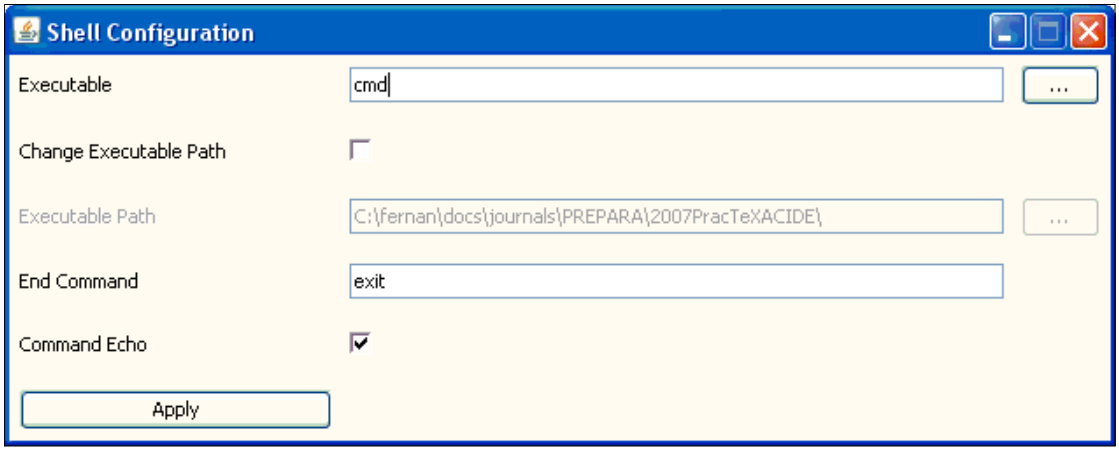

Figure 8: Shell Configuration

4 Configuring ACIDE to Work in a LaTeX Environment

Following the previous section, it becomes clear that ACIDE can be configured to work in a LaTeX/TeX environment. At least, in the current development stage, the following can be configured (actually, they are):

- Lexicon: For syntax highlighting as seen in Figures 1 and 2.
- Toolbar commands: TeXify (latex+bibtex+latex+latex), LaTeX (latex), BibTeX (bibtex), Yap, and dvi2pdf (dvi2pdf) among others (see Figures 1 and 6).
- Shell: For interacting with an operating system console (see Figures 1 and 8).
- Compile project: For compiling the main file (see Figure 4).

5 Conclusions and Future Work

This paper has presented ACIDE, an alternative to other available IDEs, with features meeting our initial requirements. It cannot be thought of as a complete tool since it is emerging and in an alpha development status. It should otherwise seen as a tool that, in time, might provide a steady stream of features following our requirements. For LaTeX/TeX users, there are much better tools in the free market today, but ACIDE might be competitive once it is more mature.

There are many goals yet to be completed. Some have been posed in Section 2 and others have been indicated in the following sections, while some have become evident after the tool description and configuration. Apart from fixing numerous bugs (including operating system dependent calls, which makes this tool work only for Windows, up to now), we have to implement state-of-the-art features in file editors, programming environments, and shell interactions. Despite the assumed limitations, we hope that this tool will consolidate into a friendly and powerful application, amenable to the users for whom it is designed; in particular, to system implementors, database users, document writers, and developers. It can be downloaded from `http://acide.sourceforge.net`. We hope to provide the version described in this paper as soon as possible, and to provide a better and more stable implementation by September 2007.

6 Acknowledgements

I am grateful to the students who developed this system under my guidance, which I hope will be useful for them as they develop their own projects in the future. These students are Diego Cardiel Freire, Juan José Ortiz Sánchez, and Delfín Rupérez Cañas, who worked on this project in the computer science course "Sistemas Informáticos" (Computing Systems) for graduates, a course intended for fifth-year students of the Computer Engineering studies of the Computer Science Faculty at the Universidad Complutense de Madrid, Spain. Also, thanks to the projects TIN2005-09207-C03-03, and S-0505/TIC0407 which supported this work. Finally, thanks to the free, open-source project ANTLR [2] which we use in our project, as well as to the Java and Eclipse initiatives. This article was prepared with MikTeX 2.4 [11], ACIDE [1] (cf. Figure 1), and BibMgr [4].

References

[1] ACIDE. A Configurable Integrated Development Environment. http://acide.sourceforge.net.

[2] ANother Tool for Language Recognition. http://www.antlr.org.

[3] Berlios. http://www.berlios.de.

[4] Bib Manager and Word Citer. http://bibmgr.sourceforge.net.

[5] Crimson Editor. http://www.crimsoneditor.com.

[6] Eclipse. http://www.eclipse.org.

[7] JBuilder. http://www.borland.com/jbuilder.

[8] JCreator. http://www.jcreator.com.

[9] JEdit. http://www.jedit.org.

[10] LaTeX Editor. http://www.latexeditor.org.

[11] MikTeX. http://www.miktex.org.

[12] Texmaker. http://www.xm1math.net/texmaker.

[13] TexNicCenter. http://www.toolscenter.org.

[14] WinEdt. http://www.winedt.com.

[15] WinShell. http://www.winshell.org.

[16] P. Arenas, A.J. Fernández, A. Gil, F.J. López-Fraguas, M. Rodríguez-Artalejo, and F. Sáenz-Pérez. $\mathcal{TOY}$. A Multiparadigm Declarative Language. Version 2.3.0, 2007. R. Caballero and J. Sánchez (Eds.), Available at http://toy.sourceforge.net.

[17] F. Sáenz-Pérez. Datalog Educational System User's Manual. Technical Report 139-04, Facultad de Informática, Universidad Complutense de Madrid, 2004. http://des.sourceforge.net.