LOGiT, development of a video analysis tool

Tjin-Tham-Sjin, A.M.

Award date:
1994

Institute for Perception Research
PO Box 513, 5600 MB Eindhoven

21.11.1994

1245

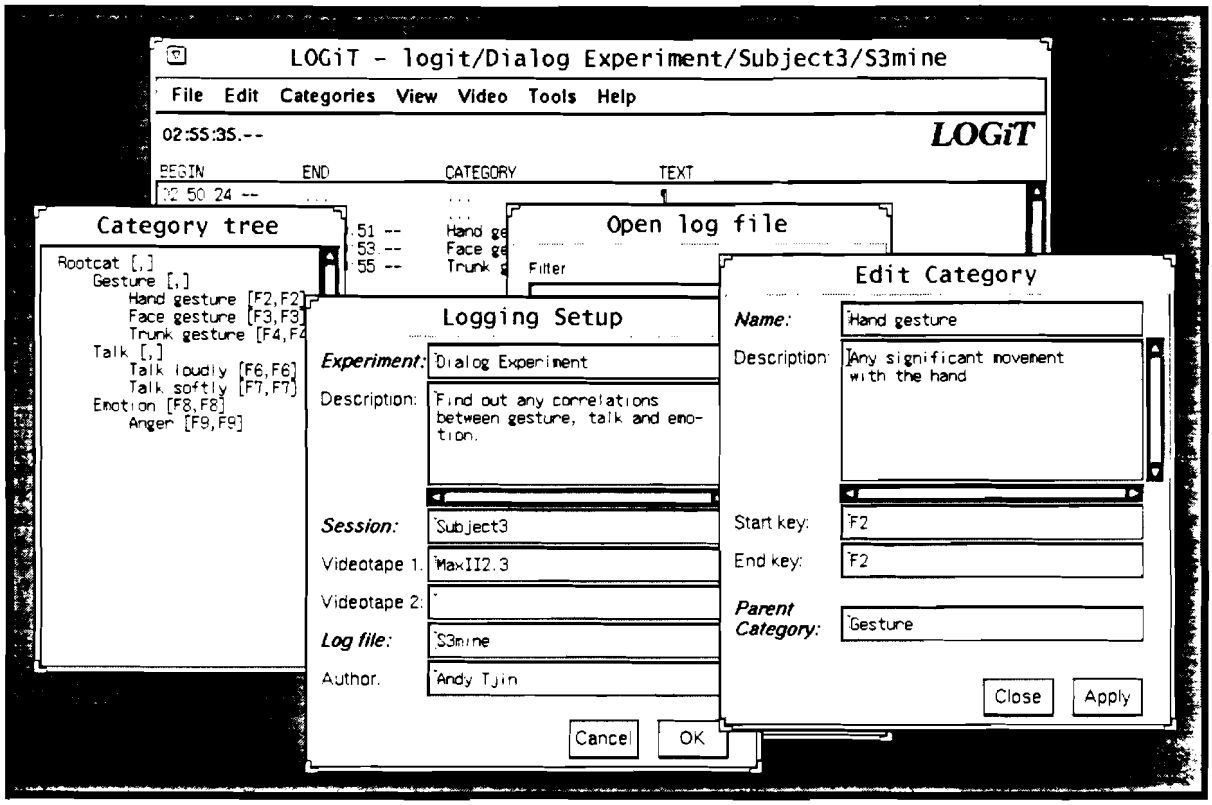
Rapport no. 1018

LOGiT, development of a video analysis tool

A.M. Tjin-Tham-Sjin

LOGiT, development of a video analysis tool

Master's thesis by A.M. Tjin-Tham-Sjin



Report on graduation project
performed from November 1993 to October 1994
as appointed by Prof. Dr.Ir. F.L. van Nes (IPO)
and Prof. Dr.Ir. J.E.W. Beneken (TUE-EME).
Supervised by Ir. J.H.M. de Vet (IPO).

INSTITUTION OF PERCEPTION RESEARCH and
SECTION OF MEDICAL ELECTRICAL ENGINEERING
FACULTY OF ELECTRICAL ENGINEERING
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Summary

Within the research field of information ergonomics video-equipment is often used to record the behaviour of subjects, in order to make a behavioral analysis later. For this purpose it is important to make adequate textual representations of the video tape content, the log files. A small literature research showed that no commercially available logging tool fitted the video laboratory at the Institute for Perception Research (IPO). A special logging tool has been developed.

By means of interviews with the potential users the user requirements were established. It turned out that different approaches of logging, user definitions of categories and (optionally) the control of the VCR had to be supported. From the user requirements a concept model of the tool was built and evaluated with the potential users. After the concept had been adapted to the users' wishes, an object-oriented model of the logging tool could be finished. This object-oriented model was the starting point for the implementation of *LOGiT* with XDesigner in C++ and Xmotif.

The tool developed retrieves the time code from a VCR and lets the experimenter type in text and/or define behavioral categories. The log-file format is prescribed as a standard text file, so that the file can be analyzed in more detail with other applications.

Evaluation of the tool proved that it could be improved on some user-interface details, of which some have already been adapted. On the whole the tool turned out to be living up to the expectations of the users. Three of the four logging approaches are currently being supported. Although the video control and the timeline features of *LOGiT* are not finished yet, *LOGiT* has now been installed in the IPO video laboratory and is ready for video analysis.

Samenvatting

Binnen het onderzoeksveld van informatie-ergonomie wordt vaak video-apparatuur gebruikt om het gedrag van proefpersonen op te nemen, zodat er later een gedragsanalyse gemaakt kan worden. Voor dit doel is het belangrijk dat er geschikte tekstuele representaties van de inhoud van de videoband worden gemaakt, de logging-files. Een klein literatuuronderzoek wees uit dat geen enkel commercieel verkrijgbare logging-tool geschikt was voor het videolaboratorium op het Instituut voor Perceptie Onderzoek (IPO). Daarom is een speciale logging-tool ontwikkeld.

Door middel van interviews met de potentiële gebruikers werden de gebruikerseisen vastgesteld. Het bleek dat verschillende logging-benaderingen, gebruikersdefinitie van categorieën en (eventueel) de besturing van de videorecorder moesten worden ondersteund. Vanuit de gebruikerseisen werd een conceptmodel van de tool gemaakt en geëvalueerd met de potentiële gebruikers. Nadat het concept werd aangepast aan de wensen van de gebruikers kon het objectgeoriënteerde model van de logging-tool worden voltooid. Dit objectgeorienteerde model was het uitgangspunt voor de implementatie van *LOGiT* met XDesigner in C++ en Xmotif.

De tool leest de tijdcode uit een videorecorder en de experimenteerder hieraan tekst laten toevoegen en/of gedragscategorieën laten definiëren. Het log-fileformaat is voorgeschreven als een standaard tekst-file, zodat de file gedetailleerder geanalyseerd kan worden met andere applicaties.

Evaluatie van de tool toonde dat het op sommige user-interface-details kon worden verbeterd; sommige daarvan zijn al aangepast. Over het geheel bleek de tool de verwachtingen van de gebruikers na te leven. Drie van de vier logging-benaderingen worden op dit moment ondersteund. Hoewel de videobesturings- en de tijdslijnfuncties van *LOGiT* nog niet voltooid zijn, is *LOGiT* nu geïnstalleerd in het videolaboratorium van het IPO en klaar voor video-analyse.

Table of contents

Appendices:

CHAPTER

1 Introduction

In these days of fast improving technology mankind is increasingly making use of electronic devices. Many of those devices are complicated in use and because even the most gifted technical designer can overlook features of the user interface, the importance of user interface evaluation and research in man-machine communications is growing.

The main research field of the Information Ergonomics research group (*InfoErgo* group) at The Institution of Perception Research (IPO) in Eindhoven, is the field of man-machine communications. In their research video equipment is often used to record the behaviour of subjects, in order to make a behavioral analysis later. In this way for example bottle-necks in the user interface of a device may be found.

One of the most important kinds of data which are attained during the video experiments is temporal data. With the video recordings time instances and time intervals of user actions are often measured to make conclusions on the subject's behaviour. At each time instance or time interval the experimenter can add some interpretation or description of what happened then. This process is called *logging*. Thus during the logging, the video data is transformed into readable text data.

The measuring of time instances and time intervals can be done in several ways. One can simply take a stopwatch and write down the relative time instances during display of a video recording. One can also record a time code on the video tapes and copy it into the data file manually. In this case we see a clear cross reference. Every time code in the data file is referring to a specific and exact point on the video tape and every time code on the video tape can be placed in the context of the data file. The third way of measuring is letting the time code on the video tape be retrieved automatically when an experimenter thinks it is necessary.

To support this last way of logging, a special video analysis tool is required. The tool has to be able to read a time code from a video player, display it and offer the user of the tool a possibility to add some comment to each time code. Although a lot of logging is being done in the InfoErgo group, a video analysis tool is not yet available. This report describes the development of a video analysis tool for the equipment that is being used currently at the IPO, I named the project *The Logging Tool Project.*

CHAPTER

2 The Logging Tool Project

2.1 Introduction to logging

Let me describe the process of logging and the process of behavioral analysis with video recordings. As I describe the process of logging, I will introduce some logging specific definitions which are marked in italics.

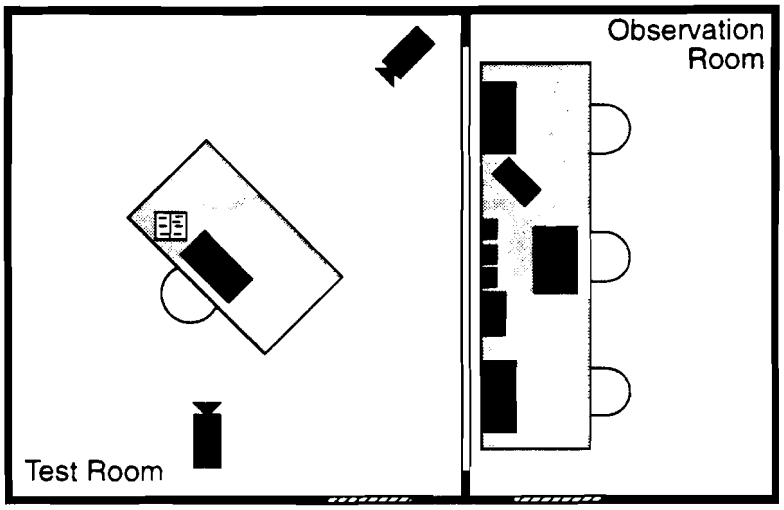


Figure 1. Hypothetical usability laboratory

A floor plan of a typical experiment setting is pictured in figure 1. In the *test room* one or more subjects are placed with some tasks to perform. During the tasks all subject actions are video taped with video cameras and in some usability labs a one-way mirror is placed between the observation room and the test room, so that the experimenters can look into the test room, but the subject cannot see the observation room. In the *observation room* the experimenters are controlling the video recording equipment and annotating what happens in the test room, possibly using workstations.

Because the information on the video tape is hard to handle for experimenters, all this information must be converted into readable (textual) data. This is done by marking every event on the video tape or simply writing down the time an event takes place. Optionally interpretation or descriptions can be related to every event. This process is called *logging*.

Tape contents

An *event* is merely an occurrence that happens during the experiment. Therefore an event is sufficiently defined by a time instance or a time interval on the video tape. Sometimes several events have such similarities that they can be clustered into categories. Thus a *category* is a name for a collection of all possible events that suffice to certain constraints. These constraints are defined by the experimenter. For example one can imagine that during one experiment two subjects are asked to perform tasks with a new device. During those tasks the subjects may talk to one another. Two obvious categories would be TALK1, for every event that subject 1 is speaking, and TALK2, for every event that subject 2 is speaking.

Several categories can imply a bigger category and thus one category can exist of smaller *subcategories*. For example the two example categories TALK1 and TALK2 could be subcategories of a category TALK, which indicates speech by any of the subjects. An example of a possible category tree can be found in figure 2. In this figure the root category is called "Event".

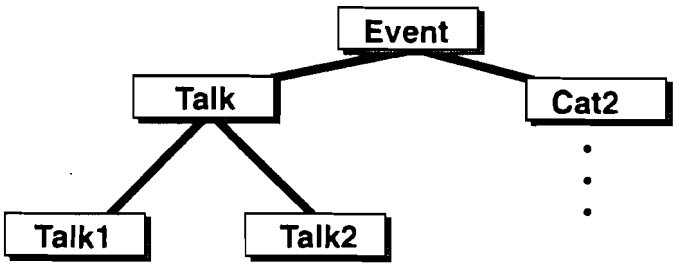


Figure 2. Possible category tree

Process

During one experiment several video recordings are made in which one or a few experimental variables are varied. Experimental variables are for example the subjects and the setting. Each separate video recording is called a *session*.

Each session can be logged during the actual time a session is recorded, which is called *on-line* or *real-time* logging, or after the session already has been recorded on video tape, which is called *off-line* logging. The main difference between on-line and off-line logging is the possibility to stop and rewind the video tape during off-line logging.

One logging *phase* is a whole run through the video recording of one session, during which new logging information is added or updated. So one experimenter can log a session in as many phases as he would like to. An on-line logging phase is always the first phase in a logging session (if it is applied at all, of course).

It's possible for different users to log one session. All logging information of one session of one user is stored into one *log file*. So several log files can describe one session. One experiment consists of several sessions and on each session several

users make their log files (see figure 3). Most of the time one session will be recorded on (a part of) one video tape, but because of efficiency in use of the tapes one session might be recorded on (parts of) two tapes.

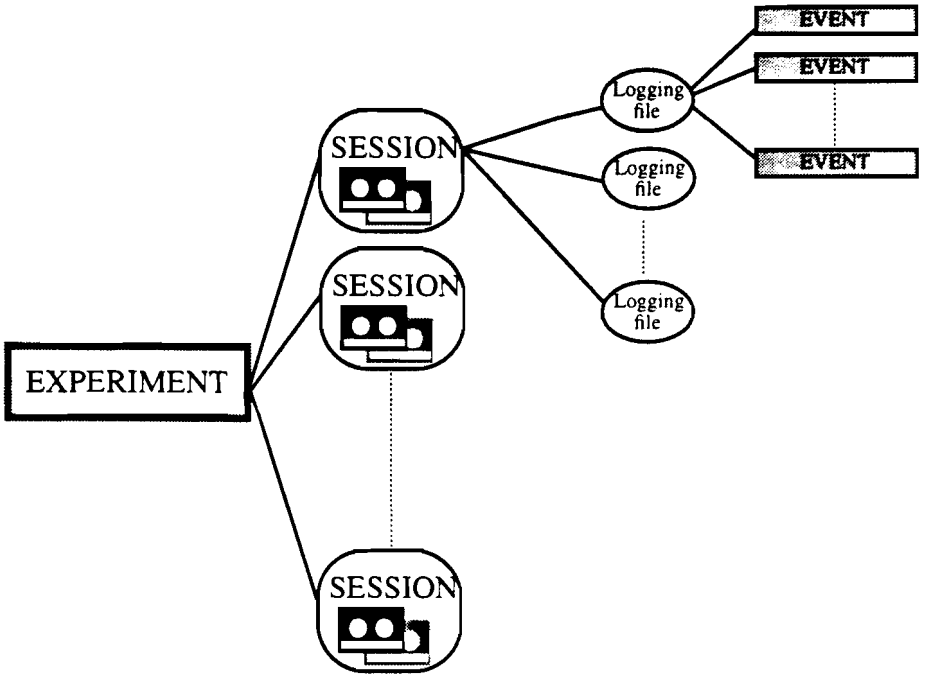


Figure 3. Experiment construction

Tool to support process

Each session might be edited into a summarizing compilation. Therefore a script must be written. A *script* consists of several *clips* (= time intervals in the video tape), which can be played consecutively. A clip can be an event if that event is a time interval, or a time interval around one event if that event is a time instance.

Nowadays several computer based tools exist to support the process of logging in behavioral analysis with video recordings. Such tools are generally called Computer-Aided USability Engineering tools, or *CAUSE* tools. Some of these CAUSE tools will be described in the following section. The reason for usability engineers to use computer-based tools are depicted in [Hoiem, 94]:

- Computer-based tools are more convenient and less time-consuming than paper-based approaches for viewing video segments, as well as for recording and organizing annotations to the segments (Kennedy 1989, Roschelle and Goldman 1991)
- Computer-based tools enable data analyses that would be impossible or at least very time-consuming to do manually (Hammontree *et al.* 1992, Kennedy 1989, Kinoe 1989, Roschelle and Goldman 1991, Sanderson 1990)

2.2 Existing CAUSE tools

A small literature research shows that logging tools already exist, but only few are commercially available. Many professional usability engineers try to find better and more efficient ways of collecting and analyzing their data. In a current survey of 33 Usability Professionals Association members, over 70% currently use some form of software data collection tool. Of these 80% have created their own tool while 20% have purchased one [Weiler, 93]. In this chapter a short summary of some of the tools will be given. This will give an image of what logging tools can offer at the moment.

The simplest logging tool one could use is a text editor or word processor. *Simple "keystroke" logging* in Table 1 does not mean that logging the subjects' keystrokes is possible, it means that the experimenter can simply use one keystroke to log an event, rather than having to position the mouse pointer and click the mouse button.

The term *Category logging* in Table 1, implies offering the user an option to register user defined category code with one simple action like a keystroke or a mouse click.

Table 1. Comparison of CAUSE tools

Tool name	Computer Systems	Simple "keystroke" logging	Category logging possible	Text comment possible	Video Control for	File export convenience	Reference
Text editor	all	no	no	yes	none	yes	--
CVideo & Video-Noter	Mac	yes, start and end times.	yes	yes*	Sony VCR's, Pioneer Laserdisc.	no	[Rochelle, 91] & CVideo brochure
Camera	PC	yes, start and end times	yes	no	any VCR	yes	[Vlugt, 92]
DRUM	Mac	no, with buttons	yes	yes	Sony VCR's	yes	[Macleod, 93] & DRUM brochure
Observer & Reviewer	PC	no, with buttons	yes	yes	Panasonic 7300 AG 7350 AG	yes	[Hoiem, 94]
OCS-tools	PC	?	yes	no	Panasonic incl 7500 AG	yes	OCS-tools brochure '93
VANNA	Macintosh	no, with buttons	yes	yes	any VCR	yes	[Harrison, 92]
Ideal IPO-tool	SUN	yes, start and end times	yes	yes*	Panasonic 7500 AG	yes	--

As can be seen in Table 1, what we wanted of the new tool was not available at that moment. I must add that at the end of the development process of our new tool, another tool drew our attention that coincidentally (?) looked very similar to the

tool we had built. That tool is called Timelines [Owen, 94], it has all the features that the ideal IPO tool in Table 1 has and even turned out to work in a similar way, except that it runs on an Apple® Macintosh instead of a SUN®workstation.

The * in the "Text comment possible"-column means that links can be created to non-textual information. This is of course also the case in the ideal situation, but not taken into account in this report and left for further development of the tool.

2.3 General concept

So as we couldn't find a suitable existing tool, a new logging tool had to be developed. The main purpose of the logging tool is support of the logging phase in behavioral analysis with video-recordings by means of linking events to descriptions or interpretations of these events. The result should be a log file in a simple data format, which is easily linked to other data processing tools.

The optional purposes are control of the VCR by the tool and support of the detailed analysis. The latter means that the tool offers possibilities to view log data in several formats, so that the user can draw conclusions on the subject's behaviour more easily.

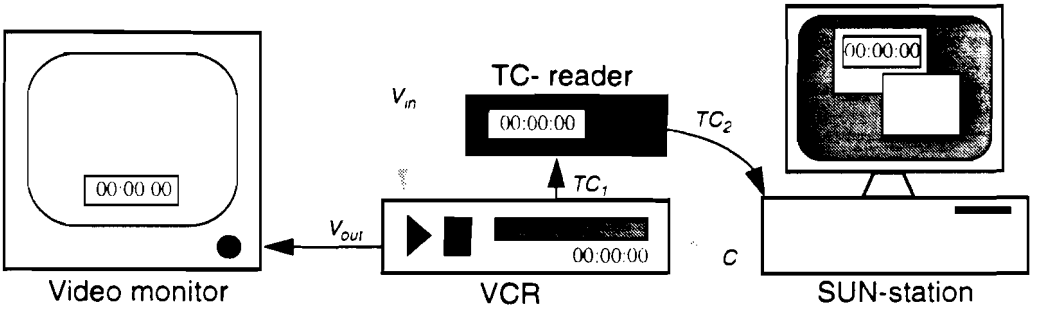


Figure 4. Configuration of the VCR-system

The tool must fit in the configuration of the VCR-system and SUN computer network at the IPO. This means that the tool is developed for:

- 1 or 2 Panasonic AG-7500 VCRs with TV-monitor
- 1 Fostex 4010 Time Code Reader & Generator
- 1 SUN-station, in an X-windows environment.

In figure 4 a camera video-signal (V_{in}) is recorded on tape in the VCR. The VCR sends video signal on tape (V_{out}) to the television monitor. The VCR-time code (TC_1) is read and reformatted by the time-code reader which sends the reformatted version (TC_2) to the computer. Optionally the VCR can be controlled by the computer with a control signal (C) .

CHAPTER

3 User requirements

Some requirements had already been expressed a few years ago, when a group of experimenters on the IPO established some demands for a logging tool. Because this tool had to be designed from the drawing board it could as well satisfy the specific needs of the IPO InfoErgo group. Therefore a user analysis and task analysis was performed first. After that, as the tasks for the tool were established, a model for the user interface of the tool could be made.

3.1 Task analysis

A few questions needed to be answered before the technical design of the tool could start. Jakob Nielsen gives an overview of the usability engineering lifecycle, which can be seen as a guideline for developing a *usable* tool [Nielsen, 93]. The first step in the overview is get to *know the user* in order to acquire a feel for how the product will be used. Since the exact definition of the logging tool was not known at the start of this project, Nielsen's first step was followed. Inspired by Nielsen, a few guidelines for finding out the tasks of the tool were made:

Know the user:

a. Individual user characteristics. Who are the users?
b. The user's current and desired tasks. What do they *need*? How do the future users *want* to be supported in their tasks? Do different users want different possibilities for the tool?
c. Functional analysis. How should the tool help them? What are the exact tasks that the logging tool has to be able to support?
d. Conceptual design.
e. Evaluation of the conceptual design. How will future users make use of several features?

After these points the design is ready for implementation. After implementation even more evaluation can be done to see if the tool is exactly what the users want.

3.2 Execution of the task analysis

Establishing the group of potential end-users wasn't very hard, since the tool would be built for a specific group of users: the InfoErgo group members at the IPO. Though the tool will eventually be constructed to be implemented easily on different platforms, for the task analysis and user interface design interviewing several InfoErgo group members would suffice.

Because the group of users was known, some presuppositions could be made. All users are familiar in the field of behavioral experiments, are familiar with window based platforms on computers, have at least one university degree and have no eye-sight problems. Most of the future users will be using the tool in the observation room or in their own rooms where other group members could be present. Therefore the tool must not hinder other persons in the room.

11 experimenters were asked:

- *Question:* What is your field of work? (to understand for what purposes the tool would be used)
  Answers: From the answers one could conclude that logging is used for fundamental behavioral research - such as dialogue research -, transcriptions of speech and man-machine interaction evaluation and research.
- *Question:* How do you perform logging without a logging tool?
  Answers: All experimenters simply copy the time code of the VCR by hand. Some of them simply use pen and paper for this purpose.
- *Question:* Do you use categories during logging? If so, how?
  Answers: Some experimenters don't use categories at all, for example the transcribers. Some experimenters don't use categories in their first logging phase but use the first phase to establish the categories and use the categories later. Some experimenters make a hypothesis before starting the experiment. From the hypothesis they can derive expected categories, so that they even use categories during the first phase of logging.
- *Question:* Do you structure your categories (in case they used categories)?
  Answers: Most of the category-using experimenters do not. Some used the category tree as described in section 2.1.
- *Question:* How do you think you would use the logging tool and what are your expectations?
  Answers: Most of the potential users already had an idea of how the time code should be retrieved and that category or comment would automatically be filled out. If they didn't have an idea, a possible plan of logging was presented and the future users could give positive or negative feedback on that.

From the results of the interviews a paper mock-up of the tool was designed. This paper mock-up was shown to all interviewees, so that they could comment if their wishes were presented correctly and so that they could get a chance to come up with new ideas after their initial ideas were shown in a more concrete form. As the tool would be designed for an X-windows/UNIX environment, the paper mock-up contained all the windows of the user interface of the tool with all their foreseeable features (mostly menu items). Finally a presentation of the final paper mock-up to the whole group of future users was given to evoke last user wishes and needs. The final windows that were presented are given in the figures of section 3.3.

3.3 Tool requirements

The future users' wishes and needs and the functional general concept (Chapter 2) lead to the draw up of the tool requirements, which will be described in this sec-

tion. To understand all tool features one should have a look at figure 6, which shows the main

window design of the tool, but before the window will be elucidated, a few important user requirements are discussed.

3.3.1 Approaches of logging

Among the future users preferences for 4 different *logging approaches* could be distinguished. These approaches all imply different scenarios of working with the tool and therefore have to be supported by the tool:

1. **Text-oriented**
   Goal:
   This approach is used to create a quick table of contents of the video tape. The focus is on unstructured textual comments.
   Operation:
   No time code is retrieved until the first key on the keyboard is pressed. Then the text which is typed in is immediately connected to the time code of the first keystroke and both time code and text are shown on the screen. The text-entry is closed by pressing the [enter]-key.

2. **Fast category-oriented**
   Goal:
   This approach is used for a detailed (partial) (on-line) analysis with pre-defined categories, but without the need of textual comment. In this case events of different categories succeed one another very fast, so that no time is left for additional comments.
   Operation:
   No time code is retrieved until a predefined key is pressed. Then the category belonging to the predefined key is immediately connected to the time code; both are shown on the screen. Text-inputs can be entered and exited both by striking the [enter]-key. Text is also shown on the screen. If the logged event is time-interval dependent then pressing the end-of-interval key retrieves the time code which will be shown on the screen.

3. **Slow category-oriented**
   Goal:
   This approach is used for a detailed (partial) (off-line) analysis with predefined categories and additional textual annotations. In this case events of different categories succeed each other rather slowly, so that there's enough time to log textual comments between two succeeding events.
   Operation:
   No time code is retrieved until a predefined key is pressed. Then time code and category are both shown on the screen. The cursor is immediately put in a text-field where text can be edited. Closure of this text-field is done by striking the [enter]-key. If the logged event is time-interval dependent then pressing the

end-of-interval key retrieves the time code and this will be shown on the screen.

4. **Function keys oriented**
   Goal:
   This approach mixes approach 2 and 3, where categories can quickly be logged by using the function keys as the predefined keys, and additional comments can be added by using the rest of the keyboard. The main advantage is that, while text is being added, categories can still be logged without leaving the text-entry mode.
   Operation:
   Text-fields can be typed in by the alphanumeric keyboard. The category keys are all defined under function keys. Pressing a predeifined category key retrieves the time code, beginning and end of events are shown as well as the category. The advantage of this approach is that it doesn't need two different modes for text-entry or category-key capturing.

As a default the text-oriented approach will be supported, because this is the only approach with which the user can start logging without predefining keys.

3.3.2 Set-up of categories

Events can be of different kinds of categories. A possibility is offered to define logging categories in the tool. Because users must be able to start with the tool right away, the defaults are set in such a way that a category set-up is not required.

The properties of the categories are:

- *Name*, this is the name of the category that will be shown in the tool, whenever this category is referred to.
- *Description*, this is the specification of the category by the user.
- *Keys*, one key can be assigned to the start of an event of this category, one key can (in addition) be assigned to the end of an event of this category. These keys can be the same key if the user wishes to toggle between start and end of an interval event. Setting the end-key implies a *time-interval* event. The assigned keys might not only be keys on the keyboard, but might be a button on an auxiliary input device. Key-assignment is not obligatory for the user; omitting key-assignment simply means that this category is not selected for the current logging phase.
- *Parent category*, states whether a category has a parent category and the name of that parent category. Recursively a parent category must also be a defined category.

As a default there will be one predefined "root" category, called "Event", with as default property settings (as mostly used for on-line logging):

- *Name* = Event
- *Description* = [empty]

- *Startkey* = F1
- *Endkey* = [none]
- *Parent category* = [none]

These settings must be easily alterable.

3.3.3 Video control

It's preferable to control the video recorder from within the tool. 'Ordinary' control functions, such as play, stop, fast forward and fast backward must be available. Also logging-specific functions must be available:

- Wind or rewind to time code *t1*.
- Start playback at time code *t1*.
- Play segment [*t1*, *t2*] from the video tape then stop.
- Play or wind to one selected event from the log file.
- Repeat selected segment *N* times.
- Make a *script*, i.e. a file with clips from the video tape. A clip is a piece of film on the video tape that is defined by its starting time code and its ending time code on the tape. The time-instances in this file can e.g. be obtained by playing the tape and pressing a button on the right time-instance.
- Play a script on the video player.
- *Soundplay*. This function rewinds the tape just for several seconds and then plays the tape, so that the sound on the tape will start exactly where it was cut off when the video player had stopped.

3.3.4 Analysis support

One of the most important features of the tool is that the log file is exportable in a standard text format, from which the events' information can easily be derived, such that detailed analysis (that is statistical analysis) can be done in external applications. The archiving of the video tapes and log files is also supported in the log-file format.

A time-line presentation of logged categories has to be obtainable from the log file. In this way certain patterns of event-occurrence can be visualized. An example of a

time line presentation is shown in figure 5. Every time-interval is drawn as a rectangle, every time instance is drawn as a thick short line.

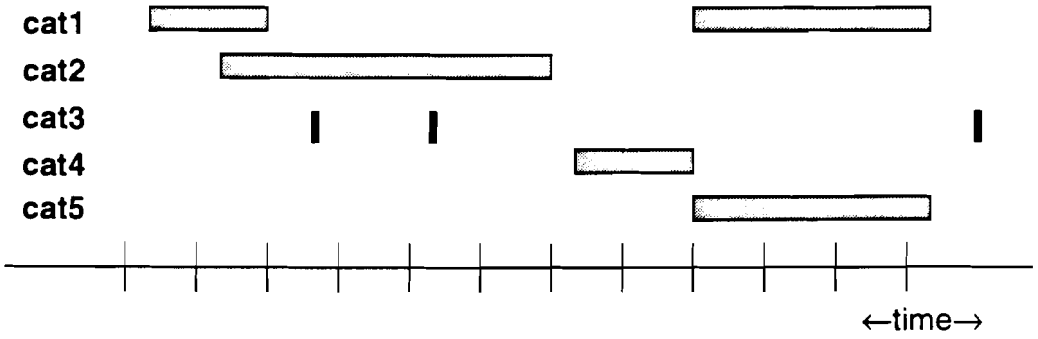


Figure 5. Example of a time-line presentation

It has to be possible to perform some filtering and merging of the log files before exporting the files to other applications.

3.3.5 Conceptual design

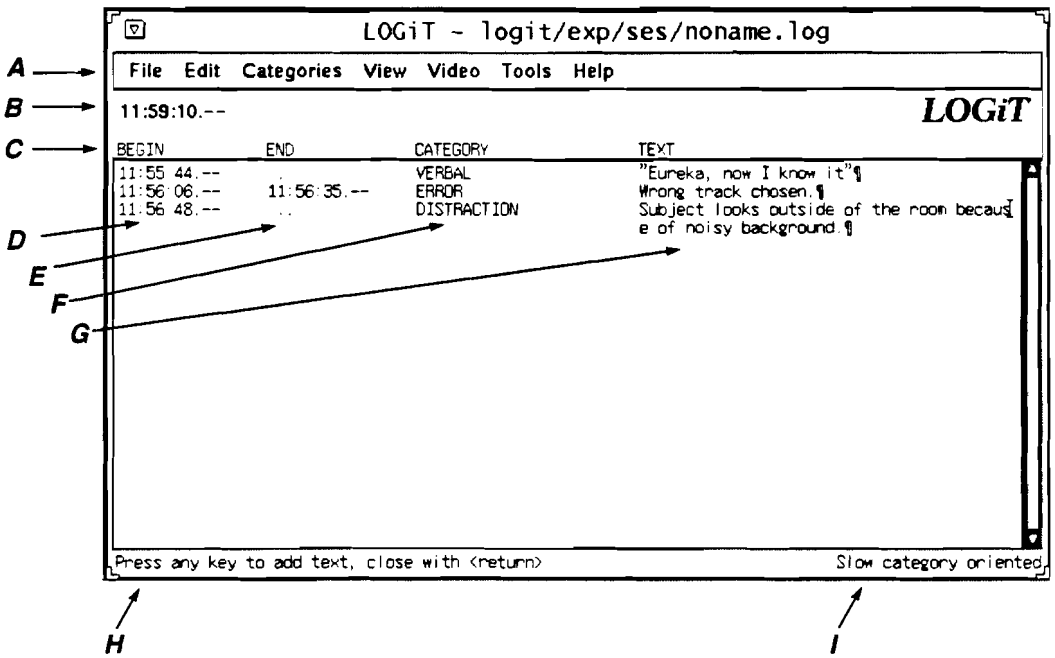


Figure 6. Main window of the logging tool: the Logging Window

All functional characteristics of the main window are explained below:

A The Menubar

B The Current Time Display; this label shows the time code that the VCR is displaying at the moment.

C Column names; the texts on this horizontal line give the titles of the columns below them.

D The Begin Column; for the time codes which indicate the beginning of an event.

E The End Column; for the time codes which indicate the end of an event.

F The Category Column; for the names of the categories of the event.

G The Text Column; for any textual comments on the event.

H The Guidance Label; this label shows all guidance texts. A guidance text is a text which tells the user what can be done next.

I The Approach Label; this label shows the approach that the user has chosen.

D, ***E***, ***F*** and ***G*** together will be called the *EventsField*.

All menu items in the menubar of the main window are described as follows:

Menu 1.

File	
New	Automatically starts Setup then makes new log file.
Open ...	Opens file box, so that the user can enter or choose a log file.
Save (needed/not needed)	Saves the current log file.
Save as ...	Opens file box, so that the user can enter or choose a file-name. Saves current file under that name.
Import ...	Opens file box. Allows the current log file to be merged with the selected log file
Setup ...	Opens Setup dialog box. Allows to make changes in the experiment setup
Use categories ...	Opens file box. Allows for importation of category definition of another log file.
Exit	Terminates the tool.

Menu 2.

Edit	
Undo/redo	Undoes the last alteration in the log file.
Cut	Deletes the selected event or text part and copies it to the clipboard.
Copy	Copies the selected event or text to the clipboard.
Paste	Pastes the clipboard after the selected event or cursor.
Split event	Allows one event to be split into two events of the same category and time codes. The cursor in the text column decides where the text of the initial event will be split.

Menu 2.

Edit	
Connect events	Allows two events to be merged into one event. The time codes of the first event will be the resulting new time code.
Set approach ...	Shows toggle menu with: Text oriented, Fast categories, Slow categories, Function key oriented. Only one of these approaches can be chosen at the same time (this is called the Radio Box behavior).

Menu 3.

Categories	
Show/hide tree	Opens/closes window with the category tree information
Add category...	Allows additions in category definitions in dialog box Edit Category. As a default the category which is currently selected will be the parent of the new category.
Change category	Allows edits in predefined category definitions of the selected category.
Delete category	Deletes selected category from the user definition with confirmation from the user.

Menu 4.

View	
Show/hide Begin-time column	Toggle item. Show or hide the Begin Column in the Logging Window.
Show/hide End-time column	Toggle item. Show or hide the End Column in the Logging Window.
Show/hide Category column	Toggle item. Show or hide the Category Column in the Logging Window.
Show/hide Text column	Toggle item (The toggle squares can also be functionalized by show/hide, conform to the rest)

Menu 5.

Video	
Show/hide control panel	Opens/closes window with the video control buttons
Speed ...	Allows with dialog box for entering the speed of playing the video tape.
Soundplay on/off	Soundplay = rewinding the tape before playing it in such a way that no sound on the tape is lost by the observer.
VCR selection ...	Shows toggle menu with: Player, Recorder. Only one these approaches can be chosen at the same time.

Menu 6.

Tools	
Script editor ...	Opens Script window, with options to filter the current log file with the script that is entered.
Time line presentation ...	Opens dialog box for showing a time line presentation.

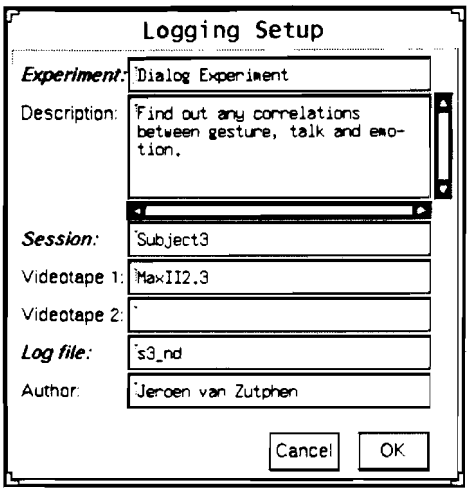


Figure 7. The Setup Dialog Box

The italicized items in the dialog box Logging Setup of figure 7 *must* be filled out if the user wants to log. All the other items are optional. This dialog box appears when a new log file is wanted. It also appears to modify the setup of an already existing experiment.

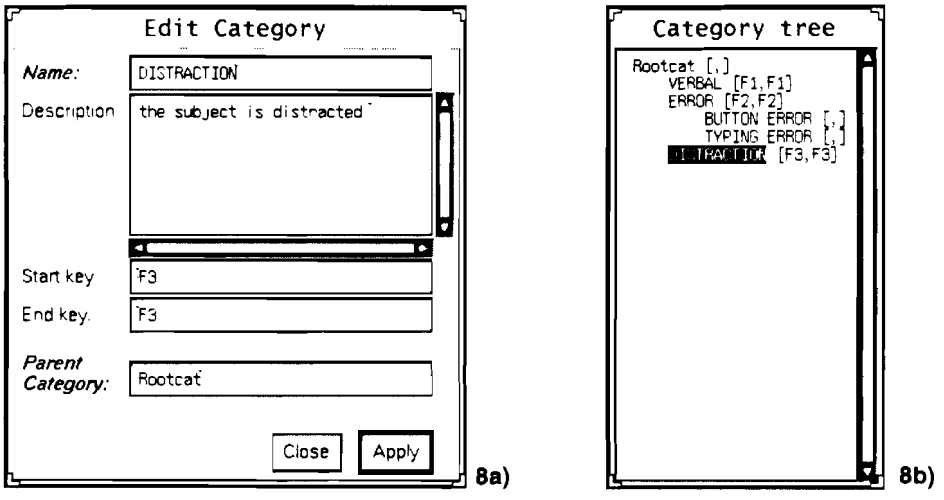


Figure 8. a) The Edit Category Dialog Box, b) The Category Tree Window

The italicized items in the Edit Category Dialog Box in figure 8 *must* be filled out if Apply is pressed. These items always have default values, so that defining cate-

gories should not be time consuming. All the items are optional. To select a category, the category can be clicked on with the mouse within the Category Tree Window as displayed in figure 8. Selected categories are always highlighted in the display.

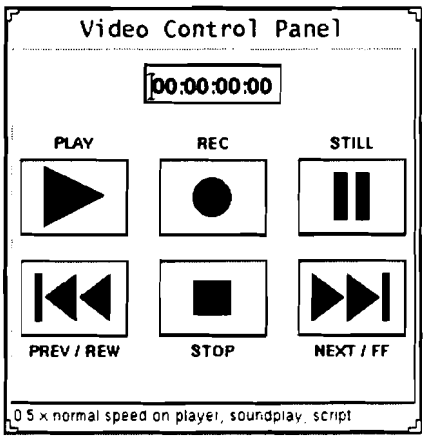


Figure 9. The Video Control Panel

The buttons on the Video Control Panel, figure 9, have the same functions as on a normal VCR-system. On the status bar information is given about playing speed, VCR selection and Soundplay.

When the VCR is not playing or winding (thus the time code is not running) the time code can be entered in order to play or wind to a certain time code. Some research had been done in order to find out how the VCR should be controlled from the SUN-station. The 34-pin assignment table of the VCR is included in appendix E.

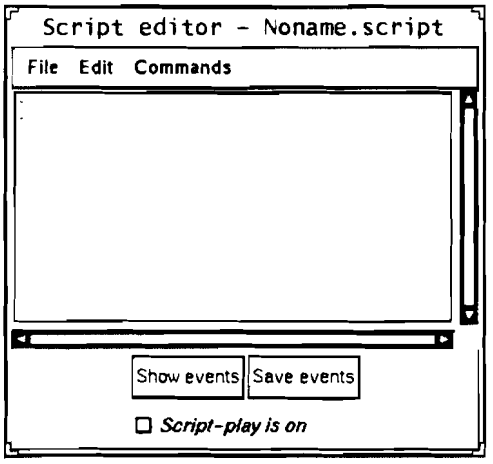


Figure 10. Script Editor Window

With some kind of macro language the user can define a script with the Script Editor as shown in figure 10. One can think of if-statements, repeat loops, time constraints etc. that will define one unambiguous script of clips or events.

Table 2. **The buttons in the Script Dialog box**

Button name	Description
Save events	Filters all time intervals implied by the script to the current log file.
Show events	Mark all the events implied by the script in the current log file.
Script-play	Script-play means that the control panel behaves on the script: plays and browses between all script events.

The *File* menu contains all standard file operations. The *Edit* menu contains all edit functions that can be used for the script. It would also be possible to use the edit menu in the main window for this purpose, but because this window should be seen as a tool itself, the choice was made for an edit menu in this window as well.

When the script works on the control panel the Play Button will start playing all events that are implied by the script, the Rec Button will start recording all events that are implied by the script from the player to the recorder (the recorder will thus be still when the player is searching for a next event). Both the FF and the Rewind Button will wind the tape to the previous or next event.

The filebox in figure 11 was *not* included in the paper mock-up, for it was presupposed that all users were familiar with standard file boxes such as the Motif-feel file box that was used for the tool. Because of completeness it is shown here, so all the windows in the design of the logging tool will have been shown in this section.

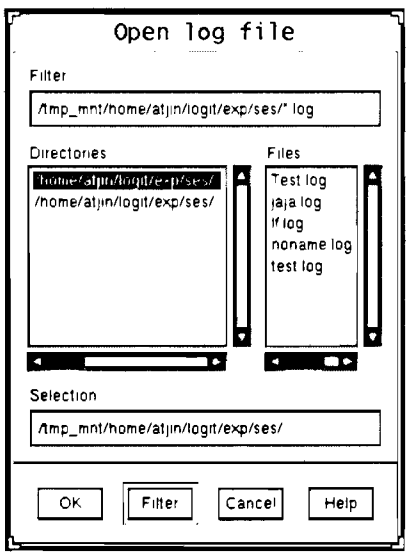


Figure 11. The File Box

A summary of all the windows in the final logging tool is given in table 3.

Table 3. **All windows and dialog boxes in the logging tool**

Name:	Figure number:	Finally implemented:
Logging Window	5	Yes
Setup Dialog Box	6	Yes
Edit Category Dialog Box	7a	Yes
Category Tree Window	7b	Yes
Video Control Dialog Box	8	No
Script Editor Window	9	No
File Box	10	Yes

CHAPTER

4 The object-oriented model

After the user requirements were established a design method had to be chosen in order to come to the formal specifications of the tool. We chose for an object-oriented design method. First of all the meaning of object-oriented design will be discussed. Then I will discuss the objects found for the Logging Tool Project and eventually a more detailed discussion of the methods of the objects.

4.1 Object-oriented design and programming

The design method chosen for the Logging Tool Project is that of object-oriented design. In an object-oriented approach (OOA), opposed to a conventional procedural approach, all kinds of *objects* are being distinguished in the system that is to be designed. The objects are defined in such a way that the whole system is described by the objects and by the communication between those objects.

The very important properties of OOA are *data abstraction, encapsulation* and *inheritance*. In [Coad, 1990] definitions of these terms are given:

- *Data abstraction* = The principle of defining a data type in terms of the operations that apply to instances of the type, with the constraint that the values of such objects can be modified and observed only by the user of the operations.
- *Encapsulation* = Information hiding, a principle used when developing an overall program structure, that each component of a program should encapsulate or hide a single design decision. The interface to each module is defined in such a way as to reveal as little as possible about its workings. This OOA-characteristic is also often referred to with the more familiar but weaker term *modularity*, which alludes to the factoring of a large program into units that can be modified independently [Tesler, 1986].
- *Inheritance* = Properties or characteristics are received from an ancestor. It simply means that a hierarchy of objects is possible, in which the children inherit all characteristics of the parents but can also have new characteristics of their own.

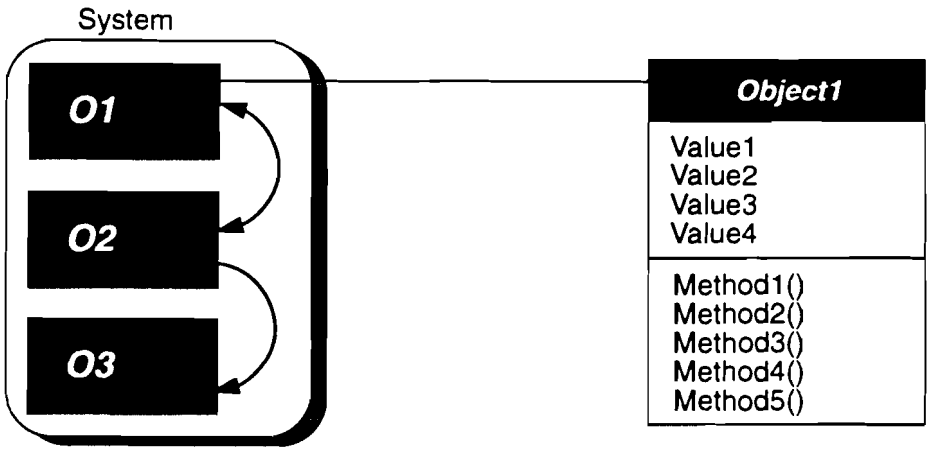


Figure 12. Data abstraction and encapsulation

All this is achieved by seeing your system as a whole and recognizing the objects in your system. Every *object* is thus an encapsulation of *attributes* (or sometimes called *values*) and exclusive *methods* (or sometimes called *services*).Thus an object is an abstraction of a part of your system, with some number of instances in the system. By classifying the objects, you can model how objects relate to each other. Via their methods objects can communicate with each other.

For example in figure 12 the system is described by three instances of objects, which are called O1, O2 and O3. O1 is an instance of Object1. In the description of Object1, one can see how all the data concerning instances of Object1 is encapsulated. The methods of Object1 are the only things that influence the values of instances of Object1. So if Method1() is usable by O2, O2 can communicate with O1. Let's assume that O2 is an instance of Object2 (which is not described in figure 11), and that Object2 is a child of Object1. Then O2 will have the same structure of values and methods with possibly some additions. That's inheritance.

In object-oriented design objects can be related by friendship. This is depicted with the arrows: O2 is a friend of O3. As the opposite may appear to people, friendship does not need to be mutual for objects. One object is a *friend* of another if one can use all the private information of the other.

This may all seem very abstract at the moment, but as I will describe the process of my OOA in this chapter I'm sure it will all become clear. The greatest advantage of OOP (object-oriented programming) is the very structured and uniform handling of data structures. Once already made, objects are very easily used again for the making of new objects that resemble the old objects. Rewriting pieces of code does not affect code of other objects. Once the whole object model is made, the implementation should not be very hard to realize.

Now the decision for an OOA was made, it was almost obvious to decide to use C++ as the programming language. C++ is a language that supports object-oriented programming fully. Because not too much time was to be spent on deve-

loping the user interface, a high level user interface design program was used: XDesigner 3.0.

4.2 Objects in the design of the Logging Tool

The first step in OOA is to recognize all the objects in your system. In this section a summation of all the objects will be given (table 5). A more detailed description of all the objects' attributes and methods as they are used in the final implementation is given in Appendix A. For clarity one example, that of the VideoSystem, is given in table 4.

Table 4. **Description of object Videosystem**

Attribute:	Description:
CurrentTime	The time code, a time-variable attribute. This attribute cannot be set by other objects.
Mode	{Idle, Play, Record, FF, FB, SF, SB}
F: Play ( )	The VCR is set to playing the tape
F: FF ( )	The VCR is set to fast forwarding the tape
F: Rewind ( )	The VCR is set to rewinding the tape
F: Rec ( )	The VCR is set to recording
F: SetSpeed (NewSpeed)	The playing speed of the VCR is set to NewSpeed.
F: Stop ( )	The VCR is set to idle state
F: Pause ( )	Holds the video image on the VCR. The VCR is set to still.
F: GetCurrentTime ( )	Retrieve the current time code and return it.

All values of the object are given first. The methods are preceded by "F:" .

Table 5. **Summation of objects**

Object:	Brief description:
VideoSystem	Contains all data of the VCR that is to be connected to the tool.
Event	Contains all data of one event, e.g. begin time, end time, category and textual comments.
LogFile	Actual external file(s) containing/to contain all information concerning the current logging session.
Setup	Contains all experiment data, like experiment name, session name, name of the log file and author.
Category	All data that comes with the category. E.g. the parent category, the category name, the category key definitions.
CategoryTree	The data structure that keeps the organization of the categories intact.
Logging	All logging tool settings.
VideoControl	All video control settings of the tool
TimeLine	All time line settings
Clipboard	Data concerning the last object that was copied or cut.

Except for all the objects described in table 5 there would of course appear a lot of objects that are necessary for the screen layout of the user interface. Those are in fact all the windows that are described in section 3.3. Those windows form the interface between the user and the objects described above. Because the behaviour of the windows would depend a lot on the design tool (Xdesigner) and the libraries from which the windows would originate (Xmotif), none of the user-interface windows were described at this point. It was assumed that the windows would show the user correctly what the objects in the system needed.

Because the script handling objects were of lower priority and time constraints for the Logging Tool Project had to be kept in mind, those objects were not considered at this point.

The next step was to describe which objects were friends, as described in section 4.1. How the objects of the Logging Tool relate to each other is shown in figure 13.

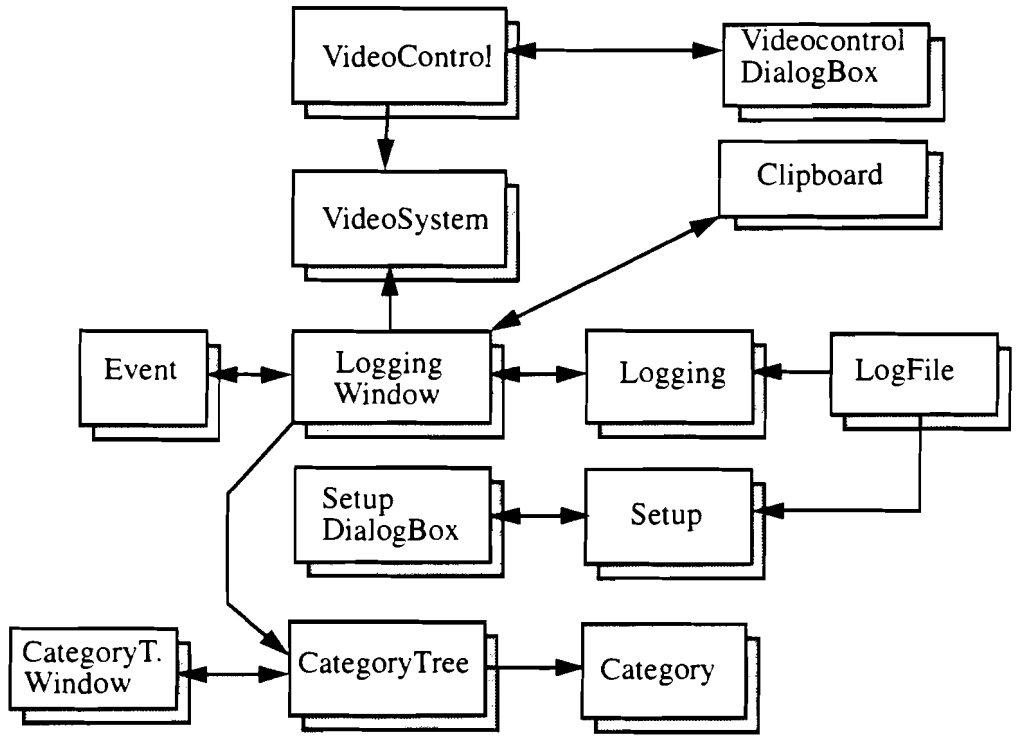


Figure 13. Object friendship diagram of the Logging Tool

In pure object-oriented programming objects are not allowed to observe or change values of other objects, not even of friends. For this purpose special methods should be constructed like GetValue() and SetValue(). Such methods would simply return the wanted value or set the wanted value, which thus in fact has exactly the same effect as direct access, except that the methods are a bit more inefficient by charging the memory stack some more. For this reason I decided not to apply this

rule of OOP and just use direct access where simple get- and set-procedures would else be called.

In the end it appeared not necessary to make a special separate object for an event. The object Event was replaced by an object EventsField, which contains a string existing of all events that currently are logged. In the model of figure 13 Event can just be replaced by EventsField, which makes the model valid for this change.

4.3 State transition diagrams

The four logging approaches of section 3.3.1 can all be regarded as modes in the tool. From one mode it is possible to switch to another mode with a menu manipulation, and as a default the tool will start in Text Oriented mode, because that is the only mode you can immediately use after start-up, without having to predefine category keys.To make clear how the tool will react on user input some state transition diagrams were made. These transition diagrams give a structured representation of how certain parts of the tool will behave.

What actually happens in the tool during logging depends on the state of Logging and can best be described by separate state transition diagrams. In these diagrams <key> means a user action via the keyboard, \ is used to indicate exceptions. So <key>\<enter> means all user keyboard actions except pressing the enter key. We assume that one enters these diagrams when the approach of logging is set.

All *Enter text*-states can be entered by clicking the mouse in the Text Column (the diagrams only depict keyboard actions). All other states can be entered by clicking the mouse in all other columns but the Text Column.

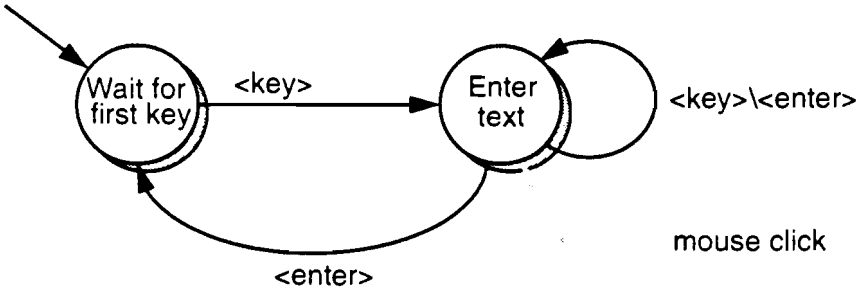


Figure 14. State transition diagram when LoggingApproach = Text oriented

When the logging approach is set to *Text Oriented* then the user enters the state transition diagram in figure 14.

a. *Wait for first key*: In this state the tool is waiting for a key to be pressed by the user. Once a key is pressed, the time code is retrieved and the pressed key is put in the text field.
b. *Enter text*: In this state the tool puts every pressed key, except for <enter>, in the text field. Once <enter> is pressed, the tool goes back to state a.

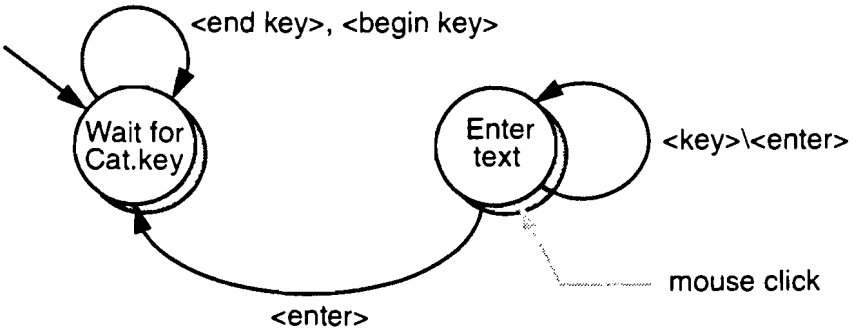


Figure 15. State transition diagram when LoggingApproach = Fast category oriented

When the logging approach is set to fast category oriented then the user enters the state transition diagram in figure 15.

c. *Wait forCat.key*: In this state the tool is waiting for a category key, predefined for the beginning of an event, to be pressed by the user. Once such a key is pressed, the time code is retrieved and the matching category name is put into the category field. If a category key for the end of an event is pressed then the time code is retrieved and put into the end-time field of the first event of the same category of which the ending time code hasn't been filled out yet.
d. *Enter text*: In this state, the cursor is always in the text field. Anything typed by the user appears in the text field until <enter> is pressed, then the tool returns to state c. This state can only be entered by mouse click in the Text Column, not by a keyboard action.

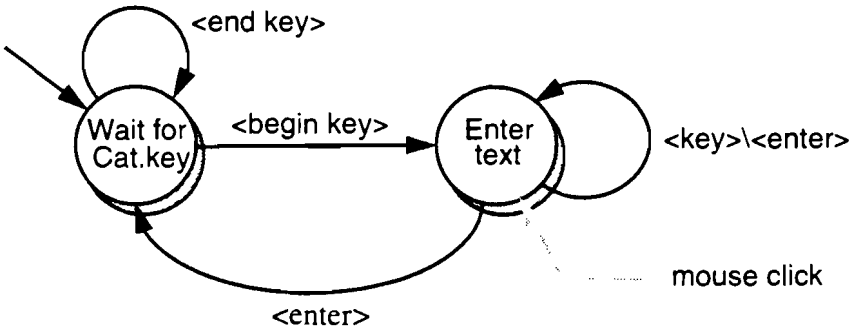


Figure 16. State transition diagram when LoggingApproach = Slow category oriented

When the logging approach is set to *Slow Category Oriented* then the user enters the state transition diagram in figure 16.

e. *Wait for Cat.key*: In this state the tool is waiting for a category key, predefined for the beginning of an event, to be pressed by the user. Once such a key is pressed, the time code is retrieved and the matching category name is put into the category field. If a category key for the end of an event is pressed then the time code is retrieved and put into the end-time field of the first event of the same category of which the ending time code hasn't been filled out yet.

f. *Enter text*: In this state, the cursor is always in the text field. Anything typed by the user appears in the text field where the cursor is, until <enter> is pressed, then the tool returns to state e.

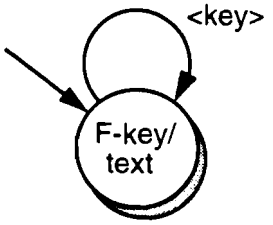


Figure 17. State transition diagram when LoggingApproach = Function key oriented

When the logging approach is set to *Function Key Oriented* then the user enters the state transition diagram in figure 17.

g. *F-key/text*: In this state the tool is waiting for a key. This can be a function key (F-key), predefined for the beginning of an event, being pressed by the user. Once such a key is pressed, the time code is retrieved and the matching category name is put into the category field. If a function key for the end of an event is pressed then the time code is retrieved and put into the end-time field of the first event of the same category of which the ending time code hasn't been filled out yet. In this state, the cursor is always in a text field. Anything typed by the user, which is not a F-key, appears in the text field until <enter> is pressed, then the cursor appears in the next text field.

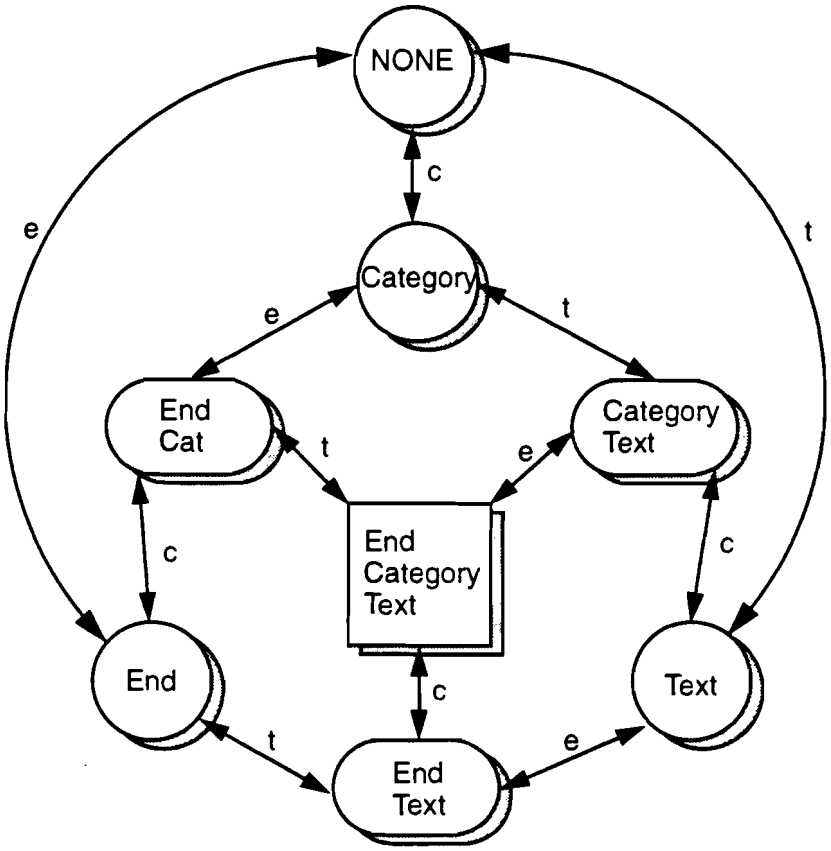


Figure 18. State transition diagram of attribute ColumsPref

The attribute ColumnsPref of the object Logging contains references to the names of the columns that are being shown on the screen. In figure 18 one can see how the content of ColumnsPref changes as a result of user actions. The actions are:

e for the EndColumnToggle in the ViewMenu
c for the CategoryColumnToggle in the ViewMenu
t for the TextColumnToggle in the ViewMenu.

Thus the combination of the user action and the state of ColumnsPref decide what actually happens on the screen. The Begin-time Column is always shown on the screen.

4.4 The object methods

So now we know what objects are in the system and we know what values and methods they accommodate. The next step towards a real program is specifying in a more detailed way how each of the most important methods will be working. This is not done in some special programming language, but just in some pseudo language which makes the algorithms clear and could even still contain functions that still had to be worked out. In this way this step is still not dependent on the programming language that is used for the implementation of the tool.

I'll give an example of how the methods were described in this pseudo language. Let's take the object VideoSystem again (table 4) for a small example. This object has the method GetCurrentTime(), which always gives the current time that's on the video system. The pseudo code for the method looks like this:

```
char Videosystem::GetCurrentTime()
{
        if F4010 is connected
        {
        F4010_gettime(String TC);
        reformat time code TC for display;
        }
        else
        {
        getlocaltime (String TC);
        reformat time code TC for display;
        }
        return TC;
}
```

CHAPTER

5 Implementation

So, now we have an object-oriented model of the logging tool and some descriptions of the methods of the objects. The next and final step in the design is then of course the programming code. As was mentioned before, the choice was made to use C++ as a programming language and XDesigner as a high level user-interface design tool. Because XDesigner uses Motif - and Xtool Intrinsics libraries, these libraries were also used for implementation.

5.1 Graphical layout

XDesigner (in short *XD*) lets one design a graphical user interface with a minimum of functionality and generates this interface in a programming language C or C++ with the use of Motif and Xtool Intrinsics libraries. One big advantage of XDesigner seems to be the possibility to define every separate user interface part as a separate object. In this way the object-oriented design of the previous chapter could still be regarded. Another advantage was that all objects could be generated and made functional one at a time; so it is easy to produce different segments of code at a time.

XD is in fact a graphical interface for designing Motif user interfaces. One can easily establish a hierarchy of so called *widgets*, which are any part of the user interface with a functionality of its own. Widget examples are windows, menu bars, buttons, text fields, labels etc. The hierarchy that was used for the Logging Window is given in figure 19.

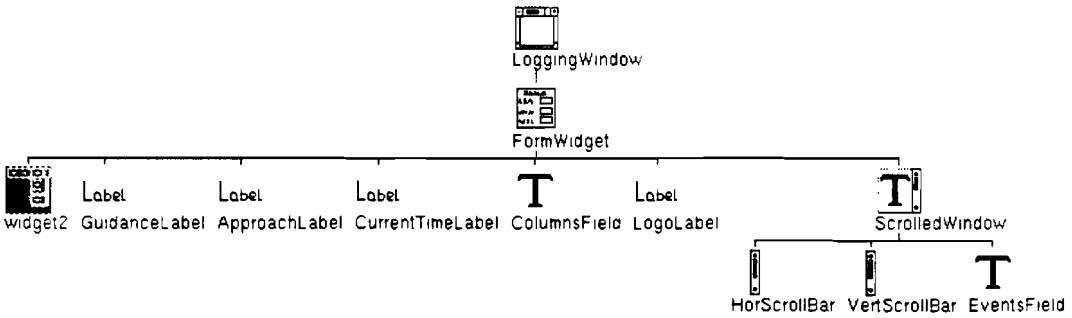


Figure 19. XDesigner hierarchy of LoggingWindow

A short explanation of figure 21: The LoggingWindow exists of several widgets. As one can see, the root in the layout is also called Logging Window. This widget is the widget that keeps all the other widgets together and controls all window decisions. The only child of Logging Window is FormWidget. This widget can be

installed to hold all other widgets in their right places, even if the window is resized or reopened.

Then you see a lot of labels; labels only indicate an amount of text or bitmap and are allowed to be changed during their life. The ColumnsField contains all the column indicators. Widget2 is the menubar, which in fact contains lots of other widgets (which are all buttons in cascade or sub menus), but those have been *folded* for the sake of the overview. Widget2 need not have a special name assigned to, because the menubar itself is not used in the code of the tool.

ScrolledWindow is a standard scrolling text window in Motif, that can be adapted to the wishes of the programmer. It contains two scroll bars and one text field, that are functionally linked to each other. Several editable text field configurations had been tested to be used for the implementation of the Logging Window. In the end this configuration was the only one that sufficed the requirements.

The graphical result of the hierarchy displayed in figure 19 has already been displayed in figure 6. Compare these two figures and see the link between hierarchy and graphical display. All windows in the user interface have been designed with XDesigner and have already been shown in section 3.3. in figure 6 to figure 11.

For all clarity I want to stress the fact that, although all windows have been designed with XD and therefore ready for use within the tool, not all of the windows have been integrated into the tool yet. Not yet integrated are: the Video Control Panel of figure 9 and the Script Editor Window of figure 10.

5.2 Functionality

After the graphical design had been generated by XDesigner, the functionality of all the widgets could be added. This is done by altering and adding C++ code to the code that was generated by XDesigner. This C++ code is using Xtool Intrinsics and Motif libraries.

5.2.1 Callbacks

An example of functionality: what should happen if the user of the tool presses the button Open in the File menu? A separate function should be written for the reaction of the tool to this user action. All those functions that react to a special user action in the application are called *callbacks*.

The first thing that comes to mind is that those callbacks should all be methods of the objects. Unfortunately, Motif cannot make an object's method a callback. So separate callbacks had to be made for every action. An example of a callback is NewCallback.

This callback is automatically called every time the New button in the File menu is pressed:

```
void
NewCallback (Widget, XtPointer, XtPointer)
{
    if (SaveButton->SaveNeeded)
    {
        NewFormWidget->ShowMe(0);
    }
    else
    {
        XmTextSetString(EventsField, "");
        SetupFormWidget->Manage();
    }
}
```

Some explanation: The arguments of NewCallback need to be of the types Widget, XtPointer and XtPointer for administrative reasons, but as they are not used in this callback, they are not given names (they are left undefined). SaveButton is another button in the File menu; it contains information whether the current log file has been changed in its attribute SaveNeeded. NewFormWidget is the Form Widget in a small window that asks the user if the last changes in the current log file should be saved. Because it has different questions to pop, depending on the button that calls NewFormWidget->ShowMe(integer). XmTextSetString is a Motif procedure that fills a text field with a certain string. In this case it clears the EventsField. Manage() is a method that all XD-widget objects have to make it appear on screen.

To be working in a totally object-oriented way this callback should actually be appointed to an existing object, in this case the most obvious object would be the XD-object NewButton. But then one callback could not be used for other user actions and that would be unpreferable. Furthermore it would have been a bit more

inefficient, because the XD-callback would merely call the object's method which would contain exactly the same code. The code would then look like:

```
void
NewCallback (Widget, XtPointer, XtPointer)
{
    NewButton->Action();
}

void
NewButton::Action ()
{
    if (SaveButton->SaveNeeded)
    {
        NewFormWidget->ShowMe(0);
    }
    else
    {
        XmTextSetString(EventsField, "");
        SetupFormWidget->Manage();
    }
}
```

So it was decided that the callbacks would not be totally object-oriented. The callbacks would still make use of all the objects' methods and values (in the pieces of code indicated by "->"). Thus the decision can be justified by the fact that you could see all the callbacks as methods of the matching window (in this case the Logging Window, because that's where the New button is) and as long as they would be used as such, the object-oriented approach would not be lost.

Callbacks that still have to be implemented are the ones for editing the Text Column in the EventsField of the main window and the Import function in the File menu. This is why the Cut, Paste and Copy functions in the Edit menu are not totally functional yet.

5.2.2 The EventsField

Functionality can also be found in the behaviour of the Motif text-field that was chosen to hold and show all the events. This was the hardest part of the assignment: how to adapt an existing standard Motif widget into an editable scrollable text field that complies to the wishes of the Logging Tool. However the functional specification of this particular field was not given, it was not very hard to make some guidelines which had to be followed when building the field, which I called EventsField:

- EventsField should contain a maximum of four columns: Begin, End, Category and Text.
- None of the columns should be editable, except for the Text Column.

- Comment in the Text Column should not be restricted to the width of the window. Whenever the text would be as long as the width of the window, the next character is to appear on the next line, in the same column.
- One click of the mouse in the columns Begin, End and Category should result in selecting and highlighting the relevant event. One click in the Text Column should allow the user of the tool to change the text in this column.
- The second mouse button is used to place the current category in the columns Text and Category.
- The third mouse button is used to place the current time code in the columns Text, Begin and End.

Because all these features didn't exist in the standard Motif text field, all these features of the EventsField had to be implemented by catching every cursor movement and mouse click in the text-field. In this way the cursor could never appear in the first three columns, but only in the Text Column. Some big advantages of the standard Motif text-field were that it scrolls automatically and that it can contain as many events as one would like to have. Furthermore no special data structure had to be found for the contents of an Event, because the content of an event is now established by some protocol, that demands that all events look like this:

TC-TAB-TC-TAB-CATNAME-TAB-TEXT-ENDOFEVENT

in which

TC = 11 characters which form a time code
TAB = ASCII character nr.9
CATNAME = 16 characters which form the category name of the event's category.
TEXT = a sequence of standard text characters, which is not restricted in length. A CR (Carriage Return) can occur (but cannot be added by the user of the tool) and if it is not the last CR in the event it is always followed by 40 blank characters to make sure the next text will occur in the Text Column.
ENDOFEVENT = ASCII character nr. 182 (= nr. -74 and looks like ¶) followed by a CR.

Almost the same protocol is used in the log files, except that in the log files TEXT may not contain CRs and ENDOFEVENT is a carriage return. This is not only easy for file manipulation, but also for compatibility with all other editors, word processors, data bases and spreadsheets.

5.2.3 The CategoryTree

The category tree needed a special data structure. As the name and figure 2 may already have suggested, the most obvious way to structure the category data is in a tree. The object CategoryTree consists of an unrestricted number of objects Cate-

gory. Let's take a look at the C++ definition of the object class Category_c, which was derived from its description in Appendix A.

```
class Category_c {
public:
    Category_c* FirstChild;
    Category_c* RightSibling;
    Category_c* LeftSibling;
    Category_c* Parent;

    Category_c();
    void ShowValues();// In the EditWindow
    void StoreValues();// From the EditWindow
    void MakeRootCat();
    ~Category_c();

    char Depth;
    char* Name;
    char* Description;
    char* StartKey;
    char* EndKey;
    char* ParentCat;
};
```

To make functions within the tree easier and memory problems would be out of the question, every category has pointers to his first child, his right sibling, his left sibling and his parent. The methods Category_c() and ~Category_c() are the *constructor*, respectively *destructor* of the object. The constructor is automatically called as an initializing method just before the object is declared in C++. The destructor is called just before an object is deleted.

The code for declaring the whole tree data-type is (somewhat shortened):

```
class CategoryTree_c {
public:
 Category_c *Root;
 Category_c *Current;

 CategoryTree_c();
 Category_c *AddCategory(Category_c *ToParent);
 Category_c *FindCategory(char* LookName);
 Category_c *FindKey(char* LookKey);
 void RemoveCategory(Category_c* DelCat);
                // Remove DelCat
 void SaveTree(FILE* CatFile);
 void OpenTree(FILE* CatFile);
 void ShowTree();
                // Display tree in CatTreeWindow
 ...
 ~CategoryTree_c();

protected:
 ...
};
```

All instances of class CategoryTree_c have one pointer towards a root category and one pointer towards a current category. The current category is the category that is highlighted in the category tree and displayed in the edit dialog box. The constructor of this class contains all initialization that should be done when declaring an object of this class. The constructor defines a new category and makes it the root of the tree. It also makes the root category the current category. The destructor neatly removes every category that is part of the tree, so that no memory will unreachably float.

For clarity the structure with pointers is given in figure 20. A pointer to NULL is a pointer to nothing. The end of the tree is easily found by checking if a pointer points to NULL. The pointer Current is coincidentally pointing to the second child of the root, but this could of course be any category in the tree. All methods of the class CategoryTree_c should maintain the structure of figure 20 and that's how the methods have been written.

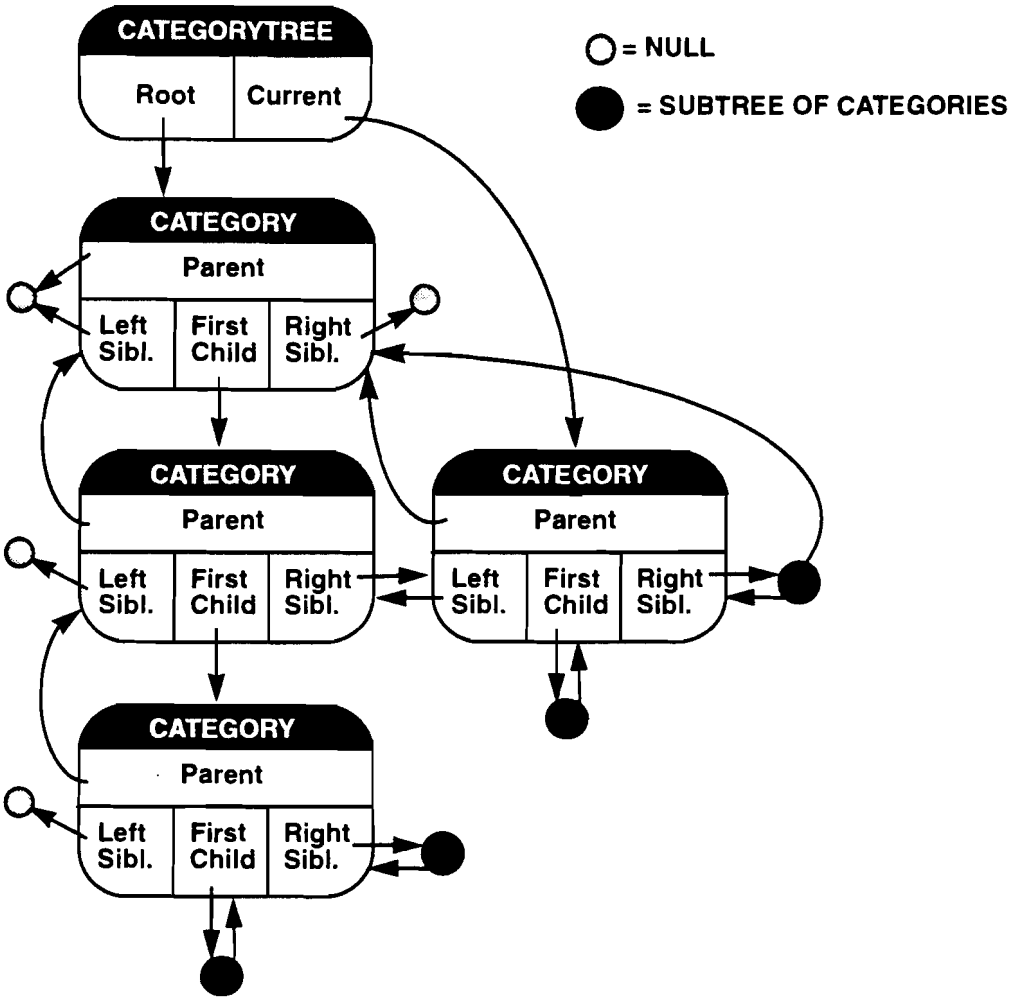


Figure 20. The category tree structure (sibl = sibling)

It is clear for example that in the method RemoveCategory() of the object CategoryTree this structure should be kept in mind. Here we see the advantages of so many pointers towards parents, children and siblings. If one category is removed from or moved within the tree, its left sibling gets another right sibling, its right sibling gets another left sibling and its parent might lose its first child.

5.2.4 File management

To support the file management according to the experiment construction in figure 3 the following protocol is used for the storage of the log files:

LOGiT makes a directory *logit* in the tool-directory. Then for every experiment a directory is made in *logit*. For every session a directory is made in the experiment directory. The session directory contains all log files concerning this session.

CHAPTER

6 Evaluation

This chapter will deal with how evaluation of the tool was done, what still has to be done and what can be improved in the Logging Tool.

6.1 Evaluation of the tool

For evaluation several potential users were asked to try out the logging tool. Furthermore the tool was tested in the observation room of the IPO. Several bugs were found and some of them could immediately be recovered. All users involved in this evaluation thought the tool was useful and thought that they would be able to use the tool right away. Some usability remarks were made during the evaluation:

- *Problem:* The function Split Event in the Edit menu resulted in a split of the event on the position of the cursor at that moment. In this way it is not easy to copy the time code to another event by splitting the event by standing on the last character of the text, so that you could also use the Split Event function as a function to mark several events at exactly one time code.
  Suggested and implemented solution: It would be preferable to place the cursor at the second part.

- *Problem:* Whenever the user wants to make a category-tree definition in the Edit Category Dialog Box, the user wants to see the category tree for an overview. This is also the case whenever a new category tree is loaded.
  Suggested and implemented solution: Automatically show the Category Tree Window whenever the Edit Category Dialog Box appears or another Category Tree is loaded from disc. This makes the user aware of the structure of the categories.

- *Problem:* A lot of users thought that the information on one category could be shown by clicking that category in the Category Tree Window.
  Suggested and implemented solution: A double click on one category in the Category Tree Window should make the Edit Category Dialog Box appear with all the selected category's information.

- *Problem:* Right now the place of the cursor in the Text Field might suggest that text can be typed at that place at that moment, while such is not the case when the tool is waiting for the logging of a new event.
  Suggested and implemented solution: The cursor may be replaced in more evident places after closing a text-entry with the [return]-key.

- *Problem:* The user gets no feedback of the tool, whenever something wrong is filled out in the dialog boxes or when something has been changed (by the user) in the dialog boxes and the user forgets to apply the changes.
  Suggested solution: Error messages have to appear whenever the user fills out something wrong in the dialog boxes and also a warning when the fields in a dialog box have been altered and not been applied.

- *Problem:* Most users had trouble with typing the exact names of parent categories, although they had predefined them themselves.
  Suggested solution: The Parent Category Field should preferably be a pull down menu which contains al the valid categories; this would avoid charging the user's memory and typing skills.

- *Problem:* The lines in the EventsField are too close to each other. When the log file contains just that much information that little 'white space' is seen on the screen, it might be uneasy to read.
  Suggested solution: The line spacing may be increased.

6.2 Evaluation of the programming material

XDesigner is very adequate for building the graphical part of a user interface. It may be quicker to firstly use prototyping languages such as Visual C++ on a personal computer. In that way ideas and user requirements can be tested earlier in the product development process. The way LOGiT was developed, i.e. without prototyping language because of lack of time, could have led to complex changes to an already existing complex code. Fortunately this was not the case.

Despite of some warnings by fellow students and colleagues C++ turned out to be rather easy to learn. Adding code in C++ and Motif to the XD-generated code was not that hard at all either, but changing features of already existing widgets is rather divergent and time consuming. A more elaborated evaluation of XDesigner combined with C++ is given in appendix D.

Object-oriented programming is a widely accepted method of programming now. The benefits of getting a good overview of large programs are not to be underestimated and certainly proved its usefulness in this project. One must take in consideration that although C++ is a fully object-oriented language, the Motif libraries are not always tuned in on C++ features.

The concepts of object-oriented programming are not very hard to understand, but it takes a while before these concepts can really be integrated in the design of a tool. It takes another way of regarding programs than one might be used to.

6.3 Future prospects

To get a quick view of what is possible with the tool at this moment and what still has to be made functional, one should take a look at figure 6. One can roughly say that all features under the menu items File, Edit and Categories are functional and that the items View, Video, Tools and Help still have to be implemented.

Not all features of the designed logging tool have been implemented yet. In fact, the tool can be used for logging (with category support) right now, but the control of the video recorders has to be researched more with respect to the communication protocols between VCR and computer. Once that is done, the Script Editor Window and the Video Control window can be made functional and integrated into the tool.

The Import function in the File menu still has to be made as well as Undo, Copy and Paste in the Edit menu. Furthermore, some warnings that will appear when the user does something that's not permitted, have to be implemented. This will enhance the usability of the tool. Finally some minor alterations will have to be made to suffice to the results of the user evaluation sessions.

Provisional plans also include porting the whole tool to a windows-platform on the personal computer, so that the tool can be used in places where UNIX-stations are not available.

What also needs some attention is researching the ways that a tool like LOGiT can simplify analysis of the log files itself. One might think of finding repetitively occurring patterns of events in log files and using the possibility of scripts controlling the VCR.

It can also be of use to do some research to find out how the analysis of video tapes with LOGiT can be standardized.

CHAPTER

7 Conclusion

After a small literature research, it turned out that no commercially available logging tool was suitable for the video laboratory at the IPO. By means of user-needs analysis and task analysis user requirements were defined for the design of the logging tool, *LOGiT*. From the requirements the user interface for the logging tool was designed with a high level user-interface design tool. An object-oriented model was established before actual implementation.

LOGiT is installed and ready for use at the IPO video laboratory. From now on it is possible to make log files of video-recorded experiments in a far more easy way than before. The supporting features of the tool are in short:

Approaches of logging

The logging process is supported for three approaches: text-oriented, fast category-oriented and slow category-oriented.

Use of categories

The definition of categories is fully supported with the option to use the alpha-numeric keyboard for a predefinition of category keys. A categories overview is given as a category tree in a separate window, in which also the key definitions are displayed.

Analysis support

The administration of video tapes and log files is supported in the Setup Dialog Box. The log file is stored in a simple data format which is easy to use in other applications.

What still has to be implemented is the function-key oriented approach, the time-line presentation of the event data and the video-control functions. The tool turned out to be living up to the expectations of the potential users. Still, the post-implementation evaluation resulted in a few adaptations that should be made to comply with the usability wishes of the potential users.

CHAPTER

8 Bibliography

8.1 References

[Coad, 90] Coad, Peter & Edward Yourdon (1990) *Object-Oriented Analysis*, New Jersey, U.S. : Prentice-Hall.

[Harrison, 92] Harrison, Beverly L. & Ronald M. Baecker (1992) Designing video annotation and analysis systems. *Proc. Graphics Interface '92*, 157-166. (survey)

[Hoiem, 94] Hoiem, Derek E. & Kent D. Sullivan (1994) Designing and using integrated data collection and analysis tools: challenges and considerations. *Behaviour & Information Technology* ***13(1&2)***, 160-170.

[Macleod, 93] Macleod, Miles & Ralph Rengger (1993) The Development of DRUM: A Software Tool for Video-assisted Usability Evaluation (submitted for publication in 1993)

[Nielsen, 93] Nielsen, Jakob (1993) *Usability Engineering*. London, Great Britain / San Diego, U.S. : Academic Press.

[Owen, 94] Owen, Russel N., Ronald M. Baecker & Beverly Harrison (1994) Timelines: a Tool for the Gathering, Coding and Analysis of Temporal HCI Usability Data. *Conference Companion CHI '94* ***24-28***, 7-8.

[Roschelle, 91] Roschelle, Jeremy & Shelley Goldman (1991) VideoNoter: A productivity tool for video analysis. *Behavior Research Methods, Instruments & Computers* ***23(2)***, 219-224.

[Tesler, 86] Tesler, Larry (1986) Object-Oriented Languages, Programming experiences. *Byte 1986* ***august***, 195.

[Vlugt, 92] Vlugt, Maarten J. van der & al. (1992) CAMERA: A system for fast and reliable acquisition of multiple ethological records. *Behavior Research Methods, Instruments & Computers* ***24(2)***, 147-149.

[Weiler, 93] Weiler, Paul (1993) Software for the Usability Lab: A sampling of current tools. *INTERCHI '93 Conference Proceedings,* 57-60.

8.2 Programming Literature

[Ferguson, 93] Ferguson, Paula M. (1993) *Motif Reference Manual for OSF/ Motif Release 1.2 (The Definitive Guides to the X Window System, Volume Six B)* USA: O'Reilly & Associates, Inc.

[Stroustrup, 91] Stroustrup, Bjarne (1991) *The C++ Programming Language, second edition.* Addison-Wesley Publishing Company.

[Mattson, 89] Mattson, Jeff (1989) *Think C™ Standard Libraries Reference.* USA: Symantec Corporation.

[IST, 93] Imperial Software Technology (1993) *XDesigner release 3 User's Guide.* VI-Corporation and IST.

APPENDIX

A Object descriptions

Object 1. **VideoSystem**

A5ttribute:	Description:
CurrentTime	The time code, a time-variable attribute. This attribute cannot be set by other objects.
Mode	{Idle, Play, Record, FF, FB, SF, SB}
F: Play ( )	The VCR is set to playing the tape
F: FF ( )	The VCR is set to fast forwarding the tape
F: Rewind ( )	The VCR is set to rewinding the tape
F: Rec ( )	The VCR is set to recording
F: SetSpeed (NewSpeed)	The playing speed of the VCR is set to NewSpeed.
F: Stop ( )	The VCR is set to idle state
F: Pause ( )	Holds the videoimage on the VCR. The VCR is set to still.
F: GetCurrentTime ( )	Retrieve the current time code and return it.

Object 2. **Event**

Attribute:	Description:
TC_begin	Time code of start of event
TC_end	Time code of end of event
Category-name	Name of a category of an event
Text	Text / interpretation
F: Show ( )	Put all attributes of event on screen

Object 3. **LogFile**

Attribute:	Description:
F: Load	Retrieve from disk
F: Save	Store on disk, add to experiment
F: Filter	Use script to keep only events implied by the script.

Object 4. **Session**

Attribute:	Description:
ExperimentName	Name of the experiment, which is also the name of the directory, in which all the log files will be situated. This directory also contains a file which contains all experiment data of all log files.(Except for event information)
Description	Description of experiment
SessionName	Name of session
VideoTape1	Name of tape on which the session is recorded
VideoTape2	Option for second tape name
LogName	Name of the current log file
Author	Name of logger
CurrentCategoryTree	In actual file, but not in this object
CurrentLoggingApproach	In actual file, but not in this object
F: Load ( )	Retrieve from experiment data
F: Save ( )	Save to experiment data

Object 5. **Category**

Attribute:	Description:
Name	Name of category
Description	Text
StartKey	Any alfanumeric key or function key or no key at all
EndKey	Any alfanumeric key or function key or no key at all
ParentCat	Name of parent category (recursive)
F: ShowValues ( )	Show all the values in the Category Tree Window
F: MakeRootCat ( )	Let this category be initialized as the root of all categories. (This should be called only once in the lifetime of a category tree.)
F: StoreValues ( )	Adapt all the values of this category to the values in the Edit Category Dialog Box.

Object 6. **Clipboard**

Attribute:	Description:
Content	Textual (actual) content of the clipboard item
Type	Type of information that is saved in the clipboard. {Event, Text, Script}!
F: Retrieve ( )	Get content from the clipboard
F: Store (content, type)	Save content and type on the clipboard.

Object 7. **CategoryTree**

Attribute:	Description:
Root	The root category.
Current	The category that is currently shown in the Edit Category Dialog Box.
NumberOfCats	Number of categories in the list of categories
F: AddCategory (ToParent))	Adds a category to the tree with ToParent as parent. Returns the just added category.
F: FindCategory (LookName))	Returns the category by the name of LookName.
F: RemoveCategory (Del-Cat))	Removes the category DelCat from the category tree.
F: MoveCategory (Move-Cat, ToParent)	Moves the category MoveCat with all its descendants to the position of a child of ToParent.
F: SelectCategory (SelCat)	highlights the category SelCat in the CatTreeWindow and makes SelCat current.
F: ShowTree ( )	Displays the whole tree in the Category Tree Window.
F: InstallKeys ( )	Install all the predefined keys of the categories to operate correctly.
F: UninstallKeys ( )	Install all the predefined keys of the categories.
F: Open (CatFile)	Load the file CatFile into this object
F: Save (CatFile)	Save this object as file CatFile

Object 8. **Logging**

Attributes:	Description:
Approach	{text, fast, slow, function} 'fast stands for 'fast category approach', 'slow stands for 'slow category approach', 'function' stands for 'function-key approach'.
ColumnsPref	P{end, category, text}, the BeginColumn is always on the screen.
Saveneeded	1 = the last save of the current log file has been changed.
TimeSource	{VCR, LOCAL}
F: ToggleSaveNeeded	Change SaveNeeded

Object 9. **VideoControl**

Attributes:	Description:
ScriptOn	Are the buttons acting on the script?
SoundplayOn	Does the play button also rewind the tape a little?
VCRSelection	{Player, recorder}
Speed	The speed of playing a video tape. (Speed is negative when the VCR is playing backwards)

Object 10. **TimeLine**

Attribute:	Description:
SelectedCategories	Categories that should appear in the time-line presentation
SelectedTime	Time interval that should appear in the time-line presentation.
To_file	Boolean: 1 = the graphic presentation will be saved, 0 = not
To_screen	Boolean: 1 = the graphic presentation will be printed in a window on the screen.
F: Print ( )	Make the presentation and print it to file or screen or both.

Because the tool is implemented with the help of the high level user-interface design tool, XDesigner, obvious elements of the U.I. are not discussed. Only objects in the U.I. that will have to communicate with the transparent objects are mentioned here with their relevant attributes:

Object 11. **SetupDialogBox**

Attribute:	Description:
ExperimentField	Widget for entering the experiment name.
DescriptionField	Widget for entering the description of the experiment.
SessionField	Widget for entering the sessioin name.
VideoTape1Field	Widget for entering the video tape's name or number on which the session is recorded.
VideoTape2Field	Widget for an optional second video tape.
LogFileNameField	Widget for entering the log file's name.
Authorfield	Widget for entering the author's name.
F:Get (Field, String)	Access the contents of the fields.

Object 12. **EditCategoryDialogBox**

Attribute:	Description:
NameField	Widget for entering the name of the category
DescriptionField	Widget for entering the description of the category
StartKeyField	Widget for entering the definition of the key that registers the beginning of an event of this category
EndkeyField	Widget for entering the definition of the key that registers the end of an event of this category
ParentCategoryField	Widget for entering the name of the parent category.
F: Get (Field, String)	Access the contents of the fields.

Object 13. **LoggingWindow**

Attribute:t	Description:
CurrentFileName	Name of the file that is displayed and of which the name appears in the title bar.
F: SetCurrentFileName (String)	Not only changes the attribute CurrentFileName, but also changes the name in the title bar of the window.
F: GetCurrentFileName (String)	
F: ShowCurrentTime ()	Display current time code in CurrentTimeLabel. Current-TimeLabel may be a seperate object.

Object 14. **CategoryTreeWindow**

Attribute:	Description:
TextField	In this field the construction of the category tree is described with a single string.
F: Get (String)	
F: Put (String)	

APPENDIX

B LOGiT manual

LOGiT stands for Logging Tool, a tool which supports the logging phase in behavioral analysis with video recordings.

B.1 Startup

Be sure the FOSTEX f4010 is connected to your SUN-station and to your VCR-time-code output if you want to use the VCR's time code. Use X11 as window platform by typing *~cadbin/startX11*.

The tool is started by simply typing "lwd" in the directory of the tool. When the VCR is not/wrongly connected to the SUN-station on which the tool is running, then at startup you will see the message: "lwd: Error initializing the f4010; check the connections". (If this message occurs on startup the tool will be working with local station-time.) Whenever a disturbance on the connection between VCR and computer occurs this message will appear in the shell, but as soon as the connection recovers the tool will use the VCR-time. If the directory in which you are starting up the tool does not contain the subdirectory "logit", this directory will be made in the course of the program.

B.2 Working with LOGiT

The main window of the tool looks like:

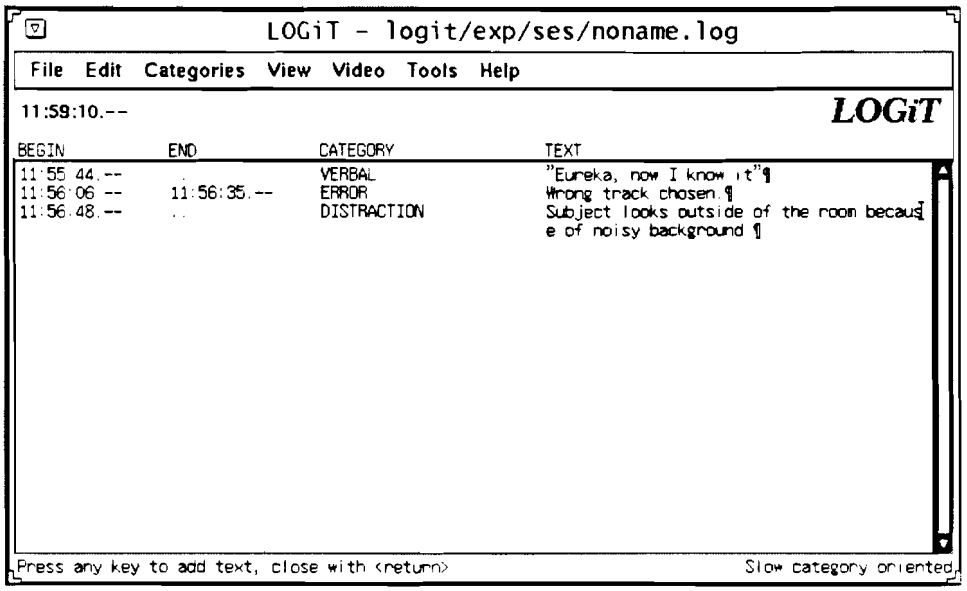


Figure B1. **The main LOGiT window**

For now only the first three menu items on the menu bar are functional: File, Edit and Categories. It is advisable to always use New or Open in the File menu to start your LOGiT session (for your own administrational sake, otherwise the tool will automatically save under a default name).

Editing in the text column is not yet perfect. Therefore the edit functions in the Edit menu are only valid for whole events (event = begin-time, end-time, category and text).

IMPORTANT!

There are several approaches of logging supported by the tool. Which approach is valid is indicated in the Approach Field in the lower right corner of the main window. The approaches can be chosen out of the Set Approach submenu in the Edit menu. The three approaches that are supported now are:

1. *Text-oriented*, no time code is retrieved until the first key on the keyboard is pressed. Then the text which is typed in is immediately connected to the time code of the first keystroke and both time code and text are shown on the screen. The text is closed by pressing the [enter]-key.
2. *Fast category-oriented*, no time code is retrieved until a predefined key is pressed. Then the category belonging to the predefined key is immediately connected to the time code; both are shown on the screen. Text-inputs can be entered and exited both by striking the [enter]-key. Text is also shown on the screen. If the logged event is time-interval dependent then pressing the end-of-interval key retrieves the time code which will be shown on the screen.
3. *Slow category-oriented*, no time code is retrieved until a predefined key is pressed. Then time code and category are both shown on the screen. The cursor is immediately put in a text-field where text can be edited. Closure of this text-field is done by striking the [enter]-key. If the logged event is time-interval dependent then pressing the end-of-interval key retrieves the time code and this will be shown on the screen.

MOUSE BUTTONS:

The first (or left) mouse button is used to select an event, if you click it in the three left columns, or to place the cursor in the Text column. In the menubar it is also used to select a menu item.

The second (or middle) mouse button is used to place the name of the current category in the Text column or in the Category column. You can select a category by clicking on the category in the Category Tree Window or by typing the name in the Edit Category Dialog Box and pressing return.

The third (or right) mouse button is used to place the current time code in the Text Column or in the columns Begin and End.

APPENDIX

C Changing functionality in the Logging Tool

C.1 File administration

All source files are saved in the directory *logitsources* and on the diskette by the same name. A short description of the files:

lwd.c:	contains the main program and all methods concerning the main window.
lwdexterns.h:	contains all the external declarations and all class definitions.
lwdstubs.c:	contains all callback definitions concerning the main window and all actions.
newwd.c:	contains all methods concerning the New Dialog Box
setupwd.c:	contains all methods concerning the Setup Dialog Box
filewd.c:	contains all methods concerning the FileBox
catwd:	contains all methods concerning the Edit Category Window and the Category Tree Window
homemade.c:	contains all methods concerning the non-Motif classes.
f4010.h:	header file of all the functions needed for retrieval of the time code from the VCR via the FOSTEX 4010.
f4010.a:	library containing the actual functions needed for retrieval of the time code from the VCR via the FOSTEX 4010.
makefile:	information for compiling and linking the source files into one executable.

C.2 Adding windows

To add extra windows in the tool, one should create a window with XDesigner. Then make sure that every widget that needs easy access is installed as a global C++-class. In this way the generated code will have proper overview and you will be protected from unnecessary child information. If the widget class is to be accessed by more objects than only its parent than make the class public. This can all be done in the "Core resources"-window, which can be popped up in the "Widget" menu-item.

If the window is ready you can pull down the "Generate"-menu and choose C++. If you are adding functionality to your widgets it comes in handy to generate a stubs file as well. In this file all the handlers will appear and the names of the call data will already be filled out, so that you don't have to look them up in the manual.

DON'T (!) make the top widget an application widget as is instructed in the XDesigner manual, for the tool already has an application widget. XDesigner will give a warning after generating the source code which you can neglect.

Declare the new window in the file *lwdexterns.h* and write the creation lines in the method *LoggingWindow_c::create* in *lwd.c*. All generated callbacks should also be declared in *lwdexterns.h* and the stubsfile should be included in *lwdstubs.c* (except for the external declarations of the stubsfile, which should all be present in *lwdexterns.h*).

C.3 Adding actions

Actions are the tool-reactions to user input from the keyboard or the mouse. If you want to use actions you will have to define them in *lwdstubs.c*. You should also put an external declaration in *lwdexterns.h*. Be sure not to forget to install the new action in the main program in *lwd.c*.

C.4 Adding callbacks

You might want to add callbacks to a widget, which you haven't define in XDesigner yet. Firstly install the callback's name in the *create*-method of the widget with the procedure *XtAddCallback*. Be sure to invent a name that refers to the widget as well as to the functionality of the callback.

The callback should be defined in *lwdstubs.c* and be of the form of an XtAppCallback (you might take other callbacks as an example). Don't forget to declare the callback's name in *lwdexterns.h*.

C.5 Compiling the sources

If everything is installed correctly you can simply type *make* in the directory *logit-sources*. The makefile is using CC for a compiler. Make sure that the Xtool Intrinsics and Motif libraries are accessible The XDesigner class definitions in the files *xdclass/h/xdclass.h*, *xdclass/lib/xdxmdialog.c* and *xdxtclass.c* should also be accessible from the make directory. For retrieval of the VCR's time code the files *f4010.h* and *f4010.a* should be accessible.

To use the latest updates of the SUN-operating system files, compile your final version on the *prles22*-machine.

APPENDIX

D Experiences with XDesigner 3.0 and C++

As I seem to be the very first person who developed a Motif application with XDesigner and C++ within Philips Research Laboratories, I was asked to write a small summary on my experiences with this combination, which is only just recently possible with the arrival of XDesigner 3.0.

Efficiency

XDesigner is a beautiful tool to quickly make a user interface in Motif. Of course, as with any high level design tool, the finally generated code is not as efficient as it would be in a lower level design. For every widget a separate class is generated in C++, but the code would be much more efficient if the only one class was made for every separate widget. What now happens is that every widget class has only one object instance. This does not affects the speed of the application, only the code length.

The speed efficiency of an XDesigner-generated code is just as efficient as a "handmade" Motif code.

Object-oriented design

XDesigner allows the user to let every user-interface widget be an object in the code. This is beneficial for the overview of the programmer, especially when an object-oriented approach is used. It is advisable to make every single widget an object. The latter can be done in the *core resources* of a widget, under section *Code generation*. Simply choose *C++ class* as the structure. It is a pity that there is no special feature on XDesigner that would make every widget an object in the code. It's rather tiresome to do this by hand for every single widget.

What is needed?

First of all the XDesigner executable and the CC compiler are needed. Besides that you need the files *xdxtclass.c, xdxmdialog.c* and *xdclass.h* for using the XDesigner classes, so that all widgets become objects in the source code. If you are using pixmaps for labels, then you also need the *xpm*-files.

And you need all libraries of *Motif, Xtool Intrinsics* and *X11*. As manuals I would advise the *Xtool Intrinsics User Reference* and *-Manual* and the *Motif User Reference* and *Manual*. These are on-line available on the Silicon Graphic machines at the IPO.

APPENDIX

E Video Control

The Panasonic AG-7500 has a 34-pins *Remote Control Connector*, which can be used for video control from a computer. The pin assignments are listed in table E1 and figure E1. Normally the pins are all *high*, every function is invoked by making the relevant pin low for 200ms. The pins 19, 30 and 33 determine the speed of the VCR.

Table E1. Connector pin assignment

PIN no.	Contents	PIN no.	Contents
1	REC SWITCH	18	CONTROL PULSE OUT
2	PLAY SWITCH	19	REMOTE 19
3	FF SWITCH	20	START MARK
4	REW SWITCH	21	EJECT SWITCH
5	STOP SWITCH	22	INSERT CH1
6		23	REC HOLD
7	PAUSE switch	24	PLAY HOLD
8	CASSETTE IN	25	FF HOLD
9	CUT IN SWITCH	26	REW HOLD
10	NOT SOURCE PLAY	27	INSERT CH2
11	SERVO LOCK	28	
12	GND	29	PAUSE HOLD
13	FRAME ADV. SWITCH	30	REMOTE 30
14	REVERSE COUNT	31	CUT IN HOLD
15	CUT OUT SWITCH	32	INSERT VIDEO
16	EDIT SWITCH	33	REMOTE 33
17	REVERSE	34	+12V

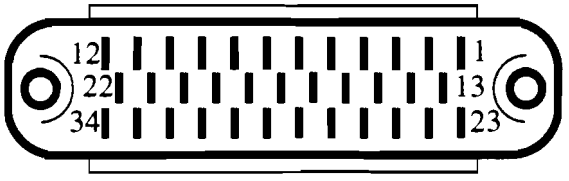


Figure E1. The remote control connector of the Panasonic AG 7500

Mailing list

Beun
Brouwer
Cremers
Freudenthal
Van Gelderen
Haakma
Van Hoe
Van Itegem
Kemp
Keyson
Majoor
Poll
Tang
Tjin (3 ex.)
Verheijen
De Vet
Westerink
Westrik

Vakgroep Medische Elektrotechniek, TUE, EH 3.05 (3 ex.)

Bibliotheek Elektrotechniek, TUE, EH 2.06

Mr. B. Thomas, PCD, SX

IPO-directeur
A.C. der Kinderen, Bibl. en Inf. Nat.Lab., WY 1.36 (8 ex.)
Dr.ir. W. Strijland, Octr. en Merken

5 archief