LogBoy Meets FilterGirl
A Toolkit for Multivariant Movies

by
Ryan George Evans

B.S. Computer Science and Engineering
Massachusetts Institute of Technology
Cambridge, MA
1991

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in Partial Fulfillment of the requirements of the degree of

MASTER OF SCIENCE
at the
Massachusetts Institute of Technology
February 1994

Signature of Author

Program in Media Arts and Sciences
January 14, 1994

Certified by

Glorianna Davenport
Associate Professor of Media Technology
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by

Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

LogBoy Meets FilterGirl
A Toolkit for Multivariant Movies

by
Ryan George Evans

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on January 14, 1994
in Partial Fulfillment of the requirements of the degree of

MASTER OF SCIENCE
at the
Massachusetts Institute of Technology

Abstract

This thesis describes two tools, LogBoy and FilterGirl, which are designed for creating multivariant movies. Multivariant movies are movies which playout differently each time they are presented. Variances in playout can be based on viewer interaction, available content or viewing context. By making use of annotated video databases, multivariant movies can be created which sequence video autonomously based on viewer preferences rather than driving playout through periodic viewer queries. Periodic viewer queries, an interface technique used by most current interactive narratives, detract from the storytelling process by interrupting the viewer's immersion in the story.

LogBoy is a video database tool which helps moviemakers create the sketchy video annotations that are needed for description-based multivariant movies. Filtergirl is a tool for creating and editing the playout constraints that guide the sequencing of video clips. This thesis describes the structure and use of LogBoy and FilterGirl as well as exploring the issues involved in creating tools for multivariant movie production. LogBoy and Filtergirl's use in multivariant movie production is illustrated with an extended example.

Thesis Supervisor: Glorianna Davenport
Title: Associate Professor of Media Technology

This work was supported in part by Kansa, Bellcore and British Telecom.

LogBoy Meets FilterGirl
A Toolkit for Multivariant Movies

by
Ryan George Evans

The following people served as readers for this thesis:

Reader:

Kenneth W. Haase, Jr.
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Reader:

Mitchel Resnick
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Acknowledgments

The work described in this thesis could not have been accomplished without the encouragment, support, patience and friendship of many people. I am grateful to them all.

Glorianna Davenport, my advisor, provided me with the chance to begin this work, the freedom to experiment, the encouragement to continue and the guidance to create something useful. Thank you for giving me the opportunity to pick up a camera rather than just watch.

My thesis readers, Mitchel Resnick and Ken Haase, kept me on track during the course of my work. Thank you both for laying the foundations for this research and taking the time to listen to my ideas.

Betsy Brown commented on drafts of this thesis and the papers that led up to it. Thank you for a friendly voice when I needed it.

Mark Halliday has been a friend and tutor during my stay here at the lab. Many thanks for taking the time to show me how and where to point a camera. I hope we never lose touch.

I am grateful to the cast and crew of "Just One Catch": Ian Dowell, Steve Dubin, Finnegan, Mark Halliday, Scott Higgins, David Kung, Graham G. Ramsay, John Francis Shavel, Davd Tamés and Dan Zentner. Their patience (and appetite) was endless during the shooting process.

The members of the Interactive Cinema Group and other residents of Macondo provided invaluble assistance when I was working and numerous recreational opportunities when I needed a break. Thanks to Thomas Aguierre Smith, Kevin Brooks, Amy Bruckman, Stuart Cody, Eddie Elliott, Stephan Fitch, Scott Higgins, Gilberte Houbart, Hiroaki Komatsu, David Kung, Lee Morgenroth, Takeshi Nitta, James Seo, John Shiple, David Tamés and Koichi Yamagata.

I can't forget my parents, Charles and Mary Evans and my brother Matthew. Their sacrifices, love and encouragement got me here in the first place and continue to support me.

Finally, thank you to J. D. Darcy Duke for being there for me and with me as long as I care to remember.

Table of Contents

1. Introduction

As digital video becomes a reality, moviemakers are beginning to experiment with new ways of presenting stories to viewers. Traditionally cinema has been a linear medium made up of strips of film glued together end to end to tell a story. Each time a linear film is projected, the viewing experience is identical. The same shots are presented in the same sequence. Digital video provides filmmakers with a way to escape from the linearity inherent in film construction and begin to explore non-linear techniques for playout. These non-linear movies range from traditional hypertext applications with video added as a media type to experimental video installations. One of the most exciting consequences of the non-linear digital medium is multivariant playout.

Multivariant playout is a term which describes movies which can present a different sequence of shots each time they are viewed. A movie might play out differently based on a viewer's preferences (e.g. removing depictions of violence from an R rated movie for family viewing), content available (e.g. a news program which updates itself with incoming stories), viewing context (e.g. a movie which chooses wide shots over close ups of the same sequence of events based on whether it is shown on a high-resolution home television or a low-resolution hand-held television) or arbitrarily (to provide a movie which can be watched several times by the same person). Multivariant playout provides a way to think about personalizing, updating and modifying movies while maintaining the underlying narrative structures that tell a story to the viewer.

Just as linear films are constructed out of a sequence of film clips, multivariant movies are constructed out of sequences of digital video clips. The difference is that multivariant movies must also contain information on how digital video clips can be substituted and rearranged to adjust for particular playouts while retaining narrative continuity. The movie must contain a sense of the story it is trying to convey and how that story can be molded to particular viewers and situations. This process is similar to the way human story tellers change the way they tell a story for particular audiences or situations. To change the way the story is told the human must understand something about the story they are trying to tell, not just recite the words in order.

Currently, even the most sophisticated computers cannot understand stories in the ways humans do. However, we can encode some of the ways in which a story can be changed for multivariant playout. The aim of the research described in this thesis is to provide a set of tools for moviemakers to encode knowledge about multiple playouts and create multivariant movies easily and intuitively. Several systems for making multivariant movies have been created. Most of these systems make use of an underlying branching structure to encode knowledge about how the movie should play out differently based on differing inputs. There are several problems with this approach. The most important problem is the way that the viewer becomes locked into a pattern of periodic interaction with the movie. This type of interaction can detract from the story being told.

The tools described in this thesis, LogBoy and FilterGirl, make use of a new type of structure for encoding information about multiple playouts. This encoding scheme, called descriptive story structures, searches databases of annotated content rather than traversing hardwired links. Descriptive story structures avoid periodic interaction and also make the moviemaker's task easier by providing a more understandable story format and easier methods for editing and augmenting existing stories.

It is important to remember that the two tools described in this thesis, LogBoy and FilterGirl, do not make up a system that understands stories. Instead they provide a way for moviemakers to encode knowledge about the story they are trying to tell and the ways it can be changed for multivariant playout.

Descriptive story structures frame the problem of building a multivariant movie as a process of annotating content and sequencing that content based on attached descriptions. Pieces of digital video, or video clips, are imported into the system and described as to their relevance to the story. Later, story constraints are built which take into account narrative structure, cinematic continuity and user preferences. These constraints guide the movie through playout by selecting pieces of digital video based on their descriptions and sequencing them appropriately.

LogBoy and FilterGirl address the two major tasks in building a multivariant movie using descriptive story structures. The first task, describing digital video clips, is achieved by using LogBoy while FilterGirl allows the moviemaker to build story constraints that govern the playout process.

LogBoy is a video database tool that allows a moviemaker to categorize segments of digital video which will be sequenced during playout. Descriptions are constructed in LogBoy which characterize each piece of video as part of the larger story and as unique segments for different types of playout. Typical categories of description include “Scene”, “Setting”, “Characters”, “Point of View”, “Framing”, “Length” and “Rating”. Each clip is described relative to a set of categories and then these descriptions are used as a basis of selection for multiple playouts.

FilterGirl is a story modeling tool that provides the moviemaker with a way of specifying constraints which guide the sequencing of clips into a story. FilterGirl allows the moviemaker to construct constraints using nested filters. Each filter is a simple mechanism which selects video clips based on their attached descriptions, weeding out clips which do not match a specified description or set of descriptions. FilterGirl’s power comes from many different ways in which filters can be combined to build stories. These combinations of filters describe the relationship of the filters to each other, but more importantly they describe the relationship of the filters to the story itself. Filters can be combined temporally (to guide story structures over time), logically (to provide for Boolean combinations of descriptions), contextually (to calculate descriptions for continuity matching or conditional selection) or based on viewer preferences (to provide for viewer interaction).

Currently, most schemes for building stories with multiple playouts (interactive stories) rely on some type of branching. Pieces of content (video, graphics, text) are joined with unchanging links which are traversed based on the viewer’s choices. Ultimately, these choices are predetermined and scripted by the author by means of a hard coded links. Under this scheme, a viewer views a piece of content and then makes a choice as to which link to traverse to the next piece(s) of content. This branching structure is traditional hypertext molded to the purpose of telling stories. This approach has some problems. First, it is difficult to add content to an existing story because adding links to a new piece of content means considering how that new piece might link to every existing piece of content in the branching structure. This can become an exponential problem. Second, it is difficult to edit branching structures because the number of links in a complicated story can make it difficult to visualize and follow a particular narrative thread through the structure. Third, and most important, the viewer must drive the story manually by making a decision at each branch point. This periodically interrupts the narrative experience and prevents the viewer from becoming immersed in the story. Periodic queries are appropriate when a viewer is

trying to find a particular piece of information, but, in the context of interaction within a fictional narrative, periodic queries can detract from the story itself.

LogBoy and FilterGirl avoid these problems by making use of description based story structures. Because each video clip is described in LogBoy and sequencing is done based on these descriptions, it becomes very easy to add clips to the database. For a clip to become part of the story, the moviemaker need only import it into LogBoy and annotate it with the appropriate descriptions. Since the filters that make up the story structure sequence clips based on their descriptions, any new annotated clips are automatically considered during the filtering process.

Collections of filters built in FilterGirl are easier to read as story structures than traditional branching structures. The reason is two-fold. First, the many types of filter combinations help illustrate exactly the type of story structure the moviemaker intends for the story. The moviemaker can create temporal combinations of filters which layout chapters in a story or conditional combinations of filters which constrain clips to provide continuity during playout. Second, the filters hold descriptions of what type of clip is needed at a particular point in the story. This is much more informative than branching structures which link specific pieces of content. By creating filters the moviemaker automatically describes what makes a clip important at that point in the story.

Most importantly, by relying on descriptions of content rather than hardwired branching structures, LogBoy and FilterGirl move the task of driving the movie playout from the viewer to the computer. This means that rather than making periodic decisions as to where the story should go next, the viewer can choose filters or set parameters, compile a sequence of clips using the filters and watch it like a linear movie. Because of this the viewer can concentrate on the narrative and become immersed in the story itself rather than being constantly interrupted by the interface. In this way, LogBoy and FitlerGirl provide movies with true multivariate playout.

This thesis describes the tools LogBoy and FilterGirl and how they work in detail. It also tries to situate LogBoy and FilterGirl in the larger context of video annotation, story understanding and movie making. When watching a multivariate movie it is easy to ascribe too much narrative understanding to the computer which generates the sequence of video clips. Viewers (and readers of this thesis) should keep in mind that LogBoy and FilterGirl cannot

understand stories, but instead provide moviemakers with a way to think about and construct multivariant stories from descriptions and constraints.

1.1. The Structure of This Thesis

The structure of this thesis is centered around an extended example of a multivariant movie. Most of the issues and systems discussed in this document use this example to illustrate various points.

Chapter 1 is the introduction to the thesis. (You are reading it.)

Chapter 2 lays out an extended example of a description-based multivariant movie, called “Just One Catch”. The chapter describes the movie in general terms, explaining the story it tells, how a user might change the way the story plays out and the pieces of content that are sequenced to put the story together. This chapter is intended to give the reader an idea of what interacting with a multivariant movie can be like. The thesis returns to “Just One Catch” in Chapter 9 to explain the relationship between LogBoy, FilterGirl and the movie.

Chapter 3 explores design decisions that must be taken into consideration when building a video database tool. Special attention is paid to ways of representing content in a multivariant movie system. Issues of temporal representation, granularity, descriptive depth, descriptive context and human interface are touched upon in the context of representation for multivariant playout.

Chapter 4 discusses various types of story structures for building movies with multiple playouts. Design issues are explored including how easily structures can be built and understood by moviemakers and how the structures affect the viewer’s experience. This chapter concludes by describing a new type of description-based story structure specifically designed for multivariant playout, layered filter sets.

Chapter 5 is an informal specification of “LogBoy”, a tool for generating, editing and viewing content descriptions for a movie with multiple playouts. The underlying representation is discussed as well as the graphical interface.

Chapter 6 explains “FilterGirl”, a second tool designed for creating story structures from sets of layered filters. About a dozen filter types are presented with examples. FilterGirl’s interface is also discussed.

Chapter 7 gives details about the hardware and software that were used to build LogBoy, FilterGirl and “Just One Catch”.

Chapter 8 provides a short discussion of the unique problems presented by description-based multivariant movie production. Planning, shooting, editing and story structure construction are discussed.

Chapter 9 returns to “Just One Catch” to explain in detail the production techniques, video databases, filters and user interface that were built to implement the movie introduced in Chapter 2.

Chapter 10 discusses previous research related to LogBoy and FilterGirl in the areas of video databases, database interfaces and interactive story structures.

Chapter 11 talks about possible future enhancements to the LogBoy/FilterGirl system.

Finally, Chapter 12 sums up the arguments, descriptions and specifications laid out in the body of the thesis.

A final note: Throughout the written portion of this thesis the terms “moviemaker” and “viewer” are used to distinguish between two types of users of the LogBoy/FilterGirl system. Moviemakers are people who create multivariant movies. Viewers are people who watch playouts of multivariant movies. This thesis makes a clear distinction between the two roles in order to present clear arguments about the usage of LogBoy and FilterGirl. However, it is important to realize that these two roles are actually two endpoints of a continuum. As viewers begin to interact they become closer to authors and as authors create movies with playouts they may have never envisioned they become closer to viewers. LogBoy, FilterGirl and multivariant movies in general begin to blur the previously clear distinctions between these roles.

2. An Extended Example, “Just One Catch”

“Just One Catch” is a small multivariant movie that was constructed with the LogBoy/FilterGirl system. This chapter lays out the story that the movie tells and describes how a viewer can change the ways it might play out. This brief description of “Just One Catch” is intended to give readers an idea of what a multivariant movie looks like. Chapter 9 will return to “Just One Catch” and explain exactly how the movie was scripted, shot, logged and structured as an example of using LogBoy and FilterGirl for multivariant production.

“Just One Catch” is a simple multivariant movie that was built to illustrate the features of the LogBoy/FilterGirl system. It can play out in different ways based on user preferences. The movie is constructed from a database of approximately 125 video clips that range in length from a 5 to 90 seconds long. A description-based story structure defines how these clips must be sequenced to create the multiple playouts of the movie.

The two sections in this chapter detail a viewer’s experience with “Just One Catch”. The first section describes how a viewer can change the way the movie plays out. The second section is intended to give the reader an idea of what a playout of “Just One Catch” is like. It includes a plot summary of the movie along with excerpts from the script and still frames.

2.1. Interactive Modes

The viewer can change the way “Just One Catch” plays out with two separate parameters. Figure 2.1 shows the viewer’s interface including playout window and parameter sliders. The first parameter is length of movie (labelled “Long” and “Short” in the figure). The playout time can be set anywhere between five and ten minutes. The story of “Just One Catch” is not substantially changed by modifying the length of the playout. Instead, clips which characterize plot points in a shorter amount of time are preferred over longer clips. The second parameter is called “Action vs. Dialog”. With this parameter the viewer can change how much the playout emphasizes the dialog-driven lunch scene or the action-packed chase scene. Again, all of the major plot points are covered during the playout, but depending on this parameter setting more screen time is allocated to action scenes or dialog

scenes. The viewer sets these parameters before playout begins and then the LogBoy/FilterGirl system calculates an appropriate sequence of shots that both tell the story and match the viewer's preferences.

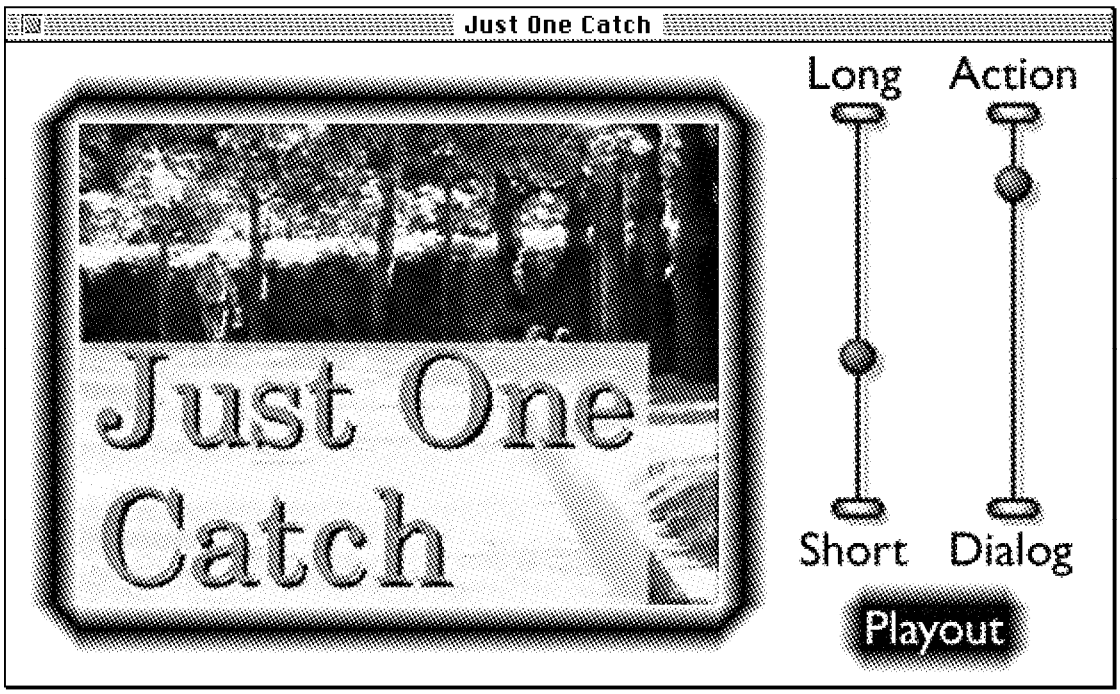

Figure 2.1 The viewer's interface.

2.2. A Plot Summary

"Just One Catch" begins with the character Ian, a Super-8 film enthusiast, strolling through the park with his new Super-8 camera in one hand and a bag lunch in the other. As he walks along he pauses occasionally to get off a quick shot of a tree or a squirrel to test out his camera. He walks by a park bench and decides to eat his lunch. Ian sits down and carefully places his camera on the bench beside him and then proceeds to empty the contents of his lunch bag onto the bench. His lunch includes a banana, a sandwich, a bag of chips and a cookie. *This section of "Just One Catch" remains constant across multiple playouts. Every time the movie is presented this section remains unchanged.*

Ian picks up the banana and starts to peel it when a friend, Graham, walks by and stops to talk. Graham says he's in a hurry for an appointment so he can't talk long, but he notices Ian's new camera. He asks where Ian purchased the camera, how much it cost and looks it over. Graham again mentions his appointment and starts to leave, but notices Ian's lunch on the bench. Graham sits back down and explains to Ian that he was in such a hurry for his appointment that he forgot to have breakfast this morning so could he spare a banana for a starving friend. Ian is glad to help out and hands over the banana. Graham thanks Ian, promises to replace the banana later and takes off in a rush for his appointment. *This section of the movie can be presented in three different ways: long, medium and short. One version is selected based on the viewer's preferences. In the shortest version Ian simply eats his banana and Graham never appears.*

Ian now gets back to his lunch, unwrapping his sandwich. He's just about to take a bite when he hears a dog bark and come running towards him. He sits the sandwich down on the bench and pets the dog as its owner comes over retrieve it. "Come on, Finnegan! Come on, let's go!" Ian notices the dog owner's Chicago Bulls hat and asks him if he saw the game last night. They talk briefly about Jordan, Barkley and the race for the championship. Suddenly, Ian hears this weird slurping, chomping sound. Both Ian and the dog owner look down and realize that the dog has finished Ian's sandwich in two bites and is trying to reach the cookie. The dog owner snaps on Finnegan's leash and gives it a firm yank. Apologizing for the sandwich, the owner and his dog beat a hasty retreat down the path. Ian is justifiably frustrated at this point, but thinks the better of kicking the dog (or its owner) as they leave. *This section of the movie can be presented in three different ways: long, medium and short. One version is selected based on the viewer's preferences. In the shortest version Ian simply eats his sandwich and the dog owner never appears.*

Ian cautiously gets back to what is left of his lunch. After looking around carefully to make sure no one is walking his way, Ian picks up his bag of chips. He gives the package a tug and then another, but it won't open. He gets a better grip and starts pulling hard. Just as the bag bursts open Ian gets the feeling he is being watched. He looks up to see a drunken bum in dirty clothes standing right over him and finishing off the last of a paper bag wrapped bottle. The bum starts ranting on about being accused of having an affair with Barbara Walters, but gets quickly to the point. He wants Ian's chips and he wants them now. Ian doesn't see any way to reason with this person so he hands over the bag of chips that he just opened. The bum stuffs the bag in his jacket pocket and shuffles down the path trying to get the last drop out of his bottle. *This section of the movie can be presented in three different ways: long, medium and short. One version is selected based on the viewer's preferences. In the shortest version Ian simply eats his chips and the bum never appears.*

Ian has almost given up on ever eating again, but then he realizes that he still has his big chocolate chip cookie left to eat. Once again he looks around cautiously and then begins unwrapping the cookie. He takes one bite, then another, then another. It seems like he might actually get to eat something this time. Then out of nowhere he hears the clacking of a skateboard on the asphalt path. Instinctively, Ian hides the cookie against his chest. The skateboarder comes zooming by and before Ian knows it the skateboarder has grabbed his brand new camera and is heading down the path at top speed. *This section of "Just One Catch", like the opening scene, is always the same. No matter how the viewer has set the parameters, any playout of the movie must move through this plot point.*

This is the last straw. Ian drops what is left of his lunch and takes off screaming after the skateboarder. They chase for what seems like hours through the park, across flower beds, under trees, but Ian can never quite catch up with the thief. Ian loses the thief a couple of times, but he seems to be playing with Ian, staying just out of reach, waiting until Ian finds him again to start skateboarding. *Playouts of this section of the movie can vary widely based on viewer input and on internal factors. Ian can chase the thief through six different settings and these settings can be portrayed in many different ways. Also, the entire length of this section can be shortened by portraying fewer than the full six settings.*

Ian finally finds the thief on the park path near a bench and decides this is his last chance so he takes off at full run. The skateboarder takes off also, but just as he reaches top speed the skateboard wheels lock on a twig lying in the path. The skateboard and skateboarder flip onto the path and the precious camera flies into the air, spinning. Ian sees the camera fly up and runs as if in slow motion with his arms outstretched trying to catch it as it falls back down. At the last second he makes an incredible dive and catches the camera just before it hits the ground. He lies there and breaths a sigh of relief as he cradles the camera to his chest with his hands. *Unlike many interactive movies, "Just One Catch" always ends in the same way. No matter how the parameters are set or which video clips are sequenced the story always ends up at this point.*

3. Video Databases

At some level every interactive narrative system must rely on representation of content. This representation may be implicit (such as a set of unlabeled links which are traversed to view content chunks) or it may be explicit (such as a news footage library which allows users to search for video based on names, dates, locations). The question of how to represent content and how to generate descriptions depends on many disparate factors including media type, end use, query mechanisms and database size. This chapter introduces some of the relevant design decisions and trade-offs which must be addressed when building a video database for use in interactive story structures. These issues include temporal representation, granularity, descriptive language, descriptive thickness and user interface.

3.1. Temporal Representation

One of the fundamental aspects of any video database is the representation of time. Video, like music and speech, is a temporal medium. To make sense, annotations on a video stream must refer to specific points or sections of time. There are two major paradigms for the temporal representation of annotations: chunk-based descriptions and stream-based descriptions.

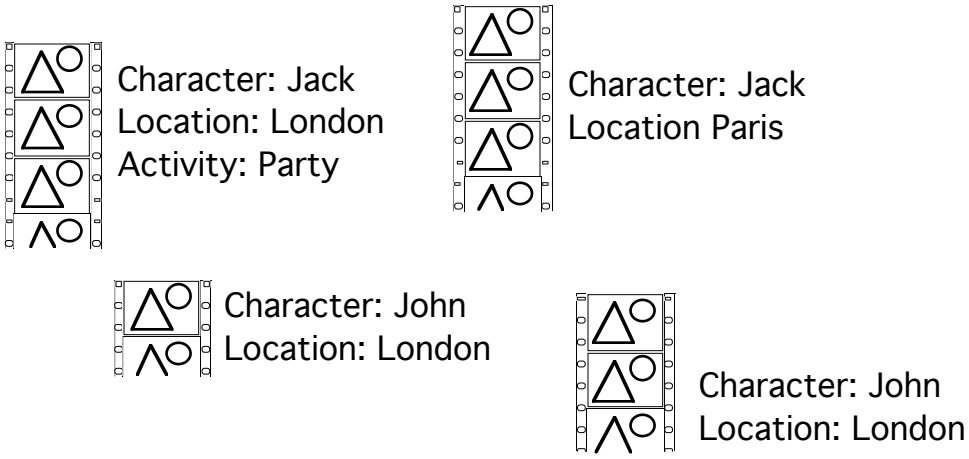

Figure 3.1 Chunk based temporal representation

Chunk-based representation requires that the image stream be divided into non-overlapping segments by choosing beginning and end points in the video. Descriptions attached to each chunk are valid for the entire duration of the

segment and the chunk is generally not subdivided. Chunk-based representations are easy to understand since they are directly analogous to relational databases (i.e. each chunk is one record in the database). On the other hand, chunking imposes a particular meaning onto the image stream by precluding alternate segmenting schemes.

Streamed representation avoids the segmentation problem by augmenting the uncut image stream with descriptions which have temporal extents. Annotations are attached by specifying time durations or beginning and end points relative to a continuous image stream. In this way overlapping description segments can be layered on top of the image stream to build a rich descriptive base. A description at any point in the video can be generated by taking a “cross-section” through the descriptive layers. Stream-based representations provide a more flexible and extensible paradigm than chunking since descriptions can be layered onto previous descriptions and the image stream is never segmented.[ASD 92]

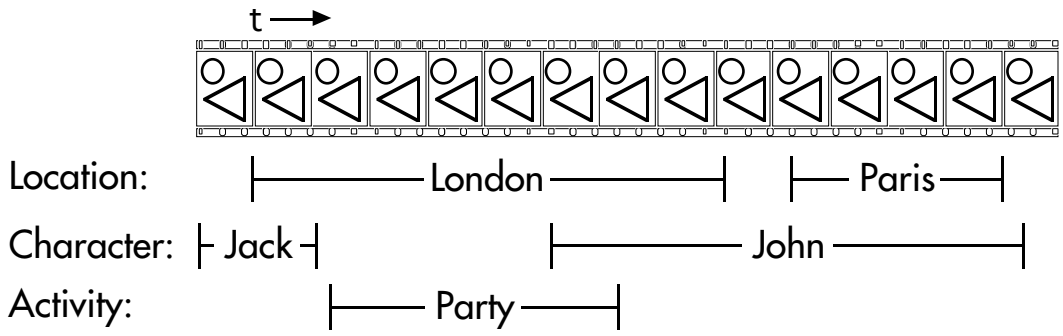

Figure 3.2 Stream based temporal representation

Despite the descriptive flexibility of the stream-based metaphor, chunking is a more appropriate technique for a video logging system which is to be used in description-based story telling. Under a stream-based metaphor any computational shot selector must not only pick a good shot based on attached descriptions, but must also generate that shot (i.e. pick good beginning and end points) since no chunking has been imposed on the stream by the description process. A chunk-based metaphor implies a clear distinction of duties: the moviemaker decides what constitutes a good shot while the shot selection mechanism decides what constitutes a good sequence of shots.

3.2. Granularity

Granularity is one of the most basic aspects of a video database. The term granularity refers to the relative size of descriptive segments attached to an image stream. Databases with a fine granularity have small descriptive

segments while databases with a coarse granularity have large descriptive segments. Both fine and coarse grained databases can be used for building narratives depending on what type of story the moviemaker is trying to tell and what types of personalization parameters will be available to the viewer. A moviemaker who uses a fine grained database for building a multivariant movie will be thinking about building cinematic sequences shot by shot. This technique places a larger burden on the moviemaker to provide more detailed story structures. On the other hand, a moviemaker who uses a coarse grained database will be able to start building narrative structures at a higher level, but will lose the flexibility of being able to change the way the movie plays out on a shot by shot basis.

Any video database tool designed for building description-based story structures should provide for moviemakers working with a fine-grained or coarse-grained database or a combination of both. This means that the database tool should have a scalable interface which does not favor one particular time scale. The database should also be able to display the length of clips or descriptors relative to each other, providing a way to compare pieces of the database by granularity.

3.3. Descriptive Depth

Another important characteristic of video databases is descriptive depth. Descriptive depth is a term which refers to the relative number of descriptions attached to a particular piece of video. For instance, a database with a thick descriptive base has many annotations while a database with a thin descriptive base has few attached annotations. In general, thick description bases are used for multi-purpose, multi-user databases in which searches take place for many different purposes. The thickness of the descriptions allows multiple users to search based on varied characteristics of the video. On the other hand, thin or sketchy description bases are generally useful only to a single person and perhaps for only a single purpose. The thin descriptive base narrows down the selection of descriptive characteristics to only those which are relevant to the task at hand.

Description-based interactive narratives make use of sketchy descriptive bases because the domain of queries is limited by their relevance to the story structure. Queries to databases are made within the context of sequencing the movie so only descriptions relevant to the flow of the narrative need to be recorded. Thick descriptive bases are more suitable to video databases where the user is making queries directly rather than indirectly through the context of a narrative. In this case the creator of the database generally wants to

record every possible description to give maximum searching power to the end user. A news footage database is a good example of a video database which could make use of a thick descriptive base.

Any annotation system for use in description-based story production should both encourage sketchy descriptions and take advantage of the fact that a minimum number of descriptive structures will probably be used. This means that graphical interface metaphors can be used which would otherwise break down as the number of descriptive characteristics grows large.

3.4. Descriptive Language

Descriptive language becomes an issue in the design of any database tool. Descriptive language is a term which refers to the actual form of descriptions in the database. These descriptions range from unstructured free text to fully canonicalized iconic languages. The trade-off between structured and unstructured languages is similar to the trade-off between thick and thin descriptive depths. Highly structured and canonicalized languages are good for databases which will be searched by many different people for many different purposes. This is because similar descriptions will be grouped together into the same categories. However, the set of descriptors must be chosen carefully so that every possible annotation type is accounted for. At the same time duplication of annotation will result in difficult to search databases. Unstructured descriptive languages are much easier to log in because the intended descriptions do not need to be translated into a canonical language. The drawback is that it is difficult for multiple users to generate or retrieve content in such a database since each user will word descriptions slightly differently.

Description-based story structures are generally built by a single moviemaker for a single purpose (the construction of a multivariant movie). For this reason, annotation languages for description-based story structures should tend towards the unstructured side of the spectrum. More structured languages provide too many categories for personal use and might not provide the proper categories for a moviemaker's very personal descriptions. It must be kept in mind, however, that databases for description-based stories must ultimately be searched to find appropriate clips. To facilitate this, the database should allow the moviemaker to create new descriptors and easily describe new content using the set of already created descriptors.

One descriptive language that addresses most of these problems is keyword classes [AS 92]. Keyword classes allow moviemakers to create new keywords

easily and organize them into categories (e.g. names, places, etc.). These classes and keywords are then available for use as descriptors for specific pieces of video content.

3.5. Descriptive Context

Most interfaces to video logging applications are structured around the content that is being logged [Bruckman 91; Davis 91]. The person annotating the video examines a segment of video and then generates all of the descriptions relating to that piece of content. The interface represents the video once as an unchanging stream and repeats descriptors as necessary to show annotations. This representation answers the question “What are all of the descriptors attached to this piece of video?”. This linear interface paradigm is adapted from traditional database applications which usually require the user to generate or edit one record at a time.

There is another interface paradigm which is better suited to building databases for multivariant movies. This interface paradigm is centered around the attached descriptions rather than the video stream. Each descriptor is represented in the interface once and video clips are repeated as necessary to show how they relate to the descriptors. This representation answers the question “What are all of the pieces of video that have this descriptor?”.

This second representation presents a kind of descriptive context for the moviemaker in which pieces of video can be easily compared to other pieces of video relative to a particular descriptor. This is very helpful in description-based multivariant movies because most of the logged characteristics of the footage are both very subjective and gear primarily towards building on particular type of playout experience. Characteristics such as “mood”, “humor content” and “action level” are often used as descriptive types. When using subjective characteristics, descriptions must be created with other similar descriptions in mind. The moviemaker must be able to answer questions like “How funny is this chunk of video relative to the database as a whole?” and also be able to easily log one characteristic of the database continuously without being constrained to the linear structure of the video itself.

A descriptive context allows the creator to attach descriptions within the more relevant context of the entire set of characteristics. Systems centered around descriptive contexts require a very different type of interface which can display and manipulate the entire base of video content relative to one particular characteristic. The importance of providing a relevant global

context as well as detailed local information is addressed in G. W. Furnas' Generalized Fisheye Views [Furnas 86].

3.6. Overhead View of Descriptions

When a moviemaker is creating or editing a database for use with a description-based story structure, descriptive coverage becomes an important concern. Coverage is a term borrowed from traditional movie making terminology which means having enough footage to be able to edit a sequence which tells the intended story. Coverage in a description-based story structure means not only having enough footage to be able to tell the intended story (or stories in the case of a multivariant movie), but also having enough correct annotations to be able to find the right clips.

The creator needs ways to query for specific combinations of descriptions as well as get a general encompassing view of all of the logged footage. Dividing the logging process along characteristics rather than the content itself provides a descriptive context which allows the creator to look for missing annotations or pieces of footage. A graphical representation of annotations within each of these characteristics can further assist the moviemaker to get a quick feel for the layout of descriptions. To help the moviemaker make more specific queries about descriptive coverage, the logging tool can be extended with a simple query system which graphically depicts pieces of video content which match combinations of characteristics.

4. Story Structures for Multivariant Playout

To create a cinematic narrative, pieces of film or video content must be sequenced in meaningful ways. Traditional film editors sequence images relative to their understanding of the structure of the story being told. The rules of cinematic language constrain the ways the editor can tell the story in a manner understandable to the audience.

To create a multivariant movie cinematic sequencing can be different for each playout of the movie. Because of this, sequencing of video clips must take place computationally. To accomplish computational sequencing, segments of video content must be ordered according to the story that is being told, to the accepted rules of cinematic language and to input from the viewer. These sequencing constraints are expressed with story structures which the computer uses as a guide for sequencing video.

Computational story structures can take many different forms including hypertext links, world models and description-based story structures. Several issues must be considered when selecting a story structure for a particular narrative presentation or storytelling system. The type and amount of viewer interaction in the final presentation plays a role in the decision process as well as issues like extensibility of the story, understandability of the structure to the moviemaker and type of content.

This chapter briefly describes several different types of story structure schemes and then explores issues related to selecting a story structure for storytelling with multivariant movies.

4.1. Schemes for Story Structure Specification

Story structure specification schemes range from simple tools for traditional screen writers to systems based in artificial intelligence which generate stories from descriptions of characters, locations, goals and plans. This section describes some of the different types of story structures that have been used to create interactive narratives.

4.1.1. Hardwired Links

Hardwired links are the most common type of story structure used in interactive narratives. Hardwired links are essentially hypertext systems adapted to storytelling. In this type of linked story structure individual pieces of content (video, graphics, text) are joined temporally with typed links. The viewer interacts with this system by viewing one piece of content and then choosing which of the outgoing links to follow. Many familiar games and multimedia applications are based on this type of structure.

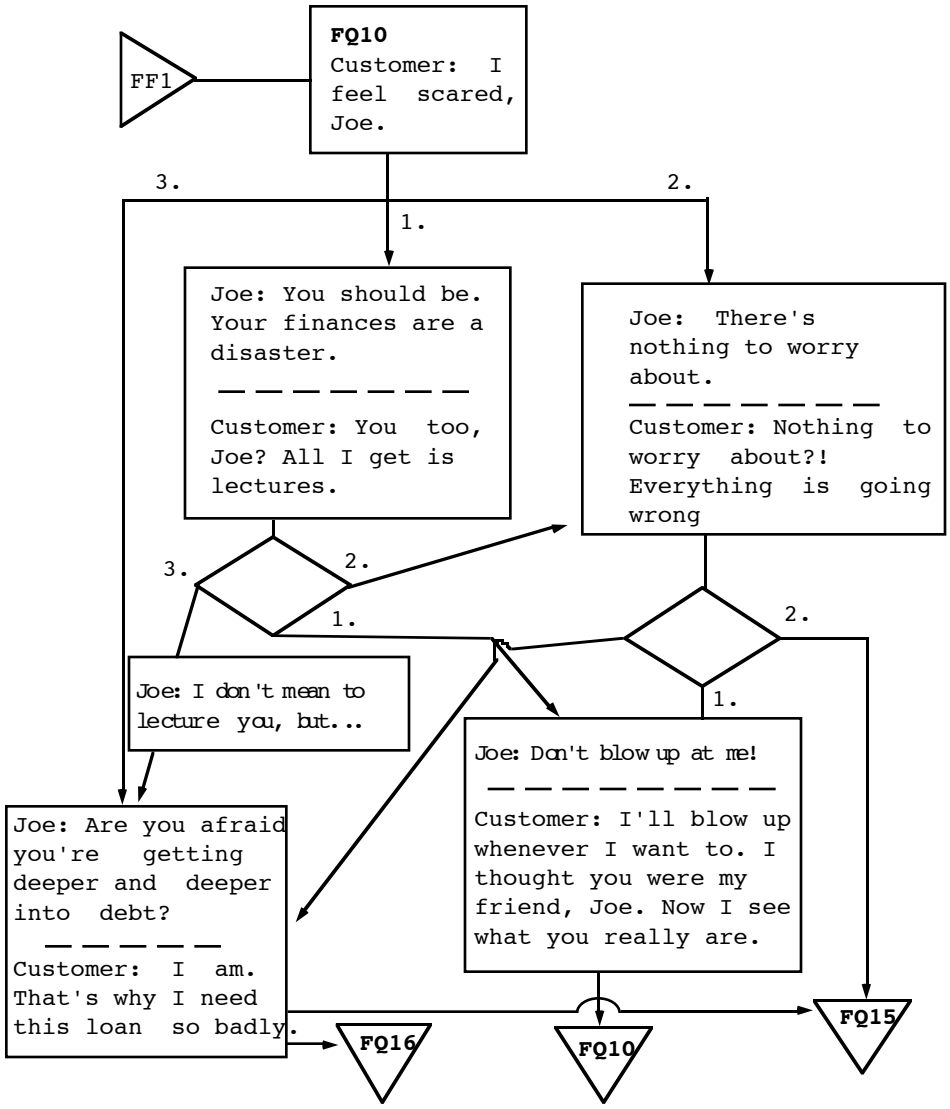

Figure 4.1 A story structure built with hardwired links [Iuppa 88].

A classic example of a story structure based on hardwired links is illustrated in the book *Advanced Interactive Video Design* by Nicholas V. Iuppa [Iuppa 88]. Here a flow chart (Figure 4.1) shows the story structure behind a video

disc training application designed to teach people skills to bank tellers. The disc tells a story about a man in financial trouble who is applying for a loan. At branch points the viewer, playing the role of the teller, can decide how to react to the questions and comments of the applicant. The flow chart shows video clips (represented by rectangles) and branch points (represented by diamonds). The labeled triangles connect to other points in the larger structure. At each branch point the system pauses and asks the viewer to make a decision

Many computer games are also built on top of a branching structure similar to this one. Often the branching structure represents a physical layout of a space and the viewer navigates the space by selecting which direction to move at each branch point.

4.1.2. World Models

A few systems have been built which use symbolic models of a physical space, characters and objects within that space to create a simulated world from which stories can be generated. In these worlds, characters have goals and create plans computationally to achieve those goals within the simulated world. A story is generated by allowing the characters to play out their plans within the world. The story is composed of a (usually textual) description of the character's attempts to achieve their goals.

A classic example of generating stories from world models is James Meehan's "Talespin" system [Meehan 81]. In Talespin characters are created who want to achieve simple goals. Characters create plans to achieve these goals which can include moving to other physical spaces, manipulating objects, communicating with other characters (honestly and dishonestly) and negotiating with other characters to get something they want.

Another more recent experiment into generating stories from world models is Joseph Bates' Oz project [Bates 90]]. This project is more sophisticated than "Talespin" in two ways. First, viewers become a character within the simulated world, manipulating objects and communicating with other characters. Second, there is a "drama manager" that has global control over all the elements of the simulated world. The drama manager can guide a story over time according to the author's specifications.

4.1.3. Description Based Structures

Description based story structures rely on databases of annotated content to put together multivariant movies. These databases are searched for

appropriate pieces of video, graphics or text to present in sequence to the viewer and create a story. The driving mechanism for description based story structures is some type of filtering process which selects content based on their annotations. The selection criteria can include plot points, continuity concerns and viewer preferences.

One of the first multivariant movies built with a description-based structure is the New Orleans Interactive project by Glorianna Davenport [Davenport 87]. This movie contains about three hours of video which spans four years and illustrates the city of New Orleans and its officials planning for and executing the 1987 Worlds Fair. Footage of city council meetings, community meetings and interviews documents a large scale urban renewal project on the waterfront of New Orleans. The movie was used as a teaching aid in urban design classes at MIT.

Within New Orleans Interactive, video descriptions are used such as date the video was shot, who appears in the video, what part of the city is being portrayed and what part of the World's Fair project is being discussed. Viewers can select a particular theme to follow across the entire movie such as "The role of the Jax Brewery in the renewal project." or "The mayor of New Orleans' affect on the World's Fair selection process." A movie is created by filtering out video clips which match the theme and sequencing them in chronological order.

4.2. Issues in Selecting Story Structures

Several different issues become important when a moviemaker is selecting a story structure language for a particular interactive narrative. These issues include the type and extent of viewer interaction, how the narrative might later be extended and how understandable/editable the story structure itself needs to be. This section describes these issues in detail.

Amy Bruckman's "The Combinatorics of Storytelling: *Mystery Train* Interactive" [Bruckman 90] is an interesting paper which discusses some of the issues of extensibility and understandability that are presented in this section.

4.2.1. Interactive Modes

When selecting a story structure for a multivariant movie the moviemaker must consider how viewers will interact with the system and how much control should be given to the viewer. Viewers can interact by stating preferences before the movie starts, by periodically choosing from a list of

alternatives or even selecting a few pieces of content from a larger database. Interaction also involves giving control over to the viewer. Interactive control can be defined by how much flexibility the viewer has in changing the way the movie plays out. At one end of the spectrum lie linear movies in which the viewer has no control over how the movie plays out. At the other end of the spectrum lie systems in which the moviemaker plays little part in guiding the story.

4.2.1.1. Conversational Interaction

Conversational interaction is the most widely used interface mode for interactive narrative. Interaction takes place in a manner that is similar to a conversation going on between the computer and the viewer. The viewer examines a piece of content presented by the computer (e.g. video, graphics, text) and then the computer queries the user to determine how the viewer wants the story to continue. The viewer then makes a choice and the computer presents another piece of content. In this type of interaction the computer and the viewer take turns driving the story, much like in a human conversation. This type of interaction is similar to the way most people interact with computers.

Conversational interaction is a natural outgrowth of a branching story structure. The viewer examines the content at a node and then is presented with a list of links to follow to the next content node. One of the problems with this type of interaction is that it is often difficult for the viewer to become immersed in the flow of the story. Conversational interaction presumes that the viewer will pause periodically to make a decision about where the story should go next. This causes a periodic interruption in the narrative which interrupts the reverie of story telling. This constant barrage of questions is generally appropriate when the user is searching for a particular piece of information, however when a user is trying to be entertained or understand a story, questions detract from the experience.

4.2.1.2. Browsing

Browsing is a less used interactive mode that provides a much more flexible environment for the viewer. In a browsing environment the viewer is presented with a collection of loosely organized pieces of content. Viewers are then free to make their own connections between the pieces to build up a story. The resulting experience is little like skimming a book. Instead the viewer looks carefully at a small set of pieces of content and mentally constructs a coherent narrative from them. Some interactive narratives

which use browsing change the available pieces of content based on estimations of the viewers preferences. These estimations come from looking at previously selected pieces or explicitly querying the viewer.

One successful system which makes use of a browsing interactive mode is the Elastic Boston Map [Perey 92]. The map was built by students in Professor Glorianna Davenport's 1992 Elastic Movie Time class. The map presents different graphical views of the city of Boston. These graphical views are populated with short pieces of video which give a portrait of a place, interview a resident, advertise a shop or give a particular person's view of the city. Many different people contributed video clips to the database. These pieces of content are organized in several categories such as neighborhood, type of content (e.g. shops, landmarks, interviews, etc.) or a particular person's vision of the city. As the viewer moves through the system the selections of video clips change based on the viewer's preferences. A kind of path is found through the pieces of video and a loose narrative thread is built up in the viewer's mind. The threads take various forms such as a portrait of a place, a particular person's vision of the city or a tour through a neighborhood.

Under a browsing interactive mode almost all of the burden of constructing the narrative falls on the viewer. The moviemaker provides only a loose organization for the content and the viewer strings the pieces together. One of the biggest problems with a browsing interactive mode is that it is almost impossible for a moviemaker to know exactly what type of story the viewer will come away with.

4.2.1.3. Fluid Interaction

A new type of interactive mode has recently been explored which recognizes the viewer's need for uninterrupted immersion in the narrative and the moviemaker's need for narrative control. This paradigm, called fluid interaction, allows the viewer to personalize the playout of the movie within constraints provided by the moviemaker. After personal parameters have been set the movie plays out much like a linear film, but personalized for one particular viewer. Parameters such as violence level, length of playout and point of view can be imagined which would affect the content of the movie. Rather than the constant interruption of conversational interaction, fluid interaction provides viewers both control over the way the movie plays out and the opportunity for immersion in the story telling process [Galyean 92].

Fluid interactive modes fit most easily onto world models and descriptive story structures. The reason is that the viewer no longer drives the narrative.

In the case of a world model story structure the computer simulates actions in the world over time resulting in a story. With description-based story structures, the computer drives the narrative within the viewer's parameters and the moviemaker's constraints. This means that the computer must have some algorithm for selecting pieces of content within the set of constraints for a particular playout.

A few successful description-based movies have been built to explore fluid interaction. "An Endless Conversation" was among the first [DEH 93]. "An Endless Conversation" is a small scene which presents a dialog between two characters, Tom and Dave. As the scene progresses, the characters ask questions, tell stories and react to each other's statements. The viewer can change two parameters before the movie plays out, "Rating" and "Pacing". The rating parameter has two settings "R" and "PG" (two of the American movie ratings). Under the "R" setting rude comments are included in the dialog. Under the "PG" setting no rude comments are included. The pacing parameter allows the viewer to change how snappy the dialog is as it plays out. A fast paced dialog uses shorter utterances, while a slow paced dialog uses longer utterances. The viewer sets these parameters before the movie starts and then the movie plays out based on those parameters.

4.2.2. Extensibility

Another factor in selecting a story structure for multivariant playout is extensibility. Linear films are generally not changed once the final editing is finished. The movie is planned, shot, edited and projected with little change in content or editing after this process is finished. (The recent profusion of "Director's Cuts" seems to be changing this rule.) Multivariant movies by definition can be different each time they are watched. This opens the door for movies in which the content base or story structures are periodically updated.

To accommodate for this possibility moviemakers must take into consideration how a story structure might be augmented or changed in future versions. The extensibility of the content base and the story structure must both be considered.

Hardwired link structures are perhaps the most difficult structures to change or extent due to the fact that the links connect individual pieces of content. In the worst case a moviemaker must look at how a new piece of content might link to every other piece of content already in the database.

World models and description based structure are the easiest story structures to extend with new content. In a world model structure the moviemaker need only create new characters, places, objects and goals. The world simulator will then incorporate these new features into subsequent narratives. Description based structures search for pieces of content based on their descriptions rather than relying on links to particular pieces of content. This means that annotated pieces of content added to the database will become part of newly generated narratives.

4.2.3. Understandability

Ease of comprehension is another important factor in selecting a story structure for multivariant playout. A moviemaker must be able to easily transcribe a vision for a multivariant story into a computational story structure without having to mentally translate story ideas into arcane structures. The story structures built for the movie must map easily from narrative structure to multivariant playout.

Hardwired link structures make it easy for a moviemaker to see exactly how a movie might playout. The links show exactly what piece of content might be presented next based on the user's preferences. A big problem with link structures, however, is that the resulting web of pieces of content and annotated links bear little resemblance to the narrative structure that the moviemaker is trying to convey. It is difficult to translate the links into the coherent beginning, middle and end that make up a story. One of the reasons for this is that links preclude abstraction within their structure. Abstraction allows a creator to divide a story into conceptual chunks such as sentences, paragraphs and chapters or shots, sequences and scenes.

World model story structures make it easy for a moviemaker to see how a character might react to a particular situation or why a character is attempting a particular action. This is because goals and plans are explicitly laid out within the world model. However, if a creator is interested in plot points and temporal structures in the story, world models break down in terms of understandability. This is because a small change in the model of the world can have a great effect on the way the story plays out. It is almost impossible to generalize about how a change in a character's goals, the layout of the world or placement of objects might affect the resulting story.

Description based structures provide moviemakers with the easiest translation from narrative to computational story structure. One reason is that description based structures allow narrative abstraction. The

moviemaker can build a story structure which clearly lays out a beginning, middle and end with more complicated structures nested inside of each. Another reason that description based structures are easier for moviemakers to work with is the fact that they are built from descriptions of content rather than specific pieces of content. This means that the story structure contains a record of why a particular piece of content should be used at any point in the story.

4.3. Layered Filters as Description Based Structures

This thesis describes two tools, LogBoy and FilterGirl, which allow moviemakers to build movies using description based story structures. Description based story structures can be implemented in many ways ranging from sophisticated template systems to simple annotated chronologies. FilterGirl uses layered filter sets as a way of building simple or complex multivariant movies. Filters, in this context, are simple selectors which take a video database as input and return some subset of that database as output. A simple filter might select only clips which are described as containing the character “Jack” (see figure 4.2).

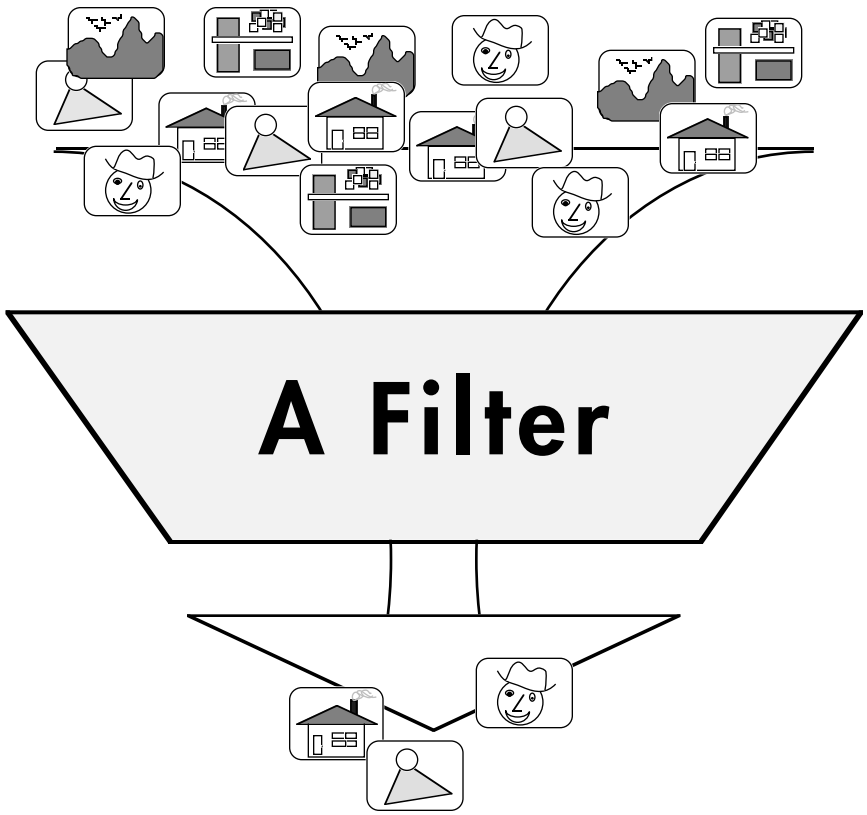

Figure 4.2 A simple filter.

The power of layered filters comes from combining (layering) the filters in different ways. Filters can be combined with simple set operators such as intersection, union and negation. With Boolean operators each filter to be combined is run on the database and then the resultant sets are combined. Filters can be combined in sequence temporally. Temporal combination allows creators to layout the sequence of the resulting playouts. Filters can also be combined contextually to build complex dependencies between filters. Contextual combinations of filters resemble simple rules which can use transcripts of playout, lists of active filters, real time clocks and viewer input to select which filters to run.

FilterGirl further adds simple abstraction constructors so that complex but understandable story structures can be built. This is achieved by defining combinations of filters as filters themselves which can be named and combined again with other filters.

One pitfall in using layered filter sets is over constraining the video clip selection process. This can happen when filters are layered which require mutually exclusive descriptions to be attached to a single video clip. Any tool for moviemakers which makes use of layered filter sets must take this into account. The simplest way of addressing this problem is by prioritizing layered filters. The driver which runs the filters can then try to match as many filters as possible from most important to least important before an empty set of video clips is reached. Some kind of simple debugging tool which flags over constraint situations can also help.

5. LogBoy Design

LogBoy is a database tool for creating and editing descriptions which are attached to video clips. LogBoy differs from traditional video loggers and computer database applications in three respects. First, LogBoy is designed specifically as a tool for creating descriptions for multivariant movies which rely on description-based content selection. As such it provides the moviemaker with an "overhead" view of the database which can point out database properties in relation to a particular story structure. Second, LogBoy allows the author of a multivariant movie to engage in truly iterative design since it is part of a single-platform, integrated toolkit for creating and editing. Finally, LogBoy encourages sketchy descriptive bases for content. Sketchy descriptions, annotations which form a thin, highly-personal description base, provide an efficient representation paradigm for content designed for limited reuse (e.g. multivariant movies).

5.1. Video Clips

LogBoy's basic unit of content is the video clip. A video clip is a non-overlapping segment of video with a specified beginning and end point. Under this paradigm all descriptive structures associated with a clip are valid for the entirety of the segment. The presentation of a multivariant movie is ultimately made up of a computationally selected sequence of video clips. Video clips are, by definition, always shown in their entirety. One of the major reasons that LogBoy uses the video clip as a database model is to avoid the problem of computationally choosing valid in and out points for a segment of video. This provides a well defined division of labor between the human author and the computational clip selector. The human decides what constitutes a good clip while the clip selector (with guidance from human created story structures) decides what constitutes a good sequence of clips.

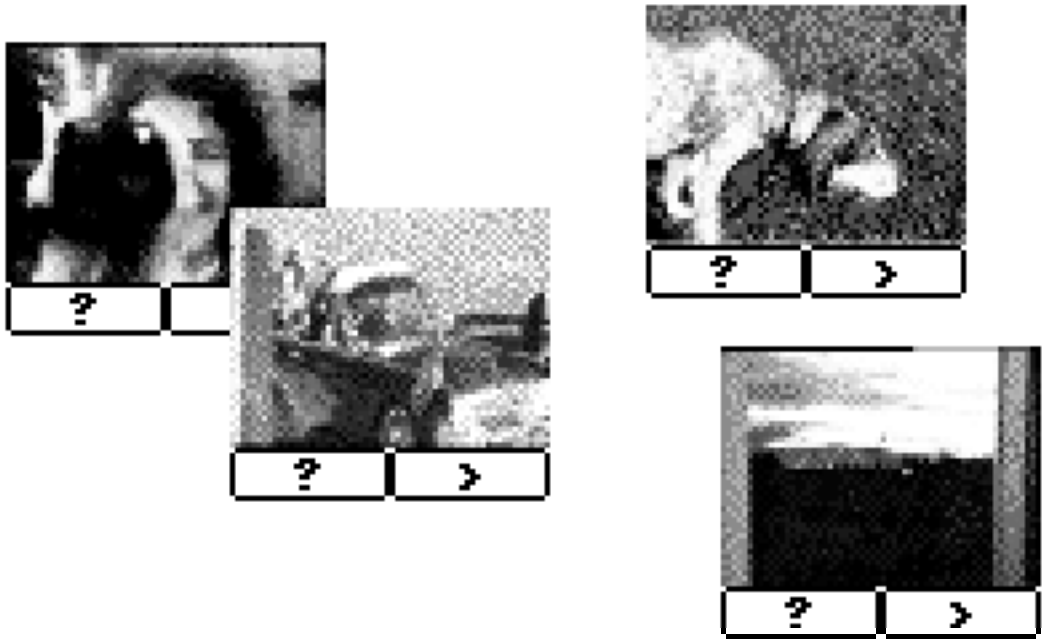

Figure 5.1 A collection of four video clip icons.

Video clips are represented graphically within the LogBoy environment as draggable icons. Figure 5.1 shows a collection of four video clip icons. Each icon shows a key frame from its associated video clip along with two buttons used for obtaining more information. The info button, indicated by the question mark, opens a window which displays important information about the video clip. Currently, this includes the location of the clip on the hard disk, the title of the clip and descriptions which have been attached to this clip in LogBoy. The playout button, indicated by the greater-than sign, opens a digital video playout window in which the video clip can be previewed in its entirety.

5.2. Slots and Values

Descriptions in LogBoy are stored in a simple slot and value structure. This means that each database (collection of video clips) has some associated characteristic types called slots. Each clip has one or more values associated with it for each slot. This means that each clip has the same set of slots as every other clip in the database, while the values within those slots are unique to each clip. Typical slots might include “characters that appear within the clip”, “location where clip takes place” or “part of the story”. (Often slots are given shorter, mnemonic names such as “Characters” or “Setting”.) Values take on meanings only in relation to these slots. For example, a particular clip might have the value “Betty” within its “Characters” slot or

the value “New York” in its “Setting” slot. The slot/value structure is discussed in more detail in Chapter 2. The LogBoy system allows the moviemaker to associate zero, one or multiple values with each slot for a video clip. Figure 5.2 shows a conceptual model of slots and values attached to video clips.

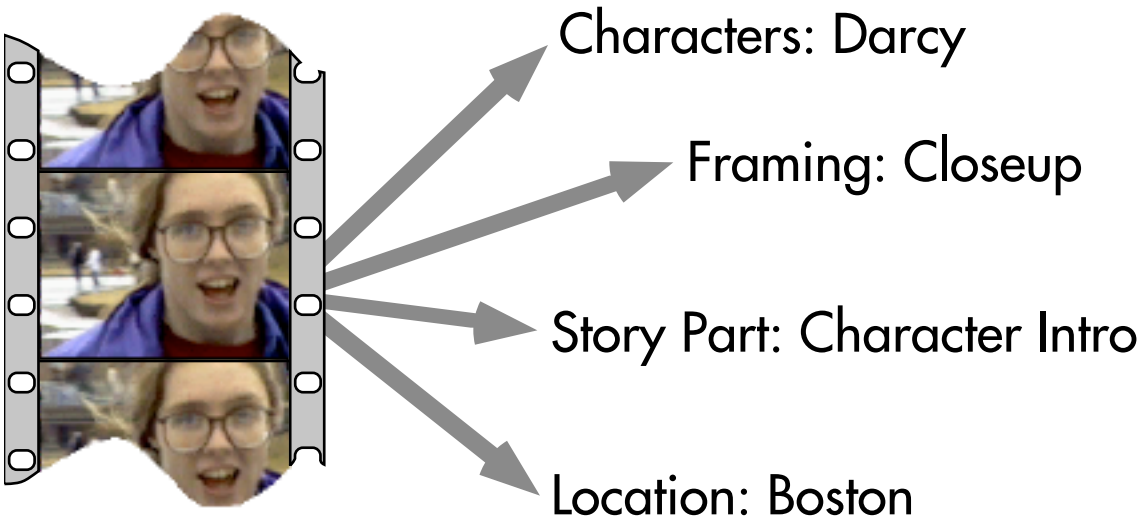

Figure 5.2 A conceptual model of slots and values attached to a video clip.

The LogBoy application does not limit the moviemaker to a specific set of standard slot types, nor does it limit the number of slots that may be associated with a clip database. This is because LogBoy is designed to create video databases in conjunction with multivariant story structures. In creating content-based multivariant movies, the descriptions in the content database are generally paired closely with the content selection mechanisms and interactive story structures. This means that the creator can use sketchy descriptive bases and highly personalized description spaces that might not make sense within some more generalized content selection scheme.

5.3. A Graphical Description Space

LogBoy represents the three major components of its database structure (video clips, slots and values) graphically to create a description map of the database. As described above, video clips are represented on the screen as key frame icons. The position of an icon on the screen gives the user information about the descriptions attached to it. Additionally, these icons can be repositioned to change their descriptions.

To show the descriptions attached to a video clip, slots and values are also shown graphically on the screen. Slots are represented as windows within the

Macintosh environment (called “slot windows”) and values are represented as colored, rectangular areas within slot windows (called “value areas”). If a video clip icon is positioned within a value area which is in turn located within a slot window then that value is attached to that video clip for that slot. Repositioning the clip icon into a different value area will change its attached description for that slot. Multiple slot/value pairs may be attached to a particular video clip by using multiple instances (copies) of its key frame icon.

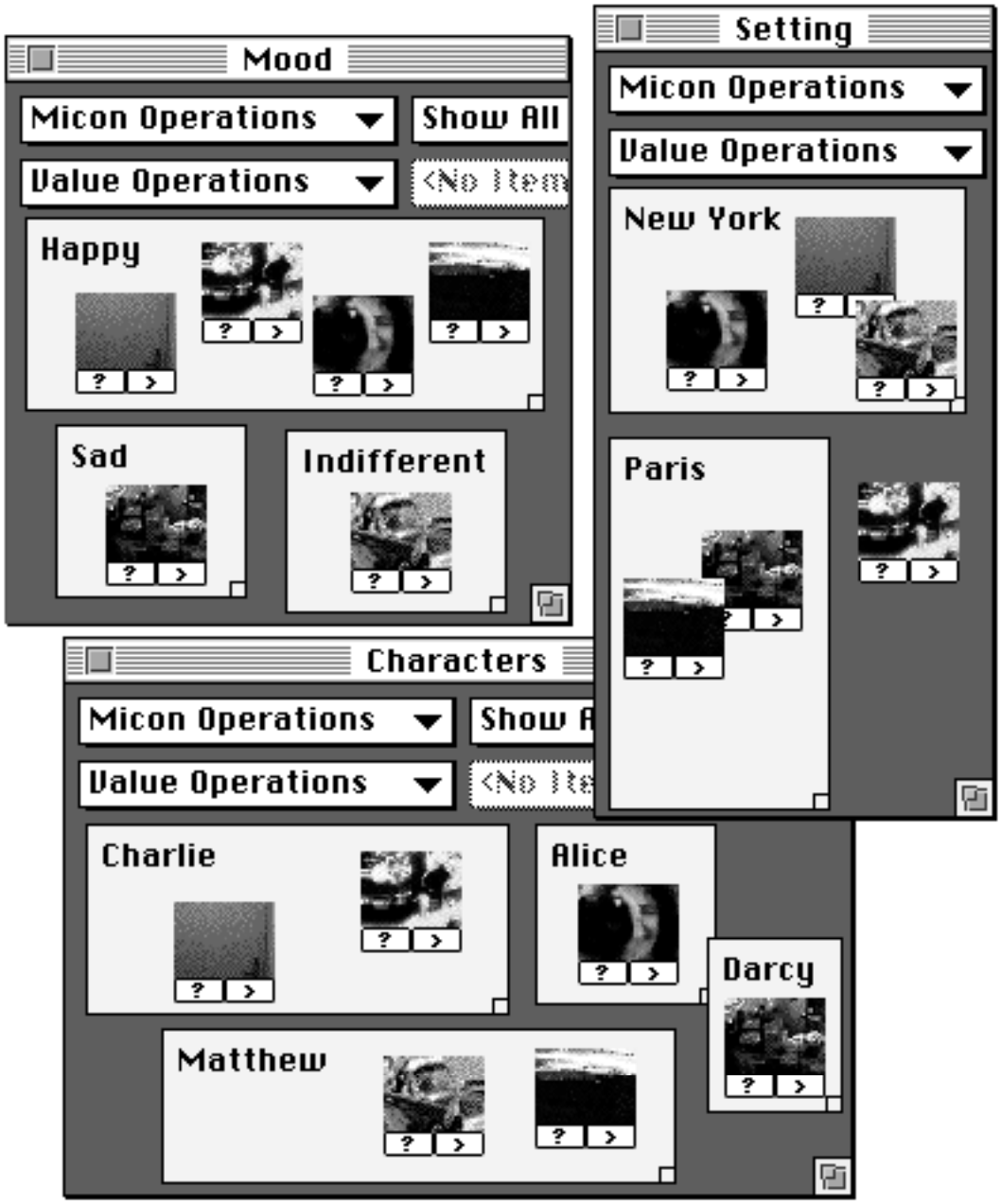

Figure 5.3 A typical LogBoy session with three slot windows.

Figure 5.3 shows a typical LogBoy session including clip icons, slot windows and value areas. In this example the database currently includes six video clips and three slots. Each of the three windows (“Mood”, “Setting”, “Characters”) depicts the descriptive structures for its associated slot. Each slot window contains value areas and video clip icons. Video clip icons within value areas gather slot/value pairs based on the window in which they appear and the value area in which they reside. The same icon can appear in multiple slot windows so that multiple slot/value pairs may be associated with it. For example, the icon which appears in the “Alice” value area of the “Characters” slot gets the “Characters”/ “Alice” slot value pair. The same icon also appears in the value area “Happy” within the slot window “Mood” and the area “New York” in the window “Setting”. Thus the slot/value pairs associated with this clip are “Characters”/ “Alice”, “Mood”/ “Happy” and “Setting”/ “New York”. Pressing the info button (marked with a question mark) on the bottom of the clip icon opens a description window for that clip which lists all of the associated slot/ value pairs (see Figure 5.4).

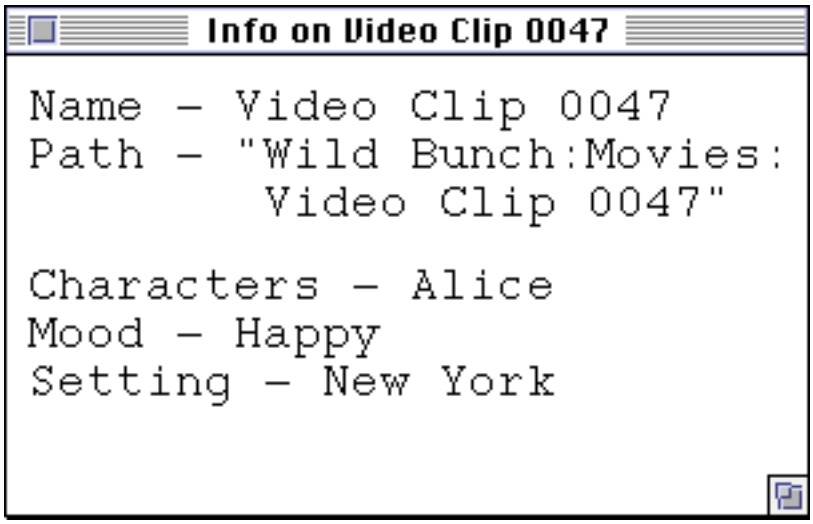

Figure 5.4 A clip description window showing the slot/value pairs attached to a video clip.

LogBoy not only displays descriptions graphically, but the logging process is also graphical. Slot/ value pairs attached to clips can be changed by simply dragging the clip icon to another value area within a slot window. Value areas as well as slot windows are draggable and resizable. Value areas can be added and deleted through the “Value Operations” menu available in each slot window. Slot windows and video clips can be manipulated via LogBoy’s control window. (A typical control window is shown in Figure 5.5. This control window corresponds to the session shown in Figure 5.3.) The control window

contains a list of all currently defined video clips in the database (listed by their titles) along with a list of all currently defined slots in the database. Slots can be added and deleted via the “Slot Operations” menu. Video clips can be added and deleted via the “Clip Operations” menu. Slot windows can be opened by double clicking on their title in the list. Video clip description windows can also be opened by double clicking on their description in the list.

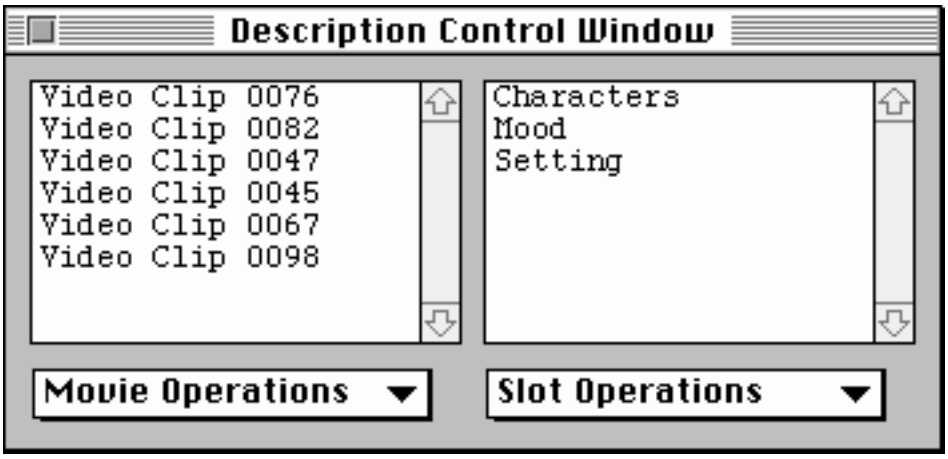

Figure 5.5 The LogBoy control window showing a list of video clips and a list of slots.

5.3.1. Slot Window Filters

Slot window filters provide a way for the creator to effectively make simple database queries and also manage icons within slot windows by showing or hiding the clip icons in a slot window. Slot window filters work in a similar way to the story filters described in the next chapter, however slot window filters (currently) only come in one flavor while story filters come in about a dozen different types. Slot window filters simply include or exclude clips that have a particular description (slot/value pair) attached to them in LogBoy.

Slot window filters can be applied via the two menus available at the top of each slot window. The top menu allows the user to select between “Show All Clips”, “Show Only If” and “Show Only If Not”, while the bottom menu allows the user to select a slot/ value pair from a list. All possible slot/ value pairs are available in this menu even if they are not currently used. Figure 5.6 shows the effect of different filters on the “Characters” slot window originally shown in Figure 5.3. The top window in Figure 5.6 shows the “Characters” window with a “Show Only If” “Mood = Happy” filter applied. This filter hides all clip icons which do not meet this descriptive criterion. Similarly, the bottom window shows the “Characters” window with a “Show Only If” “Setting = Paris” filter in effect. This hides a different selection of icons.

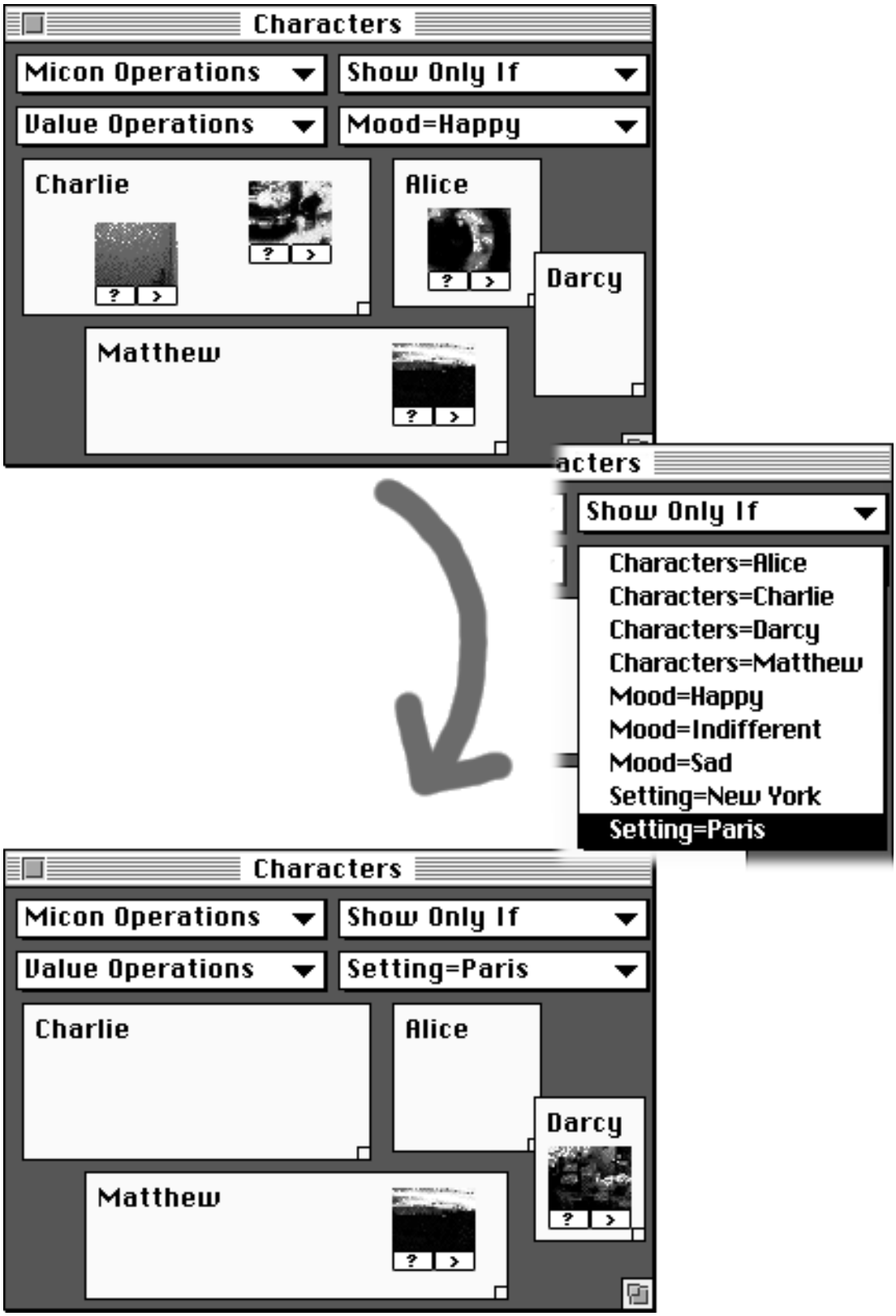

Figure 5.6 Slot window filters can be applied with the menus available at the top of every slot window.

Slot window filters are helpful in two ways. First, the moviemaker can make simple Boolean queries in a slot window. For instance, in the first window it is very easy to pick out all of the clips which both contain the character "Matthew" and are described as being "Happy". Second, the set of video clips within a particular slot window can be simplified to include only clips which are relevant to the characteristic described by the slot. This can be helpful in more complicated databases when a particular characteristic (i.e. slot) is not relevant to an entire set of clips. For instance, imagine a multivariant movie which includes both a chase scene and a dialog scene. Each of the clips in the dialog scene might need to be categorized according to the topic of the conversation depicted. To this a "Conversation Topic" slot would be created in the database. Obviously, this characteristic has no bearing on clips intended for use in the chase scene. A slot could be created called "Scene" with two values "Chase" and "Dialog". If all of the clips in the database were logged within this slot then within the "Conversation Topic" slot window all of the irrelevant clips could be hidden. In many cases this greatly increases the understandability of a slot window and reduces visual complexity.

5.4. LogBoy and Multivariant Movies

LogBoy's graphical interface to the database provides the moviemaker with a way to visually scan and evaluate a collection of clips for specific descriptions, anomalies and general trends. In LogBoy it is very easy to quickly get an idea for how the database as a whole is divided under a certain characteristic. For instance, in the small database shown in Figure 5.3 we can see very quickly that there are many more "Happy" clips than "Sad" clips in the "Mood" slot window.

This interface works particularly well for designing databases for computational shot selectors and multivariant movies. This is because the moviemaker must make sure that the shot selector has a fully populated database into which it can make queries. The selector cannot ask the database for a close-up of Alice in New York when none exists. The graphical nature of LogBoy allows the moviemaker to quickly scan the database to make sure it is fully populated. FilterGirl also helps in the task of fully populating a database. This process is described in the next chapter.

These same qualities which make LogBoy ideal for use in creating description-based multivariant movies detract from it's use as a more general database tool. The LogBoy interface has a practical limit on the number of clips which may be stored in one database. The screen and windows can become very cluttered with icons when working with large databases. This is

almost never a problem when creating multivariant movies since the number of clips for a particular movie is generally small because each clip is designed for use in that movie alone. Similarly, there is a practical limit on the number of descriptive structures (slots and values) that can be built in one LogBoy database. Again, screen real estate can become cluttered with slot windows and value areas. Luckily, this is even less of a problem in creating multivariant narratives since only those descriptors which are relevant to a particular story will appear in any one database.

6. FilterGirl Design

FilterGirl allows a maker of interactive movies to create, edit and preview simple story structures in conjunction with a database of video clips created in LogBoy. FilterGirl's design centers around filters which come in several different types. Each filter takes a set of annotated video clips as input and returns a subset of those clips, narrowing down the selection to the most appropriate clip for that point in the narrative. The several filter types provide a flexible, hierarchical language within which time dependent and context dependent movie structures can be built. This chapter describes the filter types and their behaviors, how the author can create and edit filter instances, how filter based movies can be previewed and debugging options.

6.1. The Filter Types

At the simplest level a filter takes a set of video clips as input and returns some selected subset of those clips as output (see figure 6.1). Filters can select clips based on descriptions which have been created in LogBoy (e.g. all of the clips containing the character Nicole). More complicated filters can be constructed by taking advantage of the simple interface between any particular filter and the clip database (i.e. a set of clips as input, some subset of those clips as output). This simple interface allows the creator to combine multiple filters in many interesting ways. Set operations (e.g. set intersection or set union) become possible for clip selection by chaining the inputs and outputs of filters. Dynamic, time-dependent story structures can be modeled by swapping filters with other filters as the viewing experience plays out. Additionally, filters can be selected in real time based on viewer input or more complicated context dependent means (e.g. rule bases).

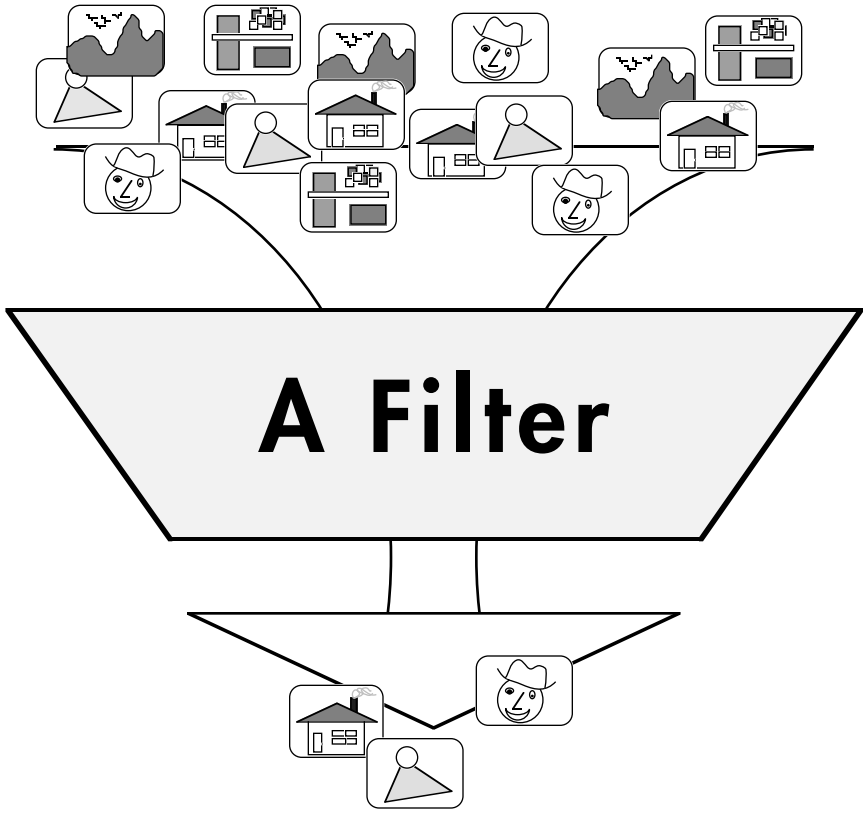

Figure 6.1. A typical filter.

FilterGirl effects many of these filter combinations by providing the creator with a varied set of filter "types" from which to create story structures. Filter types have been constructed which implement clip selection by set operations, simple templates, time dependent structures, context dependent means and viewer interaction. The more complicated filter types may contain other, simpler filters within themselves and combine their behaviors according to predefined rules. For example, the "intersection filter" filter type (see below) implements traditional set intersection by containing two or more other filters within itself. Each of the defined filter types is described below.

6.1.1. Basic Filter

The basic filter is also sometimes called the identity filter. It outputs exactly the same set of clips which are given to it as input. It is useful as a place holder in time dependent structure when the creator does not want to filter out clips, but needs to place a filter between two other filters. It is also useful when creating new types of filters since the basic filter forms the CLOS super-class from which all other filter types inherit their basic behavior.

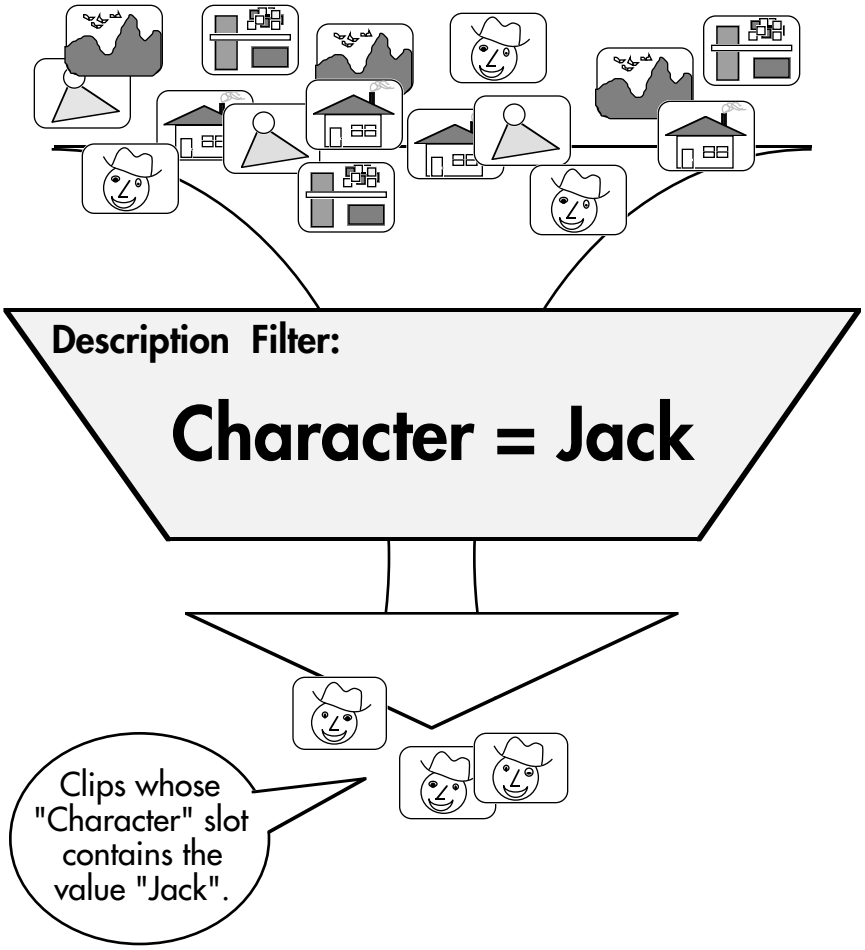

Figure 6.2. A description filter which selects clips described as containing the character Jack.

6.1.2. Description Filter

The description filter is the simplest and most widely used of all of the filter types. Any particular instantiation of a description filter selects only clips which have a particular description attached in the LogBoy application. Thus a moviemaker can create a description filter which selects only clips in which a particular character appears, which take place at a certain location or have some other particular characteristic described in a LogBoy database. Each instantiation of a description filter only selects clips on one specific description. Figure 6.2 illustrates a description filter which selects only clips in which the character Jack appears. When selecting on combinations of descriptions the creator must rely on more complicated filters to combine description filters (e.g. the intersection filter). As noted in previous chapters, descriptions in LogBoy are annotated as slot/value pairs (e.g. Character/Nicole). This is the format in which descriptions are specified within a description filter.

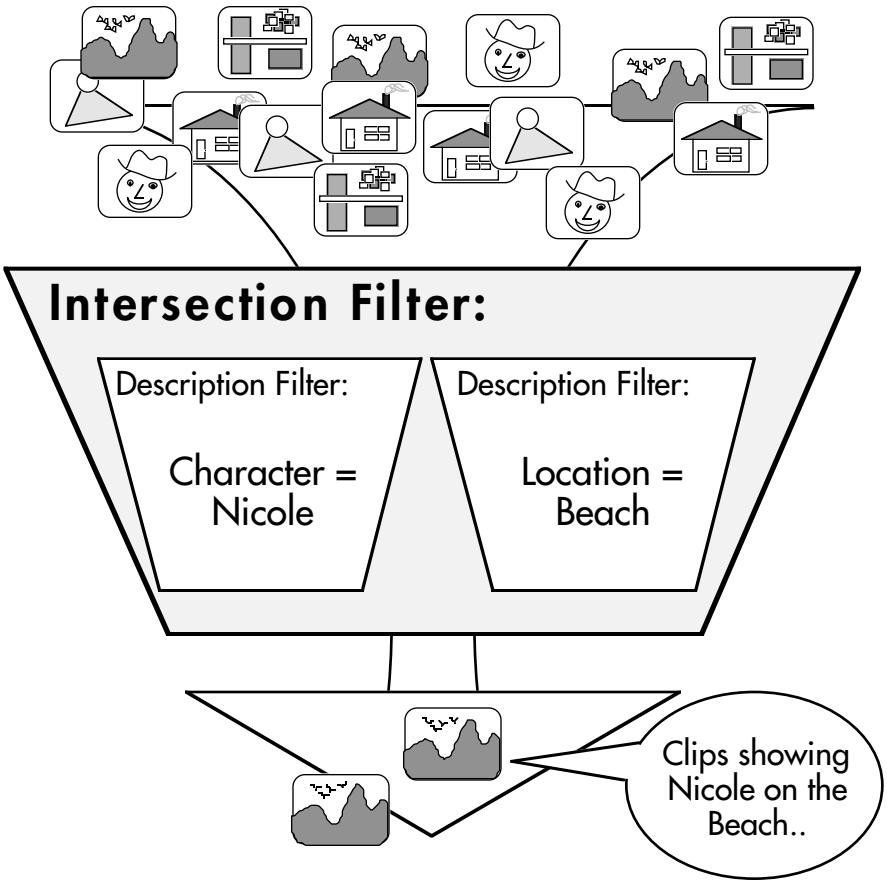

Figure 6.3. An intersection filter which selects clips described as showing Nicole at the beach.

6.1.3. Set Operation Filters

The set operation filter family implements simple set operations such as intersection, union, negate, etc. As such, they are not dependent on an internal clock, viewer interaction or the context of their operation. The set operation filter types provide the means to produce Boolean combinations (e.g. “and”, “or”, “not”) of slot/value pairs.

6.1.3.1. Intersection Filter

The intersection filter allows the moviemaker to combine two or more filters with a set intersection operation. Each of the combined filters is run in turn and the common clips which are selected by all of the filters are gathered as the output of the intersection filter. Figure 6.3 shows an intersection filter instance which combines two description filters. The first of the description filters selects only clips in which the character Nicole appears. The second description filter selects clips which are located on the beach. The intersection filter combines the outputs of its contained filters using set intersection so the

output of the filter consists only of clips which are described as containing Nicole at a beach location.

6.1.3.2. Union Filter

The union filter implements the set union operation on two or more inner filters. Each of the inner filters are run in sequence and all of the clips returned by the filters are gathered as the output of the union filter (duplicate clips are removed). For example, a union filter could be used to combine a “Character = Jack” description filter with a “Character = Nicole” description filter to return all clips in which either Jack or Nicole appears.

6.1.3.3. Negate Filter

Any instantiation of the negate filter type contains exactly one filter inside of it. The negate filter essentially inverts the behavior of its contained filter by returning all clips given as input to the negate filter which are not returned by the inner filter. A simple example: by putting a “Location = Beach” description filter inside of a negate filter one can create a filter which selects all clips which do not take place at the beach.

6.1.4. Context Dependent Filters

There are two generalized types of context dependent filters: the rule filter and the eval filter. Their behavior can depend on arbitrary aspects of the movie playout state and viewer interaction. Other filter types not in this section also depend on particular parts of the movie context, but rule filters and eval filters allow the moviemaker access to any part of the movie state by allowing arbitrary lisp expressions.

6.1.4.1. Rule Filter

The rule filter makes use of an arbitrary lisp predicate (a predicate is a function which returns either true or false) to choose which of two contained filters to make valid. If the predicate returns true the “then filter” is chosen. If the predicate returns false the “else filter” is chosen. The predicate function has access to all parts of the movie state such as viewer interfaces, previously shown clips (the movie transcript), the entire clip database and the state of other filters in the movie. For example, suppose one wanted to build a filter which would introduce the character Jack only if he has not been introduced yet. A rule filter could be built with a predicate which would reference the movie transcript to see if any clips of Jack have been shown yet. If clips of Jack have been shown then it might choose a different character. If clips of Jack have not been shown then it would choose an introductory clip of Jack.

6.1.4.2. Eval Filter

Rather than containing and combining other filters, the eval filter contains arbitrary lisp code which evaluates into a filter object. This allows the moviemaker a way to construct filters on the fly which can depend on any part of the movie state including the clip database itself, the transcript of previously shown movie clips, viewer interfaces and the state of other filters in the movie. Eval filters also provide a way to hook into existing lisp programs which might be useful in creating more complex interactive behaviors (e.g. network applications, expert systems, etc.).

6.1.5. Template Filters

The template family of filters includes filter types which apply their contained filters over time. This makes it simple for the creator to sequence clip playout over time or across a viewing experience. Cinematic structures such as shot/reverse shot and narrative structures such as beginning/middle/end can be built with template filters.

6.1.5.1. Shot Template Filter

The shot template filter is the most familiar of the template filters. It provides the capability of making filters valid in sequence, one filter per clip, as the viewing experience plays out. This capability allows the creator to guide the playout of the final movie at the finest granularity possible, clip by clip. Figure 6.4 shows a shot template filter which contains four filters inside of it. The filters represent four steps in the process of a character, Woody, cooking an omelet. In this example, the shot template filter is currently set to the third internal filter, meaning that the first two shots have already been shown in this movie and that only the third filter will be used in selecting a subset of the clips on the input.

Any instantiation of the shot template filter type can be set to repeat when it reaches the end of its filter sequence. This option can be used to create simple looping structures.

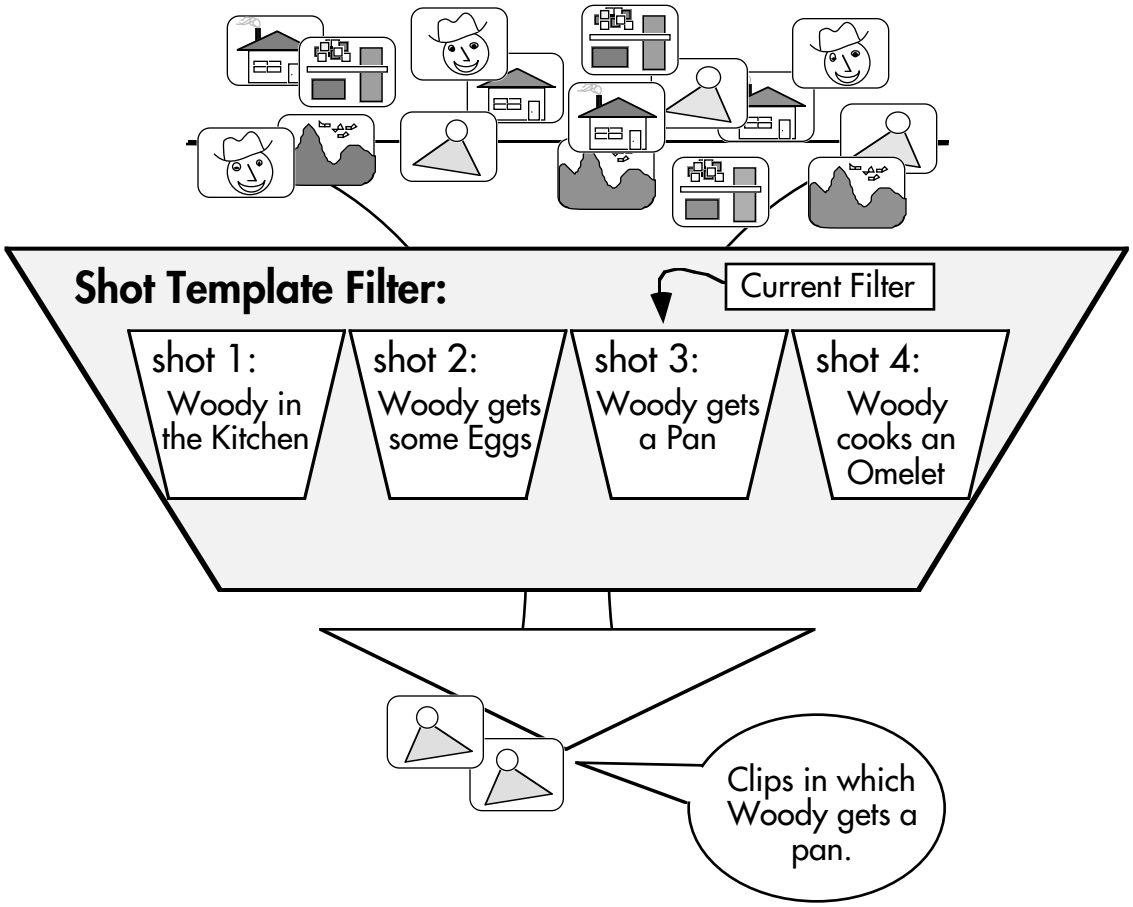

Figure 6.4 A shot template filter which builds a sequence about Woody cooking an omelet.

6.1.5.2. Time Template Filter

The time template filter allows the moviemaker to specify clocked lengths of time over which specific filters are valid. This is done by associating a "start valid time" and an "end valid time" with each filter inside the time template filter. The time template filter watches the system clock and routes its input through the appropriate internal filters. For example, we could make a movie which starts with 60 seconds of clips featuring the character Nicole, continues with 120 seconds featuring the character Jack and ends with 30 seconds of clips featuring both characters. If two or more filters are scheduled to be valid during the same time period their outputs are combined with set intersection, as with the intersection filter. If other combinations are desired more complicated filter structures can be built with multiple time template filters. (Note: Filter valid times only specify when filters are turned on and off. The clips that are selected may in fact run over these times.)

The time template filter has a repeat mode much like the shot template filter, which zeros its internal clock when all filter valid times have been run through. Also, the time template filter can run off of one of two separate internal “clocks”. The first choice is the absolute clock provided by the system. This behaves as expected. The second choice is to add up the length of all clips which have already been viewed and match this time against the filter valid times. This allows the viewer to skip over unwanted clips and move on to later clips without disturbing the normal playout.

6.1.5.3. Advancable Template Filter

The advancable template filter is the most powerful of the template filters. It is useful when the moviemaker wishes to abstract the development of a multivariant movie into sections or scenes. The advancable template filter works much like the shot template filter in that it contains a list of filters within itself and makes them valid (turns them on and off) in sequence. Unlike the shot template filter, the advancable template filter does not move to the next filter in its sequence after each shot is played out. Instead, it advances to the next filter when it receives a special signal.

This signal, called the end of filter signal, can be generated by any type of filter which can contain another filter. This is because the end of filter signal can take the place of a normal filter. When an end of filter signal is called upon to filter a set of clips, it instead sends a signal to the advancable template filter which contains it. This signal tells the advancable template filter to call on the next filter in its sequence to narrow the clip set.

For example, suppose a moviemaker wanted to make a simple multivariant movie which was divided into two conceptual chapters, an introduction and a conclusion. Each of these chapters might consist of several types of shots which must be called in sequence. The creator could make two shot template filters each of which represented one of the chapters in the movie. To combine the two sequentially the creator would add an end of filter signal to the end of each shot template (remember, the end of filter signal can take the place of any other filter.) Then, the two shot templates could be combined sequentially in an advancable template filter. When the first shot template, the introduction, finished it would send an end of filter signal and the advancable template would move to the next shot template, the conclusion. When the conclusion chapter finished it would also send a signal and the advancable template would then be completed. See Figure 6.5 for an illustration of a two “chapter” advancable template filter. In the figure, four shots have already played out in the viewing experience. Since the first shot template filter

contained in the advancable template filter contains two shots (plus the end of filter signal), the current filter setting is the third filter of the second shot template.

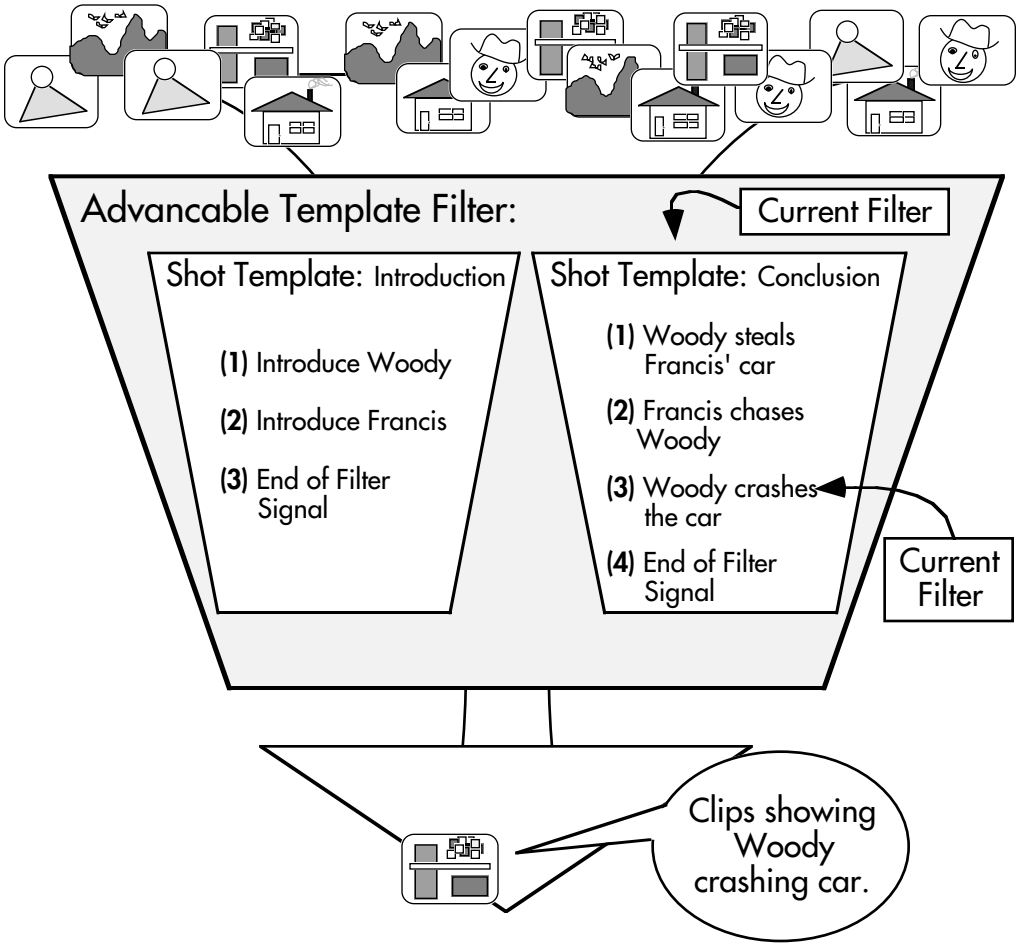

Figure 6.5 A simple advancable template filter that builds a sequence about Woody stealing Francis' car.

Some details about advancable template filters: First, more interesting behaviors can be constructed with the advancable template by embedding end of filter signals inside of context dependent filters or interaction filters. This allows for movies of variable lengths. Second, if an end of filter signal has no advancable template filter "containing" it to catch the signal, then the signal is caught by the top level filter driver, the movie experience ends and the viewer is notified that the experience is over. Third, advancable templates have a repeat flag just like shot template filters, which allow the construction of simple looping structures.

6.1.6. Interaction Filters

Interaction filters provide simple on-screen interfaces to the viewer which allow him or her to easily change the behavior of specific filters, thus affecting the playout. Currently only one simple type of interaction filter has been implemented, but more complicated interaction filters could be built which were more suited to particular types of multivariant movies (e.g. sliders for ranges of values, graphical dialog items which related to the theme of the multivariant movie).

6.1.6.1. Global Variable Filter

The global variable filter allows moviemakers to build viewer interfaces to their movies and easily connect them into a filter structure. With the global variable filter this connection takes place through a global variable in the lisp environment. The filter itself contains a list of filters of which one will be selected as the active filter and the name of a global variable. The global variable polls the global variable each time it is called upon to filter the database. The global variable must contain an integer and this integer is used as an index into the contained list of filters. This very simple mechanism allows moviemakers to build whatever interface components they wish and easily hook them up to the filter structure.

Figure 6.6 shows an instance of a global variable filter and the global variable it polls to influence its selection. In this example the global variable must contain an integer between 1 and 3 inclusive which is then used as an index into the list of contained filters (baseball, biking, skiing). The figure also illustrates a simple radio button interface component which lists the three sports from which the viewer can select. (Radio buttons allow the viewer to choose one and only one button from the list.) This radio button set feeds into the global variable, changing its value. The global variable filter polls this value to find out which contained filter it should choose.

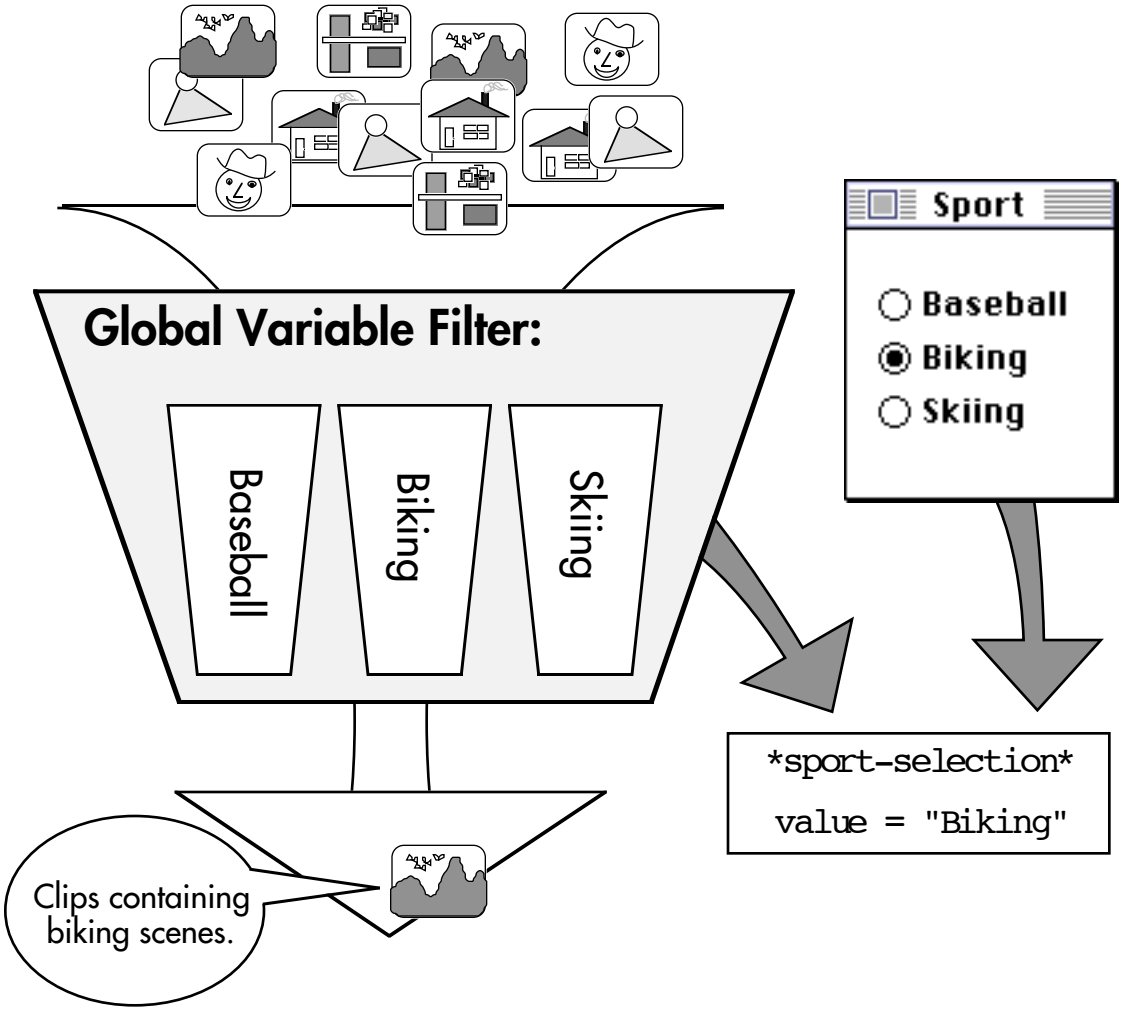

Figure 6.6 An instance of the Global Variable filter type showing how it can poll a global variable which is updated by a user interface component.

6.1.7. Canonical Filters

The canonical filter family includes specialized filters which could be created with the other filters types, but are so commonly used that they are included as part of the primitive filter types.

6.1.7.1. Continuity Filter

The continuity filter is a specialized type of context dependent filter which keeps track of descriptive “variables” as the movie plays out. It allows the author to specify one descriptive slot which is to be matched or not matched from clip to clip as it plays out. In this way, continuity constraints can be modularized and expressed separately from the other filter types that they are combined with. For instance, a moviemaker might be building a short sequence of a man taking his dog for a walk. The database could contain

shots of the same man walking both a poodle and a greyhound. As a continuity constraint the moviemaker wants to make sure that the man is always walking the same dog during the sequence. A temporal filter could be built that describes how the sequence plays out over time (e.g. establishing shot, medium shot, close-up, medium shot from another angle). A continuity filter would express the fact that the dog should be the same type throughout the entire sequence. It would be combined with the temporal filter (perhaps with an intersection filter) abstracting out the continuity constraint from the temporal playout.

Options available within the continuity filter include “match slot value to last clip’s slot value”, “match slot value to next to last clip’s slot value”, “match slot value to first clip’s slot value”. The inverse of each of this is also available (e.g. “do not match slot value to last clip’s slot value”). This provides “continuity” for the movie at the most basic level. For example, a continuity filter could be used to make sure all the clips in a movie featured the same character. More complicated continuity structures can be built with the eval filter.

6.1.7.2. Suppress Duplicates Filter

The suppress duplicates filter removes clips which have already been presented to the viewer (i.e. appear in the clip transcript). This is often a desired behavior when several similar types of clips might be used in repeated sequences.

6.1.8. Filter Combinations

Most of the power of using FilterGirl as a language for dynamic content selection comes from combining the many different types of filters in interesting ways. Some of the simplest combinations are easy extensions of the filter types mentioned above. For example, the intersection filter can be used to combine two or more description filters and create a more closely specified selection of the database (e.g. clips in which the characters Nicole and Woody appear on the beach flying a kite).

Another interesting use of the intersection filter is to combine the different types of template filters. For example, we could use a intersection filter to combine two shot template filters to create a dialog between two characters. The first template filter would define the structure of the dialog (e.g. a question then a story then a reaction and so on) while the second template would define the cutting back and forth between the two characters (e.g. first show a clip with the character Dave then the character Thomas). The

intersection filter would take the intersection of these templates at each point resulting in a question from Dave then a story from Thomas and so on. In this way we have abstracted away the structure of the dialog from the character specification. Now, assuming our clip database is rich enough, we can change the dialog structure without modifying the way the characters appear and we can modify the characters that appear without changing the nature of the dialog (e.g. an argument between the two characters or a normal dialog between new characters).

Many other interesting types of combinations can be imagined such as advancable template filters which advance based on signals from interaction filters, advancable template filters which advance based on signals from rule filters, eval filters which create time template filters (to give the viewer control over playout time) or time templates combined with repeated shot templates.

6.2. Creating and Editing Filters

Instances of all of the filter types can be created and edited interactively with the FilterGirl application. The command center of FilterGirl is the “filter control window”. The filter control window contains a list of all currently defined filter instances along with a list of currently running filters and controls for editing and manipulating the filters. Figure 6.6 shows a snapshot of the filter control window during a typical session with FilterGirl. The list in the upper left corner includes all currently defined filter instances while the list at the bottom includes all of the currently running filters. Remember, even if a filter is not shown in the running list, it can be called during a filter session if it is contained within and called by one of the running filter. The buttons along the right hand side allow the author, among other things, to start a playout with the currently running filters, create new filters and open editing windows for filters.

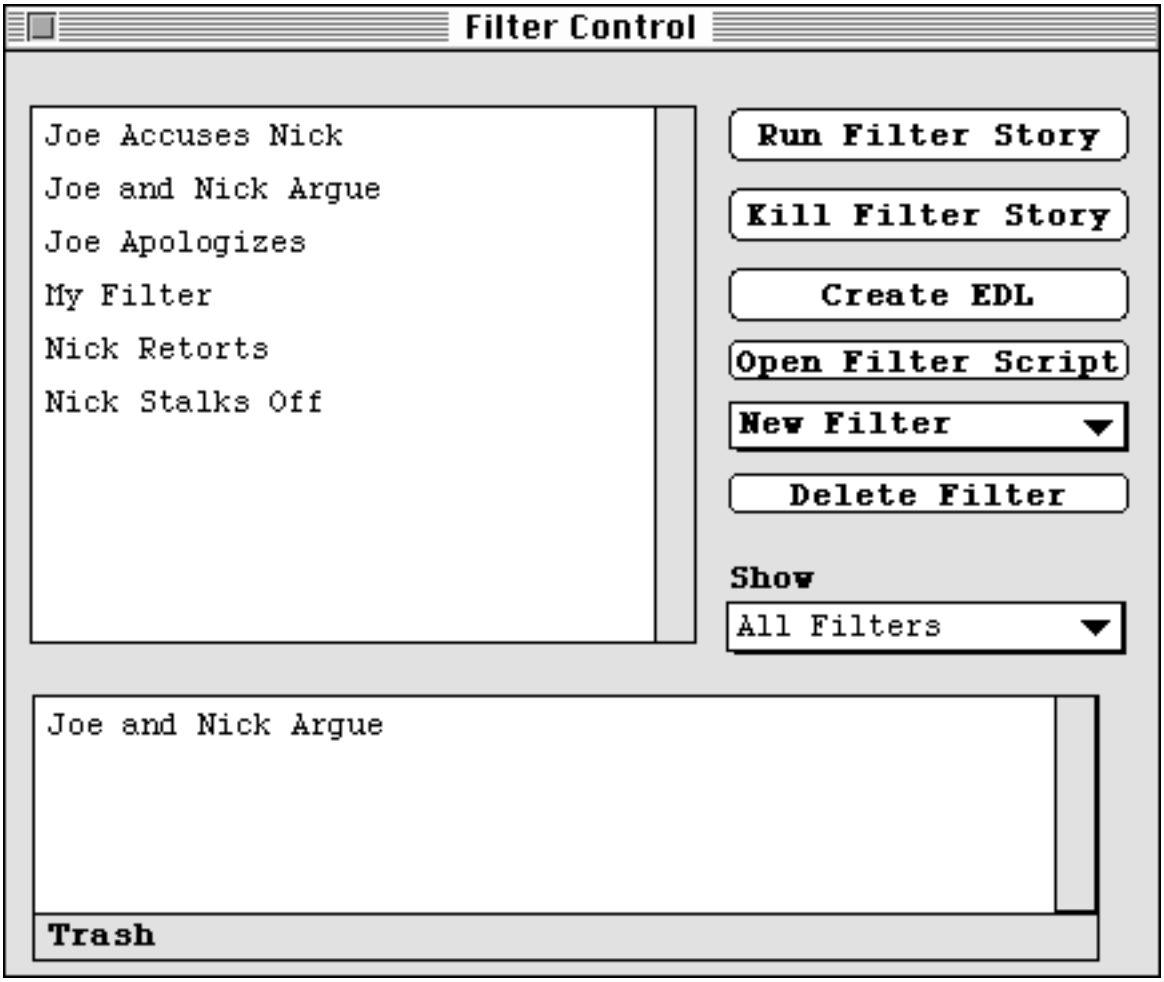

Figure 6.6 The filter control window during a typical FilterGirl session.

To create a new filter instance, the moviemaker selects a filter type from the "new filter" pull down menu and a new instance is created. An editing window is opened for the new instance that allows the author to change the internal parameters which vary the behavior of the filter. Opening edit windows for existing filter instances takes place in a similar way. The filter instance is selected in the top list and the "open filter script" button opens the editing window. Double clicking on a name in any filter list (including lists of filters included within other filters) has the same effect.

Figure 6.7 shows a prototypical filter editing window. It is an editing window for the basic filter type and includes features common to all filter editors. All editing windows display the filter type, the filter name and comments about the filter. They also include a button which allows the user to redefine the attributes at any point. Redefining the filter records all of the currently displayed (perhaps new) information as the parameters of this filter instance.

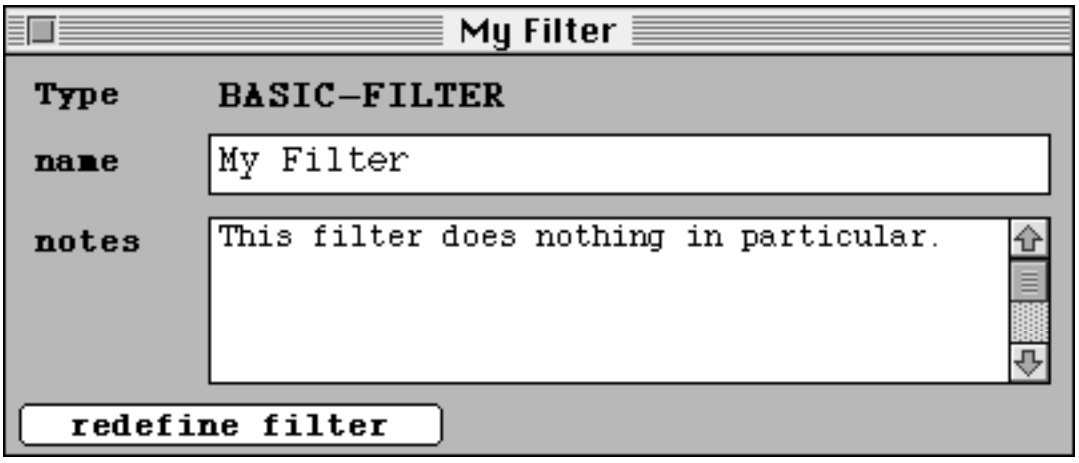

Figure 6.7 An editing window for an instance of the basic filter type.

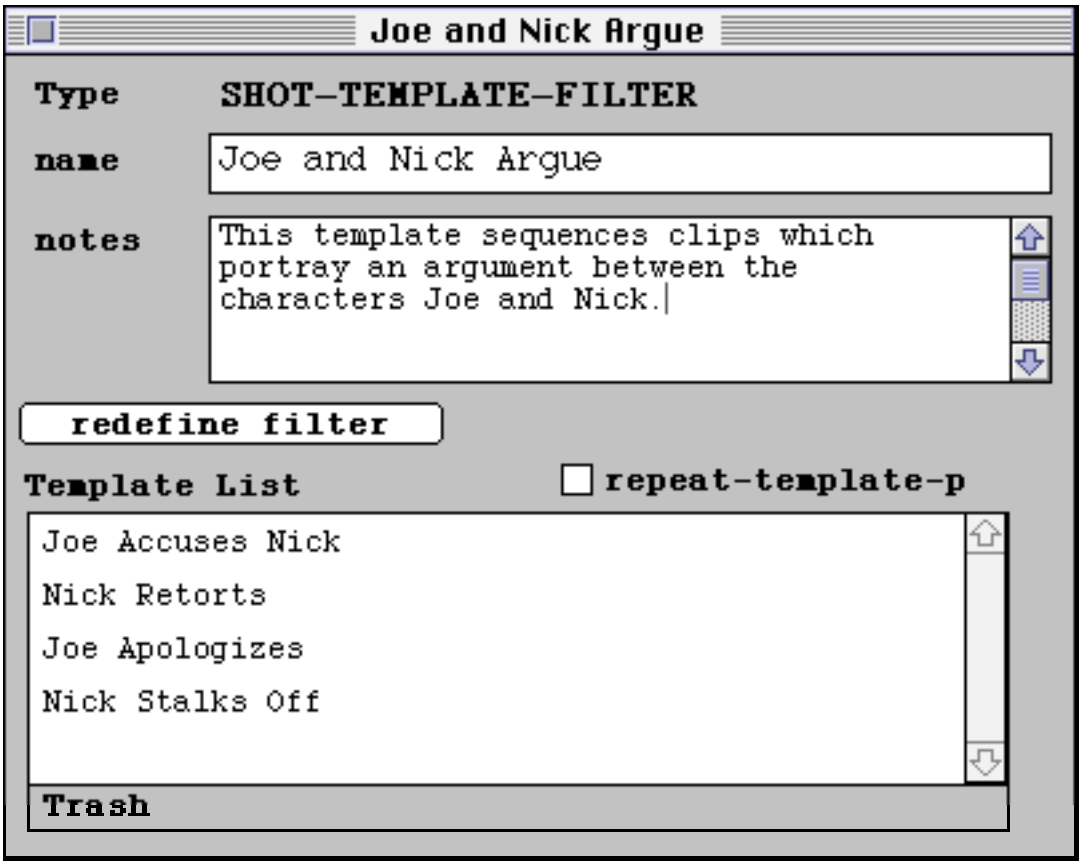

Figure 6.8 An editing window for an instance of the shot template filter type.

Edit windows for more complicated filters add more parameters below the common parameters. Figure 6.8 shows an edit window for a typical instance of the shot template filter. In addition to the filter type, instance name and comments there are two more parameters which correspond to the parameters described in the section on shot template filters above. First, the

template list contains an ordered list of all of the filters within this shot template. Filters can be added by dragging and dropping from any other filter list (including the filter control window). Filters can be removed from this list by dragging to the “trash” icon at the bottom. Second, the repeat-template-p checkbox is a flag which tells the shot template filter to repeat when it reaches the end (rather than sending an end of filter signal). Edit windows for other filter types work in similar ways by adding interfaces for the various parameters below the standard information.

Filter sets can be saved out through the menu bar. By selecting “Save Filter Set” from the main menu, the author is prompted for a file name to which all currently defined filters are to be saved. Similarly, a filter set can be loaded through the same menu.

6.3. Running a Filter Story

FilterGirl provides facilities for previewing filter based story structures while filters are being built. This requires not only that a set of filters be defined in FilterGirl, but also that a clip database created in LogBoy be loaded. Loading a clip database is accomplished through FilterGirl’s main menu. Previewing takes place by pushing the “run filter story” button on the filter control window. This opens a playout window which displays the movie as a viewer would see it. The playout window includes buttons to pause, resume and skip ahead one clip in the current playout.

Previewing the movie is accomplished by applying in sequence the filters in the running filters list to the loaded database. Each application of the filters results in some non-empty subset of clips. One is chosen arbitrarily and presented to the viewer. When FilterGirl has finished presenting the clip, the filtering/presentation process is repeated. This continues until an end of filter signal or an end of movie signal is passed out of one of the filters in the running filters list. Either of these signals will end the current playout.

There is one caveat to the layered filtering process explained in the previous paragraph. The empty set rule (described at the end of Chapter 3) is in effect during the filtering process. This means that the filters in the running filter list are applied in top to bottom order. If any of the filters returns an empty set of clips then FilterGirl stops the filtering process and returns (as its ultimate output) the input to the filter which returned the empty set. This provides a simple way to assure that FilterGirl will never come up empty handed when trying to select a clip to present to the viewer. This rule also means that the moviemaker must prioritize the running filters. In this

prioritization scheme, the most important filters (e.g. basic story structure) come first in the list while the less important filters (e.g. user interaction, stylistic structures) come later in the list.

6.3.1. Debugging Output

Used by itself, the playout window simulates the viewer's experience closely, but provides a poor representation of the inner workings of the layered filtering process. To help provide a clearer view of how the filters manipulate the database FilterGirl has a debugging output which presents textual information regarding the input and output of specific filters in the running filter set.

The debugging output feature of FilterGirl can be turned on via the main menu. In the debugging output the filter layers are listed down the left-hand side of the text window while a histogram of the number of clips presented as input to each filter layer is printed on the right-hand side of the window. The two streams of information are interleaved so that the filter layers are synchronized with their input and output on the histogram (i.e. the number of clips fed into the filter layer is printed on the line directly above the filter layer's line and the number of clips output from the filter layer is printed on the line directly below). Additionally, each filter layer name is indented to represent how deep it is within the filter hierarchy. Unindented filter names are at the top level (i.e. the running filters list in the filter control window) while each successively indented filter name is contained within the previous filter name. This interleaved filter and clip histogram output is repeated for each clip selection. When the empty set rule is called into effect (i.e. when one of the filter layers returns an empty set of clips) a warning is printed in the debugging output window.

```
*********************************************
>> Joe and Nick Argue <<
*********************************************
>> Joe Accuses Nick <<
*********************************************
>> Speaker = Joe <<
**********************
>> Utterance Type = Accusation <<
*********
>> Listener = Nick <<
**
>> Suppress Duplicates <<
*

Selected Clip: “Digital Film Clip 23”


*********************************************
>> Joe and Nick Argue <<
*********************************************
>> Nick Retorts <<
*********************************************
>> Speaker = Nick <<
*************
>> Utterance Type = Rebuttal <<
*******
>> Suppress Duplicates <<
***

Selected Clip: “Digital Film Clip 08”
```

Figure 6.9 Part of a typical debugging output.

Figure 6.9 shows a typical FilterGirl debugging output for two clip selections. There are two top level filters in this scenario. The first is a shot template filter named “Joe and Nick Argue” while the second is a suppress duplicates filter named “Suppress Duplicates”. In the first selection the filter “Joe Accuses Nick” has been selected from the shot template filter list. It is a intersection filter which contains three description filters which each successively narrow the selection. The suppress duplicates filter narrows the selection to a single clip which is then presented to the viewer. The second clip selection is very similar, except now the next filter in the shot template filter list has been selected and it contains two description filters.

The debugging output provides a good picture not only of how each filter affects the clip selection process, but also exactly which filters are called upon during each stage of the movie experience. This provides an interactive previewing and debugging environment for multivariant movies.

6.3.2. Creating an “EDL”

FilterGirl has a second previewing option which can be used to create multiple linear non-interactive edits from a set of filters. This feature, rather than actually playing each digital movie file, creates a list of the pathnames of each digital movie file. This list can then be used to create a linear digital movie which represents a specific playout of the multivariant movie. The list of pathnames serves a similar function to creating an edit decision list (or EDL) in a traditional video editing suite. Conceivably the pathname based EDL created by FilterGirl could be used as a template for creating higher quality linear analog edits of the same material if a lookup table was created to translate between digital movie pathnames and time code numbers on a physical video tape.

7. System Implementation

This chapter briefly describes the implementation of LogBoy and FilterGirl and the issues involved. There are four sections pertaining to the four major software subsystems: video, database, filters, interface.

7.1. Video

The LogBoy and FilterGirl applications currently use Apple's QuickTime software as a disk-based digital video standard. Disk based digital video storage provides the random access video capabilities essential to computational movie generation. Digital video has several benefits in addition to other random access formats (e.g. videodisc). First, images from the video source material can be easily sampled and manipulated within the screen space. One example of this is LogBoy's video clip icons which incorporate a still frame. Second, QuickTime is a read/ write format. This means that source material can be added to the database quickly and easily. Read/ write formats allow a more flexible creative process as the moviemaker is not committed to a particular set of source material at any particular point in the production process. This also opens the door on serialized movies to which the creator is continually adding content. Finally, QuickTime digital video can be edited on the desktop using a Macintosh computer. Software packages allow the creator to digitize, compress, edit and present footage all on the same piece of hardware.

7.2. Database

The video clip database holds the locations of the digital video clips along with the descriptive slots and values attached to each clip. The LogBoy application must be able to create, present, edit, load and save this database structure while FilterGirl must be able to load and access the descriptors. Framer[Haase 92] was chosen for the database implementation because it supports all of these characteristics along with providing other sophisticated features which can be made use of in the future.

Framer's hierarchical structure maps easily onto LogBoy's extensible slot/ value database architecture. Framer files are implementation independent and can be read with libraries provided for several platforms in several computer languages. This flexibility allows moviemakers to experiment with

data paths other than the intended LogBoy -> FilterGirl. For instance, a Framer database created in LogBoy could be loaded into some other story structure driver or databases could be created in another logger and loaded into FilterGirl. Framer also provides inferencing capabilities which can be taken advantage of by sophisticated story structure editors/ generators.

7.3. Filters

FilterGirl is designed around a hierarchical filter language which supports personalized video sequencing. This filter language is based on Macintosh Common Lisp's (MCL's) implementation of the Common Lisp Object System (CLOS). CLOS provides class inheritance which is useful in situations where there are many types of objects which have similar, but slightly different characteristics. For example, every filter type exhibits a different filtering behavior, however each filter must implement the empty set rule (explained in Section 6.3). This is implemented with a basic filter type from which all other filter types inherit their basic behavior. Thus, the empty set rule is implemented once for the basic filter type and all other filter types make use of this implementation.

7.4. Interface

Both LogBoy and FilterGirl use a window based graphical user interface to allow the moviemaker to create and edit databases and story structures. Both interfaces are created in Macintosh Common Lisp's (MCL's) implementation of interface objects which is based in CLOS. Many of the interface objects in both LogBoy and FilterGirl make use of class inheritance to reduce the amount of duplicated code when many objects share behaviors. For example, all of the filter editing window types in FilterGirl have several common interface items. The window definitions are built with a class hierarchy such that there is a basic filter editing window class which implements all of the common characteristics and each different type of filter editing window inherits from this superclass. This is analogous to the inheritance scheme for the filters themselves. Each of these specific editing window types can specialize the canonical window to suit its own needs. Similarly, LogBoy has two types of interface items, clip icons and value areas, which share the behavior of dragability. This behavior is implemented with a "draggable-object" superclass from which they both inherit.

8. Multivariant Movie Production

Building a multivariant movie is, in many ways, different from building a traditional linear movie. Traditional movie production goes through several steps beginning with a story idea in the filmmaker's mind. This idea goes through three written stages, plot treatment, script and shooting script, which get more and more specific about exactly how the movie is to be shot. Storyboards are then created which graphically layout how each shot must look. After all of the planning stages are taken care of then production can start. This involves actually recording the scenes onto film. Postproduction mainly involves editing the film, but can also include special effects, dubbing and music.

The creation of a multivariant movie begins in much the same way, however, the moviemaker's vision must encompass multiple playouts. The underlying narrative structure must account for playing out a story in many different ways or presenting many different types of stories. More practical differences appear during the building of the multivariant movie. The preproduction (or planning) process of a multivariant movie must not only include a story structure with multiple playouts, but detailed shot lists or storyboards must be constructed to insure proper coverage during shooting. The actual shooting of the multivariant movie is similar to linear production, except that much more content must be gathered to account for many playouts. Postproduction not only includes editing (in this case editing video clips rather than the entire movie), but also logging video clips and tweaking the story structure.

8.1. Preproduction

The preproduction process for a multivariant movie includes the traditional steps of refining a story idea and planning for shooting. But multivariant pre-production also includes considering the different ways in which the movie might play out and what video clips will be needed to account for these multiple playouts. The moviemaker must go into production with a detailed list of video clips to gather during production. This is because multivariant playout often calls for multiple versions of the same scene with slight differences (e.g. R rated dialog with rude words and PG rated dialog without rude words). Many similar versions can become difficult for the moviemaker to keep track of and plan for.

One way a moviemaker can begin to visualize many slightly different versions of video clips is by mapping a multivariant story onto axes of playout. Each axis of playout defines a different way in which multiple playouts of a single movie might vary. Most often these axes come from parameters selected by the viewer, but they can include variations controlled by other influences (e.g. available news stories) or random factors. For instance, a multivariant movie might allow a viewer to control the rating (either PG or R) and the amount of humor (funny or serious). In addition the movie might include a random factor which controls whether the movie takes place in Boston or New York (to add variety and try to get viewers to return for multiple viewings). This movie would have three axes of playout: rating, humor and location. These three axes define a three dimensional space which must be filled with video content. Each video clip that makes up the multivariant movie must have alternate versions which take into account every possible combination of the various axes of playout. In this case, each clip must have every combination of an R rated and a PG rated version, a funny and a serious version, a Boston and a New York version. This adds up to eight versions for each clip.

This exponential problem of adding axes of playout may seem to pose an insurmountable task for the moviemaker, but in reality it is not nearly as bad as it seems. In many instances a clip will work equally well for different settings of an axis (e.g. a close up of a character that will work in either the Boston version or the New York version). Very few clips in the movie will actually have eight different versions.

This axes technique of visualizing needed content breaks down in some cases, especially when selected video clips depend heavily their context (e.g. what clips have already been presented). But on the whole it provides a simple way of imagining the amount and types of content needed to populate a database for a multivariant movie.

Axes of playout provide one way for a moviemaker to plan shot lists for production. Another way to help plan for multivariant movie production is by using the same tools that are used to build the movie itself. The moviemaker must build some kind of multivariant story structure (either on paper or on the computer) as part of the story scripting stage. If the moviemaker were to formalize this story structure on the computer (with a tool like FilterGirl) and then simulate running the movie, a detailed list of the required video clips could be produced. The filter mechanism can work on dummy clips which contain no video (perhaps illustrated with a video frame) and compile

sequences for possible playouts. The annotations attached to the clips in these virtual playouts would then form a kind of simple shot list that the moviemaker can then flesh out to help guide production.

8.2. Production

Production for a multivariant movie is not so different from production for a linear movie. The camera must still be pointed at the characters and someone must yell “Action!” and “Cut!”. The main difference is that much more content must be gathered to cover all of the different versions of playout.

A device that could ease the amount of logging necessary during the postproduction process is some type of smart camera. This smart camera would transcribe information already known about the footage onto the recording media as it is being shot. The reason that this is possible is that at the time the footage is actually shot almost everything is known about it (e.g. what scene in the script is this portraying, what characters appear, what is the focal length of the camera, how does the camera move). Currently, the moviemaker returns to the studio and the footage itself is the only record of the shoot. Each video clip must be manually matched to the appropriate descriptions.

A smart camera like this would require a recording deck that could record textual data alongside video and audio data, a computer system with an on-line version of the script (i.e. multivariant story structure) which can interface to this text track and perhaps sensors on the camera which record lens adjustments, movements, etc. By matching each point on the video tape with a particular scene in the script during the shooting process the moviemaker has already accomplished much of the logging process. Such a smart data camera is described briefly in [DAP 91].

8.3. Postproduction

Postproduction for a multivariant movie is very different from postproduction for a traditional linear movie. The postproduction process for a linear movie generally consists of editing pieces of video content end to end to make a single long playout. Other tasks might also be required such as creating special effects or compositing footage.

Postproduction for a multivariant movie has three separate tasks that must take place after shooting. First, rather than edit the footage into one long playout video clips must be extracted and constructed from the video footage. These clips will later be sequenced for individual playouts. Second, these

video clips must be logged for use within description based story structures. Generally the categories for logging are decided upon long before shooting starts, but now the clips must be organized according to these categories. Third, the story structure must be constructed that will guide the various playouts. Usually this story structure is completely or partially constructed during preproduction, but when the annotated video is actually worked into the story structure many inconsistencies and faults will appear that must be corrected.

One interested thing to note is that logging the footage and constructing/modifying the story structure do not generally take place in sequence. Instead they are simultaneous processes which feed off of each other. As the story structure becomes complete, problems in the log will become apparent and as the log gets corrected, problems in the story structure must be fixed. This iterative process is because of the symbiotic nature of the two processes in trying to make a coherent story. Lee Morgenroth has explored this concept including the idea of a story structure driver that automatically generates appropriate video logs [Morgenroth 92]. LogBoy and FilterGirl are described in this thesis as two separate applications, however they can both be run simultaneously to provide the moviemaker with an environment in which this iterative process can take place.

8.4. Top Down/Bottom Up Movie Making

One of the interesting artifacts of using a nested story structure like layered filters is the way that the story structures get built. Much like a procedural programming language the filters for a multivariant story can be built either in top down or bottom up style. These two strategies work best for different types of movies.

Top down style story structuring seems to work best for projects where a single moviemaker is carrying a single vision from idea to movie. The top down style allows the creator to divide the movie into conceptual chapters (perhaps using a temporal filter like the advancable template filter) and then work on filling in the details of those chapters. This is much like building an outline for a novel and filling it in.

Bottom up style story structuring works well for projects in which many people are collaborating to produce a finished project. One example is “Boston: Renewed Vistas”, a multivariant movie currently under construction at the MIT Media Lab. “Boston: Renewed Vistas” uses footage which has been

collected by twelve students in a class taught by Professor Glorianna Davenport. The footage portrays the process of a huge highway construction project going on in downtown Boston. The students have collected footage on planning meetings, community meetings, interviews with officials and portraits of neighborhoods affected by the construction.

The students in the class are making use of a bottom up style of multivariant movie making. Each student edits, logs and structures their personal footage, constructing small vignettes about a place, a person or a process. These vignettes are then collected into larger multivariant pieces that might make up a portrait of a place or illustrate a stage in the construction process. These larger pieces are collected into an interface from which viewers can express preferences or interests and get a coherent movie.

This bottom up, collaborative movie making points towards a new type of movie which will allow creators to add small pieces to a larger whole and viewers to get a coherent narrative which illustrates a sampling of the entire collection.

9. “Just One Catch” In Detail

“Just One Catch”, the multivariant movie described in Chapter 2, was built using the LogBoy and FilterGirl system. The movie itself is composed of approximately 170 video clips annotated according to eight characteristics in LogBoy. Almost every defined filter type is used to build the story structure for “Just One Catch” including Boolean, temporal, contextual and interaction filters. This chapter explains in detail the video database and story structures that were built in LogBoy and FilterGirl for “Just One Catch”.

The story portrayed in “Just One Catch” can be divided into five separate scenes: opening, lunch, stealing, chasing, retrieving. The opening scene shows the entrance of Ian into the park and creative credits for the movie. The lunch scene includes the three people that meet Ian on the park bench while he is eating lunch. The stealing and chasing scenes portray the skateboarder stealing and running with Ian’s Super-8 camera. The retrieving scene ends the movie with Ian getting his camera back.

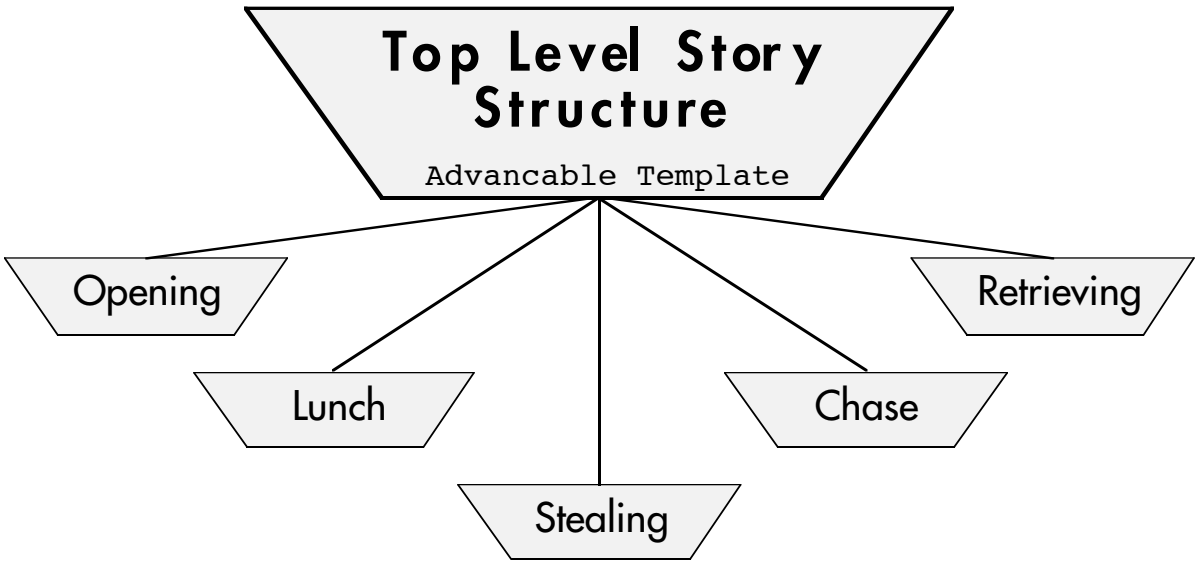

Figure 9.1 The highest level filter structure showing the five scenes of the movie.

This five part structure is echoed at the highest level of the filter set which makes up “Just One Catch”. The top level filter, called “Top Level Story Structure”, is an advancable template filter made up of five filters. Each of the filters presents one scene in the movie. Each of the filters is run in

sequence to create a playout of the movie. This five part structure is illustrated in Figure 9.1.

"Just One Catch" is different from many interactive movies in that it always begins and ends in the same way. The middle of the movie, however, can play out in hundreds of different ways. The first scene, the opening, is always the same. To accomplish this the "Opening" segment in the top level advancable template filter simply contains a shot template filter with a single description filter which selects the opening clip. The opening clip includes the credits and Ian walking down the park path, sitting down on a bench and opening his lunch bag.

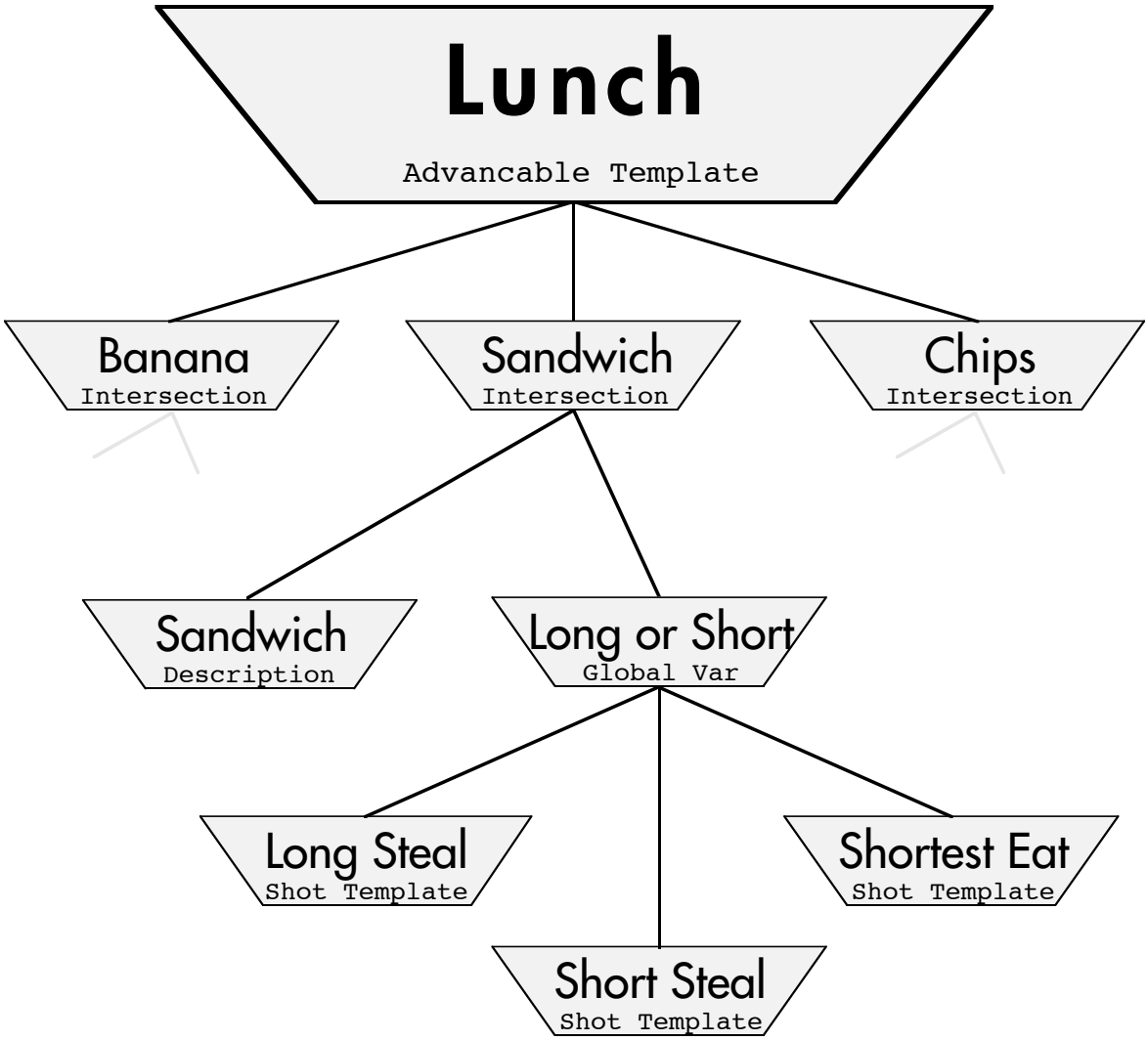

Figure 9.2 The filter structure for the lunch scene showing the three segments (one for each lunch item) and the global variable filter that selects one version of each.

The second scene, the lunch scene, consists of three distinct parts: eating the banana, eating the sandwich and eating the chips. Thus the second filter in the "Top Level Story Structure" filter is another advancable template filter called "Lunch Scene" which holds three filters: "Banana Segment", "Sandwich Segment" and "Chips Segment". This structure can be seen in Figure 9.2.

"Just One Catch" can be manipulated by the viewer to have a longer or shorter playout and to have more action or more dialog. The three segments of the lunch scene will mold themselves to fit these preferences. To create a longer total movie or to increase the amount of dialog in the movie there are three possible versions of each lunch segment. These three versions, ranging from longest to shortest, include a long version in which a secondary character (Graham, the dog or the bum) talks Ian out of one part of his lunch, a shorter version of the secondary character talking Ian out of his lunch and the shortest version in which Ian simply eats that part of his lunch.

Each video clip in the lunch scene is described with which lunch item Ian is eating (banana, sandwich, chips) and which version is portrayed in the clip (short, medium, long).

The selection of shorter or longer segments in the lunch scene is accomplished through the use of a global variable filter which is intersected with a description filter. The description filter ensures that only clips portraying the correct lunch item are selected while the global variable filter is used to select a long, medium or short version of the segment. The viewer's sliders which manipulate the total length parameter and the action vs. dialog parameter feed into a global variable in the FilterGirl environment. These global variables then influence exactly which version of each segment is chosen.

An interesting consequence of the fact that three different versions of each of three different segments can be selected is that variations in continuity must be taken into account. For instance, if a segment portraying Ian eating his banana and then laying the peel on the bench beside him is shown then subsequent shots of the bench must show the banana peel. If instead Graham talks Ian out of his banana then subsequent shots of the bench must show no banana or banana peel.

This matching is taken care of with continuity filters within each lunch segment which select different shots of the bench. Thirty-two different versions of a close-up of the bench were collected which include all possible combinations of present and absent lunch items. Each of these thirty-two different closeups is described with which lunch items are present and which

item Ian is picking up to eat next. A continuity filter selects which combination of items is shown based on which versions of lunch segments (short, medium, long) were previously shown.

The stealing scene is much like the opening scene in that it always plays out in the same way. It is represented by one clip in the database and the filter at this point in the movie always selects this clip.

The chase scene also changes the way it plays out based on the viewer's preferences. Each playout of the chase scene is constructed of several chase segments. These segments are in turn constructed shot by shot from the database. To create a longer playout or to add more action to the movie a global variable filter selects how many chase segments to add to a particular playout (more segments for a longer movie or more action).

Video clips which make up the chase scene portray Ian and the skateboarder running through different locations. These are described with where they take place (one of six locations), how the shot fits into the development of a chase segment (entering into a space, exiting from a space, insert shot, whole) and what point of view is used (Ian's, the skateboarder's or neutral).

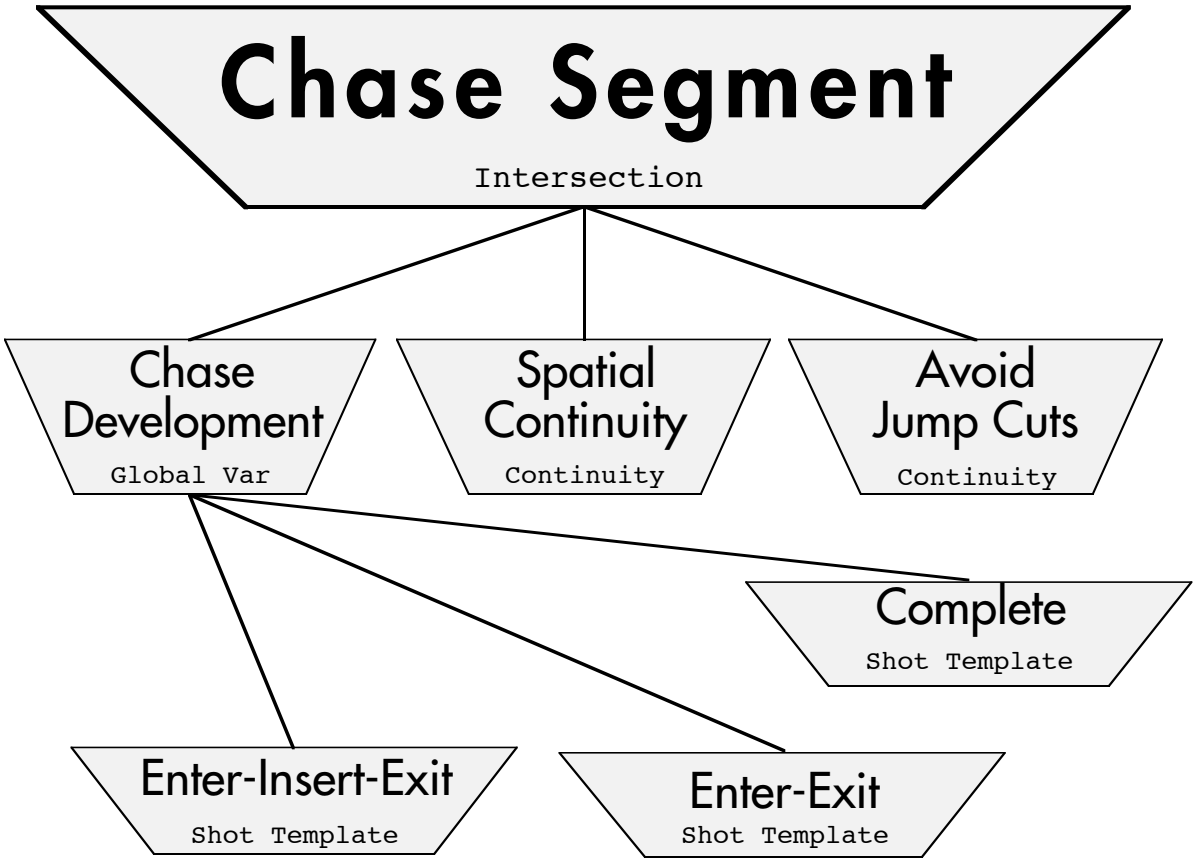

Figure 9.3 The filter structure for a single chase segment. Several of these are sequenced using a global variable filter to create the chase scene.

Each chase segment is built based on three filters which take advantage of these descriptions. The "Chase Segment Development" filter takes care of the temporal development of the segment. The "Spatial Continuity" filter takes care of changing locations logically. The "Avoid Jump Cuts" filter handles basic cinematic language constraints. These three filters are combined with an intersection filter so that they are run in parallel on the database (i.e. all three constraints affect the playout of a chase segment). The filter structure of a chase segment is illustrated in Figure 9.3.

The first of these filters, the "Chase Segment Development" filter, builds each chase segment temporally. This is accomplished with one of three shot template filters (selected randomly with a select random filter): enter-insert-exit, enter-exit or whole. The enter-insert-exit shot template filter builds the chase segment across three shots: entering into a space, an insert (close-up) of one of the characters, exiting from the space. The enter-exit shot template does the same thing in two shots with no insert shot. The whole template selects only video clips which portray Ian and the skateboarder entering and exiting from the space in one shot. By structuring each segment around an

entrance and an exit it becomes possible to sequence chase segments in arbitrary orderings.

The “Spatial Continuity” filter simply makes sure that each video clip selected for a chase segment takes place in the same physical location. This is accomplished with a continuity filter which matches the location slot to the first location of the chase segment.

The “Avoid Jump Cuts” filter defines one aspect of traditional cinematic language, the jump cut. In the simplest terms a jump cut can be defined as a discontinuity in movie time (a cut) with no change in the physical location of the camera. Each video clip in the chase scene is described with a point of view which indicates where the camera is physically located relative to the action. Clips described as having an “Ian” or “skateboarder” point of view are shot from that character’s location. Clips described as having a “neutral” point of view are shot from some other third-person viewpoint. The “Avoid Jump Cuts” filter is a continuity filter which makes sure that the point of view of any clip in a chase segment does not match the point of view of the most recently present clip. This avoids jump cuts in the simplest way by making sure that the movie never cuts to a camera angle which is similar to the most recent camera angle.

The final retrieving scene is the same for every playout so it is represented by a single video clip in the database. The filter active at this point in the playout always selects this clip.

All together, “Just One Catch” was constructed using approximately 170 video clips, 8 descriptive slots containing 42 descriptive values and 34 filters of 9 different types.

10. Related Work

This chapter illustrates previous work related to the LogBoy and FilterGirl system. The first section looks at precedents in database interfaces and video databases. The second section traces work in description based story structures.

10.1. Database Interfaces

LogBoy's graphical layout of database information has its roots in the Spatial Data Management System (SDMS) of MIT's Architecture Machine Group [Bolt 80; Bolt 79]. SDMS provided the user with a graphical "world view" populated with iconic representations of photographs, letters, television sets and hand-held calculators. A large screen close-up of the world view permitted the user to examine and manipulate a particular section of the data world. While SDMS allowed graphical perusal of information (including video information) and some limited manipulation of data structures, the data structures and interface were hard-coded. A later SDMS prototype developed at the Computer Corporation of America by C.H. Herot[Herot 80] graphically displayed data derived from a symbolic database.

More recent graphical interfaces for querying symbolic databases can be found in Chris Hancock's Tabletop system [HKG 92] and in Anselm Spoerri's Query Spreadsheet [Spoerri 93]. The Tabletop system creates icon based displays using Venn diagrams and/or two-dimensional tables. The display space is similar to LogBoy's slot windows, however data can only be viewed graphically. Any manipulation of the underlying data structures must be carried out on a textual representation of the database. Spoerri's Query Spreadsheet provides for a graphical visualization of complex database queries which can give a comprehensive overview of multiple combinations of Boolean query constraints. This is accomplished through an iconified generalization of Venn diagrams.

A system which makes use of positionable video micons in a two-dimensional space is Eddie Elliott's Video Collage [Elliott 93]. This system acts as a kind of "video sketchpad", allowing the user to freely grab, arrange and analyze multiple pieces of digital video footage. In the current version, symbolic annotations cannot be attached to video footage, however bitmapped sketches can be superimposed on the collage to provide visual cues to the user.

10.2. Video Databases

Amy Bruckman's Electronic Scrapbook [Bruckman 91] includes a video annotation system with design goals similar to LogBoy. The Electronic Scrapbook's logger is intended for describing home movie footage for use in a story template system which would automate some of the editing process. The Electronic Scrapbook makes use of an inferencing mechanism which aids the moviemaker by trying to "guess" descriptions for new pieces of footage. The logger makes use of a chunk-based temporal representation, but the logging interface uses a linear logging paradigm which precludes logging within a descriptive context.

Thomas Aguierre Smith's Stratograph [AS 92] and Marc Davis' Director's Workshop [Davis 91] represent two approaches to logging video using a stream-based temporal representation. The Stratograph system was originally designed for annotating video footage shot in the course of scholarly research. The stream-based representation provides an ideal setting for the multiple layers of description that must be added to the video. The Stratograph gives the user a keyword-based descriptive language that organizes keywords into "keyword classes". Keyword classes help the user to maintain consistent annotations across several logging sessions. These keyword classes were echoed in the way LogBoy presents value areas within slot windows.

Davis' Director's Workshop also makes use of a stream-based representation for video annotation, however the descriptive language is very different. The descriptive language is based on a hierarchy of icons intended to present a complete ontology for logging video. This approach is intended to allow multiple users to retrieve video from large databases by constraining the descriptive language to a canonical form.

10.3. Descriptive Story Structures

Amy Bruckman's work on the Electronic Scrapbook also included a facility for modeling stories. Users who had logged their home movie footage could then automatically generate simple stories by selecting from a range of story structures. These structures included themes such as "our son at age two" and "important firsts in the lives of our children". The story structures are built using selection criteria to produce a set of relevant video segments. These segments are then sorted based on chronology and filtered based on other criteria such as "sensitive subject matter" or "bad audio" to produce a sequence.

Homer is a system developed by Lee Morgenroth [Morgenroth 92] which works with Aguierre Smith's Stratograph to allow moviemakers to produce linear movies more efficiently. Moviemakers are presented with a time line which represents the final linear presentation of the movie. Descriptions are then layered onto this timeline and searches into the video database are used to fill the timeline based on these descriptions. The moviemaker then refines the descriptions until the desired movie is achieved.

A final project, Ben Rubin's Constraint-Based Cinematic Editing system [Rubin 87], used three fixed filtering layers to produce a movie with multiple playouts. Each of the layers would look for repeated action, length of clip, narrative importance and cinematic grammar to produce personalized playouts for viewers. Viewers were able to change how long the entire playout should take and what the pacing should be.

FilterGirl builds on top of all of these works by providing a more generalized story-structure environment to moviemakers. FilterGirl is not fixed into a already existing ontology for describing video clips, allowing creators to log only characteristics of the video which are important. Also, FilterGirl provides ways of building contextual constraints across playouts. This can help creators to produce multivariant movies which adhere to traditional continuity rules across multiple playouts.

11. Future Directions

There are many ways that the LogBoy/FilterGirl system could be extended in the future. These include enhancements to the logging and story structure interfaces, reworking the underlying filtering paradigm and experimenting with new types of movies suggested by a description based story structure system.

One of the biggest problems, currently, with using FilterGirl is the fact that FilterGirl's filtering mechanisms do not provide any type of lookahead into the playout of the movie. This can present problems when the database is not fully populated. For instance, at a particular point in a chase scene between two cars the filters might select two appropriate clips: a clip of the two cars turning onto a residential street or a clip of the two cars entering onto the highway. If the database does not include the correct clips to allow the chase scene to resolve on the highway, but does include clips to allow resolution on the residential street then the residential street clip should be chosen. This choice would be based on availability of subsequent clips.

Currently, FilterGirl does not provide any way to select clips based on the availability of subsequent clips. One simple way to do this is to simulate running the movie to completion every time a clip is selected. This means that when the filters present several possible clips for a point in the database each one could be checked for viability by seeing if the movie would playout to completion without running into a dead end (i.e. a situation with no available clips). Some kind of debugging interface could also be provided to help the moviemaker to find out exactly where and how the movie is breaking down.

FilterGirl's interface is currently usable, but not a very pleasant environment for creating stories. In general, too many windows must be created even to construct a simple story. A simpler interface should be provided that allows the moviemaker to easily scan all of the defined filters. This might be similar to outlining tools that writers often used during word processing. Filters which contain other filters could be "folded" and "unfolded" to hide or reveal their inner structure. A scripting language for creating filters is another possibility.

LogBoy's interface currently works very well, however it could use a more traditional "clip at a time" interface for fine tuning descriptions. The interface

would take the place of the current clip description window. It would allow the moviemaker to not only view the descriptions attached to a particular clip, but also modify them. This would require an algorithm for automatically moving clip icons within slot windows when their descriptions are changed through this new interface.

New types of movies can be created with tools like LogBoy and FilterGirl. One way is to abandon the traditional moviemaker/viewer model that has been used throughout this thesis. With LogBoy and FilterGirl viewers can take on roles that are now the responsibility of the moviemaker.

For example, imagine a system which includes several predefined categories for logging video footage and a few predefined filter sets. The viewer (now closer to a creator) logs footage based on the predefined categories and then runs the filter sets on these newly created video logs. This could provide viewers with a way of categorizing their personal footage and easily creating simple stories. Amy Bruckman's "Electronic Scrapbook" system does something similar to this for home movies.

Other new combinations of the moviemaker's and viewer's roles are possible. Viewers could build their own filter sets from scratch for previously logged footage, possibly making their own music videos or designing business presentations. Several moviemakers could collaborate on a single movie by each providing a set of annotated video which will then be combined with other video and appropriate filter sets. Glorianna Davenport's "Boston: Renewed Vistas" project is one model for this type of collaboration. Also, a moviemaker could create a type of movie serial in which each episode does not consist of a complete movie in itself, but instead a set of annotated footage that incrementally extends an existing database. As new footage was added new sequences and stories would appear for the viewer.

12. Conclusion

It has only been possible for the past few years to consider creating a movie under viewer control with multiple playouts. Description-based story structures seem to be the most promising method of building these multivariant movies. Description-based structures allow moviemakers to structure narratives that are based on descriptions of content rather than on hard-coded links to specific pieces of content. This is an advantage in terms of extensibility and understandability. Most importantly, description-based structures can provide viewers with personalized playouts that are also uninterrupted by periodic interaction.

Description-based story structures rely on video databases with complete, correct annotations. This means that video loggers must be designed specifically for building description-based movies. LogBoy addresses this in several areas, including temporal representation, descriptive depth and descriptive context. It provides moviemakers with an environment in which relevant descriptions can be simply and easily created, scanned and examined.

FilterGirl gives moviemakers a way to think about building description-based story structures that is centered around sets of layered filters. Individually, these filters are simple selection mechanisms, but when combined in various ways can create multivariant movies with complex dependencies and sequencings.

Description-based multivariant movies not only require revisions in the ways movies are currently produced, but also hint at new mediums for narrative expression. These new forms not only encompass movies which play out in multiple ways, but also new types of collaborations between moviemakers and viewers. Multivariant movies blur traditional distinctions between moviemaker and viewer as it becomes possible for viewers to personalize playouts and create original movies using existing video databases and story structures.

13. Bibliography

[AS 92] Aguierre Smith, Thomas. *If You Could See What I Mean...: Descriptions of Video in an Anthropologist's Video Notebook*. Master's Thesis (MIT Media Lab: 1992).

[ASD 92] Aguierre Smith, Thomas and Glorianna Davenport. "The Stratification System: A Design Environment for Random Access Video." In *Proceedings of Workshop on Networking and Operating System Support for Digital Audio and Video* (San Diego, CA: 1992).

[Bates 90] Bates, Joseph. "Computational Drama in Oz." In *Working Notes of the AAAI-90 Workshop on Interactive Fiction and Synthetic Realities* (Boston, MA: 1990).

[Bolt 80] Bolt, Richard. " 'Put That There': Voice and Gesture at the Graphics Interface." In *Proceedings of SIGGRAPH '80* (1980).

[Bolt 79] Bolt, Richard. *Spatial Data Management*. (Cambridge: MIT, 1979).

[Bruckman 90] Bruckman, Amy. "The Combinatorics of Storytelling: *Mystery Train* Interactive." Interactive Cinema Group: Working Paper (MIT Media Lab: April 1990).

[Bruckman 91] Bruckman, Amy. *The Electronic Scrapbook: Towards an Intelligent Home-Video Editing System.* Master's Thesis (MIT: 1991).

[Davenport 87] Davenport, Glorianna. "New Orleans in Transition, 1983-1986: The Interactive Delivery of a Cinematic Case Study." Remarks given at The International Congress for Design Planning and Theory (Boston, MA: August 1987).

[DEH 93] Davenport, Glorianna, Ryan Evans and Mark Halliday. "Orchestrating Digital Micromovies." *Leonardo* (**26**-4: 283-288, 1993).

[DAP 91] Davenport, Glorianna, Thomas Aguierre Smith and Natalio Pincever. "Cinematic Primitives for Multimedia." *IEEE Computer Graphics and Applications* (67-74, July 1991).

[Davis 91] Davis, Marc. "The Director's Workshop: Semantic Video Logging with Intelligent Icons." In *Proceedings of AAAI-91 Workshop on Intelligent Multimedia Interfaces* (Anaheim, CA: 1991).

[Elliott 93] Elliott, Edward Lee. *Watch • Grab • Arrange • See: Thinking With Motion Images via Streams and Collages.* Master's Thesis (MIT: 1993).

[Furnas 86] Furnas, George W. "Generalized Fisheye Views." In *Proceedings of CHI '86: Human Factors in Computing Systems* (Boston, MA: 1986).

[Galyean 92] Galyean, Tinsley A. "Continuous Variables for Interactive Film." Computer Graphics and Animation Group: Working Paper (MIT Media Lab: February 1992).

[Haase 92] Haase, Ken. "Framer: A Database System for Knowledge Representation and Media Annotation." Music and Cognition Group: Technical Note (MIT Media Lab: August 1992).

[HKG 92] Hancock, C. M., J. J. Kaput and L. T. Goldsmith. "Authentic Inquiry With Data: Critical Barriers to Classroom Implementation." *Educational Psychologist* (**27**-3: 337-364, 1992).

[Herot 80] Herot, C. F. "Spatial Management of Data." *ACM Transactions on Database Systems* (**5**-4: 493-514, 1980).

[Iuppa 88] Iuppa, Nicholas V. *Advanced Interactive Video Design: New Techniques and Applications*. (White Plains, NY: Knowledge Industry Publications, 1988).

[Meehan 81] Meehan, James. "Tale Spin". In *Inside Computer Understanding*, edited by R. Schank and C. K. Riesbeck. (New Jersey: Lawrence Erlbaum Associates, 1981).

[Morgenroth 92] Morgenroth, Lee Hayes. *Homer: A Video Story Generator.* Bachelor's Thesis (MIT: 1992).

[Perey 92] Perey, Christine. "A New Map of Boston." *The QuickTime Forum* (**2**-3: 1-6, 1992).

[Rubin 87] Rubin, Benjamin. *Constraint-Based Cinematic Editing.* Master's Thesis (MIT: 1987).

[Spoerri 93] Spoerri, Anselm. “Visual Tools for Information Retrieval.” In *Proceedings of Visual Languages ‘93* (Bergen, Norway: 1993).