RC23148 (W0403-084) March 12, 2004
Computer Science

IBM Research Report

Design and Implementation of SIP Network and Client Services for Enabling Collaborative Applications

Aameek Singh
Georgia Institute of Technology
College of Computing
801 Atlantic Drive
Atlanta, Georgia 30332

Priya Mahadevan
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, Dept. 0114
La Jolla, CA 92093-0114

Arup Acharya, Zon-Yin Shae
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Design and Implementation of SIP Network and Client Services for Enabling Collaborative Applications

Aameek Singh[†♣] Priya Mahadevan[‡♣] Arup Acharya[¥*] Zon-Yin Shae[¥]

[†]College of Computing, Georgia Tech
aameek@cc.gatech.edu
§ Work done at IBM Research

[‡]Computer Science & Eng., UCSD
pmahadevan@cs.ucsd.edu

[¥]IBM T.J. Watson Research Center, Hawthorne, NY
{arup, zshae}@us.ibm.com
* Contact Author

Abstract

The Session Initiation Protocol (SIP) has been widely adopted for VoIP and Instant Messaging systems. However, the deployment has been primarily on a per-application basis. In this paper, we describe the design and implementation of a generic SIP client service and API that is available to all applications and provides unified mechanisms for functions like conference joins, pub-sub, call transfers etc. The service allows a user to plug in a softphone of his choice, and to use any one of multiple softphones, IP or PSTN phones on a per-session basis. We demonstrate the utility of this API by enhancing the user experience for existing applications as well as describing new converged applications. This client service is supported on the network side by prototyping a number of building blocks in addition to integrating a commercially available packet-audio mixer.

1. Introduction

Session Initiation Protocol (SIP) [1] is a popular choice for establishing media sessions and Instant Messaging. There are several IP softphones and hardphones available in the market today that are SIP capable which are being used for VoIP (Voice over IP). In addition to point-to-point calls, SIP is also being used for multi-party conference calls [2]. Typically, each SIP application such as an IM client or a softphone rolls out its own implementation of SIP. One of the popular JAVA APIs, the JAIN API [3] provides a low-level API to applications to invoke SIP call flows.

We view SIP as a new control pipe to the client desktop beyond just IM and VoIP and offer SIP as a generic service that is available to all applications on the client. Current deployment of SIP has been primarily on a per-application basis, lacking a unified mechanism across various applications. This prevents the system from exploiting all of SIP functionality in an integrated manner. In this paper, we describe the design and implementation of a SIP service which provides a generic SIP API to all applications in the form of a small set of SIP-specific primitives like conference join, event notification, call transfer. In addition, the service is designed such that it is easy to plug in a softphone of user's choice, thereby offering the opportunity to select one of multiple devices such as one of multiple softphones, IP or PSTN phones on a per-session basis. We demonstrate the richness of this API by enhancing the user experience for existing applications such as native SIP click-to-call in web browsers and enabling ad-hoc conferencing in a non-SIP aware messaging client, as well as describing new converged applications such as web-browsing with out-of-band control information passed via the SIP service, as well dynamic multi-conferencing support for multiplayer network games. This client service is supported on the network side by prototyping a number of building blocks such as a conferencing server with pub-sub support and the ability to create conferences on-the-fly, web sites that use applets to control client SIP service and a gaming server that maps changing gaming contexts to one of multiple conferences. A commercially available packet-audio mixer was integrated into the network for media support.

2. SIP

SIP is a control protocol that allows creation, modification and termination of sessions with one or more participants. SIP is used for voice and video calls either as point-to-point or multiparty sessions. It is independent of the media transport which for example, typically uses RTP [4]. SIP is also used for Instant Messaging and Presence [4]. SIP allows multiple end-points to establish media sessions with each other: it supports locating the end-points, establishing the session and then, after the media session has been completed, terminating

the session. A SIP infrastructure consists of user agents, registration servers, location servers and SIP proxies deployed across a network. SIP defines a set of messages such as such as INVITE, REFER etc. to setup sessions between *user agents*. These methods are routed through SIP *proxies* and that are deployed in the network. SIP uses the Session Description Protocol (SDP) [5] in the message body to provide information about the session like media type, transport protocol, IP addresses and port numbers of endpoints. Details can be found in [1].

3. Client-side SIP Service

Our SIP service acts as a client side system service (like telnet, ftp) running on a particular port. It offers an API to applications at a higher functional level than call control, as offered by existing SIP APIs such as JAIN SIP [3]. More importantly, the API is targeted at the operating systems level so that the API is available to all applications independent of the application execution environment such as a JVM. There are two principal modes in which applications can invoke this API: (a) by directly sending messages to the specified port of the service, and (b) SIP was registered as a protocol in the operating systems registry (Win2K in our case) along with a protocol handler. Consequently, any existing or new application that looks up the registry to determine protocol handlers would automatically be able to handle SIP URIs by invoking the associated protocol handler. SIP was thus made into a first-class protocol recognized and handled by the operating system in a similar fashion such as a mailto: or http: URIs. The specified protocol handler would then invoke a pre-determined function from the SIP service API. An important requirement of our SIP service is that the supporting API be extensible. Thus, the current set of functions which will be described later, is not by any means complete, but rather to illustrate an initial set that we found useful across multiple application scenarios.

A second motivation for a client-side SIP service is to offer all *control* functions that SIP has to offer, within a single service, instead of separate application bundling in specific subsets of full SIP functionality. For example, an IM client may incorporate publish-subscribe mechanisms of SIP, but may not allow an audio call to be setup, while a softphone may incorporate call control functionality but not necessarily support Presence and IM functionality. In addition, when two such applications are executed concurrently, there is often a problem with sharing common port numbers (such as port 5060). More importantly though, this leads to narrowly focused SIP applications. Our motivation for an application-independent SIP service is to enable new applications that combine multiple *control* features of SIP in interesting ways.

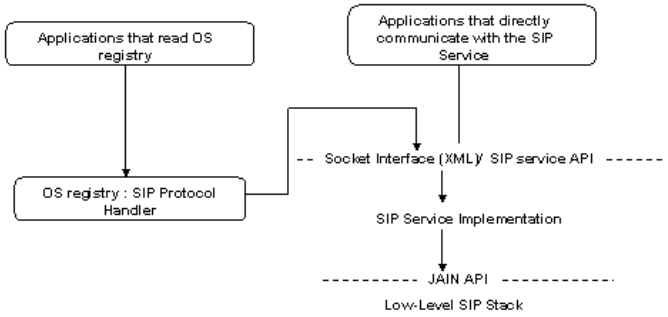


Figure 1: SIP Service Architecture

Lastly, a requirement of our proposed SIP service is the ability to easily plug in a softphone of user's choice and also offer a choice of end-device for user interaction. A user may choose a device with features that is best suited for the type of session, e.g. a cell-phone may have a built-in camera, or the desktop phone may offer good speakerphone support. This requirement has several advantages: firstly, it frees the SIP service from providing media I/O capability, which is best offered by specialized devices, while retaining the control of such devices from the SIP service. In terms of realizing this requirement, it implies that the SIP service be designed to allow integrating end-devices in different ways, e.g. some softphones may simply be launched with a SIP URI as a parameter while some softphones only allow indirect interaction using a web server, in which case appropriate HTTP POST messages are required. In addition, when an external device is used, the user may still like to retain control of the session (rather than offloading both media and control to the device). This is especially true, when a user wants to utilize special SIP functions like SUBSCRIBE/NOTIFY, which a non-SIP device cannot provide.

The SIP service also allows users to switch devices in the midst of a session. This requirement places a burden on the design of SIP Service in that it should allow easy integration of other SIP devices and also PSTN devices. This is achieved through

[3] We used an IBM-internal version of the Lotus SameTime client. The latest commercial release of Lotus SameTime and Web Conferencing (version 3.1) supports SIP for inter-gateway communication but not for client-server interaction [10].

device-specific *wrappers*, which are responsible for translating such higher level functional demands to lower level device commands, either through SIP or non-SIP methods such as HTTP post.

3.1 API

Applications communicate with the SIP service using XML messages, which encode SIP service API calls. The use of XML allows standardized mechanisms of interaction with the SIP service and also provides extensibility to the API. New functions can be easily added by using appropriate XML messaging tags. Below, we define an initial set of API calls:

3.1.1 ExternalJoin

This command is used to call a particular party when no end-user device is selected. The format for such a message is:

<ExternalJoin id=SIP URI />

This indicates that a call is to be made to a SIP URI such as sip:1619@research.ibm.com. The SIP service then pops up a dialog box asking the user to select from one of a set of end-user devices such as softphone, IP phones etc. Once a device has been selected, the SIP service uses the appropriate wrapper's primitive operations to make the call.

3.1.2 Join

This command is used to call a particular party by using a particular device. The format for such a message is as follows:

```
<Join id=SIP URI1>
   <Use dev= dev-id/>
</Join>
```

This indicates that a call is to be made to destination URI1 using a end-user device pointed to by *dev-id*. The *dev-id* is either the local softphone identifier, which indicates the SIP service to launch the appropriate softphone, or the SIP URI for an external hardphone device. Softphones are especially distinguished in this manner since we can easily launch them using their specific wrappers. However, it is also possible to use a similar hardphone mechanism using the SIP URL for the softphone as the chosen device. In cases when a SIP URI is used, the SIP service needs to establish a control path with the appropriate devices and set up a connection. This can be achieved in two modes, referring to the SIP service involvement in the entire process.

***Transfer* Mode**: In this case, the SIP service establishes a session between the end-device identified by URI2 and the called party URI1. The call is completely transferred to URI2. In this scenario, the SIP service has no further control on the call and the media transfer takes place between endpoints identified by the two URIs.

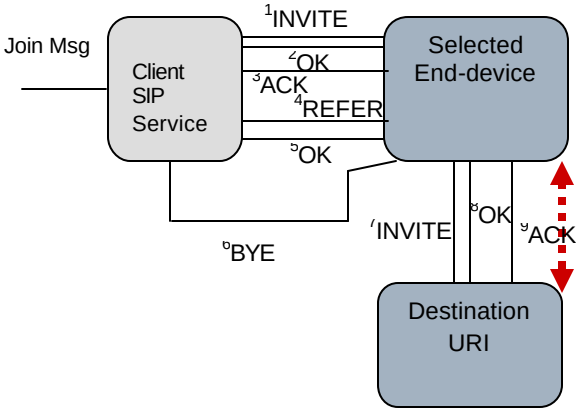


Figure 2: SIP Client Transfer Mode

Loop Mode: The SIP service acts as a Back2Back User Agent [7] between the two endpoints. The media path is still end-to-end between the two URIs; however, the SIP service stays in the control path between the two end-points. *This is useful for the SIP service to receive event notifications.*

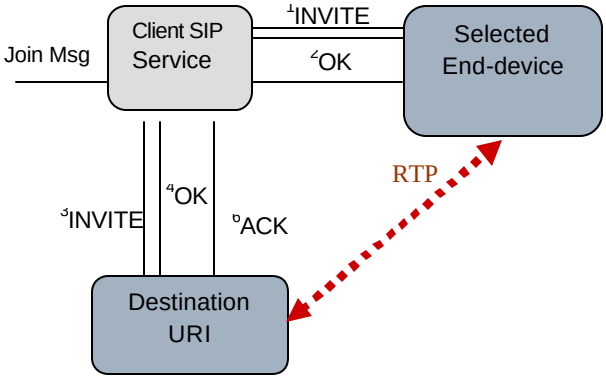


Figure 3: SIP Client Loop Mode

For example, in a conference call, a user may subscribe to join/leave events (using SIP SUBSCRIBE) and be notified of other participants joining/leaving the conference. Staying in the loop allows the SIP client to display any such notifications. This would not be possible using the transfer mode since the device represented by URI2 may either be incapable of handling the events or exposing them to the user in an appropriate way and then capturing user response to those events. The loop mode is also useful in the context of the next API call described below.

3.2.2 SameDeviceJoin

SameDeviceJoin allows a user to **seamlessly** switch the called endpoint (e.g. conference) without

changing the end-device currently in use. The format for a SameDeviceJoin message is:

<SameDeviceJoin id= SIP URI/>

This functionality cannot be realized by dropping the entire call and setting up a new call from the same end-device to the new target, since this would mean hanging up the external device and picking it up again, i.e. the switch would be perceptible. We need to provide a seamless switch. We use the loop mode SIP service, i.e. the SIP service is on the control path between URI1 and URI2 created by an earlier *Join* command in loop mode. The steps involved in realizing this function is to drop the leg to the current called party (URI1), setup a new call to the endpoint referred to by the URI in the SameDeviceJoin message, and then exchange the IP addresses and port numbers of the two media endpoints corresponding to URI and URI2.

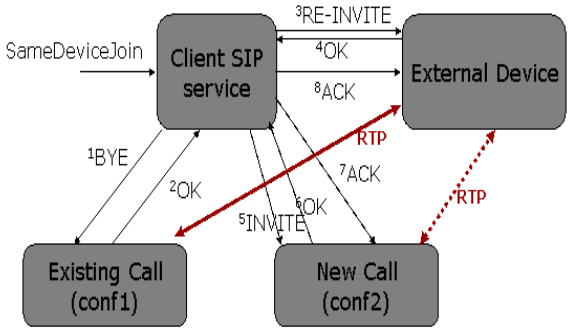


Figure 4: SIP Same Device Join

3.2.3 Multi-Invite

This API calls is specific to conferencing, and instructs the SIP service to invite specified additional participants to the current conference. In case the invoking user is not in a conference, a new ad-hoc conference is created and desired participants are invited to that conference. The format of the message is as follows:

<Invite>
<Add id=SIP URI1 />
<Add id=SIP URI2 />
.....
</Invite>

The mechanisms of creating and setting up ad-hoc conferences are detailed in Section-4.

4 Network Infrastructure

In order to support the SIP client service described above, we designed and implemented a number of building blocks within the network infrastructure.

The first of these is a **conference** control server, which when coupled with a commercially available SIP-controllable media mixer, provides a network service for setting up conferences and mixing audio streams. The unique properties of our conference control server are the ability to create ad-hoc on-the-fly conferences, and also to support event notification services for events related to conferencing. The conferencing service enables the SIP service client to create and control conferences on the fly, and offer that as a primitive to client applications.

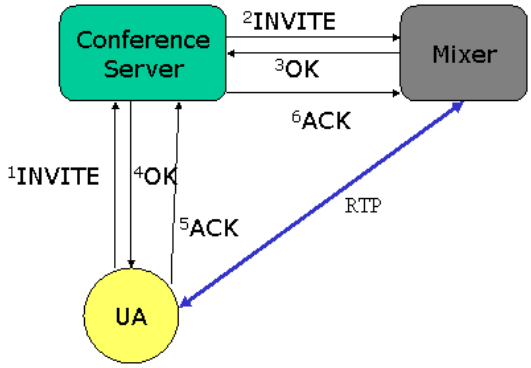
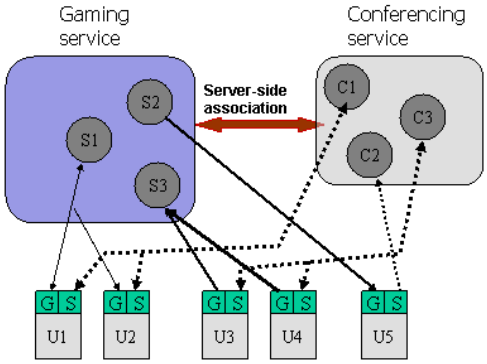


Figure 5: SIP Workflow for joining a conference

To establish an ad-hoc conference, a user generates a unique ID (e.g. a username appended by a random number), creates a conference URL of the form "*sip:<unique-ID>@<conf-server address>*" and sends an INVITE. The SIP proxy in our network is configured to forward any unregistered SIP URLs to the machine in the domain field of the URL (the conference server in our case). On receiving the INVITE, the conference server (CS) creates a new conference (if no such conference id exists; else the user is added to an existing conference as a participant). The setup is shown in Figure-5.

A second application building block that we introduced is modeled as a primitive gaming server which uses the conferencing server to enhance the gaming experience. *The purpose of this building block is to show that SIP allows multiple services in the network to be composed in interesting ways*, rather than to demonstrate gaming per se. The gaming server should be viewed as representing an *application server with multiple concurrent states*, such that application clients are each associated with one of the server states. This basic abstraction is augmented by associating clients sharing a common state at the application server with a common conference, demonstrating that the conferencing service is useful not just as a standalone service but perhaps more so, when

combined with another network service. This service composition can be achieved either by coupling the game and conference servers, or by coupling the game client with the SIP service API at the client-side. Both approaches are feasible.



S1, S2, S3 : Concurrent states of the game server
C1, C2, C3 : Corresponding Conferences
U1, U2,, : Clients
G : Gaming client
S : SIP Service

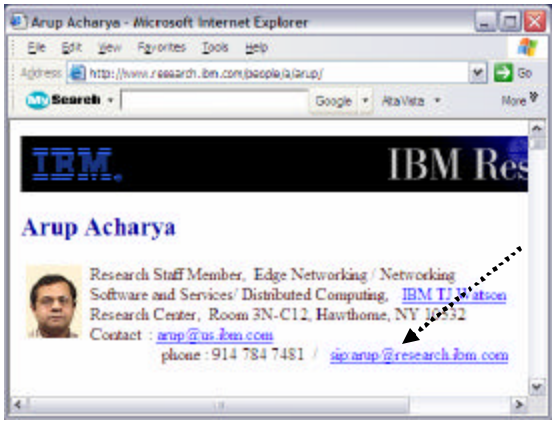
In a corporate enterprise setting, the "game service" is offered as an add-on service to conferencing that allows employees to participate in *multiple simultaneous conferences*, presented visually to the user as a set of boxes : dragging the mouse to a specified box seamlessly switches the employee's current active conference without any perceptible break in audio (i.e. without requiring the employee to hang-up and dial in to the new conference).
The feature of the gaming service that we wish to highlight is the ability to seamlessly and automatically switch the associated conference when a game client changes its gaming context (such as a 'dungeon'). This is where the functionality offered by the SIP service *SameDeviceJoin* comes in handy. The game client can invoke *SameDeviceJoin* when the user moves to a different game context, passing the SIP URI of the conference to the SIP service.

The final building block we introduce is a web-site that is cognizant of the SIP service API at the client. It can use embedded applets in its pages to instruct users to join particular conferences associated with those web pages. For example, a discussion forum web site, using this mechanism will instruct the viewers of a particular forum to be in a single audio conference and hence exchange their views via voice. More details in Section-5.

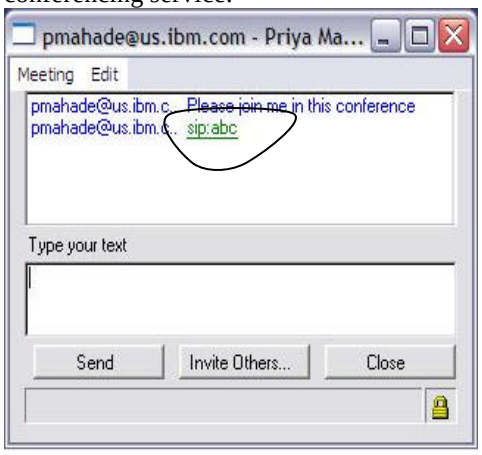
5 Prototypes of SIP-enabled Applications

In this section, we describe some of our prototype applications that combine the SIP service API and building blocks in the network infrastructure.

SIP URI's in web-browser: Since SIP is n protocol recognized by the Windows registry, browser programs that refer to the registry invoke the SIP protocol handler when an user clicks on a SIP URI embedded in a web-page. The protocol handler initiates an *ExternalJoin* SIP Service API call, which asks user input for device selection and then sets up a session to the URI. Note that the browser code was unmodified.



Enabling a non-SIP aware IM client: We selected an IM client and server system[3] that uses a proprietary non-SIP protocol, modified the client code to recognize SIP URI's within message bodies, highlighted the SIP URIs as 'clickable' links, and on user click, invoked the *Join* SIP Service API to create an ad-hoc conference via the conferencing service.



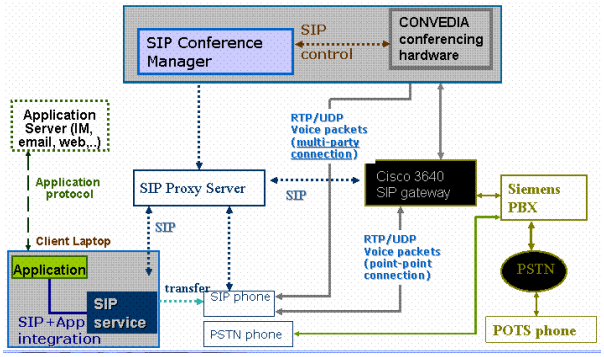
We followed certain naming conventions to identify a URI as a conference URI (e.g. the host name in the URI is conf.ibm.com), and hence, the user would need to supply just the conference name (e.g. *abc* in the example). The key points to note: (a) the functionality of the IM system was enhanced without changing the application's native client-server protocol (b) the messaging capability of the

application is used to inform other participants of a conference (SIP URI) and (c) this highlights how the SIP client and network services are useful for a class of applications whose client code may be amenable to modification but not the server code.

Multi-conferencing / Gaming with seamless conferencing: The "game" consists of four quadrants and a user can move in any of the four directions. When a user crosses a quadrant boundary, he is seamlessly conferenced in with users in the new quadrant. In the screenshot shown, *arup* and *edie* are able to hear each other, while *aameek* is not able to hear either. If *arup* were to move into the top-left quadrant, then *arup* would be added to the conference associated with that quadrant, and *arup* and *aameek* will hear each other (if *aameek* continues to remain in that quadrant).

Community Web Browsing: This application comprises of a group of web-pages, such that viewers of a web-page are informed of all concurrent viewers of that web-page using SIP's event notification mechanism as well as being conferenced together. In addition, whenever a user moves to a different page, her conference automatically switches to the one associated with the new page. To facilitate this switching, we again use the client SIP API. All "enabled" web pages contain a signed applet which simply writes a *SameDeviceJoin* message to the client SIP service socket. As a result whenever a client loads the page, the applet writes the SameDeviceJoin command and the user is brought into the conference of that particular web page. Note that in case there is no active device, the SameDeviceJoin acts as an ExternalJoin, requiring user input for device selection. In addition, to provide users with information about other viewers at that time, we use SIP based SUBSCRIBE/NOTIFY features notifying users of all Join and Leave events. This application is an attempt at community based communication. For example, this would allow a community of movie fans reviewing a particular movie to participate in a voice discussion amongst online fans in real-time, as opposed to text chat.

The SIP client service and network services were integrated with the existing SIP pilot network within IBM Watson. This is shown in the following network diagram, which shows the SIP client service and a conferencing service (gaming and SIP enabled web-site is not included in the diagram)



6 Conclusions

In this paper, we designed an implemented an extensible client-side SIP service API and network building blocks that proved to be useful in enhancing existing collaborative applications as well come up with new applications such as seamless conferencing for gaming and community-based web-browsing.

References

1. J. Rosenberg et al. SIP : Session Initiation Protocol. RFC 3261. IETF, June 2002.
2. Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, IETF Apr98.
3. JSR 32 : JAIN™ SIP API Specification. Java Community Process.
4. H. Schulzrinne et al. RTP : A Transport Protocol for Real-Time Applications. RFC 1889.IETF, Jan 96..
5. B. Campbell, editor. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428.IETF, Dec 2002.
6. Igor Miladinovic and Johannes Stadler, "Multiparty Conference Signaling using SIP", International Network Conference, 2002.
7. B. Campbell et al. The Message Session Relay Protocol. SIMPLE Working Group Internet-Draft, Jan 2004.

8. J. Lennox and H. Schulzrinne. A Protocol for Reliable Decentralized Conferencing. NOSSDAV 2003.
9. J. Rosenberg et. al. Best Current Practices for Third Party Call Control in the Session Initiation Protocol. Internet-Draft , Dec 2003.
10. http://www.lotus.com/products/lotussametime.nsf/wdocs/about

,