8. Story Models

This chapter shows how a simple "story model" can be made from combinations of functions, and how something approaching a narrative can be made with combinations of these simple story models. The results of sample story models are presented and evaluated.

8.1 Database Searches

Once a user has a database of annotated video footage, what can it be used for? The most obvious application is to perform simple database searches. For example, one might want to search for all of the shots containing a particular person, object, location, and the like. The function `segments-with-property` takes any slot-value pair. For example, one could search for:

```
(segments-with-property 'significant-objects 'cake)
```

The optional keyword argument `:segments-to-search` defaults to all known segments of the current session. This can be used to efficiently perform a logical "and." For example, the following statement finds all segments of cake at a picnic:

```
(segments-with-property 'events-of-segment 'picnic
   :segments-to-search
     (segments-with-property 'significant-objects 'cake))
```

Other simple search functions include `segments-without-property` and `segments-with-property-or`. The following statement finds all segments either at the beach or containing water which do not have a problem with the video quality:

```
(segments-with-property-or
   '((significant-objects 'water)
     (locations-of-segment 'beach))
   :segments-to-search
     (segments-without-property
         'camera-properties 'problem-with-video))
```

8.1.1 Checking the Knowledge Hierarchy

Note that searching for `problem-with-video` will also return subsets of `problem-with-video` such as `too-dark`, `out-of-focus`, and the like. Similarly, searching for `toys` will return `dolls`, `bubble-stuff`, and `frisbees`. The system checks for subsets unless this option is disabled by setting the keyword argument `:check-subsets` to `nil`. This implements a useful kind of inheritance.

8.1.2 Searching for People

In videos of children, it is often desirable to look for all of the segments of a child at a particular age. It is also convenient for the system to convert between a person's name ("Alicia") and the unit name (`humans.1`). The function segments-with-person-at-age takes the arguments months, days, years, to-months, to-days, and to-years. If the end date is not specified, then it is assumed to be one plus the most-specific start date specified. For example, the following statement finds all shots of Alicia at age 0 to 1:

```
(segments-with-person-at-age "alicia" :years 0)
```

Whereas this statement finds all shots of Alicia at age 18 to 19 months:

```
(segments-with-person-at-age "alicia" :years 1 :months 6)
```

Ages are determined by examining segment dates and the individual's birth date. Variable month lengths and leap years are properly accounted for.

Writing lisp code to perform simple searches and combinations of searches is easy; however, designing and writing interface code to make this functionality available to the user is a challenge. This task is left for future work.

8.2 Simple Story Models

In the current data set, a search for all segments with Alicia at age 0 to 1 years returns 36 video segments. This list is useful as raw source material for making a more polished presentation; however, it not pleasant to watch from start to finish. The list of shots is in random order, contains segments that have technical problems (such as dark video or garbled audio), contains sensitive subject matter (such as a child being born), and contains overlapping segments.

In the Electronic Scrapbook, a simple `story-model` consists of a database search function and a set of criteria for how to process the raw list. The goal is to transform a long, unorganized list into something that is enjoyable to watch in sequence.

8.2.1 The Basic Search

A `story-model` is an Arlotje object with slots that hold functions. The function `run-story-model` executes those functions and returns a list of segments:

```
(defun run-story-model (my-story-model &rest args)
  (if (compound-story-model-p my-story-model)
    (apply 'run-compound-story-model my-story-model args)
    (progn
      (if (not *running-compound-model*)
        (setf *discards* nil))
      (let* ((initial-list
              (apply
               (get-value my-story-model 'story-model-function)
               args))
             (result-list  nil)
             (sort-function
              (get-value my-story-model 'sort-function))
             (after-sort-function
              (get-value my-story-model 'after-sort-function)))
        (setf result-list
              (remove-segments
               initial-list
               (get-value my-story-model 'remove-criteria)
               (get-value my-story-model 'remove-threshold)))
        (setf result-list
              (remove-overlaps
               result-list
               (get-value my-story-model
                          'remove-overlaps-criteria)))
        (if (and sort-function
                 (not (failurep sort-function)))
          (setf result-list (apply sort-function result-list args)))
        (if (and after-sort-function
                 (not (failurep after-sort-function)))
          (setf result-list
                (apply after-sort-function result-list args)))
        result-list))))
```

The slot `story-model-function` holds a database search algorithm used to generate the initial list of segments to be manipulated.

8.2.2 Discards

Any segments later removed from this initial list are added to the list `*discards*`, which is reset at the start of a new `story-model`. Discarded segments can be examined by the user by clicking on the “discards” button (see Figure 8.1). The user can recover segments from the discards window simply by dragging them back into the scrapbook window. Segments can also be rearranged or discarded by being dragged to the trash window. All decisions made by the system can be changed; creative control remains in the hands of the user.

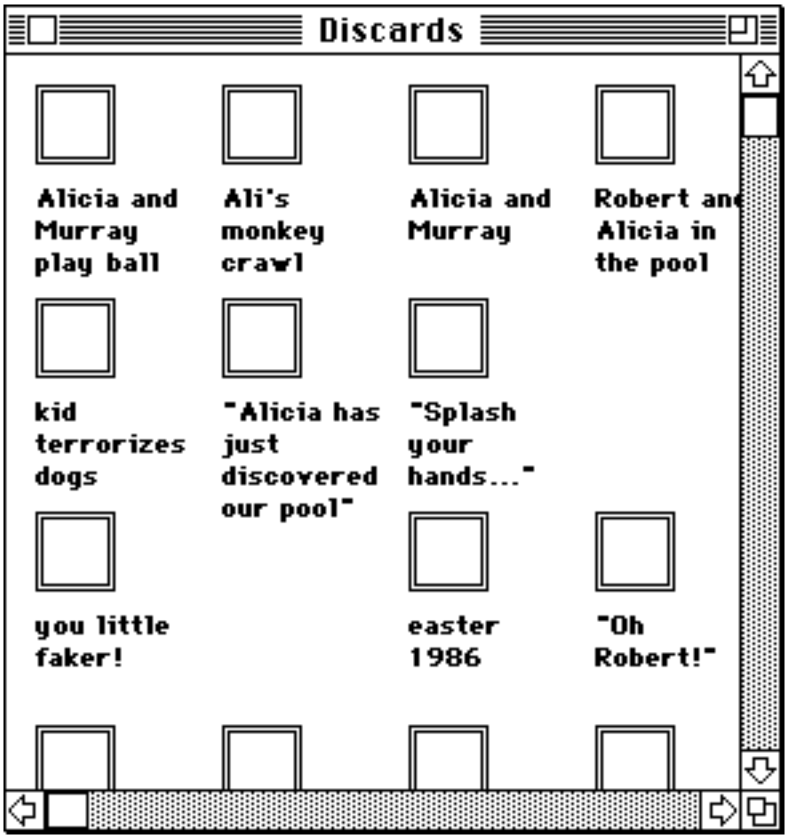

Figure 8.1: The discards window.

8.2.3 Filtering

Segments with undesirable properties can be removed by the function `remove-segments`. Each `story-model` has a slot `remove-criteria`. These criteria are passed on to the function `score`.[1] All segments with a score higher than the `remove-threshold` are discarded. The definition of the slots `remove-criteria` and `remove-threshold` and their default values are:

```
(define-unit remove-criteria
  (works-like 'prototypical-slot)
  (makes-sense-for 'story-model-objects)
  (value-defaults-to
     '((camera-properties major-problem-with-video 101)
       (subjective-properties-of-segment
          sensitive-subject 101)
       (camera-properties minor-problem-with-video -35)
       (camera-properties problem-with-video 51)
       (longer-than 3000 51)
       (longer-than 4000 51))))
```

[1]See section 7.4.1 for more discussion of the score function.

```
(define-unit remove-threshold
  (works-like 'prototypical-slot)
  (makes-sense-for 'story-model-objects)
  (value-defaults-to '100))
```

These defaults can be paraphrased as "discard anything which is sensitive subject matter. A long segment is bad and a very long segment is very bad. Discard anything with a major problem with the video; ignore minor problems with video." Since these criteria combine, a segment which is long but not very long (3000 to 3999 frames) will be kept unless it also has a problem with the video.

These criteria can be changed for individual `story-model`s. For example, in the `important-firsts story-model`, the video quality criteria are eliminated -- if one is looking for shots of a child's first steps, it is less important if the video is dark or out of focus.[2] Similarly, if one wants to watch a child being born, then sensitive subject matter should not be filtered out. The stylistic criteria are content-dependent.

8.2.4 Removing Overlapping Segments

It is undesirable to show the same video multiple times in the same story, unless one has deliberately chosen to do so. The system can help the user by detecting overlapping segments and using the scoring function to choose between them. The default is to choose the shortest segment which is not too short. The scoring criteria are kept on the `remove-overlaps-criteria` slot of the `story-model`:

```
(define-unit remove-overlaps-criteria
  (works-like 'prototypical-slot)
  (makes-sense-for 'story-model-objects)
  (value-defaults-to '((shortest-weighted 30)
                        (longer-than 30 31))))
```

This feature is quite useful, but the length-based selection criteria are unsatisfying. Consider, for example, this sequence:

(In-point one.)
Mari: Let's stand up. [Help's Alicia to stand] Aaaw! We're so strong. We're so big and cute!
Rose: Is that a pinafore or part of the dress?
Mari: It's a pinafore. It's part of the dress. Give Mommy your hand.
(In-point two.)
Robert: Don't help her stand. Put her down.
Mari: [Holds Alicia by one hand.] She's in third position. [Everyone laughs.] Look at her feet!

[2]See section 8.5 for more discussion of the important-firsts story model.

Robert: That's a riot.
[Alicia falls.]
Mari: Oh, look at that face.
[Alicia begins to cry.]
(Out-point two.)
[Mari picks up Alicia. Camera zooms in close.]
[Alicia stops crying as she is picked up.]
Mari: Smile for Daddy! [Alicia smiles.] That is a big smile. [Mari tickles Alicia, lifting her up and down. Alicia giggles.] You little faker! You little faker!
(In-point three.)
[Mari continues to tickle Alicia. Both smile and laugh.]
(Out-points one and three.)

The long version of this segment tells a story. The impact of the story is different if one begins with Robert saying "don't help her stand" and ends with Alicia crying. On the other hand, if one takes merely the shot of Alicia and Mari laughing and giggling, then the meaning is entirely different -- the story of Alicia trying to stand is eliminated.

The close up of Mari tickling Alicia is a beautiful shot. If one used subjective judgements of quality to select video, then that last segment would be chosen over the others. However, that criterion is as unsatisfying as segment length.

The complete segment is 55 seconds long. Substantially different meanings can be made from this video, depending on exactly which chunk of it is used. How is a computer to choose between them? One might first ask, how does a person choose between them?

8.2.5 Sorting

After overlapping segments are removed, the system sorts the remaining list. The default sort function is chronological:

```
(define-unit sort-function
  (works-like 'prototypical-slot)
  (makes-sense-for 'story-model-objects)
  (value-defaults-to
   #'(lambda (list-of-segments &rest args)
       (declare (ignore args))
       (sort (copy-list list-of-segments) #'in-point-less-than-p))))
```

In the `kids-at-same-age story-model`, the video is sorted by the age of the children. Instead of showing Alicia and Danielle in 1986, it shows them each at the same ages.[3]

[3] This story model and the important-firsts story model (see the next section) were designed with positive goals in mind: "wouldn't it be nice to see both kids just learning to crawl?" These

```
(define-unit kids-at-same-age
  (member-of 'story-model-objects)
  (story-model-function
   #'(lambda (&key
              (list-of-kids nil)
              &allow-other-keys)
       (let ((list-of-criteria (list )))
         (dolist (child list-of-kids)
           (pushnew (list 'people-in-segment
                          (human-name-to-unit child))
                    list-of-criteria))
         (segments-with-property-or list-of-criteria))))
  (sort-function
   #'(lambda (list-of-segments &key
                               (list-of-kids nil)
                               &allow-other-keys)
       (age-sort list-of-segments list-of-kids
                 :up-to-second-oldest T)))
  (after-sort-function
   #'(lambda (list-of-segments &key
                               (list-of-kids nil)
                               &allow-other-keys)
       (person-pattern
        list-of-segments list-of-kids
        '((camera-properties major-problem-with-video -101)
          (subjective-properties-of-segment sensitive-subject -101)
          (camera-properties minor-problem-with-video 35)
          (camera-properties problem-with-video -51)
          (longer-than 3000 -51)
          (longer-than 4000 -51)
          (shortest-weighted 30)
          (shorter-than 30 -51))))))
```

The results of this `story-model` will be discussed in section 8.5.

8.2.6 One Last Pass: the After-sort Function

It is sometimes desirable to process a list further after it has been sorted. This is performed by the `after-sort-function`. The default is for no `after-sort-function` to be used; however, the code in the previous section provides an example of how it can be used. After the video is sorted by the age of the children, it is desirable to assure that the list alternates between children. This is accomplished by the **`person-pattern`** function, which works rather like the `remove-overlaps` function. When a sequence of more than one shot of the same child is detected, a scoring polynomial is used to choose between them.

---

models have the unfortunate side effect, however, of encouraging the viewer to competitively compare children's accomplishments.

8.3 Compound Story Models

A `story-model` can be made up of other `story-model`s. With this feature, one can begin to construct narratives. The `important-firsts story-model` is made up of key events in a child's development:

```
(define-unit important-firsts
  (member-of 'compound-story-models)
  (components '(kids-birth kids-bottle-fed kids-learning-to-sit-up
               kids-burped kids-learning-to-crawl-backwards
               kids-learning-to-crawl kids-first-solid-food
               kids-learning-to-stand-up kids-learning-to-walk)))
```

Each of the component `story-model`s can themselves be made up of other `story-model`s; any level of depth is possible.[4]

8.4 Results of a simple story-model: `child-at-age`

In the current data set, a simple database search for all shots of Alicia at age 0 to 1 years returns 36 segments:[5]

```
Alicia Being Born (long)
"Ali, this is your Dad"
Alicia just born
close up of Alicia as a newborn
Mari nursing Alicia
hiccups
Alicia and the rings (long)
rings in the mouth
"Oh Robert!"
Enid feeding Alicia
Christmas 1985
returning from a hike
Alicia eats solid food
Easter 1986
you little faker!
"Smile for Daddy!"
"She's in third position"
Mari tickling Alicia
Alicia's dexterity
Alicia discovers the pool (long)
"Splash your hands..."
"Alicia has just discovered our pool"
"Swim to Daddy!"
"Ali can crawl backwards"
```

---

[4]The `run-story-model` function is recursive. If a `story-model` is compound, then `run-story-model` calls itself on each of the parts.

[5]The titles of the segments were subjectively, rather than systematically selected -- I picked something that would remind me of the segment. Consequently, these titles may not be meaningful to the reader. There is not enough space here for complete segment descriptions; the titles should convey a general sense of the results.

```
kid terrorizes dogs
Where are the puppies?
"Two trusty dogs."
Robert and Alicia in the pool
"Swim swim swim!"
Alicia and Murray
"Wanna go for a ride?"
Ali's monkey crawl
Alicia crawling
Alicia and Murray play ball
"This game is getting tiring."
Alicia squeezes cake
```

The total running time of the list is 18.6 minutes.

The `story-model` “child-at-age” applies the default `story-model` filtering and sorting functions to the same base search function:

```
(define-unit child-at-age
  (member-of 'story-model-objects)
  (story-model-function #'(lambda (&key
                                   (kid nil)
                                   (years nil)
                                   (months nil)
                                   (days nil)
                                   (to-years nil)
                                   (to-months nil)
                                   (to-days nil)
                                   &allow-other-keys)
                            (funcall 'segments-with-person-at-age
                                    kid
                                     :months months
                                     :years years
                                     :days days
                                     :to-months to-months
                                     :to-years to-years
                                     :to-days to-days))))
```

Running this `story-model` with the same arguments, Alicia at age 0 to 1 years, returns a list of 20 segments:

```
"Ali, this is your Dad"
close up of Alicia as a newborn
hiccups
rings in the mouth
Enid feeding Alicia
returning from a hike
Alicia eats solid food
"She's in third position"
"Smile for Daddy!"
Mari tickling Alicia
Alicia's dexterity
"Swim to Daddy!"
"Ali can crawl backwards"
Where are the puppies?
"Two trusty dogs."
```

```
"Swim swim swim!"
"Wanna go for a ride?"
Alicia crawling
"This game is getting tiring."
Alicia squeezes cake
```

Four segments are eliminated by the remove-segments function and twelve are eliminated by remove-overlaps. The total running time of the resulting list is 5.2 minutes, 70% shorter than the original.

The result is surprisingly coherent -- it begins with Alicia being born and progresses to her first birthday party. The shots selected are reasonably representative.

There is, however, one jump cut. The shots "Smile for Daddy!" and "Mari tickling Alicia" are both close ups of Mari holding Alicia, and the transition is awkward. To prevent this, the system could be made to check for two shots together that have the same focal length, location, and people present. However, without precise camera angle information, this might eliminate acceptable transitions.

8.5 Comparison of two approaches: `kids-at-same-age` and `important-firsts`

Video does not always need to be presented chronologically. For example, it is interesting to watch a group of siblings at the same ages rather than the same years -- instead of showing Alicia and Danielle in 1990, one can show Alicia and Danielle at 18 months old. As discussed in section 8.2.6, the `kids-at-same-age` model sorts video by the age of the kids. In cinematic terms, this uses the *parallel action* model of editing.[6] The results of the `kids-at-same-age story-model` for Alicia and Danielle are:

```
"Daad!  Enough!"
close up of Alicia as a newborn
Danielle and Baby Mickey
returning from a hike
Danielle crawling backwards
"Smile for Daddy!"
"What's so funny?"
"Swim to Daddy!"
"Danielle, come feed the goats."
```

[6]Parallel editing is a fundamental cinematic technique. The first known example is a 1903 black and white, silent, short by Edwin S. Porter called *The Life of an American Fireman*. See [Cook 81].

An alternative, more knowledge-based approach is to show the children at key stages in their development such as learning to crawl or eating their first solid food. This is done by the `important-firsts story-model`:

```
close up of Alicia as a newborn
Danielle as a newborn
"Ali can crawl backwards"
Danielle crawling backwards
Alicia crawling
Danielle crawling
Alicia eats solid food
"She acts like she eats it all the time!"
```

This presentation has more structure and comes closer to being a "narrative." The difference highlights the benefits of a knowledge-based approach. Selection of segments in this model is made more often based on content than stylistic criteria, and the result has greater coherence.

The `kids-at-same-age` model for this data has nine segments and is 2.1 minutes long. It is interesting to note that the `important-firsts` model has fewer segments but is longer: it has eight segments and is 2.9 minutes long. This is indicative of the fact that the `important-firsts` model makes less use of stylistic criteria. The stylistic criteria are weighted to give preference to shorter segments.

The `important-firsts` model does, however, contain a few dark shots. The video quality criteria were removed from this `story-model` (and from its component `story-model`s.)[7] If one is looking for a shot of Junior's first steps, it is less important that the shot might have technical problems -- it is an important moment. On the other hand, if one is merely looking for any shot of junior at age 3 months, then video and audio quality are useful selection criteria.

It could be argued that the greater randomness of the `kids-at-same-age` model is more interesting than the more formulaic `important-firsts` model. Certainly both ways of telling this story have merits. However, randomness is a poor substitute for inventive combination.

A higher goal would be for the system to make "creative" suggestions. However, the term "creativity" is as ill-defined as the term "intelligence." Humans often define "creativity" and "intelligence" as everything computers can not do. To avoid using these terms, one might say that the system will always benefit from additions to its vocabulary of ways to combine video based on both stylistic and semantic criteria.

---

[7]See section 8.3 for more discussion of compound story models.