M-Studio: An Authoring Tool

for Context-Aware Mobile Storytelling

by

Carly M. Kastner

S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2002

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

[February 2003]

December 12, 2002

Author _____ 12/12/02

Department of Electrical Engineering and Computer Science
December 12, 2002

Certified by _____ 12-12-02

Glorianna Davenport
Principal Research Associate, Interactive Cinema Group
Media Laboratory
Thesis Supervisor

Accepted by _____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER

M-Studio: An Authoring Tool
For Context-Aware Mobile Storytelling

by

Carly M. Kastner

S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2002

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

December 2002

ABSTRACT

The pervasiveness of high-speed wireless networks and handheld computers provide a channel for context-aware video delivery. Mobile cinema is a new form of motion picture experience in which discrete cinematic events are delivered based on a consumer's navigation through space and time, via this new channel. M-Studio is an authoring tool that helps mobile story creators design and simulate location-based narratives. The tool provides the author with a graphical interface for linking content with a specific geographical space, a framework for developing story structures for multi-threaded narratives, and a simulator that allows the author to evaluate the story threads that might unfold depending on the path taken by the viewer. The tool also directly generates the XML code that is used by a story server to deliver cinematic sequences to handheld devices. M-Studio has been used in the creation of two mobile narratives.

Thesis Supervisor: Glorianna Davenport
Title: Principal Research Associate, Interactive Cinema Group, MIT Media Laboratory

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

The pervasiveness of mobile networks and handheld devices has created an enormous potential for context aware research. Furthermore, the increased availability of 802.11b networks, with data transfer rates of 11Mbps, has made the playback of video over network feasible for handheld devices. Although the use of high-speed mobile networks has overcome the hurdle of storage space for video on mobile devices, there are still limiting factors. With their small screen size and low resolution, handheld devices cannot compete with desktop computers and televisions for standard video playback. However, the mobile device's context aware capabilities can be used to augment the user's viewing experience.

One particularly powerful piece of information is the user's location. With this knowledge, a system can be designed such that users can experience video in the place it was actually shot, enhancing their viewing experience by using the surrounding location to create an immersive environment. Furthermore, a handheld device is always with users, allowing the story to come to them as they travel through space and time.

This wireless-enabled location-aware handheld device can become a platform for mobile cinema. Mobile cinema is a motion picture experience that is received over a wireless network by a human who is moving through space and time. In the case of a full entertainment experience, mobile cinema may involve tens, hundreds, or even thousands of cinema objects and sequences. Designed story structures will require that these scenes be delivered discretely in space and time and perhaps relative to other contextual conditions as well. The characteristic that a multiplicity of scenes can be delivered discretely in time and space fundamentally distinguishes mobile cinema from other cinematic forms.

In order to explore mobile cinema, we developed M-Views and M-Studio, consisting of a Compaq iPAQ handheld computer equipped with a wireless network card and a location detection system, a central server that delivers story events client iPAQs based on their user's situation in space, time, and story. M-Studio is an authoring tool designed to aid in the creation of mobile cinema.

Every genre of cinema and method for content delivery presents its own unique challenges to story creators. It requires authors and production teams to collaborate in the planning and development process, changing scenes and storylines as the script evolves. Some of the specific challenges of mobile cinema include the association of content with context

information, understanding the multiplicity of story experiences made possible by the viewer's navigation through space and time, and reproducing the discrete, non-continuous presentation of the video that will be experienced by a viewer. The author may also have to confront the technical details of the delivery mechanism. These challenges can be partially addressed by an authoring tool that provides story visualization and simulation capabilities from a context-aware perspective.

One of the first problems confronted by any story designer is the task of tailoring a story for its delivery medium, in this case, delivery to a handheld device over a mobile network. The mobile, context-aware platform has many features that can be used to enhance stories, but it can be difficult to decide how to incorporate them into a story. First, an author must decide what type of context information to focus on. Location is relevant to most mobile applications, but additional levels of context can be added. A story can be presented differently depending on the weather or the time of day. By creating a focus on historical events at a particular location, a user could be given the ability to "time travel". M-Studio lets authors define contexts and use them in the planning process, allowing them to concentrate on this information throughout story development and visualize its role in the story structure.

Another challenge mobile cinema presents involves designing an appropriate story structure. Since story events are delivered discretely and predicated on a viewer's navigation of space and time, many possible paths through a story can emerge. The spatial nature of mobile cinema invites the creation of multi-threaded stories, where many events are taking place at the same time but in different locations. In this complex world, the viewer may not have all the information about the story at a given time. However, the system may be designed such that the user can influence the outcome of the story by making choices about where to physically navigate to next. To handle this complexity, M-Studio provides a mechanism to define connections between discrete events and visualize the resulting story structure as a whole.

These multi-threaded story structures can quickly become quite complicated, and the author must be aware of all the different ways a story can play out as the viewer navigates through time and space. M-Studio will provide simulation tools that will allow the author to see how the story will unfold given a certain viewer path, and also generate all the possible plot lines that can unfold from the story's configuration.

Finally, M-Studio will handle the process of server communication during story distribution. Rather than forcing the author to learn a scripting language, M-Studio will create story scripts directly from the author's input into a graphical user interface. It will also package and upload media content directly to the server.

The next portion of this paper looks more closely at mobile cinema and present examples of different types of content that could be developed for the platform. I then discuss the evolution of the M-Views project and its need for an authoring tool. Section 4 examines related work developed from both commercial and academic arenas. Section 5 looks at the methodology used for developing M-Studio along with the major system goals. In section 6, I present a detailed description of the technical implementation of the tool and explain major design decisions. Section 7 gives the results of our preliminary evaluations of M-Studio. Finally, I discuss future areas for improvement and development of M-Studio, along with current conclusions about the project.

2. MOBILE CINEMA

In order to understand the purpose of M-Studio, one must first understand the nature of mobile cinema and some of the complications it presents to the creation of stories. How does mobile cinema differ from linear cinema? What features are needed in a pre-visualization and authoring tool in order to help authors construct mobile cinema?

One of the features of mobile cinema that distinguishes it most from other forms of storytelling is that scenes are delivered discretely depending on a viewer's position in space and time. This removes the guarantee that events will be delivered in a particular order (or that events will be delivered at all) that is implicit in traditional story forms. Conventional fiction stories are comprised of a certain sequence of locations, characters, and actions. Each character's actions follow a story arc: they are faced with a situation to which they respond. Their response changes the situation, which may cause a new predicament to arise, calling for the character to take another action, until we finally reach a conclusion or steady state. These changes in situation result in changes of the state of the character: worldly wealth, self-confidence, trust of friend, etc. Jerome Bruner writes in Acts of Meaning, "Perhaps its principal property is its inherent sequentiality: a narrative is composed of a unique sequence of events, mental states, happenings involving human beings [usually] as characters or actors. These are its

constituents. But these constituents do not, as it were, have a life or meaning of their own. Their meaning is given by their place in the overall configuration of the sequence as a whole, its plot or fabula." [1]

Mobile cinema represents a departure from this concept of story. Unlike traditional cinema, where the physical nature of celluloid imposes a linear sequence on scenes, in mobile cinema, individual scenes are delivered discretely, and the medium does not guarantee their sequentiality. Mobile cinema creators must confront this issue as they plan content for a platform like M-Views. Although there is one overarching story, there can be many different narrative paths through that story, each composed of discretely delivered story pieces. These pieces, or clips, must be designed to exist within potentially many different narrative paths. A tool like M-Studio can aid the mobile cinema creator by providing a way to visualize and simulate of these paths during the story design process.

To better illustrate the concepts of mobile cinema, I will present several examples of possible content types for the M-Views platform. One obvious application for mobile cinema is a game-like story, reminiscent of role-playing games. Viewers would take on the role of a character in the game and try to accomplish a goal, like finding a missing object or catching a bad guy. In this type of story, users would be very active, and their actions would influence the plot of the story a great deal, for both themselves and other participants in the game, perhaps. On the other end of the narrative spectrum, you can imagine a daily soap opera delivered to viewers as they move throughout the day. Although viewers would not actively participate, what they got out of the story would be influenced by where their daily travels take them; perhaps compelling them to use the M-Views messaging tools to share what they have learned with other viewers.

Mobile cinema does not need to be narrative. One could use the context-aware features of the platform to deliver very customized stand-alone scenes to users. M-Views technology could be used to virtually embed a piece of historical video in a building. The video could be made to seem more relevant to the user by selecting this piece of video based on what time of day or what season it was. This idea could be also be used, albeit less enjoyably, to deliver customized advertising. For instance, a pedestrian a block away from a Starbucks on a cold day could be invited to drop in for a cup of hot cocoa. On a hot day, they would be impelled to come and enjoy an iced coffee.

The M-Views platform could also be useful as a tour guide. A person new to a place could be presented with information about a place as they walk by it. The system could also include suggestions about where the visitor might want to go next. As the tourist makes more choices, the system might even be able to better customize its recommendations.

In addition to different types of content, different styles of productions can be used to create mobile cinema. Clearly, a traditional production, with a single director and a centralized writing process can easily be applied to mobile cinema. In this style, all the scenes in the story would be written and directed by the same person or persons, just like they would be for traditional, non-interactive cinema. This type of production would likely arise when one person has an idea for a story, and brings others together to implement this vision. One advantage of this approach is that having the same person responsible for all scenes makes it easier to ensure overall continuity. However, the multi-threaded nature of mobile cinema also suggests a more distributed production form. Rather than having all scenes in a story designed by the same people, with very specific notes, a team could decide on central story principles and then work separately to develop individual threads. This concept could even be taken so far as to have several production teams working on location simultaneously to produce different parts of a story. This sort of production would likely be used when several people had stories to tell that all fell under a unifying concept. The idea works well with some of the central goals of mobile cinema, allowing a production to provide highly customized content. However, in this case there needs to be a way for these separate teams to unite their work and ensure that each story element is effective within many encompassing narrative paths.

These different types of content and productions could be executed using M-Studio. To explain the features of M-Studio, I will generally rely on examples from the two productions it has been used in creating. The first, "Another Alice"[2], is a murder mystery with a simple story structure. In the beginning, viewers receive an email telling them to meet a professor in her office. When they arrive, they meet one of her assistants, who informs the viewer the professor is dead. It is the viewer's job to find out what happened by following the characters. In each clip, the character tells the user where they are going next, compelling the user to follow them. Characters often run into each other, giving the user a choice of who to follow. The user has to decide whom to trust, hunt down clues, and chase after characters to catch the killer. The final outcome of the story depends on what path the user takes.

The production for "Another Alice" was quite centralized. One author designed the general story idea and all the characters. She also determined most of the major plot points in each character's storyline. There was some collaboration with other production members, who had ideas for certain scenes, but almost all of the writing, both of overall story and specific dialogue, was done by one person. Each scene used the same cameraperson and the same editor.

The second production, "Mind in Hand", is less demanding than "Another Alice". Clips are designed to come to users as they move through campus, rather than having the user track down clips. The story is designed to lend a dramatic flavor to the life of several MIT students as they journey through the day. The production mixes in documentary scenes to convey the spirit of the campus and the Institute. Someone visiting campus for the first time would not only get information and history about the places they visit, but would also learn about the motivations of some "typical" MIT students. Some of the major storylines include two people being set up on a blind date by a friend, a student who is contemplating dropping out to pursue a music career, and a girl who receives a mysterious email from another student, saying that he had found her father's class ring. Although the path the user takes affects what characters they learn about, the story itself is not changed by user choices.

The production for "Mind in Hand" was highly collaborative, with several people working together to develop story threads and weave them together. Characters were initially developed completely independently by separate people and brought together in a group decision after seeing how these characters could be related. Once the basic characters and major plotlines were set, several members of the writing team fleshed each character's day out into a set of scenes. A few story team members then wrote dialogue for each of the selected scenes. So, there are scenes with different final authors. A single filming team shot each scene, to create a consistent look across the piece. However, different people edited the footage into individual scenes.

3. THE M-VIEWS PROJECT

The M-Views project is a platform for exploring the potential of mobile cinema from the perspective of both creators and consumers. It was originally conceived of by Pengkai Pan and the Interactive Cinema Group at the MIT Media Laboratory. Its research has encompassed the

development of the hardware and software that comprise the platform, as well as the aforementioned story productions.

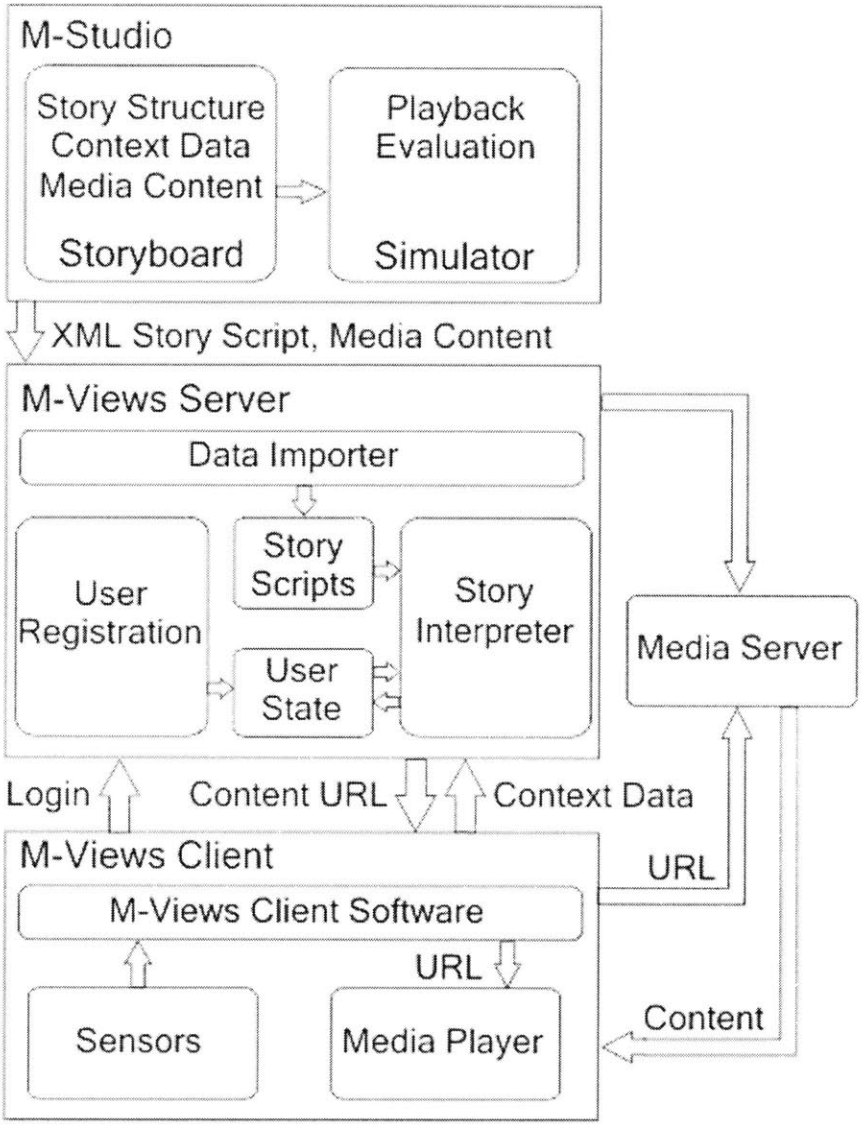

Figure 1: M-Views System Architecture

The design for the M-Views delivery system consists of a location-aware handheld computer with a wireless network connection that communicates with a central server and receives video clips streamed over the wireless network. Both the client and the server have gone through a few revisions. The current client consists of a Compaq iPAQ with an 802.11b wireless network card and a positioning system. Several different location detection schemes have been tried, including GPS, infrared beacons, and 802.11b triangulation. Each scheme has its advantages and disadvantages, which are discussed in Section 6.1.1. The iPAQ could be

outfitted with further sensors to provide more context information. The client software is designed to emulate an email program, where story events are received as messages in the user's inbox. The client also works as an instant messaging tool, allowing multiple users to communicate by sending text messages or forwarding video events.

The client communicates with a central server by sending it its current position and context information every few seconds. The server returns a story event if there is one under those conditions. The server has gone through several iterations that have increased its flexibility in presenting stories. In initial designs, it was only able to look up events based on location and time conditions. The new version, developed along with the messaging client by David Crow, maintains a user state, allowing for more control over narrative structures. The server also provides web-based story administration capabilities.

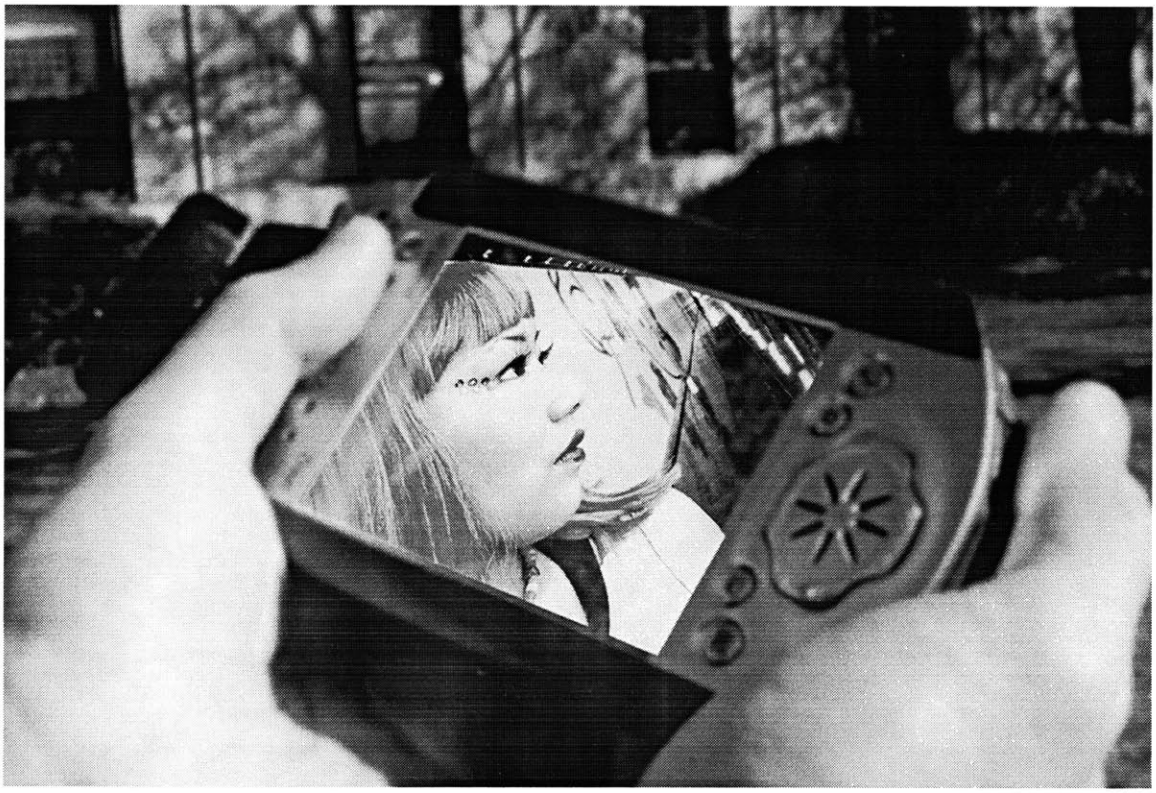

Figure 2: The M-Views Client

M-Studio was developed fairly early in the M-Views project, although its role changed quite dramatically. Most initial concepts for M-Views productions focused on the idea of customized individual scenes rather than structured narratives. So, my initial project, done as a UROP, was a tool that allowed the easy annotation of video clips with location and time information and provided a map to see where scenes in a story were located. The annotation

information could then be exported to the early server along with the video clips that comprised the story. As the story productions teams began working, however, we found a desire to do more narrative productions, like "Another Alice". To allow this, the server was adjusted so clips could be placed into parallel storylines and looked up based on what track the author was following. To cope with this change, the annotation tool had to be changed to allow the author to place these scenes into parallel storylines as well as annotate them with context. It soon became clear that for full narratives, a tool like this would be necessary to allow authors to structure their stories and examine story threads through simulation. As the project progressed, new production forms were suggested, and all the tools of the M-Views platform were updated to accommodate more creative desires and provide a more flexible platform for authoring and presenting content.

4. RELATED WORK

There are many examples of desktop multimedia authoring tools. In addition to providing sequencing utilities, these tools also provide controls for user interactions and scripting languages. For example, Macromedia Director[3] allows video and graphical elements to be arranged with score or a timeline. It also offers keywords to allow interactive control of elements. It provides a visual interface for editing multimedia presentations, allowing authors to inspect and edit the properties and behaviors of multiple content objects. Director's Lingo scripting language allows for a greater of control over interactive elements and supplies data tracking between objects for advanced users. Another Macromedia tool, Authorware[4], offers interactive controls, including conditional statements and the tracking of data between multimedia elements. However, these tools are targeted for desktop multimedia applications. They only handle a limited set of predefined user interactions, like clicking a button. In order to fully harness the capabilities of mobile storytelling, one must be able account for indirect interactions, such as changes in location or other context. Furthermore, these tools do not allow for the development of parallel storylines.

Authorware and Director both use scripting languages to provide fine control and customizability to their productions. Scripting languages can be time consuming to learn and work with, which places a burden on authors which may make them not want to invest time in using these tools in early production stages, when there might be major revisions.

There has been research into developing several rapid prototyping tools that facilitate the preproduction process. DEMAIS[5] provides a sketch-based storyboard, which allows users to turn rough sketches into a story prototype. It also supports integration with multimedia and interactive controls. This tool allows users to experiment with layout and behavioral aspects of their production. MediaDesc[6] is a similar tool that uses a timeline with different views, allowing a user to visualize and test early design ideas. M-Studio offers similar initial design capabilities. Story design is an iterative process, requiring many revisions and changes to story structure. M-Studio should let users to prototype stories easily, allowing them to quickly change story structure. However, these other prototyping tools are intended for the design of flexible multimedia presentations. M-Studio is used to develop stories for the mobile M-Views platform, which imposes certain constraints on the way content will be viewed. So, while M-Studio must be a flexible prototyping tool like DEMAIS or MediaDesc, it must also make the limitations and requirements of the mobile platform clear to the author, who can then incorporate these ideas into the story design.

To understand how to provide structure to multi-threaded stories, we can look to other research in the area of interactive narrative. One such project, Kevin Brooks' Agent Stories [7], provides "a story design and presentation environment for non-linear, multiple-point-of-view cinematic stories." The tool is intended to aid the writer of non-linear story in structuring and revising narratives before they are produced as video. In order to provide story structure, Agent Stories allows the author to provide structural links between clips. Links are used to define relationships between story elements. For instance, clips can be said to precede or follow each other or have to be seen along with another clip. Less explicit connections can also be made by specifying which clips tend to support or oppose ideas in other clips. This structure of clips and links define a story web that can be navigated by traveling its links as narrative paths. M-Studio uses a similar technique to provide structure to a non-linear narrative using flags. Like story links, flags are used to express connections between clips that can be understood by both M-Studio simulation tools and the M-Views server. M-Studio can travel these flag links to find possible narrative paths through the story.

Another example of a tool that allows authors to annotate a database of clips and then play out resulting paths is the LogBoy/FilterGirl toolkit [8]. This project also targets cinematic experiences that vary depending on viewing context. The LogBoy tool allows authors to create

annotations for clips that will be used to determine playback of the story. FilterGirl allows the author to specify and test a set of playback constraints that will be applied to the annotated video database. M-Studio provides similar tools, targeted at the mobile platform. The flag editor allows the user to provide structural annotations, while the story simulator allows authors to create, test, and save different sets of contextual constraints to simulate against.

The Viper project [9] is a tool for creating responsive video programs. The tool allows a user to create relationships between clips in a database that express their structure. It also uses heuristic measures to express what a particular clip represents or other content information like how it was shot. This tool is used to provide some automatic editing capabilities for the database of video. Clips are selected from the database by specifying properties and constraints, based on the annotations made for the previous stage. The system can also create templates to determine these viewing parameters. Again, M-Studio will provide similar capabilities for annotation and simulation.

While these tools provide some simulation capabilities, M-Studio must allow users to simulate in a mobile context. Another context-aware content delivery system, GeoNotes [10], provides a similar simulation feature. GeoNotes allows users to annotate real positions in space with virtual Post-Its. In order to demonstrate and test its functionality, QuakeSim was developed. QuakeSim is based on Quake III Arena, and allows the user to simulate walking around an area, posting notes and discovering those left by others. M-Studio uses a similar idea for its simulation interface. A simulator sends context information to the system, which returns clips that the simulator can play back for the user. The M-Studio simulators will have to accommodate different kinds of context information and keep track of other possible user interactions with the story.

5. GOALS AND METHODOLOGY

From the beginning of the M-Views project, one of our goals has been to make our platform widely available to both viewers and authors. Our hope is that by making content creation accessible to a large audience, we will encourage the growth of mobile cinema as new type of cinematic experience. Many novel systems for presenting content have been developed. For instance, there are context-aware virtual tour guide systems such as CyberGuide[11] that provide customized, interactive tours of cities and landmarks. Role-playing games and other

video games can present stories in an interactive way. However, these systems have not focused on allowing independent users to create content for the platform. The story creators usually work with people who have knowledge of the backend of the system. So, only people associated with the platform developers can produce content.

There are arguments for this sort of proprietary system. If content developers work directly with the platform developers, they can create highly customized content and ensure its quality and appropriateness for the delivery system. However, one of the goals of the M-Views project was to create a platform that could be used by people who have no knowledge of the implementation of the backend. Therefore, it was necessary to develop tools that would allow anyone to author for the platform. However, we did not just want to create a publicly available scripting language, since there is usually a learning curve associated with them, and some experience with programming is often necessary, especially when creating complex stories. Therefore, it was necessary to creating an authoring tool with a simple-to-use graphical interface to facilitate in story creation.

The main component of the story structuring system is the flag model. The flag system allows authors to specify as much or as little ordering or structuring onto a story as they need. The model is simple. Each clip has a set of flags that must be true in order for the clip to play. These flags are evaluated by looking at a table of global values that are maintained during the playback of the story. After a clip is viewed, the flag table is updated as specified by the clip. This system allows authors to make seeing one clip contingent on having seen another, or on having met a particular character or visited a certain location. This flags are also used to implement more extensive context information, like time of day, or weather.

The mobile nature of our platform also begged for simulation tools that would allow the author to visualize play out of the story based on projected movement of the viewer through space and time. Even if a story creator was comfortable using a lower level scripting language to create a story structure, without authoring tools, there would be no way to test this without actually running the story on the server and experimenting with it using a client iPAQ. Given that location based stories require users to physically move through space, this becomes very impractical. The use of more extensive context information, like time of day or weather, makes a full simulation even more inconvenient. Furthermore, this method of evaluation gives the

author no convenient way to record the paths he or she has taken, making it difficult to keep track of story problems that may exist in the script.

From the beginning, we envisioned an authoring tool that could not only to write the low level code, but could also allow authors to understand the salient features of content designed for the mobile platform. The authoring tool can reinforce the notion that story events are always tied to their locations. Furthermore, it can make it easy to understand that by giving the story spatial as well as temporal dimensions, it is possible for many things to be unfolding at once in the story at different places. Finally, the authoring tool can support the role of the active viewer in stories. End users can make decisions that change how the story unfolds. These aspects are quite different from linear, passively experience narratives, and thus require the author to approach the story creation process from a new perspective.

By simplifying the creation process and making these tools widely available, we hope to encourage independent content creation. We believe this is necessary for widespread adoption of the platform. When dealing with a new form, like non-linear mobile narratives, it is important to have a supply of compelling content that is frequently updated, which is difficult to achieve this if all content is being created 'in house', especially considering the time-consuming nature of video production. By allowing any interested party to create content, we not only achieve the goal of having more stories available, but the stories can become more customized. Although it is possible to create a location-based story that could be adapted to work in many different places, using generic locations, most of the content will likely be localized to a particular place, like a city or campus. So, rather than having one centralized team struggle to creation a production that is generalized enough for it to play well in many different venues, it would be better to rely on local production groups, who could make their content more appealing by tailoring to the special aspects of their area. By supporting these independent productions, we can increase the supply of engaging content, and thus increase interest in and use of the M-Views platform.

The development of M-Studio was closely tied with both the development of the M-Views client and server, and the M-Views story productions. As story teams developed new concepts, the software developers had to work together to find new means to accommodate different story forms and features. The two teams worked together to focus on how to support content creators from the authoring side and mobile viewers in the content delivery side. These

new ideas were then tested with existing and emerging story content to validate and refine our designs.

6. SYSTEM IMPLEMENTATION

M-Studio seeks to support authors throughout the mobile cinema design and production process. To do this, it provides several utilities that work together, as shown in Figure 3.

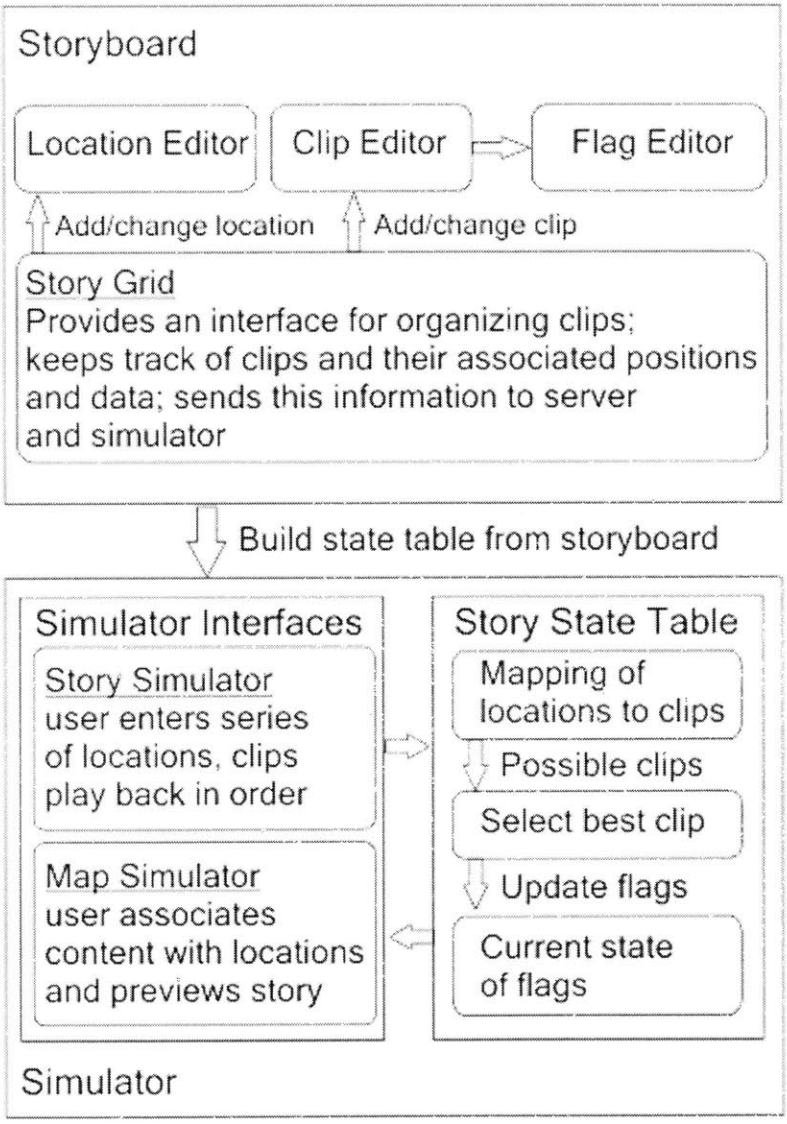

Figure 3: M-Studio Architecture

It allows authors to layout multi-threaded stories that unfold in time and space using a storyboard where the user can enter and annotate clips with locations. M-Studio also provides authors with a flag-based system for developing story structures by defining relationships

between clips. It also offers simulation capabilities that allow authors to see how a story would unfold given a particular physical user path. An author can also generate all the possible paths through a story. Finally, the tool helps to abstract away details of server implementation by automatically generating XML story scripts and exporting them to the server along with media content. M-Studio was implemented entirely using Java and Swing, primarily for ease of development and portability across platforms.

6.1 Storyboard

In the initial design for the M-Views server, each path through the story was represented as a separate storyline. Events in the storylines were associated with locations laid out in absolute time, with each clip playing only in a given stretch of time after the viewer had started the story. When there were multiple clips available at a given time and location, the clip would be selected based on what storyline the user had been following. The only opportunities to switch paths occurred when two characters were in the same place at the same time. No additional data beyond location, time, and storyline was used. Thus, the storyboard was developed with this structure in mind, providing an easy way to layout parallel storylines and associate clips with locations.

The storyboard is the central component to the authoring system, providing data to both the simulators and the server. It is the space where the author builds his or her story. It keeps track of the different locations where clips can take place, the possible storylines, and the positions of all the clips.

To create a storyboard, the user first specifies the length of time of the story, and a list of storylines (Both of these attributes can be later modified). With this information, a new storyboard is built. The storyboard is displayed as a grid of cells, with each cell representing a particular time in a storyline. Each of these cells can be filled with a clip. In addition to displaying the clips, the grid keeps track of the relationship between clips as they are entered. This can be useful in structuring the story.

Parallel storylines are a useful abstraction for a multi-threaded story. Multi-threaded stories allow authors to create stories where the viewer does not have all the information in the story, introducing the element of user choice. The goal of this system is to allow an author to visualize all story interactions in a snapshot and as they would play out. Storylines can be used

in different ways depending on the author's goals. They can represent completely independent, self-sufficient plot lines, or pieces of a story that can come together or diverge.

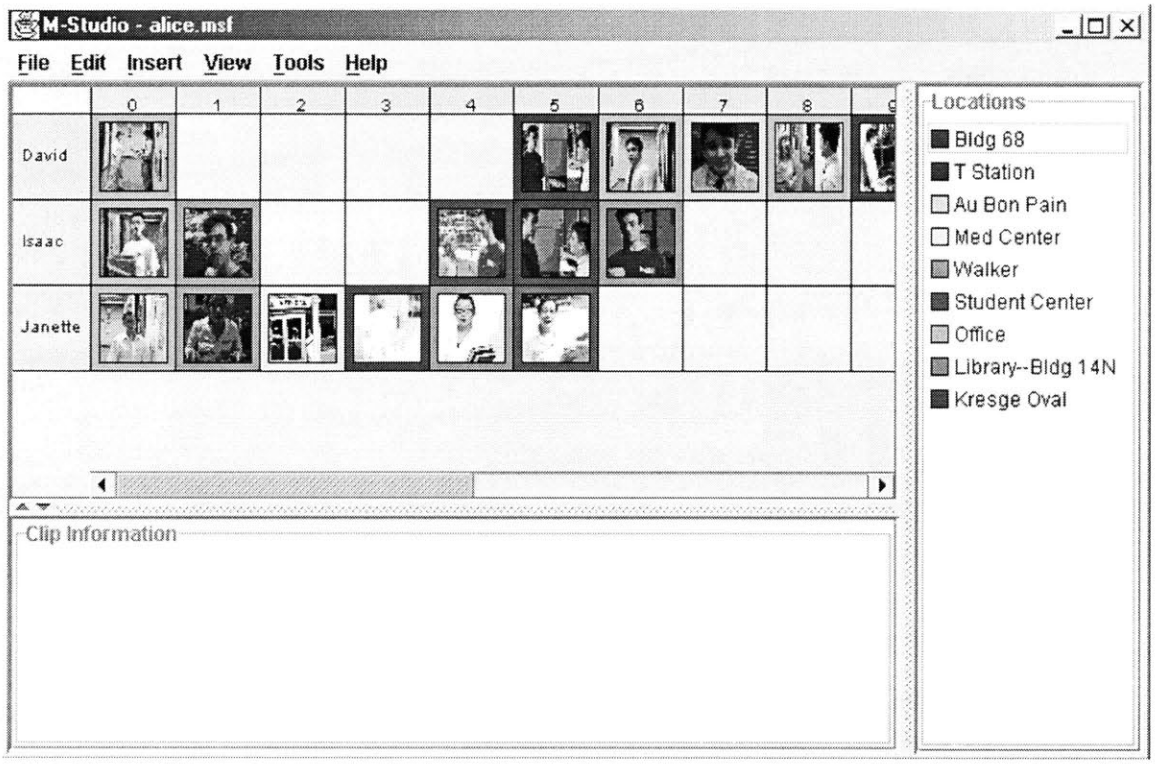

Figure 4: The Storyboard for "Another Alice"

The storyboard layout, as shown above, seeks to present the maximum amount of information about clips and the overall story structure. Images and short titles provide a convenient reminder about the content of clips. This technique of annotated thumbnails is used in many video-sequencing utilities. However, M-Studio must also put forth context information. We chose to use color as a compact way to represent the location associated with each clip. Location is the most crucial piece of context to view, because it influences story structure. By seeing every clip in the context of its location, it is easy to see places where different storylines can intersect and imagine the physical path each character might take. The user can easily refer to the color-coded location list to the right of the storyboard to recall which location is which, and can view the location on a map by selecting it from the list. The user can also view more detailed information about each clip, including any notes made about it by simply clicking on a clip.

6.1.1 Location Editor

Once the storyboard has been set up, the user can begin by entering locations that will represent the settings for clips with the location editor. This tool provides a convenient way for users to define and visualize the locations where their clips take place. Each location is assigned a name and a unique ID, used by the storyboard for tracking purposes. Each also has a visual representation, consisting of a position on a map and a color. The use of colors for locations allows for a simple visual representation. Authors can easily see what color is associated with each clip, and then refer to the location list to determine the name of the location. In the location editor, shown in Figure 5, they can associate the name on the list with a physical location on the map. Locations are extensible, so they can be used to accommodate new location detection schemes where the context information that is passed to the server is changed.

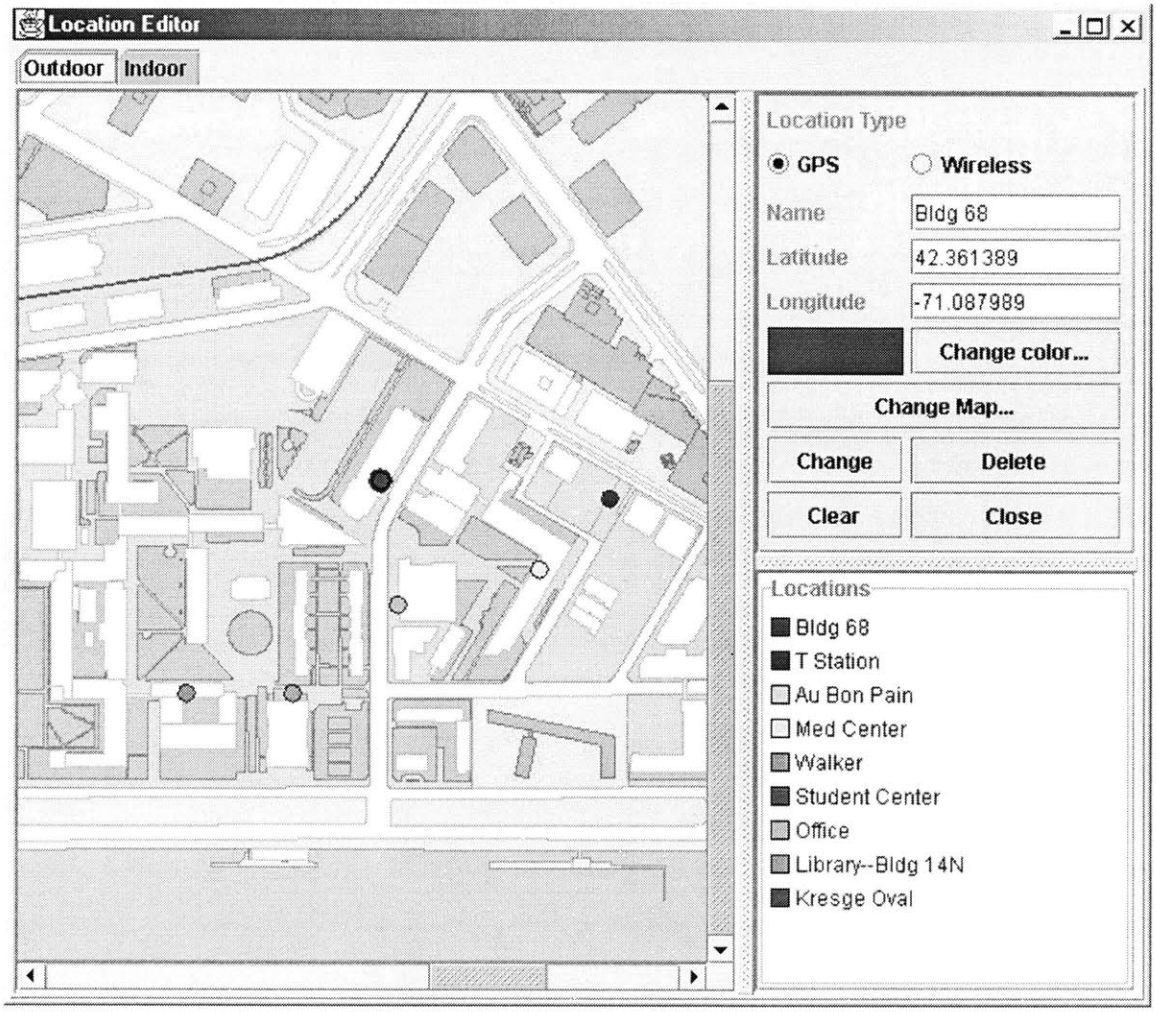

Figure 5: The Location Editor

M-Studio currently supports three different location detection schemes. The first scheme, which was used for "Another Alice", is the Global Positioning System. So, we created GPS locations that used latitude and longitude coordinates for context information. Since GPS does not work indoors, we created infrared locations that were read by scanning infrared beacons. For these locations, the identification string of the beacon is the context information. The problem with infrared beacons is that they are difficult to set up in uncontrolled and outdoor locations. However, 802.11b wireless networks were becoming more pervasive, providing complete coverage to areas like the MIT campus. So, the server was able to triangulate position based on signal strengths from wireless access points, allowing for a consistent location detection framework that did not rely on extra hardware on the client. This implementation uses maps that are coordinated with the server, so a location's map position works as its context information. The server then matches that position to a predetermined set of signal strength data.

The maps used in the location editor provide a visual frame of reference for the author, letting them remember the exact position for a location. The map can also provide other useful information, like what other locations are nearby, or the approximate distance from one location to another. Authors can input new map images using the map editor to represent new areas where location hotspots can be placed. A story can unfold over many different maps. There are both indoor and outdoor maps. The only difference is that indoor maps allow the author to consolidate the different floors for a building into one map, adding an extra dimension to the representation.

6.1.2 Clips

After some locations have been specified, the author can start entering clips. Clips are abstractions that encapsulate all the data known about a clip. Clips know the files associated with them and their metadata, in this case time, storyline, and location. It is necessary to track this information with the clip to make saving and loading storyboards more convenient. This information also allows for users to easily move clips around the board. Users can also provide thumbnail images for their clips, making them easy to identify on the storyboard.

There are two different types of clips, one that represents a standard video clip, and one that represents other forms of clips, such as images, text, or sound. These other kinds of clips can be useful in the process of writing the initial story. An author could include a textual

description of the scene, a photo of the location where it would be shot, or an audio clip of possible dialogue or music for the scene. These pieces can be combined as the user sees fit. If the user wants to provide something more complicated (like a series of shots staging a scene), the user can insert a web page. Video clips can replace these clips when shooting and editing has been completed. Alternatively, they could be left in and used in the final story, depending on what file formats the target server accepts.

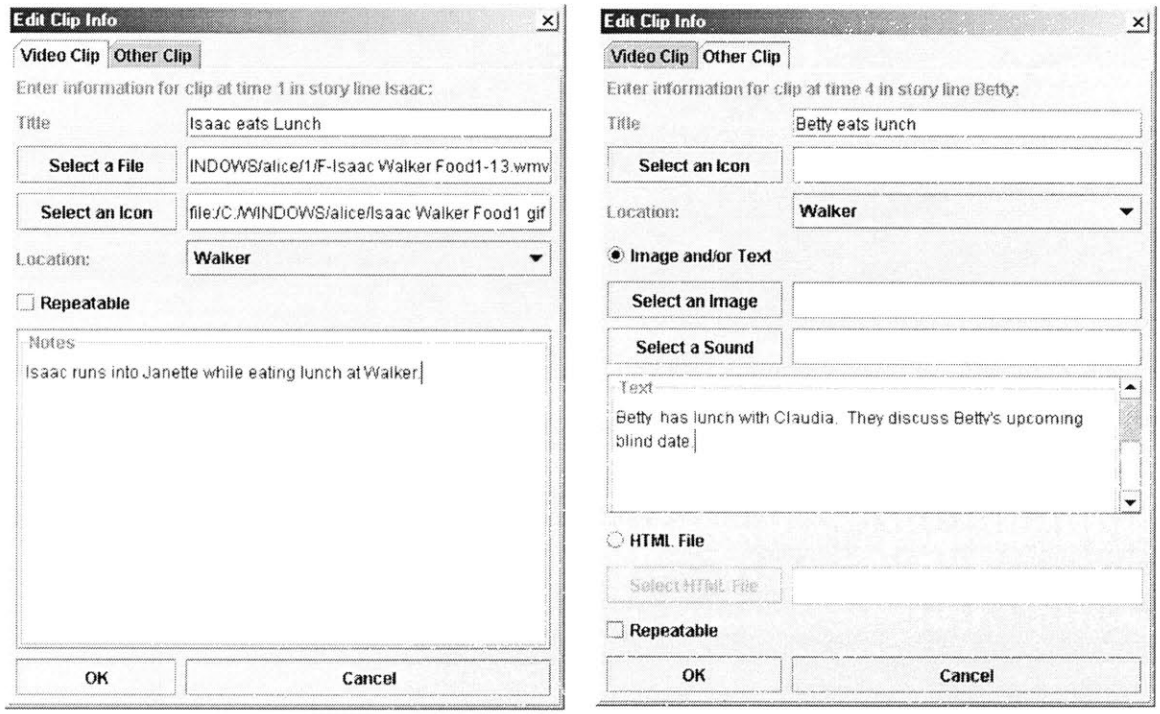

Figure 6: The Video and Text Clip Editors

6.1.3 Location View

After the preliminary version of M-Studio was developed, a new version of the server was created. This new server sought to improve the flexibility of the story structure. Location was still the most important piece of information in deciding which clip to play, we wanted to allow users to create more complex and interesting story structures where the outcome could be influenced by what path the viewer chose to take. Whereas previously the storylines were very strict, and crossover could only take place when two clips occurred at the same place and time, the new server would allow for greater possibilities. We also wanted to be able integrate the new server smoothly with M-Studio. So, we developed a flag-based system for stories. Like before,

each clip would be associated with a location. However, now instead of being placed into a strict storyline and time slot, it would be associated with a set of flags. The flags would represent a set of global conditions maintained by the server. Each clip would require certain flags to be true in order to play. After the clip was played, it would update the global flags, thus influencing what other clips could be played, allowing authors to develop a more complex story structure. We also decided on an XML schema to simplify the task of communication between the authoring tools and the server.

Clearly, M-Studio needed to be altered to accommodate these new features. In addition to providing a way for users to create flags and associate them with clips, there needed to be different ways to view and layout clips that allowed the author to analyze more complicated or less defined structures. To decide what other tools would be necessary to support these new story structures, I observed the story meetings for the development of the new story.

At the first story meeting, the authors began by planning characters. Each writer separately developed a character that would be on the MIT campus for a day. Some examples were students, professors, staff members, prospective students and parents, and tourists. The characters were given short biographies and a central conflict in their lives. Then, the authors decided what their character would be doing at three different times of the day, morning, afternoon, and evening. Each participant brought his or her character to the first story meeting.

The goal was of this meeting was to produce two short scenes at two different locations. We began by comparing the characters everyone had created. We started to build a list of possible locations for scenes based on the characters' days and other locations we wanted to include because they had historical or architectural significance. Immediately, there was interest expressed in having one big chart that could encapsulate all the information about characters, locations, and time. There was some debate over how to pick the two scenes to write first. Picking two locations first might cause us to steer characters to places they would not go. At the same time, picking characters to start with might cause us to force some unnatural intersection between stories. So, we started out by plotting each character's day separately, to decide where a good starting point would be. The authors made a grid with time slots and decided where each character would be at any given time on a normal day. Using this chart, the authors found similarities in these schedules and were able to find very logical places that certain characters would meet.

This first story meeting made it clear that the original storyboard design was on the right track, particularly for a story like this one. Since this story focused on different lives unfolding at the same time, parallel story threads were a good starting point for development. This view allows an author to develop the stories first, and then see what locations emerge as the key points in a story.

At the next story meeting, the authors had condensed their character information into a new chart, which had been divided into three major time slots, morning, afternoon, and evening, and was then sorted by location. Although this view made it harder to follow what path any one character took, it is easier to see where character's lives intersected, facilitating the writing process. Also, as the story moved into production, it was also useful to be able to organize scenes to be shot on the basis of location, so scenes at the same location could be grouped together. Since this production was also intended to play across the span of a day, the location-sorted chart was also useful in determining story density. Once the list of story locations was finalized, it was easy to decide where to place certain scenes in order to produce the fullest possible coverage of campus.

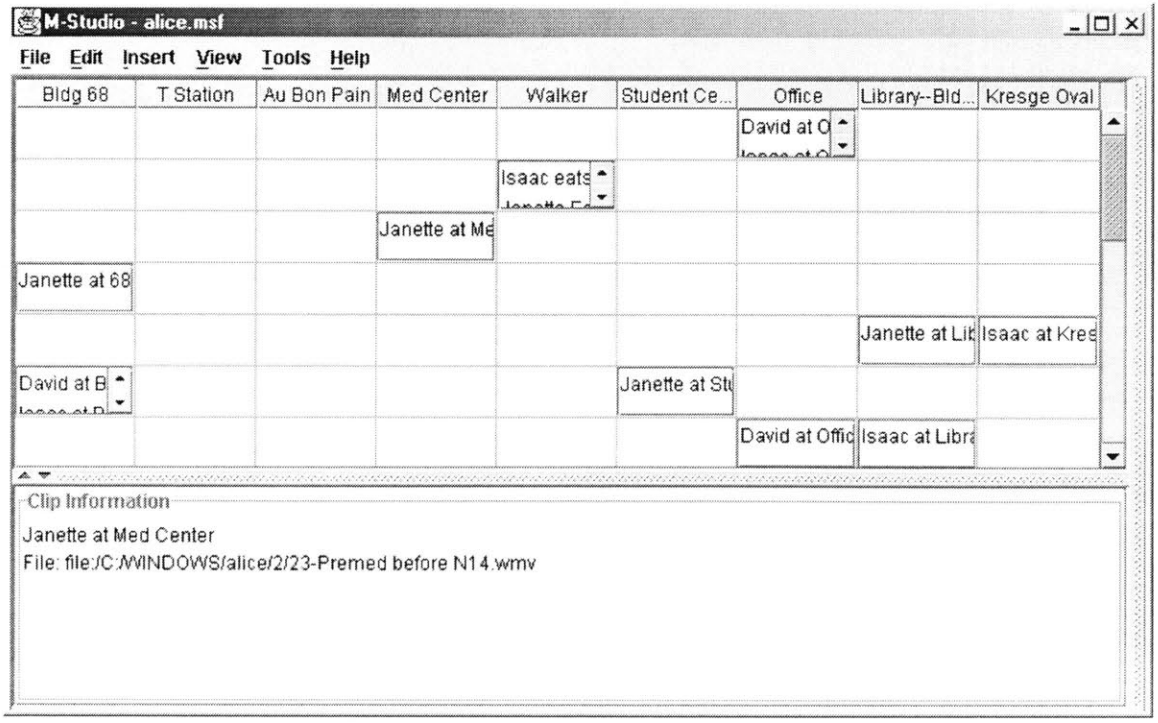

Figure 7: The Location View of "Another Alice"

Since both the storyline and location views had clear benefits, I decided to use them both in M-Studio. The original storyline structure was maintained, and a new location table was added. Both views were synchronized. When adding a clip to the location table, the user was prompted to select what storyline the clip belonged in so it could be placed in the story view. Both views still had the same clip sequencing and editing utilities. The display for the location table was kept simple, with a timeline associated with each location, illustrating what happened at that place throughout the day. Each cell in the table could contain a list of clips, representing that more than one thing could be happening in the same place at the same time. For simplicity, clips in the location table could only be displayed by their titles rather than with thumbnails.

As writing made way for shooting and production, it also became important to have easily distributable copies of the story charts and production notes. So, I added tools that could quickly export the storyboard to HTML in several different formats. One option is to generate an HTML version of the storyboard, sorted either in location or storyline views, using either thumbnail images or text to represent the clips. Each clip is linked to either its associated video, or an automatically generated web page containing all of its information, depending on the clip type. All of this material could then be copied and placed on a website in a central location so all team members could refer to it. Alternatively, the authors could produce a full script containing all of the text and images that had been entered for each scene. The scenes could be sorted by storyline or by location, whichever is more convenient for the authors. These scripts could be generated and printed for use on location during shooting.

6.2 Flags

One important thing to note about the location and story views is that, under the new server structure, they do not necessarily represent anything concrete about the story structure. If an entire storyboard was created with no flags and exported, the server would have no information how to structure the story beyond the location of each clip. The user must create the necessary flags to provide the desired story structure. To help alleviate that burden, M-Studio can automatically generate flags based on the relative positions of the clips in the storyboard. To understand the process of generating flags, we must first understand what flags are and how they are used in the story creation process.

Flags are a way of maintaining the state of a story, which allows for the creation of structure. Under the previous server model, if a person walked to the right place at the right time, it was as if a clip was waiting there for them. With the flag model, the story experience does not just depend on a user's position, but rather the whole of how the user has interacted with the story. For instance, a user's decision to follow one character instead of another could not just cause them to miss one clip, but could change the course of the narrative altogether. So, rather than a user's story experience just being a function of context, it is also a function of past actions.

There are many different ways to provide structure to a narrative. The common form for cinema is linear narrative, where events unfold in a predetermined sequence. Although this model could be useful for some mobile applications, like a guided tour, it fails to fully exploit the potential of the platform. A better model could be represented by the old server implementation, where events are embedded in space and time, and the user has to navigate through them. However, this does not provide the ability to actually change the narrative based on the user's behavior. A programmatic model would be able to do this. This would be a model that could be represented by the game engine in a role-playing or other narrative-based video game. Although this provides almost limitless flexibility, it would almost definitely require the use of a scripting or programming language, which would create a barrier of entry for story developers. Also, once the story was created, it would be difficult for others not involved in development to modify it. Flags provide a balance between the flexibility and simplicity needed for story creators. Since flags allow events to be dependent on past user state, they allow for the possibility of creating changing storylines that unfold based on the user's actions. Flag structures can be as simple or complex as the author desires. It would be very simple to use flags to emulate simpler narrative models, such as linear stories or independent clips only associated with time and space. With customization, flags can even approach the story control provided by a scripting language. However, they are still easy to use. In M-Studio, they are created and manipulated through simple graphical user interfaces. Flags can even be automatically generated based on the structure laid out in the storyboard, using simple rules.

The structure of flags is very simple. Each clip has two sets of flags associated with it: a requirement clause and results clause. Each clause contains a mapping from flags to their values. A flag can be thought of as a global variable, a piece of information about the story state. The

value of the flag can change as the story continues. M-Studio tracks this information for use during simulation by keeping an internal flag structure. Flags for a story are kept in a central table, mapped to their current values. The flags in the requirement clause all must evaluate to true in order for the clip to play. To evaluate a flag for a clip, the current value of the flag is looked up in the table. That value is compared to the required value for the flag according to rules specified by the flag value. If the condition is met, the flag evaluates to true, and the next flag is checked. If all of a clip's flags are true, then the clip will play. If the clip does play, the flags in the results clause are evaluated, updating the current values in the flag table according to the specified function. In the M-Views system, location is used as top-level constraint. So, when the user arrives at a particular location, the server makes a list of clips associated with that location, and then evaluates which ones could play based on their flags.

6.2.1 Flag Types

M-Studio provides several different flag types for authors to use to customize their stories. The most basic type of flag is the *global flag*. A global flag consists of a single integer value that is evaluated and updated as clips play. When used in the requirement clause, a global flag is given a comparative operator and an integer value. When it is evaluated, the current value for the flag is fetched from the flag table and compared against the given value using the given comparator. In a simple case, a global flag requirement might state that a clip can play if the current value of `foo` is greater than five. In the results clause, the global flag is updated by a value using an operator. For instance, after a clip plays, the result might be that the value of `foo` is incremented by two. Global flags are extremely flexible and can be used to implement almost anything, if used cleverly enough. One obvious use would be using these flags to keep track of health in a game. Every time users see a clip where they are injured, their health is decremented. All of the clips in the game require health to be greater than zero in order to play. If the value of the health flag drops below zero, a game-ending clip might play. That clip would require that health be less than zero in order to play. Global flags can also provide an easy approximation for Boolean values by using the equality and setting operators. Seeing a clip could set the value to one or zero, depending on the circumstances. Clips requiring the flag could then test whether the value was true or false using the equality operator.

A logical extension of the global flag is the *multi-valued flag*. The multi-valued flag consists of a list of values. So, clips can require that a certain value be a member or not be a member of the list. When a clip is seen, items can be added to or removed from the list associated with a flag. The requirements can also be multiple valued, requiring that at least one item in the requirement list also be a member of the current flag value list. A good example for this would be keeping track of characters a user had met. This technique was used in a version of "Another Alice". Viewers meet one of three characters based on what time the story begins. After that, viewers can only see clips with characters they have already met. However, sometimes the character they are watching meets another character in the story. That character is added to the list of characters met, and the viewer can start seeing clips starring the second character. The flag structure to represent this is very simple. Each clip in a character's storyline has the characters met flag placed in its results clause, with the action to add the character's name to the list when the clip is played. For every clip in a storyline after the introductory clip, the characters met flag is also placed in the requirement clause with the character's name, stating that the given character must be in the current list of characters met in order for the clip to play. For clips with more than one character, both characters are placed in the requirement clause, stating that if either character has been met, the clip can play. Both characters are placed in the results clause, too, since after having seen the clip, the viewer will have met both characters.

When structuring a story, one of the most important things to know is what clips the user has already seen. To provide this facility, we use a special structure to keep track of the clips that have been seen. Clips are listed in the order they were seen, and mapped to the time they were seen by the user. Then, the author can refer to this data to impose a structure on the clips. These *clip flags* are an extension of the multi-valued flags, allowing the author to require a set of clips to have been seen or not seen in order for another clip to play. This is clearly important to maintaining narrative structure. For instance, suppose that a key plot point is introduced in one clip. Other clips that refer to this point will not make sense unless the viewer has seen the first clip. Therefore, all clips that refer to that issue can be made to require that the first clip has been seen. Additionally, the author can require that another clip be the last clip seen by the user in order for a certain clip to play. This technique can be used to enforce linearity between clips by simply creating a flag that requires that the previous clip be the logical precedent in for each clip.

This creates a structure where there are only specified linear stories. The viewer will have to find the entire story.

Clip flags cannot be set. Although this would have been possible to implement, we decided to maintain the abstraction that the clip structure could not be modified, mostly due to security concerns for the server. Instead, authors can use global flags to create desired effects. For instance, to achieve the effect of having multiple clips be counted as seen when one clip plays, an author would use a global flag that is set to true any time one of that group of clips plays. All the clips in the group would be predicated on that global flag's value being false, meaning that none of the clips in the group had been seen yet.

There are also other types of flags that cannot have their values set. Instead, the values must be supplied from another source. These flags represent factors the users cannot influence by their actions, such as the passage of time or external conditions, like weather.

One type of flag that relies on external information is the *relative time flag*. This allows the author to set timing between clips. An author would specify a trigger clip and the minimum and maximum amount of that can pass and associated this flag with a clip that would be triggered. When checking to see if the clip can play, the system consults the clip table to find out what time the trigger clip played and compares this with the current time to see how much time has passed. If the amount of time passed falls within the specified range, the flag evaluates to true. If the clip has not played yet, the flag is assumed to be true. (The author can add a clip flag to force the trigger clip to have played). One example of the usage of this flag would be the ending to "Another Alice". At the end, the viewer catches the killer as she is attempting to flee. To simulate the effect of a chase, two possible ending clips were created, one where the killer was caught, and one where she gets away on the train. The viewer five minutes to catch up to the killer. To achieve this, each clip was created with a timing flag that triggered off the clip where the killer started to escape. The clip where the killer was captured was given a time requirement of less than five minutes, while the clip where the killer got away was given a requirement of more than five minutes, to prevent the clips from overlapping. This idea could be used to create an active game-style story or enforce pacing. For instance, if there are multiple possible clips that could play at a single location, and they are not mutually exclusive, a timing flag could be used to stagger them over time.

Sometimes, an author might want to focus on absolute rather than relative time. For this, M-Studio provides flags that allow the user to set a range of dates and times that a clip can play during. In M-Studio's flag structure, this *date/time flag* is evaluated using date and time information supplied by simulators. If the current time falls in the acceptable range, then the flag evaluates to true. Obviously, these flags cannot be set because the user cannot affect the passage of time. There are many uses for these flags. An author might want to increase realism by providing alternate versions of clips shot at different times of the day to match the viewer's experience. Or, these flags can be used in story structure. For instance, in "Mind in Hand", there are several starting points set throughout the day. So, depending on what time the viewer starts the story, different pieces of the story will play, allowing the viewer's day to be synchronized with the day of the characters in the story. A viewer who starts the story in the morning will see clips of characters waking up for class or going for a morning run, while a viewer starting in the afternoon might see a character at lunch or working in lab. To implement this, the clips were divided into groups of morning, afternoon, and evening. These groups were all given flags corresponding to different blocks of time during the day. Thus, a user who spent the whole day on campus could see the entire story, but the pieces would be spaced out across the day.

There are other possible types of context information the server could obtain from clients and keep track of. To support this possibility, M-Studio provides context flags. *Context flags* provide a generic framework for authors to associate clips with any other kind of context data they need for their stories. Of course, this requires some advanced knowledge or contact with the server and client. The server does allow for importing custom evaluators, but the author would still need to find a way to get the context information to the server from the client. In M-Studio, however, this kind of information can be entered directly by the user during simulation. There are two kinds of context flags, one that handles numerical values and one that just matches descriptive strings. The numerical flags exist for cases when the data being used will definitely have a number value, like the current temperature, for instance. These flags are implemented using the same framework as global flags, allowing the user to test the current value. The string matching flags allow the author to implement any basic non-numerical context idea, such as triggering clips based on the current weather conditions. These flags are based on multi-valued flags, allowing the author to specify all the possible values that would satisfy this context

condition. So, if it were rainy, windy, or snowy, a clip might play of a few students on campus complaining about the Boston weather. If it were sunny, they might instead be expressing relief to finally have a nice day. This idea again leverages the mobile context-aware capabilities of the platform to allow for more customization in a story. Another use for context flags could be a multi-player game. The server could track the positions of all the players, and when two of them got near each other, incite some event. To do this, a video clip for the event would be created and tagged with a numerical context flag that represented a user's distance from another player. The server would automatically update that value each time a user's position was updated, and if it got low enough, that event would be allowed to play.

Even after creating flags, there may be a set of conditions under which there are multiple clips available to the user at a given location. At this point, the server must choose which clip to deliver to the user. There are several different methods the server could use. It could always pick the first clip in the list. Or, it could pick a random clip to introduce an element of uncertainty in the story. It could also use a rotating stack model, where after a clip is used once in that situation, it is moved to the bottom of the list and the next clip used. This would ensure that the same result is never given twice in a row. Another idea is to use heuristics to break these ties. A *heuristic* is a particular semantic term associated with a clip. These terms are given ratings within the clip. So, a clip dealing with a romantic subplot would be given a high rating for romance, while an action-packed clip would get a high rating for action. Or, the terms could refer to specific plot elements. For instance, in "Mind in Hand", you could create a heuristic referring to the missing ring and associate it with all the clips that deal with the issue. Heuristics are tracked just like flags. Each time a user views a clip, its heuristic values are incremented by the ratings associated with that clip. This keeps a running score of the types of clips the user has seen most. When the server has multiple possible clips at the same time, it looks at those clips heuristics and rates them based on the user's scores. The clip that best matches the user's heuristics is played. Since there is an element of user choice involved in these stories, this model tries to pick the clip the user would be most interested in. For instance, in the case of the missing ring, if a user had a high rating for the ring heuristic, it would imply that the user had been trying to follow the ring plot. Therefore, the viewer would probably prefer to see a ring clip over any other, to learn more about that story.

6.2.2 Creating and Editing Flags

In order to enable the users to create and edit flags, M-Studio provides a flag editor, which allows users to view and change all the flags associated with a given clip. The flag editor is sorted into three sections, one to edit requires flags, one to edit results flags, and one to edit heuristics. The editor consists of two components, a flag table and an editing panel. The flag table lists all the flags currently associated with the clip and their values. Selecting a flag from the table opens it in the editing panel. The editing panels are customized for the given flag types. For instance, the editing panel for clip flags provides users with a list of clips to choose from while the global flag-editing panel allows a user to choose from a list of operations. This interface allows the user to quickly examine and specify flag values, rather than using a direct scripting language. The flag editor also allows the author to create new flags, or add existing flags to other clips. Also, the author can specify a flag for one clip, and then add it to related clips. The flag editor gives the option to add flags to all clips, or to clips in a particular time slot, storyline, or location. For example, to create the previously described flag specifying what characters had been met, one would first make a new multi-valued flag. In the flag editing panel, one would specify that the character met in this clip was the character corresponding to the storyline.

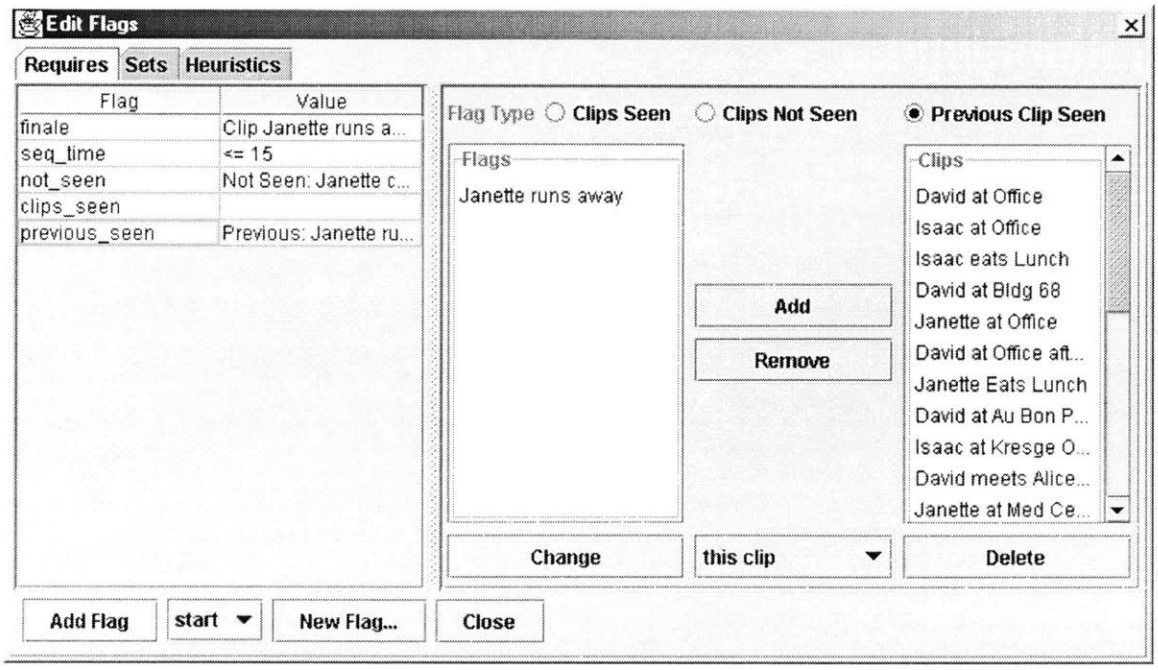

Figure 8: The Flag Editor, shown here editing a clip flag

Then, the flag would be added to all clips in the storyline. This process would be repeated for all storylines, and then specific clips where crossover occurred would be edited manually. This provides a powerful capability. In essence, the author creates a rule for clips of a certain type. Then, the storyboard uses its data structures to find all these clips and apply the rule to them, simplifying the production process.

This idea was extended to allow the storyboard to automatically generate certain flags that might be commonly used to structure stories. M-Studio can examine the structure of the story grid and analyze what clips might precede and follow each other. So, as the user adds and moves clips, it can automatically assign and update flags. Users can turn the flagging rules on and off as they go, allowing them to create an initial flag setup, and then move on to fine-tuning.

One convenient rule imposes sequential time on all the clips. This rule assumes that any events placed in the same time slot happen at the same time. Therefore, after seeing an event in a certain time slot, the user can no longer see events from earlier time slots because they are assumed to have occurred already. The user can jump into the future, but cannot backtrack in the story. This is useful in any linear narrative, because it prevents the viewer from seeing clips out of order. To generate this effect, the storyboard creates a flag, and places it in the requires clause of every clip, requiring the flag's value to be less than the time index of that clip in order for it to play. After each clip plays, the value of the flag is set to that clip's time index. This is easily updated as clips are moved around the storyboard.

The other key structural element that the story grid can determine is which clips logically precede and follow other clips. To execute a stricter linear narrative, clip flags need to be used to enforce sequencing between clips. M-Studio can automatically generate flags to this effect if it assumes that any clip that structurally precedes another needs to be seen in order for the clip to play. It does this by looking for the previous clip in the timeline for the same storyline as the given clip that has been inserted in the storyboard. It places this clip in the requirement clause of a flag listing the clips that have to be seen. It then finds all the clips that would logically follow it (if there are any) and updates those clips' requirement clauses to contain the inserted clip. M-Studio can even detect opportunities for crossover between storylines. If there is another clip at the same time and location as another possible preceding clip, it can also be added as another option in the requirement clause. This can also be done with for stricter plots by sequencing using previous clips. With this technique, a clip does not only require that a preceding clip have

been seen, but that it was the last clip to be seen. This method creates very tight plot lines, with the only option for divergence coming at crossover points between two storylines.

To generate these flags, the storyboard relies primarily on the physical layout of clips within the story grid. However, when clips are being moved or deleted, there needs to be a way to find all clips that are related by flags that might be affected by these actions. So, the storyboard maintains an additional graph structure to track relations between clips. The graph consists of a mapping from parent clips to a list of child clips. Clip A is the parent of Clip B if the requires clause of Clip B contains a clip flag or a timing flag that refers to Clip A. So, whenever a clip is moved or deleted, the flag structure can quickly provide a list of clips that will be affected by this action. The graph structure is updated any time a clip is added and when clip or time flags are created or updated. When a clip is updated, its flags are analyzed, inserting itself as a child clip where appropriate.

The graph also keeps track of possible entry points into the story. Entry points are clips that can play before anything else in the story happens, when no other clips have been seen, and all global flags are initialized to zero. When a clip is updated in the graph, it is checked to see if it is an entry point. This is done by comparing all the requirement flags to initial conditions and seeing if they would pass. Context flags are ignored because it is assumed that the circumstances under which they would be met could occur at the beginning of the story. This knowledge of starting clips is needed for the simulators to determine possible locations to start from.

Overall, the uses for the graph structure in this form are fairly limited. However, they are necessary because determining which clips directly affect other clips would otherwise require a search of every flag of every clip in the storyboard. To balance this tradeoff between memory usage and speed, M-Studio only maintains a small graph of clip relationships. There are obviously other relationships between clips that are not represented here. In theory, any two clips that both modify and require the same flag can affect each other. However, this information is irrelevant to making sure flag relationships are still valid when clips are repositioned, so it is excluded.

6.2.3 Story Tree

The full scale of relationships between clips becomes important when a user wants to view all the possible paths through the story. One downside to the storyline and location views

is that they do not accurately represent splitting in a storyline. For instance, in one of the finales for Another Alice, the killer can either be caught or get away, depending on how quickly the viewer chases after her. Both of these scenes involve the same character, so it would be logical to place them in the same storyline. However, they are mutually exclusive, so they should not follow each other in sequential order. Although the storyboard does not necessarily impose this order with flags, the layout could still be confusing for the user. The author could create a separate storyline to handle a character's diverging path, but this is cumbersome, and still does not provide an easy-to-read visualization that the events in one path are happening instead of the events in another, rather than just concurrently with them.

The tree view was created to address this concern among authors. This view represents all the possible paths a user could take through the story in a tree form. Each node in the tree represents a clip, and from that clip branches all the other possible clips that could be seen after the current path had unfolded. This tree is a good way to see the effects of the flag settings by allowing the user to view which clips can follow others. The author can also see where story threads can branch and come together by opening and closing different story paths.

We do not maintain the entire story tree in memory because it can become very large and change greatly when minor flag modifications are made. Therefore, it makes more sense to rebuild the necessary portions of it from scratch when modifications are made. Also, this tree is not needed for reference when straightforward modifications are being made to the storyboard. The story tree only needs to exist in memory when the author wants to interact directly with it, examining story threads and altering the flag structure. It is not a data structure used in the backend of the storyboard.

The story tree is composed of nodes. Each node has a single parent node and a list of child nodes. In this tree, each node corresponds to a clip. Since each node can only have one parent, clips can appear many times in the tree. Each path is drawn out explicitly. To build the tree, a root node is first created. The entry clips to the story are determined and they are added to the root node. Each of these clips is now a leaf in the tree. This data structure is operated on recursively. When it is called, it iterates through each of the current leaf clips in the tree. Each of these clips has a story path from the root node. For each of these paths, a flag structure is created, giving the state that would occur if the given set of clips had been played. For this state, all other possible clips are tested to see if they would be able to play. All of those that could play

are added to the current leaf node. If no clips are found to add to the current path, the path is completed and the end node removed from the list of leaf nodes. This process is repeated for all leaf nodes. Then, the recursive process is called on the new tree structure. When no more clips can be added to the structure, the whole tree is returned, encapsulated into one root node.

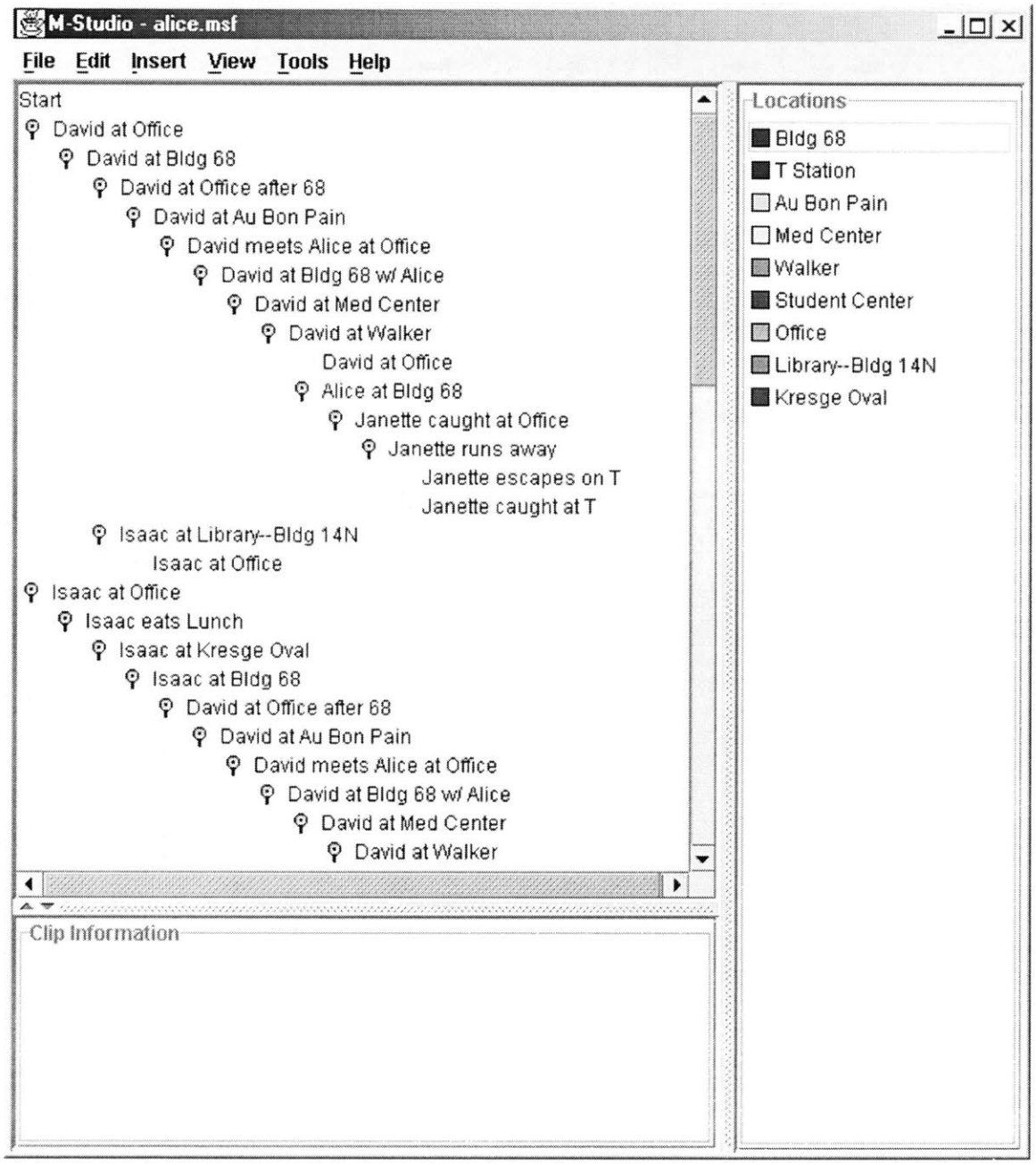

Figure 9: The Story Tree for "Another Alice"

The resulting graph will vary in size depending on the nature of the constraints placed on the story. If there were no constraints on a story except that the clips were not repeatable, the number of possible paths through the story would be equal to the factorial of the number of clips in the story. Of course, a story that explicitly specified many possible preceding clips could also be quite large. In contrast, the internal graph structure of an unconstrained story would be empty, while the graph for the widely constrained story would be quite large.

To reduce the amount of computation needed every time the story tree is altered, the tree can be rebuilt from a specific point. So, if a clip in the graph is edited, the tree assumes that this will not affect the ordering of the clips in the path before this clip. This results in a smaller portion of the tree that needs to be built. To do this, M-Studio refers to the graph structure to find all the current parent clips of the clip being edited and starts rebuilding from those points.

In addition to allowing for conventional editing functionality, the story tree also allows for further automatic flag generation using the adding and removing of connections between clips. When one clip is dragged or pasted onto another, M-Studio tries to make that clip the child of the clip it was dropped on. It does this by creating a clip flag, making a parent-child connection in the graph structure and removing any negative clip flags that would prevent one clip from following another. Of course, this will not necessarily have the intended effect, as other global flags might be interfering in the ordering. Similarly, removing a connection only deletes any clip flags that might tie the parent and child clips together.

6.3 Simulation

From the beginning of the development of M-Studio, it was clear that a static visual representation could not fully capture all the possible ways in which a story could unfold. An author could look at individual threads in the storyboard and see where paths could cross, but it was difficult to immediately understand what would happen if the viewer took a certain path through space and time. Also, there way no way to get an idea what the user experience would be in watching a certain path without manually playing each clip, one by one. The creation of flags and additional context data only further complicated the issue. An author needed to know if the flag structure was creating the desired effects, and ensure that clips were not being shown out of order. Even when the general order was correct, an author needed a way to determine if the sequences made sense, even under certain conditions, like when a user may have missed a

piece of the story. For instance, at the end of "Mind in Hand", two characters who are supposed to meet for a blind date are shown missing each other on their way to meet. However, if viewers have never seen any clips related to their date, they have no way of knowing that there is any significance to this event. To address these difficulties, a set of simulation tools was created.

The story simulation tools are all built on the same interface. When a simulation is started, a new, empty flag table is created. The simulation interface queries the flag table with location and context data, much like the M-Views client would query the server. The flag table, which also maintains a mapping from locations to clips, first finds a list of all the clips that could play at the requested location. It then iterates through these possible clips and checks their requirement flags against the current state of the flag table with the extra given context information, if provided. It maintains a list of all the clips that pass these flag tests. If there are multiple clips that could play at this time, it then arbitrates between them, first by picking out all the clips that are in the same storyline as the previously seen clip. (This action represents the assumption that the user is attempting to follow a particular storyline). If there are still several choices, it will make its choice by picking the clip that would logically follow the previously seen clip according to time slot.

If there are no clips from the same storyline as the previous clip, M-Studio will try to use heuristics to decide between clips. For each clip, a score will be computed based on how well the clip's list of heuristic values matches the current heuristic table, which represents the categories that the user has shown interest in so far. This score will be calculated by treating the clip's and the user's set of heuristic values as a vector (with zeroes for any values that are unspecified) and taking the dot product. So, a clip that is rated highly in categories the user has shown interest in will produce a higher score that is rated higher in other categories. The clip with the highest heuristic score will be returned. If there are no heuristic values specified, M-Studio will again refer to relative time slots to narrow down the list, and then choose a clip at random.

After a clip is selected, the flag structure acts as if it has been played, and updates the flag table accordingly. Each flag in the results clause of the selected clip is evaluated, and the corresponding entry updated according the specified operation and value. The categories in the heuristic table are also incremented by the amount specified in the heuristic list of the clip. The simulator interface can then query the table again with new location and context data. When the

simulation is complete, the flag table will be set back to its empty state, ready for a new simulation.

Based on this model, the simulator interfaces must gather context information to query the flag table with, and present the resulting clip back to the user. M-Studio currently provides two such tools, the story simulation and the map simulator. The story simulator allows for rapid playback of possible sequences through the story, while the map simulator tries to emulate the end user experience.

6.3.1 Story Simulator

The story simulator provides the author with a timeline to enter a sequence of locations and other context data. The simulator then produces a sequence of the video that would be seen by an end user who took that path. This tool allows for the easy viewing of sequences. The author can enter locations into the timeline by simply clicking on clips in the storyboard. Entire storylines can be entered automatically, allowing the author to watch the entire path of one character, or check that the flag settings are allowing this path to unfold properly.

Figure 10: The Story Simulator, in timeline view, simulating "Another Alice"

This tool is also useful for examining sequences that cross between storylines. When crossing into one storyline from another, a viewer comes into that story with a lack of knowledge about what has previously happened. Authors would want to check these sequences carefully for continuity and understandability. Also, the author might want to check the effects of changing context and time information on the story. For instance, the ending of "Another Alice" is time dependent. This tool would allow authors to compare the results of a path that reaches the final location quickly versus the path that reached it slowly by entering in two different times at the ending location.

The story simulator uses two different views. The timeline view allows a user to quickly enter a series of locations. The location table view allows a user to specify what the exact time and contextual conditions would be as these locations were visited. In the table view, each cell can contain a list of values for the different types of context information used by the story, with the column of the cell denoting which location this event takes place at and row representing sequence in the timeline.

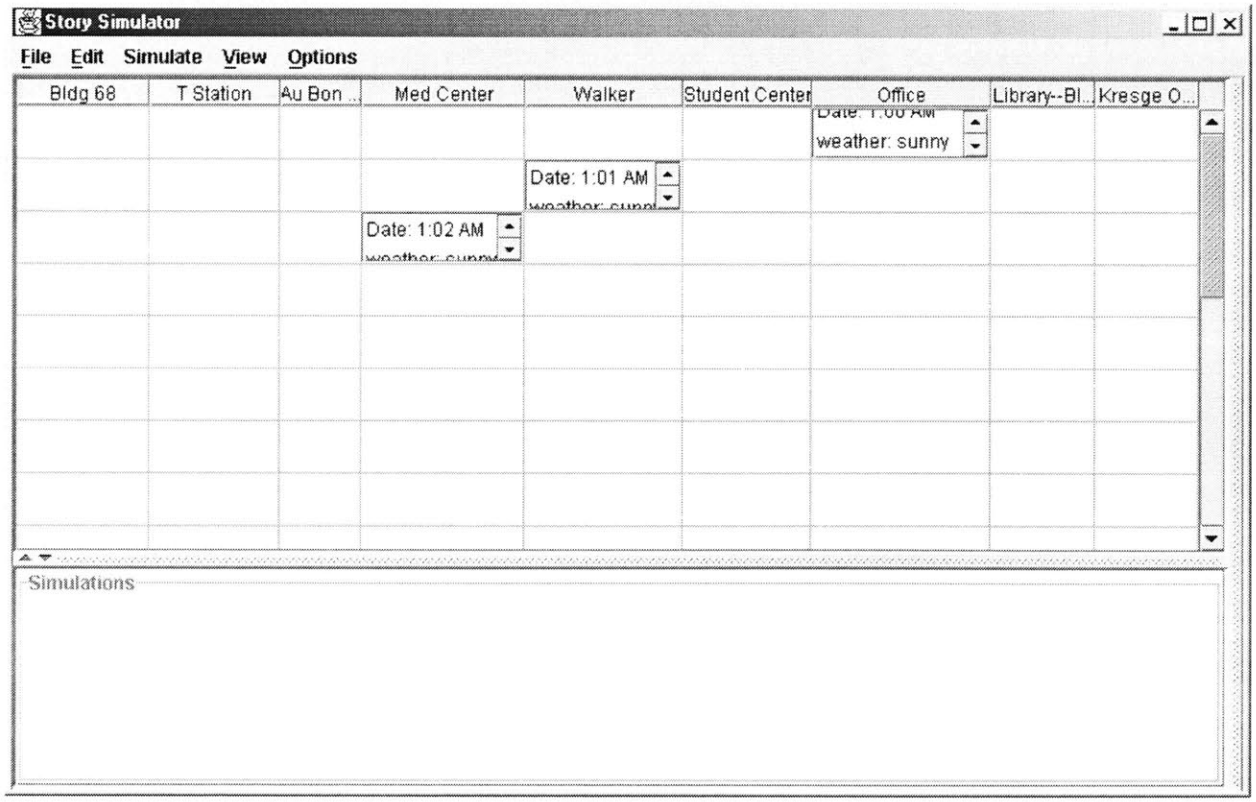

Figure 11: Specifying Simulation Constraints in the Location View

To ease the task of entering context information, default values can be set, and the times can be automatically generated off a starting time, based on time slots. The timeline view and table view are synchronized, so the user can easily switch between them. If the user has entered in a path of locations in the timeline, and date or context information is needed, the user is prompted to enter that information before simulation begins, and those values are used as default throughout, with times incrementing at one minute per slot in the timeline.

To produce the simulation, the simulator queries the flag table as previously described with each piece of location and context data. It keeps a list of the returned clips, and then builds a simulation based on the type of clip that is used. If the clips are all video, a temporary WVX play list is created and launched in Windows Media Player, allowing the author to view the entire sequence. If entirely text and image clips are used, they are joined together into a web page, producing a kind of script for the whole sequence. If a mixture of video and text clips is used, they are combined by embedding Media Player elements into a web page along with the text and images.

6.3.2 Map Simulator

The goal of the map simulator is to allow an author to understand how a story unfolds in space and time without actually playing it on the client. Obviously, no desktop simulation can reconstruct the mobile experience, since one of the platform's major objectives is to exploit the viewer's surroundings in the video experience. Also, desktop simulations cannot detect technical subtleties, such as the actual quality and range of location detection systems. Clearly, story creators should execute a physical run-through of a production before widely releasing it. However, it is not feasible to do this every time the story structure changes. Narratives like "Mind in Hand" are intended to unfold over the course of an entire day, making it impractical to physically simulate it all at once. The map simulator allows the author to interact with an animated map and adjust time and context information as the story plays out.

A map simulation begins with the user selecting a starting point for the story. After the introductory clip plays, the map simulator asks the user to select where to go next and supply the current time and context information, as needed. When the user selects the next location, a marker appears on the map and "steps" towards the next location. When it arrives, the flag table is queried with the current context information. If there is a clip at this location, it is played.

Then, the user is prompted to enter the next location. This interaction allows the user to better understand how clips are related to their locations and how a viewer actually has to traverse space to see a clip. Actually viewing the map may also help detect technical difficulties, such as sequential clips that are situated at locations spread too far apart. Although the map simulator does not directly detect this, the author might also notice that another location is passed by quite closely when a viewer moves between two other locations. This could cause other clips to be triggered. Furthermore, the map simulator solidifies the idea of decision-making on the part of the viewer and how this impacts the structure of the unfolding narrative.

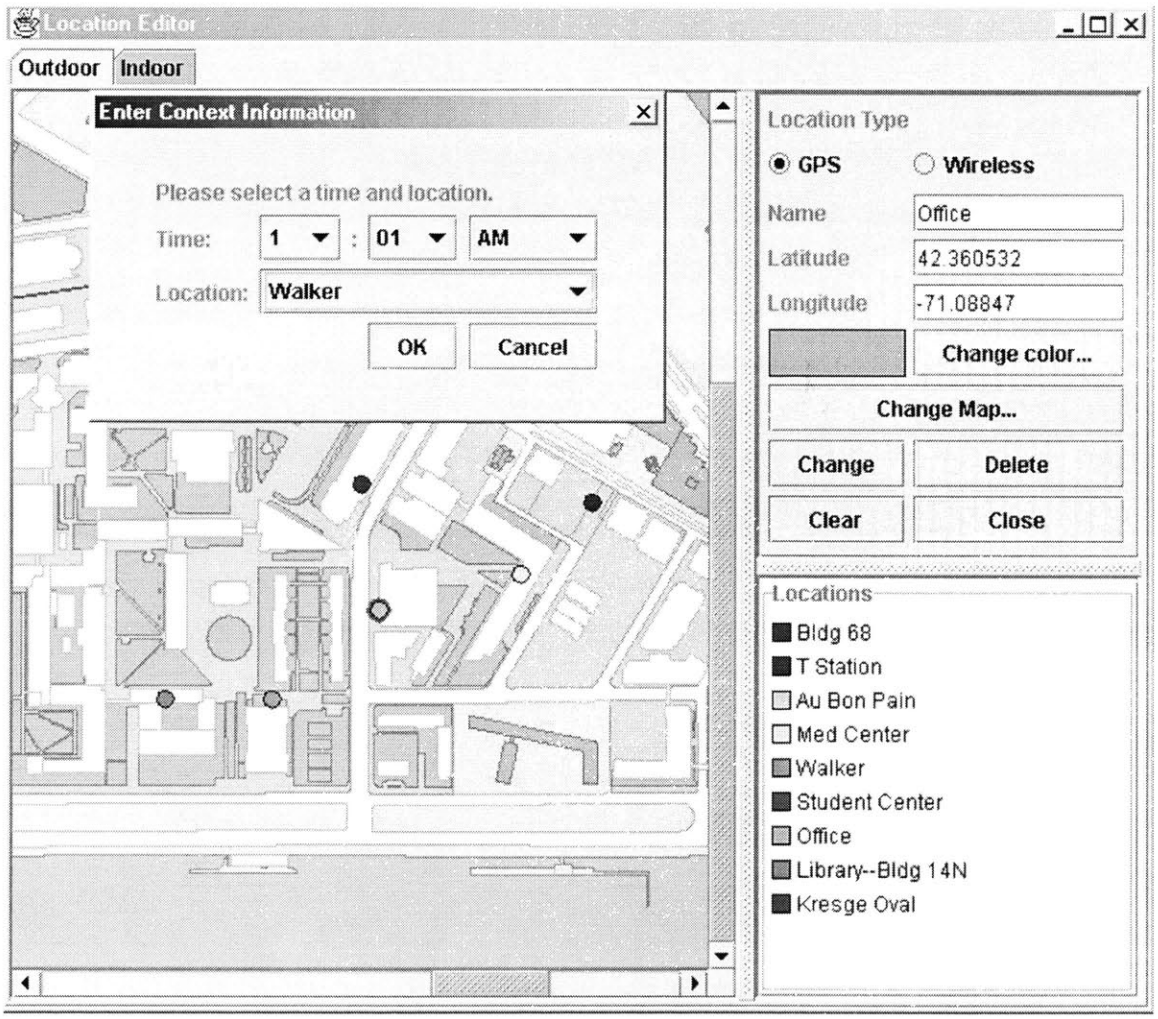

Figure 12: Map Simulator

6.3.3 Storyline Generation

Although simulation was always a necessary component of M-Studio, the creation of the flag structure made it even more critical. An author needed a way to test that the flag setup was producing desirable story paths. An incorrect flag structure can both allow for unwanted paths and prevent logical paths. Furthermore, it is hard to actually predict which paths should be allowed without actually seeing them. Although it is possible to test all the different paths through the story with the story simulator, this would be cumbersome, particularly for a loosely constrained story with many events. Therefore, M-Studio needed a way to generate all the paths through the story automatically, allowing the user to quickly understand the scope of the possibilities for the viewer experience and identify problems. Thus, the path generator was created to parse all the possible paths through the story structure. Since the number of story paths could be numerous, it also gave the author the ability to generate paths subject to constraints of location, time, and context.

The story tree, which gives the author the ability to view paths through the story and directly manipulate flags, was actually built using the path generator. So, the path generation algorithm is similar to the one previously described. However, instead of using nodes, it builds up individual paths. When there are no constraints, the generation function starts off with an empty flag table and a list of all the clips in the story. It then determines which clips could play under these initial conditions by evaluating flags. It then creates a path for each of the possible starting clips. The generating function is then called recursively with each of these paths. Each time the generating function is called, it evaluates the given path to update the state of the flag table. Then, it again looks through all the possible clips that could be next in the path and finds those whose flags allow them to be added. For unconstrained generation, context flags are not used. However, timing flags are evaluated. To do this, the generator assumes that one minute passes between seeing two clips. If a clip fails because less than the minimum amount of time needed to meet a timing constraint has passed, the function will increase the time increment until the flag evaluates to true. The time that each clip would be able to play is saved along with the clip when it is added to a path.

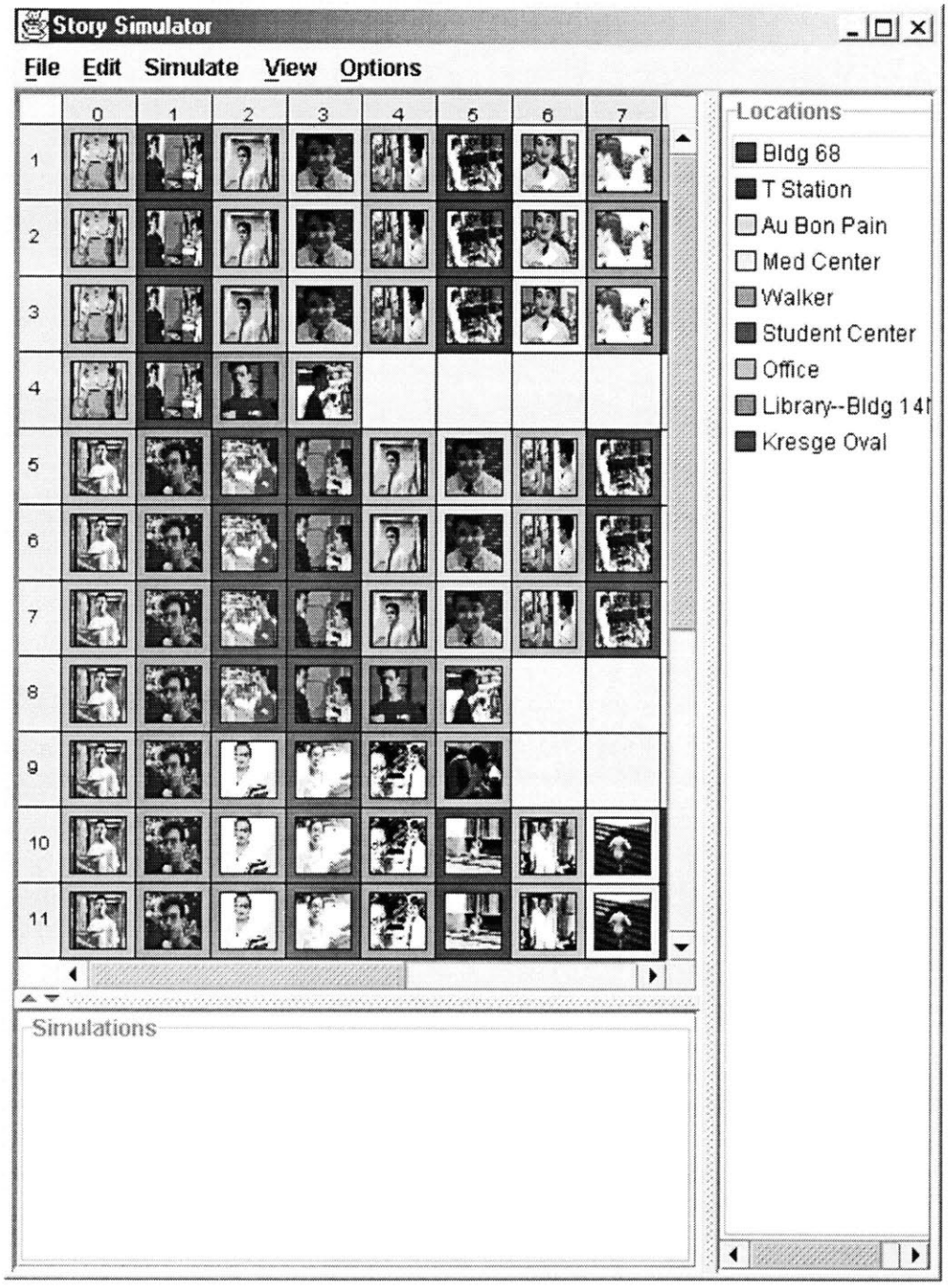

Figure 13: Story Paths for "Another Alice"

After the generator has determined a list of possible clips, it creates a new path for each clip by adding them to the previous path the generator was currently working on. The generation function is then called again on each of these new paths. When the generator finds that there are no further clips to be added to a path, it adds it to a list of completed paths and goes on to work on the next path. Of course, it is possible for the viewer to stop the story part way through a path

and never reach the end during actual playback, but the generator will keep adding clips as long as they are available, just like the user would continue to receive clips as long as they played the game. When all paths are completed, the list of paths is returned to the simulator. The display is simple, with each path laid out in a table, with the option to play each one. The clip display is the same as the one used by story layout panel, allowing the user to see the clip's location and title or thumbnail image.

To optimize the search, the generator tries to immediately eliminate clips it knows that cannot be played, given the last clip in the path, before passing the new path to the next round of generation. For instance, if the last clip is not repeatable, it is eliminated from the list of possible next clips. Or, any clip that requires a previous clip that is not the last clip seen is removed. Similarly, any clip that requires that the last clip not be seen is eliminated. For a well-specified story, these optimizations provide a dramatic speedup. For less specified stories, the overall performance decreases slightly.

Large and unconstrained stories can generate a huge amount of possible paths. To handle this issue, the storyline generator produces 150 paths at a time. When it reaches the limit of paths, all calls to the generator are saved with their current state. It then notifies the user that there are more paths to be seen. The user can then restart the generation after examining the current set of paths. If clips are repeatable, then a path could theoretically be infinite. To prevent this, M-Studio places a constraint three times the total number of clips in the story on the length of any possible path.

6.3.3.1 Generating with Constraints

Although it is important to understand the scope of all the possible paths that can unfold during story play, an author may be more interested in what is possible under a specific set of circumstances, particularly in the cases when the total number of possible paths is large. So, the storyline generator allows users to generate subject to constraints of location, time, and other contextual information. This information can be entered directly through the story simulator's user interface. The author can then select any combination of the three types of information, and generate all the possible paths that would occur, given those conditions.

Since location is a key component to all M-Views stories, it is clear why an author would want to know all the possible story outcomes for a particular path through space. Location is

also an easy way to narrow down a large set of paths. Or perhaps, the author knows that the user will always start at the same place, making storylines with other starting points irrelevant. The storyline generator allows the author to specify a complete or partial path through space, and find all the possible storylines that could unfold given that path. When the generator looks at the path the user specified, it assumes the locations correspond to an ordering. Therefore, the first clip played in the path must be at the first specified location, with the second clip at the second location and so forth. If a slot in the timeline is left blank, it is considered to be a wild card. This means that a clip can be seen at any location between the previous location and the next location. However, there could also be no clip that played between these two locations. Similarly, if a user left three blank slots between two locations, then at most three clips could play at any location between the two specified locations.

The generation method with location constraints works similarly to unconstrained generation. It first checks flags to see what clips could play under the current user state. If there is a location constraint specified for this slot in the story, then it must be met for the clip to be a possibility. Also, each path must be sure to match the given location path before it can be confirmed as a viable path through the story. To match the pattern, it goes through each location constraint one by one to see if it is met by the appropriate point in the story path. If the path matches, it can continue to be operated on. This search also performs the same optimizations as the unconstrained search. However, for each slot with a location constraint, it can quickly reduce the number of clips it has to search by only checking clips at the given location.

Time information can be used on its own or in combination with the location information. Time can be useful to help focus on a specific set of paths for a story. For instance, in "Another Alice", the viewer is assigned a character to start following based on the time of day the story starts. So, specifying a start time would allow the author to focus on one character. For "Mind in Hand", the day is split up into three discrete sections based on time. So, time constraints would allow the author to analyze one section of the day at a time. When locations are involved, those constraints are evaluated first. Then, it is determined if a clip could play at the specified times. As with location generation, the entire path can be explicitly specified, or wild cards can be used. With a time wild card, any clip is allowed, so long as that clip can be played between the previously and next specified times. If there are multiple wildcards, a one-minute spacing between clips is assumed. For instance, imagine there were two time constraints, each associated

with a location, one at 12:00 and the other at 12:06. They are separated by three wildcards. If the generator were to find a wildcard clip that played at 12:05, the next clip would have to meet the 12:06 time constraint and the associated location constraint, because there would be no time for another wildcard clip to play. Without locations, the time-based generation still follows the same wildcard rules. The time specifications are also used in evaluating timing flags. So, when determining how much time has passed between two clips, it will look directly at the specified times.

Context-based generation can be used for the same reasons as time-based generation, to narrow focus on particular story paths. Or, it can be used in combination with location and time constraints to evaluate a particular set of circumstances. Again, context constraints can be fully or partly specified. When a slot in the specification is left blank, all context flags are ignored during evaluation. If the context for a particular slot is not fully specified, and a flag refers to an unspecified value, the flag is evaluated to false.

6.3.3.2 The use of storyline generation in "Another Alice"

One practical example of the usefulness of the storyline generator is the creation of two different flag structures for "Another Alice". This story was created for the original server structure, and thus did not have any direct flag specification. However, there was a general idea of how the story should work. The story always starts out with the viewer receiving an email telling them to go to the office. There, the viewer meets one of three characters, depending on the time of day. At the end of every clip, the characters tell the user where they are going next, though they are not always truthful. Characters had independent story threads, except when they were at the same place at the same time as another character. Users could only see scenes from one storyline until they met another character. Then they could see scenes from either storyline. Also, one of the possible endings involves the viewer chasing after the suspected killer. The ending depends on whether the user arrives at the final location fast enough to 'catch' the killer.

The first attempt at a flag structure for the story assumed that the user would go where the character told them. Thus, in order to see one clip, the viewer would have to have seen the clip before it, except in crossover situations, where there were two possibilities for the previous clip. Thus, the entire story was implemented using clip flags requiring the previous clip. Time-of-day flags were created to decide which introductory clip would be shown. The finale was

implemented using timing flags. The overall story was very easy to create using a few custom timing flags in addition to the automatically generated clip flags. A quick check in the storyline generator showed there were 18 possible paths through the story, and each one made sense. This was somewhat interesting, because by the original author's estimation, there were only 12 paths through the story, again highlighting the importance of automated testing. However, this made for a very narrowly constrained story. It did not allow the viewer to miss clips. If the user did not follow the character's directions, the story would simply stall until the user went to the right location to get the next clip, even if, according to the idea of the story, there could be something happening at the user's current location.

To make the story experience closer to the original intent, a new flag structure was created. The time flags to choose a starting point and the timing flags to create the ending stayed in place, but all the clip flags were removed. Instead, a multi-valued flag representing characters met was created, and added to each storyline. The crossover clips were updated by hand to allow for viewers who had met either character in the clip to view it. The sequential time flag was also used so the clips would still execute in order. This setup seemed to capture the original design for "Another Alice" much better than the first flag structure.

However, doing a storyline generation on this version revealed that the total number of paths through the story had jumped from 18 to a few thousand. It quickly became clear that several undesirable effects were occurring. One problem that immediately became clear was more of a technical one. It was possible for clips that occurred at the same location to all happen in a row. Given the design of the client, which queries the server every five seconds, it would be almost impossible to get away from a location hotspot quickly enough to not receive all the clips at that location at once. One way to deal with this would be to create a flag that keeps track of the location of the most recent clip seen and have each clip require that the previous location visited was not the same as the location for the clip. A less restrictive model could instead use timing flags, and enforce a minimum amount of time that has to pass between clips at the same location.

Generating with location constraints revealed another problem. When there were clips in the story that played at the same location, it was possible to get a later clip before an earlier clip, jumping the user ahead in the story and preventing them from ever seeing the early clip. To

avoid this, a flag was created to determine whether or not the early clip could still play. If the early clip was still available, then the later clip could not play.

Another major issue was that while the remaining storylines were still technically correct, they were ultimately unsatisfying. If the user happened to wander into the location of a clip that happened late in the story, time was pushed forward and the bulk of the story was missed. To fix this issue, a flag was created to track the number of clips that had been seen so far. Each clip was then associated with a minimum number of clips that had to have been seen in order for it to play. So, if the user had just started the story and walked through the location of the final scene, it would not play until the user has viewed more scenes.

The final version of "Another Alice" approximated the desired results more closely than the first version, but it required a significant amount of changes from the original plan. This highlights the need for an author to carefully test the story structure before publishing the story to the server. The effects of flags are difficult to determine by hand, and the author can often be hampered by assumptions of what valid story paths should be.

6.3.4 Tree Explorer

In order for the author to be able to better understand how the user decision process affects how a story unfolds, we created the tree explorer. This tool allows the user to see all the possible clips that can be seen at any time in a story path. The user can then select a clip and see how that decision effects how the rest of the story unfolds.

When started, the tree explorer determines all the possible starting clips and displays them. The user can then select which one to start with. The explorer then evaluates what all the next possible clips could be, given that the user made that first choice. It determines this by checking all possible clips against the flag table, just as it does when it is building the story tree or generating story paths. These options are presented to the user, who can then select another clip to add to the path. If at any time, only one clip is available, the explorer will automatically add it to the path, and find the next set of possible clips. This process will continue until it has a choice to present to the user.

Since the tree explorer prunes possible paths as it goes, it requires far less memory and time to run than the story tree or the storyline generator, making it a good solution for stories that are too large to be viewed as a whole story tree. It also provides an easy way for authors to

visualize what the possible choices could be available at any given time. They might discover an unwanted choice is present, or that a clip is being unnecessarily restricted. The explorer also provides a mechanism for the user to back up as far as necessary in the tree so they can see what would happen if a different choice had been made.

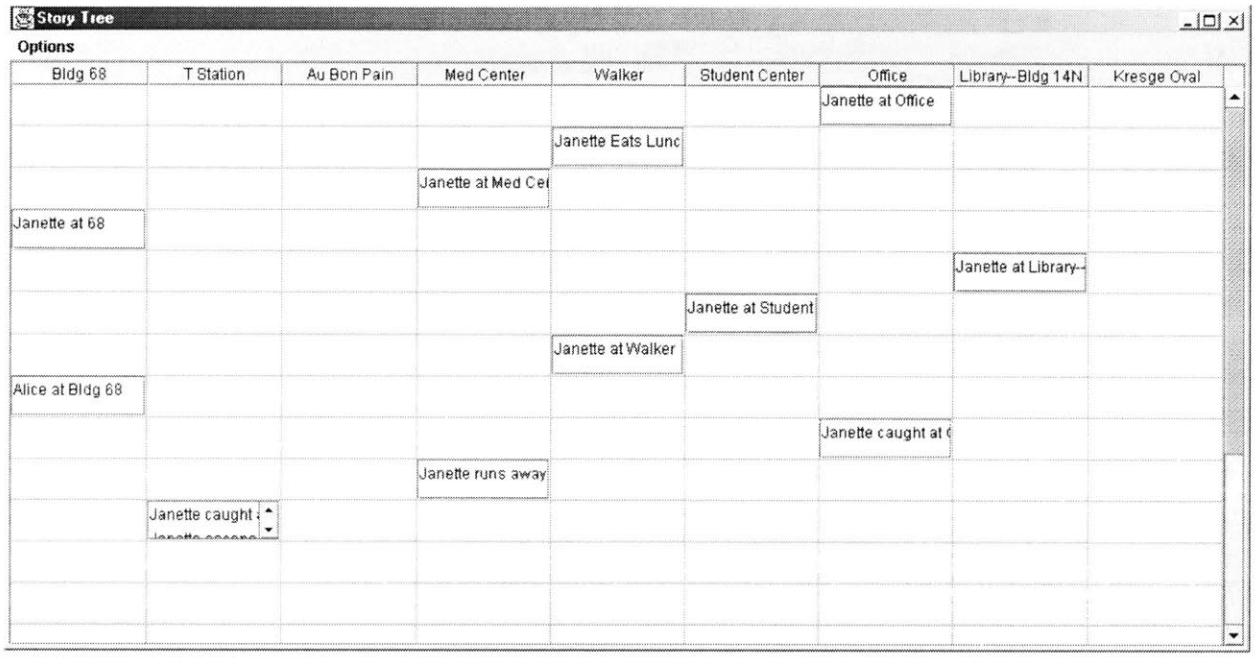

Figure 14: The Tree Explorer

6.4 Server Communication

Communication between M-Studio and the M-Views server was designed to be as simple as possible. The server is contacted only at the very end of the story creation process, when the author is prepared to publish the story. This eliminates most of the burden of handling M-Studio requests from the server, and allows authors to use the authoring tool without an active network connection.

In the server model, a story is comprised of a list of events that are associated with locations. Each event also has requires and results flags, just like in M-Studio. Every time the client queries the server with context data, the server first determines if the client is at a location where it could receive events. If it is, the server examines all the events at that location and determines which ones could play given the current user state. Ties can be broken using

heuristics or a random selection model. If an event is sent to the client, its results flags are evaluated and the user state is updated.

We decided on an XML schema for communication between the server and M-Studio. M-Studio iterates through all the clips in the storyboard and formats them as server events to produce the XML document. This document is then sent to the server along with all the media content for the story. The server then stores this XML story script until a client requests to play the story. It then parses the script into story events so it can start evaluating queries from the client.

All the information needed to create an XML story event is contained in an M-Studio clip. To describe the process of creating these events, we will refer to Figure 15, an XML story event from "Another Alice". Each story event must be tagged with an identification number, so the clip's own unique ID is used. Within the event header, there are several fields. M-Studio generates the context field by looking at the location associated with the clip. Within the implementation of each location type, there is a function to produce a string describing the location in a form the server can understand. In this story, GPS was the location detection system, so latitude and longitude coordinates represent the context. Along with location, it is specified whether or not the clip is repeatable. These two pieces of information are in the event header to allow the server to quickly narrow down the choices for a particular query before it starts parsing flags. The story field is used to keep track of what story events are coming from when an end user is subscribed to multiple stories at once. This field is derived from the name of the M-Studio file. The remaining fields in the header are used to create the message to send to the client, which was designed to resemble an email browsing program. All the fields except the subject will be filled in automatically when the server sends the event to the client. The subject field is derived from the clip's title.

```
<ID24 Context="42.3595596,-71.09008984" Date=""
       From="" Repeatable="false" Story="alice" Subject="Isaac
      eats Lunch" To="">
      <Requires>
            <_ expr="?LS" value="ID25"/>
            <seq_time expr="LTE" value="1"/>
      </Requires>
      <Message>
            <MediaURL>
                  /alice/F-Isaac Walker Food1-13.wmv
```

```
            </MediaURL>
            <Text>
                  Isaac tries to enjoy lunch at Walker,
                  but runs into some unexpected company.
            </Text>
      </Message>
      <Results>
            <seq_time oper="=" value="2"/>
      </Results>
      <Heuristics>
            <Food value="3">
      </Heuristics>
</ID24>
```

Figure 15: A sample XML story event

After the header, the XML object is divided up into four subsections. The first section corresponds to the requirement clause of a clip. It contains a list of requirement flags that must all evaluate to true in order for the clip to be able to play. Each requirement flag is tagged by the flag name. In the case of special flags, like clip flags, timing flags, or date and context flags, the null tag is used to point the server to a special evaluation function stated in the expression field. In the case of clip flags, the expression `?LS` tells the server to refer to its user state table of clips that have been seen, and find which was the last clip to play. The server will then see if its ID matches the value required by this flag. The second required flag is the sequential time operator. This is a global flag, so it simply looks up its value and determines if it is less than or equal to one, in this case.

The next part of an event is the message. This is used to create the message object that is sent to the client and displayed in a user's inbox. In addition to the message headers already specified, each message can also have a text component and a media component. When a message is opened on the client, its text will be displayed. The user can then press a button to open the associated media file in the appropriate player. The text is taken from the notes the author enters about each clip. The media URL refers to the location the clip will be placed on the server when it is uploaded.

After a clip is sent to a client, the results section of the event is evaluated. Results flags work like requirement flags. They are tagged with the name of the flag they modify. Clip flags, timing flags, and context flags cannot be updated. Like with requirement flags, there are two

fields, an operator and a value. The server will retrieve the current flag value and update it by applying the requested operation with the given value.

The final section is optional. It allows authors to specify heuristics to break ties when multiple clips can be played at the same time. A heuristic is specified like a flag, except it only needs one field to specify its value. This value will be used to determine the best match when the server is picking clips. It will also be used to update the current heuristic values when a clip is played.

The full details of the XML schema with all currently supported keywords and operations can be found in Appendix A.

Once M-Studio has generated this XML document, it will contact the server. The server will create a directory for the story, and M-Studio will upload the story script along with all the media content. The server will then make the story available for subscription by M-Views users.

7. EVALUATION

Evaluating M-Studio is a challenging task. Although scenarios can be created to test the usability and intuitiveness of the tool, these tests do not address the issue of how M-Studio affects the task of story creation for the mobile platform. To truly understand these issues, we must examine the role of M-Studio in the development of actual mobile cinema productions. Therefore, the evaluation of M-Studio has focused on its use in the two M-Views productions, "Another Alice" and "Mind in Hand".

"Another Alice" was designed for the first iteration of the M-Views server, which relied on strict storylines and absolute time along with locations when choosing clips, so flags were not used. M-Studio was still in early development at this point, so only the main storyboard and story simulator existed. To evaluate the tool, we asked the author of "Another Alice" to use the storyboard to build her version of the story. She had no problems doing this, because the story conformed very well to the model of distinct storylines and it unfolded in absolute time. She found the interface for associating clips with locations to be intuitive. After looking at the storyboard, she commented that the story would have been designed differently had she started off using this tool. She said that had she designed the story with M-Studio, she would have made it denser, so there were fewer time gaps between clips. She also said she would have made it more complex, adding in more crossovers. She commented that she had been trying to do a

similar grid layout on paper, to help visualize the crossovers, but that it was difficult to do. The simulator was useful to viewing sequences of clips together, but because the crossover was so strictly defined, there were no unexpected results.

Another lesson learned in the field evaluation of "Another Alice" is that M-Studio cannot completely replace field tests. While M-Studio is useful in understand story structure, it cannot simulate some of the more practical aspects of the mobile experience, such as GPS signal being blocked by buildings or going too far out of range to receive wireless signal. The story developers must investigate all these issues when they decide where to place their story in space.

After the new server was developed and the flag system was introduced, "Another Alice" was updated to fit this new model. As described in the earlier section on storyline generation, it was not immediately clear what the best flag structure would be for the story, and it took a great deal of testing to finally determine the optimal setup for capturing the original intent of the piece. It was obvious that the addition of the flag structure, while adding much needed flexibility, had complicated the design process. Now, the author did not just have to layout the clips in the right order, but had to understand the subtler aspects of the story structure. Automatically generated flags made it easy to create a fast prototype of the story structure, while the generation tools helped to analyze that structure and find what changes were needed. So, while the new flag structure added complexity to the design process, the additional tools in M-Studio preventing it from overwhelming the user.

The development of the flag structure was also a major issue in "Mind in Hand". One major goal of the story team was to produce each clip so that it could stand alone as a mini-movie in the story. So, theoretically, the overall story structure should not have been affected by the content of each clip. The initial plans involved five major storylines, each corresponding to a major character, who was the primary star of a clip. The issue then became how to time these clips. Each storyline had its own order, but the clips had to be ordered in relation to clips in other storylines. The design team initially thought they needed fewer timeslots than they ended up using, because they did not realize how explicitly clips would need to be sequenced to produce the desired effect.

At first, the clips were divided into three time periods, morning, afternoon, and evening. However, within those periods, they needed to be laid out with respect to each other. For instance, one character's morning focuses on him going for a run. Since this takes place over

several clips, they could not all happen at the same time. Furthermore, a user could not see the first clip of him tying his shoes in preparation for the run after seeing him running already. So, sequential time was used within each of the three major blocks to prevent seeing clips out of order. This became our initial prototype for simulation.

Simulation raised several issues. One major point was causality. For instance, one of the final clips of the story involves one character returning a ring he found to its owner. It was discussed whether or not a user should be able to see this clip without having seen any of the previous clips pertaining to the ring. While all the scenes for this story were designed to work as standalone scenes, there are story arcs that travel through the story. The question became whether or not to enforce them. Also, there was the issue of the time buckets. A user who shows up late in a time division or turns off his or her iPAQ for a period of time can miss many clips. Also, the sequential time model was preventing some backtracking that did not result incoherent storylines. Furthermore, simulation showed that if a user went to a location from the last time slot of a block, no more clips could be received until the next time block started. Clearly, a different model was needed.

The next proposed model involved starting the story based on time of day, but then requiring a certain number of clips to run to switch time segments. This would give the time ambience effect desired by the authors, but not trap viewers in a particular time of day. However, this conflicted with another important goal of "Mind in Hand", to encourage messaging between users. If an absolute timeline were used, the users would be having the same experience at the same time, and would be more likely to share and discuss it. This would be less likely to occur if the stories were happening asynchronously. This is a point that was not be revealed through simulation, but rather through some knowledge of the design of the client and messaging system.

Another idea for handling the time issues was to trigger clips off the time that had passed from when the user started the story. This would require the use of numerical context flags and a custom evaluator for the server. This would prevent users from getting stuck in a certain time of day, while also keeping them from leaving a time block too quickly and missing out on a large chunk of the story. It was also decided that the time blocks were too large for the amount of content that currently existed. A denser and more realistic experience could be had by breaking time into smaller pieces. This would also eliminate much of the need for a strict sequential time

operator. The possible bad orderings that could still exist would be easy to detect with the storyline generator and could be fixed with individual flags.

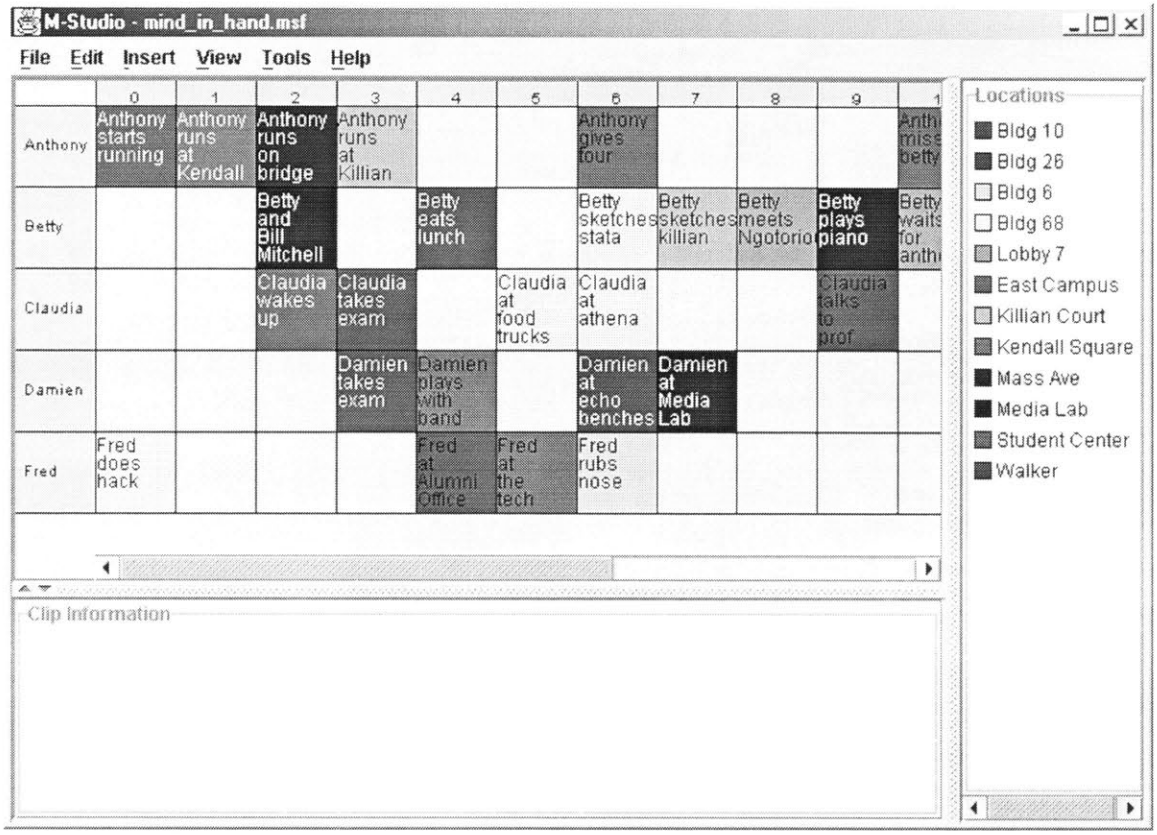

Figure 16: A preliminary storyboard for "Mind in Hand"

A major benefit of M-Studio in this production was that the effects of these different structures could be easily simulated from the desktop. After entering the basic story paths, we could produce many different versions of the flag setup and run simulations on them. The simulator allowed us to input location and time information, so we could recreate many different possibilities that we were considering to ensure that users got a desirable experience no matter how they used the device. One major goal of "Mind in Hand" was that it could be enjoyed by anyone visiting the campus. This meant that the story had to accommodate people who had to do things other than chase after clips in the story. M-Studio's ability to generate paths constrained by times allowed us to create a simulation where the user turns the iPAQ off for an hour. This simulation could be saved and tested all the different story structures. Using the map simulator, we could follow several different likely paths through campus to see how an ordinary user might travel through the story. This allowed us to see if these paths produce a compelling storyline or

if a user had to go out of his or her way to find key pieces of the story. Although it is impossible for an author to predict all the different scenarios in which a viewer will user the device, being able to see the results for a given path is a powerful tool. Doing physical, real-time tests with the iPAQ would be very time consuming, particularly for a story like "Mind in Hand", which is intended to unfold over the course of a day.

Also, since the video for "Mind in Hand" was being edited while this evaluation process was occurring, the ability to make text clips was a convenient feature. All the scenes were entered with text comments from the story scripts. As more video became available, it was easily placed into the story. Laying out "Mind in Hand" was also easy, even though it was not as linear as "Another Alice". There was still a fairly strong structure based on storylines, although there were several clips where we had to decide which story to place them in because they included more than one main character. M-Studio's interface for manipulating clips within the storyboard made the task of rearranging clips' timing with respect to other storylines fairly simple.

Since software development for M-Studio has been taking place concurrently with story development, it has not been possible to examine a case where authors used M-Studio extensively in pre-production. It would be interesting to perform a test in which authors actively used the tool from the beginning of the creation process and compare the results with a test where it was not used to see how it would influence story structures. Would it cause people to impose unnecessary constraints on their story creation? Would authors plan their scenes differently when they understood the many possible sequences that they seen in? An author might approach a scene differently knowing all the different events that might lead up to it, and what different pieces of information a user might have when viewing that scene. The author might decide that there should actually be different versions of the scene that depended on the previous path taken.

Another goal of "Mind in Hand" is that it be evolving. The hope is that other story teams could produce their own segments about MIT campus life in the past, present, and future. As these segments are added into the story, a complex and changing web should emerge, creating an ongoing and expanding experience for visitors to campus. Clearly, M-Studio will play a role in the creation of this story web. Will the visual interfaces to the story allow new creators to find a

place to incorporate their content? What other tools might be necessary for authors to integrate and connect separately produced stories?

To fully understand the implications of M-Studio and the M-Views platform, long term study will be needed. When M-Views if fully and publicly deployed, we will be able to see the effects of M-Studio on independent user content creation. Will people take advantage of the tools provided to produce their own content for the M-Views system? What will this content look like? Most importantly, will this supply of original content attract more users to the M-Views platform?

8. FUTURE WORK

Evaluation showed us that the most challenging part of developing the story seemed to be selecting an appropriate story structure and designing a flag setup to produce this structure. Therefore, one area of future work for M-Studio should be to provide further flag generation tools. Presently, flags can only be generated on explicit structural elements, like relative ordering, crossover, and location. It would be more desirable to be able to generate structure based on narrative elements within the story. To do this, users could annotate their clips with semantic information. This information could be used to imply connections between clips that do not have any obvious relation in the physical story structure. For instance, in Mind in Hand, if each clip focusing on the ring was annotated as such, the tool could suggest flag connections between those clips. A rule could then be created to enforce an ordering amongst clips that refer to the same plot point. To further extend this idea, the semantic annotations could be combined with a common sense reasoning tool, allowing M-Studio deduce structural elements, like that finding a ring would have to precede returning a ring. This would take some of the burden off the author in determine how to arrange flags to express such connections.

The use of semantic information will also make stories more extensible. One goal of "Mind in Hand" is that it be an evolving story web that can be contributed to by many story makers from the MIT campus. If the web grows very large, however, it will be difficult for new contributors to find places to insert their storylines. However, if annotated, the creators will be able to search the web for pieces that refer to similar topics as they ones they are considering making stories about.

In the longer term, we hope to be able to construct multi-player context-aware narratives using M-Studio. Although the server currently allows messaging and video sharing between users, it does not allow them to actively collaborate in a situation where one user's actions could affect another user. This idea could result in a very compelling interaction model where users work could together to follow different paths through the story, or compete against each other in a real-time, mobile game. However, the addition of these possibilities can make storylines even more unconstrained, and places an additional burden on the story creator. To help the author tackle these issues, M-Studio would need to provide a way for authors to predicate clips on conditions shared globally throughout the game. Simulation tools would also need to be enhanced to take into account the effects of multiple viewers, perhaps allowing several people to participate in the same simulation at once. This new model may be the compelling form that encourages widespread use of the M-Views platform and the availability of tools to support authors in story creation may inspire more creation of original content.

9. CONCLUSION

The field of context-aware research offers exciting potential for the development of new technologies in many areas, including story delivery. The ability to gather data about a viewer's environment provides for many possibilities in customized video and narrative. The M-Views platform was designed to examine these possibilities by using handheld computers, positioning systems, and the wireless network to allow for the playback of location-based stories. This platform created the possibility for story structures that go beyond the standard linear story. Mobile storytelling creates the potential for multiple storylines to be unfolding in different places at the same time, making it possible for a viewer to navigate between these storylines by physically moving through space. Stories can be customized to match a viewer's interests, designed to provide an immersive experience, or used to augment a physical space.

Although these new story forms offer exciting possibilities, they can be quite difficult to realize due to their potential complexity. It became clear early on that authors would need tools to allow them to associate their content with context information, layout and visualize parallel story threads, and simulate all the paths users could take through their narratives. M-Studio was developed with these goals and tasks in mind. The location editor allows authors to quickly add the element of context to their clips. Several different storyboard views provide simple ways for

authors to layout their content in parallel story threads. The flag editing system creates a framework for developing flexible story structures. To allow authors to understand these structures, several simulation tools were created. These simulators allow the author to specify paths through space and time and view the resulting story playback. Additionally, story path generation tools provide a means for authors to see all the possible ways a story could unfold.

Since authors are working in a new medium, their tools should ideally help them to better understand the platform and allow them to target their stories to it. That is why M-Studio was designed to also be a pre-production tool, allowing creators to build storyboards out of textual descriptions and images prior to shooting. By evaluating their proposed story structure early on, authors can identify issues with their plans before investing energy in the costly and time-consuming shooting process. Since mobile cinema is a new technology, authors may not understand what will and will not work on the mobile platform. M-Studio allows authors to create simple prototypes of their stories and understand how they will work in a mobile context from their desktop computers.

We also hoped to make the technical details of the client and server invisible to the story creator. To this end, we created an XML schema that could be automatically generated by M-Studio. This alleviates the burden of hand-scripting a story from the user. Authors do not have to learn a scripting language to create a complex story structure. Rather, they can rely on M-Studio's graphical interface to design and test a story, and then export the resulting story script and video directly to the server, without having to focus on any technical details of implementation. Advanced users who do want more fine control have the option of creating custom flag evaluators for the server and manually editing M-Studio's generated XML.

Evaluation of M-Studio has illustrated its usefulness for both planning and simulating stories. Without this kind of tool, it is difficult to understand the possibilities presented by each arrangement of clips and flags. We learned many important lessons working with the two M-Views productions, "Another Alice", and "Mind in Hand". The original version of "Another Alice" showed the importance of examining the relationship and densities of parallel story threads. Reworking "Another Alice" for the new server structure gave us an understanding of the complexities of designing a flag structure and the importance of simulation. When developing "Mind in Hand", M-Studio allowed for the creation of many different prototypes for flag structures, and the simulation tools allowed the authors to select the best one. However, the

examination of story threads cannot provide the full scope of the experience, making field simulation also necessary.

Emerging technologies like handheld devices and high bandwidth wireless networks will continue to revolutionize the way that we receive and experience information and content. However, for these new platforms to gain acceptance and widespread usage, they will need to provide a supply of compelling content. While these new ideas for cinematic experiences present an exciting potential, this potential will not be realized unless support is provided for authors venturing into these forms. Hopefully, tools like M-Studio can offer the necessary aid to encourage the examination of novel ideas in storytelling, thus opening new channels for delivery of information and content.

REFERENCES

1. Bruner, Jerome. Acts of Meaning. Cambridge: President and Fellows of Harvard College, 1990.

2. Pan, P., C. Chen, and G. Davenport. The Birth of "Another Alice". *Computers and Fun 4.* University of York, November 2001

3. Macromedia Director: http://www.macromedia.com/software/director/

4. Macromedia Authorware: http://www.macromedia.com/software/authorware/

5. Bailey, B.P., J.A. Konstan, and J.V. Carlis. DEMAIS: Designing Multimedia Applications with Interactive Storyboards. *Proceedings ACM Multimedia*, 2001.

6. Caloini, A., D. Taguchi, K. Yanoo, and E. Tanaka. Script-free Scenario Authoring in MediaDesc. *Proceedings ACM Multimedia*, 1998.

7. Brooks, K.M. Do Story Agents Use Rocking Chairs? The Theory and Implementation of One Model for Computational Narrative. Proceedings, ACM Multimedia, 1996.

8. Evans, R. LogBoy Meets FilterGirl: A Tool for Multivariant Movies. 1994.

9. Bove, V.M. and S. Agamanolis. Responsive Television. MIT Media Lab.

10. Bylund, M. and F. Espinoza. Using Quake III Arena to Simulate Sensors and Actuators when Evaluating and Testing Mobile Services. *In Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems*, 2001.

11. Abowd, G., C.G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5): 421-433, October 1997.

APPENDIX A: XML Schema

This appendix is intended to give more specific detail about the XML story scripts that M-Studio generates for the M-Views Server. This may be useful to people who wish to alter the way M-Studio produces XML or authors who want to write their own custom story scripts.

General Format

The M-Views XML story script can be viewed as a list of possible story events. Each story event would correspond to a clip in M-Studio. The format of the XML file is very simple. It starts with the XML header tag. Then, a tag corresponding to the name of the story (`storyName` in Figure A1) is inserted. This story tag will wrap all the story events.

Each story event has a unique ID, shown here as `ID1`, `ID2`, etc. This header tag also has several attributes that are used to generate the corresponding message for the M-Views client. The `Context` field refers to the `location` of the event, and should be a string representation for whatever location detection system (GPS, infrared, 802.11b triangulation, etc) is being targeted. The `Repeatable` field determines whether the event can be seen more than once, and can take a value of true or false. The `Story` field should refer back to `storyName`, the name of the overall story this event takes place in. The `Subject` field should be the title of the event. The server automatically fills in all the other fields when the event is sent, and should be left empty.

Within this event header, there are four major subheadings. The first, the `Requires` clause, consists of a list of flags that must evaluate to true in order for the clip to play. Each tag refers to a specific flag (or the null flag in the case of special flags) and provides an expression that will be evaluated with respect to the given value to determine the state of the flag. More detail about the structure of flag tags is given in the next section.

The second section is the `Message` clause, which contains information about what content to send to the viewer. The `MediaURL` tag should be a pointer to the media file associated with this event. It can be the local address of the media on the server, or a globally available URL. The content type of the media is not restricted, but it should correspond to whatever media players are available on the target client. The `Text` field specifies the text message that will accompany the media when the user receives the event in the client inbox.

The `Results` clause is the next section of the story event. It consists of a list of flags that are altered after this clip has been played. The tag refers to a specific flag that is updated, and gives an operation to perform on the selected flag with a certain value. Special operations are discussed in the next section.

The final section, `Heuristics`, is optional. If it is provided, the server will use heuristics to break ties when more than one clip is available to play at a given time. Heuristics are descriptive terms that provide some information about the content of the clip. Each heuristic tag corresponds to a category that describes something about what happens in a clip. Each tag also has a value, which describes how strongly the given category is represented in a clip. When breaking ties, the server will look at the current state of the heuristics for the story and see which

possible clip has values that best match this. When a clip plays, the categories it represents are incremented by the given value in the heuristic table.

```
<?xml version="1.0"?>
  <storyName>
      <ID1 Context="location" Date="" From="" Repeatable="false"
           Story="storyName" Subject="clipTitle" To="">
         <Requires>
            <variableName expr="evaluator" value="number"/>
            <OR>
               <_ expr="?S" value="clip1ID"/>
               <_ expr="?S" value="clip2ID"/>
            </OR>

            ...

         </Requires>
         <Message>
            <MediaURL>
               storyName/filename.mov
            </MediaURL>
            <Text>
               Text message to be displayed along with video.
            </Text>
         </Message>
         <Results>
            <variableName oper="operation" value="number"/>
            <listVariable oper="$+" value="string"/>

            ...

         </Results>
         <Heuristics>
            <categoryName value="number"/>

            ...

         </Heuristics>
      </ID1>
      <ID2>
            ...
      </ID2>

      ...

  </storyName>
```

Figure A1: An Example XML story script

XML Flags

In XML, tags represent flags. The tag names correspond to the flag names. Each of these tags has two fields, the expression/operation field, which corresponds to the action taken on the flag, and the value field, which corresponds to the value used in the requested comparison or operation. All flag tags must end with a closing backslash.

Numerical Flags

The most basic flags correspond to integer values. By default, basic mathematical operations and comparisons are supported. If any expressions are preceded by a ! (logical not operator), they are negated. The value field should always be an integer.

Table A1: Default Numerical Expressions/Operations

Expression	Description
Comparisons	
==	equals
!=	not equals
#LT	less than
#GT	greater than
#LTE	less than or equal to
#GTE	greater than or equal to
Operations	
+	increment
-	decrement
/	divide by
*	multiply by

List Flags

The server also supports flags that perform list operations. These lists can be tested to see if they contain a string or list of strings. They are updated by adding and removing values from the list. The value field for both comparisons and operations should be of the form of comma delimited values, e.g. `value="string1,string2,string3"`.

Table A2: Default List Expressions/Operations

Expression	Description
Comparisons	
$=	List contains
!$=	List does not contain
Operations	
$+	Add to list
$-	Remove from list

Clip Flags

Clip flags can be used in the requirement clause to test if certain clips have or have not been seen. Because they refer to the list of clips that have been seen, rather than using a specific tag name, they use the null tag (an underscore). The value of a clip flag should be a string corresponding to the tag name of a single event being referred to. Multiple clips can be

referenced by using <AND>/<OR> blocks (described later). The form of a clip flag tag should look something like this:

```
<_ expr="?S" value="ID1"/>
```

One special tag is used to express the fact that no clips can have been seen for this clip to play:

```
<_ expr="!?S" value="-1"/>
```

Table A3: Clip Flag Expressions

Expression	Description
?S	Clip must have been seen
!?S	Clip must not have been seen
?LS	Clip must have been the last clip seen
!?LS	Clip must not have been the last clip seen

Date/Time Flags

There are XML formats for both absolute and relative time flags. They can only be used as requirement flags. Again, since they do not refer to the actual flag table but other values, the null operator is used.

Absolute Date/Time Format:
The value field is of the form D<MM/DD1-MM/DD2>, T<HH:MM AA1-HH:MM AA2>, where the first item is the span of dates (in month/day format) and the second is the span of times. If only date or only time is used, only one item should be used, but it should still be preceded by a D or T. An example:

```
<_ expr="?DT" value="D01/01-02/02, T9:00 AM-5:00 PM"/>
```

Relative Time Format:
This flag expresses the minimum and maximum time that can have passed since a previous clip has been seen for this clip to play. Its value string should have three entries: the ID of the clip being triggered from, the minimum amount of time that has to pass for the clip to play, and the maximum amount of time that can have passed for the clip to play. All times are in minutes. If there is no limit to how much time can pass, then the maximum amount of time should be –1. This example shows that between 5 and 10 minutes should have passed since ID1 played to see this clip:

```
<_ expr="?RT" value="ID1,5,10>
```

<AND>/<OR> Tags

Sometimes, an author might want to make a clip trigger if any one of a group of flags is true. However, by default, all flags listed under the requirement clause must be true for a clip to play. By nesting flags inside <OR> tags, the author can specify that the requirement is met if any of these flags evaluate to true. <AND> tags can be nested inside <OR> tags to allow for more control over statements. For instance, the following expression states that either flag A or both

flags B and C must be true for this clip to play. Clearly, <AND>/<OR> tags are not legal within the results clause.

```
<OR>
    <A ...>
  <AND>
    <B...>
    <C...>
  </AND>
</OR>
```