Abstract

This paper discusses aspects of the design of a graphical interface for the LUCY program. Also included are a list of preliminary specifications for most of the functions in the current LUCY program.

Design of a LUCY Interface

by UChin Kim

Thesis Supervisor: Bruce R. Lewis

Contents

List of Figures

Design of a LUCY Interface

1 Design Considerations

1.1 Global Considerations

1.1.1 Interface

The most obvious decomposition of LUCY is into two separate interfaces. One interface is the user interface, used for asking questions, browsing through the public browser, etc. The other interface is for the answerers, which we will, for now, call the reply interface.

Advantages of this dichotomous breakdown of the LUCY interface include: 1. We can limit access to the reply interface and hence access to private material. 2. We can limit the user's access by limiting the user interface. 3. The user's interface is not cluttered with useless functions.

I can think of no disadvantages to implementing two separate interfaces. It wouldn't necessarily be more work because there will be much reuse of existing concepts and structures between the two modules.

1.1.2 CLI Interface

While menus are easier to use initially, some users will indubitably prefer a command line interface, especially the person or persons representing LUCY. Ideally, the user should be able to interchange mouse clicks with keyboard commands. This is a nice feature to have, but not a necessary one. Time constraints will affect whether this feature will be included.

1.1.3 Help Function

Help should provide exactly the help information that the user needs. This suggests that a browser type of interface might be useful. The user could browse through a hierarchy of help windows to locate the specific help that he needs. This is reminiscent of the Emacs Info program, which allows the user to browse through on-line documentation through a series of menu selections that specify what section of the documentation the user is interested in. The user can quickly find out the information he needs to know.

1.1.4 Error Trapping

In an article on user interfaces printed a few years ago (I don't remember where), the writer suggested that every error message should be composed of three parts. The first part tells the user what went wrong. The second part tells the user why it went wrong. And the third part gives the user suggestions on how to fix it or what to do next.

This is a very useful decomposition of an error message, because much of the problem of error messages is that they are rarely oriented toward the typical user who isn't interested in what a "Segmentation error" is, but is interested in how to get his program working again. Programmers tend to program error messages that assume a lot of related and crucial knowledge on the part of the user, that the user often doesn't have, and doesn't care to have.

1.1.5 Menu Organization

One basic strategy involves limiting the number of menus to about seven available menu selections at any given time. Having significantly more than seven menu selections results in the user spending much of his time scanning through the list of menus.

Menu buttons should be easy to understand. Otherwise, we defeat one of the purposes of using menus. The menus will be arranged in a hierarchy. Due to the resources feature of Motif, we can defer the exact arrangement of the menus until runtime. It should be relatively straightforward to experiment with different menu layouts to find a reasonably optimal layout.

1.2 Specific Considerations

1.2.1 Browser

Ideally, the public browser would be able to display questions arranged according to topics and subtopics. Users could then focus on the questions they are interested in, instead of having to browse sequentially through all the questions.

However, if the number of question-answer pairs in the public browsery never grows too large (less than 100 question-answer pairs), the sequential method would be preferable in terms of simplicity of implementation.

Also, multiple files could be set up so that each file contained question-answer pairs corresponding to one topic. Users could view the files they were interested in. If the number of question-answer pairs under a given topic grew too big (greater than 200 question-answer pairs), the topic could be split up into two sub-topics (e.g. questions under the topic SEX could be split up into two sub-topics: CASUAL_SEX and INTIMATE_SEX) Sequential browsing would be adequate for using this method. What needs to be determined is methods for splitting up topics into sub-topic as well as combining topics.

For the hierarchical browser method, we could use one large file or alternatively multiple files to keep track of the question-answer pairs, plus topics and sub-topics. A person using the reply interface could categorize a question by choosing from the list of current topics, or by specifying a new topic if appropriate. Also, the user could specify the topic for his question.

Regarding using the browser, the user could specify first a topic and then, if necessary, specify a sub-topic to narrow in on a range of questions. The hierarchical browser would require some way to keep track of topics and sub-topics. Possibly, we could have a special file(s) to keep track of topics and sub-topics as well as their related filenames.

1.2.2 Editor

The editor interface should be able to emulate any of several different editors, such as: Emacs, vi, EZ. In practicality, I will probably incorporate an Emacs-like interface for the editor. Appropriate help screen(s) will guide users not familiar with Emacs.

The editor should have two distinct sections: one section for displaying and editing the question, and one section for displaying and editing the answer. Users would only be able to edit the question. Repliers would be able to edit both fields.

1.2.3 Fields

Some parts of LUCY require answers to a sequence of questions. Default answers should be provided in a field environment, such that the user has the option of accepting the default responses to all questions or editing selected parts of the field. This has the significant advantage of keeping user responses to a minimum. It can be tedious for a user to answer a set of questions every time he wants to ask a question.

1.3 Other Considerations

1.3.1 User Anonymity

One way to handle user anonymity would be to incorporate a dummy name feature into LUCY[1]. If a user wants a personal response from LUCY, he can specify a dummy name that he can later use to obtain LUCY's answer to his question. Metaphorically, the user is specifying a key by which he can later

[1]Michael Mermelstein '94 suggested this idea to me.

access the answer to his question. No other user can access this answer without the key. This allows LUCY to respond to anonymous users who want personal relies.

When the user has retrieved the personal reply via his keyword, LUCY would then be able to delete the question-answer pair from its private queue.

Possible problems with this include the following: the user might forget his keyword, users might have trouble using the feature, another user might be able to guess a user's keyword and thereby obtain an illicit question-answer pair (and the genuine user would never get his reply).

To handle the case of a user forgetting his keyword (and thereby be unable to access LUCY's reply), we could either have a provision that keeps a limit on the maximum number of question-answer pairs in the private queue, so that if the queue is already at maximum size, we discard the oldest question-answer pair. Or we could decide to discard all question-answer pairs in the private queue that have been in the private queue for longer than n days. Possibly, instead of discarding private question-answer pairs that have been unaccessed, we could post them in the public browsery so that the user might be able to get his reply after all. Anonymity would be preserved in any case.

Users that are not familiar with the safetybox metaphor as used in this approach might have initial difficulty with it. Help windows should provide necessary help in this respect.

To guard against rogue users that try to guess keywords and thereby access a question-answer pair that they don't own, we could set a limit on the maximum number of failed guesses that LUCY will allow before exiting to the system. Of course the rogue user can always restart the LUCY program again, but this wastes the rogue user's time so that he may consider not worth the trouble. In any case, the likelihood of this happening is relatively small. If this turns out to be a problem, we can always program additional safeguards into the program.

1.3.2 Dialup Access

For LUCY to work via dialup access, LUCY must use pure text (no graphics). The simplest way is to have two versions of LUCY, one version to use X Windows, and the other version to use a terminal-based approach. This design will, of course, be the X Windows version.

1.3.3 Forwarding

When LUCY forwards a problem to an expert, she must arrange for a way to get a reply back from the expert. The question should be sent by e-mail, since a given expert is more likely to be familiar with e-mail than with LUCY. The straightforward way to handle forwarding is to do it manually, i.e. mark that a question has been forwarded, as well as marking that a question has been successfully answered.

1.3.4 Motif

My major reason for using the Motif is that I am familiar and comfortable with it. Motif provides the necessary tools I need to build the LUCY interface.

2 Implementation Strategy

My basic implementation strategy is to implement the interfaces separately from the LUCY program. Then, after I get the interfaces working, I can add parts of the existing LUCY code incrementally until it is complete.

I will have to make sure initially that my design will allow me to glue the functional parts of the LUCY code (as opposed to the interface parts) with my interface. This will involve examining the existing source code to determine how the current interface and functional parts are currently composed.

In regard to implementing the interface separately, I will also have to write appropriate stubs and drivers in conjunction with the interface. But this should be a simple matter.

The advantage of implementing the interface separately is that it will allow me to ensure that whatever bugs arise will be due to my interface code and not due to the functional code. A remote danger is that, after I write the interface, I may find it cumbersome to fit the two sections (interface and functional) together. But I should be able to avert this problem by studying the existing design in detail.

3 Preliminary Specifications

Name	Show
Function	Displays question and information related to it.
Inputs	A way to select the question
Notes	Simple interface with limited input requirements. Display procedure and editor seem to have an abstraction in common.

Name	Archive
Function	Archive a question or set of questions
Inputs	Select a question or set of questions
Notes	limited input requirements, Add cancel and help buttons.

Name	Edit (Comment)
Function	Lucy can edit a QA pair so that it can be published. Or add comments, or reply to a question.
Inputs	Keychars, edit keys, end-of-input del.
Notes	Emacs like functionality as a default. User should be able to specify either vi, Emacs, EZ, if possible.

Name	Browse
Function	Allows user to browse through public QA pairs.
Inputs	Topic Selection and sub-topic selection. Question selection, DisplayQuestion button.
Notes	Meaty command. Will require considerable overhead and design considerations. Possibly, add either a print feature or save feature. Save would be better because user could later modify it to add LaTeX or Scribe formatting commands before printing. Possibly, can use browser also for LUCY private browsing.

Name	List
Function	Lists the questions in the queue and their statuses.
Inputs	Some way to page through extensive question lists. Or navigate hierarchical waters.
Notes	Perhaps also add selection mechanism like UNIX 'ls' does – e.g. ls -t sorts entries by time of last modification.

Name	Reply
Function	Reply to a user.
Inputs	e-mail address input end-of-input
Notes	Buttons should include: Help, Save, Cancel which are pretty self-explanatory.

Name	Ask
Function	Allows user to enter a question and send it to LUCY. Also allows some misc. options.
Inputs	Text of question, end-of-input misc: topic heading, personal reply query, publish boolean
Notes	Miscellaneous Inputs are a pain to specify. Perhaps have defaults so user doesn't have to go through the rigamarole. Use fields.

Name	Publish/Post
Function	Lucy can make a QA pair public
Inputs	Select QA pair, Select topic or sub-topic
Notes	Also a do-it button to effect the operation.

Name	Quit
Function	Performs necessary cleanup before shutting LUCY down.
Inputs	quit button.
Notes	Confirm quit if some things are in limbo.

Name	Help
Function	Provides advice to the user on how to proceed. Tells the user about the system. Provides useful information that user can quickly access.
Inputs	Information filtering inputs – vague.
Notes	Browser might be a useful basis for the help. Help should be an independent window so user can refer to it while using LUCY.

Name	Forward
Function	forward question to an expert.
Inputs	A way to select the question. e-mail address of expert.
Notes	Limited interface.