Metafilters for the Digital Micromovie Orchestrator

by

John F. Shiple

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

at the Massachusetts Institute of Technology

May 1993

Author_____

Department of Electrical Engineering and Computer Science
May 17, 1993

Certified by_____

Professor Glorianna Davenport
Thesis Supervisor

Accepted by_____

Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

Metafilters for the Digital Micromovie Orchestrator

by
John F. Shiple

Submitted to the
Department of Electrical Engineering and Computer Science

May 17, 1993

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

Abstract

Metafilters bring the Digital Micromovie Orchestrator (DMO) into the next stage of its development. The DMO defines a method by which a director can bring together targeted material based on some story model which can be computationally defined and orchestrated using sketchy descriptions and filters. The DMO uses two modules, a logging module and a shot selection module, to create a personalized movie from a database of digital video clips. The logging module describes the content of the video clips using sketchy descriptions. The shot selection module determines on the fly which clip will be played next in the personalized movie. Templates provide the basic structure of a movie while filters determine which clips can be played next. The *Metafilters* concept offers a high level abstraction which defines a standard method to the creation and operation of filters in the DMO. In addition, *Metafilters* were conceived as the controlling structure of the DMO responsible for managing filters and templates and are the result of making the filter methods and processes more robust and functional. Extending the DMO through *Metafilters* makes it more powerful and independent by allowing the director more freedom and flexibility in creating stories at various levels of granularity. In implementing *Metafilters*, the director is given a powerful tool to script his own personalizeable movies. This paper defines the current *Metafilters* abstraction and describes and evaluates the issues surrounding the *Metafilters* abstraction.

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

Introduction

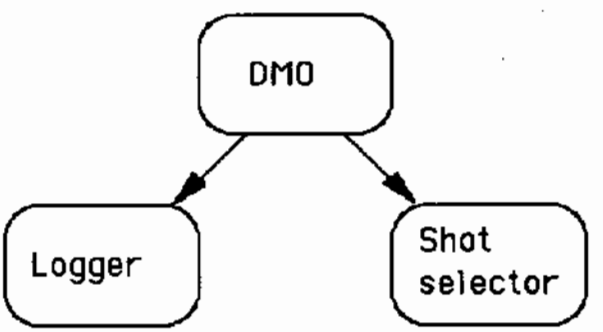
The Digital Micromovie Orchestrator (DMO) creates a personalized movie from a database of clips with descriptions attached, a defined template structure, and a set of filters and filtering processes. Templates give stories their basic structure. Clips represent the basic contents of the pieces in a template. Filters define how the DMO processes the clips and templates and also determine which clip will be played next.

The *Endless Conversation* was the first instance of a DMO. The movie had two characters, Dave and Tom. Basically, the story concept was one character would ask a question, the other character would answer it, and the character who asked the question would issue a rebuttal. The *Endless Conversation* used this simple template structure and fairly simple filters to structure the story that was presented. The filters gave the viewer control over such parameters as the pacing of the clips (whether they were long or short in length) and the rating of the clips (PG or R).

In the *Endless Conversation*, the director establishes a discrete pattern of content for the viewer in the form of question, answer, and response. The DMO is intended to be generalizable to any story where the director can establish this sort of discrete pattern of content. In addition, the *Endless Conversation* is an example of a movie that does not follow a single predetermined linear ordering. The DMO becomes useful when a single linear ordering of predetermined selections is not desirable. Examples of a predetermined single linear ordering are today's motion pictures and

most television shows. An example of movies that do not have to be linear include training videos, which could react to the individual and vary the instructional training accordingly. Another example is reviewable movies, which could be displayed differently (in a shorter time period for example) each time they are viewed so the viewer could either get more information out of them or quickly review old material. It is therefore useful to develop a personalizable story concept to deal with situations where a single linear ordering of predetermined selections is not advisable.

The basic structure of the DMO involves two modules, the logger module and the shot selection module:



Basic structure of the DMO

The two modules work together to bring about a personalizable movie that falls within the limits of the director's parameters. The director determines the parameters during the creation of the movie, and this is how the movie is made personalizable.

The logging module allows the director to describe the content of the video clips. By giving each video clip a "sketchy" description (described later), the director gives the DMO the ability to make the right decisions as to which clip should be played next. A major step in the construction of a movie is the annotation of the video clips. The importance of the logger to the DMO can not be underestimated.

Through filters and templates that the director creates, the shot selection module controls the flow of the story and determines what clip will be played next. Shot selection is determined at run time, and this is how the system is continuously able to incorporate to the viewer's choice of parameters. The director creates templates to give the story it's basic structure. Filters are created by the director specifically for the templates and to parameterize the clips in order to make the story more personalizable.

Aside from templates, a filter is the basic means for determining which clip will be played next. A filter provides constraints to the shot selection module for the overall clip list. The shot selection module uses these constraints to filter through the list of all the video clips. A smaller list of video clips that can then be passed through another filter is the result. For example, the pacing filter lets only clips that are short in length pass through if the pacing filter is set to filter short clips. The director uses filters to give the viewer control over the parameters of a movie.

Metafilters were conceived as the controlling structure of the DMO responsible for managing filters and templates and as providing a common framework in which filters may be created and then used. Before *Metafilters*, the DMO could not update its filters or

templates, add new filters or templates, or do any other such tasks that would modify the filters, templates or filtering process to suit the DMO's needs. *Metafilters* solve this problem by providing methods for adding and removing filters, updating the filtering process, and modifying the filters and templates.

Previously, filters and classes were not standardized and could not be shared among different DMO's. The continuity filter used in the *Endless Conversation* does not have the same form as the continuity filter designed for the *Map Guides* (these two instances of the DMO are described later) even though *Map Guides* uses the same concept of a continuity filter. Furthermore, templates were defined differently between different instances of DMO's. The *Endless Conversations* defines templates using macros:

```
(defmacro create-dialogue-template (actor1 actor2)
        `(list
                '((character ,actor1) (clip-type question))
                '((character ,actor2) (clip-type statement))
                '((character ,actor1) (clip-type response))
                '((character ,actor2) (clip-type question))
                '((character ,actor1) (clip-type statement))
                '((character ,actor2) (clip-type response))))
```

According to *CyberCritics*, templates are defined by functions:

```
(defun create-critic-template ()
        '(((critic-name woody) (clip-type movie-intro) (movie princess))
                ((clip-type movie-clip) (movie princess))
                ((critic-name woody) (clip-type opinion-query))
                ((critic-name cyrus) (clip-type opinion))
                ((critic-name woody) (clip-type opinion-rebuttal))))
```

To solve this problem, *Metafilters* provide class systems to solidify filters and templates as a classes, enable inheritance in designing filters, and simplify the design of filters.

A to B is a new instance of the DMO that exhibits the power and functionality that *Metafilters* bring to the DMO. The basic story line of *A to B* is as follows. A main character travels from one place to another, and at each place he encounters another character that he knows (a friend, significant other, etc.) and has a conversation with the other character. There are two templates necessary for this story line, a motion template and an encounter template. *Metafilters* must swap in the appropriate template when necessary. The motion template handles the main character's motion from place to place while the encounter template handles the interaction between the main character and the other characters. The details of the motion and encounter templates are described in appendix B.

To make *A to B* personalizable, several filters are necessary. An emotional filter controls the emotional state of the characters in the story. The viewer can change the emotional atmosphere of the movie by changing the emotional state of the characters. A "more" filter lets the viewer stay in the motion template longer, allowing the viewer to possibly watch the scenery or just increase the time it takes the main character in getting from one place to another. However, not all filters are designed just to make the experience personalized. Continuity filters are included to make sure the characters are talking about the same thing in the encounter template and to keep the means of transportation consistent in the motion template. With these filters and templates, *A to B* demonstrates the functional capabilities of *Metafilters* and the power *Metafilters* give to the director.

History

The DMO was created to study the ways in which a director can make a personalized movie by giving the viewer a certain degree of interactivity. Each instance of the DMO creates a personalized movie through the viewer's decisions as to what the parameters should be. By giving the viewer a certain degree of interactivity, the DMO is able to make a personalized movie for the viewer.

As stated previously, the *Endless Conversation* was the first instance of the DMO. The *Endless Conversation* was fairly limited because it had an immutable set of filters. The director had to know beforehand all the filters he was going to use and could not easily add new filters to the instance of the DMO. In addition, the template structure was fairly simple, however it was the first instance of the DMO ever created.

After the *Endless Conversation*, subsequent DMO's attempted to increase the complexity of the template structure and introduce more complex filter structures. *CyberCritics* was an instance of a DMO that had two critics discussing various films, much in the same way that Siskel and Ebert critique films. *CyberCritics* was more complex because the template structure was more complicated--it was not just a simple question, answer, response template. In addition, not only did *CyberCritics* have to keep track of what the critics were talking about, it had to keep track of where the critics were looking (so they could look at each other and at the clips that were playing), and which film previews had already been shown. However, there was no standard among respective DMO's, so one

couldn't easily take code from one DMO and use it in another. For example, the director of *CyberCritics* could not take the continuity filter that the director of the *Endless Conversation* because each DMO used different representations for the filter structures.

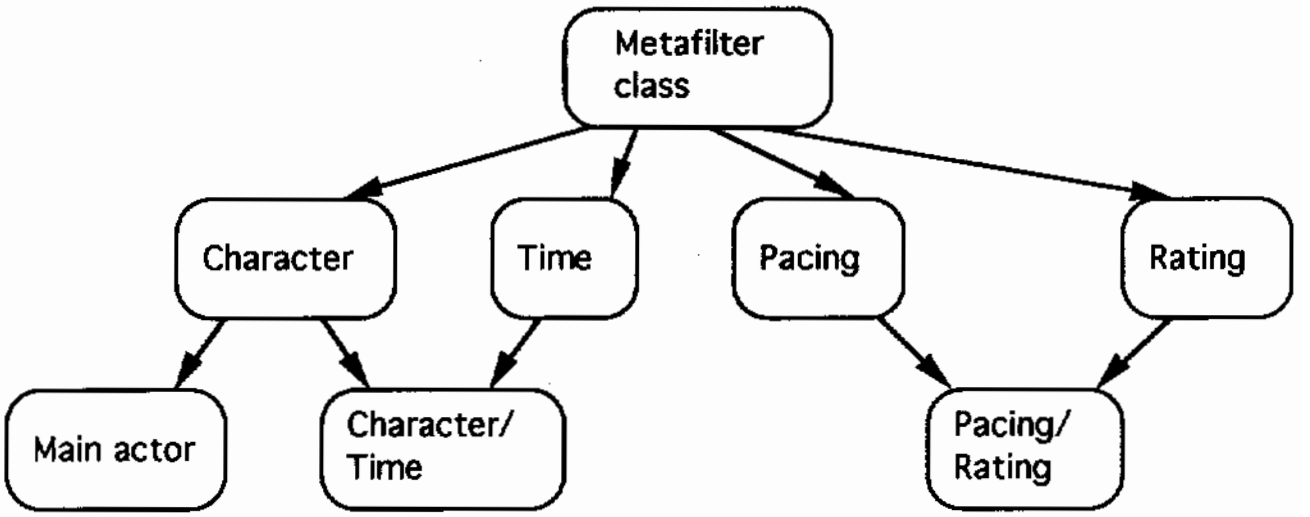
In August 1992, the template system was overhauled for the *Map Guides* DMO and made object oriented. *Map Guides* gives the viewer a tour of the South End/Back Bay of Boston. In *Map Guides*, the viewer has several possible tour guides from which to choose, and each tour guide offers a different perspective of the South End/Back Bay. For example, one guide gives a historical view of the area while another guide gives a tour of the various galleries along Newbury Street. At certain points in the tour, the viewer and guide meet other guides, and the viewer has the option of either staying with his original guide or going along with the other guide. *Map Guides* was made using two templates: a tour guide template and an intersecting guides template. The tour guide template takes care of the tour aspect of the DMO while the intersecting guides template handles the interaction between two guides when they met. Because *Map Guides* has two templates, the DMO must be able to swap in the appropriate template at the right time. An ad template was created to inject advertisements into the tour at random intervals.

The process of making *Map Guides* brought the DMO together somewhat, yet there was not a standard for the filtering system--which was a major part of the DMO. There grew a perceived need for some sort of device or system that would manage the filters, filtering processes, and templates. *Metafilters* arose from this need for such a system.

Metafilters

Metafilters are the controlling structure of the DMO responsible for managing filters. *Metafilters* deal with, on the basic level, filtering in the DMO, but also do much more. *Metafilters* are responsible for swapping filters in and out, adding or removing filters, filtering filters, and more, thus making the control structure of the DMO more fluid and flexible. In addition, *Metafilters* provide a layered filter structure with which the shot selection module may work.

Metafilters come from a redesigning of the filter processes of the original DMO to incorporate the swapping, adding, and removal of filter modules. *Metafilters* are designed with an object oriented approach in mind. The actual filter modules have a class structure with inheritance and mixin capabilities--much like the template structure now has. As an example of inheritance, one could create a pacing filter and a rating filter, and then simply create a pacing-rating filter by making a filter class with the pacing and ratings filters as parents. Here's a diagram of a possible *Metafilters* structure:



As before, filters can be created for specific templates, such as the pacing-rating filter in the *Endless Conversation* or for generic cases, such as the continuity filter of the *Endless Conversation* and the *A to B* DMO. *Metafilters* are flexible, simple and easy to work with in the DMO.

Implementation

Implementation of *Metafilters* actually began with the overhaul of the template system back in August, 1992. The DMO is written in Macintosh Common Lisp™ and makes use of Apple's digital video technology Quicktime™ to present the clips. Because the DMO is written in Common Lisp, the template structure was redesigned around a class system to make it object-oriented. Each template has a slot, template-skeleton, that describes its basic structure:

```
(defclass template ()
        ((template-skeleton :initform nil :initarg nil)))
```

A templates can also define other slots that may be important to itself. The dialogue-template used in the *Endless Conversation* defines actor1 and actor2 slots to keep track of the actors when it creates the template-skeleton for an instance of the dialogue-template. In addition, each template has an *update-template* method to specify how the template should update itself when either the template runs out or something else tells it to update itself (usually a filter). Examples of template classes are provided in appendix A. This design change suggested that the existing filters could be improved upon as well by making them a class too. So, the first step towards implementing *Metafilters* was to make the filters object-oriented.

Making the filters object-oriented involved the creation of a metafilter class:

```
(defclass metafilter ()
        ((constraints-function :initform nil :initarg nil)))
```

The constraints-function slot contains the function that returns the constraints specific to the filter. The filtering process of the DMO then uses the constraints that are returned by the constraints-function to filter the clip list. The pacing filter used by the *Endless Conversation* represents an example of how to code a simple metafilter:

```
(defclass pacing-filter (metafilter)
        ())

(defmethod initialize-instance :after ((pf pacing-filter) &key)
        (with-slots (constraints-function) pf
                (setf constraints-function
                        #'(lambda (&key dialogue-transcript)
                                (declare (ignore dialogue-transcript))
                                (if (eq *pacing* 'slow)
```

```
(list (list 'not 'pacing 'fast))
(list (list 'not 'pacing 'slot)))))))
```

Creating an instance of a pacing-filter defines the constraints-function for the pacing-filter and gives the viewer control of this filter through the parameter *pacing*. The parameter *pacing* tells the DMO to filter out clips according to how long they are.

The next step to defining the *Metafilters* class was to define a method to get the constraints from the filter:

```
(defmethod constraints ((m metafilter))
        (apply (slot-value m 'constraints-function)
                (list :dialogue-transcript *story-transcript*)))
```

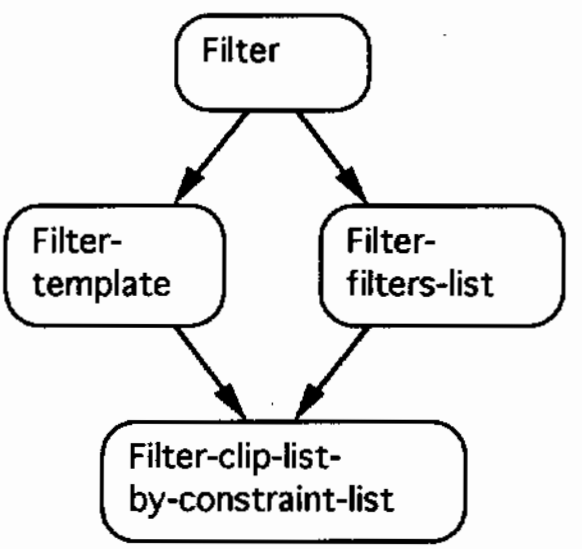
The constraints method applies the constraints-function to a transcript of the story that has already played out. Most filters don't do this, but some filters, such as the continuity filter, make use of the transcript to filter clips based on the clip that just played.

After filters were made into a class, the filtering processes had to be rewritten to incorporate the changes that were made. The DMO has to be able to filter the clip list in two ways, according to the current point in the template and the current filters that are being used. Three functions were written to handle this.

The first function takes the list of all the clips and produces a final clip list of all the possible clips that can play according to the template and the filters. Of these remaining clips, it randomly chooses one of them and returns that clip. The first function is the most general of the three functions and so is simply called *filter*. *Filter* controls the other two functions and uses them to produce the final clip list. The second function filters the clip list according to

the template and is therefore called *filter-template*. *Filter-template* takes a clip list and filters it according to the template's constraints, which is basically the current point in the template. The third function takes the constraints from the filter objects and filters the clip list according to the current filters. The DMO stores the filters in a list through which the function iterates and so the third function is called *filter-filters-list*.

The basic filtering process that both *filter-template* and *filter-filters-list* use is called *filter-clip-list-by-constraint-list*. Here's a module dependency diagram that illustrates the hierarchy of the filtering functions:



Filter-clip-list-by-constraint-list takes a constraints list, iterates through all the clips, and gets rid of clips that don't fit the constraints. If no clips fit the constraints, then no clips are removed and the original clip list is returned. *Filter-template* uses the current point in the template as the constraints list and only calls *filter-clip-list-by-constraint-list* once. *Filter-filters-list*

uses the constraints returned by a filter's constraints-function and calls *filter-clip-list-by-constraint-list* repeatedly because it is an iterative process. The filtering work is actually done by *filter-clip-list-by-constraint-list.*

After the filtering functions were designed, the next step was to overhaul the updating of the templates in order for *Metafilters* to be able to manipulate them. Functions to advance the template and refresh the current template were written to give *Metafilters* the desired control. *Advance-template* is called each time a clip finishes to make sure that the DMO is always at the correct point in the template. Whenever a template is finished, some filter calls for a new template to replace a template, or some other event that calls for updating a template occurs, *refresh-current-template* uses the template's *update-template* method to make sure the template is up to date.

When all the templates and filters had been written to incorporate the concepts of *Metafilters*, the story driver was updated to use the new functions. When this was done, the DMO was finally made object-oriented by creating a DMO class to finalize the implementation of *Metafilters.* Methods to start a story, kill a story, and place clips in their correct position on the screen were written to make playing a personalizable movie clear and easy to use. Other methods to add, remove, and swap filters were created to enable the DMO to do just that--add, remove, and swap filters.

At this point the DMO implements the ideas that *Metafilters* embodies. The DMO now has functions and methods that give it control over the filters and templates that it uses. The DMO can

update its filters and templates, add new filters or templates, or do any other such tasks that modify the filters, templates or filtering processes. The DMO is standardized. Different instances of a DMO can swap filters and templates with no problems (as long as the clips are logged correctly for each DMO).

As an exercise, *Map Guides* was reimplemented using *Metafilters*. This was done to show that by replacing the story driver and filtering functions and rewriting the templates and filters, DMO's without *Metafilters* could be easily changed to use *Metafilters*. The filters were rewritten to use the metafilter class and appropriate methods for each filter were written. In addition, methods for updating the existing template class were written to bring templates in line with the *Metafilters* concept. The story driver was removed and the new driver was installed. Finally, a *Map Guides* DMO class was created and methods for it were written to bring *Map Guides* up to date. The conversion was successful and showed how easy it was to reimplement older DMO's to incorporate *Metafilters*.

Creating a Personalized Movie

To create a personalized movie, the director has to go through a process that involves several steps--filtering, scripting, shooting, and logging. This section describes each step in detail and shows how the process applies to the *A to B* DMO.

Filtering

The first step in creating a personalizable movie is to create a basic template and filter structure for the movie. The director designs templates to give a rough outline as to what the story will be. As an example, in the *Endless Conversation*, the template is question, answer, and response. In *Map Guides*, the template structure is a little more complex with two templates, the intersecting-guides template and the tour-guide template, that are swapped in and out by the DMO as needed. In creating *A to B*, two templates were created: a motion template to handle when the main actor is traveling from person to person and an encounter template to handle the interaction of the main actor with the other characters he meets. The DMO manages the two templates to give the effect of the main actor moving from place to place meeting the other characters in the story.

After the director has created a basic template structure, he creates the filters that will be needed by the movie. With *Metafilters*, the director can use old filters, generate new filters, or easily mix and match new and old filters to come up with the desired mix of filters. *Metafilters* are what will make the DMO personalizable by enabling the director to give the viewer some control over the parameters of the movie. The *Endless Conversation* provides specific pacing and rating filters to give the viewer some control over the speed at which the clips are played and the content of the material presented (there are PG and R ratings for clips). *CyberCritics* makes use of a continuity filter to make sure the critics are talking about the same movie. The continuity filter that *CyberCritics* uses exemplifies the *Metafilters* concept of reusing

filters because it is the same continuity filter that the *Endless Conversation* uses. It is interesting to note that if the director has a good idea of what he wants to do, he can generate the templates and filters in less than an hour. *A to B* also uses this same continuity filter to keep track of the topic of conversation in the encounter template as well as the means of transportation in the motion filter. In addition, *A to B* has other filters--an emotional filter that can be thrown into the DMO to give the actors feelings and a "more" filter that lengthens the amount of time spent in the motion template.

Scripting

When the director has come up with the basic structure for the new movie, he must come up with a shot list that describes all the shots he will need for the DMO. It is important for the director to script several possibilities for each shot so that the personalized movie can be more personalizable. The director uses the templates and filters he designed in the previous steps to come up with the several possibilities for each shot and to write each shot's "sketchy" description. The director need only provide the relevant descriptions to the DMO as dictated by the templates and filters because this is the only information the shot selection module needs to personalize the movie. The story concept and the director's vision for the filters determine the overall parameters of the sketchy descriptions. Sketchy descriptions are also easier for the director to use because they use less information than "thick" descriptions that attempt to log all the semantic content of a clip.

Therefore, sketchy descriptions easier to enter into the computer. These sketchy shot descriptions are important later in the process when they are actually attached to their respected clips. The shot list for *A to B* (appendix C) lists all the shots that are needed by the DMO to orchestrate the movie and has the sketchy descriptions for each shot. Also, *A to B's* shot list is organized in such a manner that it is easy for the director to add shots on the spot or to change any shot if shooting conditions necessitate changes.

Shooting

After the shot list has been developed, the director films the required material. Just as in the filming of a traditional film, the director must shoot the same shot multiple times to allow the editor the opportunity to choose the best clip to be used, unless he has taken this into account in his shot list. Thus, the more variations on a given shot the director has, the more personalized the movie will be because the DMO will have more clips from which to choose. The variations in a given shot must take into account the interaction filter settings (the parameters of the personalized movie). For example, if the director wants to make use of a pacing filter, he needs to shoot clips that have a slow pacing and a fast pacing. In addition, some shots must be as generic as possible so that they may be used in multiple places, thus cutting down on the number of specific clips that have to be shot. The greeting clips, such as "Hi!" or "How are you today?", in the *Map Guides* are generic enough that only a few are needed to handle all the possible sequences in the movie. The shot list for *A to B* includes a large

number of generic shots that simplified the filming of each shot. In the motion template, generic shots of the main character entering a car or putting on in-line skates eliminate the need for the director to shoot the character doing these activities at each location through which the character travels. This greatly reduces the number of clips that need to be shot as the DMO can use these generic clips multiple times and still convey a sense of what is going on. The shots that had more descriptions attached to them required greater care in shooting because the constraints were tighter than in shooting the generic clips. Shooting is a very time consuming process, especially when compared to the time it takes to create the templates and filters for the DMO.

Logging

After the footage is digitized, the director must break it up into clips using the logger module to attach the sketchy descriptions from the shot list to the digitized clips so that the DMO can choose the correct clips to play. Again, like the shooting stage, the director's actions are remarkably similar to what a traditional film director does. The logging module places the clips into a large database that the shot selection module can use to play the movie. Logging is an extremely important step, for if the clips have no descriptions, the DMO will not be able to figure out the content of the clips, and the DMO will not function properly. Logging *A to B* was made easier because of the organization of the shot list and by the specialization of the logger to facilitate logging. Ryan Evans, a graduate student in the Interactive Cinema at the Media Lab, and

James Seo, a UROPer in the same lab, designed the logger that was used. James was particularly helpful in specializing the logger to make logging *A to B* easier.

As soon as the director is done logging, the movie is ready to be orchestrated by the shot selection module. The templates designed in the first step provide a basic structure for the story while the filters created personalize the story according to the viewer's decisions. The sketchy descriptions enable the shot selection module to correctly choose the next clip to be played according to the constraints given it by the templates and filters.

Future Directions

Metafilters bring the DMO into what is possibly the last stage of the development of the DMO as it is defined in "Orchestrating Digital Micromovies", a paper by Professor Glorianna Davenport, Ryan Evans, and Mark Halliday. From this point, there are several paths that DMO "technology" may take. The first path involves utilizing the DMO in new and innovative ways to explore cinematic realms, multi-threaded narratives, and stories told from different points of view. It would be very easy for someone to design a narrative with multiple points of view. Changing points of view would be as simple as having *Metafilters* swap in one character's filters for another character's filters. Multi-threaded narratives could be explored in a similar manner.

A second path for the DMO involves using the DMO as a tool for other applications. Making the DMO part of a larger whole instead of

being an application by itself has tremendous potential. Using the DMO as part of some program that gathers news and then presents it according to a viewer's parameters is an excellent example of a practical application for the DMO. The viewer could create filters for the kind of stories he wants to view. The DMO would use these filters to sort through the news for the day and present a sampling of the possible news stories. The DMO also lends itself to use in the home. After creating a bunch of home movies, a person could log the movies and then create filters so that he could present them to others. Filters to keep the movies in chronological order or to only use movies with children in them could be designed.

Finally, the interface between the director and the filters he creates is a little weak--the director has to know how to program in Macintosh Common Lisp. Any system that would improve upon this interface would be a step in the right direction. There is a system that is being worked on right now that uses DMO "technology" as one of it's tools. Ryan Evans' project in the Interactive Cinema Laboratory at the Media Lab has tools for designing filters and templates, logging clips, and uses the DMO as a playback mechanism. The future possibilities for the DMO as part of a larger project are very bright.

For the immediate future, there is a very clear goal--implement a new instance of the DMO using *Metafilters*. *Metafilters* were tested using the *Map Guides* DMO, however, *Map Guides* was not designed with *Metafilters* entirely in mind. To fully exploit the potential of *Metafilters*, a new DMO was designed--*A to B*. As stated and described previously in this paper, *A to B* is the new instance of

the DMO, however it has not yet been completed. The immediate goal for the future is to finish the implementation of *A to B*.

Conclusion

The DMO was designed to make a personalized movie. *Metafilters* are the next step in making the DMO personalizable, easily expandable, independent, portable, and truly flexible. As a result, *Metafilters* facilitate the director's ability to design personalized movies and increase the level of creativity the director can put into his work. It is hopeful that the new DMO, *A to B*, is representative of the new power and capabilities that *Metafilters* have given the DMO.

Appendix A:

Sample Template Classes

```
(defclass ad-template (template)
        ())

(defmethod initialize-instance :after ((at ad-template) &key)
        (with-slots (template-skeleton) at
                (setf template-skeleton (list (list 'clip-type 'ad)))))

(defmethod update-template ((at ad-template))
        (let ((next-template (choose-rand-item (list *previous-template* *ad-template*))))
                (if (equal *rejected-guide* 'noone)
                        (progn
                                (setf *rejected-guide* nil)
                                (setf *current-template* *previous-template*))
                        (progn
                                (if (equal next-template *ad-template*)
                                        (progn
                                                (unless (equal *current-template* *ad-template*)
                                                        (setf *temp-skip-intro-talks* *skip-intro-talks*))
                                                (setf *skip-intro-talks* nil))
                                        (setf *skip-intro-talks* *temp-skip-intro-talks*))
                                (setf *current-template* next-template)))))


(defclass dialogue-template (template)
        ((actor1 :initform nil :initarg actor1)
         (actor2 :initform nil :initarg actor2)))

(defmethod initialize-instance :after ((dt dialogue-template) &key actor-1 actor-2)
        (with-slots (template-skeleton actor1 actor2) dt
                (setf actor1 actor-1)
                (setf actor2 actor-2)
                (setf template-skeleton
                        (list
                                (list (list 'character actor1) (list 'clip-type 'question))
                                (list (list 'character actor2) (list 'clip-type 'statement))
                                (list (list 'character actor1) (list 'clip-type 'response))
                                (list (list 'character actor2) (list 'clip-type 'question))
                                (list (list 'character actor1) (list 'clip-type 'statement))
                                (list (list 'character actor2) (list 'clip-type 'response))))))

(defmethod update-template ((dt dialogue-template))
        (with-slots (template-skeleton) dt
                (setf template-skeleton
                        (list
                                (list (list 'character actor1) (list 'clip-type 'question))
                                (list (list 'character actor2) (list 'clip-type 'statement))
                                (list (list 'character actor1) (list 'clip-type 'response))
                                (list (list 'character actor2) (list 'clip-type 'question))
                                (list (list 'character actor1) (list 'clip-type 'statement))
                                (list (list 'character actor2) (list 'clip-type 'response))))))


(defclass intersecting-guides-template (template)
        ((guide1 :initform nil :initarg guide1)
         (guide2 :initform nil :initarg guide2)))

(defmethod initialize-instance :after ((igt intersecting-guides-template) &key guide-1 guide-2)
        (with-slots (template-skeleton guide1 guide2) igt
```

```
        (setf guide1 guide-1)
        (setf guide2 guide-2)
        (setf template-skeleton
                (list
                        (list (list 'character guide1) (list 'clip-type 'greeting))
                        (list (list 'character guide2) (list 'clip-type 'greeting))
                        (list (list 'character guide1) (list 'clip-type 'activity-query))
                        (list (list 'character guide2) (list 'clip-type 'activity-response))
                        (list (list 'character guide2) (list 'clip-type 'invitation))
                        (list (list 'character guide1) (list 'clip-type 'goodbye))
                        (list (list 'character guide1) (list 'clip-type 'goodbye))))))

(defmethod update-template ((igt intersecting-guides-template))
        (with-slots (template-skeleton guide1 guide2) igt
                (setf template-skeleton
                        (list
                                (list (list 'character guide1) (list 'clip-type 'greeting))
                                (list (list 'character guide2) (list 'clip-type 'greeting))
                                (list (list 'character guide1) (list 'clip-type 'activity-query))
                                (list (list 'character guide2) (list 'clip-type 'activity-response))
                                (list (list 'character guide2) (list 'clip-type 'invitation))
                                (list (list 'character guide1) (list 'clip-type 'goodbye))
                                (list (list 'character guide1) (list 'clip-type 'goodbye))))))


(defclass time-template (template)
        ((beginning-length :initform nil :initarg nil)
         (middle-length :initform nil :initarg nil)
         (end-length :initform nil :initarg nil)
         (beginning-scale :initform (list 0 3) :initarg nil)
         (middle-scale :initform (list 3 8) :initarg nil)
         (end-scale :initform (list 8 10) :initarg nil)))

(defmethod initialize-instance :after ((t-t time-template) &key allotted-time)
        (setf (slot-value t-t 'beginning-length)
                (list (* allotted-time (/ (start (slot-value t-t 'beginning-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'beginning-scale)) 10))))
        (setf (slot-value t-t 'middle-length)
                (list (* allotted-time (/ (start (slot-value t-t 'middle-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'middle-scale)) 10))))
        (setf (slot-value t-t 'end-length)
                (list (* allotted-time (/ (start (slot-value t-t 'end-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'end-scale)) 10)))))

(defmethod update-template ((t-t time-template))
        (setf (slot-value t-t 'beginning-length)
                (list (* allotted-time (/ (start (slot-value t-t 'beginning-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'beginning-scale)) 10))))
        (setf (slot-value t-t 'middle-length)
                (list (* allotted-time (/ (start (slot-value t-t 'middle-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'middle-scale)) 10))))
        (setf (slot-value t-t 'end-length)
                (list (* allotted-time (/ (start (slot-value t-t 'end-scale)) 10))
                        (* allotted-time (/ (end (slot-value t-t 'end-scale)) 10)))))


(defclass tour-guide-template (template)
        ((guide-name :initform nil :initarg nil)))
```

```
(defmethod initialize-instance :after ((tgt tour-guide-template &key guide)
        (with-slots (template-skeleton guide-name)
                (setf guide-name guide)
                (setf template-skeleton
                        (list
                                (list (list 'character guide) (list 'clip-type 'intro-talk))
                                (list (list 'character guide) (list 'clip-type 'place-talk))
                                (list (list 'character guide) (list 'clip-type 'moveon-talk))))))

(defmethod update-template ((tgt tour-guide-template))
        (let ((next-template (choose-rand-item (list *previous-template* *ad-template*))))
                (if (equal *rejected-guide* 'noone)
                        (progn
                                (setf *rejected-guide* nil)
                                (setf *current-template* *previous-template*))
                        (progn
                                (if (equal next-template *ad-template*)
                                        (progn
                                                (unless (equal *current-template* *ad-template*)
                                                        (setf *temp-skip-intro-talks* *skip-intro-talks*))
                                                (setf *skip-intro-talks* nil))
                                        (setf *skip-intro-talks* *temp-skip-intro-talks*))
                                (setf *current-template* next-template)))))
```

Appendix B:

A to B Motion and Encounter Templates

***A to B* Motion template:**

-Approach means of transportation
-Start means of transportation
(get in car, get on skateboard, put on skates...)
-Motion
(may go through this part multiple times)
-Stop means of transportation
-Leave means of transportation

Location	Means
Esplanade	bike, walk
Med Center pavilion	walk, rollerblades
Newbury street	bike, car, skateboard
Harvard bridge	rollerblades, skateboard, car

Shots	#/means	#means	total
Showing means	2	5	10
Start means	2	5	10
Go	5-6	5*2	50-60
Stopping means	2	5	10
Exit means	2	5	10
		Total:	90-100

Basic filters:
Transportation consistent filter
"More" filter

Notes: The motion clips will be site specific shots. The means clips will be generic shot.

***A to B* Encounter template:**

-Intros
-QAR (Question/Answer/Rebuttal)
-Byes

Characters:	Topics:
Angry Youth	#1
Technoboy	#2
Girlfriend	#3
Grad Student	#4

Shots: characters	#/character	#characters	total
Intros	4	4	16
QAR's	9	4	36
Byes	4	4	16
		Total	68

Shots: main char.	#
Intros	6
QAR's	16
Byes	6
Total	28

Total shots: 96

Topics:

Evening plans--What are you doing tonight?
borrowing money--Can I borrow some money?
school--How's school going?
Boston--Are there any cool places to see in Boston?

Filters:

Emotional filter
Topic filter

Notes: The intros, QAR's, and byes will try to be dynamic in the sense that it will randomly choose which person goes first.

Appendix C:

A to B Shot List

A to B Motion Shot List

A-to-B template			
Template piece	**Means**	**Shot**	**Location**
Showing means			
	bike	approaching bike (left/right?)	generic
		getting on bike	generic
	walk	close-up putting on shoes	generic
		close-up tying shoes	generic
	car	approaching car (like bike?)	generic
		entering car	generic
	rollerblades	close-up putting on skates	generic
		close-up lacing skates	generic
	skateboard	close-up hand spinning wheels	generic
		close-up skateboard	generic
Starting means			
	bike	2 generic shots starting to peddle bike	generic
	walk	starting to walk from sitting	generic
		starting to walk from standing	generic
	car	close-up key turning in ignition on	generic
		close-up hand shifting car into drive	generic
	rollerblades	starting to skate from sitting	generic
		starting to skate from standing	generic
	skateboard	2 generic shots starting to skate	generic
Motion w/ means			
	bike	5 shots of skating through various parts	Esplanade
		5 shots of skating past various shops	Newbury St.
	walk	5 shots of walking through various parts	Esplanade
		5 shots at different parts of pavilion	Med. Center P.
	car	5 shots of car traversing different parts of bridge	Harvard Br.
		5 shots of car through various intersections	Newbury St.
	rollerblades	5 shots skating around pavilion (tricks too)	Med Center P.

A to B Motion Shot List

		5 shots at various parts of bridge	Harvard Br.
	skateboard	5 shots past various shops	Newbury St.
		5 shots traversing bridge	Harvard Br.
Stopping means			
	bike	2 generic shots stopping bike	generic
	walk	stop sitting	generic
		stop standing	generic
	car	close-up key turning ignition off	generic
		close-up hand putting on parking break	generic
	rollerblades	stop sitting	generic
		stop standing	generic
	skateboard	2 generic shots stopping skating	generic
Exiting means			
	bike	2 generic shots of getting off bike	generic
	walk	close-up removing shoes (put on others--like Mr. Ro	generic
		close-up untying shoes	generic
	car	leaving car	generic
		walking away from car	generic
	rollerblades	close-up removing skates (put on shoes?)	generic
		close-up untying skates	generic
	skateboard	close-up hand picking up skateboard	generic
		getting off skateboard	generic

A to B Encounter Shot List

Encounter template				
Template piece	**Character**	**Topic**	**Topic/Dialogue**	**Emotion**
Intros				
	Main char.		Hi.	
			Hey there.	
			Hey, what's up?	
			Smile	
			Hi, I haven't seen you for a long time!	
			Wave	
	Angry Youth		Hey man, long time no see.	
			Glare	
			Sneer	
			Leave me alone.	
	Raverboy		Hey, how's it going?	
			Hey, nice to see you outside Cyberspace!	
			Hey dude, nice to see ya!	
			Hey there compadre!	
	Girlfriend		Hi there honey!	
			Hey there, I haven't seen you for a while!	
			Hello. Finally I get to spend some time with you...	
			Hello sailor!	
	Grad student		Hi, haven't see you in lab for a while.	
			Huh, oh hi.	
			Howdy!	
			What's up?	
Questions				
	Main char	Evening plans	What are you doing tonight?	Angry
		Evening plans	What's going on tonight?	Happy
		Evening plans	Are you doing anything tonight?	Sad
		Money	I want that money back that you borrowed from me.	Angry

A to B Encounter Shot List

		Money	You wouldn't have that money you owe me do you?	Happy
		Money	I really need some money...	Sad
		School	How was that test you took?	Angry
		School	How's your school going?	Happy
		School	What's up with your classes?	Sad
		Boston	Doing anything in Boston tonight?	Angry
		Boston	Want to go into Boston tonight?	Happy
		Boston	Anything cool going on in Boston?	Sad
	Angry Youth	Evening plans	What are you gonna do tonight?	Angry
		Money	Can I borrow a little cash man?	Happy
		School	How are you doing in your classes?	Sad
	Raverboy	Money	Could you give me back that $50 you borrowed?	Angry
		School	How's school going?	Happy
		Boston	What're some cool places to see in Boston?	Sad
	Girlfriend	Evening plans	Can I see tonight?	Sad
		School	Did you do well on your test?	Angry
		Boston	Want to go into Boston later?	Happy
	Grad student	Evening plans	What are your plans for tonight?	Happy
		Money	Can I borrow 5 bucks? Just 5?	Sad
		Boston	Are you going into Boston tonite?	Angry
Answers				
	Main char.	Evening plans	2 or 3 answers similiar to the following answers (for	
		Money	the other characters) with the same topic	
		School	and emotional states.	
		Boston		
	Angry Youth	Evening plans	Why should I tell you?	Angry
		Evening plans	Get away from me and leave me alone.	Angry
		Evening plans	I'm gonna get wasted and then go clubbing.	Angry
		Money	Sure man.	Happy
		Money	No problemo I got it right here.	Happy
		Money	Uh, sure, I guess so.	Happy
		School	Not too good--I bombed a test today.	Sad

A to B Encounter Shot List

		School	I'm totally bogged down with work, I have no free time.	Sad
		School	I think I failed my test today.	Sad
	Raverboy	Money	What?!? I never borrowed any money from you!	Angry
		Money	I said I'd have it by next week, alright!!	Angry
		Money	Could you please ask me a few more times for your $?	Angry
		School	Pretty cool--I've learned how to jack into the Internet!	Happy
		School	Things are going great!	Happy
		School	This terms almost over!!!	Happy
		Boston	There aren't any good clubs here--they all close at 1 or 2.	Sad
		Boston	Not much--the police are really strict here.	Sad
		Boston	I don't know--I've not gone out for a couple weeks now.	Sad
	Girlfriend	Evening plans	I can't see you--I've got a test tomorrow. I gotta study.	Sad
		Evening plans	I feel pretty sick right now...	Sad
		Evening plans	I'm sorry, but I can't...	Sad
		School	I can't believe I studied all night for that...	Angry
		School	I hate that class--they're biased against me...	Angry
		School	I'm so mad at myself for failing that test.	Angry
		Boston	Yeah, that sounds like a lot of fun...	Happy
		Boston	Sure, I'd be psyched to go out tonight.	Happy
		Boston	Certainly!	Happy
	Grad student	Evening plans	I'm gonna meet some old friends of mine.	Happy
		Evening plans	I'm going to see that movie we talked about earlier.	Happy
		Evening plans	I'm going out of town!!!	Happy
		Money	Sorry, someone stole my wallet yesterday...	Sad
		Money	If I had any....	Sad
		Money	No, I gotta go grocery shopping later...	Sad
		Boston	No, I have to work late tonight.	Angry
		Boston	Me? I hate Boston--it's so crowded.	Angry
		Boston	If someone hadn't stolen my car, maybe I would...	Angry
Responses				
	Main char.	Evening plans	2 or 3 answers similiar to the following responses (for	
		Money	the other characters) with the same topic	

A to B Encounter Shot List

		School	and emotional states.	
		Boston		
	Angry Youth	Evening plans	Sorry I asked!	Angry
		Evening plans	Gee, that's cool.	Angry
		Evening plans	Aw, be quiet.	Angry
		Money	Thanks!	Happy
		Money	That's pretty cool!	Happy
		Money	Thanks for the cash!	Happy
		School	I'm sorry.	Sad
		School	That's too bad man.	Sad
		School	Don't worry, things'll work out.	Sad
	Raverboy	Money	You had better give me my money soon.	Angry
		Money	You'll be sorry if you don't pay me back.	Angry
		Money	What's up your butt?!?	Angry
		School	Excellent!	Happy
		School	Alright!	Happy
		School	That sounds really good.	Happy
		Boston	Oh, ok.	Sad
		Boston	Really? That sucks.	Sad
		Boston	I hope things pick up around here.	Sad
	Girlfriend	Evening plans	Oh, I wish I could see you.	Sad
		Evening plans	I really wish we could go out tonight.	Sad
		Evening plans	It's seems like we never get to see eachother anymore	Sad
		School	That sucks!	Angry
		School	That even makes me mad!	Angry
		School	That's pretty crummy.	Angry
		Boston	Sounds like a plan!	Happy
		Boston	Cool--we'll talk later about it.	Happy
		Boston	Alright!	Happy
	Grad student	Evening plans	That sounds great.	Happy
		Evening plans	Cool beans.	Happy
		Evening plans	Sounds fun!	Happy
		Money	Thanks anyways.	Sad

A to B Encounter Shot List

		Money	Ok, I'll ask someone else.	Sad
		Money	Darn, I guess I won't eat tonight then.	Sad
		Boston	Dang that sucks.	Angry
		Boston	It sucks to be you.	Angry
		Boston	That's nuts!	Angry
Byes				
	Main char.		See ya, wouldn't wanna be ya...	
			Bye.	
			Catch you later.	
			See ya.	
			Wave.	
			See you later.	
	Angry Youth		See ya hoser.	
			Catch ya later.	
			Grunt	
			Bye.	
	Raverboy		Bye now!	
			Rave on!	
			Live on, love on, rave on...	
			Catch ya on the flip side...	
	Girlfriend		See you later dearie!	
			Au revoir mon amour!	
			Bye Bye.	
			Smile	
	Grad student		See ya later.	
			Wave.	
			See you in lab.	
			Bye.	