

Virtual Director: Automating a Webcast

Erik Machnicki and Lawrence Rowe
Computer Science Division, EECS
University of California, Berkeley
Berkeley, CA 94720
(machnick@cs.berkeley.edu,
rowe@bmrc.berkeley.edu)

Abstract

This paper presents a system designed to automate the production of webcasts, the Virtual Director. It automates simple tasks such as control of recording equipment, stream broadcasting, and camera control. It also automates content decisions, such as which camera view to broadcast. Directors can specify the content decisions using an automation specification language. The Virtual Director also uses a question monitor service to automatically identify questions and move the cameras to show the audience member asking the question. We discuss the implementation of the Virtual Director and present the results of its use in the production of a university seminar series.

1. Introduction

With the growth of broadband networks, Internet broadcasting, also called webcasting, is becoming more and more popular¹. Examples of Internet webcasts are presidential debates, sporting events, technical conferences, and rock concerts. This increased popularity has driven demand for tools that make the production of high-quality webcasts simpler and more cost-effective. This project focuses on automating a webcast, specifically low-cost production of high-quality, multiple-stream lecture webcasts.

Lecture webcasting has several advantages. Students who cannot attend a lecture may choose to watch a live webcast. This includes students who might regularly attend, but for some reason, cannot attend on a particular day. It also includes students who, due to geographic constraints, are unable to attend the live lecture. A live lecture webcast allows students to watch the presentation at the same time as their peers, but more importantly, it allows remote viewers the opportunity to interact with the speaker, thereby providing a more valuable learning experience. A webcast can be recorded for future replay on demand, thereby benefitting viewers who are unable to watch the live webcast due to scheduling conflicts. A

¹ We use the terms “broadcast” and “webcast” interchangeably. The Virtual Director is used in the production of webcasts, but the concepts and ideas also apply to any form of broadcasting.

recent study of a lecture webcasting system at U.C. Berkeley showed that up to 75% of the students in some classes watch lecture webcasts, and 90% of the lectures were played on demand by students studying for examinations [Rowe 2001].

Although lecture webcasting has many advantages, it is also a complex and costly operation. The lecture hall or classroom, called the *studio classroom*, must be set up to capture audio and video (e.g. cameras, microphones, audio mixers, encoding machines, routing switchers, etc.). Once the production environment is set up, the production of the webcast itself requires a skilled director to produce the broadcast. We use the term “director” to refer to the person that manages the production of the webcast. This task involves different responsibilities than the direction of a traditional television program. These responsibilities include controlling pan/tilt cameras, controlling recording devices, and selecting which camera to broadcast (i.e., shot selection). A typical live television production may involve 5-10 people to operate equipment and produce the webcast. To reduce the number of people required to produce a webcast, we developed an application, called the Director’s Console (*dc*), which allows one person to produce a webcast [Yu 2001]. *dc* is a framework and user interface for controlling software processes (e.g., audio/video capture processes [Eriksson 1994], streaming media recording and playback systems [Schuett 1999], video effects systems [Wong 1998], transcoding processes, etc.) and audio/video equipment (e.g., routing switchers, pan/tilt cameras, etc.) used to produce a webcast. The system is extensible so new services and control interfaces can be added to the system.

dc requires manual entry of commands by a human director. Ideally, an organization would like to webcast many lectures, but the cost of employing a human director and production assistants to produce a webcast limits production. The Virtual Director (*vdc*) is designed to automate production of a lecture webcast. Automation includes shot selection, equipment operation, and software control. It is important to note that our goal is not to replace the human director – automation systems are unlikely to do as good a job as a human director – but rather to simplify webcast production so that one person can produce several webcasts simultaneously. The trade-off for cost reduction will be lower quality productions. The research question is how much quality is lost due to automation.

Traditional broadcasts are composed of one audio and one video stream. The director switches between different camera views or uses a special-effects system to compose graphics and video into one video stream (e.g., composition, chroma key, picture-in-picture,

etc.), but the broadcast has only one video stream. In contrast, a webcast may include multiple streams and optionally allow the viewer to select the streams he wants to watch. For example, many lecture webcasts produce two video streams: one stream is the speaker or audience and one is the presentation material (e.g., transparencies, slides, VCR, etc.) being projected on a large screen in the classroom. We will refer to these two streams as the *speaker* and *content* streams. This multiple-stream feature of webcasts mitigates the effects of reduced quality.

The goal of the Virtual Director is to produce high-quality webcasts. However, the “quality” of a webcast is not easily defined or measured. There are, however, some common techniques used by professional directors to produce broadcasts judged to be of higher quality. In the lecture environment, the following techniques and guidelines might be used: 1) Different camera views should be used to keep the broadcast interesting. Watching a close-up of a speaker will put most viewers to sleep, so the webcast should be switched between different camera views (e.g., audience camera, stage camera, speaker close-up). 2) When audience members ask questions, the webcast should show the person asking the question and switch back to show the speaker’s response. The details of these techniques, such as how often to switch camera views, are highly dependent on a given director’s style and on the type of event being broadcast. 3) Titling is used to inform viewers of the speaker’s name, the lecture topic, or the date.

Automating a broadcast is a difficult problem because of the subjective nature of quality. In general, there are no hard, well-defined rules. Human directors rely on heuristics and experience to make broadcast decisions. However, some aspects can be formally defined. For example, the idea that recording should be started at the beginning of a broadcast and stopped at the end is straight-forward and easily automated. Similarly, the rule that whatever content the speaker is showing to the audience should also be shown to the remote viewers is also well-defined and easily automated. Indeed, automation can be used to simplify the task of the director and remove the possibility of error in these well-behaved situations.

The current version of the Virtual Director automates camera switching and production of the webcast using two mechanisms: 1) a broadcast automation service and 2) a question monitor service. The broadcast automation service allows a director to define a webcast using an *automation specification language*. This specification defines what automation the Virtual Director should perform, based on a set of rules and constraints. An automation specification is defined by a set of if-then rules. When the value of a rule’s predicate changes to true, the action associated

with the rule is invoked. This mechanism can be used to provide automation of the form “when the close-up camera view of the speaker has been shown for 2 minutes, switch to the stage camera view.”

The question monitor service detects questions from the audience and switches the webcast to show the audience member asking the question. It monitors different microphones in the room, analyzes the audio to detect when a question is being asked, and signals the broadcast automation service when a question is detected. The Virtual Director itself consists of the broadcast automation service, the question monitor service, and a graphical user interface (GUI) that ties these two services together. Figure 1 shows a screenshot of the Virtual Director. *vdc* contains a frame for each service. The question monitor frame contains controls to enable/disable the service and to adjust audio level thresholds. The broadcast automation frame shows the status of the webcast and allows the director to enable/disable specific automation rules.

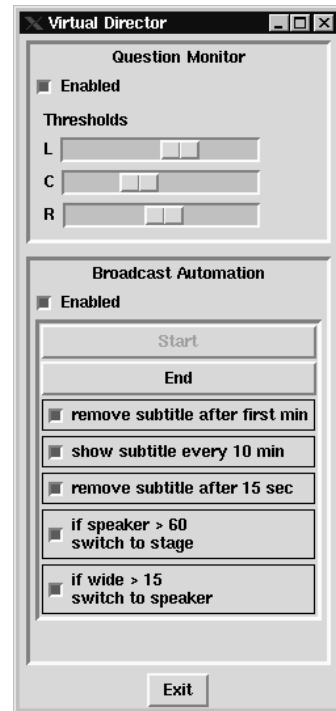


Figure 1 - Virtual Director Screen Shot

The current version of the Virtual Director does not do camera tracking of the speaker. There are several computer vision algorithms that could be employed in software [Grove 1998, Fayman 1998] to implement this feature. In addition, some cameras have a built-in tracking feature that perform quite well [ParkerVision].

This paper describes the design and implementation of the Virtual Director and

experiments performed to evaluate this technology. The remainder of the paper is organized as follows. Section 2 discusses other research related to the Virtual Director and lecture capture. Section 3 describes the studio classroom and production environment. Section 4 presents the automation specification language and gives an example of a simple broadcast specification. The broadcast automation service and the Virtual Director implementation of the automation specification language are described. Section 5 discusses the question monitor service. Section 6 presents the results of some experiments performed using the Virtual Director. Section 7 discusses insights gained while doing this research and presents directions for future work. Section 8 summarizes the research contributions.

2. Related Work

Many research groups and companies are developing systems to produce lecture webcasts. Some groups focus on the capture aspect, with varying levels of detail captured [Aowd 1999, Brotherton 1998, Virage]. Other systems focus on automated production of webcast material, either through post-production editing of the captured streams [Gleicher 2000, Mukhopadhyay 1999] or by making production decisions during the live broadcast [AutoAuditorium, Kameda 1999, Parwatar 2000, Karp 1993, Drucker 1995]. Still other groups provide automated control of the equipment in studio classroom to assist the speaker [Franklin 2000, Hirsh 1999, Franklin 1999].

The Virtual Director differs from these systems. It deals with producing multiple-stream broadcasts. The Virtual Director focuses on the automated production of live webcasts, but the automation can be shaped by individual directors. It is an extensible system, providing integration of independent services, which allows it to be used in many different production environments. The Virtual Director also automatically detects and reacts to audience questions. It is important to note that the Virtual Director is an integral part of a full-function lecture webcasting system that includes other production services (e.g., broadcast management [Wu 1999], recording and playback, special effects, floor control, etc.).

A paper published recently by Microsoft [Liu 2001] presents a system very similar to the Virtual Director. This system also provides automation of a broadcast, with full speaker tracking and audience question detection. In addition, they have also done a user study to evaluate the effectiveness of their system, which we have not done. The Virtual Director differs because it uses a broadcast specification to define the automation rules, instead of using fixed heuristics. Another difference is that the Microsoft system deals

only with camera management, while the Virtual Director fully automates a broadcast, including tasks such as control of recording equipment, special effects, etc.

3. Production Environment

This section describes the production environment and the Director's Console infrastructure on which the Virtual Director is based.

The production environment is a studio classroom at U.C. Berkeley. It is a small room (40 seats) with a presentation area at the front of the room. Figure 2 shows a floor plan of the studio classroom. The audio/video equipment includes a document camera, presentation PC, laptop computer port, and VCR connected to an LCD projector through a routing switcher. The room has two routing switchers: 1) a composite video and stereo audio switch and 2) an RGB switch. Video equipment is connected to the composite switcher, and the PC and laptop port are connected to the RGB switch. Both switches have connections to the LCD projector and a scan converter converts the RGB signal to a composite video signal so that RGB material can be captured for the webcast. Three cameras are permanently mounted in the room. A Canon VCC3 camera, with pan, tilt, and zoom capability, situated at the rear of the room is designated the *speaker camera*. Another Canon VCC3 is installed at the front of the room, facing the audience. It is called the *audience camera*. The third camera is installed in a rear corner of the room and provides a view of the entire stage area. It is called the *stage camera*. These three cameras are connected to the composite routing switcher and provide live video sources for the webcast. Other video sources include the VCR and scan converted images from the presentation PC or laptop.

The speaker camera is used to show close-up pictures of the lecturer and to follow the speaker as he moves around the presentation stage. The use of a computer-controlled pan-tilt camera means that human input is required to follow the speaker. Many research groups have developed tracking algorithms based on audio, video, and sensor detection to track speaker movement automatically. We experimented with some image-based algorithms, but decided to deploy a commercial product, specifically a ParkerVision CameraMan, because it is relatively inexpensive and performs well. Consequently, we ignore speaker tracking in this work, assuming it will be implemented by the speaker camera.

Audio is provided by four microphones: a wireless mic, worn by the speaker, and three audience mics. The audience mics are placed in the ceiling, with one in

the left, center, and right portions of the room, as shown in the figure.

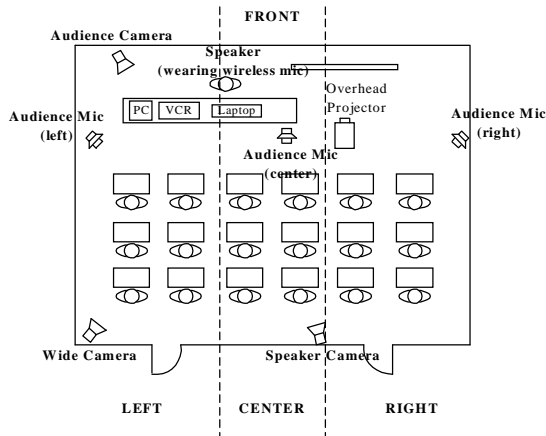


Figure 2 - Studio Classroom Setup

All equipment is connected to an AMX control computer [Panja]. A wireless panel allows the speaker to control which source is projected to the audience and to access other room controls (e.g., lights, audio levels, etc.). The AMX system also has a serial port connection to a host computer connected to the Internet so that equipment can be controlled remotely. Commands are sent to the AMX which relays the appropriate signal to the equipment. Several processes run on the host computer. One process, called *amxd* [Machnicki], is responsible for communicating with the AMX control computer. Remote applications connect to *amxd* and issue commands using an RPC mechanism (TclDP [Smith 1993]). Two *vcc3d* processes also run on the host computer. They communicate through a serial link with the Canon VCC3 cameras directly, and provide remote applications access to camera controls (e.g., pan, tilt, etc.) through the same RPC mechanism. Figure 3 shows the software and hardware architecture of the AMX system and host computer.

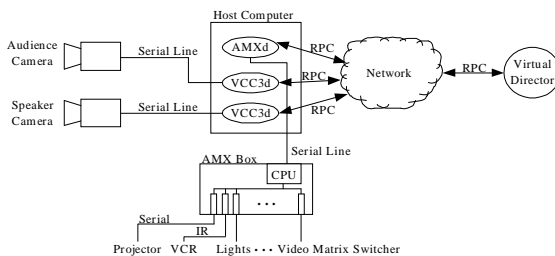


Figure 3 - AMX Control

In addition to providing remote access to the studio classroom equipment, the *amxd* process provides a callback mechanism to allow applications to monitor when significant events occur. These events, known as *AMX events*, are signaled when an application issues a command to *amxd* or when a touch panel button is pushed by the speaker. These events signal to the production environment that a particular command, such as “turn the lights on” or “show the laptop on the overhead projector” was executed. By monitoring these events, applications can monitor what is happening in the room and system

The *amxd* software provides a library that allows applications to monitor various AMX events. Applications use an RPC mechanism to register interest in specific AMX events with *amxd*. When registering, applications provide a callback procedure that is to be invoked when an AMX event occurs. Applications can register several callbacks, and each callback can be enabled or disabled. Moreover, applications can associate a filter with each callback that specifies in which types of events the application is interested. This mechanism reduces the number of callbacks invoked which reduces overhead.

The Virtual Director is built on and expands the infrastructure of the Director’s Console. The webcast infrastructure is shown in Figure 4. Video sources are routed through a matrix switch, and selected sources are sent to the encoding machines. The video is encoded and broadcast on a multicast session called the *studio session*. The scope of this session restricts it to the Berkeley campus. Audio is sent through an audio mixer to an encoding machine, which outputs the audio stream into the same studio session. *dc* allows the director to select which video sources are routed to the encoding machines and which streams from the studio session are routed to a *broadcast session* sent to the outside world. An important feature of *dc* is the ability to webcast several video streams simultaneously so the user can pick the stream(s) he wants to watch. *dc* and *vdc* control the broadcast by manipulating the equipment in the room through service interfaces.

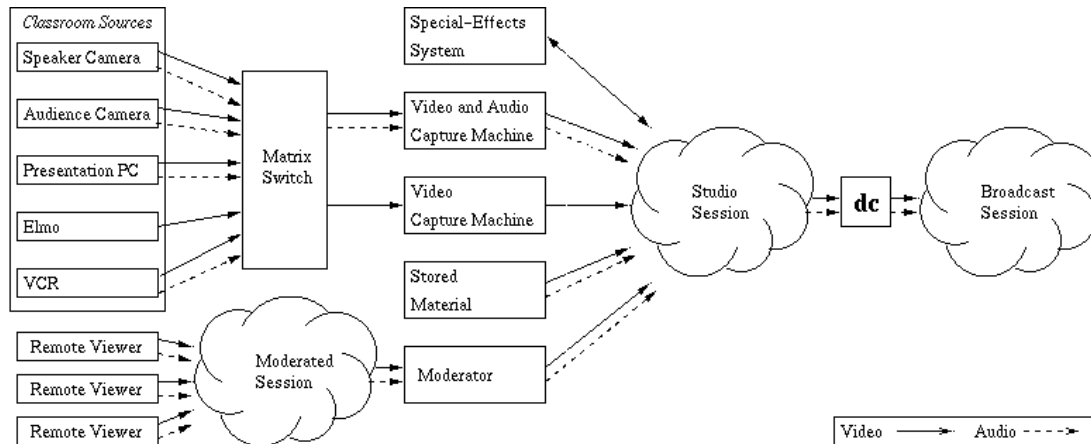


Figure 4 - DC Infrastructure

4. Broadcast Automation Service

Producing a webcast is a complicated process, requiring control of cameras, recording applications, and broadcasting equipment. In addition to broadcast logistics, many content decisions are made at an aesthetic level. For example, the director must decide which camera view should be displayed to remote viewers, or how often to show a subtitle giving the seminar title or speaker's name. While *dc* simplified the logistical tasks of a webcast, it did not address the more subtle, content-related tasks of a webcast. These tasks are left solely to the director's discretion. The broadcast automation service automates some of these content decisions. Moreover, it provides the flexibility to allow individual directors to specify different presentation styles by defining a set of rules that describe how automation should be implemented. Section 4.1 discusses the general concept of a language for specifying automation of a broadcast. Section 4.2 gives an example of a program in the broadcast automation language and explains what automation it specifies. Section 4.3 describes the Virtual Director implementation of the *broadcast automation service*, which interprets a broadcast specification and implements automation during a webcast.

4.1. Automation Specification Language

Figure 5 shows a narrative description that a human director might use to describe the production of a simple webcast.

1. Before the lecture starts, show a title screen.
2. When the lecture starts, broadcast a close-up of the speaker and the presentation slides, enable the room audio, and fade in on the speaker.
3. Show a subtitle with the speaker's name on the speaker stream for the first minute; after that, display it every 10 minutes for 15 seconds.
4. After showing the close-up of the speaker for 1 minute, switch to the stage view and show that for 15 seconds, then switch back to the close-up. Continue this throughout the lecture.
5. When the lecture finishes, disable the room audio, stop the recorder, and stop broadcasting.

Figure 5 - Description of a Typical Broadcast

This description provides details about logistical operations (e.g., start broadcasting, start recording, etc.) and content decisions (e.g., switch to the stage view every minute for 15 seconds). Moreover, it can be viewed as a template that can be applied to any webcast of a class lecture. The description specifies what actions to take and what content decisions to make. The *automation specification language* (ASL) provides a language for directors to specify such broadcast specifications, or ASL programs. The remainder of this section describes ASLs.

An ASL is composed of several basic abstractions, rules for defining how these abstractions interact, and a graphical user interface that allows manual intervention by a director. The basic abstractions are:

1. Stream – an audio or video stream
2. Source – an audio/video source that can be used in a broadcast
3. Global Clock – the master clock for the broadcast system

4. Services – abstractions for various media and control services

A *stream* has the following properties: *source*, *broadcast*, and *duration*. The *stream* properties are values that can be used in broadcast automation rules. These particular values are ones we believe are useful for specifying automation. The *source* property specifies what media source is currently being encoded and broadcast. The *broadcast* property is a Boolean value indicating whether the stream is currently being sent out to remote viewers in the broadcast session. The *duration* property is a clock value that specifies how many seconds this stream has been transmitting the current source. Changing the source resets the *duration* to zero. The *duration* of a stream is defined as zero if *broadcast* is false, i.e., the stream is not currently being broadcast. Streams might have additional properties such as volume level (e.g., audio) or subtitles (e.g., video). The stream abstraction allows the writer of a broadcast specification to ignore how a given media stream is produced (e.g., which machine is encoding the video). Note that stream properties are generic to any stream, they do not dictate any particular implementation of how the stream is created or how the *source* is switched.

The *global clock* abstraction represents the global clock for the production. It has only one property, the current value. The global clock can be used to make automation decisions relating to absolute or relative time. For example, the duration a particular source has been transmitting can determine when the source should be changed.

The *source* abstraction represents all the different media sources that can be routed to a stream for broadcast. Examples include cameras, microphone audio, computer screens, or stored material. Different production environments have different sources.

A *service* is the abstraction for processes or equipment that implement a function for the broadcast, the room, or a stream. For example, a recorder service is an abstraction for a recording device (e.g., digital archiver or analog tape deck). Operations on this service might include the usual VCR commands (e.g., record, pause, stop, etc.). There are a variety of other available services, and new services can be defined to deal with different types of equipment and software.

An ASL program specifies instances of these abstractions and a set of rules. ASL rules are similar to NSync rules [Bailey 1998], but use a different implementation. Each rule has a predicate and an action. The predicate is a statement with a Boolean value. When the value of a predicate changes from false to true, the associated action is executed. For example, the director may specify a rule such as

```
when {(source(spkr) = "speakerCamera" AND
(duration(spkr) >= 60)} {
    source(spkr) = "stageCamera"    #
broadcast stage camera on speaker stream
}
```

This rule says that the speaker stream, specified by the variable *spkr*, should be switched from the speaker camera, denoted by the constant "speakerCamera", to the stage camera if the speaker stream has been showing the speaker camera for more than 60 seconds.

An ASL program also defines a GUI component that is embedded in the Virtual Director. Buttons in the GUI can be used to invoke actions directly. The director can also enable/disable individual rules using the GUI. When a rule is defined in the ASL program, the "-checkbox" can be used to display a checkbox in the GUI that can be used to enable/disable the rule. The user interface can also display the status of various objects, such as which source a particular stream is broadcasting or how long it has been broadcasting.

The Virtual Director ASL is implemented as Tcl/Tk commands and OTcl objects in the Open Mash toolkit [McCanne 1997]. The use of Tcl/Tk provides an efficient way to prototype and test the effectiveness of the language and system.

4.2. Broadcast Specification Example

This section shows how to convert the human description presented above in Figure 5 into a broadcast specification for use by the broadcast automation service. Assume that there are two video encoding machines available. One machine encodes the slides or other content the speaker is presenting. We call this stream the *content stream*. The other stream, called the *speaker stream*, shows the speaker or audience members. Figure 6 lists an ASL program that might be used to automate such a webcast.² The GUI that will be displayed for this program is shown in the "Broadcast Automation" frame of *vdc* in Figure 1.

² Tcl/Tk syntax has been simplified to make the example more readable.

```

1. # create a button to start the webcast, define proc to call on push
2. button .start -text "start" -command "startPush"
3.
4. # create a button to end the webcast, define proc to call on push
5. button .end -text "end" -command "endPush"
6.
7. # assign speaker and content streams to video encoding machines
8. spkr = [new Stream "encodingMachine1"]
9. content = [new Stream "encodingMachine2"]
10.
11. # define actions to take when start button is pushed
12. proc startPush {} {
13.     source(spkr) = speakerCamera      # route speaker camera to speaker stream
14.     source(content) = presPC          # route presentation PC to content stream
15.     speakerCamera fadeIn              # fade in on speaker (camera service)
16.     broadcast(spkr) = true            # start broadcasting speaker stream
17.     broadcast(content) = true         # start broadcasting content stream
18.     broadcast(audio) = true           # start broadcasting room audio
19.     recorder start                    # start recording service
20.     startTime = globalClock           # record time of start of lecture
21.     startPushed = true                 # set startPushed variable to true
22.     text(spkr) = "Henry Rollins"     # show speaker's name on speaker stream
23. }
24.
25. # define actions to take when end button is pushed
26. proc endPush {} {
27.     recorder stop                      # stop recording
28.     broadcast(spkr) = false            # stop broadcasting speaker stream
29.     broadcast(content) = false         # stop broadcasting content stream
30.     broadcast(audio) = false          # stop broadcasting audio
31. }
32.
33. # rules to cover Step 3 - "show speaker's name on speaker stream for first minute;
34. #   after that, display it every 10 minutes for 15 seconds."
35. when -checkbox {(startPushed) AND ((globalClock - startTime) > 60)} {
36.     text(spkr) = ""                    # remove display of speaker's name
37. }
38. when -checkbox {(startPushed) AND ((globalClock - startTime) > 60) AND
39.              ((globalClock - startTime) % 600) = 0} {
40.     text(spkr) = "Henry Rollins"       # display speaker's name
41. }
42. when -checkbox {(startPushed) AND ((globalClock - startTime) > 60) AND
43.              (textDuration(spkr) > 15)} {
44.     text(spkr) = ""                    # remove display of speaker's name
45. }
46. # rules to cover Step 4 - "show close-up for 1 minute, stage view 15 secs"
47. when -checkbox {(source(spkr) = "speakerCamera") AND (duration(spkr) > 60)} {
48.     source(spkr) = "stageCamera"      # switch to show the stage camera
49. }
50. when -checkbox {(source(spkr) = "stageCamera") AND (duration(spkr) > 15)} {
51.     source(spkr) = "speakerCamera"    # switch to show the speaker close up
52. }
53. # specify any initial actions
54. source(content) = "titleScreen"       # broadcast title screen until
55. broadcast(content) = true              # lecture starts
56. startPushed = "false"                 # initialize startPushed variable

```

Figure 6 - Broadcast Specification File

Step 1 of the description from Figure 5 is covered by the initial actions specified on lines 52-55. Step 2 is covered by the director pushing the start button. This input event causes the startPush procedure to be called, which routes the video sources to the encoding machines, adds the speaker and audio streams to the broadcast, and starts the recorder. Step 3 is

accomplished by line 22 in combination with rules 3, 4, and 5. Line 22 causes the *speaker* stream to show the speaker's name when the lecture starts. Lines 35-37 specify that 60 seconds after the lecture has started, the subtitle should be removed. The rules specified in lines 38-40 and 41-43 are only active after the first minute of the lecture, and satisfy the "after that, display

it every 10 minutes for 15 seconds” phrase of step 3. Lines 38-40 specify that if the current time is a multiple of 10 minutes from the start of the lecture, the speaker’s name should be displayed. Lines 41-43 specify that if the subtitle has been displayed for more than 15 seconds, remove it. It is important to note that the actions associated with the *when* rules are only executed when the predicate’s value transitions from false to true. Otherwise, lines 35-37 would prevent the text from being displayed.

Note that the start and end buttons were used to specify transitions between segments of the webcast. The start button is used to signal the transition from the “ready” segment to the “active lecture” segment. Following the active lecture segment, there might be a “Q&A” segment, where the *speaker* stream is used to show the speaker and the *content* stream is switched to show the audience. The end button here just ends the broadcast, but one could imagine a short “closing credits” video being played before the broadcast ends. Transitions between these segments might be automated, either by using a strict time schedule, or by trying to infer transitions (e.g., by examining audio levels).

One apparent shortcoming of this implementation is the need for the *startPushed* and *startTime* variables. These variables are needed because we would like to synchronize actions with respect to the start of the lecture, which is signaled by the push of the start button. An ideal language might have implicit attributes associated with buttons and other objects, similar to SMIL 2.0 [Smil 2.0]. The use of Tcl/Tk for the implementation means that we must explicitly provide this synchronization.

4.3. Virtual Director Broadcast Automation Service

The broadcast automation service (BA) provides the interpretation of an ASL program like the example described in the previous section. The BA service reads a program, evaluate rules, and performs actions as necessary. This section describes the implementation of this service.

The BA service parses the program and creates a table containing all the rules, the state associated with the rules (enabled/disabled), and the actions to take when the rule evaluates to true. It is also responsible for maintaining the global clock. The global clock is actually just a counter that is incremented once a second. The Tcl “after” command is used to invoke a callback procedure every second. The callback is responsible for incrementing the current clock value, evaluating the rules, and executing any necessary actions. Because the global clock is only an approximation, this implementation is not sufficient for synchronization with events or processes outside of the broadcast automation service. It is sufficient, however,

to synchronize events and actions within a program. Note that each of the rules is reevaluated every second. A more efficient implementation based on a 4-valued predictive logic is used by NSync. Rules evaluate to “true”, “false”, “will be false at time X”, and “will be true at time X”. In the latter two cases, the rule is not evaluated until the specified time, thereby saving CPU cycles. We used the inefficient implementation because NSync does not work with Open Mash and we wanted to experiment with the application, not port the NSync code.

In addition to providing the basic framework for implementing the ASL, the BA service provides implementations of a few different stream and service abstractions. Two subclasses of the stream abstraction, notably *RvcStream* and *RealNetworksStream*, that further refine the stream abstraction are used. *RvcStream* is tied to an Open Mash remote vic application, which encodes video sources to produce RTP streams. The *RealNetworksStream* provides a Real Networks stream [Fitz 2001]. Changing the *source* property of these stream objects translates into RPC commands to *amxd* that change the connections within the video routing switcher. Callback events are used to keep the states of the *RvcStream* and *RealNetworksStream* objects synchronized with the actual video routing switch configuration. Each stream object registers a callback with *amxd*. When the source routed to the encoding machine associated with that stream object changes, the callback is invoked, and the *source* attribute of the stream object is updated to reflect the change. Callbacks are necessary because the video routing switch configuration may be changed by an external application, not just by changing a stream object’s *source* attribute. For example, the human director may decide to manually switch a stream’s source using the Director’s Console, or the speaker may change the presentation content. The use of callbacks ensures that the stream object’s *source* property is correct and makes the system more robust.

The BA service abstractions provide interfaces to other generic services that are used in a webcast. One example of this is the player service, which provides an interface to the MARS archive system [Schuett 1998]. The interface to the Virtual Director itself, which is used to enable/disable the question monitor service, is an example of a service specific to a particular production environment.

5. Question Monitor

This section describes the architecture and implementation of the question monitor (QM) service. The question monitor service monitors microphone volume levels and detects when an audience member has asked a question and where they are sitting in the

room. This information is used to switch cameras, if specified by the BA service, to show an audience member asking a question.

Recall that the speaker uses a wireless mic and that three ceiling mics are used to capture audience questions. An audience question can be spatially located in the room by comparing the signal strengths from the three audience mics. Unfortunately, no depth information can be inferred because the mics are in a line parallel to the front of the room. Instead, the room is divided into 3 sections (left, center, right), and the questioner is identified as being in one of these sections. The audience camera pans to show that section of the room.

The QM system architecture is shown in Figure 7. Each mic provides a raw volume level which is passed through a filter and fed to a selector algorithm. The selector identifies the *dominant* mic, based on the current state and filtered levels from each microphone. The *dominant* mic is passed to a state machine that decides on the next state which determines who the QM service believes is speaking and which camera view should be shown. If the QM service decides an audience member is asking a question, the *speaker* stream is switched to the audience camera and the camera is moved to show the correct section of the room. If, on the other hand, the QM service decides that the speaker is talking and the *speaker* stream source is not the speaker or stage camera, the stream is switched back to the speaker.

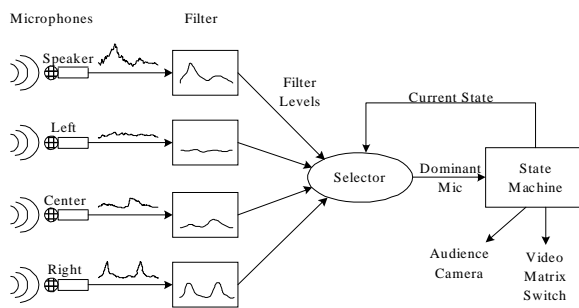


Figure 7 - Question Monitor Block Diagram

The audio level filter is necessary due to the characteristics of human speech. Human speech is marked by pauses and varying volume levels, so making a decision on one data point will not work. The filter used here is a threshold filter with *threshold* and *history* parameters. The filter counts how many of the last *history* samples were above the *threshold* level. The output of the filter is known as the filter level. The *threshold* level of a mic determines how sensitive the QM service is to that mic. A higher *threshold* means that the QM service is less sensitive to the mic.

Threshold levels can be used to adjust for noise in individual mics. The *history* parameter determines how much “memory” the filter has. A higher *history* value means that the filter level is determined by more microphone samples.

The sampling rate for the raw mic levels is 20 ms, so the filtered output levels are updated every 20 ms. The selector algorithm is invoked every *time_slice* ms. The selector identifies the *dominant* mic at the current time which theoretically indicates who is speaking. By experimentation, it was determined that a *time_slice* of 250 ms was more than sufficient to catch valid questions.

The selector algorithm chooses the *dominant* mic by examining the various mic filter levels. To begin selection of the *dominant* mic, the filter level of the current mic (e.g., speaker, left, etc) is increased by *current_mic_boost* to bias the algorithm to stay with the current mic. This heuristic allows audience members to pause slightly without losing the focus of the question monitor. The selection of the dominant mic proceeds by finding the mic with the maximum filter level.

If the mic with the highest filter level is an audience mic, its filter level is compared with the filter level of the speaker mic. The audience mic’s level must be greater than the speaker mic’s level by at least *spkr_offset_threshold* for the audience microphone to override the speaker mic. If the audience mic does not pass the *spkr_offset_threshold* test, the speaker mic is selected as the *dominant* mic. The idea here is to try to prevent false positives that may occur because the QM service detects a question when it is actually the speaker voice exciting the audience mic.

The filter level of the *dominant* mic is then compared against the *active_level* which is a minimum level that must be reached before any action is taken. Filter levels below this *active_level* are deemed to be background noise. If the *dominant* mic is active, the state machine is called with the *dominant* mic as an input.

The decision made by the state machine is based on the current state and the *dominant* mic chosen by the selector algorithm. The state diagram is shown in Figure 8.

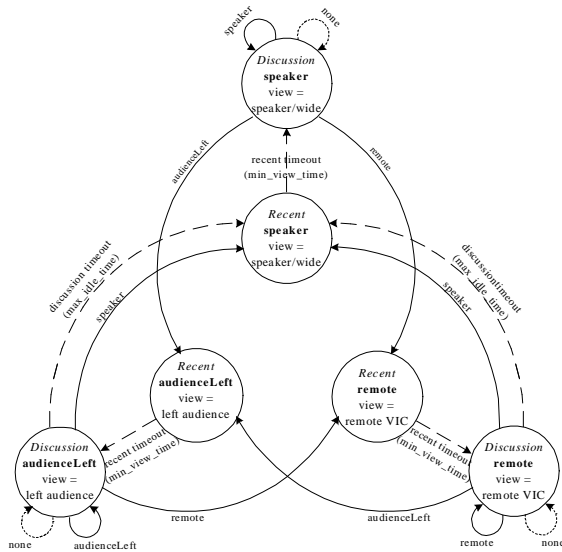


Figure 8 - QM State Machine

There are two basic types of states: *discussion* and *recent*. A *recent* state signifies that the camera view has just changed. A *discussion* state signifies that the current camera view has been shown for at least *min_view_time* seconds, and that *active* microphones can cause transitions. States can be further differentiated by their *mode*, which determines which microphone is associated with that state. The *modes* are *speaker*, *audience left*, *audience center*, and *audience right*. Each mode also determines what camera view is shown in that state. Note that only transitions to *recent* states cause the camera view to change.

Not all states are shown in the state diagram in Figure 8. The Virtual Director does not have the *remote* states, but has *audience center* and *audience right* states analogous to the *audience left* states. As you can see, each leg of the state machine is similar to the other legs. In fact, the whole state diagram is symmetric with the exception that *discussion timeouts* fall back to the default speaker state. Any number of audio inputs and corresponding states can be incorporated into the QM service, theoretically allowing it to select among sections in the room and remote viewers.

The state machine starts in the *discussion speaker* state. Transitions occur either because a timeout expires or because a *dominant* mic became active. *Active* mic inputs can cause the state to change when the current state is a *discussion* state. *Active* mics are ignored when the current state is a *recent* state. This design guarantees that the camera views are not switched too quickly, which is distracting to viewers.

There are two types of timeouts. A *recent timeout* occurs *min_view_time* seconds after a recent state is entered. It causes the state machine to transition to the associated *discussion* state. There is no other transition out of a *recent* state. Since the camera view only changes on a transition to a *recent* state, the *recent timeout* ensures that the camera view is not switched too frequently. More precisely, it ensures that the question monitor will show every camera view for a minimum of *min_view_time* seconds. We found that three seconds was a good *min_view_time*.

A *discussion timeout* causes a transition from a *discussion* state to the default state if there has been no *active* mic in the last *max_idle_time* seconds. This guarantees that a camera does not stay on an audience member if for some reason the speaker does not force a transition by talking. In practice, this timeout is rarely used because the speaker typically responds to the audience questions, which causes a state transition. Note that while in a *discussion* state, an *active* mic input associated with the current state will cause the *discussion timeout* to reset.

An *active* mic not associated with the current state causes a state transition to a *recent* state. The state machine will stay in the *recent* state until the *recent timeout* occurs. At this point, the state machine will transition to the associated *discussion* state. Once in a *discussion* state, the state machine stays in that state until an *active* mic input causes a transition or a *discussion timeout* occurs.

For example, assume the state machine is in the *discussion speaker* state. If the audience left mic is selected as *active*, the state machine transitions to the *recent audience left* state. When the transition occurs, the *speaker* stream switches to show the audience camera and the camera pans to show the left section of the room. At this point, we are in the *recent* state, so *active* mic inputs are ignored. After *min_view_time* seconds, the state machine transitions to the *audience left discussion* state. As long as the audience member keeps talking, the dominant mic will be the audience left mic, causing the state to remain *discussion audience left*. When the speaker responds, the speaker mic will become dominant and force a transition to the *recent speaker* state. When this transition occurs, the *speaker* stream is switched to show the speaker camera.

It is important to note that the state machine does not change each time period. It is only stepped when a mic is *active*. If no mics are *active*, the default action is to stay in the same state, as marked by the "none" pseudo-inputs to the *discussion* states. The reason for this heuristic is that audience questions may not always cause a mic to be active continuously. The state machine therefore assumes that until the speaker mic is activated, the audience member is still talking, regardless of the level of the audience mic. This design

makes the QM service more robust to variations in the speaking volume of audience members. In practice, we found it dramatically improved the QM's ability to maintain focus on audience members until their questions were completed.

Figure 10 summarizes the different parameters that affect the behavior of the QM service.

The next section will present the results of experiments that were performed with this algorithm.

6. Experimental Results

This section presents results from experiments performed with the Virtual Director. Section 6.1 examines the performance of the question monitor service in a real lecture environment. Section 6.2 describes the results of an experiment in which we compared the actions of a professional director to those of the Virtual Director.

6.1. Question Monitor Effectiveness

The Virtual Director was used to automate the production of several Berkeley Multimedia, Interfaces, and Graphics (MIG) Seminar webcasts in order to measure the effectiveness of the QM service. The raw mic inputs were recorded for each lecture. These inputs were fed into the QM service and the actions taken were recorded. To rate the effectiveness of the system, we looked at two metrics: recall and precision. *Recall* measures what percentage of actual questions were correctly identified. The recall measurement can be further broken down. When a question is asked, there are three possible responses by the QM: 1) the question can be correctly identified, 2) no question is identified (i.e., missed question), or 3) a question may be identified, but the section of the room is incorrectly identified. For our analysis, responses 2 and 3 have been grouped together as being incorrect. *Precision* measures how many times the question monitor identified a question, when there was no question (i.e., false positives). This can happen, for example, when the door to the room is opened or closed. These metrics are defined as follows:

$$\text{Recall} = \frac{\text{Number Questions Correct}}{\text{Total Number Questions}}$$

$$\text{Precision} = \frac{\text{Number Questions Correct}}{\text{Number Questions Identified}}$$

Both metrics have a value of 1 in the ideal situation, when all questions asked are correctly identified and there are no false positives.

The main parameter used to affect the QM service behavior is the mic *threshold*, which can be individually adjusted. This adjustment affects the

threshold filter, which determines how easily the mic can be selected as the *dominant* mic. We examined the effects of varying the audience mic thresholds while keeping the speaker mic threshold constant at 40. The *history* size of the filter and the *time_slice* of the QM algorithm were fixed at 50 samples and 250 ms, respectively. In this system with many independent variables that are not linearly related, we needed to fix some of the parameters to find the effects of the other parameters.

Figure 9, shows the effect of changing the audience mic thresholds, while keeping the speaker mic *threshold*, filter *history* size, and *time_slice* constant.

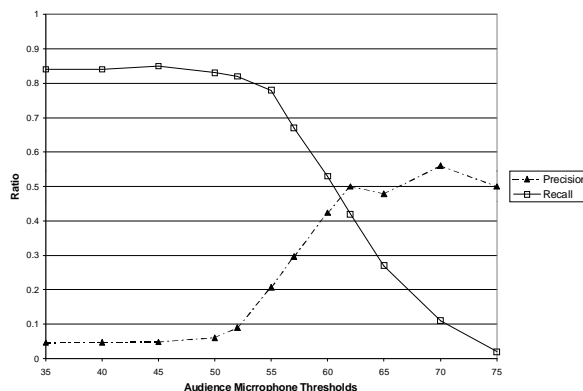


Figure 9 - Question Monitor Effectiveness

As you can see, the recall and precision metrics are inversely proportional. To increase recall, the mic filters must be made more sensitive to noise by decreasing the *threshold*. However, this causes the number of false positives to increase, thus decreasing precision.

The choice of a particular threshold level is dependent on the relative value given to correctly identifying questions versus trying to reduce the number of false positives. Fortunately, the lecture broadcast environment is one where absolute correctness is not required. Missing a few questions just means that the speaker is shown while a question is being asked. While this error is not ideal, it is in no way “incorrect” and does not seriously reduce the quality of the webcast. Similarly, an occasional false positive is acceptable. Indeed, occasionally showing the audience makes the webcast more interesting to remote viewers. QM effectiveness only becomes significant at the extremes, where all questions are missed, or where the audience is shown every 10 seconds due to a large number of false positives. As long as the thresholds can be chosen to avoid these extremes, the QM service is an effective tool.

Parameter Name	Where Used	Description	Value Used
threshold	mic filter	determines how sensitive QM service is to this mic	speaker = 40, audience = 60
history	mic filter	determines how many of the last samples are used in the filter level	50 samples
spkr_offset_threshold	selector algorithm	amount by which audience filter level must exceed speaker filter level for audience mic to be dominant	8
active level	selector algorithm	filter level that must be reached for mic to cause a transition	10
time_slice	selector algorithm	how often the filter levels are checked	250 ms
current_mic_boost	selector algorithm	amount by which filter level of current mic is boosted before being compared to filter levels of other mics	5
min_view_time	state machine	minimum time a camera view can be shown	3 sec
max_idle_time	state machine	maximum time in a discussion state with no active mics before transitioning to default speaker state	7 sec

Figure 10 - Question Monitor Parameters

It should be noted that selection of the various mic thresholds is a highly experimental task. Variations in the room configuration (e.g., air conditioning, projector noise, echoes, etc.) influence system effectiveness. Moreover, individual mics may be different, so thresholds must be separately tuned for each one. The studio classroom we used had many of these problems. The overhead projector is close to the center microphone, so the sensitivity of that microphone is much lower than that of the left and right microphones. Another problem that shows the sensitivity of the system to outside forces was the installation of equipment in a closet in the adjoining room. The low level vibration caused the right microphone to be activated erroneously. This unexpected error reduced the effectiveness of the right microphone. Indeed, when questions are broken up into different parts of the room, as in Figure 11, you can see that the different mics have significant differences. The left mic performs the best. The center mic has a recall and precision of 0, because none of the center questions were correctly identified by the QM service! What usually happened is that a center question would be identified as a left or right question. For this reason, the camera views were set up to overlap at the edges. The external excitation of the right microphone can be seen in the poor precision curve of the right mic.

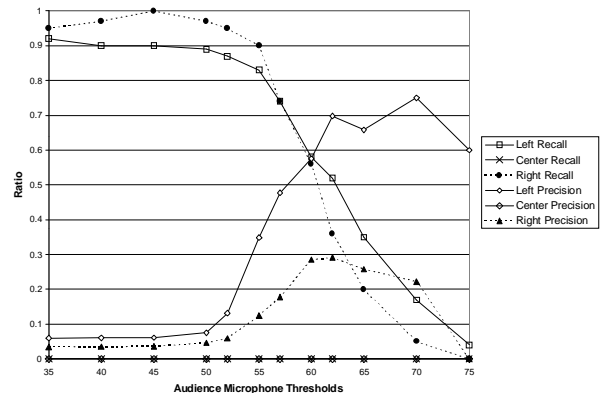


Figure 11 - QM Effectiveness by Section

The question monitor did perform fairly well for the left section of the room however, showing that the QM algorithm is sound. Improvements in room configuration as well as more sophisticated filtering would also improve system effectiveness. Section 7 discusses some of these improvements.

6.2. Human Director Comparison

An experiment was performed to compare webcast content decisions made by a professional director and the Virtual Director. The professional director (Vince Casalaina) has won a local Emmy, and has years of experience in the television production industry. We designed a mock lecture that lasted 15 minutes that incorporated most kinds of interesting events that

might take place during a real lecture. These events include questions from the audience from different sections of the room, writing on the whiteboard, and changing sources on the overhead projector. The human director produced a webcast of the mock lecture and the results were recorded, including the raw microphone inputs. We then fed the microphone inputs to the Virtual Director and generated a shot selection list using the heuristics shown in Figure 12.

1. If the close-up view has been shown for more than 120 seconds, switch to the next view [next view is “stage” the first and second time, “audience” the third time, cycling back to “stage” after the third switch]
2. If the stage view has been shown for more than 30 seconds, switch to close-up view
3. If the audience view has been shown for more than 30 seconds, switch to close-up view
4. Fade-in on the speaker when starting
5. Fade-out on the speaker when ending
6. If the content stream is not showing what is on the projector, switch the content source to be the same as the projector source

Figure 12 - Original Virtual Director Heuristics

The camera selections made by the human director and software automation are shown in Figure 13. Remember that our webcast model consists of two streams: the *speaker* stream that shows the speaker or audience, and the *content* stream that shows the presentation content (e.g., slides, video, etc.). Only the *speaker* stream is shown in the figure.

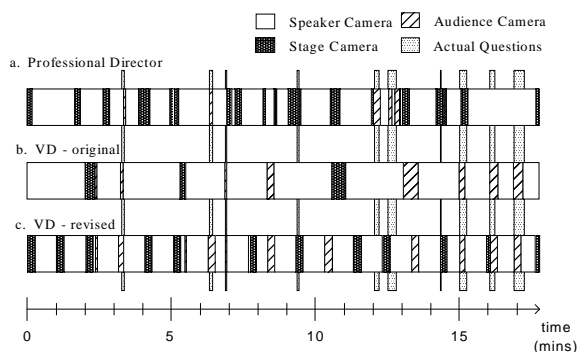


Figure 13 - Human Test Shot Selection

The first thing we notice is that the human director started the lecture with the stage camera and then switched to the speaker camera close-up about 10 seconds into the broadcast. The Virtual Director started with the close-up speaker shot and faded in. Another observation is that the human director tended to stay on individual camera views for shorter time

periods. The typical pattern was to show the close-up of the speaker for approximately 45 seconds, then show the stage camera view for about 15 seconds. In contrast, the Virtual Director showed the close-up of the speaker for 2 minutes, then the stage camera view was shown for 30 seconds. The heuristics also specified that every third switch from the speaker camera went to the audience camera which then panned over the audience. The human director never used audience camera panning. This content decision may have been because the room was relatively empty or because the human director was unfamiliar with the tools. Similar to the opening, the human director switched to the stage camera at the end of the lecture to end the broadcast. The Virtual Director heuristics specified that the close-up of the speaker should be shown and the camera faded out to end the broadcast.

The mock lecture included an example of the lecturer switching the overhead projector to show the presentation PC instead of the laptop. The speaker projected slides from a laptop computer, and other online material was projected from the presentation PC, which had Internet connectivity. When the source to the projector was switched, the human director switched to the stage view for about 5 seconds before returning to the close-up view of the speaker. This type of heuristic was not incorporated in the Virtual Director’s broadcast specification.

Another interesting difference was the delay when switching the *content* stream. When the lecturer changed the projector to display the presentation PC, the video source of the *content* stream needed to be changed. The human director, on the average, took almost 7 seconds to update the *content* stream. *vd* on the other hand, propagated the change in less than a second.

The professional director and the Virtual Director also handled questions differently. The human director switched the content stream to show the audience instead of the lecture slides when the presentation was finished. I.e., the two streams showed the audience member asking a question and the speaker answering the question side by side. The Virtual Director has no mechanism for detecting this situation and switching the streams to show these two views. Adding this capability requires human intervention to recognize an extended period of questions during which the presentation material being projected is unimportant. Automating this detection may require more sensors (e.g., recognizing when a speaker uses a pointer on the projected image) or more sophisticated heuristics. Another difference between the professional director and *vd* involved the handling of long audience questions. When the audience member asked a relatively long question, the human director switched to show the speaker listening to the question for a few

seconds, and then switched back to the audience member until the question was completed. The question monitor currently does not have this type of heuristic, though it could be implemented by using timeouts in the *audience discussion* states.

As expected, the human director did a better job of detecting questions and switching the *speaker* stream to show the audience. The QM service correctly identified about half of the questions, and missed the other half, giving a recall of 0.5. There were two false positives, so the precision was 0.71. The professional director correctly identified all but one question, though it is hard to directly compare the human and Virtual Director because the human director used the *content* stream to show the audience at the end of the lecture. Assuming that the human director would have correctly identified the last few questions, his recall was 0.9, and his precision was 1.0.

The lecturer writing on the whiteboard produced some interesting observations. When the lecturer wrote on the board, the human director zoomed in on the whiteboard so remote viewers could see what was being written. When the lecturer had finished and was summarizing what was written on the board, the human director switched to the stage camera view. The Virtual Director currently has no way of knowing that the speaker is writing on the board, so heuristics can not be specified to deal with this situation. Adding simple computer vision analysis or a whiteboard recording tool like a Mimo [Virtual Ink Corporation] might yield more information and enable more complicated heuristics to deal with these types of situations.

In terms of shot selection, many of the differences, such as durations of camera views, were caused by the fact that the heuristics used by the human director did not match the Virtual Director heuristics. We rewrote the heuristics to match the heuristics observed from the human director (see Figure 14). Figure 13 c shows the results of producing a webcast with these revised heuristics.

1. *If the close-up view has been shown for more than 45 seconds, switch to next view [next view is “stage” the first and second time, “audience” the third time, cycling back to “stage” after the third switch]*
2. *If the stage view has been shown for more than 15 seconds, switch to close-up view*
3. *If the audience view has been shown for more than 15 seconds, switch to close-up view*
4. *Show stage camera when starting*
5. *Show stage camera when ending*
6. *If the content stream is not showing what is on the projector, switch the content source to be the same as the projector source*

Figure 14 - Revised Virtual Director Heuristics

As you can see, the shot selection decisions closely match the style of the human director. The transitions between camera views are more frequent, and the opening/closing uses the stage camera. However, these heuristics produce very regular behavior, which is not like a human director. Randomness could be used to make the Virtual Director behave more similar to the human director. For example, the duration of the stage camera could be specified as 15 seconds, plus or minus a random time of 0 to 5 seconds. This example illustrates the flexibility of the broadcast automation service to allow each director to tailor the style of webcast content decisions.

7. Experience and Future Work

The Virtual Director has been used to webcast several Berkeley MIG Seminars. This section explains what worked well, what did not work well, and offers some possible solutions to improve *vd.c*. It then discusses two directions for future work.

The Virtual Director was clearly useful when automating simple, straight-forward logistical tasks. For example, the startup of a webcast was reduced to the push of a simple “start” button which automated the actions of turning on the “on air” warning light outside of the classroom, starting the recorder, initiating broadcast of the *speaker* and *content* streams, and fading in on the close-up of the speaker to begin the seminar. All these actions occurred almost instantaneously and the use of the Virtual Director removed the possibility of a human director forgetting to do one of them, as has happened in the past.

While the use of buttons in the BA service was almost always beneficial, the effectiveness of the rules-based automation was not as clear. Some rules were simple and produced predictable results. One example is the “*content* stream follows projector” rule which guarantees that whatever video source is displayed on the projector to the local audience is also broadcast to remote viewers. Without this rule, the director must pay attention to the lecture and manually switch the video source being fed to the *content* stream when the lecturer changes the projector input.

Other rules produced good automation in the general case, but caused problems in a few, rare instances. A good example of this type of situation is the rule-driven camera switching. The rules specify a maximum duration for a given camera, and which camera to switch to at the end of that duration. In the general case, this produces a professional looking video, with perspective changes to keep the webcast interesting. Moreover, the use of customizable rules allows each director to tailor the content decisions to

match his directing style. The problem is that the camera switching heuristics are completely time-based, and do not incorporate significant events in the lecture itself. For example, consider the case where a lecture is given by two people, with the first lecturer speaking for 30 minutes, and then turning the microphone over to the next speaker. When the second lecturer begins to talk, the webcast should show a close-up of the speaker with a subtitle giving his name, or at least show a stage view during the transition. However, it may be that the Virtual Director happens to be showing the audience when the transition occurs and the new speaker begins to talk. To properly automate camera selection in this situation, the system must know that the speaker has changed.

On the whole, we found that the BA service provides a powerful mechanism for specifying automation tasks in a simple manner. Indeed, the Virtual Director originally consisted of several additional services, which were later incorporated into the broadcast specification. The “*content stream follows projector*” rule is one of them. There was originally a separate service that was responsible for implementing the “*content stream follows projector*” heuristic. Later, we realized that this application could be expressed in a simple ASL rule, such as the following:

```
if {source(content) != source(projector)}
{
    source(content) = source(projector)
}
```

All that was needed was an abstraction for the projector that provided the current input source. The entire content follower service was then reduced to 3 lines of Tcl/Tk in the broadcast specification.

Although the specification of rules-based automation provides a great deal of flexibility, the definition of the rules themselves is fixed throughout the webcast. This rigidity can lead to problems. Consider the camera-switching rule used in the Berkeley MIG seminar:

```
if {(source(spkr) = "speakerCamera") AND
(duration(spkr) > 120)} {
    source(spkr) = [next]
}
```

where “next” selects the next camera from the list: stage camera, stage camera, and audience camera. This rule says that after the speaker has been shown for two minutes, switch to the next camera view. The next camera view is the stage camera the first and second time, and the audience camera the third time, after which, the process repeats. However, the director may notice that the classroom is unusually empty and decide that the audience view should not be shown.

Unfortunately, this change cannot be made dynamically in the current framework. Individual rules can be enabled or disabled but they cannot be modified. The obvious solution is to support a dialog box to change the “next camera” list, but the current system cannot do it.

The QM service was less successful than the BA service. It performed acceptably for an automated webcast, but was clearly not as effective as a human director. Some problems were caused by the audio problems inherent in our particular studio classroom. On the other hand, questions from the left section of the room were identified quite well. Given the simplicity of the algorithm and the low cost of computation, the resulting performance is not surprising. Using more complex analysis of the audio data, such as speech detection, frequency filtering, and echo cancellation of the projector noise could significantly improve performance.

Directions for future work can be broken into two broad categories: 1) improve the existing system and 2) extend the functionality of the Virtual Director. One possibility for improving system performance is advanced audio processing, such as speech detection and frequency band filtering to improve the QM effectiveness. The automation specification language could also be improved by providing more abstractions and a more powerful set of basic object properties. For example, there is currently no way to directly say “three minutes after the button is pushed, do this....” Ideally, buttons would have a “push time” associated with them, and this property would be inherent in the language itself. The implementation of the BA service could be improved by using a constraint system similar to NSync.

The interaction of the QM service and the BA service also needs to be further examined. The current version of the Virtual Director does not address this interaction. Both services have some control over the audience camera and the selection of sources for the *speaker* stream. However, they currently operate independently, with the only interaction being that the BA service can disable and enable the QM service. Research needs to be done to see what constraints and interfaces are needed to prevent problems such as rapid camera switching, where a given camera view is shown for too short a time, and to determine which service should dominate in the case of conflicts.

One promising direction to extend the Virtual Director is to formally study content decisions by human directors and develop system extensions and heuristics to mimic their behavior. In particular, different content, such as a presidential debate or a town meeting, will use different heuristics than a university lecture.

One goal of *vdc* is to enable a single person to produce several webcasts simultaneously. As discussed above, the system is not perfect, so human intervention will occasionally be needed. Management of several simultaneous webcasts is a complicated task, even with the help of the Virtual Director. A framework that can assist with this management is needed. The framework must detect when a particular webcast needs attention and notify the director. One possible approach is to have several *vdc*s integrated into a single application. When a problem is detected, the appropriate interface is displayed and the director is prompted to fix the problem.

Another area in which *vdc* falls short is remote viewer participation. Currently, there is limited support for remote viewers to ask questions. The Virtual Director already contains most of the infrastructure required. Remote viewers can transmit audio and, optionally, video into the studio session. The audio could be given as another input to the question monitor service, and the appropriate video stream could be displayed when the remote viewer asks a question. Of course, a floor control monitor is required to prevent mischievous viewers from disrupting the webcast at will.

8. Conclusion

This paper described the design and implementation of the Virtual Director, a system composed of a broadcast automation service and a question monitor service, that can be used to produce automated webcasts. While we did not aim to replace the human director in the production environment, the system reduces the number of tasks requiring human intervention, and simplifies the webcast production process. The ultimate goal is to have one person produce several webcasts at the same time.

The broadcast automation service automates some mundane, logistical aspects of a webcast and reduces the complexity of the production process. This service uses customizable heuristics to switch between cameras which results in semi-professional looking webcasts. We performed an experiment to compare the actions of a professional director to the actions taken by the Virtual Director. The automated system performed adequately but can still be improved.

The question monitor, though not perfect, did perform acceptably in a lecture situation. We examined the performance of the system, looking at recall and precision, and discussed how they are inversely proportional, which requires directors to weigh the tradeoffs between reducing false positives and increasing recall. We also discussed the impact of the performance of the question monitor, arguing that a

few false positives or missed questions are not harmful to the quality of a webcast.

Our experience has shown that the Virtual Director is a valuable tool in the production of university lecture webcasts.

Acknowledgments

I would like to thank my advisor Lawrence Rowe for being so helpful and supportive of this project. I greatly appreciate the time he spent with me discussing problems and providing insight. I would also like to thank all those who helped with the experiments. Many thanks go to Vince Casalaina for his directing and comments on the system, to the audience members who read their lines with grace: Sharad Agarwal, Timothy Fitz, Jose Maria Gonzalez, Lloyd Lim, and Lawrence Rowe, and to Matt Delco for his assistance in gathering data from webcasts. I would also like to thank Richard Shu for his help in providing the initial data for the question monitor system. Finally, I would like to express my sincere appreciation for funding which was provided by NSF Internet Technologies Program Grant 9907994, NSF Instrumentation Equipment Grant 9512332, and a GAANN fellowship.

References

1. G Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. IBM Systems Journal, vol.38, (no.4), IBM, 1999. p.508-30.
2. AutoAuditorium. www.autoauditorium.com.
3. Brian Bailey, Joseph A. Konstan, Robert Cooley, and Moses Dejong. Nsync – A Toolkit for Building Interactive Multimedia Presentations. ACM Multimedia '98, Bristol, UK, 1998.
4. Jason Brotherton, Janak Bhalodia, and Gregory Abowd. Automated Capture, Integration, and Visualization of Multiple Media Streams. Proceedings. IEEE International Conference on Multimedia Computing and Systems, Austin, TX, USA, 1998.
5. Steven Drucker and David Zeltzer. Camdroid: A System for Implementing Intelligent Camera Control. 1995 Symposium on Interactive 3D Graphics, Monterey, CA, USA, 1995.
6. H. Eriksson. The Multicast Backbone. Communications of the ACM, vol. 8, 1994.
7. Jeffrey A. Fayman, Oded Sudarsky, and Ehud Rivlin. Zoom Tracking. Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium, May 1998.

8. Timothy Fitz. FASTMedia: A Framework for Streaming Multimedia Distribution and Access. Masters report, 2001.
9. David Franklin, Shannon Bradshaw, and Kristian Hammond. Jabberwocky: You don't have to be a rocket scientist to change slides for a hydrogen combustion lecture. Proceedings of the 2000 International Conference on Intelligent User Interfaces, New Orleans, LA, USA, 2000.
10. David Franklin, Joshua Flachsbar, Kristian Hammond. The Intelligent Classroom. IEEE Intelligent Systems, vol.14, (no.5), IEEE, Sept.-Oct. 1999. p.2-5.
11. Michael Gleicher and James Masanz. Towards Virtual Videography. ACM Multimedia 2000, Los Angeles, CA, USA, 2000.
12. T.D. Grove, K.D. Baker, and T.N. Tan. Colour Based Object Tracking. Proceedings of the Fourteenth International Conference on Pattern Recognition, Brisbane, Australia, August 1998.
13. H. Hirsh, M.H. Coen, M.C. Mozer, R. Hasha, J.L. Flanagan. Room Service, AI-style. IEEE Intelligent Systems, Vol. 14, (no. 2), March-April 1999.
14. Yoshinari Kameda, Hideaki Miyazaki, and Michihiko Minoh. A Live Video Imaging for Multiple Users. Proceedings IEEE International Conference on Multimedia Computing and Systems (vol.2), Florence, Italy, 1999. p.897-902 vol.2.
15. Peter Karp and Steven Feiner. Automated Presentation Planning of Animation Using Task Decomposition with Heuristic Reasoning. Proceedings Graphics Interface 93, Toronto, Ontario, Canada, 1993. Canadian Inf. Process. Soc, 1993. p.118-27.
16. Qiong Liu, Yong Rui, Anoop Gupta, and JJ Cadiz. Automating Camera Management for Lecture Room Environments. SIGCHI'01, Seattle, WA, USA, March-April, 2001.
17. Erik Machnicki.
bmrc.berkeley.edu/~machnick/amxd.
18. Steven McCanne, Eric Brewer, Randy Katz, Lawrence Rowe, Elan Amir, Yatin Chawathe, Alan Coopersmith, Ketan Mayer-Patel, Suchitra Raman, Angela Schuett, David Simpson, Andrew Swan, Teck-Lee Tung, David Wu, Brian Smith. Toward a Common Infrastructure for Multimedia-Networking Middleware. Proceedings of the IEEE 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video, St. Louis, MO, USA, May 1997.
19. Sugata Mukhopadhyay and Brian Smith. Passive Capture and Structuring of Lectures. Proceedings ACM Multimedia 99, Orlando, FL, USA, 1999. p. 477-87.
20. Panja. www.panja.com.
21. ParkerVision. www.parkervision.com.
22. Jyoti Parwatar, A. Engbretson, T. McCartney, John DeHart, Kenneth Goldman. Vaudeville: a High Performance, Voice Activated Teleconferencing Application. Multimedia Tools and Applications, vol.10, (no.1), Kluwer Academic Publishers, Jan. 2000.
23. L.A. Rowe, et.al. BIBS: A Lecture Webcasting System. Technical Report, Berkeley Multimedia Research Center, U.C. Berkeley, May 2001.
24. Angela Schuett, Randy Katz, and Steven McCanne. A Distributed Recording System for High Quality Mbone Archives. Proceedings First International Workshop on Networked Group Communication, Pisa, Italy, November, 1999.
25. Angela Schuett, Suchitra Raman, Yatin Chawathe, Steven McCanne, and Randy Katz. A Soft-state Protocol for Accessing Multimedia Archives. Proceedings 8th International Conference on Network and Operating Systems Support for Digital Audio and Video, Cambridge, UK, July, 1998.
26. Brian C. Smith, Lawrence Rowe, and Stephen C. Yen. Tcl Distributed Programming. Proceedings of the 1st Tcl/Tk Workshop, June 1993.
27. Synchronized Multimedia Integration Language (SMIL 2.0) Specification.
www.w3.org/TR/smil20.
28. Virage. www.virage.com.
29. Virtual Ink Corporation. www.virtualink.com.
30. Tina Wong, Ketan Mayer-Patel, David Simpson, and Lawrence Rowe. A Software-Only Video Production Switcher for the Internet Mbone. SPIE Multimedia Computing and Networking, January, 1998.
31. D. Wu, A. Swan, L.A. Rowe. An Internet Mbone Broadcast Management System. SPIE Multimedia Computing and Networking, January 1999.
32. Tai-Ping Yu, David Wu, Ketan Mayer-Patel, and Lawrence Rowe. dc: A Live Webcast Control System. SPIE Multimedia Computing and Networking, January 2001.