

# Using Session Initiation Protocol to build Context-Aware VoIP Support for Multiplayer Networked Games

Aameek Singh  
Georgia Institute of Technology  
Atlanta, GA  
aameek@cc.gatech.edu

Arup Acharya  
IBM T.J.Watson Research Center  
Hawthorne, NY  
arup@us.ibm.com

## ABSTRACT

Multiplayer networked games are the trend of the day. Receiving a major boost from various commercial ventures like Microsoft Xbox® [19] and Sony Playstation® [13], the networked gaming industry is set to grow dramatically. These multiplayer games allow geographically dispersed and possibly distant players to participate in a single game. In order to provide interaction amongst players in such environments, text messaging and recently, real-time voice interaction through VoIP is used. However, such interactions are mostly out-of-band (not based on game contexts), user-initiated and limited in operability, failing to exploit the entire potential and functionality of VoIP.

In this paper, we present mechanisms and design of a prototype that allows game-context based VoIP communication between players. Thus, in addition to allowing players to talk to each other to coordinate teammates and activities (through a static team-based audio conference) as in some of the current systems, it supports communication among players based on shared contexts like the same physical location or room within the gaming environment. We use the Session Initiation Protocol (SIP) [14] to realize VoIP and describe mechanisms for building network gaming services using SIP. We also propose a sophisticated gaming scenario, in which VoIP is used to relay information about another player's distance and location with respect to the recipient, e.g. players farther away sound farther away.

## 1. INTRODUCTION

The area of networked gaming has been growing steadily. With increasing network speeds and better infrastructural support, it is now possible to have players in opposite parts of the world participating in a single game. Various online services like Xbox Live [19], GameSpy Arcade [8], OMGN [11] and other independent game server hosts have been increasingly becoming popular. One of the important features of a good multiplayer game is the ability of players to interact with each other. Till recently, it was limited to only the game based mechanisms like shooting. The use of instant

messaging (IM) technology allowed players to send IMs to other players in the game, e.g., to coordinate with a teammate or to talk trash to an opponent.

The recent upsurge in Voice over IP (VoIP), primarily through better networking support and the development of an efficient and increasingly acceptable Session Initiation Protocol (SIP), brought about another valuable addition to the whole environment. Now instead of sending text messages, users can send and receive real-time voice. This allows for team members to continuously talk to each other and thus strategize and coordinate activities on the fly. This is achieved using VoIP conferencing (similar to regular telephone conferencing) and making teammates or all players participants of the conference. However, this mode of interaction is still limited in many ways.

- *Game Context Independent:* First and foremost, such inter-player interaction is game-context independent. This means that a player, Bob will always be talking to another player, Alice even when Bob is fighting demons miles inside the earth's crust while Alice is tackling aliens lightyears into the space. The gaming experience can be extensively improved if the interaction allows for taking the game-context into consideration, thus allowing players only in the same game room to interact, for example.
- *Requirements for User Initiation:* The underlying VoIP support is static in nature, requiring user initiation to change any current environment. For example, the switch from one audio conference to another, if allowed, requires the user to perform actions like pushing certain buttons on gaming console, whereas it is desirable to provide seamless transitions from one conference context to another.
- *Operability:* Many of the VoIP technologies being used bind users to particular consoles/audio devices, whereas it would be much better to allow a user to use any kind of VoIP device.

In this paper, we try to overcome these issues and also try to create a futuristic gaming environment. Specifically,

1. We describe the design and implementation of SIP based VoIP mechanisms to perform tighter coupling of voice communication with the game environment. We achieve this by enabling the game server (for a centralized scenario) to do SIP communication with a VoIP conference server or by directly enabling game clients to do this communication. We also discuss possible gaming and conferencing architectures.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM'04 Workshops, Aug. 30+Sept. 3, 2004, Portland, Oregon, USA.  
Copyright 2004 ACM 1-58113-942-X/04/0008 ...\$5.00.

2. Our framework provides for *seamless* switching of conferences based on gaming-contexts, without requiring any user involvement in the process. In addition, using SIP allows us to use a multitude of audio devices, thus uncoupling the game from the audio console.
3. We also present means of achieving “ad hoc” on-the-fly audio conferencing, which is responsible for the dynamic nature of audio sessions. Using such a feature allows us to extend gaming by allowing a group of participants to engage in a conference on-the-fly.
4. We also propose a sophisticated gaming environment, which performs an even closer integration by using enhanced audio mixing and provides much more accurate simulation of the real world scenarios.

## 2. SIP AND VOIP

VoIP refers to the technology that enables sending voice data over the IP network. The traditional Public Switched Telephone Network (PSTN) has a different backbone of infrastructure than the IP based internet, we are familiar with, and is only used to transmit audio data. On the other hand, VoIP enables transmission of audio on the IP network itself, which otherwise is primarily used to transmit data. This integration presents great opportunities for both service providers and consumers. While VoIP has various advantages like lower costs, greater consumer control and location flexibility, its typical problems have been Quality of Service (QoS) and standardization. However recently, with better infrastructure support, increasing network speeds and emergence of a widely acceptable and efficient protocol - SIP, VoIP is beginning to show its true potential.

SIP is an HTTP like protocol, which distinguishes between the process of a session establishment and the actual session. The session between two user agents (UAs) is established using SIP signaling mechanisms which involve sending an INVITE, an OK response and an ACK to the response [14]. These messages contain user parameters (using Session Description Protocol - SDP) for choosing appropriate IP address and port which will be used for actual data transmission as part of the session. This path for data is typically called *media path*, though any kind of data (even other than multimedia) can be transmitted. The IP/port combination can be for any networked device like an IP phone, game console or a PC. Typically, the media data is sent using RTP and signaling is accomplished using either TCP or UDP. The ability of a UA to accept certain encoding mechanisms is also negotiated through SDP as part of the signaling messages. Any of these parameters can be changed using the RE-INVITE message, which is identical to the INVITE message except that it can occur within an existing session. The session is terminated by using a BYE and an OK message. In addition, SIP allows UAs to refer a UA to another by using the REFER message. This instructs the UA to establish a session with the referred UA.

The UAs are identified by SIP URLs, which is a unique HTTP-like URL of the form *user@host*, for example, `sip:aameek@gatech.edu`. The mapping of the SIP URL to the appropriate physical UA device is done using intermediate SIP proxies, location and redirect servers, which form an overlay network. All UAs REGISTER with a SIP registrar server (can be at the SIP proxy itself), which maintains the address of the UA device. Then, all requests

for the SIP URL are routed to the appropriate device for that particular UA. An extension of SIP also supports SUBSCRIBE/NOTIFY mechanisms, in which UAs subscribe to certain events at another UA and can be notified whenever that event occurs. We refer the reader to [14] for more intricate details about SIP. The overall architecture of such an environment is shown in Figure-1 [17]. SIP was our automatic choice for VoIP because of its simple yet functionally rich design and its widespread acceptance.

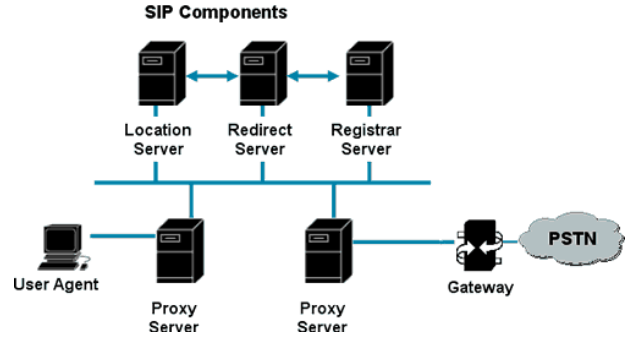


Figure 1: SIP based VoIP Architecture

## 3. ARCHITECTURE

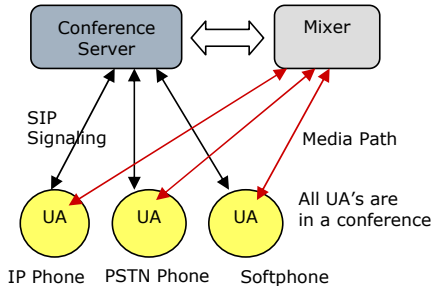
In this section, we will first discuss audio conferencing architectures using SIP based VoIP. Then, we describe the integration of gaming infrastructure with this underlying conferencing architecture. Note that this section only explains the components involved in the infrastructure and broad interactions between the various components. The protocol details and workflows will be discussed in Section-4.

### 3.1 SIP based Conferencing

The goal of audio conferencing is to allow multiple users to communicate in a group. Similar to regular PSTN conference, it means that every participant hears voices of all other participants. This indicates the need of a *media mixer*, which can mix (combine) voice signals from a set of users into a single signal for the recipient. Thus, for each participant the mixer will mix the voice signals of every other participant and transmit a single cumulative signal. The media is exchanged between the mixer and the participant on the media path of the SIP session. Audio mixing can be achieved both by specialized hardware devices or software mixers. There are various commercial off-the-shelf mixers available today like [3] and many of them are SIP enabled. They can perform SIP signaling and have specific provisions within SIP (like particular arguments within an INVITE message) for setting up resources for audio conferences.

In addition to the mixer, we also use a *conference server* (CS), which maintains state information about various participants and conferences. The CS is the controlling agent responsible for setting up sessions for every participant including establishing the media paths of the participants with the mixer. Using SIP, a conference is also identified by a SIP URL and in order to join a particular conference, a UA will send an INVITE for that URL which leads to the CS adding the participant into the conference. In our framework, the UAs do SIP signaling (establishing, changing, terminating sessions etc.) only with the CS, while sending and receiving

media from the mixer. This allows greater control at the CS, leading to better integration with gaming (as explained later). Note that it is also possible for the conference server to initiate a conference-join, thus inviting a particular UA to a conference (by sending the INVITE message). The process of the CS inviting a UA into a conference is called a *dial-out*. The overall conferencing architecture is shown in Figure-2, also showing various kinds of client devices which can be used as SIP UAs. We will discuss the exact SIP workflows involved in setting up a conference in Section-4. It is noteworthy to mention that there exist alternative architectures like those based on decentralized conferencing [5, 10]. We opt to demonstrate our ideas using the centralized server model for ease in exposition, though the ideas can be applied to other conferencing architectures.



**Figure 2: Multiuser Conferencing**

Using SIP as the vehicle for facilitating the context-aware VoIP support provides excellent interoperability allowing users with distinct audio devices to participate in a multiplayer game. For example, it is easily conceivable for the user to play a game, with the client hosted on a PDA and doing the voice interaction using a mobile phone interfaced through a SIP gateway (many providers provide such IP-PSTN gateways).

### 3.1.1 Adhoc Conferencing: URL Routing

In our architecture, a conference is identified by a valid SIP URL of the form *sip:conf-id@conf-server*, where *conf-id* is a unique identifier. The CS registers this conference URL with the SIP proxy and the SIP lookup mechanism allows user requests, to join a particular conference, to be routed to the CS. Typically resources can be reserved with the CS for a future conference to be held at a specific time. This allows the CS to register the conference URL with the SIP proxies, thus setting up appropriate routing of requests for that conference. Additionally, an interesting property of SIP proxies enables us to create adhoc *on-the-fly* conferences. An adhoc conference is one, in which users do not reserve resources in advance, rather, come up with a conference URL on-the-fly and invite participants into the conference. As described later, adhoc conferencing plays a key role in providing context aware VoIP and in addition, provides very interesting means of interaction in multiplayer gaming. The challenge in creating such conferences is to provide a mechanism in which the conference request, an INVITE message for the conference URL, is correctly routed to the CS, even when there is no exact entry at SIP proxies for that particular URL. This is achieved using a feature which allows SIP proxies to be configured to route SIP requests based on

domain names of the SIP URL (the CS in this case), when the exact URL is not registered. Therefore, even when the conference URL is not registered with the proxy, a message of the form *sip:conf-id@conf-server* is routed to the CS. The CS can then look up its current state information and act appropriately. If the conference does not exist, it can create a new conference, add the initiator as a participant and reserve resources with the mixer. In order for the CS to distinguish between contexts, it is important for each conference to have a unique URL<sup>1</sup>.

## 3.2 Integration with Gaming Infrastructure

There has been significant amount of research on multiplayer networked gaming infrastructures [2, 7, 15, 1, 6]. The models primarily fall into two categories – (i) centralized, and (ii) decentralized gaming. In this subsection, we present integration mechanisms for both the models.

### 3.2.1 Centralized Gaming

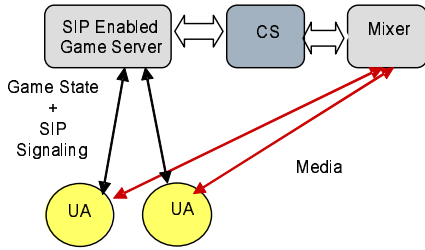
In *centralized multiplayer gaming*, there is a game server (GS), which maintains the entire *game state*. Players have local clients which are primarily responsible for displaying their game state and to communicate with the game server. The game client notifies the game server of any move/action the player takes and the game server is responsible for notifying every other *appropriate*<sup>2</sup> client of that move. The GS is also responsible for the entry and exit of players. Clearly scalability and reliability are critical issues for such an infrastructure. It has been tough to scale up the infrastructure for large scale multiplayer games and it always remains a single point of failure. However, it has many advantages as well. For example, it is much easier to set up a game server and coordinate between all the players. In addition, such an infrastructure has much lower risks of cheating.

To provide context-aware VoIP support, we need to be able to couple the game server with the conference server. The GS would now also need to maintain state about the players' audio sessions (which conference they are in) and appropriately coordinate with the CS whenever there is a need to adjust. This information can be derived from other game state parameters like location, teammates' position, shared contexts or can be an independent policy coded in the GS. We refer to this as the *audio session policy*. An example of a location based policy is that the gaming arena is divided into zones and every player in the same zone is a participant of a particular conference. When a player joins the game, the GS can set up its conference with the CS (depending upon the zone it joins in). Also, when a player changes its zone, the GS identifies it based on location parameters sent in the game state and automatically switches the conference to the one for the new zone. The switching is done using certain SIP signaling which provides a seamless transition. We discuss these workflows in detail in the next section. Recall that the media path would finally be set up with a media mixer; so the path for the entire connection would include of interaction of the game client with the GS, the GS with the CS, the CS with the mixer and finally the client's audio device with the mixer. The architecture is shown in

<sup>1</sup>The uniqueness of a URL can be guaranteed by using a system policy like a number appended to the username or global numbering

<sup>2</sup>It is possible that some move does not effect a particular client at all. The GS does not need to notify that client

Figure-3.



**Figure 3: Centralized Gaming Architecture**

Notice that the GS and the CS act as SIP back-to-back UAs<sup>3</sup>. This might slow down the control mechanism for heavily loaded game and conference servers. There are two possible ways in which this design can be optimized:

1. *Merge CS and Mixer*: If the mixer is dedicated to a single CS, it is feasible to integrate the mixer with the CS. This would allow using faster communication mechanisms between the two as opposed to network communication using SIP in the current architecture.
2. *Merge GS and CS*: Another possible alternative is for the GS to perform the functionalities of the CS as well. This would be especially useful for dedicated gaming services. Merging the GS and CS allows a closer integration of the two thus making it easy to maintain the game state and the audio session states in close proximity.

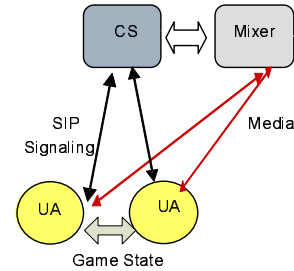
The choice of the optimization would vary with the needs of the application. For example, a dedicated mixer for a CS can be merged with the CS thus reducing the control hops, while it might be infeasible to do this if the mixer is shared across various conference servers. Also, the second choice would be infeasible if the CS is not dedicated to the gaming service only or if the gaming service decides not to complicate the GS, by utilizing a distinct CS.

### 3.2.2 Decentralized Gaming

As mentioned earlier, centralized gaming has scalability and reliability issues. There have been many decentralized gaming architectures proposed [2, 6, 9]. In general, the basic idea of decentralized gaming is to allow each client to communicate with every other *appropriate* client and the updates are exchanged using such direct interaction. In this scenario, the clients are aware of all the gaming rules and policies, which earlier were enforced by the GS. Using similar semantics, we code the audio-session policy within the game clients. Now, the clients will need to communicate with the CS themselves i.e. based on the policy and the game state of the clients, they perform SIP signaling with the CS, appropriately setting up their audio sessions. Again let us look at an example of a location based policy. In decentralized architectures the clients are made aware of the division of the arena into the zones. Thus whenever the client changes its zone, it updates its audio session within the game context.

<sup>3</sup>A B2B UA is a logical entity that receives requests from one party as a UA Server (UAS), responds to them by generating requests for another party, thus acting as a UA Client (UAC), and also maintains state information.

The architecture for such a scenario is shown in Figure-4 and the workflow is discussed in Section-4.



**Figure 4: Decentralized Gaming Architecture**

It is important to note that using a CS and media mixer might cause some centralization. However, there are two helpful facts. Firstly, the need to do SIP signaling with a CS arises only when a client changes its zone (or any other policy parameter requiring audio session transitions). Secondly, the media mixers are dedicated hardware components with the capacity of handling numerous conferences and participants. In addition, it is always possible to use decentralized conferencing architectures like those proposed in [5, 10]. We do not focus on such architectures in this paper.

Providing seamless dynamic conferencing in a decentralized gaming architecture requires an additional component at the game client. This is because of the requirement to shield the audio-session transition (ending one session and starting another) from the client. In the centralized scenario, the GS acting as a B2B UA provided this feature. We will discuss this in detail in the next section.

## 4. SIP WORKFLOWS

In this section, we discuss exact messaging and workflow details that are involved in the total infrastructure. First, we discuss the SIP workflow for setting up and joining a conference. Recall that, we provide appropriate routing mechanisms (by SIP proxy configurations), in which any SIP URL which does not have an entry in the registry is routed to the machine in the domain field of the URL (conference server, in our case). This allows us to have on-the-fly conferencing.

### 4.1 Multiuser Conferencing Workflows

The SIP workflow for a conference joining process is shown in Figure-5. The <sup>1</sup>INVITE message contains UA SDP indicating the media path information (IP/Port along with choice of encoding etc.). The CS extracts this media path information from the SDP and sends *that* as media parameters in the <sup>2</sup>INVITE message. This means that the CS is instructing the mixer to perform media exchange with the IP/Port of the UA, while doing signaling with the CS. The Mixer then sends its SDP in <sup>3</sup>OK. The CS extracts that info and sends it as part of SDP in <sup>4</sup>OK, thus informing the UA of the IP/Port of the mixer (for its media path). After the ACKs, media path is established between the UA and the mixer. Note that the CS needs to keep an open dialog with the mixer on behalf of every participant in the conference. It is through this open dialog that various session controlling messages can be transmitted (using SIP). For example, if a user wants to leave the conference, it can send a SIP BYE message to the CS and the CS sends BYE to the mixer

through the open dialog for that particular participant, advising the mixer to free resources for that participant and adjust conference audio mixing. In the workflow figures, the SIP signaling done by the CS always depicts the signaling done on the UA's behalf through this open dialog.

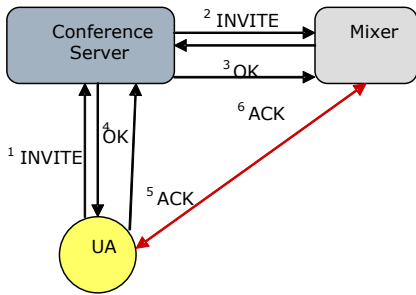


Figure 5: SIP Workflow for a Conference Join

## 4.2 Gaming Workflows

We distinguish between two main kinds of conferencing that can be used with multiplayer games - *Static team-based* and *Dynamic context-aware*.

**Static Team-Based Conferencing:** For static team-based conferencing, there will be one conference for each team. The GS maintains the conference IDs for all the conferences (it can be made same as the team name, thus keeping it unique). The first player to join a team will cause the GS to establish the conference context. We avoid reserving resources in advance, since SIP based conference creation process is extremely fast and does not hurt performance when the team is initialized. The complete SIP workflow for this process is depicted in Figure-6. This conference stays static throughout the game, i.e. the user is always part of its team's conference irrespective of the game state. A similar mechanism can also be used if the players wants to be able to talk to its opponents only, for example, when there are multiple players all playing solo against each other.

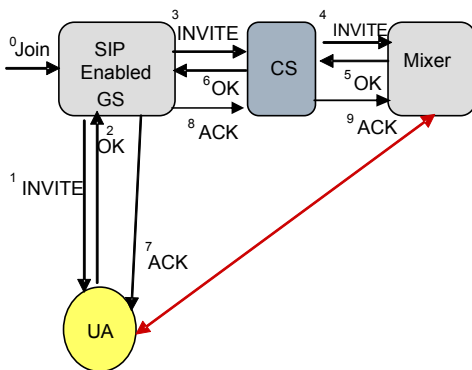


Figure 6: Game Join

When the user joins the game it sends a join message<sup>4</sup> to the game server. It also needs to specify the choice of its SIP audio device for the game. This can either be done

<sup>4</sup>Not a SIP message, but the regular message that initializes the game

through the join message itself or can be stored as a user profile characteristic at the game server. The GS initializes the game like it ordinarily does, but also sets up the audio conference accordingly. The message details are as follows. <sup>1</sup>INVITE invites the SIP UA to join the conference. The GS does not initialize the CS until the UA has accepted the INVITE. This is to prevent allocating resources when the user fails to respond. As a result, the mixer does not reserve any ports for this UA and the GS sends the first INVITE message *without* any SDP. The UA then sends <sup>2</sup>OK with its SDP, indicating the IP/Port of the UA device. The GS extracts the media path information from that SDP and sends *that* as media information in <sup>3</sup>INVITE. The CS completes the conferencing transaction with mixer and sends <sup>6</sup>OK with the appropriate SDP. The GS will then extract the media path information from that message and sends it as SDP in <sup>7</sup>ACK. It will then also ACK the CS which in turn ACKs the mixer. This completes the audio session initiation process. The audio session is terminated whenever the client leaves the game (which can be voluntary or forced by the GS based on the gaming policy).

Notice that it is easy to provide static conferencing in a decentralized gaming architecture as well. In the absence of the GS, the clients perform SIP signaling directly with the CS. The client directly sends an INVITE message, with its IP/Port info in the SDP, to the CS for the conference based on its team. The CS uses that IP/Port info to set up the media path with the media mixer. The workflow is similar to a regular multiuser conferencing (Figure-5) and we omit the details. The session is terminated when the client sends a BYE message while leaving the game.

**Dynamic Conferencing:** The static scenario is similar to what some of the current systems provide. Using our infrastructure, it is also possible to have context-aware dynamic conferencing. For example, players can talk to other players based on game contexts like same physical location, access to communication device like a phone booth in the game, a wireless handset etc. This is especially interesting for Role Playing Games (RPGs) [18, 12], where players portray a certain character and characters need to interact with each other. Rather than having a specified set of interactions, context-aware VoIP support can let them talk to each other and hence form on-the-fly alliances, betrayals, thus increasing the overall experience. The important issues in facilitating this infrastructure are:

- *Identifying Transition Points:* The game server needs to be able to identify when the audio session for a participant needs to be changed. This is accomplished through game state information. As mentioned before, whenever a player moves/takes an action, it notifies the game server of that action. Based on game and its audio-session policies, the game server will detect when it needs to change an audio session. It can then initiate the transition, which involves terminating the old audio session and starting the session based on the new context. For example, if a player changes a room in the game arena, its audio conference should now include players in the new room and not the old one.
- *Seamless Transition:* Another important requirement is that the transition of the audio conference should be seamless and an explicit user action, like pressing but-

tons on the console should not be essential. A seamless transition requires that there should be no termination of the session with the player's SIP UA (audio device). This is because if the session is terminated (using a BYE), then the creation of the new session would require the user to explicitly make a new call or accept an incoming call. However, our architecture is equipped to handle this situation efficiently. Since we use the game server to do SIP signaling with the CS on behalf of the UA, it can just change the media path information of the session (reflecting the new media mixing requirements for the participants of both old and new conference), without explicitly ending the session with the UA. The precise SIP workflow for such a seamless transition is shown in Figure-7.

Based on the new game state of the player, the GS can decide to switch the user's conference (indicated by the <sup>0</sup>Switch message). <sup>1</sup>BYE is the indication of the GS to the CS to remove the user from its conference. The CS, in turn, sends <sup>2</sup>BYE which instructs the mixer to free up resources allocated to that user and to change its mixing for that conference. After the BYEs are OK'd by the Mixer and the CS, the GS needs to initiate the process for the new conference. The GS first confirms<sup>5</sup> the media path from the device through <sup>5</sup>RE-INVITE. The UA sends its SDP in the <sup>6</sup>OK, which provides enough information to the GS to initiate the new session. The rest of the workflow is similar to the join procedure shown in Figure-6.

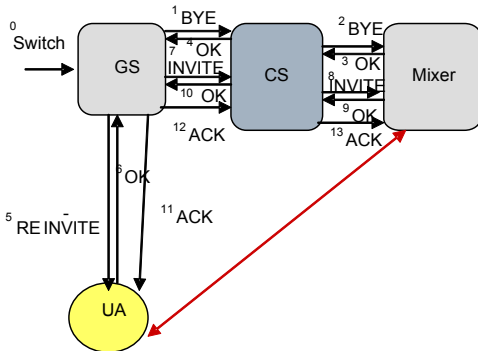


Figure 7: Audio Session Transition

Note that since a BYE is never sent to the UA, the UA does not have any session terminations. Just the new media path is negotiated and immediately set up. Our experience with a demo multiplayer game shows that this happens extremely fast and the user gets a seamless transition. We believe that using such an infrastructure can significantly improve the gaming experience.

Seamless transition is tough to achieve in the decentralized gaming architecture. This is because of the fact that while the GS hides the audio session termination (by acting as a B2B UA never sending a BYE to the client), there is no such entity to do this in the current decentralized scenario. We have two options to perform this shielding:

- *Modifying CS*: We can modify the CS to not send a BYE to the client when its audio session is supposed to

<sup>5</sup>Many SIP based audio devices change their media IP/Port information every time a new media path is set up.

*transition* and just do a RE-INVITE for establishing the change in media path. This requires the ability to distinguish between a transition and termination, in which case a BYE *has* to be sent. Originally we just simulated the transition by a termination followed by an initiation. From the onset this does not look like the ideal solution since it involves changing the semantics of the CS, which is just supposed to “coordinate” conferences and will also lead to dedicated servers for gaming.

- *Modifying Client*: A more reasonable solution is to modify the game client by adding a “light” B2B UA at the client itself. The SIP audio device of the client will communicate through this UA. We use the term “light” since it does not have to be a full SIP UA, just the ability to accept messages to initiate the audio-session transition and minimal SIP signaling capability to perform the transition. However, this could very well be a functionality of a full SIP based client service as proposed in [16] or a regular SIP UA. The architecture is shown in Figure-8 and the workflow is same as Figure-7 with the GS role being played by the light B2B UA. The only difference being the fact that the switch is now initiated by the client itself whereas in the centralized version, the GS initiated the switch.

Note that a modified client architecture can also be used for the centralized architecture when we wish to keep the GS independent of SIP. In that case, the client will exchange game state with the GS and perform SIP signaling, for initiation, transition and termination of audio sessions, with the client B2B UA.

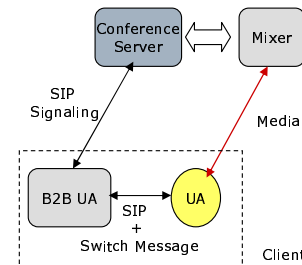


Figure 8: Decentralized Transition

## 5. USING ADHOC CONFERENCING

As mentioned before, an adhoc conference is one in which a user creates the conference on-the-fly without any apriori reservation of resources with the CS. In Section-3.1.1, we described how our infrastructure provides support for creating such conferences and allowing appropriate routing of messages. It is easy to see how adhoc conferencing can be exploited in general VoIP conferencing services. A user (say, a project manager) wishing to invite other users (project members) in a conference without prior notice, can create an adhoc conference and pass on the SIP URL through an Instant Messaging (IM) based service, allowing an immediate conference.

Now, we will describe two scenarios which explain the use of adhoc conferencing for multiplayer gaming and how it can enhance the overall gaming experience.

## 5.1 Handling Conference Creation

As detailed in previous sections, the GS is responsible for managing the audio sessions of all the players in the game. It coordinates with the CS and sets up appropriate conferences for all the participants. While it is possible for a GS to identify all possible conferences in advance (like the zone based conferencing in the location based policy), some audio-session policies can be very complex and dynamic making such an apriori identification difficult. As an example, consider a policy which says that in Room-A, players of type *demons* are in conference with players of type *aliens* and similar for numerous other type of players. As a result depending upon the game context, there can be a number of possible conferences just for Room-A itself. This makes it difficult for the GS to reserve resources for all possible conferences with the CS (which in the example scenario would include many combinations of player types). Adhoc conferencing plays a critical role in alleviating such issues. Using adhoc conferencing features, the GS can create a conference on-the-fly whenever the policy dictates a new conference. Thus, for our example scenario, there need not be any existing conference for Room-A and whenever two types of players (which should be in a conference, as coded in audio session policy) are in Room-A, the GS creates an adhoc conference (can guarantee a unique URL by using player types in the conference URL) and sets up players' audio sessions accordingly. Also, when one of the players moves out of that room, the conference can be terminated.

## 5.2 Providing Sub-Interactions

Our current infrastructure allows a game to define an audio session policy according to which, players are conferenced into appropriate context-based conferences. However, it will be interesting to allow a *player* to select a sub-group of players in the game and initiate a new conference. If such an interaction does not violate gaming policies, it can be an exciting mode of interaction. For example, assume a team based static conferencing for a simulated war game with two teams of armies. Allowing a player to talk to a select sub-group of players within the team allows for new modes of interactions providing for planning intra-team strategy (maybe, a coup). Also, if allowed it can be used to talk to members of opposite teams (gathering intelligence). This can lead to very interactive gaming, increasing the overall impact of the game. Since we cannot have any apriori knowledge of conferences in these scenarios, it certainly requires adhoc conferencing support.

Note that for facilitating the *Sub-Interaction* mechanisms, we will need to have interface support within the game. The enhanced interface should now allow a player to select and invite a subgroup of players into a conference, and depending upon game policies and players acceptance, the GS can create a new conference.

## 6. ENHANCED AUDIO MIXING FOR NEAR VIRTUAL REALITY EXPERIENCE

In this section, we propose an extension to the infrastructure, which further enhances the gaming experience. Notice that till now, all the proposed mechanisms assume a binary level of participation in a conference. Either a player is in the conference or is not in the conference. As a result, for a conference with  $n$  participants  $\{p_1, p_2, \dots, p_n\}$ , with audio

signals  $\{V_1, V_2, \dots, V_n\}$ , the signal received by a participant  $p_i$  at time  $t$  is given by:

$$R_i(t) = \sum V_j(t), \text{ where } 1 \leq j \leq n \text{ and } j \neq i$$

However, we can also think of scenarios where a more sophisticated evaluation of the signal can be better. For example, assume a soccer game in which each game player acts as one player on the field and all players are in a conference. Clearly, mixing which takes into account the distance of the speaker from the recipient would enhance the overall game. Then a forward would hear the fellow forwards and oppositions' defense the loudest, the midfielders a little less louder and not hear the defenders at all. Similarly, various other factors like voice shrillness can also be accumulated to form an overall feature vector  $\vec{X}$ . Now, the received audio signal can be given as the dot product of the feature vector with the voice vector (it will also be a vector, with each dimension corresponding to the value of attributes of the feature vector, like distance, shrillness etc).

$$R_i(t) = \sum \vec{X}_j(t) \cdot \vec{V}_j(t), \text{ where } 1 \leq j \leq n \text{ and } j \neq i$$

Such a scenario requires a sophistication at the level of audio mixing. Though currently available hardware mixers do not support a mechanism to associate a feature vector with the audio signals, software mixing can be easily enhanced to take such mechanisms into consideration. Also, it is feasible to relay this information using SIP only. For example, SIP INFO message<sup>6</sup> [4] can be used to convey the feature vectors. More precisely, whenever a player moves or takes an action, the GS would compute the new feature vectors for appropriate participants and send an INFO message to the CS/Mixer. To optimize on the communication overheads, only the minimal change information ( $\Delta$ 's) can be appropriately encoded in the message. We believe that such a scenario would be a poor man's Virtual Reality (VR) if not being very close to the actual VR experience.

## 7. PROTOTYPE IMPLEMENTATION

In this section, we describe the implementation of the prototype context-aware VoIP enabled multiplayer game, we developed at IBM T.J.Watson Research Center. The aim of the implementation was not to design an entire game, but just to demonstrate the context-aware VoIP support ideas. First, we describe the SIP infrastructure available at IBM T.J.Watson. It has a merged SIP proxy/registrar service and a SIP gateway which talks to PSTN to exchange calls between the IP network and PSTN. This allows using even a regular PSTN phone or a mobile phone as a SIP UA for voice communication.

The complete system architecture is shown in Figure-9. We developed the CS using Java. It uses an IBM built Java SIP stack and an overlying SIP interface layer which allows the control mechanism to interact with SIP using top level function calls. The manager component of the CS hosts the overall control logic and state information. It uses abstractions for a SIP conference and a SIP session as part of the state information. The manager is also responsible for administrative tasks like configuring pre-arranged conferences, showing the conference server state to the administrator etc. For gaming, we used the distinct game server architecture,

<sup>6</sup>INFO message is used to convey session related information on the signaling path.

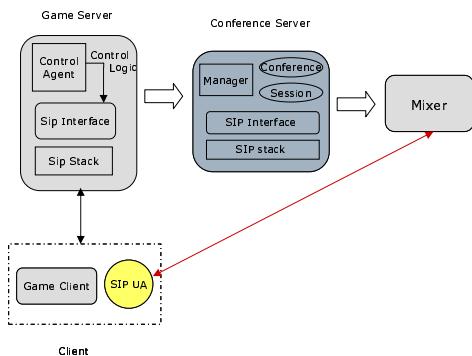


Figure 9: Prototype System Architecture

since the CS and mixers were being used for other VoIP services as well. The game server was enabled with the SIP stack and a SIP interface layer. It also hosts a control logic which defines the audio sessions for the players depending upon the game state. We used the Conveidia CMS-1000 media mixer [3]. It is a SIP enabled mixer and uses an INVITE message of the form `sip:conf=<conf-id>@mixer` to establish a conference context, where `<conf-id>` is an identifier for the conference [3]. The rest of the signaling is similar to regular SIP UAs.

The game we implemented defines a 2\*2 playing arena, with each block representing a game room and is considered a shared context. The audio session policy for the prototype was that all players in the same quadrant should be in the same conference. The game protocol was a simple state exchange mechanism, in which each player sends its co-ordinates (whenever there is an update) to the GS and the GS notifies other players of the update (to change their client displays) and also checks if any audio transitions needs to be made. The players are represented by different colored circles and the client interface consisted of displaying the arena, the position of players and a panel mapping the color of circles to player names. A screenshot at a client with 3 players in the game, is shown in Figure-10. The players can move in four directions - right, left, up and down. No other actions were defined for the prototype.

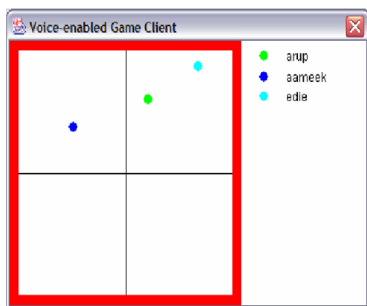


Figure 10: Client screen shot

For example, in Figure-10, players `arup` and `edie` are in a single conference and player `aameek` is not part of their conference. When `arup` moves to the left and enters `aameek`'s quadrant, it will be removed from the conference with `edie` and seamlessly transitioned into the conference with `aameek`. We tested the system with various SIP devices like PC soft-

phones, IP phones, PSTN desk and mobile phones. Our experience with the prototype implementation was very encouraging. The transitions were smooth and quick. In addition enabling servers with SIP turns out to be a relatively easy task.

## 8. CONCLUSIONS AND FUTURE WORK

In this paper, we have described mechanisms to provide context-aware VoIP support to multiplayer networked games. We use a SIP based conferencing architecture to support ad-hoc conferencing and integrate it with multiplayer games to accomplish context-awareness. SIP also provides us with the desired interoperability of audio devices. We have also proposed mechanisms to further enhance the gaming experience by using sophisticated audio mixing. We have developed a prototype of the system and it is found to have desired functional and performance characteristics. In future, we intend to look at the possibility for developing middleware solutions which can provide such generic context-aware VoIP support for a variety of gaming architectures.

## 9. REFERENCES

- [1] N. E. Baughman and B. N. Levine. Cheat-proof payout for centralized and distributed online games. In *Proceedings of IEEE INFOCOM*, pages 104–113, 2001.
- [2] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *NETGAMES*, 2002.
- [3] Conveidia. <http://www.conveidia.com>, 2003.
- [4] S. Donovan. The SIP INFO method. RFC 2976, Internet Engineering Task Force, 2000.
- [5] C. Elliot. A 'sticky' conference control protocol. *Internetworking: Research and Experience*, 1994.
- [6] S. Fiedler, M. Wallner, and M. Weber. A communication architecture for massive multiplayer games. In *NETGAMES*, pages 14–22, 2002.
- [7] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 209, 1995.
- [8] GameSpy. <http://www.gamespyarcade.com>, 2003.
- [9] L. Gautier and C. Diot. Design and evolution of mimaze, a multi-player game on the internet. In *IEEE Multimedia Systems Conference*, July 1998.
- [10] J. Lennox and H. Schulzrinne. A protocol for reliable distributed conferencing. In *NOSSDAV*, 2003.
- [11] Online Multiplayer Games Network. <http://www.omgn.com>, 2003.
- [12] The Role Playing Games Network. <http://www.roleplayinggames.net>, 2003.
- [13] Sony PlayStation. <http://www.playstation.com>, 2003.
- [14] J. Rosenberg et al. SIP: Session initiation protocol. RFC 3261, IETF, June 2002.
- [15] D. Saha, S.Sahu, and A. Shaikh. A service platform for on-line games. In *NETGAMES*, 2003.
- [16] A. Singh, P. Mahadevan, A. Acharya, and Z. Y. Shae. Design and implementation of sip client and network services. In *IBM Research Report - RC 23148*, 2004.
- [17] Terena. <http://gnrt.terena.nl>, 2003.
- [18] WebRPG. <http://www.webrpg.com>, 2002.
- [19] Microsoft Xbox. <http://www.xbox.com>, 2003.