7. Guessing

Logging video takes time. How much time depends on the nature of the footage and the amount of detail desired. The guess feature of The Electronic Scrapbook saves the user time by implementing a form of inheritance.

As was discussed in section 4.2, a simple hand-written log of two-and-a-half hours of home video took me seven hours to complete. This log included only basic information: people, people in audio only, location, significant objects, a rough description of camera properties, and a brief text description. It did not include a description of every object in the frame, what people are wearing, object movement, noises heard, a transcription of what was said, or precise camera information such as focal lengths, speed of pans, and the like. Furthermore, the pace of that particular video is fairly slow; shots are minutes long. In music-video style footage where shots are often only seconds long, it could take an order of magnitude more time to log an hour of video.

7.1. Level of Detail

There is a simple cost/benefit relationship between the amount of time it takes to log video versus the usefulness of the information obtained. It is important to find an appropriate level of detail. Do you need a description of every object in the frame? If you are a continuity supervisor for feature film production, you do [Lasky 90]. However, if you are a home video artist, then you probably do not.

The amount of description needed depends on the task at hand. Unfortunately, the task may be undefined or may change. Images can be reused for purposes that the logger could not anticipate.

Selecting an appropriate level of detail is not merely an issue of saving logging time. Large quantities of data can be unwieldy. The more data is present, the more difficult it often is to find the important elements. Political consultants know this problem best; the goal in preparing a memo for a politician is to summarize all the salient points in one page. Computer systems can help to solve this excess-data problem by organizing information such that the key features are highlighted, but more information is available on request.

7.2 Inheritance

7.2.1 Event-based Inheritance

My family took a trip to visit a pumpkin patch in October of 1990. All of the video from that family outing has certain properties in common: it is all from the same date, location, and event. The people present are also generally the same. Individuals appear in some shots, but not others; however, on the whole, Mari, Alicia, and Danielle are in the video and Robert is narrating.

One efficient way to log this footage is to create a long video segment for the entire event with this basic description. Shorter segments describing parts of the video more precisely can then inherit the outer description. Therefore, the user doesn't need to reenter the basic properties for each sub-segment.

The default description can be edited for individual segments. One danger is that a user may forget to remove elements which are not present. For example, if one included the fact that Robert is present in audio only in the pumpkin patch video, it is easy to forget to remove that property if he doesn't happen to be speaking in a particular shot.

7.3 Making Inferences Apparent

It was important to me to make what the system is doing clear to the user. Rather than having inherited descriptions automatically appear, I decided to create a "guess" button on the segment-editor window.

In the original system design, clicking on the guess button added inherited values to the segment-editor window. This was charmingly mysterious -- a largely correct description simply appears, creating an impression that the system is "smart." However, the user might wonder how the system derived a guess. This interface did not provide that information.

To make the system's actions apparent, the guess button was changed; it now creates a window full of segments which overlap with the current segment and segments which are similar to it (see Figure 7.1). The function which finds similar segments will be discussed in section 7.4. To copy a description, the user picks up a segment which seems most similar to the current segment and drops it onto the segment-editor window.

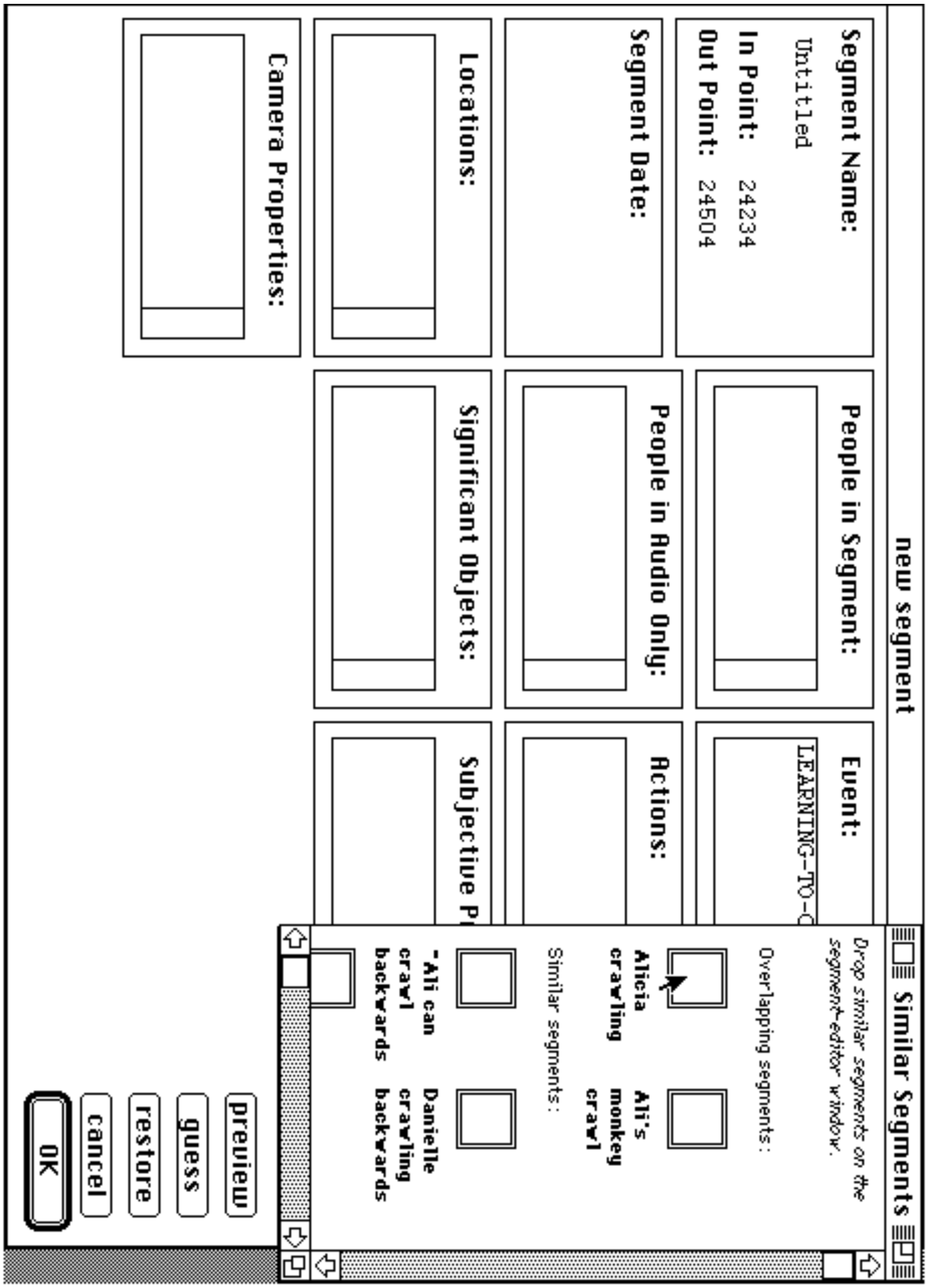

Figure 7.1: A segment-editor window and its corresponding "similar segments" window.

All of the values of the dropped segment are copied except for the segment name and in and out points. The date is also handled in a more complex way. A date logged can only be made more specific. A few examples of this are listed in Table 7A.

Table 7A: Copying Dates

Pre-existing Date of Target Segment	Date of Segment Being Copied	Resulting Date of Target Segment
none	12/21/65	12/21/65
1965	12/21/65	12/21/65
12/65	12/21/65	12/21/65
5/66	12/21/65	5/66
12/21	12/21/65	12/21/65

7.4 Similar Segments

If the user gives a partial description of a segment, it is possible to find segments which match that partial description. Similarity is determined by a scoring polynomial.

7.4.1 The Score Function

How does one compare apples and oranges? How is it possible computationally to say that "a shorter video segment is better, but don't make it too short, and it would be nice if took place outdoors." One approach is to use a scoring polynomial. In the scoring function used by the Electronic Scrapbook, the preceding statement would look like:

```
(score-segments list-of-segments-to-search
   '((shortest-weighted 100 4)
     (longer-than 30 500)
     (locations-of-segment outdoors 4)))
```

The last number in each criterion is a numerical weight indicating its relative importance. If a segment is outdoors, then it will get 4 points. If it longer than 30 frames, it will get 500 points. Most segments will satisfy that criterion. If this function is used to choose among segments, this means that all segments that are shorter than 30 frames will probably be eliminated.

Shortest-weighted returns the target segment length (the second argument) divided by the actual segment length, times the weight (the third argument). (See the code below). In the example (shortest-weighted 100 4), a segment which is 100 frames long will get 4 points. One which is 200 frames long will get 2 points. One which is 50 frames long will get 8 points. One which is 1 frame long will get 400 points. The high longer-than weight was selected to offset the high scores given to short segments, since fragments are not desirable. The two criteria work together to say "make it short, but not too short."

Any slot-value pair may be used as a scoring criterion. A list of slot-value pairs may also be given a single weight. The weight is awarded once if any one of the criteria is satisfied. Other criteria include shorter-than and longest-weighted.

The score-segments function calls the `score` function on each segment. The code for `score` is listed below:

```
(defun score (the-segment list-of-criteria)
  (let ((result 0)
        (unknown (list )))
    (dolist (criterion list-of-criteria)
      (let ((criterion-type (first criterion))
            (arg (second criterion))
            (weight (if (= (length criterion) 3)
                        (third criterion)
                        1)))
        (cond
         ((equal criterion-type 'shorter-than)
          (if (< (segment-length the-segment) arg)
            (setf result (+ result weight))))
         ((equal criterion-type 'longer-than)
          (if (> (segment-length the-segment) arg)
            (setf result (+ result weight))))
         ((equal criterion-type 'shortest-weighted)
          (setf result
                (+ result
                  (* (/ arg
                        (segment-length the-segment))
                     weight))))
         ((equal criterion-type 'longest-weighted)
          (setf result
                (+ result
                  (* (/ (segment-length the-segment) arg)
                     weight))))
         ((slotp criterion-type)
          (if
            (segment-satisfies? the-segment criterion-type arg)
            (setf result (+ result weight))))
         ((and
           (listp criterion-type)
           (slotp (first criterion-type)))
          (let  ((temp-result nil))
            (block check-slots
              (dolist (a-slot criterion-type)
                (if (segment-satisfies? the-segment a-slot arg)
                  (progn
                    (setf temp-result T)
                    (return-from check-slots)))))
```

```
            (if temp-result (setf result (+ result weight)))))
         (T
          (pushnew criterion-type unknown)))))
    (if (> (length unknown) 0)
      (dolist (c unknown)
        (format t "~&Unknown criterion: ~s" c)))
    result))
```

7.4.2 Selecting Similar Segments

To select similar segments, the system constructs a set of criteria to pass to the score function based on the existing description of the segment. Each criteria has its own default weight. These defaults are listed in Table 7B.

Table 7B: Default Weights for the Similar-Segments Function

Criterion	Default Weight
events	20
actions	10
people	4
month	3
year	3
locations	2
significant objects	2
day	1
people-in-audio-only	1
camera-properties	0
subjective properties	0

These weights were tuned to achieve useful matches. Events are particularly useful in matching segments. For example, two shots of the event `learning-to-swim` are more likely to be similar than two shots containing the object `water`.

The weights are keyword arguments to the function similar-segments. Ideally, these weights should be changeable by the user. An interface to allow this will be implemented in the future.