Chapter 4

Olive—an enhanced logging facility

Introduction

In the previous chapter, the need for a sophisticated transaction logging facility was identified. During the course of the project, such a facility was developed over two stages—the 'basic' and 'enhanced' logging facility. As the latter facility includes the former, only the enhanced will be described.

The configuration for this purpose was a PC acting as front-end to the OPAC, but independent of it. The basic conception was of a general-purpose diagnostic tool, potentially at least usable with other OPACs or retrieval systems. Section 4.1 is devoted to this general conception, treated quite independantly of specific systems. It describes an ideal; inevitably, for reasons associated with limitations on resource and technological constraints, the systems eventually produced fell some way short of this ideal. Section 4.2 discusses how the idea was implemented with particular reference to the CLSI's CL-CAT. Section 4.3 describes the system itself and how it may be used.

4.1 Idea of an interception front-end

4.1.1 Background: users, hosts, and front-ends

The simplest form of 'front-end' system is a terminal emulation program. A user working on a PC will call up such a program in order to talk to a remote host. Once the program is running, the PC is to all intents and purposes transparent: a dialogue takes place between the user and the host without intervention by the front-end.

Most terminal emulation programs also allow the user to interrupt the dialogue, holding the channel to the host open while conducting an exchange with the front end. The purpose of this exchange might be to instruct the front-end to log the user-host dialogue (as a simple form of downloading), or to send a pre-recorded sequence to the host.

More sophisticated front-end systems may be translucent or opaque. Thus the user may instruct the front-end to establish contact with the host and interrogate it, driven by commands from the user, but without direct dialogue between the user and the host. Such a set-up may for example occur when the front-end translates from the command language to another, so that the user does not have to know the native language of the host. Complete opacity would occur if the user did not know which host was being interrogated, or even whether such a dialogue was taking place.

An alternative approach is described by the client-server model, where the functionality of the whole system is distributed between the local and remote systems—for example the local system handling all the display functions as directed by the remote system. Using this model, with a particular application design, a user would perceive the local system as providing channels of communications to the various functional parts of the application regardless of whether the processes supporting the functions ran on the local or remote system. The user can switch between windows, but when using a particular window, is normally in effect conversing directly with the processes, programs or systems associated with the window.

4.1.2 Interception front-end

The model proposed here is different again. We envisage a front-end system which in the first place operates like a terminal emulation programming with a full-screen logging facility. (Such a system could equally well be implemented as a application within the framework of a client-server model). However, we also envisage the front-end observing the user-host dialogue, and having the ability (on certain pre-defined cues) to interrupt this dialogue and take its own intitiative. In other words, rather than simply mediating in a dialogue between the user and the host, the front-end itself is an agent in a three-way exchange.

In the context of the present project, we see such interceptions as a vehicle for asking the users questions relating to their searching activity on the host (i.e. the OPAC). In other words, we wish to insert specific, highly targetted online questionnaires into the user-host dialogue. Each questionnaire would relate to a specific situation, and would not be seen by a user who did not reach that situation. Questions could then refer to and elucidate that particular situation.

Such a tool could be of use more generally in the evaluation of interactive systems and in the investigations of human-computer interaction. Further, one could imagine its use to help the user in a more direct way, by providing very context-specific help.

4.1.3 Analysis of the interception function

The idea of the front-end observing the user-host dialogue and occasionally interupting it begs questions about the conditions under which an interruption would occur. There are two separate questions here:

- At what point(s) in the dialogue would an interception take place;
- What information about the history of the search should be used to inform the interruption.

For example, one may be concerned with why some users repeat various searching activities without examining specific records, or why they switch from one type of search to another. The first question relates to when it would be appropriate to interrupt the user, and the second is concerned with identifying whether this particular instance is one of the type required.

In a screen-based menu-driven system with some form of hierarchical structure, it seems likely that appropriate interception points would occur at—or when the user returns to—the higher levels. For example, it would be inappropriate to interrupt a user as they move from examining a record to requesting information about its availability; but it might well be appropriate to interrupt if the user does not take that step, but returns to a higher level instead.

When a possible interception point is identified, however, the front-end would have to go back over the history of the search, to establish whether it demonstrates the sought pattern. The front-end must therefore keep a record of the search as it proceeds, and at specified stages (interception points) must compare the history with a template or pattern for the required type.

One major problem with such a system is the choice of level of abstraction or summarisation inherent in the history recording. At one extreme, the system could simply record every character, and leave to the pattern matching operation the necessary summarisation. However, the character level seems very far removed from the level at which one would identify the pattern; it therefore seems appropriate to look for a higher degree of summarisation in the history recording. Again, if we are looking at screen-based menu-driven systems, an obvious level is the screen itself—we would look to record the identity of, and perhaps some information extracted from, each screen visited.

If the system is to be used for online questionnaires, then the questions and answers themselves should also form part of the history. In other words, the questions to be asked at any stage may depend on the answers to previous questions.

4.1.4 Function of the enhanced logging facility

Apart from the interception function discussed above, several other functions were required of Olive, as indicated in the previous chapter.

- First, the system had to log the user-host dialogue. In its raw form, the log had to include all screen displays, not just the user commands. This also implied a means of printing the log, even where it included screen-control codes.

- Second, there had to be a condensing/summarising facility for the log. Full-screen logs are extremely voluminous, but for analysis purposes we required a terser account.

- Third, we needed to be able to play back the log onto the screen, so as to elicit further information from the user. We required this playback to be simulated real-time (i.e. each steps to take the same amount of time as it had done originally). This requires some form of time-stamp in the log.

- Fourth, as well as intercepted questionnaires, we needed to be able to include pre- and post-search questionnaires.

How Olive served these functions is described below. Possible additional facilities were discussed in the course of the project, but were not include in Olive for reasons of lack of time or resources and the need to obtain a stable system with which to collect data. These include some aspects of the interception points and conditions, discussed below, and also the following. As regards the user interface, it would have been better to have had some form of pop-up window, so that when the front-end intercepted the dialogue, it would be very clear to the user that this was indeed an interruption, and the real dialogue would eventually be resumed. Secondly, the playback system would be improved with some kind of speed control and rewind facilities.

4.2 Interception and CL-CAT

As discussed above, the implementation of the interception front-end requires an analysis of

- appropriate interception points, and
- the 'history of search' logging and associated pattern matching.

4.2.1 Search structure in CL-CAT

The menu system in CL-CAT guides the user's access to and view of the catalogue. As with many online catalogues the menu protects the user from the underlying system command language. For the user the catalogue exits as the CL-CAT menu structure and the terminal upon which information is displayed.

CL-CAT system view of interaction

From the viewpoint of the CL-CAT system the menu and terminal is merely a stream of characters fired along a serial or network line. The system deals with transactions across many terminals asynchronously. For a given number of terminals in use the system is designed to give a response time sufficient to maintain the user's illusion of an interactive search. There is no high level or logical model of the user interface built into the CL-CAT system.

The stream of characters sent out by CL-CAT consists of instructions to format text on a particular type of terminal and the text itself. The text is either menu options and instructions or information from the catalogue resulting from a search. Input from the user at a terminal is restricted to command sequences of characters—associated with function keys—and to strings or 'words' to search for in the catalogue.

User's view of interaction

On sitting down at a terminal a user is faced with an opening menu of six search options—numbered one to six—or else with a terminal left by the last user in the middle of a search. The initial menu options are 1. Subject, 2. Author, 3. Title, 4. Author-Title, 5. Keyword, 6. Shelf-mark. These may represent different ways or modes of searching to the user. A terminal includes a qwerty keyboard for text input and up to 12 labelled function keys to issue commands to the system. Some of the function keys allow the user to navigate between menu screens (e.g. refine search), some to qualify the kind of searching with a particular mode (e.g browse an alphabetic list or look for words anywhere in a part of a record) and others allow the user to page back and forth through the lists displayed from the catalogue. One function key, the restart key, always takes the users back to the opening menu.

Menu selection takes the form of selecting a number perhaps being prompted to type in text—e.g. authors last name and first initial—and also being prompted to press a particular terminal key after entering text. For instance, selecting 4 gives author-title prompts and a message to press return after having typed in first the author and then a few title words. Other options require the user to select a number, enter some text and then select a particular terminal key to press according to whether they wish to 'Browse' or 'Search'. The distinction is not obvious but implies browsing an alphabetic list—e.g. of titles—or searching for keywords anywhere in a particular set of fields and receiving a count of the number of hits (records whose fields contain strings that match the words typed in).

The course of a user's session at a catalogue can be charted as a progression through a series of screens in selecting menu options and entering text and search commands. An example of a real session is charted here:

```
          start menu

chain 1        AU / HB / BR / CA / HB / BR / CA
                  restart
chain 2        AU / HB / BR / HB
                  restart
chain 3        SU / HB / BR

          end of session
```

The screens recorded are author (AU), heading browse (HB), brief record (BR), copy availability (CA). At the end of chain 1 the user pressed the restart key (go to the opening menu command) and started another author search. Chain 2 ended at heading browse and the user restarted, and in chain 3 selected subject from the start menu. Notice the user left the catalogue without restarting so that the new user would find the catalogue at the brief record screen.

It is possible to produce an exaustive map of all screens and paths to screens for the CL-CAT system. This exercise, of itself, is not very informative as to what users actually do when faced with many options. Some options—e.g. shelfmark—were used very infrequently. During the course of the project the library service discontinued the subject option in the opening menu of the catalogue and 'keyword'—option 5—was re-describe as 'keyword or subject'. In fact users could browse—look through an alphabetic listing—on any option, including 'keyword' by selecting the option and pressing the browse function key.

The menu structure of CL-CAT is best regarded as a means to hide the underlying search command language—qualifying searches by field labels and using boolean operators to specify a target set of sought items—and also as being based on grouping the elements within the structure of a catalogue entry (e.g title proper, parallel title, other title information as derived from MARC and AACR2 et al.). None of this relates, in a direct or obvious way, to how users think of what they are doing in searching.

For instance a user may select 2—Title option—then type in a few words and press the search function key to do a keyword search on the title fields. This is quite a good way to find some items with the keywords in the title and may be regarded as searching on a topic by the user. The system copes with this quite well since it reduces potential hits by limiting fields looked at and gives the user a greater chance of getting a small set to browse through. This may be a successful strategy in terms of outcome for the user, i.e. a good experience. Is this use of the title menu option an intentional feature of the system design or is it a feature discovered by users through finding out what get results?

It is interesting to note that many failures to find specific items both on title and title/author searches occur when users attempt to type in a *whole* title exactly as given in a reference—not just keywords from the title. The system carries out implicit 'ANDing' on all words typed in. This means a search for an item by typing in the title is very sensitive to any typing error. The length of title increases the chance of typing errors. As a result users experience fruitless searching of booklists because of typing errors. Depending on the user's competence in typing and checking miskeyings, the obvious strategy of citing a whole title as a way of finding a particular item may not in practice be found to get results, i.e. a bad experience.

The map of CL-CAT screens and the paths between screens does not map the users view of negotiating a search.

4.2.2 Conditions for questions at Interception points

Appendix A sets out the points of interception, and kinds of conditions giving rise to interceptions, which were discussed as being potentially useful within the project.

In fact only a subset of these were implemented in Olive. This was both due to project software engineering limits and the constraints of producing a stable system with which to gather data. In the event extensive debugging time was required to generate a stable, useful system.

Partly as result of software engineering constraints the interception point was limited to the user pressing the restart key. The conditions giving rise to questions was limited to the sequence of screens visited between restarts—to a single chain[1]. Olive covered conditions of:

- occurrence of a particular screen
- occurrence of a fixed sequence of screens
- occurrence of two screens with any other screens in between

[1]See Appendix A for explanation of a 'chain'

- the ending of a chain of screens on a particular screen.

Out of all the possible conditions that may be used to trigger a given question only a very few were required by any given experiment. But actually deciding which conditions and questions were suited to the purpose of a given experiment was not an obvious matter and required trial and error with real users. This process is complicated by the fact that the user's answer to a question—say a multiple choice question—formed a condition for the decision as to which question to ask next. Everything required testing in the actual library situation with real users before each experiment could be run. It must be borne in mind that in a real library setting the catalogue system is not always operational and users come and go with the flow of day (e.g. heavy lunch time use) and period of the year (e.g. start of the term or just before exams).

4.2.3 History and pattern-matching

The mechanism for specifying which conditions would give rise to a given set of questions within Olive is a very general one and highly reconfigurable by the experimenter without reference to programming. This is absolutely necessary where a series of experiments are being carried out in an actual library setting. Corrections and setup need to be done on the spot if unforeseen problems are to be coped with.

The questions to be asked and the conditions prompting questions are set out in text files, the question file and the conditions file. These are created and modified by the experimenter using a text editor.
The format for a conditions file is relatively simple. Each line of the conditions file consists of:

- condition of screens visited
- condition of answer to previous question
- identification of question to ask if conditions are true
- code for the question type.

For example:

```
C X 7 A
C 7D 1 Y
C 1Y 2 Y
C 1N 3 A
C 1N 6 T
C 2N 8 A
C 2Y 3 A
C 2Y 4 A
C 4B 5 A
C 8B 5 A
```

This condition file deals with the condition of having visited copy availability screen and prompts for questions from the question file give below. Line one says Copy availability (code C) then independent of answers to other questions (code X) ask question seven which is a multiple choice question (code A). The next line also has the same condition (code C) but question one (a yes/no question) is only asked if the answer to question 7 is D. Each line is processed in turn, thus controlling the order in which questions are asked.

The first two lines of the question file shown in figure 1, make up the text header which appears on top of the screen each time a user is asked a question. In the case of the example above the header reads 'Olive questionnaire' then on the next line 'Catalogue evaluation project'. The text header includes format instructions—Esc[1:36m—to highlight the header on top of the display screen. Each question occupies the whole screen so obscuring the catalogue screen which is effectively underneath the question the user is being asked. The user on entering a valid answer to the question either gets the next question, if the conditions file gives rise to one, or else is returned to the catalogue to observe it restarting at the opening menu as the user had requested before being interrupted by Olive.
The questions themselves are of three kinds according to the type of answer which is valid:

- Boolean or yes/no questions (code A conditions file)
- Multiple choice, i.e. A or B or C or D (code D in conditions file)
- Text questions, requiring free text input (code T)

4.3 Olive system description

The Olive system consists of two stand alone programs. The major program —named major.exe—was written under contract for the project by staff of the Systems Science Department of City University. The second program called quest.exe was created by the research assistant to the project for the purpose of administering pre- and post-search online questionnaires.

4.3.1 How to use major

On being called major displays the following menu:

```
                Welcome To the OPAC System

          A.      Start a Session
          B.      Print a Session
          C.      Replay a Session
          D.      Initialize Start of Day.

               Enter Selection, (Enter to Quit System):
```

These options are designed to be used by the experimenter. On pressing the key 'A' the catalogue opening menu appears on the display and the PC functions exactly like any catalogue terminal of the VT100 type. When the system is running library patrons do not distinguish the the PC from dumb terminals if the main unit of the PC is mounted of the terminal desk (e.g. on the floor). All the user sees is the visual display unit and the keyboard. If the experimenter wishes to distinguish use by each individual patron then a 'call major menu' key is pressed once the patron has left the catalogue. This saves the session in a log file and next session can be set up by selecting menu option A. Each session is stored in a separate log file and sessions greater than one hour have been successfully logged. The terminal can be left to run unattended where there is no need to log individual patrons' use of the catalogue.

The sessions were logged anonymously with the following text displayed as a notice next to the terminal.

Olive

When you use this terminal Olive—Online Interactive validation and evaluation—may appear on the screen to ask you a few questions. Olive will keep an anonymous record of your search to help us with the experiment to evaluate the library catalogue.

4.3.2 The structure of the major program

A front-end system which takes the place of a dumb terminal has to extract information about the dialog between user and system from the low level character stream sent to the terminal. Although for the user the system is screen based the actual screens are only identifed by format character sequences to clear the screen and home the cursor. There is no logical system screen object. With the City University CL-CAT system each screen could be identified by extracting the text from the character stream which represented the top line of each screen following a clear screen and home cursor instruction. This works because each screen has its own identifying heading on the top line of the screen.

Major employs the communication library supplied by Zortech Ltd. This enables an IBM AT PC to act as a terminal and receive the stream of characters directed to the serial port by the CL-CAT system.
Major consist of the following functional parts each written in the C programming language.

- major.c—main module giving menu of options to call program actions. Covers starting and logging a session, printing a session to printer, replaying a session. All keyboard processing including recognition of CL-CAT function keys. Functions needed to open and set up serial port called from the communications library.

- vtemu.c—module to parse format effect characters from input stream effectively providing for the display of text characters on the terminal screen in the intended place.

- repl.c—module to parse characters from the log file to simulate (replay) the display of a particular logged session on the terminal in real time.

- prtemu.c—module (with shtlog.c) to format a text file for dispatch to a printer. Also produces a summary of each session at the head of each print out.

- old.c—module to intercept the user and prompt with question from the questions file and get valid answer to store. The number of question asked and answer given is logged and reproduced in the printed version of the session (prtemu.c above). This module matches the pattern found in the conditions files against a record of the screens passed through in an actual session, using a mechanism similar to regular expression matching found in many UNIX[2] utilities.

4.3.3 Problems in writing and testing major

There were many problems in ensuring the link to the CL-CAT system. The bought in communications library included source code but testing for possible causes of problems was not helped by the lack of adequate documentation for the source. In the course of the project an entirely new library computer was installed to run the CL-CAT system. The major program after teething problems provided a reliable link to the new system. The nature of the conditional interception—asking users questions if conditions were matched—required extensive testing. Faults were found which occurred sporadically. These had to be corrected before use in actual experiments, and caused delay in the data collection stages. The best circumstance for testing was observation of sessions with actual library patrons to see if the correct interception and questions were asked.

Some elements of the program design were inadequate. For instance printing logs of schedules (the main analysis was from the printed log) was very time consuming since major had to be instructed to print each log one at a time. A stand alone print program which processed a whole day's sessions would have been far more effective.

4.3.4 How to use quest

Quest was developed as a stand alone program to present online questionnaires to users rather than make major any more complex. It is useful for questionnaires administered before the start or after the end of a search.

On being called, quest displays questions from a text file (quest.txt), one a at time, in a window on the top halve of the screen. A message is displayed in a separate window prompting the user for an answer. Three types of question can be put to user:

- T free text question

[2]UNIX is a trademark of AT&T Bell Laboratories

- Y yes or no questions
- A multiple choice question (A or B or C or D).

In the case of text questions the user is prompted with an editing window into which they may type and edit the text describing their answer. Quest goes through the questions in order indicated in quest.txt and exits after the last questions. A maximum of 200 questions can be asked. Each question (including prompt and help) is limited to under 1000 characters. In quest.txt every question must begin thus:

```
start mark   question number   question type   next questions by ans
```

for example:

> 1 T3 (start question 1 a text question and then go to question 3)
This is the text of question one.

or

> 1 A2B3C3D4 (start question 1, multiple choice, go to 2 if A. . .)
Text of question.

A question is ended by a blank line (no spaces on line).

The user responses are written to the end of a text file (answer.txt) preceded by the question number. The first question (question one) is preceded by the time and day—taken from the computer clock—that the answer session started.

4.4 Limitations of the Olive system

As it stands the Olive system only intercepts when the user presses a restart key. This may minimize interference with the user's search. However for certain kinds of question—for example asking the users about their interpretation of what a high number of hits means—interruption while a particular catalogue screen is being displayed may be more effective. This would require the Olive system to extract information from the screen —e.g. number of hits—and decide to interrupt immediately. This level of program complexity was not attempted.

Olive is currently designed specifically for the CL-CAT system at City University. Most aspects of the program could be altered to work with other systems. The interception code is the most CL-CAT specific part of major and would require changes for each kind of OPAC. Development of the interception function to cope with other kinds of online searching (including command language based systems) would be the most challenging aspect of generalising the approach of the Olive system.

Figure 1. Example of Olive question file

```
                   Esc[1;36mOLIVE QUESTIONNAIRE
                   Catalogue evaluation projectEsc[0m
1
Have you found the book you wanted?
Please answer Y or N

2
Is it available for loan?
Please answer Y or N

3
What did you look for in the catalogue
[a] A book (or books) some of whose details you know
[b] Something on a topic
[c] Combination of A and B
[d] Give up
Please select A or B or C

4
What will you do? -- now that the book is available for loan
[a] Look for just that book on the shelves
[b] Look for that book and something on the shelves
Please select one option [a]-[b].

5
Will you look for:
[a] another book with similar details
[b] something else on the same topic
[c] something else on a different topic
Please select one option [a]-[c].

6
Please could you summarise what you wanted?
Type in a few phrases or a sentence -- For example
1)      The book "Wealth of Nations by Adam Smith"
2)      Information about industrial relations in the car industry
                   [To END press enter twice]

7
>> RESTART -- go on with your searching
                   Press C: to Continue
>> END -- finished with the catalogue
                   Press D: when Done

8
What will you do? -- now that the book is not available for loan
A) Try the catalogue again another time
B) Go to the shelves
C) Reserve the book
D) Give up
Select: A or B or C or D
```