10. Future Work

10.1 More Data

10.1.1 Birthday Parties

To date, the system has been tested with only one data set. It would be beneficial to see how it performs on a variety of source material.

The idea of editing video of a child's birthday party was used as a thought experiment during the process of system design. It would be interesting to see how the system performs on this structured information.[1]

10.1.2 Scaling

It would also be beneficial to work with a much larger data set. Some of the power of the underlying knowledge representation is currently unnecessary, because half an hour of video is manageable without it. The larger the volume of data, the more the system's design will be tested.

10.1.3 Storage, Memory, and Speed Issues

There are technical limitations which currently prevent using a larger amount of data. Right now the system is limited to half an hour of video, the amount that will fit on one side of a laser disc. If the system is converted to digital video, then the limitation will be the amount of available hard drive storage space.

The amount of memory is also an issue. All data is currently loaded into memory. When lisp runs out of memory, it stops to garbage collect. Garbage collection takes approximately 30 seconds on a Macintosh IIx computer. With 30 minutes of video data loaded, the system currently stops to garbage collect on every significant action. For example, it garbage collects every time the user clicks on the "guess" button or runs a `story-model`. Using a faster computer with more memory can alleviate this problem. Ephemeral garbage collection,[2] which has just recently been implemented in MCL, will also ease this problem.

Significantly less memory would be required if information were stored in a relational database rather than in memory. However, searching and inferencing mechanisms access a large amount of data. Repeatedly getting

---

[1] See section 4.1 for an explanation of the use of the phrase "structured information."

[2] With ephemeral garbage collection, certain kinds of memory can be freed for reuse in the background without interrupting the user.

data from disk would be slow. Therefore, it is desirable to keep as much information in memory as possible.

A number of improvements can be made to the software to increase its speed. It seems likely that the system is being slowed down by constantly recreating list browsers and hierarchically sorting their contents. These could instead be recycled. The system would need a mechanism for detecting when a list needs to be updated. For example, the system currently makes a new hierarchically-sorted list of possible locations every time a segment is created. It would be preferable to use the same list over and over, and recalculate it only after a new location has been created.

However, it is only a speculation that remaking list browsers is slowing the system down significantly. The best way to improve the speed of the code is to run a speed-checking program and find out exactly which functions are using up most of the system's time.

The underlying knowledge representation language is part of the culprit. The author, Kenneth Haase, promises that his next version of Arlotje will be faster.

10.2 Digital Video

The system was designed to be used with digital video. With the release of Apple Computer's QuickTime™ digital video software, this has become more practical. Preliminary work on this conversion is under way.

10.2.1 Still Frames to Represent Segments

First and most simply, it would be desirable to have each video segment represented by a digitized still frame of video. The icon for video segments is an empty picture frame; this frame is meant to be filled (see Figure 10.1).

Figure 10.1: A video segment.

10.2.2 A Linear Representation of the Source Material

If the user is to digitize video from tape, then it will be necessary to write a driver for the tape deck and an interface for digitizing chunks of video. This interface would benefit from a graphical, linear representation of the source tape.

Users could mark in and out points of properties on this linear representation, and the properties would be inherited by all video segments in the selected range. For example, if entering the park is marked at frame 20100 and leaving the park is marked at frame 24500, then any video chunks selected from that range are assumed to be in the park.

This is analogous to the process currently used in the Electronic Scrapbook in which a user makes one segment for the entirety of our trip to the park, and all overlapping segments can inherit that description.[3] In a linear timeline representation, however, each property can be inherited separately. Furthermore, the process of inheritance can be more easily visualized by the user.

If a great deal of description is to be done on a timeline, then it will get too crowded to read. Ideally, the timeline should be expandable to multiple layers or "tracks." For example, one track could show who is present and another the actions being performed. The audio could have a separate representation from the video. Particular tracks could be combined, separated, shown, or hidden [Davis 91].

10.3 An Interface to Story-Models

There is not yet an interface to story-models. Story-models are currently executed by typing lisp code. Designing an interface to story-models is essentially an issue of visual programming [Shu 88]. Visual programming is an entire field of research. What would such an interface look like? Some preliminary ideas are presented here.

First, the user needs a list of available story-models. Names of story-models will not be sufficient to communicate what each model does. It would be useful for a text description to pop up when a model is highlighted.

A dialog box for simple story-models must show the functions used by the model. How are these to be displayed? Scoring criteria such as `'((shorter-than 300 50) (subjective-properties sensitive-subject-matter 101))` are easier to understand than lisp code, but still require significant explanation. Is

[3] See section 7.2.

there any simpler way to present this information and give the user the ability to modify it?

An interface is needed to help the user to construct the base search function. Different levels of assistance can be provided to the user. The user could be required to write lisp code. Alternatively, the user could be provided with a database query language which would then be transformed into lisp code. At an intermediate level of assistance, the user might be guided through the process of constructing a query with carefully designed dialog-boxes and menus. Ideally, however, the user should be provided with a graphical programming language in which queries can be easily constructed and understood.

The task of making an interface to story-models raises fundamental questions of human-computer interface design. A simple interface requiring significant knowledge on the part of the user will be constructed in the near future. The task of designing a visual programming language is an open research issue.

10.4 A Graphical Interface to the Underlying Knowledge Representation

The need for a graphical representation to the underlying knowledge representation was discussed in section 6.1.3. A graphical representation would enable the user to visualize the knowledge base and watch inferences being made. A graphical representation is also needed to log complex data structures such as actions. For example, the action "walking" should be given a person, a starting location, an ending location, and perhaps other modifiers. Consider representing a video clip in which Cynthia walks briskly from the beach blanket to the water. One might further need to represent the fact that the blanket is on the sand, which is part of the beach. The representation becomes more complex if one wants to indicate why she walked to the water--to rescue Amy's ball from the waves.[4] Such a representation has many levels, and is best displayed and manipulated graphically.

Three different graphical representations are envisioned. The first is a planar representation in which units are connected by arrows showing their relationship (see Figure 2.1). The second is a timeline showing how the linear source material changes over time (see section 10.2.2). A timeline could also be used to represent events in real time. Each of these schemes meets different needs. How they can be combined into a coherent whole is an open question.

---

[4]The issue of how much knowledge representation is necessary was discussed in sections 2.4 and 6.6.1.

Future versions of Arlotje will come with a graphical interface [Haase 91]. This will reduce the amount of interface coding necessary to construct an application with Arlotje, and it will also help programmers to understand its workings.

10.5 Making the Interface Reconfigurable

The interface and underlying representation could be made reconfigurable by the user. This would make the logging system useful for new subject domains. For example, if one were making a scrapbook about the zoo, one might want to add a pane to the segment-editor window for animals, separating them from objects or people. An interface to enable the user to make that kind of change could be made which would not require much knowledge on the part of the user. Object-oriented programming makes this possible; one can define a new instance of `list-pane`, and the system will do all the work of creating a corresponding `list-browser` and the like.

It is perfectly feasible for the user to redefine the base ontology. The user could select which ontology to load: "home video 1," "home video 2," "zoo," "vacation 1," and the like. The challenge comes in the design of story-models. Story-models currently incorporate specific domain knowledge. The model `important-firsts` looks for specific event categories like `learning-to-crawl` and `first-solid-food`. Strategies for managing specific events are built into the system. This is how the system helps to anticipate the needs of the user. While the programmer might provide a different set of story models for the "vacation" ontology, it would be difficult for the user to construct these without programming.

10.6 Annotations

10.6.1 Titles

It is feasible for story models to automatically generate titles for their scrapbook page. For example:

```
(run-story-model 'child-at-age :kid "Danielle" :years 2)
```

The above model could generate the title "Danielle at age 2." The `important-firsts` model for Alicia and Danielle might generate the title "Important Moments for Alicia and Danielle." Since each story-model can have its own method for creating a title from its arguments, title generation poses no technical problems.

Users, of course, would need to be able also to generate and edit titles by hand. Graphically editable objects like Hypercard fields would be ideal.

10.6.2 Histories

The system also needs to provide documentation for how a particular `story-model` was generated. If users are to learn about the system and about video in general by running `story-model`s multiple times, then they must be able to look up how a particular story was generated -- what `story-model` if any was run, what arguments were used, and how it was subsequently modified by the user.

10.6.3 Why

Documentation would be useful down to the level of individual segments. Why was this segment included in this list? The answer will help to explain the workings of the system to the user, and may help to detect errors in the representation. For example, suppose the user sees a segment which seems out of place and selects the "why" button. The system responds that the shot was included because it contains Alicia at age three; however, the user notes that Alicia is not actually there -- the shot was incorrectly logged.

A "why" feature would be particularly helpful if the system were given more ambitious methods for sequencing shots. Why is this close up of the sofa here? Because the system detected a jump cut and was searching for a shot of an object in this same room to act as a transition.

10.7 Scrapbook Page Types

Story-models are currently used to help find and arrange logged footage. This process can be inverted: the system's knowledge about typical story types could be used to help log footage.

One approach is to give the system a set of specialized scrapbook page types which make inferences about segments put on them. For example, creating a "child-at-age" page for Danielle at age 1 would automatically put the title "Danielle at age 1" on the page. It can then be inferred that every video segment placed on that page contains Danielle at that age. Similarly, one could create a "vacation" page for our trip to Italy in 1989. Every segment placed on that page would be assumed to be from that time and place, unless different information is given.

The key problem that again arises is how to inform the user that inferences are being made. For example, one might place the segment "getting on the

plane" on the page "our trip to Italy," and while it was shot in 1989, it did not take place in Italy. How is this error to be prevented?

It would be possible for the system to create a dialog box asking the user to confirm each inference. However, the constant interruptions would make the feature more of a nuisance than a convenience.

Another approach would be to create an inferences window in which text descriptions of inferences are sent. The system might write that "I am assuming that segment 'Amy buys a leather jacket' took place in Italy in 1989." This would be less intrusive, because the user is not interrupted.

There is the danger, however, that the user might not notice these messages. If a large number of inferences are being made, the resulting stream of text is likely to be ignored. One way to reduce the number of such messages is to only notify the user of an inference when it is *used* as part of a retrieval or composition process. There is a trade-off between the convenience of inferences and the cost of errors.

10.8 Object Permanence

10.8.1 Instances

In the current implementation of the Electronic Scrapbook, when one selects the object "sofa" for a particular video segment, the unit `sofa` is placed on the segment's `significant-objects` slot. `Sofa` is the collection of all sofas. One has effectively said "there is a sofa in this shot."

It would be possible instead to create a specific instance of the collection `sofa`. The advantage is that this instance could then be annotated -- all sofas are not pink, but this specific sofa is pink. The unit would be called something like sofa.4 and its `print-name` might be "the pink sofa by the fireplace." `Sofa.4` would have a `my-color` slot holding the value `pink`. Other slots could record its typical location and its typical position in that location.[5]

When a user indicates that there is a sofa in a particular video segment, the system could try to figure out if it is a known sofa. If it is in the living room in Mill Valley, then it is likely to be "the pink sofa by the fireplace." If the total amount of data in the system is small, then the user could be asked to select from a list of all known sofas.

---

[5]Sofas have usual locations, most houses have fixed locations, and frisbees have no fixed location when in use, but may have a usual location when not in use (depending on how neat the owner is.) These details would need to be represented in order to track individual objects.

By tracking specific instances of objects, the system is able to accumulate knowledge. One might search for everything that is pink and this would return Alicia's favorite blanket, Danielle's tricycle, the sofa in the living room, the hand towels in the guest bathroom, and the tiles at the entrance of the Duomo in Florence. This level of detail of representation is only practical if the system tracks specific objects.[6]

Unfortunately, selecting which if any of the known sofas for the new instance of sofa is non-trivial. "The sofa in the living room at home" is different before and after July 1987, because the family moved and bought new furniture. It is possible to create to separate instances of sofa, `sofa.4` and `sofa.6`, and later obtain new information which indicates that they are really the same. The two units would need to be merged. Similarly, the system might infer that the sofa in two segments were both `sofa.4`. If it is later discovered that the sofa in one segment has small gray flowers on it, then the question arises: does the sofa in the other segment also have small gray flowers, or are these really two different sofas?

10.8.2 Temporal Extents

To say that "there is a sofa in this segment" is very general. One can get more specific by saying that "`sofa.4` is in this segment." However, even that is a generalization. Strictly speaking, there is a *temporal extent* of `sofa.4` in this segment, `sofa.4` at 11:50 am to 11:51 am on July 12th, 1991. This allows one to represent the fact that `sofa.4` in 1987 is brand new, but `sofa.4` in 1991 has a coffee stain on the arm. Similarly, one can not simply say that Alicia is 3' 6" tall. Alicia in August of 1991 is 3' 6" tall, but Alicia in August of 1986 is only 2' tall.

[6]Whether this degree of detail is *desirable* is discussed in sections 2.4 and 6.6.1.