Construction of a WYSIWYG LaTeX Typesetting System using Object-oriented Design*

J.H.M. Lee, J.C.K. Leung and C.C.K. Wong
Department of Computer Science
The Chinese University of Hong Kong
Shatin, Hong Kong

Abstract

This paper reports the design of a sophisticated WYSIWYG LaTeX-like typesetting system ViEwTeX. Our approach is based on the Model-View-Controller (MVC) user-interface paradigm, as originated from the Smalltalk community. According to the paradigm, the ViEwTeX system is decomposed into the sub-editor (model) module, the redisplay (view) module, and the user-commands (controller) module. Each module is responsible for a different aspect of ViEwTeX. Represented as separate objects, these modules co-operate by communicating via a controlled and structured pattern. Such decomposition facilitates encapsulation and code reuse. To demonstrate the feasibility of our approach, a prototype implementation of ViEwTeX is constructed according to the design using C and Motif.

Keyword Codes: D.1.5; D.2.10; I.7.2
Keywords: Object-oriented Programming; Design; Document Preparation

1 INTRODUCTION

LaTeX [6], a macros layer built on top of TeX [3], is a sophisticated typesetting system producing high quality mathematical manuscripts and monographs. LaTeX is command-based. Typesetting using LaTeX is no simpler than writing a computer program. This *user-unfriendliness* sets LaTeX back from gaining the acceptance of many potential users. On the other end of the spectrum, there are popular WYSIWYG wordprocessors, such as WordPerfect and Microsoft Word, widely available on personal computers. Output from these wordprocessors is not as aesthetic as that of LaTeX, both in terms of word and line spacing, but they are interactive in nature. Users can view and modify the layout of a document in real-time as it evolves. Wordprocessing is also no longer "programming."

To capture the best of both worlds, we propose ViEwTeX, an interactive WYSIWYG LaTeX wordprocessor. Its design is based on the Model-View-Controller (MVC) user-interface paradigm [5], which originates from the Smalltalk [2] community. According to the MVC paradigm, a user-interface application should be decomposed into a model, a view, and a controller. The *model* is the sub-part containing all of the non-visual non-interactive computer semantic information. The *view* contains all of the visual computer

*This project is supported by a UK/HK Joint Research Scheme grant (A/C 242700500) and a RGC Earmarked grant (A/C 221500260).

semantic information. The *controller* is responsible for capturing the user's attempts to manipulate the object. In this paper, we identify the model, the view, and the controller aspects of ViEwTEX. A reliable and efficient communication protocol among the modules is also proposed. To demonstrate the feasibility of our design, a prototype of ViEwTEX is constructed.

2 ViEwTEX DESIGN USING MVC

We adopt the terminology of [1] and call the model, view, and controller respectively *sub-editor*, *redisplay*, and *user-commands*. The sub-editor hides the details of how an edited text is stored from the rest of ViEwTEX. That includes a basic buffer for storing text, the current position of the cursor, and various markers to remember symbols, special format, and cross-references. The redisplay is responsible for ensuring that all changes to the buffer are promptly reflected on the display. The user-commands captures keystrokes and mouse clicks, and interprets these inputs before informing the sub-editor.

2.1 Overall architecture

The three modules work cooperatively in maintaining a user-interface application. Efficient and reliable communication is essential. A standard communication pattern of the MVC paradigm consists of four separate communications. First, the user performs an action, which is intercepted by the user-commands. Second, the user-commands interprets this action as a request to change the state of the sub-editor, and informs the sub-editor. Third, the sub-editor changes its state and then informs its dependent redisplay that it has changed. Fourth, the redisplay then asks the sub-editor for its current state and updates itself accordingly.

2.2 Sub-editor

In ViEwTEX, the text is stored into a one dimensional array of characters. Besides, the text buffer must also contain two gap pointers (gap-start and gap-end) in order to implement the paged buffer gap technique [1], allowing ease of text insertion. When a user prepares a document, we cannot know, in advance, the amount of work the user will do. Thus, the text buffer should be allocated dynamically. A doubly linked list for text buffers is a suitable choice. Apart from text buffer, mark, symbol and cross-reference information is also necessary in the implementation of the sub-editor.

From time to time, it is useful to be able to remember positions within a text buffer. A mark is an object that can remember position. There can be any number of marks within a text buffer, and more than one mark can remember the same position. In each mark, it contains a mark type (such as `begin` and `end` positions of the centering command) and a position in the text buffer. Each text buffer is associated with a linked list of mark which is maintained into a sorted order of the mark position.

Symbol mark and cross-reference mark are special kinds of mark, which are used to remember positions of mathematical symbols and cross-references. Each text buffer maintains also a sorted linked list of symbol marks and a sorted linked list of cross-reference marks.

2.3 Redisplay

The job of redisplay is to ensure that (1) the state of the sub-editor is displayed on the screen faithfully and (2) the amount of clock time required to make the updates visible is minimized. When the sub-editor has performed a function (e.g. insert character), the redisplay module is required to reflect the changes on the screen in a timely fashion. The redisplay module is also responsible for determining the layout of the text and diagrams on the screen.

The last responsibility is directly related to the *line-breaking* problem: how to divide long paragraphs into individual lines. Traditional algorithms, as adopted by popular wordprocessors such as Microsoft Word, WordPerfect and WordStar, use "*justification*," which considers only the current line and decides how best to finish off that line. The lack of line lookahead often results in awkward word spacing and hyphenation. LaTeX and TeX, using the *optimum* line-breaking algorithm by Knuth and Plass [4], consider a paragraph as a whole, so that the final appearance of a given line might be influenced by the text on succeeding lines.

The core of the redisplay module is Knuth and Plass's algorithm, the function of which is to find the sequence of breakpoints for the current paragraph. The result is stored in a linked-list. The redisplay will then display the content of buffer on screen according to the information stored in the linked-list. The problem of page-breaking is handled by the same algorithm.

2.4 User-commands

The job of the user-commands module is to capture user input, which can be text and commands. While text input must come from key presses, command input can be issued via *hot keys*, *hot buttons*, *pull-down menus*, and *mouse maneuver*. Thus the core of user-commands is a command loop that intercepts user input (via the keyboard and mouse), interprets the input, and informs the sub-editor of users' intention.

3 PROTOTYPE IMPLEMENTATION

The ViewTeX prototype is realized on a DECstation using the C language, OSF/Motif, Xt Intrinsics, and the Xlib library. The system consists of around 20,000 lines of C code. Features of ViewTeX can be classified into LaTeX functions and wordprocessor functions.

The main purpose of the current implementation is to verify the feasibility of the MVC design approach. Most but not all LaTeX functions are implemented. In particular, we have left out citation (*a la* BibTeX), floating bodies (*a la* tables and figures), mathematical formulas, and drawing commands, but they can be integrated into ViewTeX easily. Implemented command categories include sectioning, various symbols, cross-referencing, itemization and enumeration, document style, displayed material, etc. Worth mentioning is that sectioning numbers and reference numbers in cross-referencing are generated automatically by the system.

Wordprocessor functions of ViewTeX include file commands, copy/cut/paste, and help tools. The most common way to invoke wordprocessor functions is by means of pull-down menus controlled by mouse (or equivalent point-and-click devices). Hot buttons are provided for quick access to some most frequently used functions (both LaTeX and wordprocessor functions). Advanced users or fast typists, who despise the use of mouse, can make use of hot keys. Every function in ViewTeX has an associated hot key.

Edited documents can be saved in ViEwTeX format for future retrieval or exported to the LaTeX source file format. The latter option provides flexibilities for users to fine tune the edited documents where necessary, such as inserting TeX commands into the documents. ViEwTeX also supports auto-saving function to secure the integrity of an edited document. In case of a system crash, the edited document can be recovered from the incremental log saved periodically during the editing process.

4 CONCLUDING REMARKS

When designing a complex software system, it is essential to *decompose* it into small parts, each of which we may comprehend and refine independently. Design of a wordprocessor is no exception. In the current project, we have adopted the object-oriented decomposition approach, in which the world is viewed as a set of autonomous agents that collaborate to perform some high level behaviour. In a user-interface application, the collaborating agents are respectively the model, the view, and the controller. The contribution of this paper is three-fold. First, to the best of our knowledge, our work is the first to apply the MVC paradigm outside the Smalltalk-80 environment (which has many MVC required classes pre-defined) and find it practical to do so. The conscious choice of the C language, the most common language in the industry, is a further evidence of the feasibility of our approach. The MVC paradigm, and object-oriented design in general, facilitates encapsulation and code reuse. These properties are proved to be valuable during our implementation process. Second, our design is general enough to be guidelines for the design of other wordprocessors. Third, we have constructed a reliable and practical implementation of a WYSIWYG LaTeX wordprocessor, not yet available in the commercial market. While preparing a document using ViEwTeX, users can concentrate on designing the layout of the document instead of worrying about "programming" LaTeX commands correctly. Most parts of this manuscript were prepared using ViEwTeX.

REFERENCES

1. Finseth C.A., *The Craft of Text Editing: Emacs for the Modern World.* Springer-Verlag, 1991.

2. Goldberg A., *Smalltalk-80: The Interactive Programming Environment.* Addison-Wesley, 1984.

3. Knuth D.E., *The TeXbook.* Addison-Wesley, 1984.

4. Knuth D.E., Plass M.F., Breaking paragraphs into lines. *Software-Practice and Experience*, 11:1119–1184, 1981.

5. Krasner G.E., Pope S.T., A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

6. Lamport L., LaTeX*: A Document Preparation System.* Addison-Wesley, 1986.