A THREE-LAYER VIRTUAL DIRECTOR MODEL FOR SUPPORTING AUTOMATED MULTI-SITE DISTRIBUTED EDUCATION

Bin Yu[†] *, Cha Zhang*[‡] *, Yong Rui*[‡] *, Klara Nahrstedt*[†]
[†] University of Illinois at Urbana Champaign, IL 61801, USA
[‡] Microsoft Research, Redmond, WA 98052, USA

ABSTRACT[1]

In multi-site distributed education (MSDE), video streams from multiple sites are available. To best utilize the limited screen space at each site, we develop a customizable, automated display management system in this paper, i.e., only user-preferred streams will be shown as triggered by events and timers. The configuration of such user preference, however, is challenging because it has to be both human-friendly and machine-friendly. To address this challenge, we propose a three-layer virtual director model. In the *user layer*, we identify three categories of parameters that can represent a wide range of user preferences yet are easy to use. These preferences are then automatically translated into a machine-friendly timed automaton in the *execution layer*. The automaton is simulated dynamically, which selects a subset of streams to show on the screen through a *display layer*. Evaluation results demonstrate the correctness and efficiency of the proposed framework.

1. INTRODUCTION

To accommodate students' time and/or place conflicts, in the past decade, universities started to offer *distributed education* where the lecturer and the students are not required to be in the same classroom. A common form of distributed education is what we call *multi-site distributed education* (MSDE). For example, in the case of [7], a lecturer and some students are collocated in the same classroom (e.g., University of Toronto), while other students attend the lecture remotely as a group (e.g., IBM Research Center).

In MSDE systems, each site can have one or more cameras. Many existing systems tile all the videos on the display regardless of their relative importance 0[2]. Such systems are not scalable when the number of video streams increases. In addition, they do not make efficient use of the screen real estate because most video streams contain interesting events only at limited times. Recent systems such as AutoAuditorium [6] and iCam [8] are able to automatically generate a single video output from multiple input streams based on events and timers. Take the iCam system as an example. There are multiple software modules called virtual cameramen (VC). Each VC controls a camera, automatically tracking a lecturer or a talking audience member. The VCs communicate with another software module called virtual director (VD), which is responsible for selecting the best camera shot given the available camera shots from all VCs by using a timed automaton (*finite state machine with timers*) [5]. For example, when a student asks a question, the VD will automatically switch to the camera that shows the student; when a camera has been on air for a certain period, the VD will switch to another camera to improve the aestheticity of the output. Note that the first example is an *event-driven switch* and the second example is a *timer-driven switch*.

In this paper, we extend automated camera management systems such as iCam to MSDE. Unlike AutoAuditorium or iCam, which outputs the same video for all remote audiences, in MSDE it is often desirable that each site has its own VD. For example, the site that has the lecturer onsite may not need the lecturer video because they see the lecturer directly and can save some screen real estate for other video streams. The challenge we face is that in iCam the VD's state automaton is written in a machine-friendly scripting language, which needs to be edited by a well-trained system administrator. This configuration process is tolerable for a single site. But it can be cumbersome, inefficient and error-prone if it has to be reconfigured to support different numbers and types of VCs and produce customized output for different sites.

We observe that human users are more comfortable with the semantics of camera streams and preference among them. Hence we propose to divide the VD into three layers. At the *user layer*, a human-friendly graphical interface is designed to allow users to specify their preference among the cameras in the form of *scores*, which is an intuitive way of configuring the event-driven and timer-driven switches. This user interest model is then automatically *translated* into a machine-friendly timed automaton that will be executed by the VD at the *execution layer*. Due to the large number of possible states when the number of cameras increases, we perform such translation dynamically so that the execution layer can be run efficiently. Finally, the selected camera streams are laid out on the display device through the third layer ─ the *display layer*. The benefit of such a three-layer design is that any number of cameras capturing different views can be "compared" in a generic way by the VD without understanding their semantic differences, and users can easily reconfigure the desired camera switching behavior by changing the user interest model parameters, at system setup time or, if desired, during a live class session.

The rest of the paper is organized as follows. Section 2 presents the architecture of an MSDE system with distributed VCs and VDs on each site. Section 3 describes the three-layer VD model that is designed to be both user- and machine-friendly. In Section 4, we evaluate the proposed approach first by demonstrating the functional correctness of the automaton, and then by giving an example that shows how a VD selects different camera shots given the automaton and event triggers. We conclude the paper in Section 5.

2. SYSTEM ARCHITECTURE

Our system is built on top of the ConferenceXP platform [2], which supports real-time high quality video conferencing (based

[1] Work performed during the first author's internship at MSR.

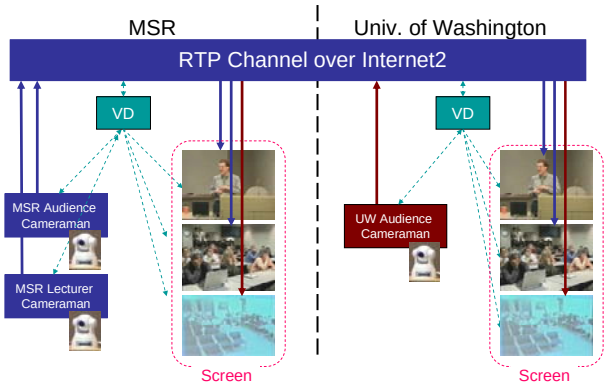


Figure 1. Example Setup for a Two-site Distributed Class between Microsoft Research and University of Washington.

on Internet2 IP Multicast) at 30 frames per second as well as text chat, shared presentation, and shared whiteboard. Our goal is to add automated camera management to ConferenceXP while leveraging its existing functionalities.

Figure 1 shows an example setup for a two-site distributed class between Microsoft Research (MSR) and University of Washington (UW). The MSR site has two VCs: "MSR Lecturer VC" and "MSR Audience VC"; the UW site has a single "UW Audience VC". All VCs send their videos via the RTP channel over Internet2. Each VC is capable of producing multiple types of views. For instance, the lecture VC automatically tracks the lecturer and produces three view types: "global view" of the front stage of the room, "close-up view" of the lecturer's head and shoulder, and "zooming view" indicating the mechanical zooming operation of the camera. Similarly, the audience VCs can produce "global view" of the audience area, "close-up view" of a talking audience member, and the "zooming view".

At each site, there is a VD which decodes all the videos, and selectively places them on the screen based on the user's preference. The VDs talk to all the VCs through a simple protocol: whenever a VC changes its view, e.g., the MSR Audience VC identifies a student asking a question and changes to a "close-up view", it will send a message to the local VD to update its state. The local VD will then broadcast this message to all other VDs. On the other hand, all VDs can broadcast their requests for any VC to perform a task such as creating a panning view. The local VD has the responsibility to operate the local VCs to implement the task. The commands from the VDs are processed at each VC in a first-come-first-serve manner, with a timer which prohibits two commands being executed in less than 3 seconds.

Some of the above architecture resembles that of our previous iCam system [8], but the VDs are now distributed. The focus of this paper is how to allow users to easily setup VDs at each site. Our solution is to have a three-layer VD model to help the configuration and customization process, which will be described below.

3. THE THREE-LAYER VD MODEL

We have chosen a three-layer design for the VD, as shown in Figure 2. The *user layer* maintains a user interest model that learns user preference among all the view types of all the cameras; the *execution layer* is responsible of selecting the camera streams to be displayed on the screen; the *display layer* shows the selected streams on the screen in a customizable way. We will describe these three layers in details next.

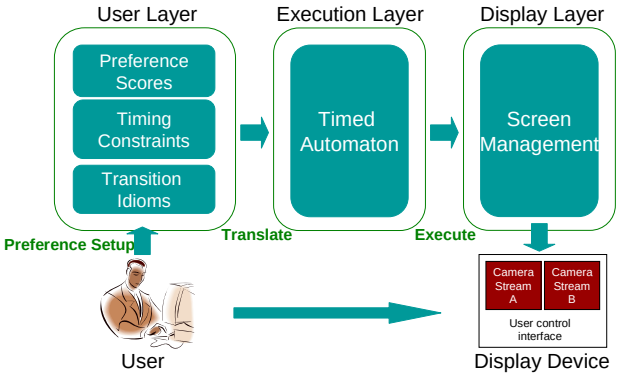


Figure 2. Overview of the VD Model.

3.1. User Layer

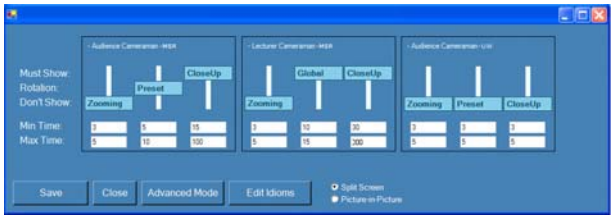
We have chosen three categories of parameters that represent a wide range of user preference patterns yet are easily configurable by the user, namely *preference scores*, *timing constraints* and *transition idioms*. Figure 3 (next page) shows an example graphic interface for the user to setup his/her preference. Note that these preference parameters can be setup as a configuration file before a class begins, or dynamically changed *during* a live class session.

***Preference Scores*:** As illustrated in Figure 3(a), the user can assign a preference score to each view type of a camera as "must show", "show in rotation" or "do not show", which represent the relative "interestingness" of a camera view compared to other cameras. Since each camera will be in one of the view types at any given time, we define a *camera's score* as the score of its current view type. If a camera is in a view type that is scored "must show", it must be selected by the VD to show on the display; if the view type is scored "show in rotation", it will be shown one by one in rotation if there is no "must show" camera; if the view type is scored "do not show", it shall not be selected unless no other cameras can be selected. We find that such a three-level preference model is intuitive and allows users to easily compare cameras capturing very different contents.

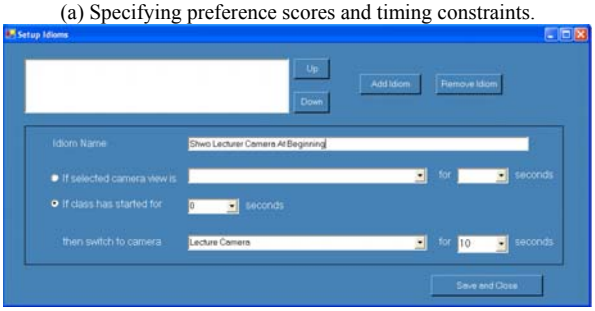
With the above preference scheme, the camera selection decision can be made as follows. If there are one or multiple cameras in view types of "must show", all of these cameras will be selected to show in a split-screen or picture-in-picture layout (Section 3.3); otherwise, if there are one or multiple "show in rotation" cameras, show them one by one in rotation; otherwise, randomly select one camera to show on screen. Note that such randomness may not be wanted and can be avoided if the user sets some camera views as "must show" or "show in rotation".

Timing Constraints: There are two types of timing constraints: *minimum show time* and *maximum show time*, which represents the minimum and maximum amount of time a certain view should be shown. A camera should not change view type if the minimum show time was not reached. The maximum show time applies to view types that are "show in rotation", and determines the length of selection before rotating to the next camera.

Transition Idioms: The preference scores and timing constraints are generally expressive enough to produce satisfactory results, but there are cases when some advanced transitions are desired. For instance, the user may want to always start the class video by showing the lecturer for 10 seconds, or, he/she may want to always show the audience view at site 2 after showing the audience view



(a) Specifying preference scores and timing constraints.



(b) Specifying transition idioms.

Figure 3. Screen Shot of User Layer Interface

at site 1. Transition idioms are designed to fulfill such tasks. They will *override* the score-based camera switching behavior.

As shown in Figure 3(b), each transition idiom takes one of the following two formats:

- If the time since the beginning of the class equals T_1 seconds → select camera C_i for T_2 seconds.
- If the camera view V_j has been selected for display for T_1 seconds → select camera C_j for T_2 seconds

In the above example, the user may specify the following two idioms: "if the time since the beginning of the class equals *0* seconds → select lecturer camera for display for *10* seconds"; "if the audience global view at site 1 has been selected for *5* seconds → select the audience camera at site 2 for *5* seconds".

3.2. Execution Layer

The goal of the execution layer is to select camera streams according to the user's preference. Similar to existing solutions [6][8], we rely on a timed automaton to make camera selection decisions at runtime. The timed automaton is a *finite state machine with timer* translated from the input parameters at the user layer: the states represent the current view types of all cameras, the selected cameras and the status of the screen output (e.g., how long the selected cameras have been displayed on screen); the transitions represent state changes triggered by camera view type change, timer events or user specified transition idioms. If there are N cameras, and each camera could be in one of M view types, the total number of possible states of the automaton is $O(M^N)$. Note that in MSDE, each site can have multiple streams selected and displayed. This is in contrast to our previous work iCam [8], where at any instance there is only one camera on air.

Due to the large number of possible states, generating a full automaton will incur exponentially increasing computation cost with respect to the number of cameras, which is unnecessary. We propose to simulate the timed automaton *dynamically* during a lecture. That is, the VD will only keep a *partial* automaton, which covers the current state of the automaton and the potential transitions that may occur at this state.

Figure 4 shows the flow chart of the algorithm for camera selection based on user preference parameters. It uses four modes to

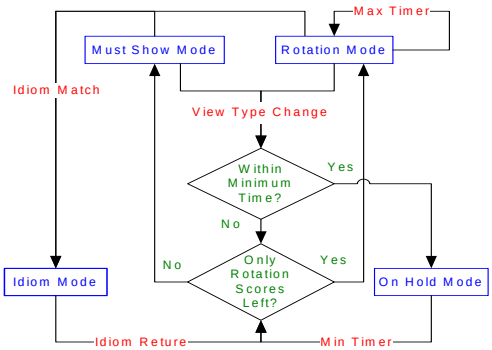


Figure 4. Flow Chat of Camera Selection Algorithm

represent the concrete states of the timed automaton: "must show mode" represents those states with at least one "must show" cameras; "rotation mode" represents those states with no "must show" cameras but one or more "show in rotation" cameras; "on hold mode" represents those states resulting from view type changes occurring within the minimum time for camera switching; "idiom mode" represents those states resulting from a matching idiom. On each transitional event ("view type change", "min timer expiration", "max timer expiration" and "idiom match/return"), the VD first decides the new mode to switch to, and then calculates the new state of the timed automaton and updates camera selection output. We will present a detailed walk through example in Section 4 to demonstrate this camera selection process.

3.3. Display Layer

The display layer manages the screen display given the camera streams selected at the execution layer. The user initializes the display layer by specifying a *rendering region* (RR) on the desktop screen to be used for video rendering. If only one stream is selected, it will occupy the whole RR. In case multiple cameras are selected, they share the RR with a screen layout choice specified by the user. Currently two screen layout modes have been implemented. With *split screen layout*, the RR will be equally allocated for all selected camera streams; with *picture-in-picture layout*, one randomly chosen camera stream occupies the whole RR while others are shown as overlay windows. We are working on supporting more layout modes, e.g., showing "must show" videos with large windows and "show in rotation" videos with small windows.

4. DEPLOYMENT AND EXPERIMENTS

We built an MSDE system on top of the ConferenceXP platform, and it is currently being used for CSE Professional Masters Program classes between MSR and UW [3]. Figure 5 shows a screen shot of the system during a test class. Since there was a conversation going on between the lecturer and the student, both the MSR lecture VC and the MSR audience VC were in "close-up" view, which was configured with "must show" score. Therefore, both VCs were selected by the VD. The two windows are shown in the split screen mode, although other modes are possible depending on the user preference.

The VDs are very easy to configure. Setting up the VD usually takes less than 5 minutes by a regular user, in contrast to about an hour by a well-trained system administrator without the three-layer model, e.g., in [8]. Next we evaluate the proposed approach first by demonstrating the functional correctness of the automaton, and

Figure 5. A Screen Shot of the System in S*plit Screen Layout*.

then by giving an example that shows how a VD selects different camera shots given the automaton and event triggers.

4.1. VD's Functional Correctness

We adopted the UPPAAL verification tool [4] to study the timed-automaton-based camera selection algorithm and analyze its functional correctness in terms of *liveness* and *reachability*. Specifically, for each case of K ($2 <= K <= 6$) cameras and M ($2 <= M <= 4$) view types per camera with randomly assigned preference scores, we have generated a timed automaton. Each state in the automaton represents a combination of view types of all the K cameras, as well as the timing constraints and camera selection output. Each transition represents a change in camera view type, a timer event, or an idiom-based transition. The automata are generated as XML files in the UPPAAL automaton description format. We then use the UPPAAL verifier to test the following properties:

Liveness: transitions in a timed automaton are constrained by clock values, and there is possibility of deadlock (no outgoing transition exists). In our system deadlock is not possible because we allow multiple video streams to be shown on the screen, which reduces the dependency between VCs. In the experiment we verified the liveness of all the randomly generated automata against the query "**A [] not deadlock**" using the UPPAAL verifier, where "A" represents the target automaton and "[]" means "invariantly". All automata we generated for different K and M values have passed the test.

Reachability: The other expected property of VD is that all the states in the timed automaton should be reachable from the initial state within limited number of transitions, as the VD is supposed to be able to support any camera view combinations and make the right camera selection decision. We verified the reachability of 5 randomly chosen states in each automaton with the following query "**E <> selected_state**", where "E <>" means "possibly". All selected states are verified as reachable by the UPPAAL verifier.

We have also tested VD's functional correctness empirically by running VD for long periods of time with different numbers of cameras, sites and random user preference parameters. Since the system was deployed, we have not seen any unexpected camera selections. In addition, because of the *dynamic partial automaton*, the run time execution of camera selection is very lightweight, and on average takes less than *1.0* millisecond on a regular PC. The camera switching introduces negligible overhead since only re-arranging of camera output windows on the desktop is required.

4.2. Example VD Walkthrough

We take the two-site class between MSR and UW as the example. Table 1 shows a set of preferences set by the system administrator

		Preference	Min Time (s)	Max Time (s)	Idiom
Audience Cameraman MSR	Preset	Show in rotation	5	10	
	Close-up	Must show	15	100	
	Zooming	Do not show	3	5	
Lecturer Cameraman MSR	Global	Show in rotation	10	15	0 sec → select for 10 sec
	Close-up	Must show	30	200	
	Zooming	Do not show	3	5	
Audience Cameraman UW	Preset	Do not show	3	5	
	Close-up	Do not show	3	5	
	Zooming	Do not show	3	5	

Table 1. Example User Preference Setup

Time (s)	Event	Selected output
0	Initial view types MSR Lec.Global, Aud.Preset UW Aud.Preset	MSR Lec
10	MSR Lec.Global → Lec.Zooming	MSR Aud
15	MSR Lec.Zooming → Lec.Closeup	MSR Lec
60	MSR Aud.Preset → Aud.Zooming	MSR Lec
63	MSR Aud.Zooming → Aud.Closeup	MSR Lec, MSR Aud
80	MSR Aud.Closeup → Aud.Zooming	MSR Lec
83	MSR Aud.Zooming → Aud.Preset	MSR Lec
215	MSR Lec Max Timer expires, Lec.Closeup → Lec.Zooming	MSR Aud
218	MSR Lec.Zooming -> Lec.Global	MSR Aud
...	...	...

Table 2. Example Scenario Walkthrough

at UW. Note that the audience VC at UW is set to be "do not show" for all the view types, because the students can see what happens locally. Initially, the MSR lecture (Lec) VC is shooting a global view, and the audience (Aud) VCs at both sites are shooting a preset view. Due to the idiom in Table 1, the initial view is MSR Lec. Then at 10^{th} sec, the idiom expires, and the MSR Lec VC starts to zoom in. The selected view is hence MSR Aud which has preference "show in rotation". Once the Lec VC changes to a close-up view, it is "must show" and the selected camera becomes MSR Lec again. The remaining steps can be interpreted similarly.

5. CONCLUSION

In this paper, we presented a three-layer VD model that automatically controls which camera streams are presented in each classroom in MSDE. The system automatically chooses the right cameras based on a timed automaton that is dynamically translated from a user-friendly interest model. This design makes deploying an MSDE system practical.

REFERENCE

[1] Access Grid, http://www.accessgrid.org/.

[2] Microsoft ConferenceXP, http://www.conferencexp.net

[3] Professional Masters Program at University of Washington, http://pmp.cs.washington.edu/dl_tech/

[4] UPPAAL Formal Verification Tool, http://www.uppaal.com/

[5] R. Alur and D. Dill. "The theory of timed automata". Theoretical Computer Science, 126(2), 1994.

[6] M. H. Bianchi, "AutoAuditorium: a fully automatic, multi-camera system to televise auditorium presentations", in Joint DARPA/NIST Smart Spaces Technology Workshop, 1998.

[7] P. Smith and K. Lyons, "User experience in the first ARISE distributed classroom", *eLearn Magazine*, http://www.elearnmag.org/subpage.cfm?section=case_studies&article=18-1

[8] Y. Rui, A. Gupta, J. Grudin and L.W. He, "Automating lecture capture and broadcast: technology and videography", in ACM Multimedia Systems Journal (Springer), 10:3-15 2004