

# The Software Architecture of An Interactive Remote Instruction System For Heterogeneous Network Environments

Ayman Abdel-Hamid, Sahar Ghanem, Kurt Maly, Hussein Abdel-Wahab  
*Department of Computer Science*  
*Old Dominion University*  
*Norfolk, VA 23529-0162, USA*  
*{hamid, ghanem, maly, wahab}@cs.odu.edu*

## Abstract

*At Old Dominion University, we have designed and implemented a Java-based distance education system, which we term IRI-h (for Interactive Remote Instruction-heterogeneous). IRI-h is designed to function on a number of heterogeneous platforms, and within heterogeneous network environments. IRI-h builds on the success of its predecessor system IRI, but attempts to avoid a number of identified pitfalls and deficiencies such as platform dependence, poor scalability, and the need for a homogeneous controlled network environment. In this paper, we present the design and software architecture of IRI-h which continues to offer audio, video, and tool sharing services within a synchronous virtual classroom environment, albeit in a platform independent manner. Furthermore, the proposed architecture accommodates class participants with no multicast capabilities, or limited connectivity bandwidth by offering tunneling, and rate adaptation services.*

## 1. Introduction

Over the past five years, the *Interactive Remote Instruction* (IRI) system has been used in the Department of Computer Science at Old Dominion University to teach undergraduate and graduate classes using university labs in distributed sites connected through a dedicated high bandwidth Intranet. The IRI system offers a synchronous virtual classroom environment, with audio, video, and tool-sharing capabilities [8]. Past experience with the IRI system has enabled us to identify a number of inherent deficiencies that limit its large-scale deployment. The main identified deficiencies were Unix platform dependence, limited scalability, and the need for a homogeneous controlled network environment [9].

The need for a multi-platform, multi-network environment scalable system prompted us to embark on

the design and implementation of a new IRI system which we termed *IRI-h* [5]. The “h” in the acronym stands for heterogeneous; to distinguish the fact that IRI-h is designed to run on heterogeneous platforms and within heterogeneous network environments. An IRI-h prototype [9] was fully implemented in Java [6] and has been tested on multiple platforms including PCs running the Windows operating system (NT, 98, 2000), and Unix machines running the Solaris operating System. The implemented prototype represents a base version that offers audio, video, and tool-sharing services within a homogeneous network environment. We are currently extending the implemented prototype to cater for heterogeneous network environments, in order to support class participants with limited multicast capabilities, or limited connectivity bandwidth, e.g., home users. Furthermore, a parallel implementation effort aims at providing recording and playback services [10] to allow for asynchronous learning possibilities.

In this paper, we present the design and software architecture of IRI-h. Section 2 presents a general design overview of IRI-h highlighting the IRI-h session concept and main design components namely, the session participant, and the session manager. Section 3 describes the session participant's software architecture along with relevant interaction with the session manager. Section 4 introduces the session manager's software architecture explaining how resource management, group communication, and late join are realized. Section 5 explains how the proposed architecture supports network heterogeneity by offering gateway services to specific session participants. Finally, the paper is concluded in section 6 with a status on the current implementation along with future work.

## 2. Design Overview

An IRI-h instance of a class is termed a *session*. A session consists of a single *session manager* (SM), and a number

of *session participants* (SP). The SM (see section 4) is a central server that represents a rendezvous point for SPs, and provides control information for all participants. The SP (see section 3) is a client running on each desktop participating in the session. The SM operates and manages a number of virtual rooms. A SP is a member of one virtual room at one instant of time. In each room, IRI-h services are available for SPs including audio, video, tool-sharing, annotation, and pointer services. IRI-h services use *shared resources* (see section 4) that are allocated and managed through the SM, e.g. group communication channels (see section 4.2). We classify a group communication channel as been unreliable, reliable, or semi-reliable. The current IRI-h prototype implementation offers a mix of unreliable and semi-reliable group communication. Unreliable group communication is provided through IP multicast [3] and is used for audio and video services. Semi-reliable group communication is offered through basic IP multicast, enhanced using a controlled retransmission policy, and is used for tool-sharing, pointer, and annotation services. In adopting a semi-reliable group communication model for data services, we rely on an eventual consistency paradigm, which improves the architecture support for heterogeneous network environments, and avoids the extra overhead if a reliable multicast protocol is used otherwise.

The SP and SM exchange control information through a *client-server approach* in which the SP connects to SM by means of a permanent TCP connection. A tradeoff has been made between reliability and scalability by eliminating any distributed servers that maintain considerable state information about the session. Only one SM manages a session, and is responsible for distributing any control or state information to all participants.

In a higher education environment, for example, some of the SPs might be located within university labs in high bandwidth multicast-enabled Intranets. Other SPs might have limited capabilities in terms of bandwidth connectivity, multicast-ability, or incurred delay, e.g. a student participating in the class from home, or from his work office. The latter SPs require the services of a *gateway* (GW) (see section 5) that offers tunneling, transcoding, and rate adaptation services.

A *Directory Server* can be deployed to offer lookup functionality for ongoing sessions. Each SM attempts to register with a designated directory server to enable such functionality. A SP can query the directory server to obtain a list of the current sessions, and actually join a selected session.

### 3. Session Participant

This section presents the SP software architecture (figure 1) along with SP interaction with the SM in the form of a

startup (see section 3.1), and a login protocol (see section 3.2). The SP operates a number of IRI-h services controlled through service managers. Supported services are audio, video, tool-sharing, annotation, pointer, and layout management. The *audio* service allows sending and receiving (playback) audio streams. The *video* service allows sending and receiving video streams. The *Java Media Framework* (JMF) [7] is used for both capturing and playing the audio and video of each participant. The *tool-sharing* service allows any participant to share any application with other participants. At the sender side, images of the windows in the application being shared are captured, compared to previous images to see if the image has changed, compressed, and transmitted over a group communication channel. At the receiver side, the images are received, decompressed, and displayed. Please refer to [4] for design details and performance results for the tool-sharing engine. The *Annotation*, *Pointer*, and *Layout Management* services are token-controlled services that can annotate, point to, or manage the layout of the SP's view of the current session, respectively.

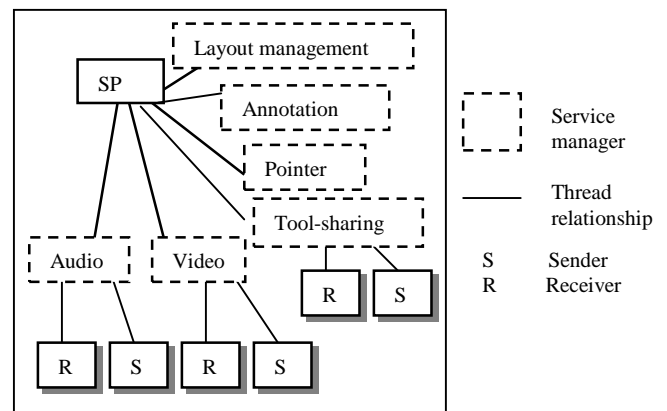


Figure 1: SP software architecture.

A SP has two views of the current session: a shared view, and a private view. The virtual room, that the SP belongs to, has a shared view that is consistent across all participants in that room. Changes made to the shared view by the presenter, the token holder of the layout management service, are propagated to other participants. The shared view consists of any received video windows, annotations, and shared tools' windows. In addition to the shared view, a participant has a configurable private view of private tools such as a collection of session monitors, and a note-taking tool. Session monitors include a "Class Monitor", a "Participant State Monitor", a "Log Viewer", and in the future a site camera view. The "Class Monitor" displays information about all members of the class, being currently logged in to IRI-h, or expected to participate in the session. The "Participant State Monitor" shows each service state at each logged in machine. The "Log

Viewer" displays output messages reported by all participants. Session monitors are provided with up to date state information through the session manager, which has a global view of all participants' state. The site camera view, operating at a low video frame rate, will provide a "site view" through a camera at each site.

### 3.1. Participant Startup Protocol

Figure 2 depicts the participant startup protocol executed between the SP and the SM. The startup protocol involves performing a multicast capability test to classify the SP, a *Round Trip Time* (RTT) calculation step, and a class monitor initialization step.

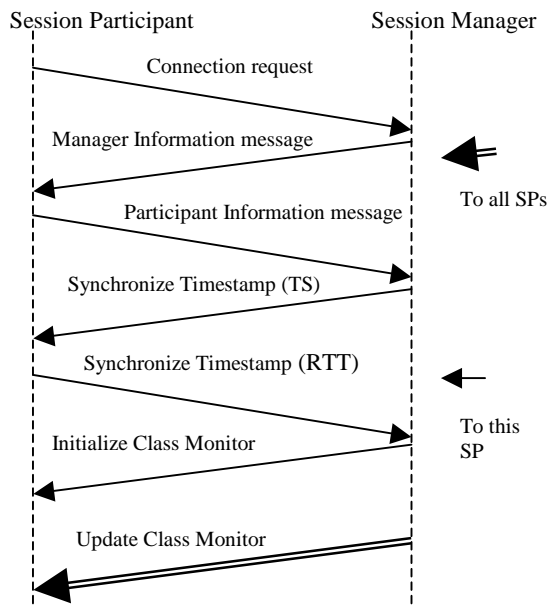


Figure 2: Participant startup protocol.

A SP starts by contacting the SM requesting to participate in the session. The SM sends the SP a *Manager Information message* notifying him of the IP addresses and ports of session servers including a list of one or more *Gateways*. In addition, the message includes the class name, semester, and the participant site. The SP contacts the *Gateway(s)* to perform the multicast capability test to choose its gateway server (see section 5), and sends a *Participant Information Message* to the SM that contains the multicast test result in addition to any participant configuration attributes such as being an audio server. The SP calculates its *RTT* and sends it to the SM to adjust its timestamp as follows: the SP after sending its information message gets the current time (*synch time*), and waits for the *Synchronize Timestamp* message from the SM, that contains the current time stamp. The SM sends the *Synchronize Timestamp* message after receiving the participant information message. Upon receiving that

message, the SP calculates its *RTT* to be equal to the difference between the current-time and the *synch* time, and adjusts its timestamp to be  $(TS + RTT/2)$ , and sends its *RTT* to the SM (for monitoring purposes). Next, The SM sends the SP an *Initialize Class Monitor* message that has the information about currently connected machines and participants expected to participate in the session. Afterwards, The SM sends an *Update Class Monitor* message to all SPs to update their class monitor state, adding the new participant.

### 3.2. Participant Login Protocol

Figure 3 illustrates the participant login protocol. After being authenticated by the SM, the SP is informed about the current started services in its current room. Meanwhile, session monitors are updated accordingly with the session's state.

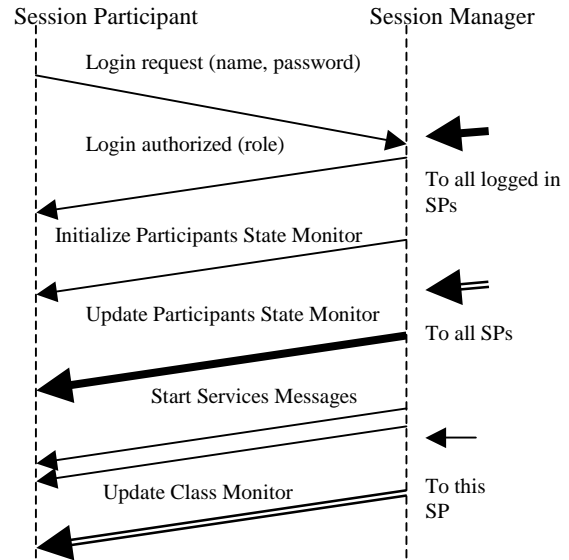
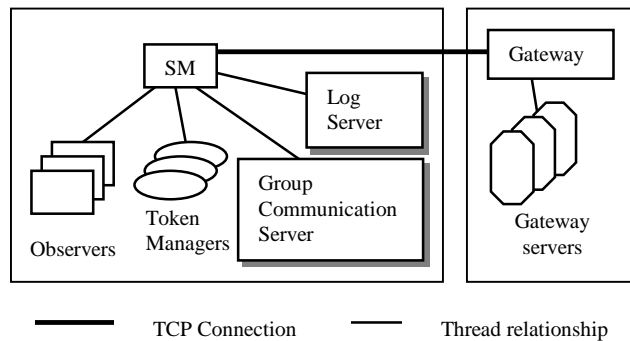


Figure 3: Participant login protocol.

First the participant sends a *Login* message that has his login name and password. The SM authenticates the SP and sends back a *Login Authorized* message acknowledging the login. The SM sends an *Initialize Participants State Monitor* message that has the state of all services of the currently logged in participants. Next, The SM sends an *Update Participants State Monitor* message to all logged in participants. The SM sends the SP a *Start Service* message to start all the services currently running in the SP's assigned room. After the participant has logged in, the SM sends an *Update Class Monitor* message to all connected machines to update the logged in participants state. Participants send every action affecting the state to the SM to take the corresponding action and update the *Participants State Monitor*.

## 4. Session Manager

Figure 4 illustrates the SM software architecture. The SM maintains the session's state including session rooms, and services running within each room. Furthermore, the SM allocates the resources needed by each service. Resources are observers, group communication channels, tokens, queue managers, or gateway servers. Observers are used to maintain the state of stateful services, such as annotation, and are used to provide a newly logged in participant with the relevant services' state, which helps solve the late join problem (see section 4.1). Group communication channels are used to transport each service's data streams, and are allocated by a *Group Communication Server* (GC server) (see section 4.2). Tokens are used within token-controlled services, and are controlled by *Token Managers* running within the SM machine (see section 4.3). Queue managers are used within video and audio services to manage the current audio/video senders (see section 4.4). Gateway servers (see section 5) provide tunneling and rate adaptation services to participants with no multicast capabilities, or limited connectivity bandwidth. Gateway servers are allocated on independent gateway machines, which can serve simultaneous ongoing sessions.



**Figure 4: SM software architecture.**

A "Log Server" is a SM component responsible for collecting all reported session messages (through SPs or SM) and saving them into a log file. The log file can be viewed from the SP interface through a log viewer service.

The SM can be invoked manually, or through an automated *session startup procedure* [9]. An IRI-h administrator pre-configures an IRI-h session from within a web browser through a Java Applet which in turn relays input configuration information to a backend Java server. When requested to start an IRI-h session, the Java server contacts an *IRI-h agent*, termed the *Host Ambassador*, residing on any of the Intranet IRI-h machines which in turn actually starts a SM, or a SP.

### 4.1. Late Join

While a session is actively in progress, participants joining the session need to know the current shared view state such as the presence of any video windows and their positions, and the presence of any annotations. Observers are threads running within the SM for pointer, annotation and layout-management services. The observer joins the service's data channel to receive every sent packet. Upon a late join (participant login), SM notifies all observers to send their latest state of each service. In order to avoid a *startup explosion problem*, a timer is set by the observer to guarantee a minimum period of time before honoring any new requests to resend the maintained state. The state is sent unreliably through the service's data channel.

The pointer service requires a *stateless observer* that keeps only the last snapshot of the pointer state, which is the last position of the pointer. On the other hand, annotation service requires a *stateful observer* that maintains the state of the latest annotation frame including any annotation objects and their respective positions within that frame. The layout management service requires a *window observer* that maintains the latest boundaries (x, y, height, width) of every uniquely identified window within the shared view.

### 4.2. Group Communication

A group communication API has been designed to provide group communication capabilities to IRI-h services in a uniform and transparent manner. Transparency implies that an API layer hides from upper code layers the implementation details of the communication channel. For example, the same annotation service code can be used unchanged for a multicast-enabled SP that utilizes a multicast-based group communication channel, or a multicast-disabled participant that utilizes a unicast communication channel.

A group communication channel identifier is a (textual name, implementation type) pair such as ("Room1-Video Group", "Unreliable Multicast"). The GC server maintains a database of mappings from a group communication channel identifier to networking entities such as a (multicast IP address, port) pair. When informed by the SM to create a new group communication channel, the GC server generates the associated networking entities and saves this mapping in its database. Later on, a SP connects to the GC server using a transient TCP connection in order to request the associated networking entities of a specific group communication channel.

Each service, requiring a group communication channel, is guaranteed a unique (multicast IP address, port) pair, across all running IRI-h sessions (if any), through the GC server allocation policy. To support

virtual rooms, each room is assigned a unique multicast address. This multicast address is used by all services within this room. It is formed as a function of the SM machine's IP address, SM server port, and the room number. The SM server port implies a *session identifier* (*Sessid*) according to the following policy: the SM upon startup tries to bind its server port to one of a fixed preset number of ports, the port index it succeeds to bind to is its *Sessid*. Assuming A.B.C.D is the IP address of the SM machine, the multicast address allocated to room  $n$  would be  $224+Sessid.n.C.D$ . Port numbers are allocated by subdividing a preset range of ports between a maximum number of allowed session managers per machine. Within each session, ports are divided between a maximum number of allowed virtual rooms per session.

### 4.3. Token Manager

A token manager is required for each token-controlled service. The SM allocates token managers, and provides a SP with their corresponding IP addresses and port. A SP connects to the token manager through a TCP channel. The token manager guarantees that only one participant is holding the token at one instant of time. The token holder is the only participant that is allowed to send data through the token-controlled service channel(s). A *stateless token* resource is allocated for pointer, and layout-management services. Alternatively, a *stateful token* resource is allocated for the annotation service. The maintained state, a frame number and a sequence number, is sent reliably from the old token holder to the new token holder. This information is crucial in ensuring the correct sequence of annotation packets, since the annotation packets are sent unreliably.

### 4.4. Participant Queue Management for Audio and Video Services

Queue-managers are used to manage participants currently transmitting their audio and video. This management is required because of the need to impose a physical constraint on the number of simultaneous video streams, and to limit the number of participants joining a discussion. The physical constraint on the number of simultaneous video streams arises from shared view landscape limitations, and the need to minimize the generated video streams' bandwidth. When there is no room in the queue, the first participant in the queue is forced to stop transmitting his audio/video and is removed from the queue. The current presenter, the layout-management token holder, is shielded from this forced remove until he is no longer a presenter. The current IRI-h prototype has a maximum video queue size of 3 allowing a maximum of three video windows to be present in the

shared view at one instant of time. Meanwhile, the maximum audio queue size is 10.

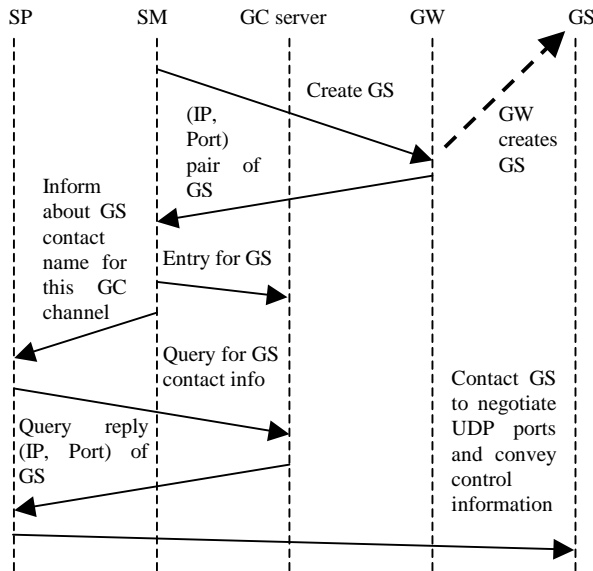
## 5. Architecture Support for Network Heterogeneity

The *Gateway* (GW) solves the network heterogeneity problem, manifested by varying connectivity bandwidths and different multicast capabilities available to SPs. This section explains how the IRI-h architecture supports multicast-disabled participants through gateway discovery, multicast capability classification, and group communication channel management. Although some SPs might be multicast-enabled, they might still require gateway support due to limited connectivity bandwidth. Extending the architecture support for multicast-enabled SPs requiring gateway service is left as a topic for future research.

A GW is responsible for classifying whether a SP is multicast-enabled or disabled by means of a multicast capability test. If this SP is multicast-disabled, a RTT measurement is conducted between the SP and the set of designated GWs for this session. The GW with the minimum RTT is selected as the candidate GW to service this SP. Other factors affecting the candidate GW selection are GW load, and network path bandwidth between the SP and each GW. Incorporating such factors in the candidate GW selection process is left for future investigation. The SP reports back to the SM the results of its multicast test, and if multicast-disabled the candidate GW. A GW listens to the related services' multicast streams, and forwards any received multicast packets to a multicast-disabled SP through one or more established UDP tunnels. On the other hand, any incoming unicast packets from the multicast-disabled SP are forwarded to the services' multicast channels.

For a multicast-disabled SP to receive IRI-h services, it needs to establish one or more UDP tunnels to its servicing GW. Hence, the group communication channel creation and query process outlined in section 4.2 is extended to accommodate multicast-disabled SPs to enable the creation of such UDP tunnels. For each new group communication channel, the SM informs the involved GWs to create an associated *Gateway Server* (GS). A GS provides the tunneling functionality for any multicast-disabled SPs receiving the associated channel. Moreover, the GS serves to negotiate UDP ports used within the UDP tunnel, and to exchange any control information between the GS and SP, such as the current allowed bit rate on this UDP tunnel. In order for the SP to discover the GS associated with a certain group communication channel, the GS contact information (IP, Port) is saved in the group communication server

database. Figure 5 illustrates a gateway-aware channel creation process.



**Figure 5: A gateway-aware GC channel creation, and UDP tunnel establishment.**

## 6. Conclusions and Future Work

In this paper, we presented the design and software architecture of IRI-h. A client-server architecture is adopted between a number of session participants and a central session manager for exchanging control and state information. In contrast, a scalable multicast-based group communication model is adopted for data streams within deployed services. Although a central server approach for session control reduces the robustness and fault tolerance of such system, it greatly simplifies the design, and achieves the desired level of scalability of supporting a maximum of hundreds of control connections to the session manager within a session. Furthermore, the proposed architecture caters for participants with limited multicast capabilities, or limited connectivity bandwidth by offering gateway services. At the time of this writing we implemented a complete base version of IRI-h providing audio, video, and tool-sharing services in a scalable, platform independent manner. Design Features described in this paper, but not yet implemented include virtual rooms, and gateway servers.

Future work is subdivided into a number of parallel tasks. We plan to implement gateway servers within the IRI-h architecture as described in this paper, and design video and tool-sharing gateway servers [2] that will perform the actual rate adaptation. In order to provide an efficient gateway bandwidth allocation strategy, a general

session monitoring and feedback architecture [1, 11] needs to be devised to report on the performance of each service, e.g., the number of dropped packets within a certain period of time. Moreover, we intend to research efficient approaches to present such feedback information in a useful, practical manner to IRI-h user community including teachers, students, and administrators. Furthermore, an inter-stream synchronization protocol is needed to synchronize the playback of received streams at each participant. In addition, we are currently pursuing the extension of our prototype implementation to provide recording and playback services within IRI-h to allow for an asynchronous learning environment. Finally, we plan to port to IRI-h a number of supporting IRI tools, e.g., a survey tool, a call student tool, and a site camera view [8].

## References

- [1] E. Amir, S. McCanne, and R. Katz, "Receiver-driven Bandwidth Adaptation for Light-weight Sessions", in *Proceedings of ACM Multimedia 97*, Seattle, WA, November 1997.
- [2] E. Amir, S. McCanne, and H. Zhang, "An Application Level Video Gateway", in *Proceedings of ACM Multimedia 95*, San Francisco, CA, November 1995.
- [3] S. Deering, "Host Extensions for IP Multicasting", Request For Comments (Proposed Standard) 1112, Internet Engineering Task Force, August 1989.
- [4] A. J. Gonzalez, H. Abdel-Wahab, and J. C. Wild, "Lightweight Scalable Tool Sharing for the Internet", to appear in *proceedings of the 6<sup>th</sup> IEEE International Symposium on Computers and Communications (ISCC)*, Hammamet, Tunisia, July 3-5, 2001.
- [5] IRI-h Home page, at URL <http://www.cs.ou.edu/~iri-h>.
- [6] Sun's Java Home page, at URL <http://java.sun.com>.
- [7] Sun's Java Media Framework (JMF) Home page, at URL <http://java.sun.com/products/java-media/jmf/index.html>.
- [8] K. Maly, H. Abdel-Wahab, C.M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica and E. Al-Shaer, "Distance Learning and Training over Intranets", *IEEE Internet Computing*, Vol. 1, No. 1, pp. 60-71, 1997.
- [9] K. Maly, H. Abdel-Wahab, C. Wild, C.M. Overstreet, A. Gupta, A. Abdel-Hamid, S. Ghanem, A. Gonzalez, and X. Zhu, "IRI-h, A Java-based Distance Education System: Architecture and Performance", to appear in the *ACM Journal for Education Resources in Computing (JERIC)*.
- [10] K. Maly, C. M. Overstreet, A. Gonzalez, M. Ireland, N. Karunaratne, "Experiences with Structured Recording and Replay in Interactive Remote Instruction", in *Proceedings of the 2nd International Conference on New Learning Technologies*, Switzerland, August 1999.
- [11] A. Youssef, H. Abdel-Wahab, and K. Maly, "The Software Architecture of a distributed Quality of Session Control Layer", in *proceedings of the 7th IEEE Symposium on High Performance Distributed Computing (HPDC-7)*, Chicago, IL, July 1998.