Implementing Multimedia Sessions using SIP Architecture

Prepared By-

Prenayan Kaul
Kushagra Pant
Rana Gaurav Goyal

VIth semester,
B.E. Computer Science,
Netaji Subhash Institute of Technology (NSIT)

Abstract

The need to enhance IP telephony services, requires convergence of both new and existing applications like voice, multimedia, email and a host of other services. In order to coordinate various services an integrable and common packet based signaling framework is required. SIP-Session Initiation Protocol- best fits this role of creating and managing multimedia sessions.

Mobility and portability are important issues in the present era of emerging communication services. SIP provides us with a mobile networking environment, which improves upon the existing MobileIP framework. The inherent flexibility of SIP allows for easy service creation and enhanced functionality.

INTRODUCTION

The two main hurdles encountered in the development of converged IP telephony services are:
Mobility – the ability of a mobile host to move from one network host to another without encountering loss of session and any delays.
Portability - provision of a horizontal platform of distributed services, which give flexibility in integrating different platforms.

Unlike earlier VoIP protocols, with the concept of vertical integration and a large number of inbuilt protocols, SIP provides a basic framework, which fits together with a number of different protocols, visualizing a web of decentralized services. SIP is an application level signaling protocol defined for initiating, modifying and tearing down of interactive, multimedia sessions between users. Its simplicity and portability gives it an edge over current VoIP Protocols like H.323, and its ability to support terminal, personal and session mobility for applications in IP telephony make it the best choice for any architecture supporting multimedia communications with mobile hosts. Combined with its flexibility SIP provides an ideal platform for third party service creation.

Overview

Section I deals primarily with the basics of Session Initiation Protocol. We explore the network architecture and explain the role of the various key components involved (i.e. User Agents and Servers). Addressing and protocol methods are also stated along with a call-setup example.

Section II addresses the issue of providing mobility and the role of SIP in creating mobile networks. We explore the limitations of existing MobileIP and explain how SIP can be effectively used to provide enhanced mobility.

Section III explores the setting up of a Multimedia Conferencing Service, in a SIP architecture, using Java. We propose the integration of Java Enhanced SIP (JES) with the JAIN SIP API, which provides us with the functionality to implement SIP methods without interacting with the SIP Protocol Stack.

Section I What is SIP ?

1.1 SIP Basics

SIP allows the two or more participants to establish a session consisting of multiple media streams. The media streams can be audio, video or any other Internet based communication. SIP defines a number of logical entities, namely user agents, proxy or redirect servers and registrar servers. SIP only establishes IP addresses and port numbers at which end systems can send and receive data and uses existing media transfer protocols like RTP and RTCP for QoS. The strength of SIP lies in the fact that it is an ASCII based peer-to-peer protocol closely associated with Internet Protocols like HTTP, SMTP, etc. and uses common MIME extensions.

Like other VoIP protocols, SIP is designed to address the functions of signaling and session management within a packet telephony network. *Signaling* allows call information to be carried across network boundaries. *Session management* provides the ability to control the attributes of an end-to-end call.

SIP provides the capabilities to:

- Determine the location of the target end point—SIP supports address resolution, name mapping, and call redirection.
- Determine the media capabilities of the target end point—Via Session Description Protocol (SDP); SIP determines the "lowest level" of common services between the end points. Conferences are established using only the media capabilities that can be supported by all end points.
- Determine the availability of the target end point—If a call cannot be completed because the target end point is unavailable, SIP determines whether the called party is already on the phone or did not answer in the allotted number of rings. It then returns a message indicating why the target end point was unavailable.
- Establish a session between the originating and target end point—If the call can be completed, SIP establishes a session between the end points. SIP also supports mid-call changes, such as the addition of another end point to the conference or the changing of a media characteristic or codec.
- Handle the transfer and termination of calls—SIP supports the transfer of calls from one end point to another. During a call transfer, SIP simply establishes a session between the transferee and a new end point (specified by the transferring party) and terminates the session between the transferee and the transferring party. At the end of a call, SIP terminates the sessions between all parties.

1.2 SIP Architecture

SIP has a distributed architecture, consisting of various function specific servers. The entities that initiate, receive and terminate requests are called User Agents.

1.2.1 SIP Components

User Agents

The two types of user agents are-
UAC (User Agent Client)- An entity that initiates a call.
UAS (User Agent Server)- An entity that receives a call.

The servers, which form the backbone of the SIP architecture-:

Proxy Server

It is an intermediary program that acts as both a server and a client to make requests on behalf of other clients. If it can, any request it receives is processed internally or it is passed on to other servers, possibly after some translation.

Location Server

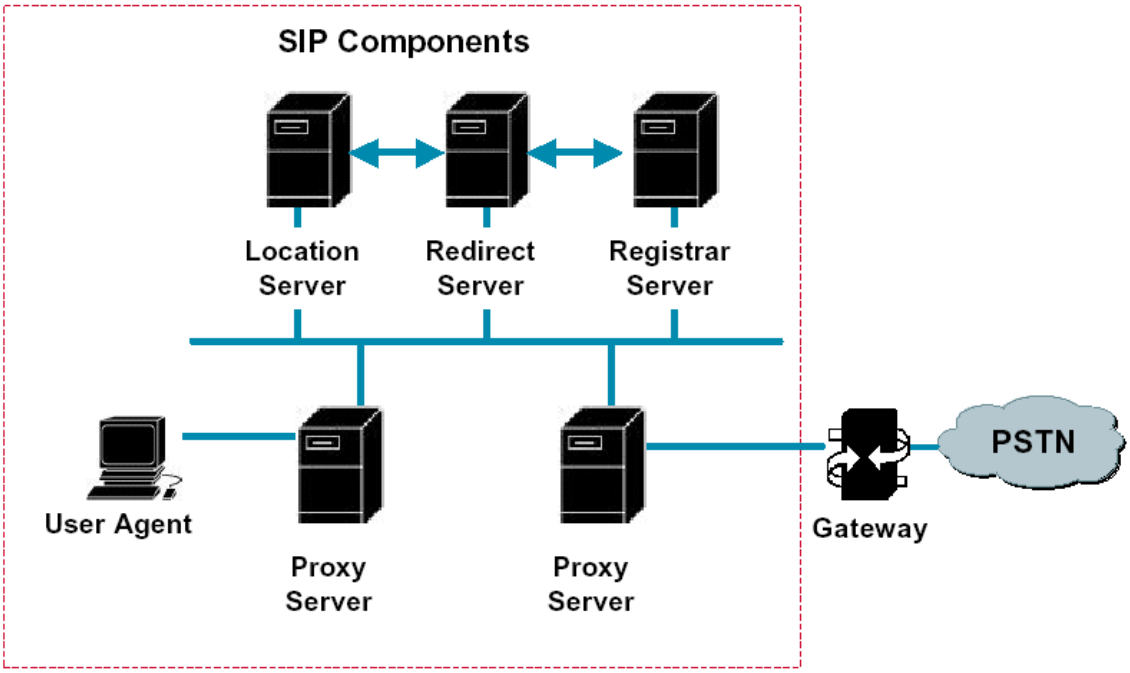
A location server is used by a SIP redirect or proxy server to obtain information about a called party's possible location(s).

Redirect Server

A server that accepts a SIP request, maps the address into zero or more new addresses and returns these addresses to the client. The redirect server redirects any request received by a proxy server to the next location on its path to the end client. Unlike a proxy server, the redirect server does not initiate its own SIP request. Unlike a user agent server, the redirect server does not accept or terminate calls.

Registrar Server

The SIP User Agents initially register their current network location with their local registrar by sending a REGISTER request A registrar server is typically co-located with a proxy or redirect server

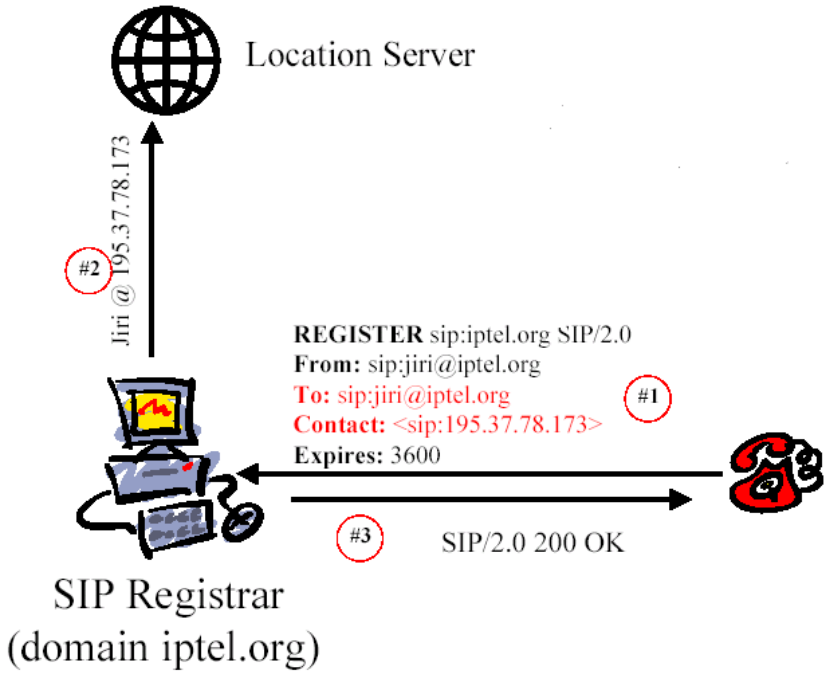


1.3 SIP Addresses

SIP gives you a globally reachable address. Callers bind to this address using SIP REGISTER method. Callers use this address to establish real-time communication with callers. The address data format used in SIP is as given below:

username@host.com

1.4 SIP Call Setup



Registration diagram

 # User registration

When a user wants to start or enter a SIP session he has to register his User Agent on the Registrar server. In order to do this he sends a REGISTER request to the Registrar server, which binds the user's address with his current location and passes this information onto the Location server.

 # Inviting a User

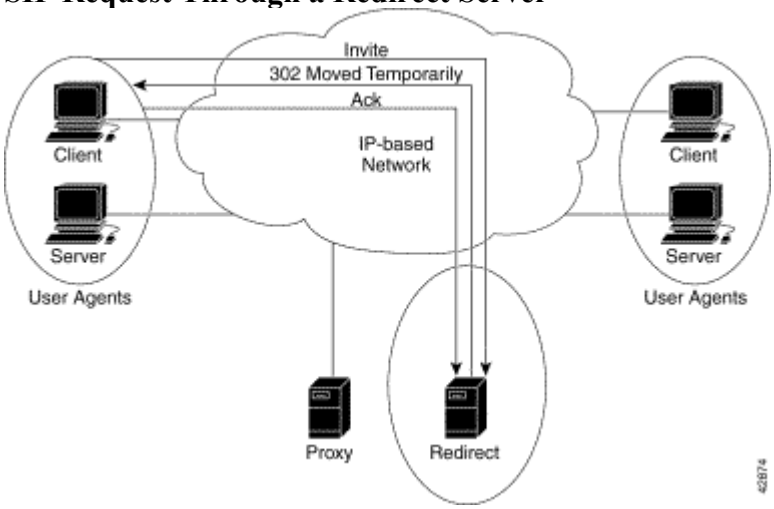
When a user initiates a call, a SIP request is sent to a SIP server (either a proxy or a redirect server). The request includes the address of the caller (in the From header field) and the address of the intended callee (in the To header field).

Over time, a SIP end user might move between end systems. The location of the end user can be dynamically registered with the SIP server. The location server can use one or more protocols (including finger, whois, and LDAP) to locate the end user. Because the end user can be logged in at more than one station, it might return more than one address for the end user. If the request is coming through a SIP proxy server, the proxy server will try each of the returned addresses until it locates the end user. If the request is coming through a SIP redirect server, the redirect server forwards all the addresses to the caller in the Contact header field of the invitation response.

 # Using a Redirect Server

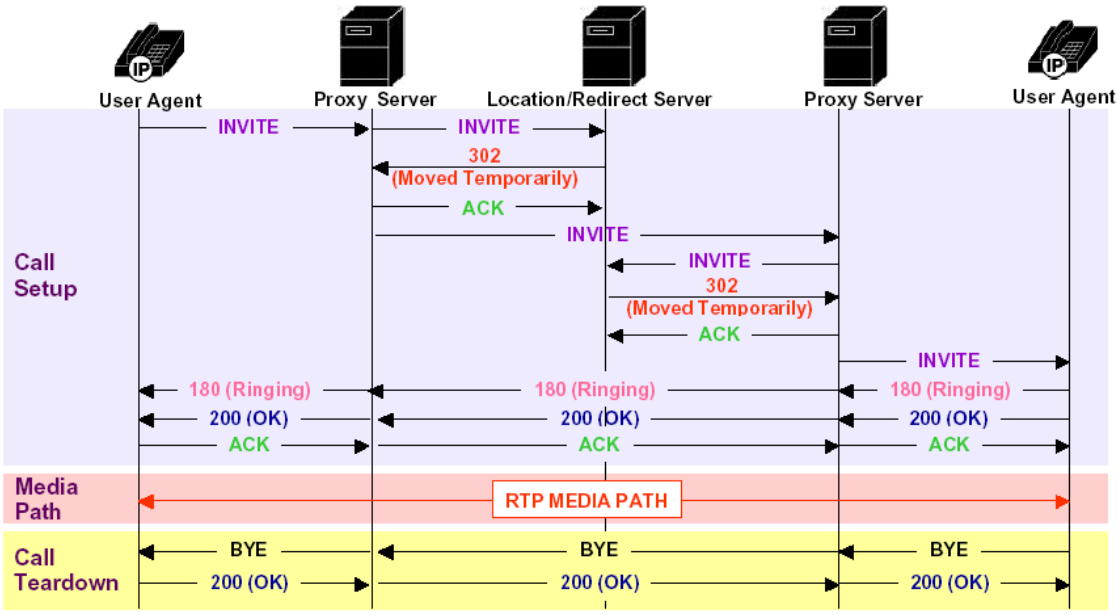
If a call is to be routed through a number of different Proxy servers Redirect server is used. When a caller UA sends an INVITE request to the redirect server, the redirect server contacts the location server to determine the path to the callee, and then the redirect server sends that information back to the caller. The caller then acknowledges receipt of the information.

SIP Request Through a Redirect Server



The caller then sends a request to the device indicated in the redirection information (which could be the callee or another server that will forward the request). Once the request reaches the callee, it sends back a response and the caller acknowledges the response. RTP is used for the communication between the caller and the callee.

SIP Call Setup and Teardown



Call Setup Diagram

1.5 SIP Methods (RFC 2453)

INVITE	- initiates sessions session description included in message body re-INVITEs used to change session state
ACK	- confirms session establishment can only be used with INVITE
BYE	- terminates sessions
CANCEL	- cancels a pending INVITE
OPTIONS	- capability inquiry
REGISTER	- binds a permanent address to current location; may convey user data (CPL Scripts)

Section II Mobility with SIP

In order to provide the full range of services for multimedia mobile communications, an application layer support is required which will be independent of the underlying technology.

2.1 Limitations of Mobile IP

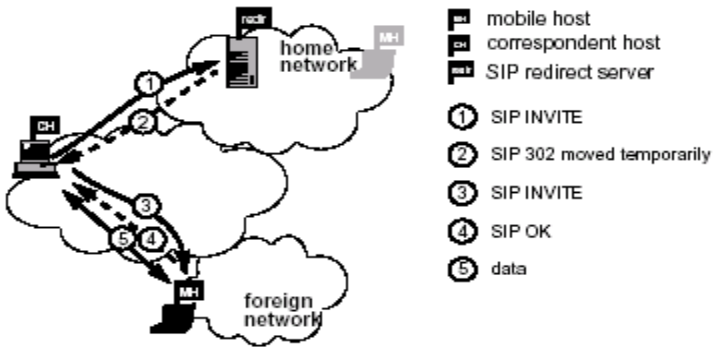
In the present mobile Ipv4 format every network host that allows its users to roam creates a home agent. Every network host that allows visitors creates a foreign agent, which registers any new mobile hosts and contacts the user's home agent giving it a care-of-address. When packets arrive at the user's home network they are routed through the home agent, resulting in triangular routing, which leads to delays. Also in Ipv4 and Ipv6 encapsulation adds between 8 to 20 bytes of overhead.
In the SIP registration mechanism a user identifier of the type user@host is bound to a temporary IP address as compared to mobile IP where a permanent IP address is bound to a temporary care-of-address. SIP provides with an mobility framework which is independent of the underlying transport network of the Service Provider.

2.2 SIP Mobility

SIP provides mobility at various levels, namely - terminal, personal, session and service mobility.

2.2.1 Terminal Mobility

When a SIP Mobile Host (MH) changes its IP address it re-registers with its " home " registrar server. If suppose the MH changes its location to a new host in that case the REGISTER request is proxied through the visitor registrar to the MH registrar. When an INVITE request for the MH reaches the home network the redirect server directs it to the foreign network.



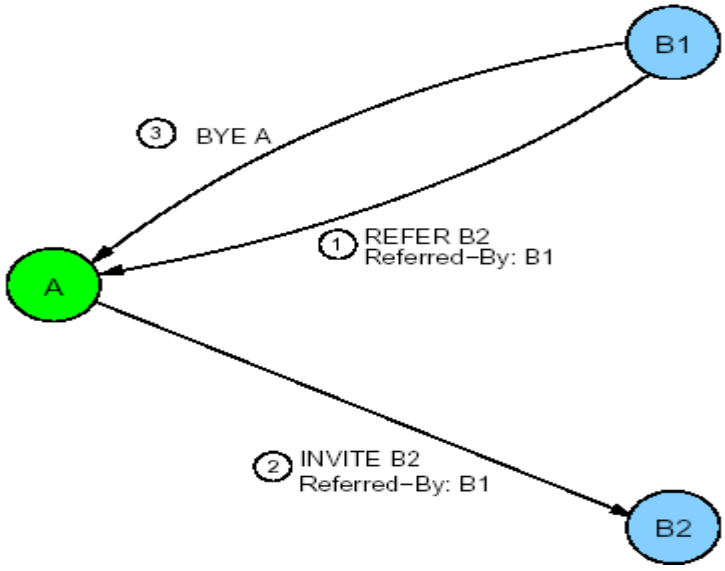
Precall Terminal Mobility in SIP

2.2.2 Personal Mobility

Personal mobility allows addressing a single user located at different terminals by the same logical address. Example, user Rana may want to be reachable on his PC, his SIP phone, his PDA .He may use all these devices at once or choose any. This is possible due the ability of proxies to fork requests, allowing him to be reached on any device via the same name.

2.2.3 Session Mobility

Session Mobility allows a user to maintain a media session even when changing terminals. IPv4 and IPv6 do not support session mobility. In SIP this can be done either by third party call control or by using the REFER request mechanism. In third party call control, the original session participant sends an INVITE request to the new session destination indicating the session parameters of the other participant. It also sends the session description to the other participant so that it can carry out further communication with the new destination. In the REFER mechanism a REFER request is sent to the participant informing of the new destination. The participant then negotiates with the new destination using normal INVITE messages.



Session Mobility by REFER Method

2.2.4 Service Mobility

Service mobility allows users to maintain access to their services even while moving or changing devices and service providers. The service data is incorporated in the 'home' server architecture associated with the user's address. For example, a user identified by rana@wonder.com would use the designated SIP server for the wonder.com domain to store his service information.

Section III Multimedia Conferencing Service based on SIP Architecture using JAVA Applications

3.1 Multiconferencing Requirements

We examine the setup of a multimedia conferencing service, which allows a central whiteboard application to communicate with various mobile agents on the field as and when desired. The main **features** of the service are as follows –

- The invitees to the conferencing session are mobile and can be moving from one network host to other.
- The service should be able to configure according to the capabilities of the communicating devices and setup corresponding compatible multimedia sessions.
- The central application can invite various mobile hosts to join the session .The various end users can also page the central application and join the conferencing session.
- The central application acts as a conference bridge allowing various attendees to communicate directly with each other.

3.2 JES – JAVA Enhanced SIP

The advantages offered by SIP in terms of user mobility and its flexibility makes it an ideal protocol for this setup. JES or Java Enhanced SIP is used which allows us to send Java applets, embedded in the SIP messages, to the various end agents. These applets act as intelligent end points on the hosts and set up the desired session. For instance the Java Applet may set up connections from the attendees machine to several video sources, connect to an electronic whiteboard to be shared for the meeting, start up a web browser to a page relevant to the meeting and set up all the audio paths with everyone. The Java Applet will also be clever enough to adapt to the different capabilities of the attendees host - for instance someone on a mobile may just have the audio set up, while a full multimedia caller might receive audio, video, data and web references.

3.3 Call Flow

The call flow for the case where a user dials the conference number is as follows-

- A user wanting to join a conference call sends a SIP invite message to the conference number from his terminal.
- The call is received on the multimedia conferencing system (MCS).
- The MCS send an ACK message back, which includes a Java Applet using the Java enabled SIP constructs.

- The users client receives the ACK and extracts and runs the Java Applet
- The Java Applet queries the exact capabilities of the users SIP client and host machine and initiates SIP sessions for the audio, video and data streams associated with the conference provided that the user's client and host can support them (for instance on a mobile - at least the current generation of voice orientated mobiles - the user might just have the audio stream set up as mentioned before).
- Depending on how the user has their JES security mechanisms set they may be prompted before the sessions are set up for the various media streams.
- Depending on the number of people in the call the SIP calls may be to either a conferencing bridge facility or may simply use a software based conferencing technique to connect the various parties.

The 'Require' request-header, incorporated into the SIP message header, indicates that Java-enhanced-SIP must be supported to process this message. The Java applet is either stored in a multipart MIME section in the body of the message or a URL to it is in the message.

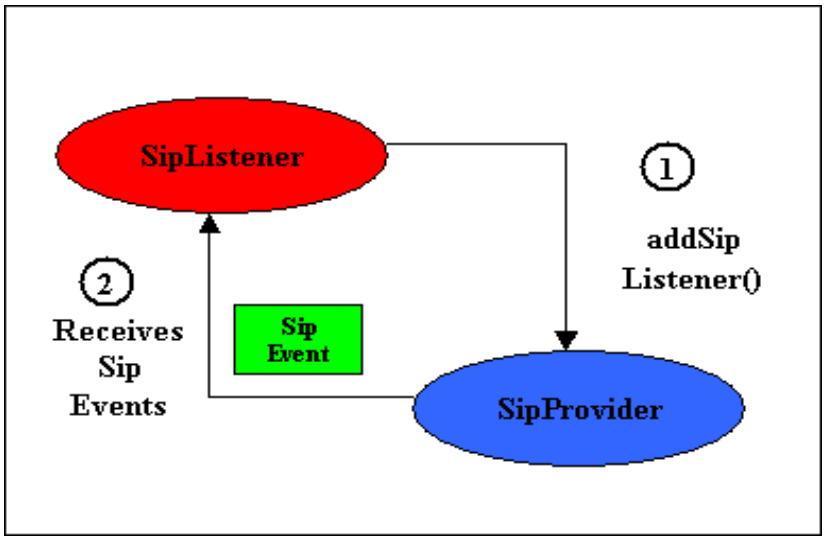
An example message header, which requires JES support :
Content-Type: multipart/mixed; boundary=3E4A567F4C8A (or URL for java applet)
Content-Length: ...
Require : org.ietf.sip.java-enhanced-sip

The Java applet creates SIP messages by interacting with the classes and methods provided by the JAIN SIP API, which acts as an interface between the application and the SIP Protocol Stack.

3.4 JAIN SIP API

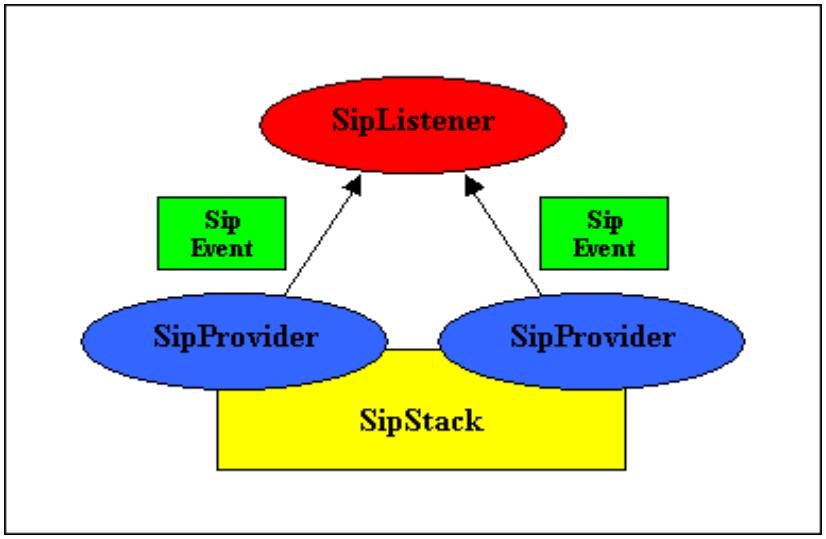
3.4.1 Overview

The implementation of the JAIN SIP API focuses around the SipListener and SipProvider interfaces. A Listener could be any SIP User application, and would use a Provider to send SIP messages into the network. To send a SIP message, a Listener would send Messages to a SipProvider.
The SipProvider interface defines the methods required to send SIP messages. This interface also defines the methods required to maintain a list of SipListeners. An implementation of this interface would listen for SIP messages from the stack and forward these messages to its registered SipListeners. The SIP messages would be sent as part of SipEvents (which also include the associated transaction id).



Jain SIP message passing

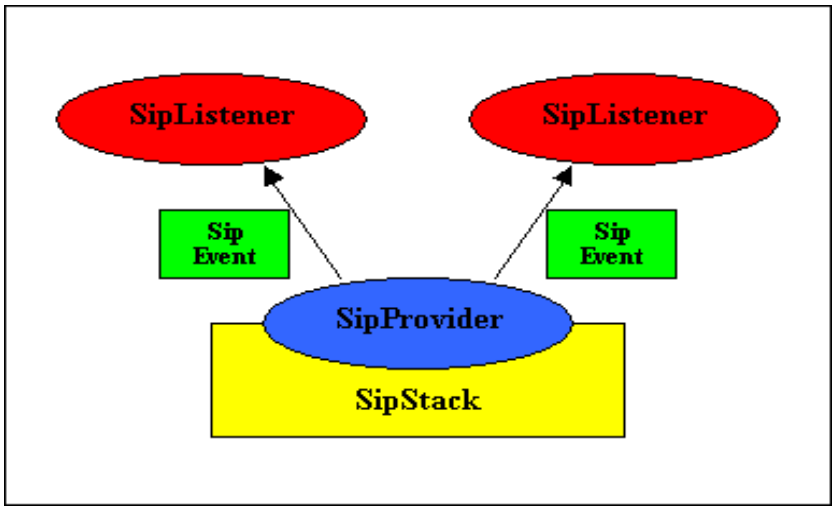
An object implementing the SipProvider interface would be vendor specific and would act as a proxy for a ListeningPoint of a SipStack. Zero or more SipListeners could register with a SipProvider, however the JAIN SIP API provides the ability to limit the number of SipListeners that may register at any one time.



SipListener registration with multiple SipProviders

The SipProvider interface defines the methods required to process any of the SipEvents sent from a Provider. An object implementing the SipListener interface would register as a SipListener of a SipProvider and would subsequently be able to receive SipEvents from that SipProvider.

When a received SIP message arrives at the SIP Stack, the SipProvider forwards the SIP message as SipEvents to its registered SipListeners. The diagram below illustrates how a SipEvent is distributed to SipListeners.



SipEvent Distribution

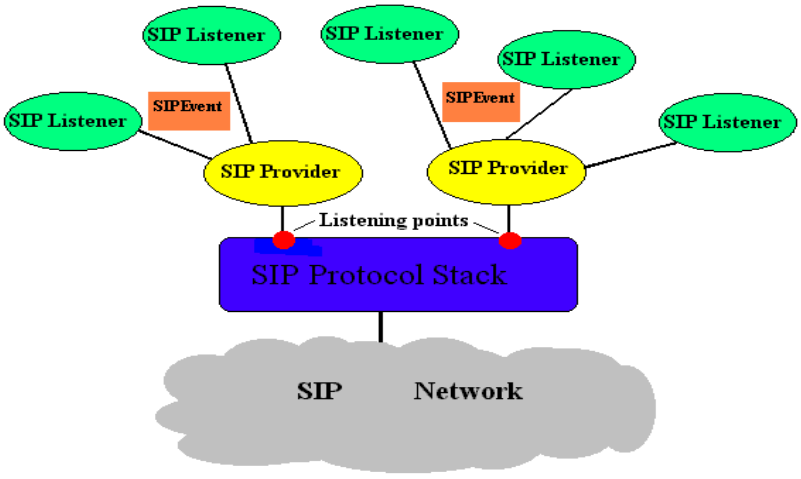
3.4.2 Future Developments

An interesting prospect arises out of the two tools examined so far, JES and JAIN SIP API. JES can be integrated within the API to enhance the functionality of the package. So we can have applets embedded in the messages to perform a variety of functions –

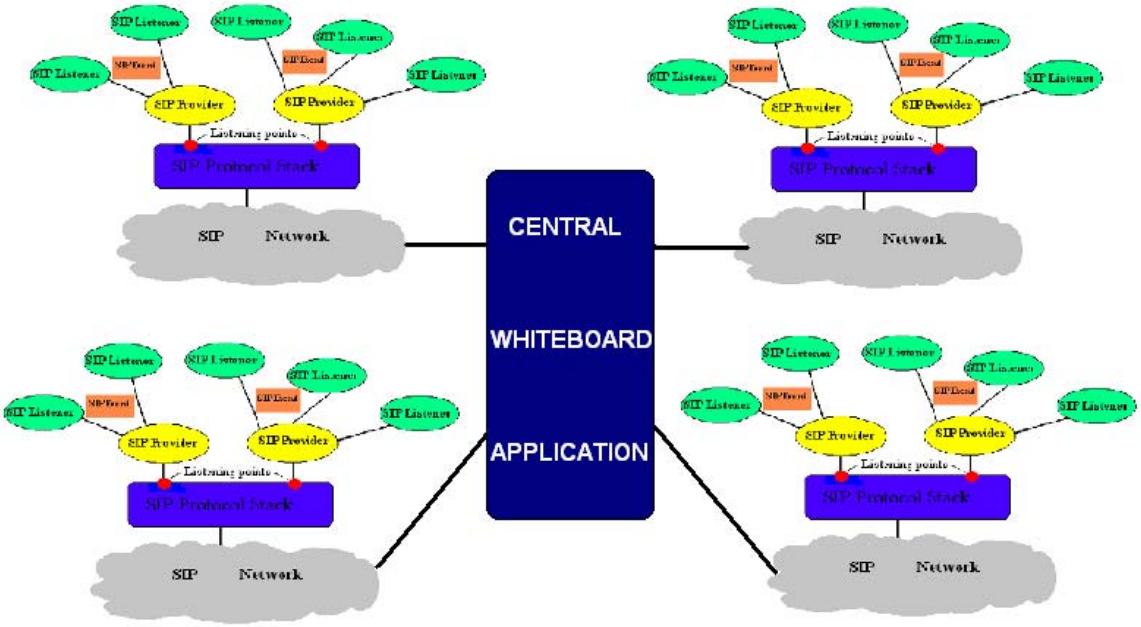
- Determining user agent capabilities to configure the various data session to be established, i.e. audio only, multimedia etc.
- Display user and session information on a real-time basis.
- To provide user-friendly display session management interfaces.

3.4.3 Multimedia Conferencing Setups

User Setup



Multiconferencing Setup

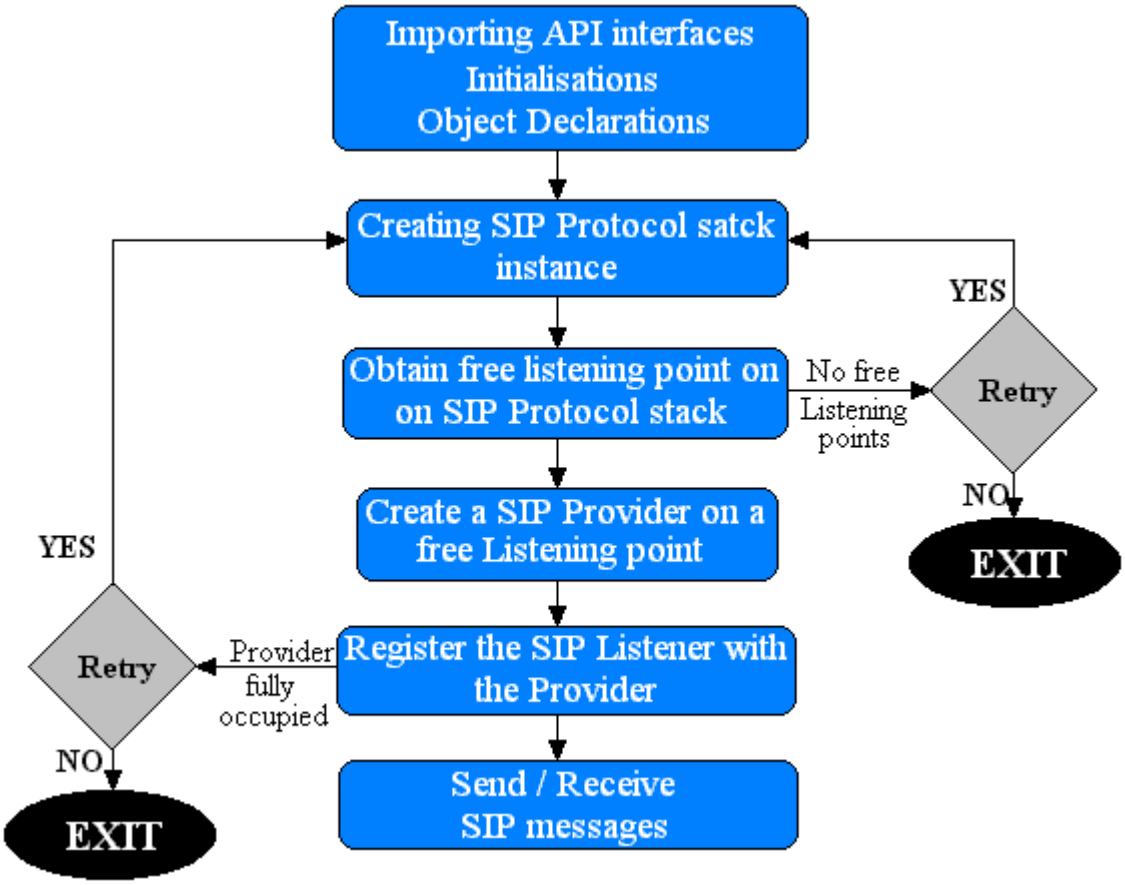


3.5 A JAIN SIP example

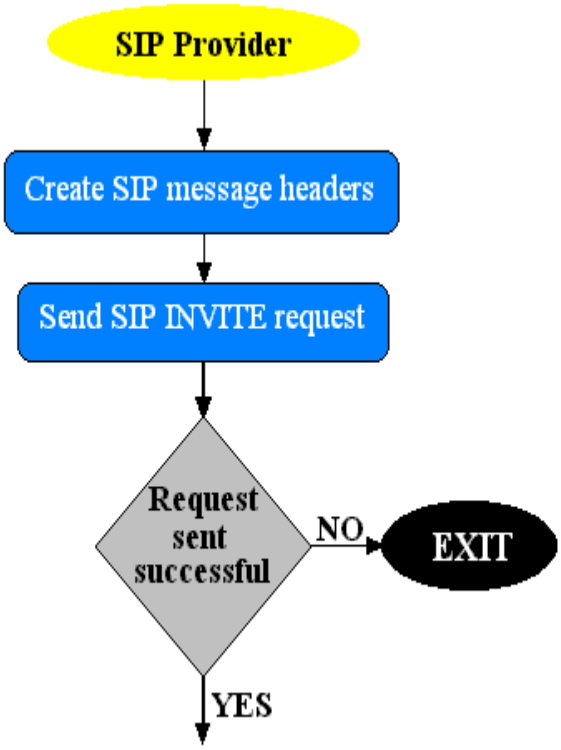
The following points illustrate how the JAIN SIP API can be used to send and receive SIP Messages. The example looks at the code of a Jain SIP User application and the steps the User application will use to create a SipProvider for communicating with the proprietary SIP stack.

3.5.1 SCHEMATICS

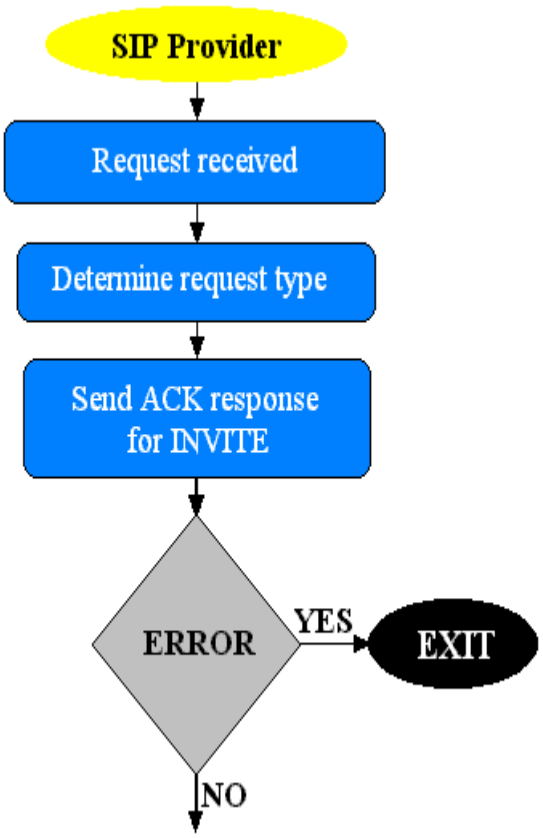
SIP Provider



Sending Message



Receiving Message



3.5.2 Sample Code

```
import jain.protocol.ip.sip.*;
import jain.protocol.ip.sip.header.*;
import jain.protocol.ip.sip.message.*;
import java.util.*;


public class Example implements SipListener
{
   private SipFactory sipFactory = null;
   private AddressFactory addressFactory = null;
   private HeaderFactory headerFactory = null;
   private MessageFactory messageFactory = null;
   private SipStack sipStack = null;
   private SipProvider sipProvider = null;
   private ListeningPoint[] listeningPoints = null;

   // Main
   public static void main(String[] args)
   {
      Example example = new Example();
      example.sendMessages();
   }

   public Example()
   {
      setup();
   }

   private void setup()
   {

      // Obtain an instance of the singleton SipFactory
      sipFactory = SipFactory.getInstance();

      // Set path name of SipFactory to reference implementation
      // (not needed - default path name)
      sipFactory.setPathName("com.example");

      try
      {
         // Create SipStack object
         sipStack = (SipStack)sipFactory.createSipStack();
      }
      catch(SipPeerUnavailableException e)
      {
         // could not find com.example.jain.protocol.ip.sip.SipStackImpl
         // in the classpath
```

```
            System.err.println(e.getMessage());
            System.exit(0);
        }

        // Set name of SipStack
        sipStack.setStackName("Reference Implementation SIP stack");

        // Try to get the SipStack's ListeningPoints and create a
        // SipProvider based on the first ListeningPoint
        try
        {
            listeningPoints = sipStack.getListeningPoints();
            sipProvider = sipStack.createSipProvider((ListeningPoint)listeningPoints.next());
        }
        catch(NullPointerException e)
        {
            System.err.println("Stack has no ListeningPoints");
            System.exit(0);
        }
        catch(ListeningPointUnavailableException e)
        {
            System.err.println(e.getMessage());
            System.exit(0);
        }

        // Register this application as a SipListener of the SipProvider
        try
        {
            sipProvider.addSipListener(this);
        }
        catch(TooManyListenersException e)
        {
            // A limit has been reached on the number of Listeners allowed per provider
            System.err.println(e.getMessage());
            System.exit(0);
        }
        catch(SipListenerAlreadyRegisteredException e)
        {
            // Already been added as SipListener
            System.err.println(e.getMessage());
            System.exit(0);
        }
    }

    // Process transaction timeout
    public void processTimeOut(String transactionId, boolean isServerTransaction)
    {
        System.out.println("Transaction " + transactionId + " timed out");
    }
```

```
// Process Request received
public void processRequest(String serverTransactionId, Request request)
{
   System.out.println("Request received with server transaction id "
   + serverTransactionId + ":\n" + request);
}

// Process of Response received
public void processResponse(String clientTransactionId, Response response)
{
   System.out.println("Response received with client transaction id "
                          + clientTransactionId + ":\n" + response);
   try
   {
      // Get method of respose
      String method = response.getCSeqHeader().getMethod();

      // Get status code of response
      int statusCode = response.getStatusCode();
      // If response is a 200 INVITE response, try to send an ACK
      if((statusCode == Response.OK) && (method.equals(Request.INVITE)))
      {
         sipProvider.sendAck(clientTransactionId);
      }
   }
   catch(SipException e)
   {
      System.err.println(e.getMessage());
      System.exit(0);
   }
 }



public void sendMessages()
{
   // Create INVITE Request to send

   SipURL fromAddress = null;
   NameAddress fromNameAddress = null;
   FromHeader fromHeader = null;
   SipURL toAddress = null;
   NameAddress toNameAddress = null;
   ToHeader toHeader = null;
   SipURL requestURI = null;
   CallIdHeader callIdHeader = null;
   CSeqHeader cSeqHeader = null;
   ViaHeader viaHeader = null;
   ArrayList viaHeaders = null;
   ContentTypeHeader contentTypeHeader = null;
   Request request = null;
```

```
try
{
   // create From Header
   addressFactory.createSipURL("caller", sipProvider.getListeningPoint().getHost());
   fromAddress.setPort(sipProvider.getListeningPoint().getPort());
   fromNameAddress = addressFactory.createNameAddress("Caller", fromAddress);
   fromHeader = headerFactory.createFromHeader(fromNameAddress);

   // create To Header
   addressFactory.createSipURL("callee", sipProvider.getListeningPoint().getHost());
   toAddress.setPort(sipProvider.getListeningPoint().getPort());
   toNameAddress = addressFactory.createNameAddress("Callee", fromAddress);
   toHeader = headerFactory.createToHeader(fromNameAddress);

   // create Request URI
   requestURI = (SipURL)toAddress.clone();
   requestURI.setTransport(sipProvider.getListeningPoint().getTransport());

   // Create CallIdHeader
   callIdHeader = sipProvider.getNewCallIdHeader();

   // Create CSeqHeader
   sipProvider.getNewCSeqHeader(callIdHeader, fromHeader, toHeader,   Request.INVITE);

   // Create ViaHeaders
   viaHeader = HeaderFactory.createViaHeader(sipProvider.getListeningPoint().getHost(),
                              sipProvider.getListeningPoint().getPort(),
                              sipProvider.getListeningPoint().getTransport());
   viaHeaders = new ArrayList();
   viaHeaders.add(viaHeader);

   // Create ContentTypeHeader
   contentTypeHeader = headerFactory.createContentTypeHeader("application",      "sdp");

   // Create INVITE Request
   request = messageFactory.createRequest(requestURI,
                                 Request.INVITE,
                                 callIdHeader,
                                 cSeqHeader,
                                 fromHeader,
                                 toHeader,
                                 viaHeaders,
                                 "body",
                                 contentTypeHeader);
}
catch(SipParseException e)
{
   // Implementation was unable to parse a value
   System.err.println(e.getMessage() + "[" + e.getUnparsable() + "]");
   System.exit(0);
}
```

```
        catch(SipException e)
        {
           // Another exception occurred
           System.err.println(e.getMessage());
           System.exit(0);
        }


        // Send Message
        String clientTransactionId = null;;
        try
        {
           // Send INVITE Request
           clientTransactionId = sipProvider.sendRequest(request);
        }
        catch(SipException e)
        {
          // SipProvider could not send INVITE Request
           System.err.println(e.getMessage());
           System.exit(0);
        }
    }
}
```

CONCLUSION

The overview of SIP architecture reveals that it is a better protocol for creating and managing multimedia sessions, without proposing extensive changes to the existing networks.

SIP also provides improved mobility support which overcomes certain drawbacks of the existing MobileIP framework and is of specific importance in the areas of real-time communication and multimedia sessions.

The JAIN SIP API provides a powerful tool to create platform independent applications, which do not require the application developer to directly interact with the propriety SIP protocol stack.

ACKNOWLEDGEMENTS

We are indebted to Prof. M.P.S Bhatia for his guidance and contribution in the making of this paper.

R E F E R E N C E S

1. **VoIP Overview -** Vovida.org (Online)
2. **Tanenbaum, A.S. -** Computer Networks (Prentice-Hall India)
3. **RFC2543 -** Session Initiation Protocol
4. **RFC3087 -** Control of service context using SIP Request.
5. **Mick 'o Doherthy -** Internet Draft, Java Enhanced SIP
6. **JAIN Overview, JAIN™**: Integrated Network APIs for the Java™ Platform(Online)
7. **S.Bessler, A.V.Nisanyan, K.Peterbauer,R Pailer,J.Stadler -** A Service Platform for Internet-telecom services using SIP.
8. **Henning Schulzrinne,Elin Wedlund -** Application Layer Mobility using SIP