

Design and Implementation of SIP Network and Client Services

Aameek Singh*
Georgia Institute of Technology
aameek@cc.gatech.edu

Priya Mahadevan*
University of California
pmahadevan@cs.ucsd.edu

Arup Acharya Zon-Yin Shae
IBM T.J.Watson Research Center
{arup, zshae}@us.ibm.com

Abstract

Session Initiation Protocol (SIP) is being widely adopted for VoIP, IM and other collaborative applications due to its simple yet rich functional design. However, one of the main drawbacks has been its per-application deployment (each application using its own SIP stack), leading to narrowly focussed development of SIP based services. In this paper, we propose a client-side SIP service and supporting network infrastructure blocks that provide unified mechanisms to execute generic SIP functions. The composition of these building blocks allows for creating richer applications, e.g. a conferencing server coupled with a gaming server provides dynamic conferencing between current occupants of a game room. The main feature of our framework is its availability to *all* applications including the ones not inherently based on SIP. Also, the SIP service API is designed to be *extensible* and in addition to providing novel higher level functional primitives like adhoc conferencing and seamless transition of sessions, it also exports a low level interface for specialized applications. Another feature of the service is that it allows a user to plug-in an end device of his/her choice on a *per-session* basis. We demonstrate the richness of the API by describing prototypes for enhancing various applications as well as new converged applications.

1 Introduction

Session Initiation Protocol (SIP) [8] is a popular choice for establishing media sessions and Instant Messaging [3, 4]. There are several IP softphones and hardphones available in the market today that are SIP capable and 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 [10, 7]. Typically, each SIP application such as an IM client or a softphone rolls out its own implementation of SIP, based on an API like JAIN [1], which provides a low-level API for invoking 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 system service that is available to all applications. We describe the design and implementation of a SIP service (Section-2) which provides a generic API in the form of a set of SIP-specific primitives like point-to-point calls, conferencing, event notification etc. In addition, the service is designed such that it is easy to plug-in an end device of user's choice, thereby offering the opportunity to select on a *per-session* basis, one of multiple softphones, an IP or a regular PSTN phone. We demonstrate the richness of this API by enhancing 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 (Section-5). This client service is supported on the network side by prototyping a number of building blocks such as a conferencing server with event notification 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 (Section-3). A commercially available packet-audio mixer was integrated into the network for media support.

2 Client-Side SIP Service

Our SIP service acts as a client side system service running on a particular port. It offers an API to applications at a higher functional level than the one offered by existing SIP APIs such as JAIN SIP [1]. However, in order to meet the demands of specialized applications that require a lower level control, we also provide a *tunnel* through our API providing direct access to SIP call flows. Importantly our API is targeted at the operating systems level to ensure its availability to all applications indepen-

*Work done while visiting IBM Research

dent of the application execution environment such as a JVM. There are two modes (see Figure-1) in which applications can invoke this API: (a) by directly sending messages to the specified port of the service, and (b) using SIP as a protocol in the operating systems (by registering a protocol handler for Windows based systems, for example). The second mode can also be provided in Linux based operating systems either by encoding appropriate plugins or inserting modules within the OS itself. It allows any existing or new application, that looks into the OS to determine registered protocols, to automatically be able to handle SIP URLs by invoking the associated protocol handler (similar to *mailto* or *http*).

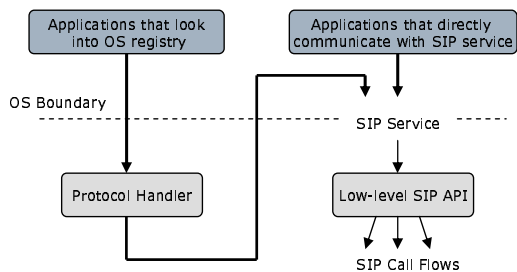


Figure 1: Client SIP Service

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.

Lastly, a requirement of our proposed SIP service is the ability to offer a choice of end-device (SIP User Agent UA) 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. 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 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 Network Infrastructure

Before we describe the detailed API, we first explain the building blocks used to facilitate SIP network services.

3.1 Conference Server

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. For every participant, the mixer combines the voice signals of every other participant into a single signal. There are various off-the-shelf SIP-enabled mixers available such as [5]. The conference server uses SIP signaling with the user agent (UA) and the mixer to establish media paths between the two. The unique properties of our conference control server are the ability to create ad-hoc on-the-fly conferences¹, and also to support event notification services for events related to conferencing.

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

¹Such conferences do not reserve resources in advance and are initiated on-the-fly, thus not registering conference SIP URL at SIP proxies. Routing messages for the conference is the main challenge.

the form “sip:<ID>@<conf-server address>” and sends an INVITE. To route such messages which have not been registered at the SIP proxy (as done for normal SIP routing), we set up the SIP proxy 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 (Figure-2).

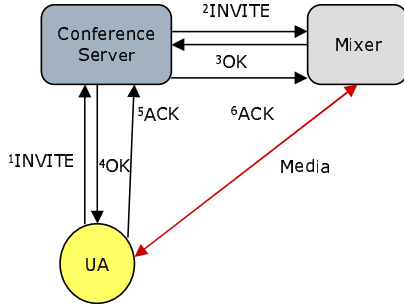


Figure 2: SIP Workflow for a Conference Join

Event notification is achieved using the SUBSCRIBE/NOTIFY messaging feature of SIP by implementing a new event package for conferencing contexts.

3.2 Gaming Server

A second application building block that we introduced is modeled as a primitive gaming server which uses the conferencing server to enhance the multiplayer 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.

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 employees 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). To perform such seamless and dynamic switches, we create appropriate API function primitives (discussed in Section-4). The overall architecture is shown in Figure-3.

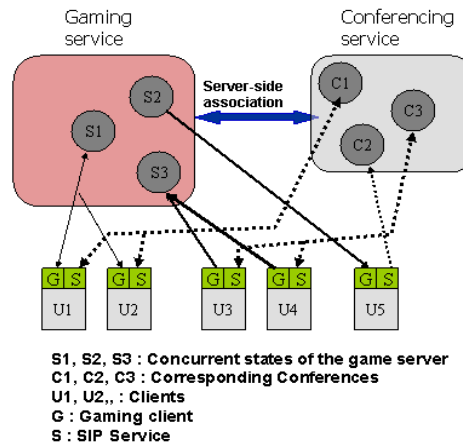


Figure 3: Gaming and Conferencing Composition

3.3 SIP-aware Web Server

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. In addition, to keep users aware of other participants, SIP event notification is implemented using the SUBSCRIBE/NOTIFY features. For example, when a new user joins a particular forum, the existing participants are notified of the new participant and displayed accordingly by the SIP service.

The idea is to create greater collaborative environments by utilizing the SIP service at the clients. Embedding a signed applet in a web page allows a web server to co-ordinate client audio sessions

and create meaningful groupings. The applet writes SIP API commands (as discussed in Section-4) to the SIP service ports and users can join conferences based on those web pages. Note that since we use the client-side SIP service, the user still gets to enjoy the call control features provided by the SIP service. Moving to a different web page will seamlessly transition the user into a new conference based on that web page. It is important to notice the distinction of control paths between the web server and the gaming application server examples. While in the gaming server the call control was done via the server itself (based on client *gaming* actions), in the web server the call control is through the client-side SIP service.

4 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:

– **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 URL />
```

This indicates that a call is to be made to a SIP URL such as sip:arup@research.ibm.com. The action associated with this function is that the SIP service 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 for that device to make the call. The wrappers were small Java implementations of primitive SIP functions for the device.

– **Join:** This command is used to call a particular party by using a particular device. The format is:

```
<Join id=SIP URL1>
<Use dev=dev-id/>
</Join>
```

This indicates that a call is to be made to destination URL1 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 URL for an external hardphone device (appropriately SIP-gatewayed if required). 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 URL 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 process.

1. *Transfer Mode:* In this case, the SIP service establishes a session between the end-device (identified by URL2) and the called party URL1. The call is completely transferred to URL2 using a SIP REFER. 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 URLs (Figure-4).

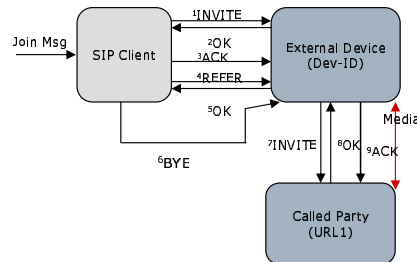


Figure 4: Transfer Mode

2. *Loop Mode:* The SIP service acts as a Back2Back User Agent [9] between the two endpoints. The media path is still end-to-end between the two URLs; 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. 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 URL2 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.

– **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 URL/>
```

This functionality cannot be realized by dropping

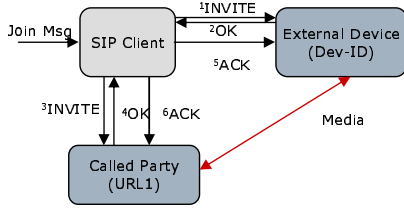


Figure 5: Loop Mode

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, i.e. never terminate the session with the external device. We use the loop mode SIP service, i.e. the SIP service is on the control path between device Dev-ID and say, conf1 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 (conf1), setup a new call to the endpoint referred to by the URL in the SameDeviceJoin message (maybe a new conference, conf2), and then exchange the IP addresses and port numbers of the two media endpoints (Figure-6).

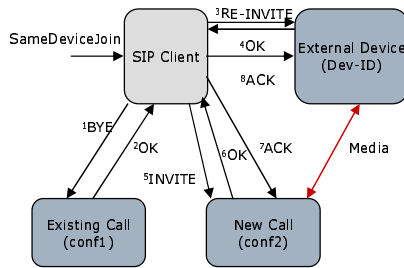


Figure 6: SIP Workflow for SameDeviceJoin

– **Multi-Invite:** This API call 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 URL1 />
<Add id=SIP URL2 />
.....
</Invite>
```

– **Tunnel:** This API call provides a tunneled access to low level SIP workflow commands (specific SIP messages). The motivation for providing such an access is to allow specialized applications to take

greater control of the SIP call flows. This mechanism is provided through an add-on plugin implementing the interface providing access to low level SIP APIs. Using the tunneling feature would require developers to write code (as opposed to exchanging XML messages), however, considering the target of specialized applications, this should not be a major hindrance.

5 Prototype 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 URLs in web-browser: Since now, SIP is a 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 URL 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 URL. Note that the browser code was unmodified.



Figure 7: SIP URLs in Web-Browser

Enabling a non-SIP aware IM client: We selected an IM client and server system [2] that uses a proprietary non-SIP protocol, modified the client code to recognize SIP URLs within message bodies, highlighted the SIP URLs as clickable links (Figure-8), 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 URL as a conference URL (e.g. the host name in the URL 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 applications native client-server protocol (b) the messaging capability of the application is used to inform other

participants of a conference (SIP URL) 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.

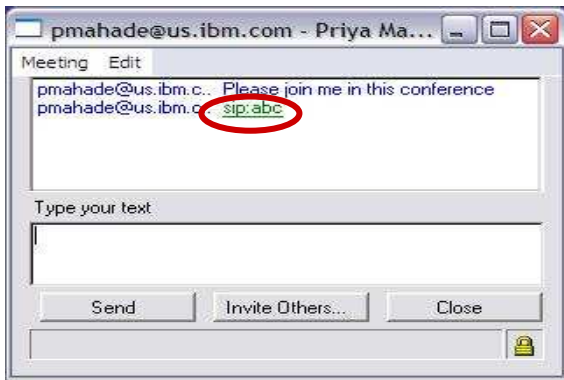


Figure 8: Enabling IM Client

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).

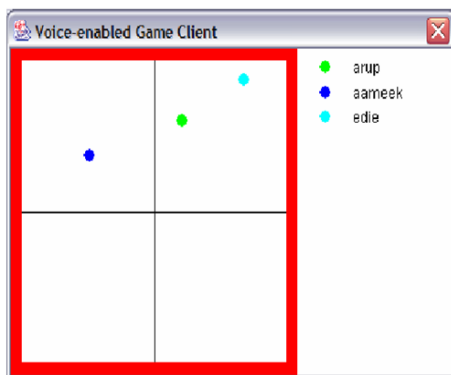


Figure 9: Multi-Conferencing/Gaming

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 SIPs event notification mechanism as well as being conferenced to-

gether. 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.

6 Related Work

There have been other efforts in designing APIs and languages for SIP services. Call Processing Language (CPL) [6] is an XML-based scripting language for describing and controlling call services. Users create CPL scripts and these run on signaling servers. Since CPL is essentially a server side utility, it focuses on network services and not end-user service. In contrast our SIP API is a client side utility and aims to bring SIP to the client desktop.

JAIN SIP [1] is a low level API and provides a standard portable interface to share information between SIP Clients and SIP Servers. Since it’s a low level API, it does not provide high-level abstractions such as SameDeviceJoin. Coexistence of a range of services on a single end system all using the same SIP stack and API is also not defined by the JAIN API.

Language for End System Services (LESS) [11] is XML based language for end system services. It mainly differs from our utility in the fact that it is a language not an API. As demonstrated by us, a wide range of applications can effectively use our APIs to use SIP effectively and efficiently. In contrast, applications will have to be programmed using LESS, thus making it much harder for existing applications to exploit the richness of SIP.

7 Conclusions

In this paper, we designed a SIP based system service which provides an extensible client-side API.

Such a mechanism provides a unified mechanism of executing generic SIP functions and makes them accessible to all applications. The infrastructure is supported by a number of network building blocks like conference server and the “gaming” server. We demonstrated the utility of such a service by enhancing existing collaborative applications as well design of new applications such as seamless conferencing for gaming and community-based web-browsing. In the future, we intend to broaden the SIP service API and provide greater programming support for customizable actions.

References

- [1] JSR 32: JAIN SIP API Specification. <http://jcp.org/aboutJava/communityprocess/final/jsr032/index.html>.
- [2] Lotus Sametime. <http://www.lotus.com/products/lotussametime.nsf/wdocs/about>.
- [3] B. Campbell and J. Rosenberg and H. Schulzrinne and C. Huitema and D. Gurle. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428, Internet Engineering Task Force, Dec 2002.
- [4] B. campbell et al. The Message Session realy Protocol. SIMPLE Working Group Internet Draft, Jan 2004.
- [5] Convedia. . <http://www.convedia.com>, 2003.
- [6] H. Schulzrinne et. al. CPL: A Language for User Control of Internet Telephony Services. Internet Draft, Aug 2003.
- [7] I. Milandinovic and J. Stadler. Multiparty Conference Signaling using SIP. In *International Network Conference*, 2002.
- [8] J. Rosenberg and H. Schulzrinne and G. Camarillo and A. R. Johnston and J. Peterson and R. Sparks and M. Handley and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, June 2002.
- [9] J. Rosenberg et. al. Best Current Practises for Third Party Call Control in SIP. Internet Draft, Dec 2003.
- [10] P.Koskelainen and H. Schulzrinne and X. Wu. A SIP-based Conference Control Framework. In *NOSSDAV*, 2002.
- [11] X. Wu and H. Schulzrinne. Programmable End System Services Using SIP. In *IEEE International Conference on Communications*, 2003.