



# Multiplayer networked gaming with the session initiation protocol

Aameek Singh <sup>a,\*</sup>, Arup Acharya <sup>b</sup>

<sup>a</sup> *College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332, United States*

<sup>b</sup> *IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, United States*

Available online 31 May 2005

---

## Abstract

With a strong push from commercial ventures like Microsoft Xbox<sup>®</sup> and Sony Playstation<sup>®</sup>, the multiplayer networked gaming industry continues to grow steadily. Multiplayer games allow geographically dispersed players to participate in a single game and in order to provide interaction amongst players in such environments, text messaging and real-time voice through VoIP is used. However, such interactions are out-of-band (not based on game contexts), user-initiated and severely limited in operability and functionality. In this paper, we present mechanisms and design of a prototype that uses the lightweight Session Initiation Protocol (SIP) to provide context-aware gaming. In addition to allowing players to talk to each other to coordinate teammates and activities (through a static team-based audio conference) as in current systems, it supports real-time communication among players based on shared contexts like the same physical location or room within the gaming environment. This is provided through seamless management of audio sessions (conferences) based on player movements/behaviors which change the shared game context. We extend our earlier work by providing a dynamic form of shared context, using a modified SIP session negotiation mechanism. In addition, through the use of SIP as the game communication protocol, we propose to make VoIP a first class member of the game state. This allows a *unified* architecture for context-aware communication and gaming. We also present 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.

© 2005 Elsevier B.V. All rights reserved.

**Keywords:** Multiplayer gaming; VoIP; SIP; Context-aware VoIP

---

## 1. Introduction

The ever growing popularity of the internet, with the multiplicity of services being offered has had a significant impact on the entertainment

---

\* Corresponding author.

E-mail addresses: [aameek@cc.gatech.edu](mailto:aameek@cc.gatech.edu) (A. Singh), [arup@us.ibm.com](mailto:arup@us.ibm.com) (A. Acharya).

industry. The area of multiplayer networked gaming, in particular, has seen dramatic growth. With increasing network speeds and better infrastructural support, it is now possible to have players on opposite parts of the world participating in a single game. As a result, many online gaming services like Xbox Live [23], GameSpy Arcade [9], OMGN [14], Playstation online [16] and various other independent game server hosts have become increasingly 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, hitting, etc. Later, the use of instant messaging (IM) technology allowed players to send instant messages to other players in the game, for example, to coordinate with a teammate or to talk trash to an opponent.

Overall, multiplayer gaming infrastructure has to deal with a plethora of networking issues. While traditional research has been on efficiency and performance issues, in this paper, we present architectural innovations which can significantly enhance the gaming experience. Our main contribution is to provide *context-aware* real-time voice communication amongst players through the use of Voice over IP (VoIP). A simple use of VoIP in the context of multiplayer gaming could be to allow team members to continuously talk to each other and thus strategize and coordinate activities on-the-fly. This can be achieved using VoIP conferencing (similar to regular telephone conferencing) and making teammates or all players participants in 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 daemons miles inside the earth's crust while Alice is tackling aliens light-years into space. The gaming experience can be improved if the interaction takes game-context into consideration.
- *Requirements for User Initiation*: The underlying VoIP support is static in nature, requiring user initiation to change any current environ-

ment. For example, a switch from one audio conference to another, if allowed, requires the user to perform actions like pushing certain buttons on the gaming console.

- *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 extend our earlier work [20] to overcome these issues and also try to create a futuristic gaming environment, which requires modifications to existing protocols and design mechanisms. Specifically,

1. We describe mechanisms to perform tighter integration of voice communication with the game. This can be done by enabling the game server to do SIP communication with a VoIP conference server (for a centralized scenario) or directly enabling game clients (for a decentralized architecture).
2. We also present means of achieving “ad-hoc” audio conferencing, which is responsible for the dynamic nature of audio sessions. Using such a feature allows us to extend gaming by allowing any group of participants to engage in a conference on-the-fly.
3. Our designed framework also 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.
4. We also describe a unified architecture which uses SIP as the game protocol itself, using its Publish/Subscribe mechanisms. Also, to provide greater flexibility in the kinds of conferences that can be created, we propose a simple modification to the SIP session negotiation mechanism. It allows participants to participate *one-way* into a conference (just sending voice input to the conference, but not getting any output streams). In addition, we describe a sophisticated gaming environment, which uses enhanced audio mixing mechanisms to provide a more accurate simulation of the real world scenarios.

The paper is organized as follows. In Section 2, we discuss the basics of SIP-based VoIP. We describe our proposed gaming infrastructure in Section 3. Protocol details and SIP workflows for the proposed design are presented in Section 4. We discuss the implementation details of a prototype multiplayer game with context-aware VoIP in Section 5. We describe a number of advanced features that extend the proposed mechanisms to create a futuristic gaming environment in Section 6. Finally, we conclude in Section 7.

## 2. SIP and VoIP

Session Initiation Protocol (SIP) is a HTTP-like protocol, which is fast becoming the defacto VoIP standard. It distinguishes between the process of a session establishment and the actual session. The session between two user agents (UAs) is established using signaling mechanisms which involve sending an INVITE, an OK response and an ACK to the OK [18]. These messages contain user parameters (using Session Description Protocol—SDP) for choosing an 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 the *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 (this session negotiation is discussed in more detail in Section 2.1). 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, and 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 discuss this in detail in Section 2.2.

### 2.1. Session negotiation

As mentioned earlier, SIP uses the Session Description Protocol (SDP) [11] to negotiate various session parameters between end-points. SDP is used to convey information about media streams in multimedia sessions to allow the recipients of a session description to participate in the session. It includes type of media (video, audio, etc.), the transport protocol (RTP/UDP/IP, H.320, etc.), the format of the media (H.261 video, MPEG video, etc.), remote address for media and the transport port.

When a UA gets an INVITE message with the SDP, its SIP UA replies back with an OK message containing the SDP with its own media information including acceptable media types. It is important to mention that typically SIP UAs do not set up a session if the two end-points do not agree on an acceptable media type/protocols. In addition, there is no provision for one-way communication, i.e. UA only sending media but choosing not to receive any. The default behavior of SIP UAs for this scenario is to reject the session establishment process. We propose extending this negotiation mechanism in Section 6, which allows a SIP session with a one-way audio stream, in order to optimize and provide more flexibility to dynamic conferencing in our gaming architecture.

### 2.2. SIP Publish–Subscribe

Another important characteristic of SIP that makes it well suited to the domain of multiplayer

networked gaming is its lightweight event notification mechanism using the SUBSCRIBE and NOTIFY messages [17]. Using these messages, a SIP UA can subscribe to certain events (defined within an *event package*) at another SIP UA and can be notified for the occurrence of these events. SIP allows for subscription expiration, subscription refreshing and unsubscribing for events as well. Publish–Subscribe has been a popular paradigm for multiplayer gaming [3] where clients subscribe to game events at the gaming server or other clients (decentralized). The notifications are used to carry the game state which is later displayed at the clients. We will describe in Section 6.3 how SIP Publish–Subscribe mechanisms can be used to communicate game-state to the participants. This will provide a more unified architecture for context-aware communication and traditional gaming.

### 3. Context-aware gaming infrastructure

In this section, we describe our proposed context-aware gaming infrastructure. We begin with the description of the underlying communication framework.

#### 3.1. Communication framework

The goal of audio conferencing is to allow multiple users to communicate in a group. Similar to a regular PSTN conference, it means that every participant hears voices of all other participants. This requires a *media mixer*, which can mix (combine) voice signals from a set of users into a single signal for the recipient. Audio mixing can be achieved both by specialized hardware devices or software mixers. There are various commercial off-the-shelf mixers available today like [4] 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 set-

ting 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 to the conference. In our framework, the UAs do SIP signaling (establishing, changing, terminating sessions, etc.) only with the CS, sending and receiving media from the mixer. This provides for greater control with the CS providing more 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 to a conference is called a *dial-out*. It is noteworthy to mention that there exist alternative architectures like those based on decentralized conferencing [6,12]. 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. Also, while we use a centralized mixer model in this paper, it is possible to have mixers at each UA and using SIP multicast between UAs. However, such an architecture does not scale well and would substantially increase the complexity of our gaming consoles.

##### 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, 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. 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 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. Gaming architectures

There has been a significant amount of research on multiplayer networked gaming infrastructures [3,8,19,2,7]. The models primarily fall into two categories—(i) centralized and (ii) decentralized gaming. In this subsection, we present integration mechanisms for both the models. Later in Section 6.3, we will describe a unified architecture for gaming which uses SIP as the gaming protocol.

#### 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 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 difficult 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' positions, 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 either instruct the player to join a particular conference at the CS (depending upon the zone it joins in), or instruct the CS directly to invite the user into an appropriate conference using a dial-out. 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 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 Fig. 1.

Notice that the GS and the CS act as SIP back-to-back UAs.<sup>3</sup> This might slow down the control

<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 moves does not affect a particular client at all. The GS does not need to notify that client.

<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.

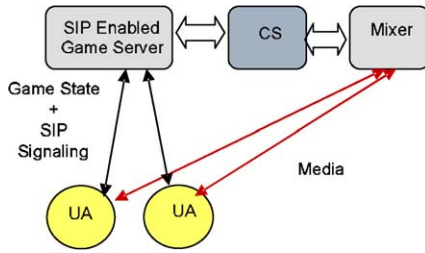


Fig. 1. Centralized architecture.

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 [3,7,10]. 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 zones. Thus whenever the client changes its zone, it updates its audio session within the game context. The architecture for such a scenario is shown in Fig. 2 and the workflow is discussed in Section 4.

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 [6,12]. 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 provides this feature. We will discuss this in detail in the next section.

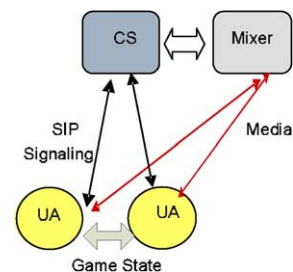


Fig. 2. Decentralized architecture.

## 4. SIP workflows

In this section, we discuss exact messaging and workflow details. We distinguish between two main kinds of conferencing that can be used with multiplayer games—*Static team-based* and *Dynamic context-aware*.

### 4.1. 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 the 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 Fig. 3. 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 want to be able to talk to its opponents only, for example, when there are multiple players all playing solo against each other.

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 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 first INVITE message contains *no* 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

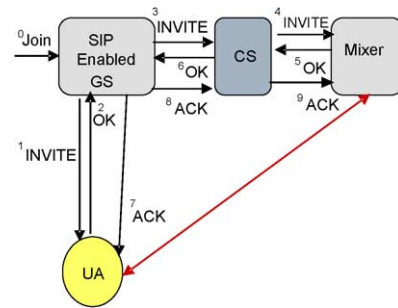


Fig. 3. Game join.

<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. 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 (Fig. 3) and we omit the details. The session is terminated when the client sends a BYE message when leaving the game.

### 4.2. 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) [22,15], where players portray a certain character and characters need to interact with each other. Rather than having a

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

specified set of interactions, context-aware VoIP support can let them talk to each other and hence form on-the-fly alliances or 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 buttons 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 Fig. 4.

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

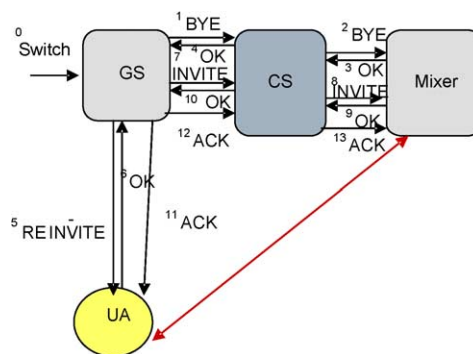


Fig. 4. Audio session transition.

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 Fig. 3.

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 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 *transition* and just do a

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

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 “co-ordinate” conferences and would 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 would 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 [21] or a regular SIP UA. The architecture is shown in Fig. 5 and the workflow is the same as Fig. 4 with the GS role being played by the light B2B UA. The only difference is 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 the SIP signaling, with appropriate messages for audio-session initiation and transition, with the client B2B UA.

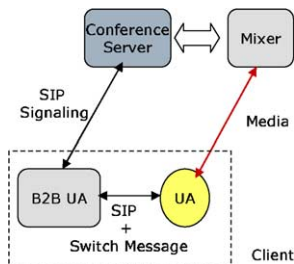


Fig. 5. Decentralized transition.

## 5. A proof-of-concept prototype

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 or to perform stress testing of the infrastructure, rather just to demonstrate context-aware VoIP design ideas using standard off-the-shelf VoIP equipment. The prototype also helps in understanding the steps involved in enabling game servers with context-aware VoIP.

The SIP infrastructure at IBM T.J. Watson consists of a merged proxy/registrar service and a gateway which interfaces with the PSTN network. This allows using even a regular PSTN phone or a mobile phone as a SIP UA for voice communication. All components are connected over a local area network.

The complete system architecture is shown in Fig. 6. 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.

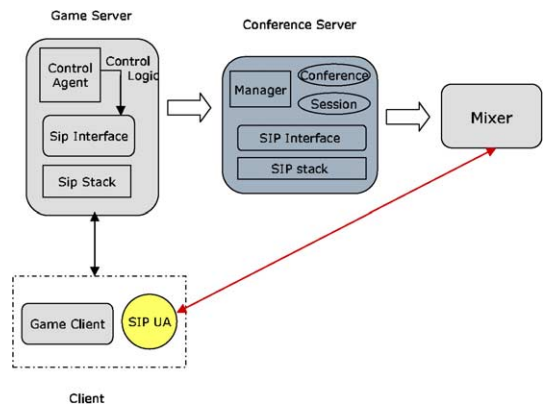


Fig. 6. Prototype system architecture.

For gaming, we used the distinct game server 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 Convedia CMS-1000 media mixer [4]. 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 [4]. 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 three players in the game, is shown in Fig. 7. The players can move in four directions—right, left, up and down. No other actions were defined for the prototype.

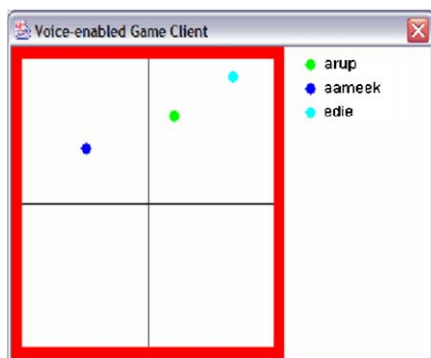


Fig. 7. Client screen shot.

For example, in Fig. 7, 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 softphones, IP phones, PSTN desk and mobile phones. Our experience with the prototype implementation was encouraging with very smooth transitions.

As mentioned earlier, we did not evaluate the infrastructure quantitatively for its performance. The efficiency and performance measures vary significantly with the design of the game and the behavior of the players (frequency of audio session transitions), while the motivation of this work is to provide architectural design ideas. We are currently working on extending a popular multiplayer game, Quake, with the proposed basic context-aware VoIP and to quantitatively evaluate its performance.

## 6. Extending the gaming infrastructure

In this section, we discuss a number of enhancements that can improve user experience. Unlike earlier proposed mechanism, these features require extending existing standard protocols and possibly, user equipment. These features can be seen as the building blocks of a futuristic gaming environment which aims at providing highly interactive multiplayer gaming.

### 6.1. Modified session negotiation for enhanced adhoc conferencing

In the discussion so far, the audio session policy always associated a participant of the game with a single conference. For example, using the location-based policy, conferences are created per each *room* and all players in a single room are part of a conference. However, consider a general dynamic version of gaming where the whole gaming field is not statically divided into rooms, rather a player can hear all other players within his/her audio range. It is important to appreciate the

difference between this dynamic room conferencing and the earlier described versions.

In this case, each player has to participate in multiple conferences. There is one conference, called *Own-Conference* or *Type 1*, for combining the audio signals that the player, say Player A, is going to receive and additional conferences, called *Other-Conference* or *Type 2*, for the players who can hear Player A (i.e. the players who have Player A in their audio range). Notice that the useful *output* audio stream for Player A is only from the Type 1 conference, since that will contain the actual audio signal Player A should hear. The output streams from Type 2 conference, containing a mix of audio signals participating in that conference, as described earlier, are not required. There are two ways to support such enhanced adhoc conferencing:

1. *Filtering at the Client*: One way is allow a client to be a part of multiple conferences using regular SIP mechanisms with each conference having different port information in the SDP in order to allow different conference outputs to be sent to different ports. However, all packets from Type 2 conferences are filtered out and only packets from Type 1 are sent to the audio device. Clearly, this leads to inefficient use of bandwidth and media mixer resources.
2. *One-way Conferencing*: A more optimal way is to allow players to participate in Type 2 conferences as one-way participants. In such a mode of participation, the player only transmits its audio signal to the conference and does not receive the audio output of the conference. This provides efficiency in bandwidth, since only useful packets are transmitted over the network and also prevent media mixer for mixing useless audio signals.

Recall that SIP UAs require a session to be equivalent in media characteristics in both directions and in case of a failure to agree on a single format, the session is not set up. Thus, in order to provide one-way conferencing we have to enhance the session negotiation mechanism. In this enhanced negotiation, a UA can send its acceptable media characteristics for *each* direction of the channel,

in its SDP. In addition, NO-MEDIA can be selected as part of the SDP media characteristics. As long as both UAs agree on both directions and there is atleast one direction with non-zero communication (both sides did not opt for NO-MEDIA) the session is set up. For our gaming scenario, each player would select the usual duplex conferencing mode for its Type 1 conferences and the one-way mode (with player to GS link set as NO-MEDIA) for its Type 2 conferences. Note that the one-way conferencing would also require enhanced audio mixing, an issue which is also discussed in Section 6.2. Note that the modified negotiation mechanism would not work for multicast-based conferencing architectures where SIP UAs open only a single channel for all communication and filter locally at clients.

Using such an enhanced adhoc conferencing mechanism can cause additional load on the mixers (an increase in the total number of conferences per player). However, it can be optimized by creating distinct conferences for players only if they are more than a pre-defined threshold distance away. Such an infrastructure would lie within the two extremes of static room-based scenarios and completely dynamic conferences as discussed in this subsection.

We believe this extended SIP negotiation mechanism to be useful even outside the gaming domain. For example, consider a user with camera-enabled handheld device using SIP to communicate with a base station. The user wants to transmit high resolution video, though only receive low resolution video in order to conserve energy in rendering. This requires setting up different media parameters in two directions. However, until such an extension is accepted as SIP standards, one-way conferencing would require dedicated communication between SIP UAs.

## 6.2. Enhanced audio mixing for near virtual reality experience

Notice that until 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

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

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.).

$$R_i(t) = \sum \vec{X}_j(t) \cdot \vec{V}_j(t),$$

where  $1 \leq j \leq n$  and  $j \neq i$ .

Such a scenario requires 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> [5] 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.

### 6.3. SIP as the gaming protocol: a unified architecture

Until recently, SIP has been primarily used as a vehicle for carrying VoIP or some presence information in IM technology. There have been recent efforts which are trying to explore SIP in other areas [21] including recent work which uses SIP to provide a mobile gaming platform for 3GPP IP Multimedia Subsystem [1]. The attractive features of SIP as a gaming protocol are its lightweight event notification mechanisms, its ability to negotiate gaming parameters (which offloads the responsibility from the game developer) and its operability with a wide variety of audio devices.

Using SIP as the gaming protocol provides a unified architecture where the regular game state and voice data is exchanged using the same protocol. We believe this to be a valuable feature for game developers who want to incorporate context-aware gaming. Also, it will substantially ease the development of gaming consoles to be used for context-aware gaming by reducing it to a SIP UA with an appropriate event package that renders the game state.

This proposed extension can also be viewed as attempting to make real-time voice a first class member of the game state itself. To illustrate this, consider a game where a player's state is defined by a six-tuple of the form:  $\langle t, x, y, z, health, weapon \rangle$  where  $t$  is the current time,  $x, y, z$  are the 3-D coordinates,  $health$  is the current health and  $weapon$  is the current weapon chosen by the player. When the player's state changes, the GS will notify other players of the modified state. We propose extending this state information by including a *voice* component. Any new audio input is considered as a state modification and the new state is notified to the other players. Thus, context-aware communication is more intricately tied in with the game.

Facilitating such gaming is relatively straight forward using SIP event notification. For example, joining a game is equivalent to establishing a session with the GS (for a centralized scenario) and other players (for a decentralized scenario). After a user has successfully joined a game, it sends a SIP SUBSCRIBE message to all events within its range. This will include any voice communication

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

related events. Whenever such events occur—a change of player location, new audio input—the GS sends a NOTIFY message to the player describing those events.

Note that adding such voice communication to the gaming infrastructures increases competition for valuable resources. We intend to explore this in detail by porting a complex multiplayer game, Quake, to the proposed SIP infrastructure. We believe that it should be possible to design an optimized conferencing architecture with distributed audio mixing to alleviate these issues to a certain extent. However, it should be evident that for less intensive games (with smaller number of players), the proposed infrastructure is a promising approach to the next generation of interactive multiplayer gaming. For intensive games, another possibility is to use SIP only to set up event stream channels and use interactive media protocols like RTP/I [13] to exchange status updates. This approach would have lower overheads than the one previously described.

## 7. Conclusions and future work

The Session Initiation Protocol is fast becoming the protocol of choice for media communication. Its widespread acceptance for VoIP and IM technologies provide an ideal opportunity to expand it further into other areas like multiplayer networked gaming which share similar requirements. In this paper, we have described mechanisms to provide context-aware VoIP support to networked games using SIP. We use a SIP-based conferencing architecture to support adhoc conferencing and integrate it with multiplayer games to accomplish context awareness. SIP also provides us with the desired interoperability of audio devices.

We also propose a unified architecture in which SIP is also used as the core gaming protocol. In addition, we describe mechanisms to enhance adhoc conferencing by modifying SIP session negotiation mechanisms. 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 the desired functional and perfor-

mance characteristics. In the future, we intend to look at the possibility of developing middleware solutions which can provide generic context-aware VoIP support for a variety of gaming architectures. In addition, we plan to port a complex multiplayer networked game to a unified SIP architecture with context-aware VoIP communication.

## Acknowledgment

We thank Priya Mahadevan and Zon-Yin Shae for their help in the design and implementation of SIP-based VoIP conferencing services at IBM.

## References

- [1] A. Akkawi, S. Schaller, O. Wellnitz, L. Wolf, A mobile gaming platform for the IMS, in: NETGAMES, 2004, pp. 77–84.
- [2] N.E. Baughman, B.N. Levine, Cheat-proof payout for centralized and distributed online games, in: Proceedings of IEEE INFOCOM, 2001, pp. 104–113.
- [3] A.R. Bhambe, S. Rao, S. Seshan, Mercury: a scalable publish–subscribe system for internet games, in: Proceedings of the 1st Workshop on Network and System Support for Games (NETGAMES), 2002.
- [4] Convedia, <<http://www.convedia.com>>, 2003.
- [5] S. Donovan, The SIP INFO method, RFC 2976, Internet Engineering Task Force, 2000.
- [6] C. Elliot, A ‘sticky’ conference control protocol, *Internet-working: Research and Experience*, 1994.
- [7] S. Fiedler, M. Wallner, M. Weber, A communication architecture for massive multiplayer games, in: Proceedings of the 1st Workshop on Network and System Support for Games (NETGAMES), 2002, pp. 14–22.
- [8] T.A. Funkhouser, RING: a client–server system for multi-user virtual environments, in: *Symposium on Interactive 3D Graphics*, 1995, pp. 85–92, 209.
- [9] GameSpy, <<http://www.gamespyarcade.com>>, 2003.
- [10] L. Gautier, C. Diot, Design and evolution of mimaze, a multi-player game on the internet, in: *IEEE Multimedia Systems Conference*, July 1998.
- [11] M. Handley, V. Jacobson, SDP: session description protocol, RFC 2327, Internet Engineering Task Force, June 2002.
- [12] J. Lennox, H. Schulzrinne, A protocol for reliable distributed conferencing, in: *Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2003.
- [13] M. Mauve, V. Hilt, C. Kuhmunch, W. Effelsberg, RTP/i-toward a common application level protocol for distrib-

uted interactive media, IEEE Transactions on Multimedia 3 (1) (2001) 152–161.

- [14] Online Multiplayer Games Network. <<http://www.omgn.com>>, 2003.
- [15] The Role Playing Games Network. <<http://www.roleplayinggames.net>>, 2003.
- [16] Sony PlayStation. <<http://www.playstation.com>>, 2003.
- [17] A. Roach, Session initiation protocol (SIP)-specific event notification. RFC 3265, Internet Engineering Task Force, June 2002.
- [18] J. Rosenberg, H. Schulzrinne, G. Camarillo, A.R. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: session initiation protocol, RFC 3261, Internet Engineering Task Force, June 2002.
- [19] D. Saha, S. Sahu, A. Shaikh, A service platform for on-line games, in: Proceedings of the 2nd Workshop on Network and System Support for Games (NETGAMES), 2003.
- [20] A. Singh, A. Acharya, Using session initiation protocol to build context-aware VoIP support for multiplayer networked games, in: NETGAMES, 2004.
- [21] A. Singh, P. Mahadevan, A. Acharya, Z.Y. Shae, Design and implementation of sip network and client services, in: ICCCN, 2004.
- [22] WebRPG, <<http://www.webrpg.com>>, 2002.
- [23] Microsoft Xbox, <<http://www.xbox.com>>, 2003.



**Aameek Singh** is a doctoral student in the College of Computing at Georgia Tech, where he is a member of the Distributed Data Intensive Systems Lab and the Center for Experimental Research in Computer Science (CERCS). His research interests are in the area of distributed storage systems, SIP/VoIP and WWW. He is a winner of the prestigious IBM Ph.D. Fellowship for the year 2005–2006. He is a

graduate of the Department of Computer Science and Engineering at Indian Institute of Technology, Bombay.



**Arup Acharya** is a Research Staff Member in the Network Server System Software group at IBM T.J. Watson Research Center in Hawthorne, NY. His current work includes high-performance server infrastructure for SIP, SIP applications such as VoIP, Instant Messaging & Presence and 802.11 based wireless mesh networking. He is the global lead for the Advanced Networking micropractice for On-Demand

Innovation Services within IBM Research, and holds a Visiting Professor position at WINLAB, Rutgers University. He has published in leading conferences and journals in the area of networking architecture and protocols, including mobile/wireless networks, holds five patents and has actively participated in conference program committees. He is currently the Vice-Chair for IEEE Conference on Mobile Ad-Hoc and Sensor Systems (MASS), 2005 and IEEE Conference of Distributed Computing Systems (ICDCS), 2006. He received his B.Tech. degree in Computer Science & Engineering from the Indian Institute of Technology, Kharagpur and a Ph.D. in Computer Science from Rutgers University in 1995. He was with NEC C&C Research Labs prior to joining IBM Research. Further information is available at <http://www.research.ibm.com/people/a/arup/>.