P2P Video Synchronization in a Collaborative Virtual Environment

Suhit Gupta
Columbia University
500 W. 120th Street
New York, NY 10027
United States
001-212-939-7184
suhit@cs.columbia.edu

Gail Kaiser
Columbia University
500 W. 120th Street
New York, NY 10027
United States
001-212-939-7081
kaiser@cs.columbia.edu

Abstract

We have previously developed a collaborative virtual environment (CVE) for small-group virtual classrooms, intended for distance learning by geographically dispersed students. The CVE employs a peer-to-peer approach to the frequent real-time updates to the 3D virtual worlds required by avatar movements (fellow students in the same room are depicted by avatars). This paper focuses on our extension to the P2P model to support group viewing of lecture videos, called VECTORS, for Video Enhanced Collaboration for Team Oriented Remote Synchronization. VECTORS supports *synchronized* viewing of lecture videos, so the students all see "the same thing at the same time", and can pause, rewind, etc. in synchrony while discussing the lecture material via "chat". We are particularly concerned with the needs of the technologically disenfranchised, e.g., whose only Web/Internet access if via dialup or other relatively low-bandwidth networking. Thus VECTORS employs semantically compressed videos with meager bandwidth requirements. Further, the videos are displayed as a sequence of JPEGs on the walls of a 3D virtual room, requiring fewer local multimedia resources than full motion MPEGs.

1. Introduction

Learning is essentially a social activity and is of paramount importance in engineering project-based courses, where a high degree of cooperation is required [11]. The Columbia Hypermedia IMmersion Environment (CHIME) system [8] [9], created by the Programming Systems Lab (PSL – http://www.psl.cs.columbia.edu) at Columbia University, was designed as a framework for distributed software development environments. CHIME's users would be software project team members who might be geographically dispersed, but could be *virtually* collocated within the same "room" or adjoining "rooms" of a MUD-like 3D virtual world. The layout and contents of this *groupspace* represent the software project artifacts and/or the on-going software process. This model is similar to the one being developed at MIT iLabs [22].

CHIME has more recently evolved into a general collaborative and information management infrastructure. The artifacts or tasks represented by the system or brought together in "rooms" need not be oriented to software development alone. The focus has extended beyond just collaboration to visualization of varied data and services through which users can collaborate and that can be represented in the virtual

environment. One such example of visualizing varied pieces of data includes viewing segments of videos that are pre-taped lectures of classes held here in the Computer Science Department at Columbia University.

Distance learning programs such as the Columbia Video Network and the Stanford Center for Professional Development have evolved from mailing (via Fedex and the like) lecture video tapes to their off-campus students to streaming the videos over the Internet. The lectures might be delivered “live”, but are frequently post-processed and packaged for students to watch (and re-watch) at their convenience. This introduces the possibility of forming “study groups" among off-campus students who view the lecture videos together, and pause the video for discussion when desired, thus approximating the pedagogically valuable discussions of on-campus students. Although the instructor is probably not available for these discussions, this may be an advantage, since on-campus students are rarely afforded the opportunity to pause, rewind and fast-forward their instructors' lectures.

However, collaborative video viewing by multiple geographically dispersed users is not yet supported by conventional Internet-video technology. It is particularly challenging to support WISIWYS (what I see is what you see) when some of the users are relatively disadvantaged with respect to bandwidth (e.g., dial-up modems) and local computer resources (e.g., archaic graphics cards, small disks). The VECTORS (Video Enhanced Collaboration for Team Oriented Remote Synchronization) plug-in was added to CHIME to allow users to synchronize on video based data. This was done by combining techniques that extract key frames from a video stream to create a semantically rich version of the video [16] and fast peer-to-peer UDP packet based synchronization [10], we allow groups of users to watch videos in synchrony, regardless of their bandwidth limitations.

We have adopted technology (developed by others, Liu and Kender [16]) for “semantically compressing" standard MPEG videos into sequences of still JPEG images. This technology automatically selects the most semantically meaningful frames to show for each time epoch, and can generate different sequences of JPEG images for a range of different compression (bandwidth) levels. This approach works very well for typical lecture videos, where it is important, for instance, to see what the instructor has written on the blackboard after he/she stands aside, but probably not so important to see the instructor actually doing the writing, when his/her hand and body may partially cover the blackboard. The remaining technical challenge is synchronizing the downloading and display of the image sequences among each of the distributed user clients, including support for shared video player actions such as pause. Further, if student groups do indeed sometimes pause the videos, or rewind to a point already available in local buffers (caches), it is desirable to take advantage of the then-idle network connection to pre-fetch future images at a higher quality level.

We have developed an approach to achieving this, using a few mechanisms working in tandem. First, the video clients communicate with each other over a distributed publish-subscribe event bus, which propagates video actions taken by one user in the group to all the other users in the group. Thus any user can select a video action, not just a “leader". Secondly, since the clients are viewing the video in a

collaborative virtual environment, they synchronize their movements using streams of UDP packets that give all virtually-near clients updates of where one is. It is these streams of UDP packets that we piggyback our video synchronization anchors. Finally, there is a centralized feedback control loop on the video server that dynamically adjusts each video client's choice of both the next image to display and also the next image to retrieve from the semantic compression levels available. The controller relies on sensors embedded in each client to periodically check what image is currently displaying, whether this image is “correct" compared to what other clients are viewing, which images have already been buffered (cached) at that client, and what is the actual bandwidth recently perceived at that client. Actuators are also inserted into the video clients, to modify local configuration parameters on controller command. The controller utilizes detailed information about the image sequences available at the video server, including image start and stop times (both the individual images and their start and stop times tend to be different at different compression levels), but unlike local client data, video server data is unlikely to change while the video is showing. A single controller is used for all clients in the same user group, so it can detect “skew" across multiple clients, and may reside on the video server or on another host on the Internet.

In the next section, we further motivate the collaborative video viewing problem, provide background on the semantically compressed video repository, and explain the technical difficulties of optimizing quality while synchronizing such semantically compressed videos. The following section presents the related work in the field followed by our architecture and dynamic adaptation model, and its implementation in VECTORS. We then summarize our contributions. This document serves to explain the overall problem, the related work in the field and the system architecture and infrastructure for the CHIME/VECTORS system along with providing demo screenshots, and will finally conclude with future directions.

2. Motivation

2.1 Purpose

The usefulness of this application stems from the fact that students often collaborate when studying for exams or preparing for lectures. By providing them with a groupware product that allows them to do this in a Same Time / Different Location setting, combined with the unifying synchronization functionality, to ensure that they will all be watching the same part of a given video in the same place, and the ability to communicate via web chat, etc, we hope to improve student productivity significantly. Furthermore, the Server-Side AI involved in breaking down video streams into their most basic, significant parts, allows for this unified experience to take place over a group of students with variable network connection speeds.

2.2 Sample Scenario.

Student A organizes a study group during a Thanksgiving Break. The professor for his course, CS1000, has scheduled a midterm exam for the day after school resumes, and has provided online

VECTORS Lecture Videos for all lectures to date. Student A, who remains on campus for their break, invites students B, C, and D to join in the study group, and schedules a meeting with them for 5:00 PM on November 28th.

On November 28th, each student connects to the Internet, (Student A via his campus connection, Student B and C via DSL at home, and Student D over a 28.8 kbps dial-up connection). Each student opens the CHIME application, which provides them with group chat and VECTORS Video capabilities. As they chat, they all agree to open their video player. Since their meeting is scheduled, the video players have a list of the group members, and the video player constantly communicates with the other players in the group, to ensure that all of them are at the same time in the video. Student A decides to start the video. The clients, after all communicating with one another, begin to play the video simultaneously. Each student hears the same audio of the lecture, though the various client players will be downloading a different set of "Key" frames to display on the screen based on their network bandwidth and performance. At a point later in the video, student D has a question. He pauses the video, which in turn, pauses the video across the entire list of players, and then asks his question over the CHIME Chat Console. Once the group agrees that the question has been dealt with, any one of them presses the Play/Resume button, and all of the students' video players resume the video from the point they left off.

This process continues for the duration of the video, or until the group declares their study session to be complete. A diagram of the connections is below:

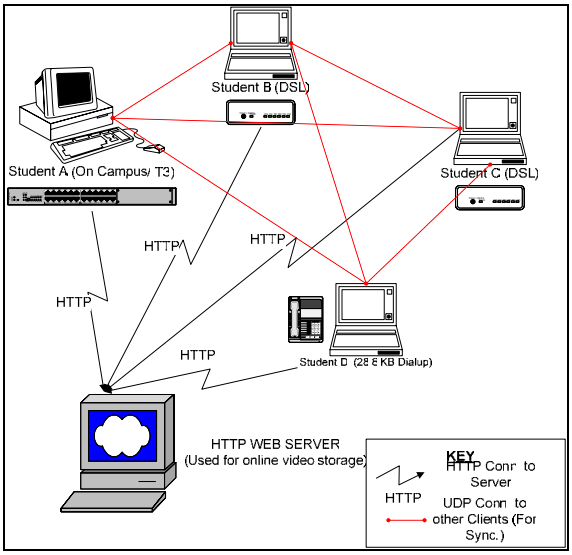

Figure 1 - Sample Scenario

2.2 Abstract Description

The VECTORS project manifests its purpose in a server-client architecture. On the server side, the software optimizes MPEG (and other continuous video formats) for use over a variety of network bandwidths by filtering out "Key Frames" – frames that are deemed to be most significant, and richest in content. These key frames (all of which are a subset of actual frames in the video) are then grouped into

smaller subsets, which are referred to as "Compression Levels." All compression levels share the same lossless audio track, but each video level is a successively smaller set of key frames. Each set corresponds to a bandwidth quality level determined by the clients, and a set of key frames, is noted in a text list, and directs the player to preload that specific set of frames, also noting where each frame (in the overall video timeline) is to begin and end being displayed.

The client, the 3D collaborative virtual environment, handles the downloading of an appropriate subset of frames, and manages the video timeline, providing playing and pausing capabilities for the end user. It is designed to facilitate group viewing of this multimedia in a MUD (Multi User Domain). Currently CHIME is the environment where this player resides. Unlike a typical player, the VECTORS player downloads a list of frames for all possible compression levels for a given video, and then switches between compression levels as it attempts to predict bandwidth performance. In addition to "smart downloading" it allows for multiple members in a group to control the group's multimedia experience by playing and pausing the video across all members of the group. This is accomplished via network communication between the various clients.

3. Related Work

There has been a rich amount of work done in the field of Collaborative Virtual Environments (CVE) over the years. The key feature of research in CVE has been the social engineering aspect and the attempt to improve the user interface over which users communicate seamlessly with others [11] [23]. Basic research in virtual environments has focused on identifying potential indicators of effective collaborative learning teams and the types of problems that may result from insufficient group interaction and support [24].

Prasolova-Forland discusses the mechanisms employed to improve social awareness in education [11] [12] and has found that the traditional technical tools are not enough, and the mechanisms offered by CVEs provide a more promising supplement to the mechanisms in use already.

Brouras et al. [4][5] describe a fairly robust virtual environment that supports education, however their environment needs to be modified by editing the VRML files whenever new materials need to be introduced into the system. This would unfortunately make the system extremely hard to operate because changes could not be made to the world during run-time, thus potentially disrupting the flow of work.

Okada et al. [17] tell us about their system that sets up a CVE for the realization of environmental education. While this helps users communicate and gain knowledge about nature, and users can set up virtual areas at runtime, the setup process of creating the environment is fairly tedious. The user is responsible for uploading all the information about the environment s/he wishes to create to the server.

Oliveira et al. [18] brings the idea of CVE and collaborative learning to Industrial training and e-Commerce. However, the environment that they describe can only be access from a set of predefined camera views. Additionally, while their system can support video on demand, they do not take into account that users may wish to view the video as a group but may not have the bandwidth resources to keep up with

users that do.

Finally, the advantage the 3D CVEs, with a MUD like interface, give over traditional web-based collaborative environments is the ability for users to see what his/her peers are doing. We have found that it is this feature that is critically missing from other approaches. Additionally, most worlds cannot dynamically add to the layout of the existing world or even change the look and feel of an existing room to suit an individual end-user. It is our goal to look at these key shortcomings as well as address the ones overlooked by our peers.

In addition to the work that has gone into virtual environments that are geared towards educational purposes, stream synchronization is a widely studied topic in multimedia research. Classifications of synchronization schemes consider whether the scheme is local or distributed (i.e., one or multiple sinks), whether they take action reactively or proactively, and whether a global clock is required. Our work does not address the problem of inter-media synchronization of multiple modalities (i.e., video and audio), where the concern is to ensure the correctly timed playback of related data originating from different streams. Our problem is instead related to intra-stream synchronization, which is concerned with ensuring the temporal ordering of data packets transmitted across a network from a single streaming source to one or more delivery sinks.

Most intra-stream synchronization schemes are based on data buffering at the sink(s) and on the introduction of a delay before the play-out of buffered data packets (i.e., frames). Those synchronization schemes can be rigid or adaptive [39]. In rigid schemes, such as [31], the play-out delay is chosen a priori in such a way that it accounts for the maximum network transfer delay that can likely occur across the sinks. Rigid schemes work under a worst-case scenario assumption and accept the introduction of delays that may be longer than necessary, in order to maximize the synchronization guarantees they can o_er even in demanding situations.

Contrary to a rigid approach, adaptive schemes [25] [32] [34] re-compute the delay parameter continuously while streaming: they try to “guess” the minimum delay that can be introduced, which still ensuring synchronization under actual operating conditions. In order to enhance quality of service in terms of minimized play-out delay, those schemes must accept some temporary synchronization inconsistencies and/or some data loss, in case the computed delay results are at times insufficient (due, e.g., to variations in network conditions) and may need to be corrected on the fly.

Our approach to synchronization can be classified as a centralized adaptive scheme that employs a local clock and operates in a reactive way. The most significant difference compared to other approaches, such as the Adaptive Synchronization Protocol [25], the work of Gonzalez et al. [30], or that of Liu et al. [28] (which can all be used equally for inter- and intra-stream applications), is that our approach is not based on the idea of play-out delay. Instead, we take advantage of layered semantic compression coupled with buffering to “buy more time" for clients that might not otherwise be able to remain in sync, by putting them on a less demanding level of the compression hierarchy.

To ensure stream synchronization across a group of clients, it is usually necessary to implement

some form of trade-off impacting the quality of service of some of the clients. Many schemes trade off synchronization for longer delays, while some other approaches, like the Concord local synchronization algorithm [59], allow a choice among other quality parameters besides delay, such as packet loss rate. Our approach sacrifices frame rates to achieve synchronization when resources are low.

Liu et al. provide a comprehensive summary of the mechanisms used in video multicast for quality and fairness adaptation as well as network and coding requirements [27]. To frame our work in that context, our current design and implementation models a single-rate server adaptation scheme to each of the clients because the video quality we provide is tailored specifically to that client's network resources. The focus in our work is directed towards the client-side end-user perceived quality and synchrony, so we did not utilize the most efficient server model. The authors believe that it would be trivial to substitute in a simulcast server adaptation model [34]. Our design also fits into the category of layered adaptation. Such an adaptation model defines a base quality level that users must achieve. Once users have acquired that level, the algorithm attempts to incrementally acquire more frames to present a higher quality video. In the work presented here, the definition of quality translates to a higher frame rate. Liu's discussion of bandwidth fairness, coding techniques and network transport perspectives lie out of the scope of this paper.

With respect to the software architecture, our approach most resembles the Lancaster Orchestration Service [34], since it is based on a central controller that coordinates the behavior of remote controlled units placed within the clients via appropriate directives (i.e., the VECTORS video buffer and manager). The Lancaster approach employs the adaptive delay-based scheme described above; hence the playback of video focuses on adapting to the lowest bandwidth client. That approach would degrade the playback experience of the other participants to accommodate the lowest bandwidth client. Our approach seems preferable, since it enables each client to receive video quality commensurate with its bandwidth resources.

Cen et al. provide a distributed real-time MPEG player that uses a software feedback loop between a single server and a single client to adjust frame rates [7]. Their architecture incorporates feedback logic within each video player and does not support synchronization across a group of players, while the work presented here explicitly supports the synchronization of semantically equivalent video frames across a small group of clients.

With this wealth of related work information, we set out to build a pedagogical system that would help non co-located students in bridging the geographical barrier and participate in group based projects with video components much like their peers that studied on-campus.

4. Our Solution

The goal was two-fold – to create a robust and dynamic collaborative virtual environment that would be a good enough framework for future plug-ins like video synchronization; and to create a near real-time video synchronization plug-in that would allow for students to participate in group based projects despite not being co-located.

4.1 CHIME

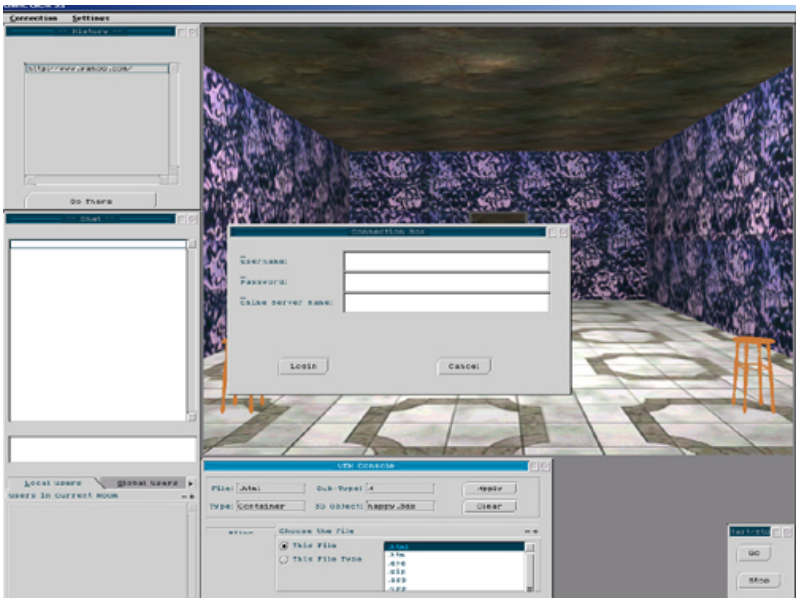

Figure 2 – CHIME environment upon startup – prompt for sources that need to be visualized

Our solution employs multiple extensible techniques that incorporate the advantages of the previous work on collaborative virtual environments. CHIME [8] [9] is a metadata based information management and visualization environment, created to serve as a homogenous environment for heterogeneous applications and data for internet and intranet-based distributed software development. It has further been extended to serve as a framework that aims at managing and organizing information. CHIME is designed around an XML-based metadata architecture, in which the software artifacts continue to reside in their original locations [8].

CHIME is divided up into the centralized server component and the distributed 3D client front-end. The client takes user input about mapping some existing data source in the 3D world and gives the user a representation of the data based on preset visualization defaults. The key insight in CHIME is to not map the actual data into the world, instead to use software to extract metadata about the data specified and visualize it. Metadata really is nothing more than data about data; a catalog record is metadata; other form of description [2]. This saves each client from potentially fruitlessly downloading large amounts data in order to visualize it, as it is not necessary that the user will actually use all the data that s/he queries.

The architectural process involves logging the user onto the system and taking the user query, communicating with the appropriate data source and extracting the relevant metadata from the data source using one of several pre-written plugs. The metadata extracted by the plug is encapsulated in XML tags so as to compress it and provide a rich stream of information to the server. The extracted metadata is then parsed and 3D model information is assigned. This information is finally sent to all the clients in the appropriate virtual spaces. This communication medium is primarily an Internet scale, publish-subscribe event system, in our case, we use Siena [19]. Example screenshot of the final result can be seen in Figure 2 and 5 and in Section 8.

The above process allows both administrators as well as users to create rooms in the world. Therefore, a user can browse through existing rooms that s/he has access to and interact with the objects that populate that space, as well as change his surrounding environment by mapping more data sources. This provides for an excellent teamwork oriented environment, especially for educational purposes as an instructor can initially create the world and populate it with class materials, for example: the class website, and data require for completion of assignments and projects. The users, once granted access to this space, can browse around much like they would a web browser. If container type objects, for example: ones that represent a directory in the back end source, are double clicked within CHIME, they would automatically create a new room for the user, extending from the current room, populated with all the objects mapped from those directories. In this way, both instructors and users can set up a hierarchy in the virtual environment that mirrors the original data source. And since the physics in the CVE does not have to match the physics of the real world, one can have an arbitrarily complex layout in the world. But the system would not limit visualization of only containers as rooms; one could set files to be containers as well where a file contains its application specific attributes. For example: an HTML file can contain links, images and text; a Word document could be comprised of images, tables, links, text, equations, etc. Given a corresponding FRAX plug for each of these elements as well as 3D objects to represent them with, CHIME could help visualize most any data type.

Objects in the world can be clicked on, moved to different spaces as well as right-clicked on to access more information. Moving objects from one room to another is possible and it does not affect the back end source. This is highly advantageous as an instructor could set up one space where all the students could meet, and it is here that the instructor could pull in materials from many data sources, giving the users access to all of them in one homogenous environment. Additionally, a user has access to a 2.5D overview map of the world in order to help navigate the world. To enhance the group experience there is not only a regular instant messaging system but also a group level chat system built in. The users are also provided with a history of their actions if they wish to revert back to a previous room. CHIME also provides additional unlinked doors in every room for users to link to other data sources.

User movement however was the most interesting aspect with respect to the VECTORS plugin as it employed a P2P model. The server knows where all the users are at any given time, but only at a room level granularity. And, as mentioned earlier, all communication between the server and the clients takes place over the event system. Since user position synchronization is a high frequency process, the publish/subscribe event system did not make for a good vehicle for this job, especially since the event system would add a large parsing overhead to each event that was as simple as coordinates in 3-space. We therefore do user synchronization using UDP packets on a peer-to-peer basis. The server sends every client an updated list of users in his/her room and the user would then in turn send position updates to each of those clients over a UDP stream. This peer-to-peer model is a proven one that works well for most commercial game systems [23] and we found it to give excellent results as well. This had the added benefit that is provided a broadcast stream over which we could also do our video synchronization.

4.2 VECTORS

One of our goals for CHIME was to integrate video synchronization for users. Columbia University offers taped courses over the internet as part of their Columbia Video Network (CVN) department. These courses work well when the class is simply lecture based geared towards individuals with assignments that do not require group work. However, for courses like Software Engineering and Operating Systems, where team based software development is one of the critical pedagogical requirements, CVN is unable to deliver a full experience, especially since the students registered for these courses are geographically dispersed. Teams of students may need to watch multiple class lectures together and collaborate on them as they are in progress.

Students are not required by CVN to have the same resources in terms of bandwidth. In order to facilitate synchronized video feeds to diverse users, we had to deliver pre-canned and pre-processed semantically structured videos over heterogeneous Internet links to heterogeneous platforms in an efficient and adaptive manner. Video thus becomes an additional legitimate resource for mutual exploration in a distributed team's workflow.

Approach:

Liu et al. [15] describe a similar project, however they are simply concerned with the QoS of the video and therefore their approach involves compression techniques working with Mpeg-7 video. Moreover, they do not have the added requirement of embedding their video stream in a CVE. Our approach involves semantic structuring of the video, using technology previously developed by Liu and Kender [16]. Given this rich video stream consisting of the most representative frames, in terms of content, of the video, our goal was to try and give each user the best possible set of frames in order to enhance the video watching experience as much as possible while staying synchronized. However, instead of following approaches like those employed in commercial multimedia applications like Real Player (http://www.real.com/) or QuickTime (http://www.quicktime.com/) that drop every n^{th} frame upon encountering network lag, which may have the negative side-effect of dropping important segments of the video, we procure separate levels of key frame density, each targeted at different bandwidth levels.

We still, however, have to give each client the correct video feed. In order to do this, our approach was four fold -

1. Pre-fetch as many of the key frames as possible at the highest possible quality to the client before a pre-determined meeting time for the group. Meeting times can be ascertained by probing the user's schedule or by simply getting this information from the student directly. Though, it turns out that most videos are watched impromptu without any prior notice.
2. Probe the clients' bandwidth and number of cached frames and report results to the system periodically.
3. React to bandwidth changes in real time by lowering/raising the client to a lower or higher quality feed.
4. Allow users to pause, rewind, etc. in synchrony while discussing the lecture material via "chat".

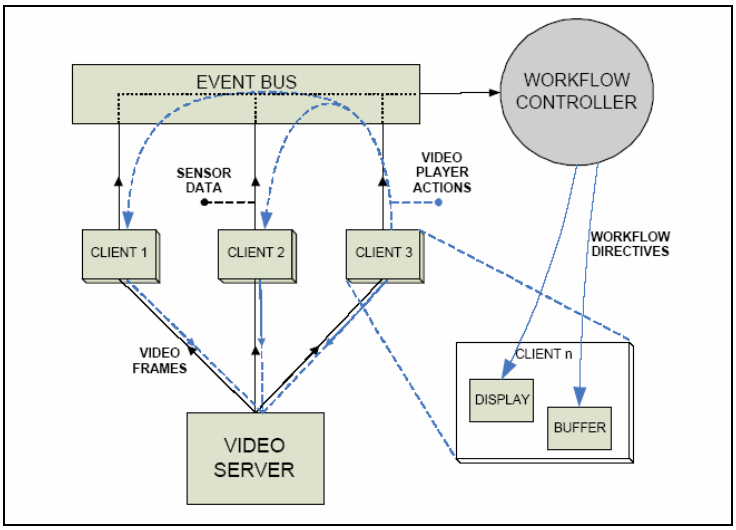

Figure 3 - The VECTORS Workflow

All the video streams are made available by the video server. Probing is done by using software probes [13] [14], and reports of any changes are sent to the respective clients. Each client receives data and based on how much video it has in cache, its current position in the video and its bandwidth, the client determines what the highest quality frame it can download next successfully before it has to view it; and downloads it. This will continue until the end of the video.

4.2.1 The server

VECTORS was proposed to analyze automatic methods for deriving semantic video structure, by finding large-scale temporal and spatial patterns, by detecting redundancies and semantic cross-correlations over long disjoint time intervals, and by compressing, indexing, and highlighting video segments based on semantically tagged visual sequences. We further explored user interaction in distributed environments in both a three-dimensional virtual world as well as a local two-dimensional client. We also analyzed various server cluster configurations, wire protocols, proxies, local client caches, and video management schemes.

The pre-canned and pre-tagged semantically structured video (Figure 4), was placed on the video server. Since the server simply provided the frames to each of the clients, the decision-making responsibility regarding synchronization fell upon the clients themselves; thus leading to a non-centralized decision-making system. The ultimate goal of the server was to analyze classes of particular server cluster configurations, wire protocols, proxies, local client caches, and video management schemes; however, in experiments, we simply treated the server as a black-box that would provide frames over an HTTP stream upon demand from a client. Example of a video frame hierarchy is shown in Figure 4, where we see two example levels of the same video stream. Level 1 has a sparse set of frames while Level 2 is denser, even though they semantically and pedagogically contain the same content. We would like to reiterate that audio was not semantically compressed and was therefore available as a separate and single file for the clients to download and play synchronously with the video stream.

Ultimately, the server consisted of two components, the semantically structured videos provided by Liu and Kender, and the scalable, proxy based video server. Since our goals lay in measuring the effectiveness of the video synchronization in the 3D virtual client, we set up a simple web-server that contained the structured video content and simply served it to the clients.

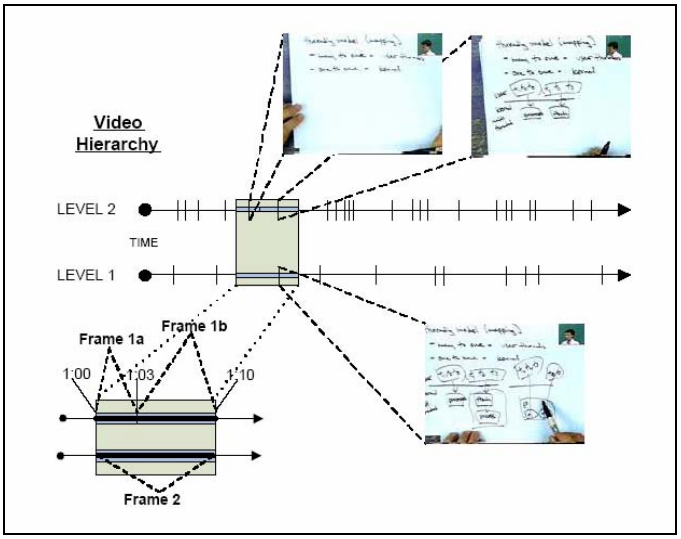

Figure 4 - Video Frame Hierarchy

4.2.2 The Client

The VECTORS Client Application, at the initial stage of development, focused on implementing, or at least making significant efforts to implement several functionalities which serve as the core of the VECTORS client side technology.

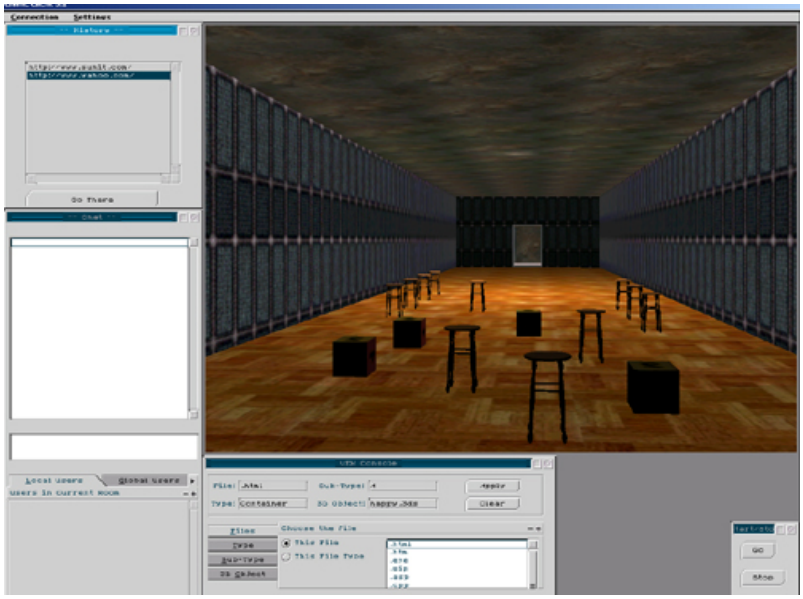

Figure 5 - CHIME world screenshot with objects populating a typical room

The client that we chose for video synchronization was the CHIME client as it provided the perfect pluggable framework that allows users to see each other in a collaborative world where they can interact with one another and objects that represent heterogeneous back end data sources. The CHIME client, as explained above, is an authoring tool and perfect for pedagogical environments.

Since CHIME had the ability to visualize heterogeneous data sources and was built as a framework, VECTORS was built as a plug-in that visualized video with the added component that synchronized the video. Some of the basic components added to the VECTORS plug-in are –

- **GroupWare Synchronization** – It provides a group-wide viewing session of a given video, each client remaining in sync with an overall video timeline. This is accomplished even if the various clients are at different network speeds (And thus are downloading a variety of different frames from the structured hierarchy that exists on the server).
- **Video Player in CHIME's 3D Environment** – The player is designed to work inside the existing 3D environment offered by CHIME. CHIME utilizes a Crystal Space graphics engine, and all aspects of the video player must comply with constraints set forth by Crystal Space to ensure error free, 3D video display.
- **Downloadable Video over HTTP** – The video components after being processed and placed on the server, consists of an audio stream (typically a highly compressed, low quality sampled MP3 file, though it could be WAV or other popular audio formats), and a set of JPEG images which correspond to frames of the video at different points in time. These components are retrieved from the Web either before the video is run (in which case they will be cached for use at runtime), or during runtime, at which point they are cached for later use. Therefore, the server, upon processing the video stream into these subcomponents, must publish them to a web server, along with some meta data (such as the number of "compression levels" and start/end times for each frame at each compression level)
- **Adjustable based on Bandwidth**- The client adjust its downloading strategy based on the available bandwidth, to switch to different compression levels offered by the server. A compression level is defined as a set of key video frames, a subset of the overall sequential list of JPEG images from a broken-down MPEG video, where each member of this subset is declared to persist over a specific time range.
- **Cache** – Videos, or portions thereof, that were previously downloaded should be stored locally for later use, in an effort to eliminate duplicate downloading. The cache should ideally store all levels of compression for a given video, and provide the best available compression level in response to any frame request. At the same time, the cache should abstract all methods of storage from the player, and simply provide the player with the location on disk of the JPEG frame file to play.
- **Cache Controller** – The intelligence behind the client that allows the users to stay synchronized.

4.3 Implementation Details

In order to get the system to work, we created a small UI within CHIME (see section 8 for figures) that activated a hook that we added into the 3D client. When activated, it would deploy a screen/portal on wall of the room that the client's avatar was in so as to display the downloaded frames within it. Each client was also gives a small cache where they could store pre-fetched video, several probes to monitor the

various variables that would control synchronization as well as a cache controller.

The probes included a cache monitor, a bandwidth monitor and a monitor that stated the exact location of the video a client was watching. These are software probes [13] that gather simple metrics and send them back to the cache controller for evaluation, over the publish-subscribe event notification system. As pointed out before, each client sends position updates via a UDP stream to all fellow clients in neighboring rooms so that fellow clients could render avatars in their respective accurate positions. The CHIME servers as well as the Video server note all the clients that start up any given video and assume that they are part of the same student group that wishes to watch the video. Updates about time index of the video that a client is watching is sent to all the other clients in the group.

Before the video even starts, the client tries to ascertain whether the user wishes to watch a particular video by looking up the workgroup calendar and starts to pre-fetch the highest density of frames from the video server so as to provide the best possible video experience. The pre-fetching module is the same component activated when a client pauses a video allowing the client to buffer the next few frames in the idle time.

The cache controller gets information about the contents of the cache, i.e. about the availability of extra frames in the timeline, as well as the position in the video and the current bandwidth (calculated by a simple ping to the server). The cache controller, since having already parsed the hierarchy of frames available in every compression level (gotten by downloading a pre-determined structured document about the frames), makes a decision about which frame to download next in the available time between current time and the time when that frame will be displayed based on available download. The cache controller also knows the duration for which each from will be displayed on the client's screen and uses this information to try and optimize on the level and density of frames to be downloaded. Any pauses by the client are simply utilized to download the highest quality and density of frames possible before the client restarts the video again.

CHIME clients synchronize with one another (peer-to-peer) by sending a time index in the UDP stream at least once every 0.33 seconds. Therefore, our aim was to keep the client always synchronized within 0.33 seconds of one another. If any client got out of sync with the others, the cache controller for that client would either instruct the client to lower or raise the level of frames that were being downloaded.

All VCR functions like play/stop and pause events were sent on the event bus since they were more major events that required action rather than just adjusting. They were also events that needed guaranteed action, something that a UDP packet cannot guarantee. All the clients play, stop or pause depending on the event sent out and acted on the same.

A workflow engine [26] is typically centralized and our workflow engine here had to keep the client in synchrony. Since that was the cache controller's job, the cache controller served as the workflow engine for this project. We found that even though the cache controller was decentralized, it provided us with good results because the logic control for each cache controller was the same. You can find results of our tests in Section 8.

5. Testing and Results

We used a test bed of up to 10 clients ranging from 400MHz laptops on a 56Kbit modem up to a 3GHz machine on a 100Mbit network. The resulting experiment kept the videos synchronized between all 10 clients within an error of approximately 4.38 seconds (for the first 7 minutes of the video), i.e. at no point was any client more than 4.38 seconds apart from any other. However, at this point, the system started showing more of a disparity especially on the laptops that do not have native 3D hardware support built in and therefore have to render the virtual environment in Software mode, thus slowing them down further. Figure 6 shows the extremely small variance between the various clients through the entire video while Figure 7 shows that even when we had a test bed of ten clients, they were essentially synchronized through the entire video content.

Note: Our lab has also developed an alternative "autonomic computing" approach to video synchronization for virtual classrooms, separately from CHIME but with better performance characteristics - it can support up to 10 clients within 10-15 milliseconds skew. See [6].

Some points to note during our test –

1) We started all the client's videos together. We did not attempt to have a client start significantly after the others to that it could "catch up" with the rest.
2) Our tests did not include any handheld devices. However, as long as a CHIME client would run on a handheld device and the PDA has internet connectivity, the synchronization should work in the same way.
3) We noticed that there was tremendous network congestion during the test. After investigation we found that the previously sparse traffic on account of the UDP streams had gone up tremendously. We found that since the position update events were relatively rare, when we used UDP streams for synchronization, the $O(n^2)$ streams (where n is the number of clients) with updates sent every 0.33 seconds from each client to every other client caused a substantial amount of traffic on the network.
4) We found the 3D client of CHIME to be an extremely heavy weight system that took up a lot of system resources on even the fastest machines used in our test. Therefore each system found it hard to cope with simple task like parsing of synchronization data.
5) Related to the above point, we found that the system stopped working after 7 minutes of run time on account of running out of system resources.

Overall, our results show that with the video synchronization works as well as the collaborative tools available. VECTORS was extremely dependent on the stability of CHIME. However, stability issues aside, the system made for an excellent environment for enriching the educational experience. In small lab tests, simulated groups could collaborate on videos well and since VECTORS operated on a highly configurable pedagogical environment, the groups were able to access relevant educational materials when necessary (or prompted by the video). VECTORS successfully supported *synchronized* viewing of lecture videos, and allowed VCR functions like pause, rewind, etc. to operate in synchrony while discussing the

lecture material via “chat”. VECTORS was successfully able to attend to the needs of the technologically disenfranchised, i.e. those with dialup or other relatively low-bandwidth networking.

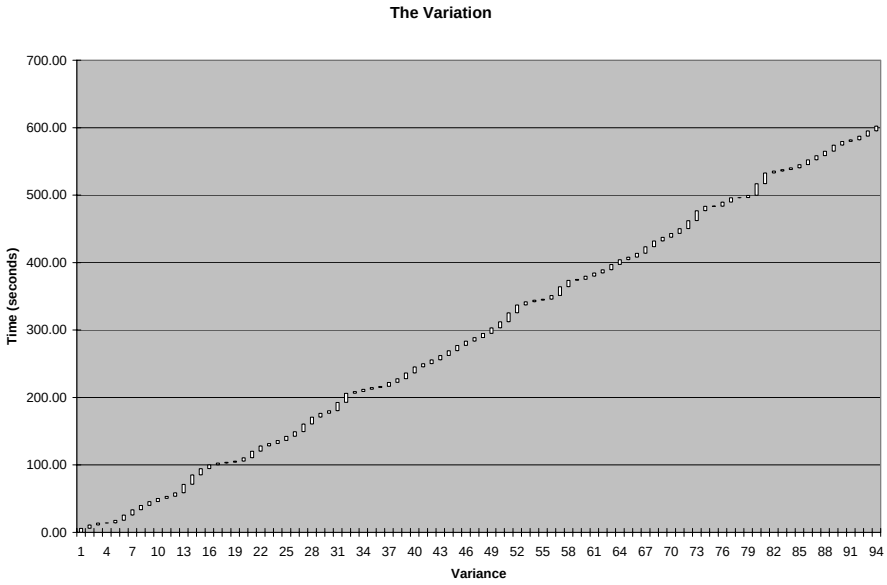

Figure 6 – Average variance between clients

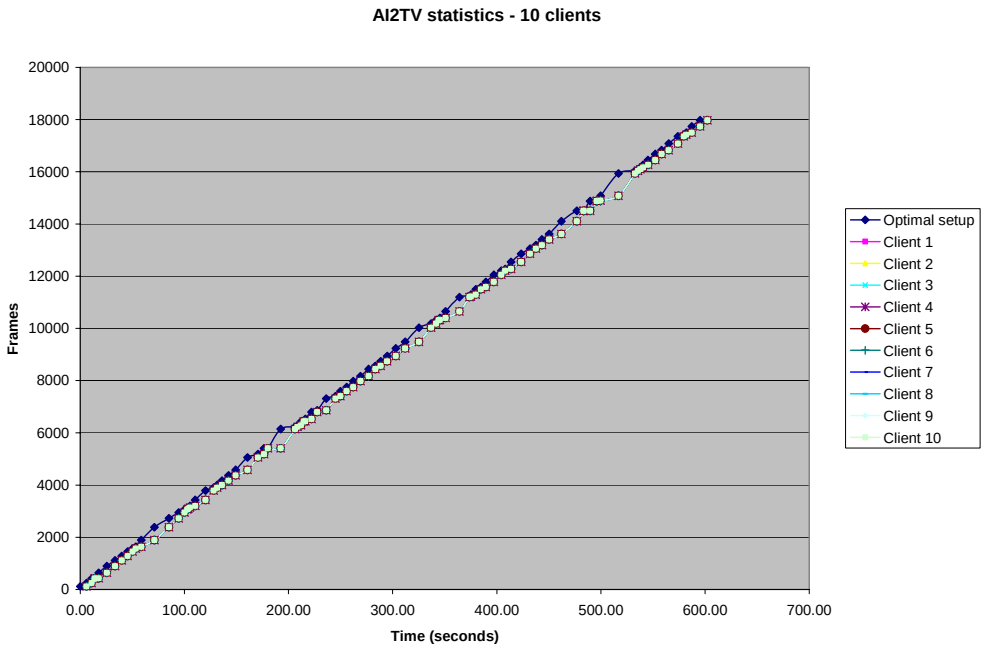

Figure 7 – Performance of 10 clients

6. Future Work

The 3D environment in its current form is extremely resource intensive in the form of the CPU and graphics card. This is mainly because of our choice of the 3D engine and the fact that we use dynamic

3D objects instead of static ones as the engine traditionally supports. One of our goals is to work with the developers of the engine to help alleviate some of the loads.

We will also be working on improving the way users view the state of their peers in the environment, so that they get a true graphical representation of which object the user is working on. Further work would also include increasing this granularity by allowing users, if given access, the ability to see what aspects of the objects the users are affecting.

We are currently using Siena as the event system that is able to pass about 400 events per second. We are in the process of moving over to the Elvin event system (http://elvin.dstc.edu.au/) that is an industrial strength event system that can transmit as many as 80,000 events per second thus significantly increasing the potential for scalability.

Additionally, introducing the concept of workgroups where users could log in and set up their schedules would greatly improve the video quality that users watch due to the pre-fetching features. Moreover, the workflow engine would be able to better track the groups.

Finally, our current system employs a very simple security model. We would like to move to one that is driven by an access control list.

7. Conclusion

We had presented a system, VECTORS, for the integration of lecture videos, with video synchronization, into a low-bandwidth virtual environment specifically designed for virtual classrooms for distance learning students.

This system has been designed as a plug-in to the previously developed collaborative virtual environment (CVE), CHIME, for small-group virtual classrooms. VECTORS uses a peer-to-peer synchronization approach to support group viewing of lecture videos. By utilizing this approach, we have found that groups of co-located or non-co-located students can work together on group based assignments. In order to cater to group members with low bandwidths, instead of going with traditional approaches that involve skipping every n^{th} frame of a video, VECTORS employs semantically compressed and pre-canned videos and adjusts the clients among various compression levels so that they stay semantically synchronized. The videos are displayed as a sequence of JPEGs on the walls of a 3D virtual room, requiring fewer local multimedia resources than full motion MPEGs.

As the results demonstrate, we have achieved a high degree of synchrony and have thus created a robust and useful pedagogical environment.

8. Figures

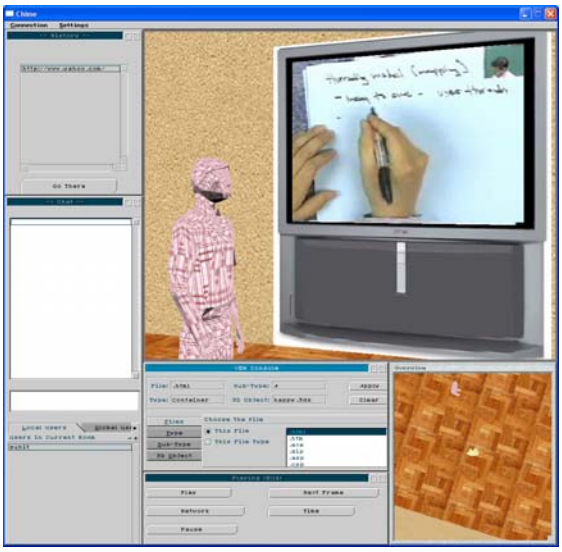

Figure 8 - VECTORS screenshot showing the video and team member

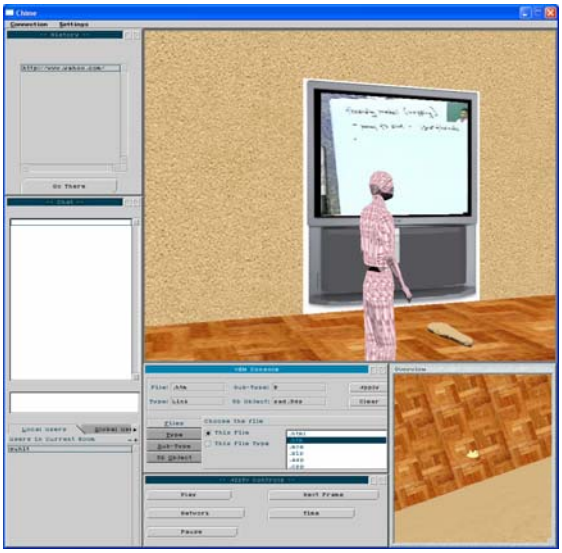

Figure 9 - VECTORS screenshot showing synchronized video and team member and objects

9. Acknowledgements

The Programming Systems Laboratory is funded in part by National Science Foundation grants CNS-0426623, CCR-0203876 and EIA-0202063, and in part by Microsoft Research.

10. References

[1] J. G. Wells, “Effects of an on-line computer-mediated communication course”, Journal of Industrial Technology, Vol. 37 Issue 3, 2000.

[2] Caplan, Priscilla. “You Call It Corn, We Call It Syntax-Independent Metadata for Document-Like

Objects", The Public-Access Computer Systems Review 6, no. 4: 19-23, 1995

[3] http://www.w3.org/AudioVideo/

[4] Christos Bouras, A. Philopoulos, Th. Tsiatsos, "e-Learning through Distributed Virtual Environments", Journal of Network and Computer Applications, Academic Press, July 2001.

[5] Christos Bouras, Dimitrios Psaltoulis, Christos Psaroudis, Thrasyvoulos Tsiatsos "An Educational Community Using Collaborative Virtual Environments". ICWL 2002: 180-191

[6] Dan Phung, Giuseppe Valetto, Gail Kaiser, "Autonomic Control for Quality Collaborative Video Viewing", Tech Report, 2004, Computer Science Dept., Columbia University – TR# cucs-053-04

[7] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. In 26th Applied Imagery Pattern Recognition Workshop. SPIE, October 1997

[8] Stephen E. Dossic, Gail E. Kaiser, "CHIME: A Metadata-Based Distributed Software Development Environment", Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, September 1999

[9] Stephen E. Dossick, "Groupspace Services for Information Management and Collaboration", PhD Thesis, Columbia University Department of Compupter Science, CUCS-001-2000, November 2000

[10] Stefan Fiedler, Michael Wallner, Michael Weber, "A Communication Architecture for Massive Multiplayer Games" Postion Paper, NetGames 2002

[11] Ekaterina Prasolova-Forland, "Supporting Social Awareness in Education in Collaborative Virtual Environments", International Conference on Engineering Education, 2002

[12] Ekaterina Prasolova-Forland, "Supporting Awareness in Education: Overview and Mechanisms", In proceedings of ICEE, 2002

[13] Philip N. Gross, Suhit Gupta, Gail E. Kaiser, Gaurav S. Kc and Janak J. Parekh, "An Active Events Model for Systems Monitoring", Working Conference on Complex and Dynamic Systems Architecture, December 2001

[14] Gail Kaiser, Giuseppe Valetto, "Ravages of Time: Synchronized Multimedia for Internet-Wide Process-Centered Software Engineering Environments", 3rd ICSE Workshop on Software Engineering over the Internet, June 2000

[15] J. Liu, B. Li, and Y.-Q. Zhang, "Adaptive Video Multicast over the Internet", IEEE Multimedia, Vol. 10, No. 1, pp. 22-31, January/February 2003

[16] Tiecheng Liu, John Kender, "A Hidden Markov Model Approach to the Structure of Documentaries", Workshop on Content-Based Access of Image and Video Libraries, 2000

[17] Masaya Okada, Hiroyuki Tarumi, Tetsuhiko Yoshimura, "Distributed virtual environment realizing collaborative environment education" Las Vegas, Nevada, United States Pages: 83 – 88, Pub. 2001

[18] C. Oliveira, X. Shen, N. Georganas, "Collaborative Virtual Environment for Industrial Training and e-Commerce", In Proceedings of Workshop on Application of Virtual Reality Technologies for Future Telecommunication Systems, IEEE Globecom 2000 Conference, 2000

[19] A. Carzaniga, D.S. Rosenblum, A.L. Wolf, “Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service”, Nineteenth ACM Symposium on Principles of Distributed Computing, Portland, Oregon. July, 2000

[20] John Kender, “Visual Interfaces to Computers: A Systems-Oriented First Course in Robust Control via Imagery”, CVPR '00 Workshop on Undergraduate Education and Image Computation

[21] N. Shivakumar, C. J. Sreenan, B. Narendran, P. Agrawal, “The concord algorithm for synchronization of networked multimedia streams”, International Conference on Multimedia Computing and Systems, 1995

[22] http://i-lab.mit.edu

[23] Steven D. Benford, David Snowdon, Chris M. Greenhalgh, Rob J. Ingram, Ian Knox, “VR-VIBE: A Virtual Environment for Co-operative Information Retrieval”, Computer Graphics Forum, 1995

[24] Thanasis Daradoumis, Fatos Xhafa, Joan Manuel Marquès, “Evaluating Collaborative Learning Practices in a Virtual Groupware Environment”, CATE 2003

[25] K. Rothermel, T. Helbig, “An Adaptive Protocol for Synchronizing Media Streams”, Multimedia Systems, Volume 5, pages 324-336, 1997

[26] Jason Nieh, Monica S. Lam, “A SMART Scheduler for Multimedia Applications”, ACM Transactions on Computer Systems (TOCS), 21(2), May 2003

[27] J. Liu, B. Li, Y.Q. Zhang, “Adaptive video multicast over the internet” IEEE Multimedia, 10(1):22-33, January/March 2003

[28] H. Liu, M. E. Zarki, “A synchronization control scheme for real-time streaming multimedia applications”, In Packet Video, April 2003

[29] Webex: Web conferencing, video conferencing and online meeting services. http://www.webex.com

[30] A. J. Gonzalez, H. Adbel-Wahab, “Lightweight stream synchronization framework for multimedia collaborative applications”, In 5th IEEE Symposium on Computers and Communications, July 2000

[31] D. Ferrari, “Design and application of a delay jitter control scheme for packet-switching internet works”, In 2nd International Conference on Network and Operating System Support for Digital Audio and Video, pages 72-83, 1991

[32] J. Escobar, C. Partridge, and D. Deutsch, “Flow synchronization protocol”, IEEE Transactions on Networking, 1994

[33] D. D. Clark, S. Shenker, L. Zhang, “Supporting real-time applications in an integrated services packet network: Architecture and mechanism”, In ACM SIGCOMM: Communications architectures and protocols, pages 14-26, 1992

[34] A. Campell, G. Coulson, F. Garcia, and D. Hutchison, “A continuous media transport and orchestration service”, In SIGCOMM92: Communications Architectures and Protocols, pages 99-110, 1992