5. Interface Design and Development

Writing interface code is time consuming, and maintaining simplicity and consistency are a challenge. The software described here enables the user to control laserdisc players, mark the in and out points of specific segments, annotate segments, organize them on pages of a scrapbook, play back individual segments, and play back lists of segments. This chapter documents in detail the design decisions made, how those decisions evolved from my original conception, and the lessons learned in the process. Issues still to be solved are discussed in chapter 10, Future Work.

5.1 Titled Icons

5.1.1 Combining Text and Graphics

In the original design for The Electronic Scrapbook, logging was to be accomplished primarily by manipulating graphical icons. Thus, to indicate that a particular video segment was taken outdoors, one would drop an icon of a sun onto it (see Figure 5.1).

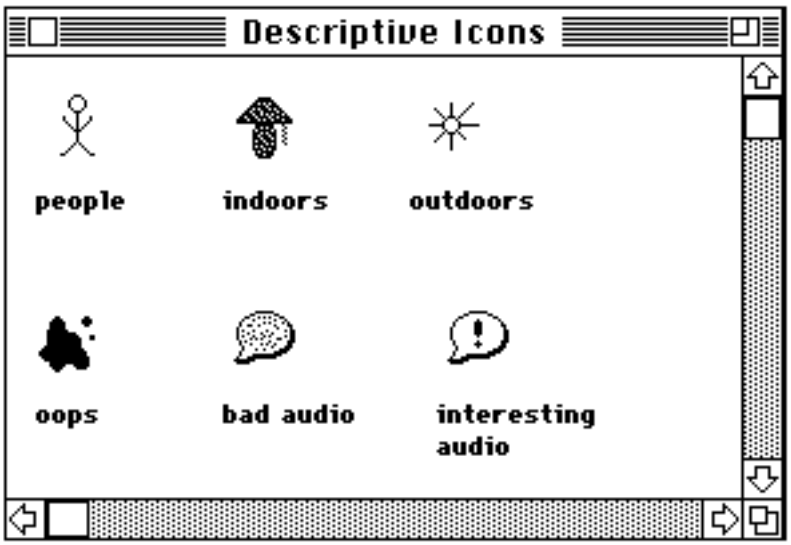

Figure 5.1: The descriptive icons window.

The meaning of icons is not always clear. Users understand the meaning of icons not only from the image itself but also from their expectations and familiarity with similar symbols. It can be difficult to create meaningful icons for unusual or abstract concepts. In his article "The Right Way to Think About Software Design," Ted Nelson writes, "You face a screen littered with cryptic junk: the frying pan, the yo-yo, the bird's nest, the high-button shoe.

Or whatever. You must learn nonobvious aspects of a lot of poorly designed screen furniture and visual toys: what they actually do, rather than what they suggest" [Nelson 90]. To help make the meaning of icons in The Electronic Scrapbook clear, they have been given text titles. Combining modalities can enhance the power of each.

In the process of system development, I discovered that some tasks were better performed with text alone. This issue is discussed in section 5.3.1.

5.1.2 Local Methods

The first part of this programming task was to figure out how to display icons and how to drag them. Next, each icon needed to execute a different action when dropped on a segment. (For example, dropping the "indoors" icon on a segment asserts that the segment took place indoors; dropping the "oops" icon asserts that there is a problem with the quality of the video.) In the Common Lisp Object System (CLOS), methods are defined by object class. In order to make each icon execute a different function when dropped on a segment, it would be necessary to make every icon its own subclass. This approach is awkward if one has a large number of icons or if one wants to generate new icons automatically.

Therefore I chose instead to give each icon a "when-dropped-on-segment" slot which holds a function. This function is executed whenever the icon is dropped on a segment. Separate slots -- when-dropped-on-view, when-dropped-on-segment-editor, when-dropped-on-titled-icon, and when-dropped-on-trash -- determine what is done when an icon is dropped on other objects. Each titled-icon can have a fairly complex, unique behavior. In other words, individual members of a class can have unique *local methods*.

5.2 Video Controller

5.2.1 Device Control

For inspiration in the design of the video controller, I examined the controller Hans Peter Brondmo developed for The Elastic Tools [BD 89] and the Macintosh desk accessory "LD-V4200 Remote," which comes with the Pioneer LD-V4200 laserdisc Player. The resulting controller appears in Figure 5.2.

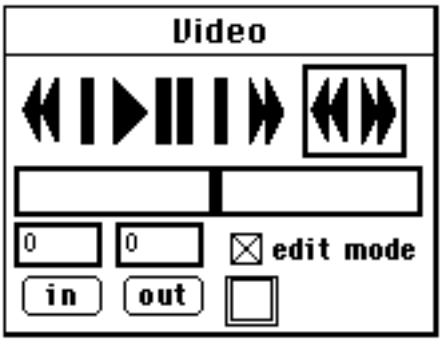

Figure 5.2: The video controller.

The video controller can manipulate multiple devices and device types. A separate controller is created for each device; each controller has a slot "my-device" which indicates the device controlled. The buttons on the controller call generic functions. For example, clicking on the step-forward button calls the function:

```
(step-forward <my-device>)
```

Each type of device has its own step-forward method. To add a new type of device, one merely needs to define a set of methods for how to operate that device.[1]

5.2.2 Paying Full Attention to the Picture

One useful feature of the video controller is the scan bar, which mimics the behavior of the shuttle knob on a standard video edit controller. When the bar is dragged to the right, the player goes forwards at increasingly high speeds, up to the maximum 240 frames/second. Dragging the bar to the left plays increasingly fast backwards. Dropping the bar at any point stops the player. Therefore, one need not look at the controller to locate a specific spot on the video; one can pay full attention to the video while moving the mouse from left to right.[2]

---

[1]Joseph Chung wrote the drivers for the serial port and the Pioneer laserdisc player.

[2]This approach was implemented at the Media Lab by Benjamin Rubin and Daniel Applebaum [Rubin 89]. Their work may be the first example of this interface technique.

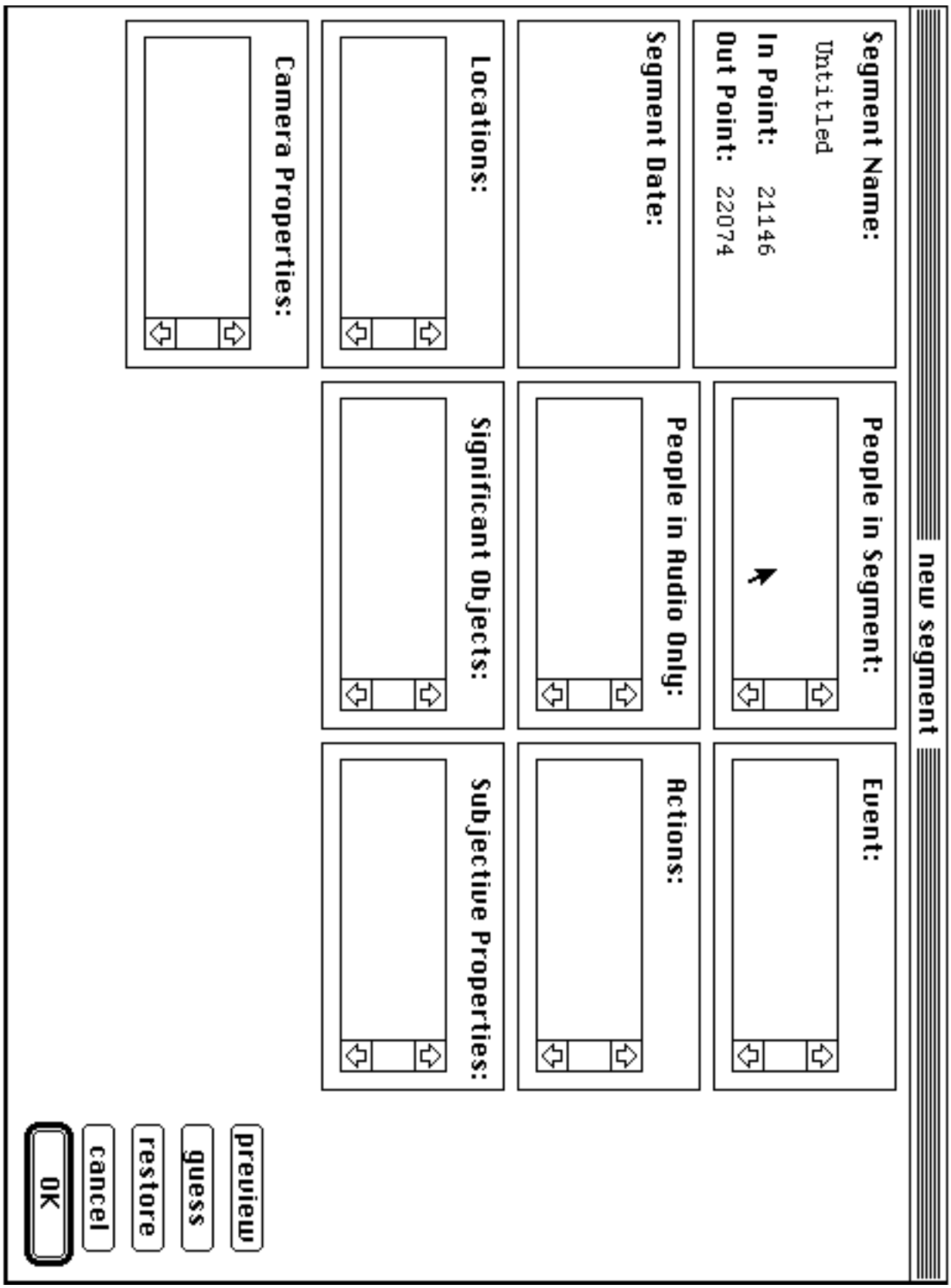

Figure 5.3: The segment-editor window.

5.3 The Segment-Editor Window

5.3.1 Text versus Graphics

In my original conception of the system design, the user would make a video segment by specifying in and out points and clicking on a "make-segment" button. The system would generate a default description of the segment based on what was already known about that video. The user would modify this default description by manipulating graphic icons.

The user needs some way to view the description of a segment. I chose to display an existing description in text rather than graphic form, because text is more compact and less ambiguous than icons. It was a logical step to allow the user to edit this text description directly. This was a slight shift in system design: there would be both text-based and graphical ways to describe a segment.

To my surprise, I discovered that the text-based interface was more pleasant to use than the graphical one. A large number of icons are needed to describe video, and it is difficult to organize them so that the correct icon can be found easily.[3] Furthermore, the repetitive action of dragging icons can be tedious. The text-based interface became the primary way to log information.

5.3.2 Panes: Avoiding Screen Clutter

The segment-editor window is divided into ten *panes,* each for a different property (see Figure 5.3). Clicking on a pane puts an editor for that property in the bottom corner of the window (see Figure 5.4).

A pane-based approach avoids the clutter and confusion of having many windows on the screen at once. This design is used on Symbolics computers and may be incorporated into the Apple Macintosh interface guidelines in the future.

[3]Marc Davis's research on cascading icons addresses this issue. [Davis 91]

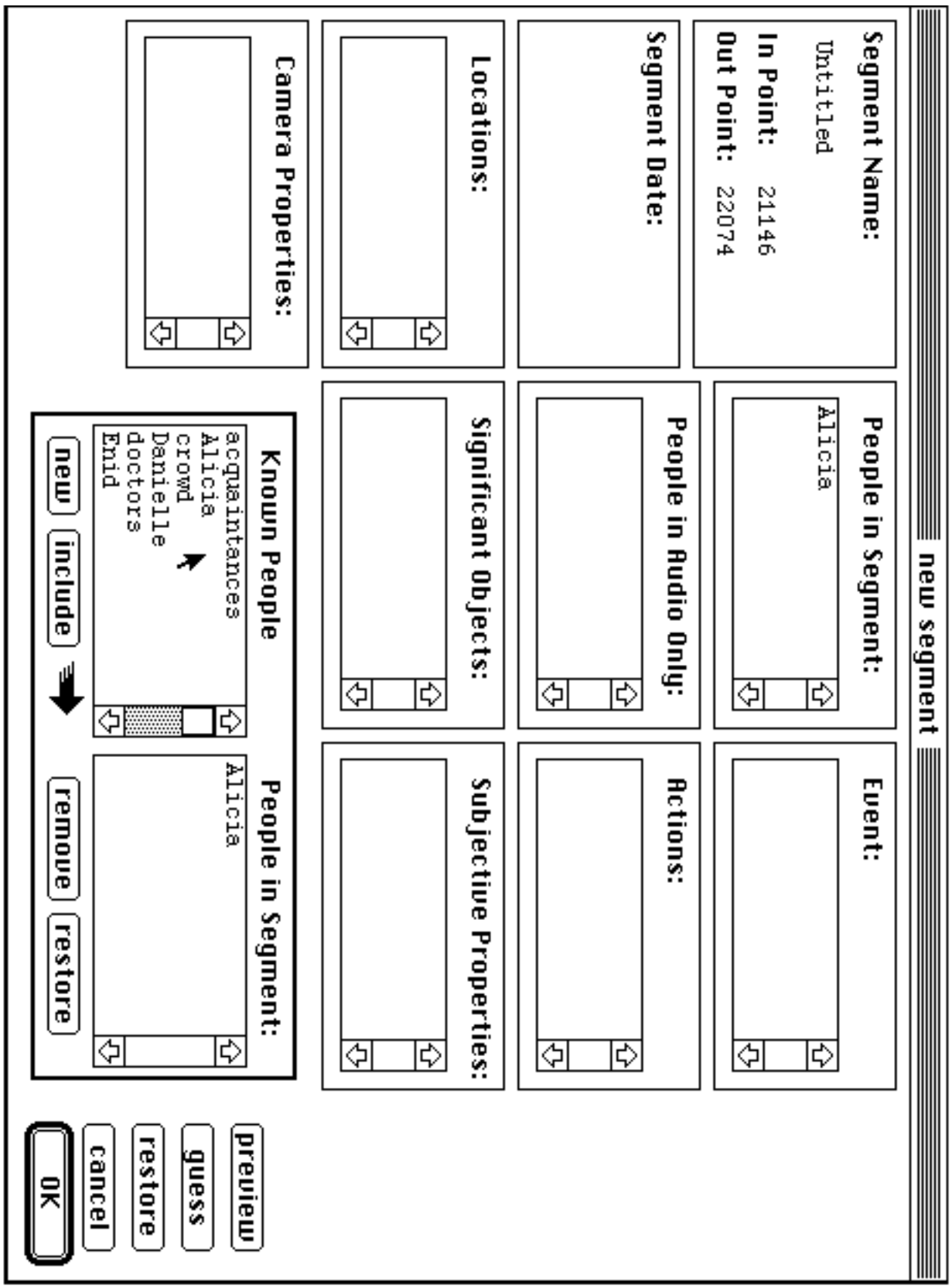

Figure 5.4: The segment-editor window, including an editor for the people pane.

5.4 List Browsers

5.4.1 A Sample List Browser

Each pane has its own editor. The editors for all panes (except the date and segment name panes) are specializations of the class list-browser. For example, the following code defines the "People in Segment" pane and its editor:

```
;; People  panes

(defclass people-pane (list-pane)
  ()
  (:default-initargs
    :pane-title "People in Segment:"
    :editor-left-title "Known People"
    :editor-right-title "People in Segment:"
    :view-size #@(170 90)
    :calculate-source-list-function
    #'(lambda (my-list-pane)
        (declare (ignore my-list-pane))
           (get-value *current-session* 'people-in-session))
    :calculate-initial-list-function
    #'(lambda (my-list-pane)
        (with-slots (my-arlotje-object) (view-window my-list-pane)
             (get-value my-arlotje-object 'people-in-segment)))
       :get-existing-value-function
    #'(lambda (my-list-pane)
        (with-slots (my-arlotje-object) view-window my-list-pane)
             (get-value my-arlotje-object 'people-in-segment)))
    :new-function
    #'(lambda (this-button)
        (make-instance
         'person-editor
         :in-video-source
         (slot-value (view-window this-button) 'my-source)
         :my-pane
         (slot-value (view-container this-button) 'parent-window)))
    :my-slot 'people-in-segment))
```

5.4.2 Lists of Options: Towards Logging Consistency

Note that all selection is done from lists; for example, one may either select a known person or add a person to the system. This helps to insure that things are referred to consistently; for example, it is less likely that the user will create separate units for "Alicia" and "Ali" as if they are two separate people.

5.4.3 Hierarchical Sorting: Visualizing the Underlying Representation

A birthday party is a kind of party. The event "birthday-party" inherits properties from the event "party." For example, all of the "likely-scenes" for a party -- guests arriving, eating, dancing, guests leaving, and the like -- are also

likely for a birthday party. Birthday parties also have additional scenes -- blowing out the candles on the cake, singing happy birthday, etc. Inheritance is a useful and efficient technique. For more discussion of issues of inheritance, see chapter 7.

The average home video artist will not be familiar with these concepts, but it is possible to make them understood. Lists in list-browsers can be sorted hierarchically. "Birthday-party" appears indented under "party" (see Figure 5.5). "Too-dark" appears indented under "problem-with-video."

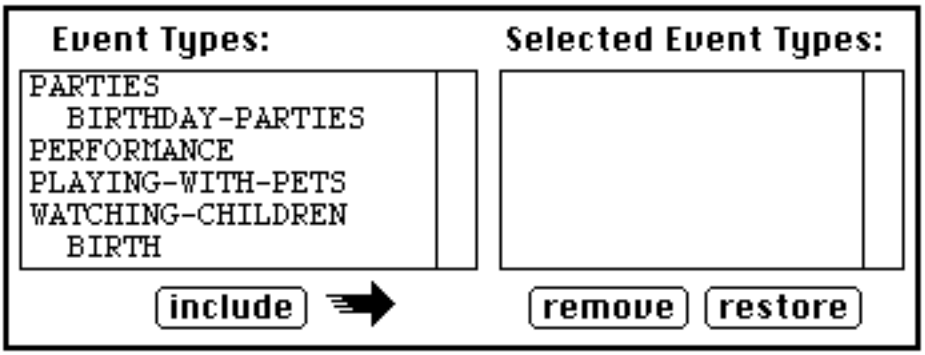

Figure 5.5: A hierarchically-sorted list-browser.

When an event is selected or double-clicked, the likely scenes for that event are listed under it. They are displayed in a smaller font, indented under the event, and sorted into their typical chronological order. For example, the alphabetically sorted list of events at a birthday party is confusing:

```
BLOWING-OUT-THE-CANDLES
CAKE-ARRIVING
CUTTING-THE-CAKE
DANCING-SCENE
EATING-SCENE
EATING-THE-CAKE
GIVING-PRESENTS
GUESTS-ARRIVING
GUESTS-LEAVING
OPENING-PRESENTS
LIGHTING-THE-CANDLES
PLAYING-GAMES
PUTTING-CANDLES-IN-THE-CAKE
SINGING-HAPPY-BIRTHDAY
THE-CAKE-SCENE
THE-MESS-AFTERWARDS
WATCHING-PERFORMER
```

However, the meaning of the hierarchically/chronologically sorted list is clearer:[4]

```
GUESTS-ARRIVING
DANCING-SCENE
EATING-SCENE
GIVING-PRESENTS
OPENING-PRESENTS
PLAYING-GAMES
THE-CAKE-SCENE
  PUTTING-CANDLES-IN-THE-CAKE
  LIGHTING-THE-CANDLES
  CAKE-ARRIVING
  SINGING-HAPPY-BIRTHDAY
  BLOWING-OUT-THE-CANDLES
  CUTTING-THE-CAKE
  EATING-THE-CAKE
WATCHING-PERFORMER
GUESTS-LEAVING
THE-MESS-AFTERWARDS
```

5.5 Video Segments

Once created, video segments are placed in the *segments window* (see Figure 5.6). The icon for a video segment is an empty picture frame. In the future, segments will be represented by miniature digitized frames of video. This will allow segments to be recognized more easily, enhance the visual appearance of the system, and make the look and feel of the system more like a traditional scrapbook.

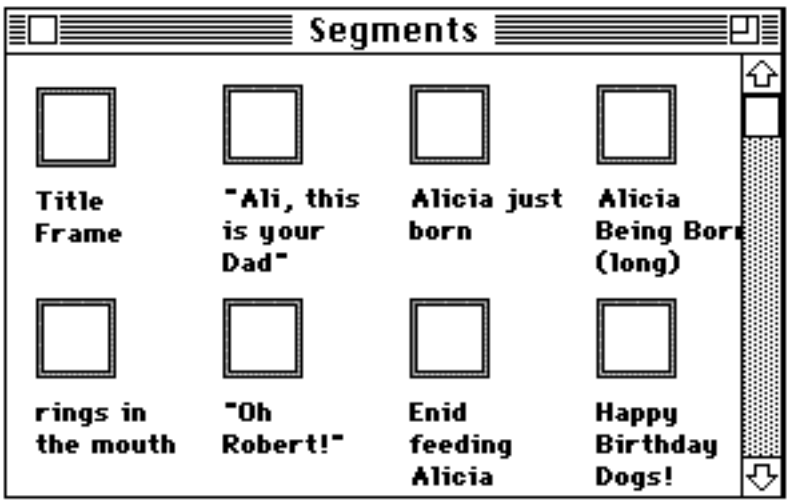

Figure 5.6: The segments window.

[4]Scenes with no usual order relative to eachother are sorted alphabetically.

Clicking on a video segment allows the user to drag it within the segments window, to a scrapbook page, to the trash, or the like. When the shift key is depressed, clicking on a segment plays back that video.

5.6 Scrapbook Windows

5.6.1 Spatial Implies Temporal Organization

Segments of video can be placed on pages in a "scrapbook window" (see Figure 5.7). Clicking on the "play" button replays all the video on the page in sequence -- left to right across rows, starting with the top row and moving down. In other words, the two-dimensional spatial organization of segments on the scrapbook page determines the one-dimensional temporal order of the video.

A scrapbook is made up of a set of pages forming a linked list, analogous to a stack of cards in Hypercard. In the future, it will be possible to generate text headings both manually and automatically.[5] This feature is not implemented yet.

[5]Each story model will have a method for generating an appropriate title for the page. See section 10.6.

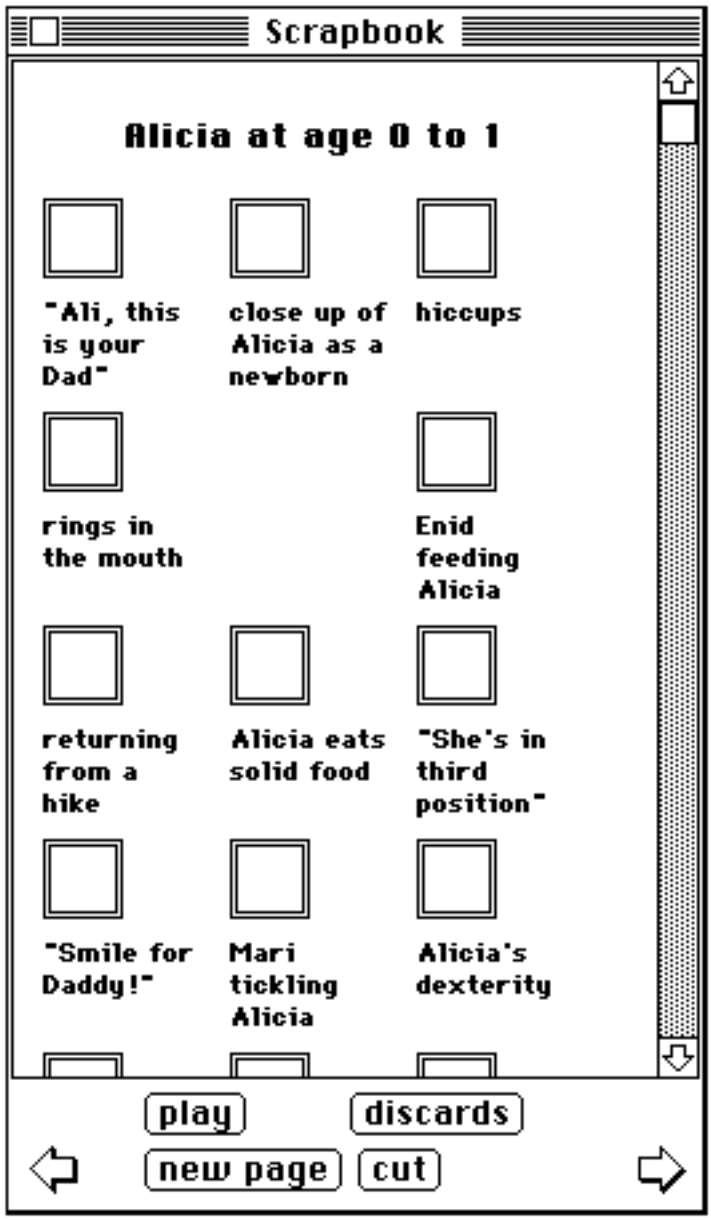

Figure 5.7: A scrapbook window.