ESA: A CLIM Library for Writing Emacs-Style Applications

Robert Strandh
LaBRI, Université Bordeaux 1
351, Cours de la Libération
33405 Talence Cedex
France
strandh@labri.fr

Troels Henriksen
DIKU, University of Copenhagen
Universitetsparken 1, Copenhagen
athas@sigkill.dk

David Murray
ADMurray Associates
10 Rue Carrier Belleuse, 75015 Paris
david.murray@admurrayassociates.com

Christophe Rhodes
Goldsmiths, University of London
New Cross Road, London, SE14 6NW
c.rhodes@gold.ac.uk

ABSTRACT

We describe ESA (for Emacs-Style Application), a library for writing applications with an Emacs look-and-feel within the Common Lisp Interface Manager. The ESA library takes advantage of the layered design of CLIM to provide a command loop that uses Emacs-style multi-keystroke command invocation. ESA supplies other functionality for writing such applications such as a minibuffer for invoking extended commands and for supplying command arguments, Emacs-style keyboard macros and numeric arguments, file and buffer management, and more. ESA is currently used in two major CLIM applications: the Climacs text editor (and the Drei text gadget integrated with the McCLIM implementation), and the Gsharp score editor. This paper describes the features provided by ESA, gives some detail about their implementation, and suggests avenues for further work.

1. INTRODUCTION

The Common Lisp Interface Manager (CLIM) [3] is specification of a substantial library for user interaction with Common Lisp applications. CLIM has a layered design, and defines a fairly large collection of interacting protocols that together make up the full library. Thanks to this layered approach, it is straightforward to add functionality to CLIM by defining new classes and new methods on existing generic functions: provided that they respect the defined protocols, such extensions will integrate seamlessly into the rest of CLIM. Customization of the standard behaviour of CLIM components is likewise straightforward, using standard CLOS means such as subclassing and auxiliary methods on protocol generic functions as in [1]. For more information on CLIM, see for example [2, 4, 5].

ESA is a library for CLIM that assists in writing Emacs-style applications. The central distinguishing feature of an Emacs-style application, for our purposes, is that it uses short sequences of keystrokes as the default method of invoking commands, and only occasionally requires the user to type the full name of the command in a *minibuffer*. A special keystroke (`M-x`) is used to invoke commands by name. This interaction style is substantially different from the one provided by CLIM by default, and therefore required us to write a different CLIM *top level*. Fortunately, CLIM was designed to make this not only possible but fairly straightforward, as the implementation of a replacement top level can build on the protocol layers beneath, just as the default top level is built.

The ESA library provides other features that are helpful in the creation of Emacs-style applications; these features need not be used, but are available if desired. A client application may choose to use an *info pane*[1] to display information about an associated master pane, such as the name of a *buffer* being edited, whether application state needs saving, or the position of the cursor in a buffer. Other optional components include a protocol for file input/output with versioning support; a framework for reading and writing buffers (with application-defined content) to external streams; and support for displaying command and keystroke documentation on-demand. However, we wish to emphasize that, for our purposes, the defining characteristic of an Emacs-style application is not its function (such as editing text files or text buffers) but its mode of interaction.

Two major applications already use the ESA library, namely Drei, the Emacs-like editor component distributed as part of McCLIM [7] (and its extension to a standalone application, Climacs [6]) and Gsharp, an editor for music scores. Thanks to the ESA library, the look and feel of these applications is similar, even though they manipulate very different objects. Other applications that use the ESA library have been written or are in the process of development, such as a directory editor, a mail client, and an info documentation browser.

ESA is developed and distributed as part of McCLIM[2], but

[1] a "mode line" in Emacs terminology.

[2] See `http://common-lisp.net/project/mcclim/` for instructions on obtaining McCLIM and ESA.

ILC'07, April 1–4 2007, Cambridge, UK
ACM 978-1-59593-618-9/07/04

has been run on other CLIM implementations in the past, and, to the authors' knowledge, it should still be possible to do so. ESA is designed as a portable layer that should run on top of any CLIM implementation, using only exported CLIM symbols, though this is unfortunately not always possible.

The rest of this paper gives more details about each of the components making up the ESA library: section 2 introduces the customized top level and command loop; section 3 discusses the support for keystroke handling using CLIM command tables, and describes the command tables available for use by client applications. Sections 4 and 5 describe the support for multiple windows and for buffer handling, respectively, following which we demonstrate an example Emacs-style application, and conclude with a discussion of the scope for further work.

2. TOP LEVEL

A key part of CLIM is the *top level* which executes a *command loop* that, in each iteration, acquires a command and any arguments that the command might need, and finally executes the command.

CLIM provides a *default top level* that prompts the user for a command to be executed. The user can satisfy this request by typing the name of a command (with completion) into an interactor pane; by selecting an entry in a menu; by issuing a gesture associated with the command; or by clicking on a presentation that has a presentation-to-command translator associated with it. Command arguments are acquired in a similar way. This default top level works fine when gestures are mouse gestures or single keystrokes using some modifier such as control or meta, but is less well adapted when commands are to be invoked by keystrokes associated with ordinary characters and when sequences of keystrokes should be used, as in Emacs.

When the CLIM function `run-frame-top-level` is invoked on an application frame, CLIM executes the top-level function associated with the frame. What top-level function is associated with a particular frame is determined by the `:top-level` option given to the standard CLIM macro `define-application-frame`. When no such initarg is given, CLIM uses the generic function `default-frame-top-level` as the top-level function for the frame. ESA applications must give the `:top-level` option with a value of `(esa-top-level)` in order to use the specific top-level function provided as part of the ESA library.

The ESA top level accumulates a sequence of keystrokes until that sequence is associated with a command. To determine whether that is the case, the top level searches a hierarchy of *command tables* (see section 3 for the details of this mechanism).

The ESA top level is also responsible for accumulating Emacs-style *numeric arguments* that are used to modify the behavior of commands. CLIM already contains a mechanism for calling commands with numeric arguments. Before a command is invoked, the value of the variable `*numeric-argument-marker*` is replaced in the list of arguments to a command by the numeric argument given by the user (the default is 1). ESA extends the CLIM mechanism by a second marker, `*numeric-argument-p*`, a Boolean value that indicates whether a numeric argument was given by the user at all. This extension allows ESA applications to tell the difference between the case where a numeric argument of 1 was explicitly given, and the case where no numeric argument was given.

Finally, the ESA top level contains support for executing Emacs-style *keyboard macros*, whereby a sequence of keystrokes is recorded for later playback. This mechanism is very hard to implement unless there is support in the top level for it, because special treatment is required for keystrokes that start and end the recording.

2.1 Command Arguments

The custom top-level function provided by ESA is integrated into CLIM in that it uses the CLIM function `accept` to acquire commands and arguments. The ESA library customizes the `stream-accept` generic function called by `accept` so that commands and arguments are prompted for in the ESA minibuffer.

The minibuffer's version of `stream-accept` calls the standard `prompt-for-accept` function, but then calls `accept-1-for-minibuffer`, an ESA-internal function, rather than the CLIM function `accept-1`. We must use our own function as we wish to turn input sensitization off, and `accept-1` does not allow this.

There is additional work required to support accepting commands for the `com-extended-command` command (invoked by `M-x`). ESA provides its own command-parser and partial-command-parser to integrate well with ESA's top level; however, there is no standard way to discover, given a command name, what arguments that command requires, and consequently the implementation of these command parsers are sensitive to the internal implementation details of the CLIM implementation.

3. COMMAND TABLES

CLIM command tables support both nesting and inheritance. Inheritance is used for code factoring as usual. An entry in a command table can be associated with a name in a menu or with a single keystroke, usually with some modifier key such as `control` or `meta`. The value of an entry can be either a command to be executed, or another command table for nesting. Multiple keystrokes such as required by Emacs-style applications are not supported, however.

The Emacs-style interaction mode assumes that the vast majority of all commands are to be invoked by sequences of keystrokes that may be ordinary characters, and that on the rare occasion that a command is needed that is not associated with a sequence of keystrokes, a special keystroke (`M-x`) must be used as a prefix. To provide such a style of interaction, ESA uses CLIM command-table nesting with a level of nesting for each single keystroke in the sequence used to invoke a command. The ESA function `set-key` takes a CLIM command, a command-table, and a sequence of keystrokes to be used to access the command in the command table. It follows entries in the command table for each keystroke in the sequence and builds nested command tables as necessary. Typically, an ESA application will also use the

```
(in-package #:esa)

(defclass example-info-pane (info-pane) ()
  (:default-initargs :height 20 :max-height 20 :min-height 20
                     :display-function 'display-info :incremental-redisplay t))

(defun display-info (frame pane)
  (declare (ignore frame))
  (let ((master (master-pane pane)))
    (format pane "Pane name: ~S; Accumulator: ~D" (pane-name master) (counter master))))

(defclass example-minibuffer-pane (minibuffer-pane) ()
  (:default-initargs :height 20 :max-height 20 :min-height 20))

(defclass example-pane (esa-pane-mixin application-pane)
  ((contents :initform "hello" :accessor contents)
   (counter :initform 0 :accessor counter)))

(defmethod (setf contents) :after (new-contents (pane example-pane))
  (setf (pane-needs-redisplay pane) t))

(define-application-frame example (esa-frame-mixin standard-application-frame) ()
  (:panes
   (window (let* ((my-pane (make-pane 'example-pane :width 600 :height 200 :display-function 'display-my-pane
                                      :command-table 'global-example-table :name "Example Pane"))
                  (my-info-pane (make-pane 'example-info-pane :master-pane my-pane :width 600)))
             (setf (windows *application-frame*) (list my-pane))
             (vertically () (scrolling () my-pane) my-info-pane)))
   (minibuffer (make-pane 'example-minibuffer-pane :width 100)))
  (:layouts (default (vertically (:scroll-bars nil) window minibuffer)))
  (:top-level (esa-top-level)))

(defun display-my-pane (frame pane)
  (declare (ignore frame))
  (with-text-size (pane 70) (princ (contents pane) *standard-output*)))

(define-command-table global-example-table :inherit-from (global-esa-table keyboard-macro-table help-table))

(define-command (com-goodbye :name t :command-table global-example-table)
    ((count 'integer :prompt "How often"))
  "Say \"goodbye\""
  (setf (contents (current-window)) (format nil "goodbye~[ x0~;~:; x~:*~D~]" count))
  (incf (counter (current-window)) count))
(set-key `(com-goodbye ,*numeric-argument-marker*) 'global-example-table '((#\c :control) #\g))

(define-command (com-hello :name t :command-table global-example-table)
    ((count 'integer :prompt "How often"))
  "Say \"hello\""
  (setf (contents (current-window)) (format nil "hello~[ x0~;~:; x~:*~D~]" count))
  (incf (counter (current-window)) count))
(set-key `(com-hello ,*numeric-argument-marker*) 'global-example-table '((#\c :control) #\h))

(defun example (&key (width 600) (height 200))
  "Starts up the example application"
  (let ((frame (make-application-frame 'example :width width :height height)))
    (run-frame-top-level frame)))
```

Figure 1: Code for the example application of section 6.

CLIM macro `define-command` with the keyword argument `:command-table` to add the command to the same command table that was used with `set-key`. This way, using a particular command table, the command can be invoked either with the associated keystroke sequence or as an extended command using `M-x`.

To allow for modular applications, the ESA library supplies several different CLIM command tables that may be used by applications through the use of the command-table inheritance mechanism provided by CLIM. The table `global-esa-table` contains commands for quitting the application (`C-x C-c`), and for invoking an extended command in the minibuffer (`M-x`). The `help-table` contains useful help commands such as `describe-key-briefly` (`C-h c`), `where-is` (`C-h w`), `describe-key`, `describe-command`, and `apropos`. The `keyboard-macro-table` contains commands for defining and executing keyboard macros with the usual Emacs key bindings. Finally, the `esa-io-table` contains useful commands for file input/output, such as `find-file` (`C-x C-f`), `find-file-read-only` (`C-x C-r`), `save-buffer` (`C-x C-s`), `write-buffer` (`C-x C-w`), and for toggling the buffer read-only flag (`C-x C-q`).

To allow for applications that might have different contexts in which different sets of commands and keystrokes may be applicable, the ESA library provides a generic function `find-applicable-command-table` to be applied to an application frame. The default method returns the command table of the current window. More complex applications can choose to define methods on this generic function that return specific command tables depending on the current context, such as the position of an application-specific cursor.

4. MULTIPLE WINDOWS

A typical Emacs-style application can contain an arbitrary number of windows for displaying application data. The class `esa-frame-mixin` provides a slot that contains a list of such windows. Applications that need to manipulate an arbitrary number of such windows typically define a CLIM application frame that inherits from this class. By convention, the first window in this list is the current window, i.e. the one to which keystrokes are to be delivered. The windows of application may inherit from the `esa-pane-mixin` class that provides a default command table for the pane.

4.1 Help Streams

In addition to supporting multiple application windows, the ESA library provides for an application to display documentation about its commands and keystrokes, in a manner similar to help invoked by `C-h` in Emacs. The default location for this help to be displayed is in a window stream created with the CLIM function `open-window-stream`, though this is customizeable by the application using the `help-stream` generic function (see figure 4 for an example of this).

However, the default implementation suffers from the fact that there is no standard way within CLIM to share input buffers between multiple frame-manager windows: there is standardized support for an `:input-buffer` initarg to extended input streams, but no standard means for acquiring such a buffer. The ESA implementation uses internals, from Silica (the windowing substrate for one CLIM implementation) or McCLIM, to get a handle on the input buffer of the parent application to pass as the value for the `:input-buffer` initarg.

5. BUFFERS AND FILES

Applications that use the ESA library may choose to use the `esa-buffer` abstraction. This abstraction is useful for applications that load the contents of a file into some arbitrarily complex buffer data structure. This abstraction provides functionality such as generic functions for creating empty buffers, for creating a buffer by reading from a stream, for saving the contents of a buffer to a stream, and for associating the name of a buffer (which is then typically displayed in an info pane) with a file name that is used to fetch and store its contents. Other convenient functionality is provided by this abstraction such as the file name associated with the buffer, whether the buffer needs saving, and more.

The ESA library supplies and uses a generic function called `buffers` that is applied to an application frame, but does not supply any default method on it. Applications are free to store an explicit list of buffers in each application frame, or to use any other method such as storing buffers in CLIM views, and only explicitly storing the views.

The `esa-io` package can be used by applications for input/output between buffers and files. Naturally, applications must define methods for reading and writing application-specific data, but the package supplies useful commonalities such as commands for loading files and saving buffers (that call application-specific routines), functionality for checking whether a buffer needs saving, and support for file versioning.

6. EXAMPLE APPLICATION

The ESA library provides an example application that can be used as a skeleton for real applications. It is a fully-functional application but with a very simple buffer structure. The example can be found at the end of this paper. It contains a single application pane with an associated info pane, and a minibuffer pane as usual. This very simple application has a single slot in the application frame itself that contains the application-specific data to be manipulated, so does not need multiple buffers or multiple windows.

In this application, the `Hello` and `Goodbye` commands accept an integer argument that, if invoked by keystrokes, is given by the numeric argument if any (or 1, if no numeric argument is provided); the application defines its own command table that inherits from `global-esa-table`, `keyboard-macro-table` and `help-table`, giving this simple application extended command processing, keyboard macros and online documentation respectively.

7. FUTURE WORK

We intend to supply complete documentation of the ESA library (for partial documentation, see appendix A), and to include a more complex example application.

One immediate prospect for future work on the library is to provide a means for applications to dynamically specify the contents of the application menu-pane in a similar manner to

the way in which they may currently specify the applicable command table. At present, the menu pane's contents are static (though submenus are dynamically generated), and so it is not currently possible to achieve mode-specific menu entries.

As we find more commonalities between Emacs-style CLIM applications, we plan to include them into the ESA library, probably as separate packages to allow for applications the flexibility of using these commonalities, providing their own equivalent functionality, or not using them at all.

8. REFERENCES

[1] S. E. Keene. *Object-Oriented Programming in Common Lisp*. Addison-Wesley, 1988. ISBN 0-201-17589-4.

[2] S. McKay. CLIM: The Common Lisp Interface Manager. *Communications of the ACM*, 34(9):58–59, 1991.

[3] S. McKay and W. York. Common Lisp Interface Manager Specification, 1994. `http://bauhh.dyndns.org:8000/clim-spec/`.

[4] R. Möller. User Interface Management Systems: The CLIM Perspective. online, 1998.

[5] R. Rao, W. M. York, and D. Doughty. A guided tour of the Common Lisp interface manager. *ACM SIGPLAN Lisp Pointers*, 4(1):17–37, 1990. Updated 2006 by Clemens Frühwirth.

[6] C. Rhodes, R. Strandh, and B. Mastenbrook. Syntax Analysis in the Climacs Text Editor. In *International Lisp Conference Proceedings*, 2005.

[7] R. Strandh and T. Moore. A Free Implementation of CLIM. In *International Lisp Conference Proceedings*, 2002.

APPENDIX

A. API REFERENCE

In this appendix, we briefly summarize the facilities provided by the ESA library that are most useful for producing an Emacs-style application. There are other hooks for customization in the full library.

Function `esa-top-level` *frame* `&key` *command-parser command-unparser partial-command-parser prompt*

This function is suitable to pass as the first element of the form argument to the `:top-level` option in `define-application-frame`. The *prompt* argument is used as the value for `*extended-command-prompt*`.

Special Variable `*numeric-argument-marker*`

Special Variable `*numeric-argument-p*`

The values of these variables can be included in *partial commands* given to `set-key`. Before the command is invoked, those values will be replaced by the numeric argument given (default 1) and a Boolean value indicating whether an explicit numeric argument was given, respectively.

Function `set-key` *command table gestures*

This function is used to associate a command with a sequence of keystrokes in a particular command table. Nested command tables will be created as necessary.

Command `com-extended-command`

This command is associated with the `M-x` key. Its role is to acquire (through the use of the CLIM `accept` function) the name of a command and the arguments to that command, and then execute that command.

Special Variable `*extended-command-prompt*`

This variable contains the string used to prompt the user for an extended command. The default value is `"Extended Command: "`, and can be altered by using the `:prompt` keyword argument to `esa-top-level`.

Command Table `global-esa-table`

Command Table `help-table`

Command Table `keyboard-macro-table`

Command Table `esa-io-table`

These command tables contain commands (and keyboard sequences for invoking them) for features that typical ESA applications might want to include.

The `global-esa-table` contains the `com-quit` command (bound to `C-x C-c`) and the `com-extended-command` command (bound to `M-x`).

The `help-table` contains the `com-describe-briefly` command (`C-h c`) which prompts for a key and shows the command it invokes, the `com-where-is` command which prompts for a command name, and shows the key that invokes it (`C-h w`), the `com-describe-bindings` command which gives a list of commands and associated keystrokes in the currently applicable command table (`C-h b`), the `com-describe-key` command that displays documentation for the command invoked by a keystroke sequence (`C-h k`), the `com-describe-command` that gives documentation for a command (`C-h f`), and the `com-apropos` command which shows commands with names that match a list of given words (`C-h a`)

The `keyboard-macro-table` contains the traditional Emacs-style keyboard macro commands `com-start-kbd-macro` (`C-x (`), `com-end-kbd-macro` (`C-x )`), and to invoke the macro the `com-call-last-kbd-macro` (`C-x e`).

The `esa-io-table` contains the commands `com-find-file` (`C-x C-f`), `com-find-file-read-only` (`C-x C-r`), a command `com-read-only` which toggles the read-only flag of a buffer (`C-x C-q`), `com-set-visited-file-name` which prompts for a new file name for the current buffer (no associated keystroke sequence), `com-save-buffer` (`C-x C-s`), and `com-write-buffer` (which prompts for a file name and writes the buffer to that file) (`C-x C-w`).

Generic Function `find-applicable-command-table` *frame*

This generic function returns the currently applicable command table for an application frame. A default method on this generic function is supplied that return the command table associated with the current window. Client code typically specializes this function to return different command tables in different contexts.

Generic Function `help-stream` *frame title*

This generic function opens a help stream for the frame with the title given. It is called by help commands that require substantial output to display help information.

Standard Class `esa-frame-mixin`

Standard Class `esa-pane-mixin`

The `esa-frame-mixin` class is used by client code to mix into application frames for Emacs-style applications. It provides a slot that stores a list of the windows that are used by the application (accessed using the `windows` generic function). It also provides other useful functionality such as a method on `find-applicable-command-table` that returns the command table of the current window.

The `esa-pane-mixin` class is used by client code to mix into application panes of an Emacs-style application. It provides a command table for the pane.

Standard Class `info-pane`

Standard Class `minibuffer-pane`

The `info-pane` class is an application pane that can be used to display information of another pane (called the *master pane*). It accepts the initarg `:master-pane` and the `master-pane` generic function can be used to access the master pane of an info pane. This generic function is typically used in the *display function* for the info pane to access information about the master pane.

The `minibuffer-pane` class is an application pane that contains a *message* to be displayed to the user.

Generic Function `buffers` *frame*

This generic function returns a list of all the buffers of the application.

Generic Function `windows` *frame*

This generic function returns a list of all the windows of the application. The first element of this list is taken to be the current window, on whose behalf the top-level loop should process keystrokes and execute commands.

B. SCREENSHOTS

In the following figures, we present screenshots of various applications using the ESA library. A snapshot of the example implementation is presented in figure 2, showing the application itself and a help window. Figure 3 shows the Gsharp score editor[3], in a state where it is accepting a filename for input. Figure 4 shows the Climacs editor[4] in Lisp mode, also demonstrating a customization of the help system (displaying to a pane inline with the application, rather than the default standalone window-stream). Figure 5 displays a mail client, currently under development.

[3] `http://common-lisp.net/project/gsharp/`
[4] `http://common-lisp.net/project/climacs/`

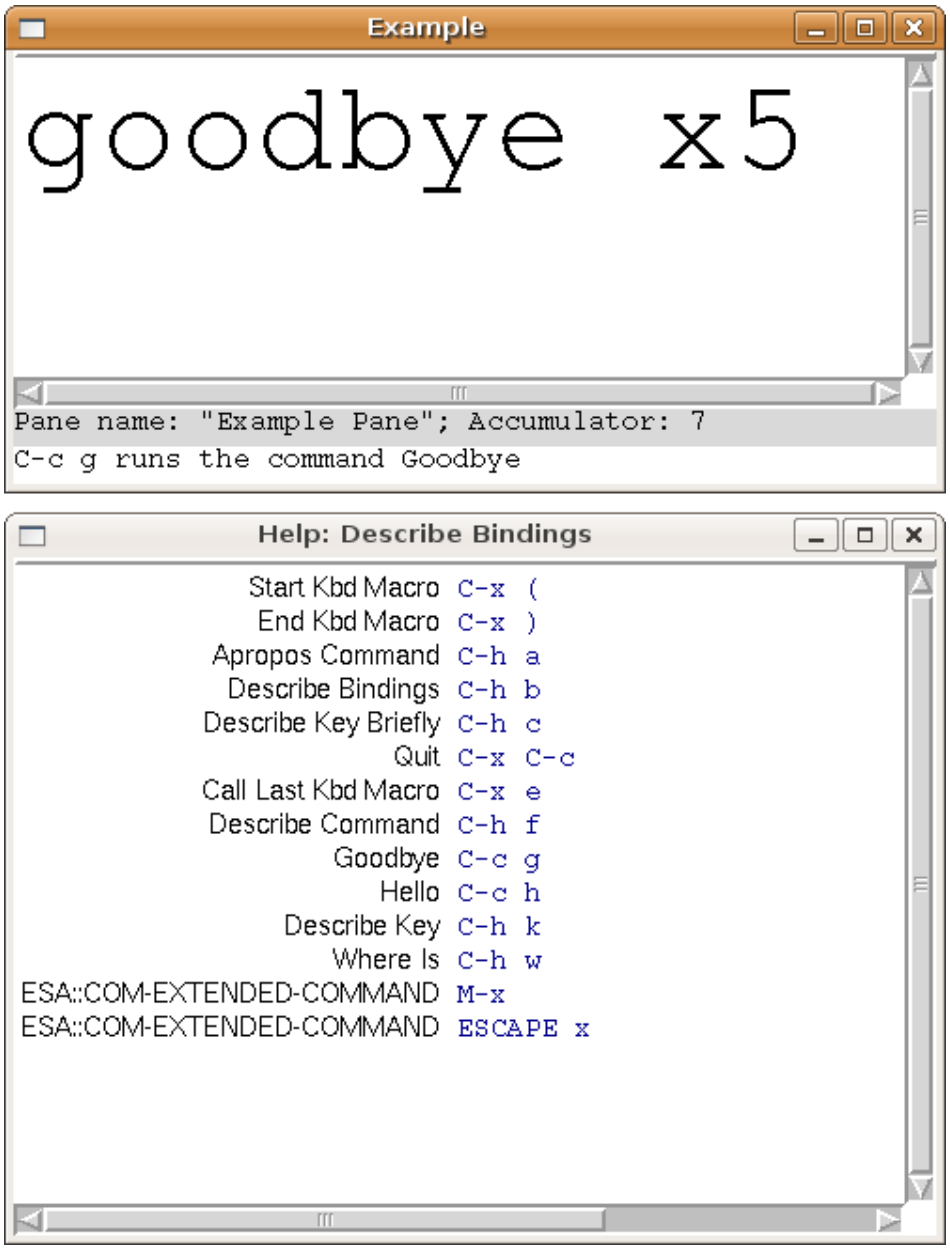

Figure 2: The example application of figure 1, with a help window showing commands and their associated keybindings. The panes in the main Example window are, from top to bottom, the example, info and minibuffer panes; the minibuffer shows the result of `C-h c C-c g`, describing the command which would be run by the key sequence `C-c g`; the last application command was performed with `C-u 5 C-c g`, running the `Goodbye` command with the argument `5`.

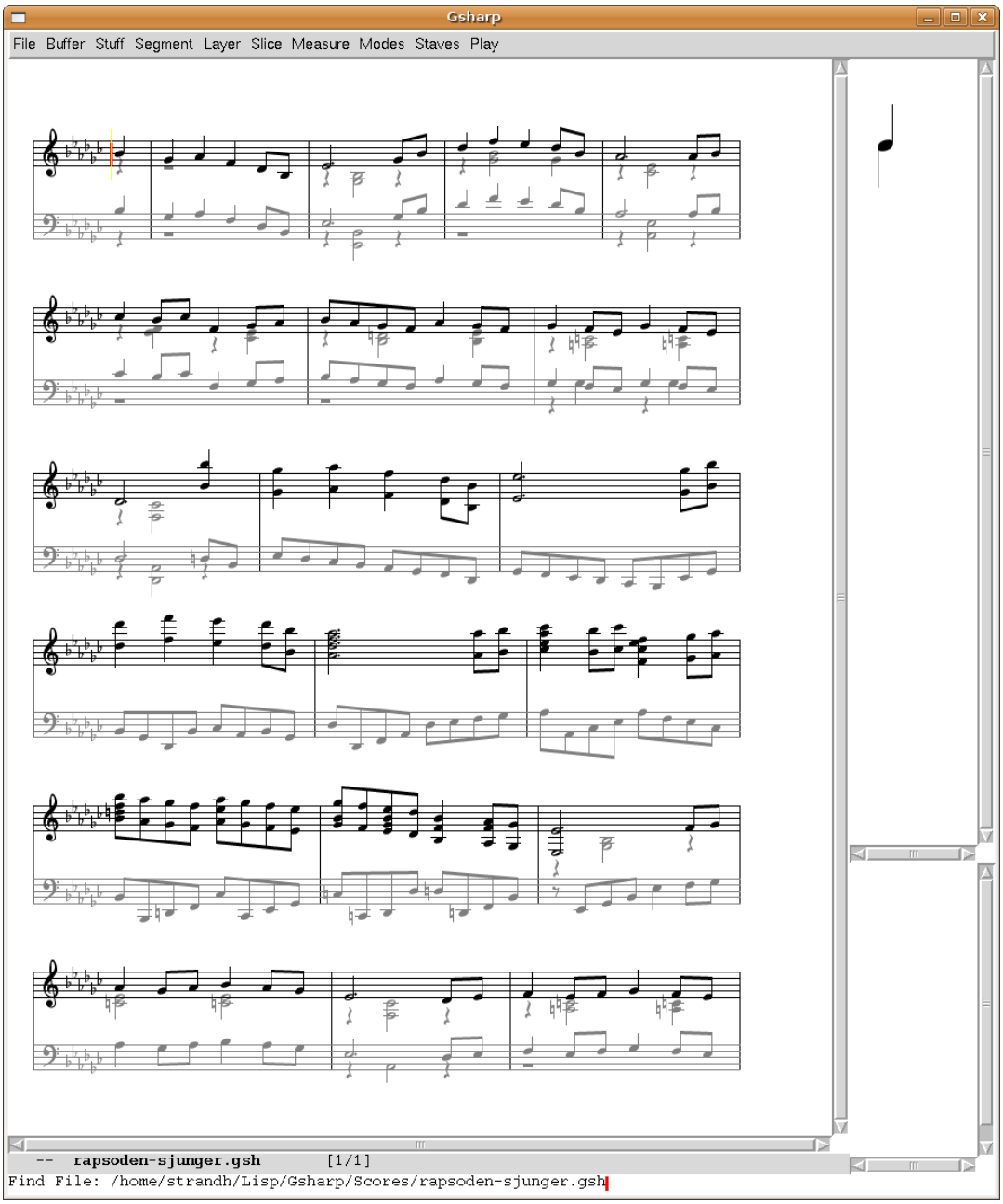

Figure 3: The Gsharp score editor. When the buffer is in melody mode, individual keystrokes move the cursor and insert or remove notes, accidentals, key signatures, and other elements of musical notation. When the buffer is in lyrics mode, the cursor movement keys remain the same but the movement commands act on lyrics rather than notes, as do the editing and insertion commands. This difference in behaviour is mediated using methods on `find-applicable-command-table`.

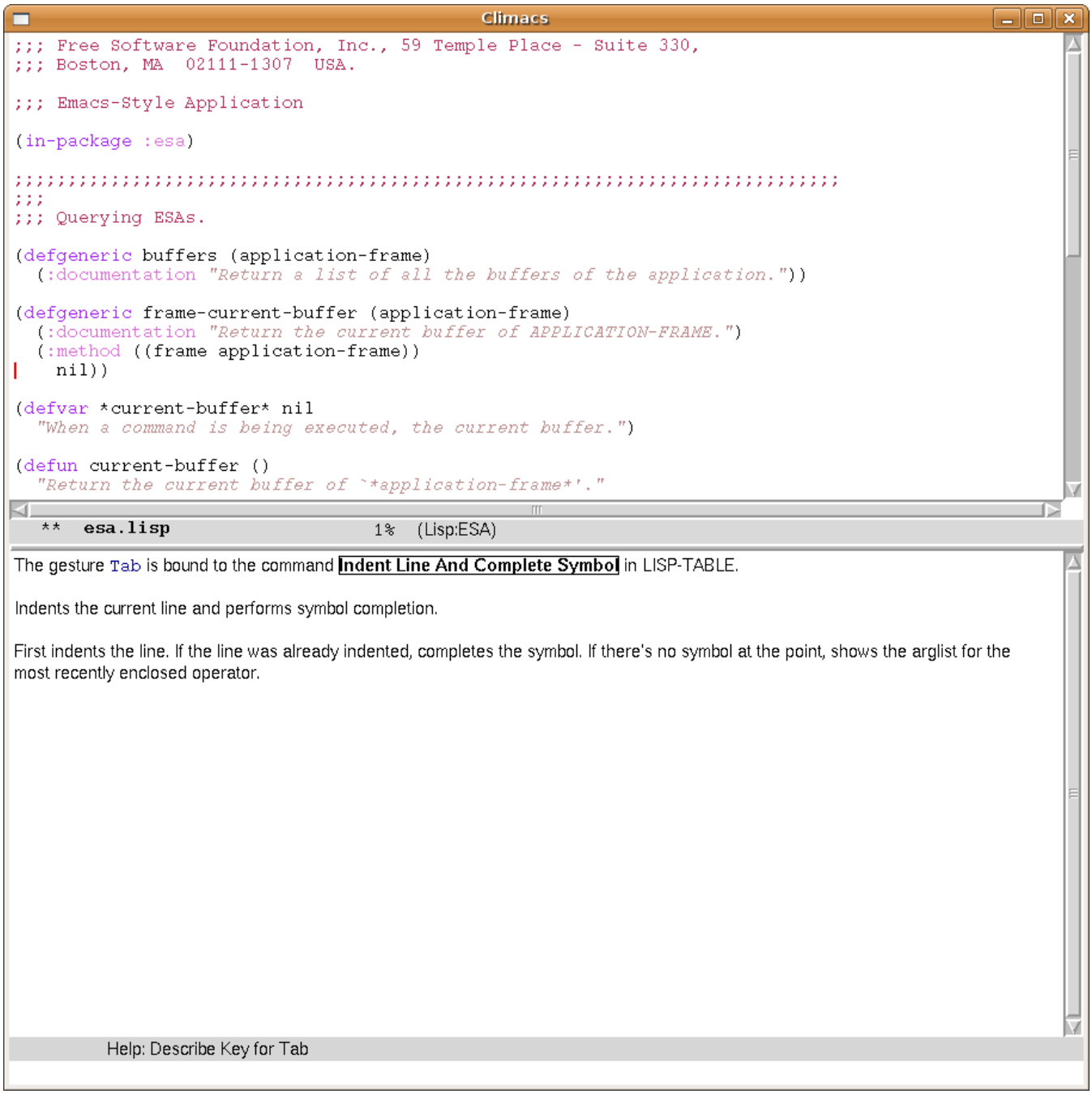

Figure 4: The Climacs text editor, with help display. As one might expect of a Emacs-Style text editor, the user's interaction with the application is strongly reminiscent of other Emacsen. Note the highlighting around the command name in the help text; this is a presentation of a command object, which is acceptable as a command; also, climacs performs its own management of help streams, unlike the example application displayed in figure 2.

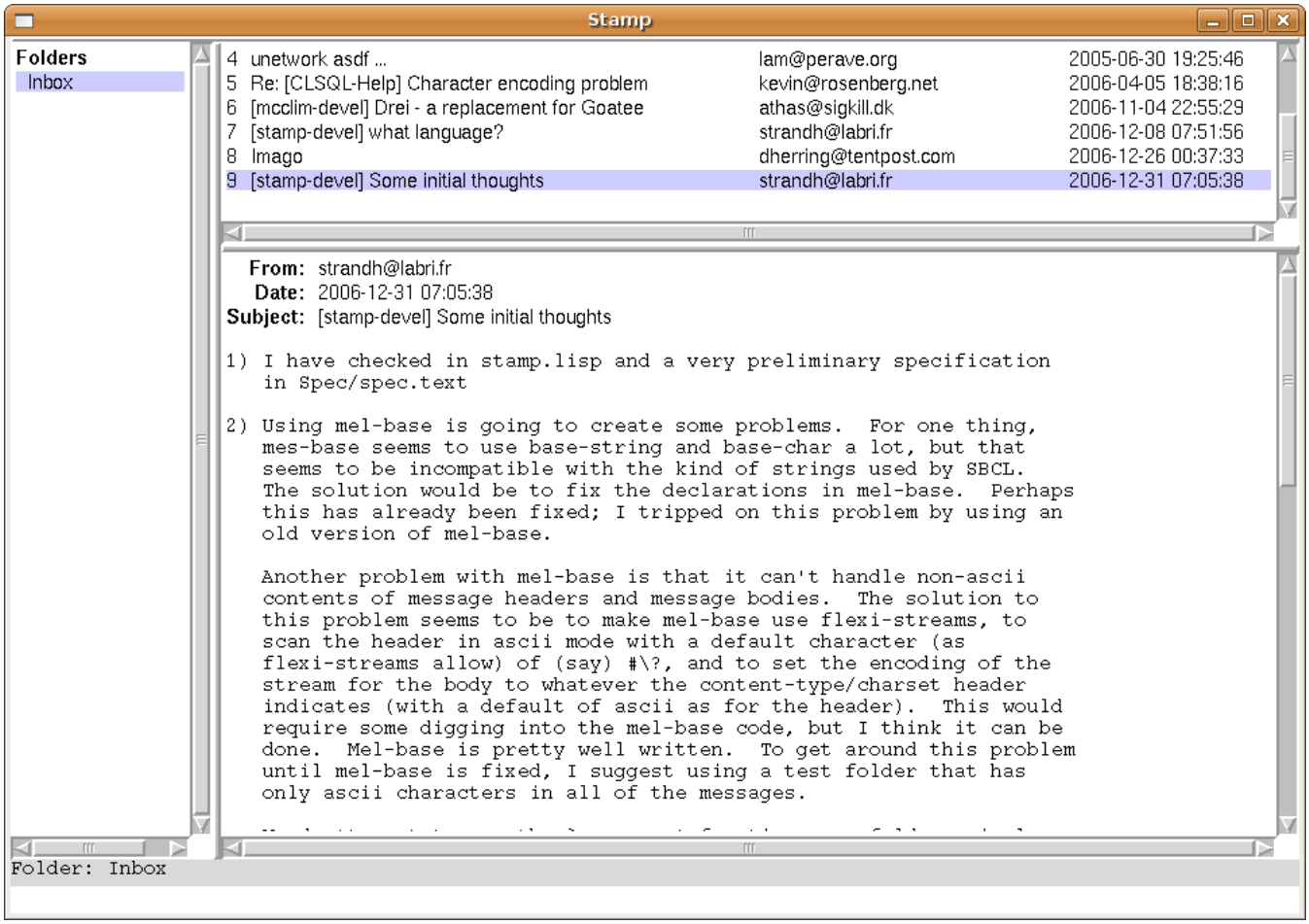

Figure 5: The Stamp mail user agent, using the info and minibuffer pane facilities provided by ESA.