National Institute of Standards and Technology

INFORMATION TECHNOLOGY LABORATORY

JAIN SIP: A SIP For the People! Architecture, Implementation, Programming

M. Ranganathan
Computer Scientist
Advanced Networking Technologies Division
N.I.S.T.
Co-Spec Lead for JAIN-SIP
Hacker at Large

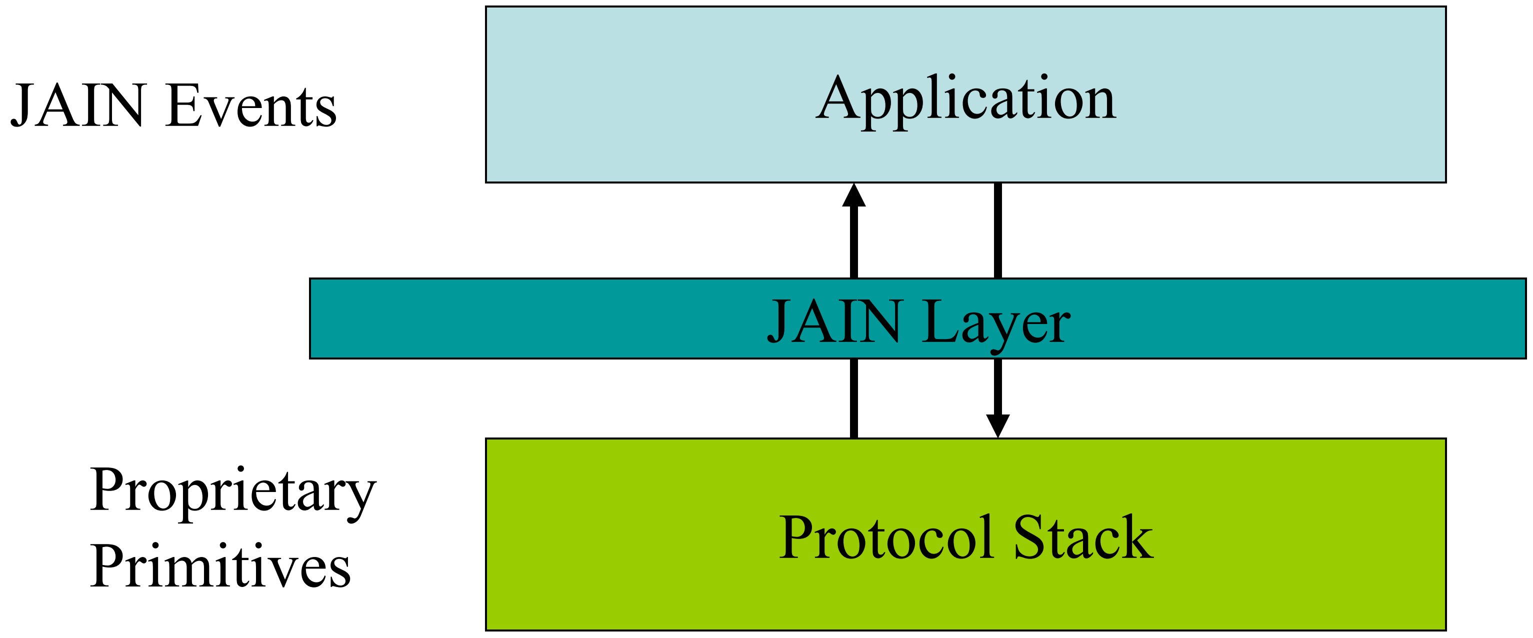
JAIN Day May 2004

SIP

- The call setup signaling protocol the Brave New World ™.
  - An application level protocol for setting up peer to peer sessions between communicating parties on the internet.
- Widely used for IP Telephony call setup, instant messaging and presence.
- Is defined in the core RFC 3261.
  - Extensions are defined in several other RFCs.
- Is connectionless and Transport agnostic
  - Protocol state is carried in the Message.
  - Protocol state does not depend on connection state.

JAIN

JAIN Provides an event-layer abstraction for applications.

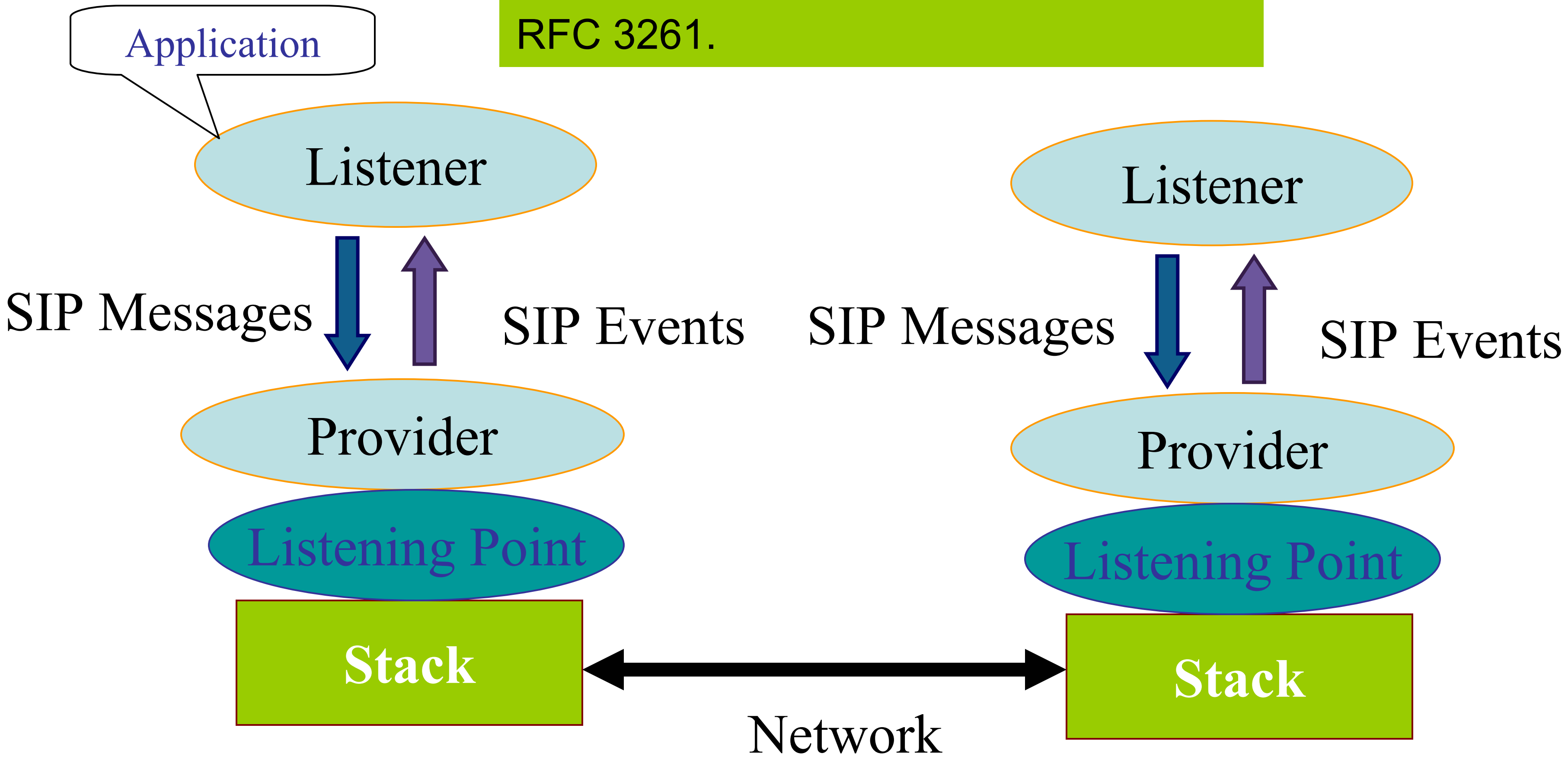


JAIN SIP

- Java-standard interface to a SIP signaling stack.
  - Standardized the interface to the stack.
  - Standardized the events and event semantics.
  - Application portability - verified via the TCK.
- Designed for the developer who requires fine grained access to the SIP protocol.
- JAIN SIP can be utilized in a user agent, proxy, or imbedded into a service container.
- Provides stack support for the following:
  - RFC 3261, 2976, 3262,
  - RFC 3265, 3311, 3428
- Joint Spec leads NIST and SUN.

JAIN SIP Architecture

The Listener is modeled after the TU of RFC 3261.

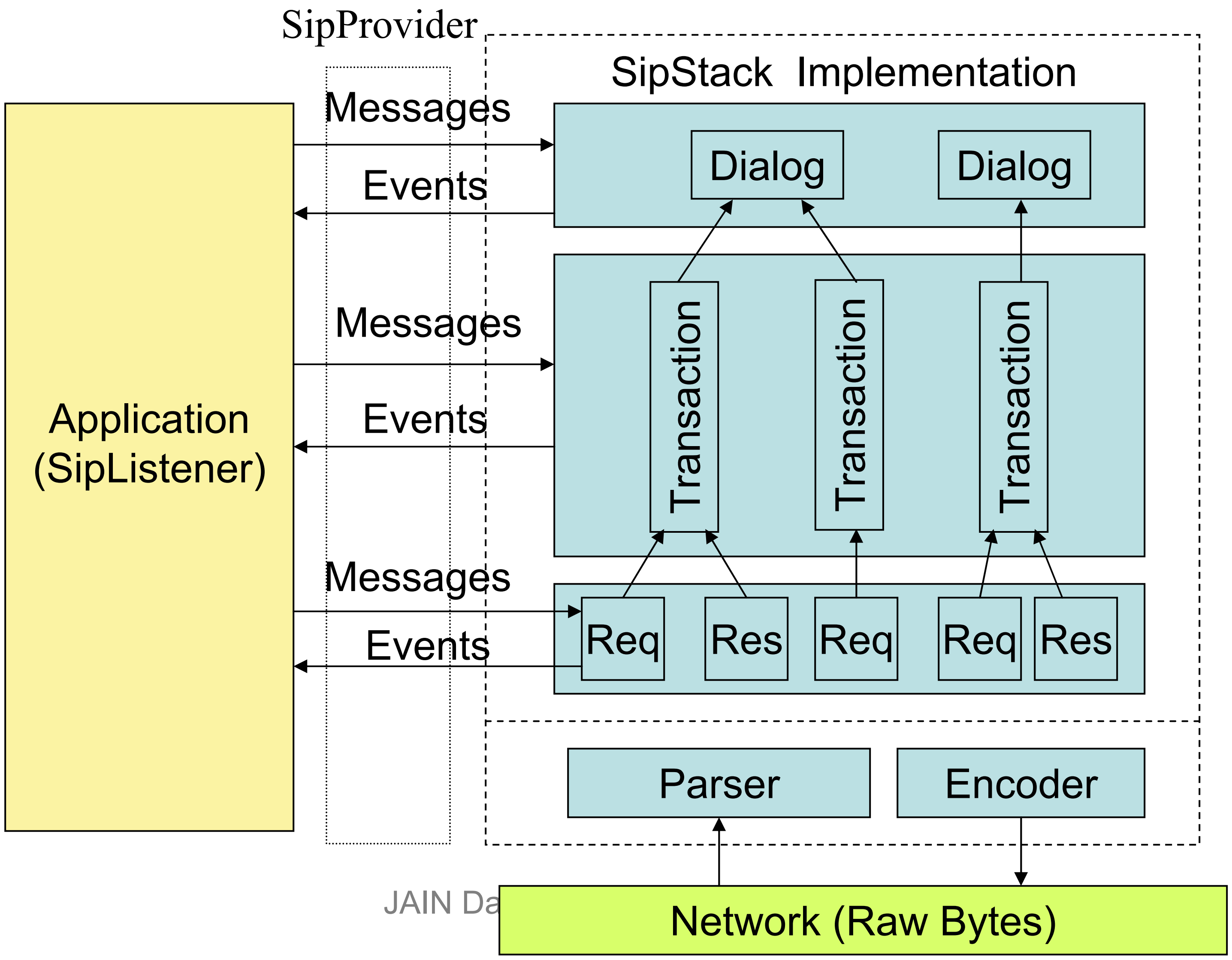


JAIN SIP is meant for people who understand RFC 3261!

NIST National Institute of Standards and Technology

ITL Information Technology Laboratory

JAIN-SIP Structure



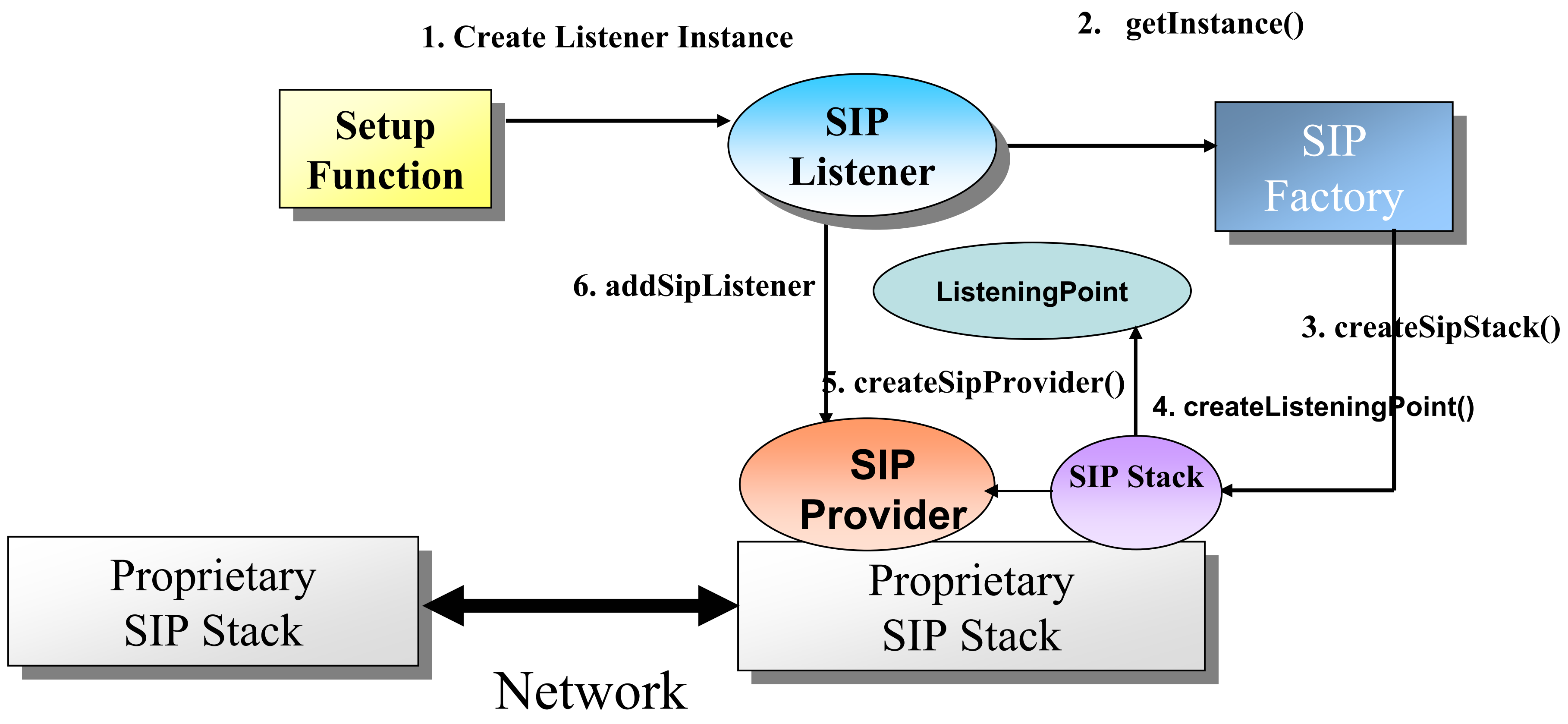
Services provided by JAIN SIP

- Provide methods to format and send SIP messages
- Parse incoming messages and enable application to access to fields via a standardized JAVA interface.

Services provided by JAIN SIP

- Invoke appropriate application handlers when protocol significant (message arrivals, *Transaction* time-outs)
- Provide *Transaction* support and manage *Transaction* state and lifetime on behalf of a user application.
- Provide *Dialog* support and manage *Dialog* state and lifetime on behalf on a user application.

JAIN SIP Object Architecture



The Application

- Typically runs in the same JVM as the stack reference (JAIN-SIP does not mandate this).
- Implements the *SipListener* interface
- Creates the stack and adds listening points.
- Registers an implementation of the *SipListener* interface to interact with the SIP Stack.
- Gets *SipProviders* from the Stack to send messages and create SIP *Transactions*

The Application

- Application MUST go via the *SipProvider* for all SIP messaging (no sneaking SIP messages through sockets)
  - Application Sends messages and access stack objects via the *SipProvider*.
- Application receives notifications from the stack as [Request/Response/Timeout] Events.

SipStack

- Manages Listening Points and Providers.
- SipStack associated with an IP address.
  - Can Have multiple Listening points.
- Application can have multiple SipStacks.
- Cannot be deleted once created.
- Instantiated by the SipFactory and initialized with a Property set.
- javax.sip.* properties are reserved and names defined for stack configuration properties.

SipProvider Interface

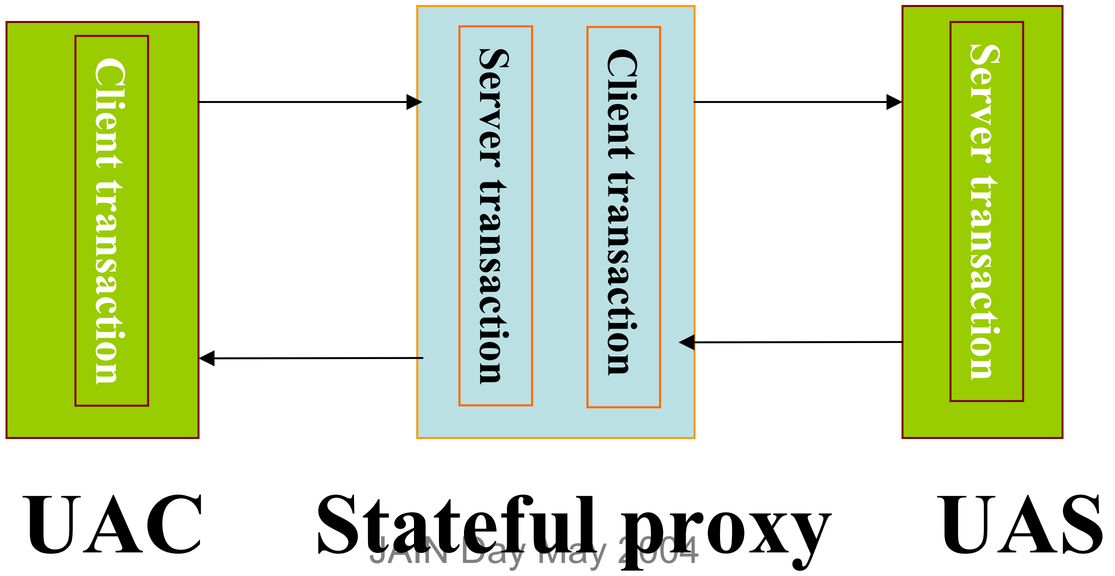
- Send Request's either statefully via client transactions or statelessly.
- Send Response's to a recently received Requests either statefully via server transactions or statelessly.
- Register a SipListener to the SipProvider.
  - Notifies Registered Listener of Events (Request/Response/Timeout).
- De-register a SipListener from the SipProvider.
  - Once de-registered, no longer receive Events from SipProvider.
- New Client and Server Transaction methods.
- Listening Point manipulation methods.
  - Only one provider per listening point.

SipListener Interface

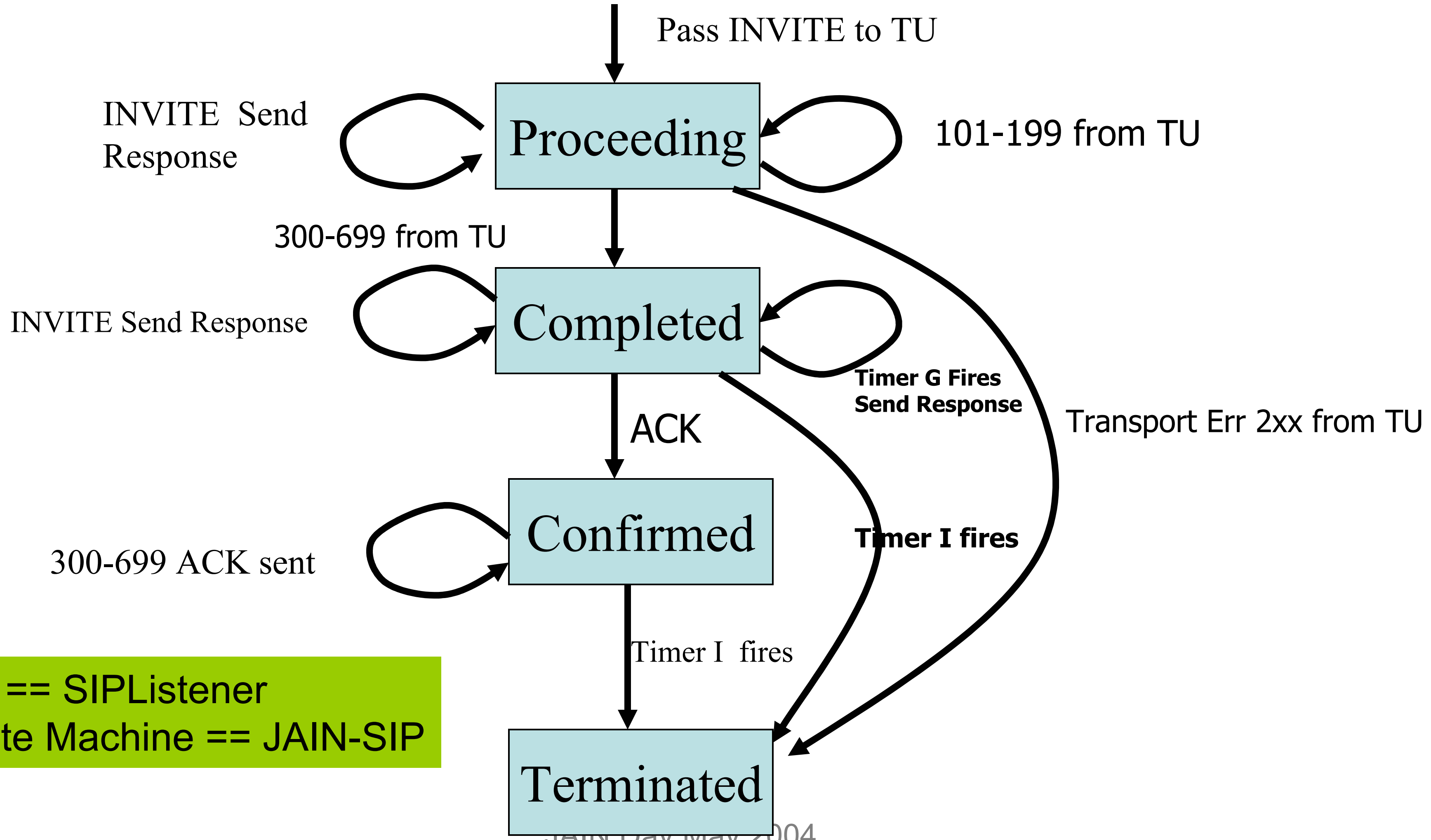
- A single SipListener per SipStack
  - all SipProviders associated to a Sipstack have the same SipListener.
- Process Request's either statefully or statelessly dependent on application.
- Process Response's to a recently sent Requests.
- Process Transaction timeouts and retransmits Timer events.

SIP Transactions

SIP transaction consists of a single request and any responses to that request.
May be identified by the incoming SIP Message (Request or Response).
JAIN-SIP Maintains State Machine for Transaction



INVITE Server Transaction State Machine -- approximately! (See RFC 3261 Ch 17)



TU == SIPListener
State Machine == JAIN-SIP

Statefull or Stateless?

- Statefull request handling simplifies design
  - stack takes care of messy transaction details, retransmission, filters duplicates etc.
- Stateless handling is "Fire and Forget"
  - Enhances scalability
  - Good for building Proxy Servers (especially IM and Presence).
- You can have both statefull and stateless handling within a given application.
- JAIN-SIP maintains an on-to mapping from SIP Messages to transactions (given SIP request may only map to one transaction).

Client Transactions (RFC 3211 Ch 17)

- Request may be sent out statefully or statelessly.
  - Stateless requests are sent using SipProvider
  - Stack implements a transaction state machine to support statefull requests.
- ClientTransaction is created to send outgoing Request statefully.
  - Application requests a new *ClientTransaction* from the *SipProvider*.
  - Application uses the *ClientTransaction* to send the Request.
- When a Response arrives, the Stack possibly associates a previously created ClientTransaction with the response and passes up to the Application .

Dialog Layer (RFC 3261 Ch. 11)

- Dialog is a peer-to-peer association between communicating SIP endpoints
  - Dialogs established by successful completion of Dialog creating Transactions.
  - Not all transactions create Dialogs.
  - A Transaction may belong to exactly one Dialog.
  - Dialog may map to multiple Transactions.
  - JAIN-SIP has extension methods to extend the standard set.
  - Dialog has a simple state machine which depends upon the state of underlying transaction that creates/terminates the Dialog.

Dialog Layer (RFC 3261 Ch. 11)

- SIP messages carry enough state to identify the Dialog directly from the message.
- For statefull operation: jain-sip maintains a dialog pointer for every transaction.
  - Dialog d = transaction.getDialog();
  - Returns null for a transaction that cannot map to a dialog.

Dialog Support

- Application may store its state using an opaque data pointer which can be accessed using Dialog.[set/get]ApplicationData
- Dialogs store Route Sets and other useful information that can be used for subsequent requests within the Dialog. JAIN-SIP makes this easy:

```
Request request = dialog.createRequest(Request.ACK);
ClientTransaction ct = sipProvider.createClientTransaction(request);
dialog.sendRequest(ct);
```

Server Transactions

- Application may choose to handle non-Dialog-Creating request either statefully or statelessly.
- Application may choose to create Server Transaction when Dialog-Creating Request arrives:
  - Automatically creates a Dialog and marks its state to null.
  - Must be done when the event is delivered (cannot be deferred).
  - All subsequent requests within the Dialog are assigned a Server Transaction (handled Statefully) within the Dialog.

Stack Configuration Parameters

- javax.sip.IP_ADDRESS
  - IP Address on which the stack listens for incoming messages (port is specified when listening point is created).
- javax.sip.STACK_NAME
  - A string identifier for the stack. You can use this for logging and management but the stack does not use it for processing.
- javax.sip.OUTBOUND_PROXY
  - Has the format ipAddress:port/transport
  - Is passed un-interpreted to the Router class.

Stack Configuration Parameters

- javax.sip.ROUTER_PATH
  - Router is a class that is consulted by the stack to route out of dialog messages.
  - Application may provide its own router implementation.
- javax.sip.EXTENSION_METHODS
  - Extension methods are methods outside the standard ones that may result in Dialog Creation.
- javax.sip.RETRANSMISSION_FILTER
  - Simplifies the complexities of handling INVITE transactions. Retransmission is handled in the stack.

Application Snippet Stack Creation

Initialize Stack using SipFactory:

```
try {
   Properties properties = new Properties();
   properties.setProperty("javax.sip.IP_ADDRESS",
                                    "129.6.55.181");
   properties.setProperty("javax.sip.OUTBOUND_PROXY",
                             "129.6.55.182:5070/UDP");
   ……// Other initialization properties.
   try {
       sipStack = sipFactory.createSipStack(properties);
   } catch(SipException e) {
             System.exit(-1);
   }
}
```

Application Snippet Request Creation

Initialize Request using Factories:

```
try {
   SipURI requestURI = addressFactory.createSipURI
                               (toUser, toSipAddress);
   // … Create other headers
   Request request = messageFactory.createRequest
       (requestURI, Request.INVITE, callIdHeader,
       cSeqHeader, fromHeader, toHeader,
       viaHeaders, maxForwards);
}
```

Application Snippet Processing Requests

Handle incoming messages as Events:

```
try {
   public void  processRequest(RequestEvent
        requestEvent) {
        Request request =
              requestReceivedEvent.getRequest();
        ServerTransaction st =
              requestEvent.getTransaction();
        // do request specific processing here
    }
}
```

Some RI-Specific Features

- JAIN-SIP is a minimal spec. Additional features are often useful in building SIP network elements.
- gov.nist.javax.sip.CACHE_SERVER_CONNECTIONS
  – Caches incoming TCP connections. If false, server will drop connection when not in use. If true then server will keep connection alive until client closes it. (default is true).
- gov.nist.javax.sip.CACHE_CLIENT_CONNECTIONS
  – Caches outgoing client connections. If true then client will keep connection open after transaction completion.
- gov.nist.javax.sip.MAX_MESSAGE_SIZE
  – Maximum size of incoming TCP messages. Allows servers to guard against DOS attacks.

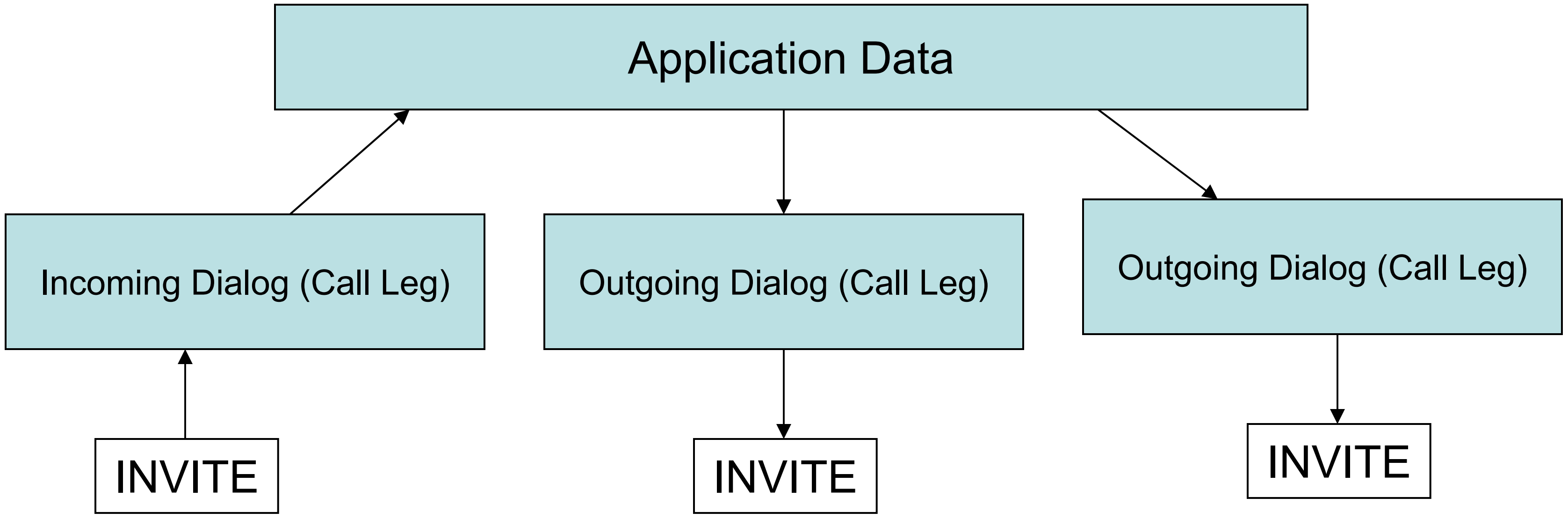
Some RI-Specific Features

- gov.nist.javax.sip.MAX_CONNECTIONS
  - Max number of simultaneous TCP connections that are supported by the RI (default is “infinity”).
- gov.nist.javax.sip.READ_TIMEOUT
  - Timeout (ms) between successive TCP reads for message content/header to prevent DOS attacks (default is “infinity”).
- gov.nist.javax.sip.STUN_SERVER
  - IP Address and port of the STUN server.

JAIN-SIP in the Server

- Servers may be statefull or stateless or combine the two modes of behavior.
- A Statefull Proxy server keeps associations between call legs (Dialogs).
- The application pointer facility greatly simplifies management of application state that may be associated with the Dialog.

Implementation Tips JAIN-SIP in a Statefull Server


Application Data
Incoming Dialog (Call Leg)
Outgoing Dialog (Call Leg)
Outgoing Dialog (Call Leg)
INVITE
INVITE
INVITE

Use ApplicationData pointer from Dialog to track associations
Advantage – when all dialogs referencing the Application Data go away, so does the application data.
When an event comes in for one of the Dialogs, you can get all the other ones

Implementation Tips JAIN-SIP in an APPLET

- You can imbed JAIN-SIP in an applet.
  - jar file is about 565 K (un-optimized with symbolic info).
  - Stable heap size about 1 Mb.
  - Combine IP Telephony signaling and web to build converged applications.
- Delete all providers and release all references for stack cleanup.
- Make sure all outstanding requests are consumed before stack cleans up.
  - Put in a sleep before you do stack cleanup.

Stack Clean Up

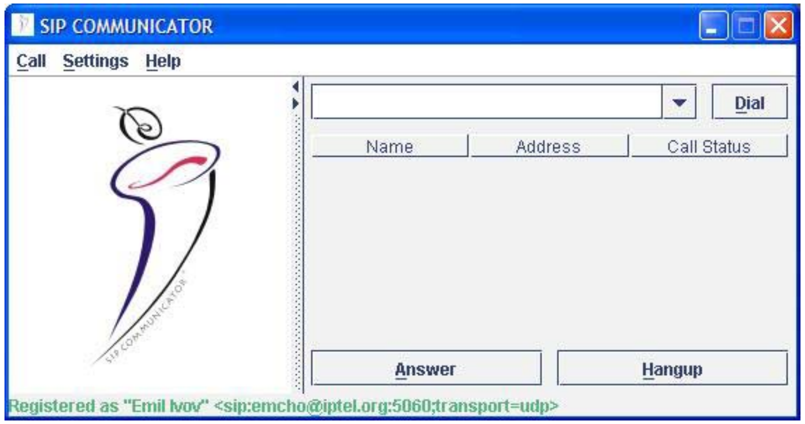
```
sipStack.deleteListeningPoint(tcpListeningPoint);
sipStack.deleteListeningPoint(udpListeningPoint);
// This will close down the stack and exit all threads
tcpProvider.removeSipListener(this);
udpProvider.removeSipListener(this);
while (true) {
        try {
                sipStack.deleteSipProvider(udpProvider);
                sipStack.deleteSipProvider(tcpProvider);
                break;
        } catch (ObjectInUseException  ex)  {
                try {
                        Thread.sleep(2000);
                } catch (InterruptedException e) {
                        continue;
                }
        }
}
```

JAIN-SIP in an APPLET

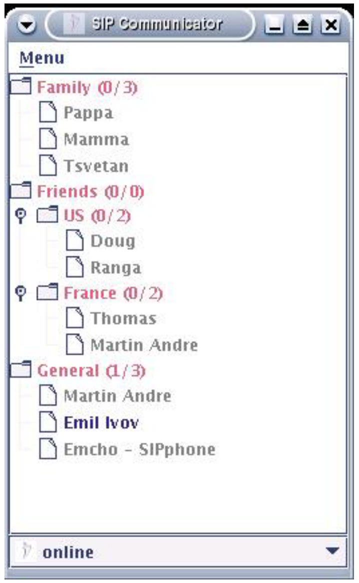
- Applet Security: Need to proxy all SIP Messages to the server from where you downloaded Applet.
  - Client MUST cache connections.
- Use Applet Initialization params to configure Applet.
- If firewall does not allow UDP, can use TCP + JMF to encapsulate voice MESSAGEs.
  - See jain-sip-applet-phone.dev.java.net.

SIP Communicator

- A fully functional 100% Pure Java SIP User Agent
- Hosted at http://sip-communicator.dev.java.net
- Based on the JAIN-SIP RI from NIST and JMF-2.1.1e
- Supports conversations over both IPv6 and IPv4.
- Uses the Stun4J stack to handle NATs
- Simple Messaging and Presence are almost there.



Registered as "Emil Ivov" <sip:emcho@iptel.org:5060;transport=udp>



STUN (RFC 3489)

- STUN (Simple Traversal of UDP over NAT)
  - NAT BOX address/port mapping discovery protocol.
  - Client talks to STUN server to determine address by which it is known to outside world.
- When STUN is enabled (see property above), SipStack.getIpAddress() returns the IP Address mapped by STUN.
- ListeningPoint.getPort() returns mapped port.

STUN (RFC 3489)

- RI incorporates STUN support
  - uses stun4j stack
  - Application needs to specify STUN server address/port in stack configuration property.
  - STUN client talks to a server to determine port mappings.
- Application must set remote IP address and port as obtained by STUN in *Contact* header.
- Application is responsible for sending periodic refresh messages to refresh STUN bindings (in case NAT box is reset)

More RI goodies

- RI includes a generalized pattern matcher facility.
  - Match feature not part of JAIN SIP but could be useful for building testers and such.
- Each field of a SIP Message can be replaced by a regular expression.
  - gov…..SIPMessage.match(SIPMessage template) returns true if the template matches the message.
  - Is aware of SIP Message header ordering equivalency rules, case sensitivity rules etc.
  - Match complexity depends upon Reg-exp complexity.

More RI Goodies

- SIP Transaction aware trace visualization
  - Trace logging can be turned on using gov.nist.javax.sip.TRACE_LEVEL and gov.nist.javax.sip.SERVER_LOG
  - Trace visualization tool to view message traces is included with the RI (useful for protocol debugging).

Take a walk On the Wild Side Converged Services

- The real appeal to Internet Telephony
  - New classes of applications / capabilities that integrate telephony with existing WWW/Internet applications.
- SIP Converged Services
  - may invoke API for a diverse range of operations (eg. Mail, http) based on information contained in the SIP headers.
  - Want a security policy for such services that can tie in diverse API into one framework.
- How to define security policy for SIP services without constraining the programming model?

NIST National Institute of Standards and Technology

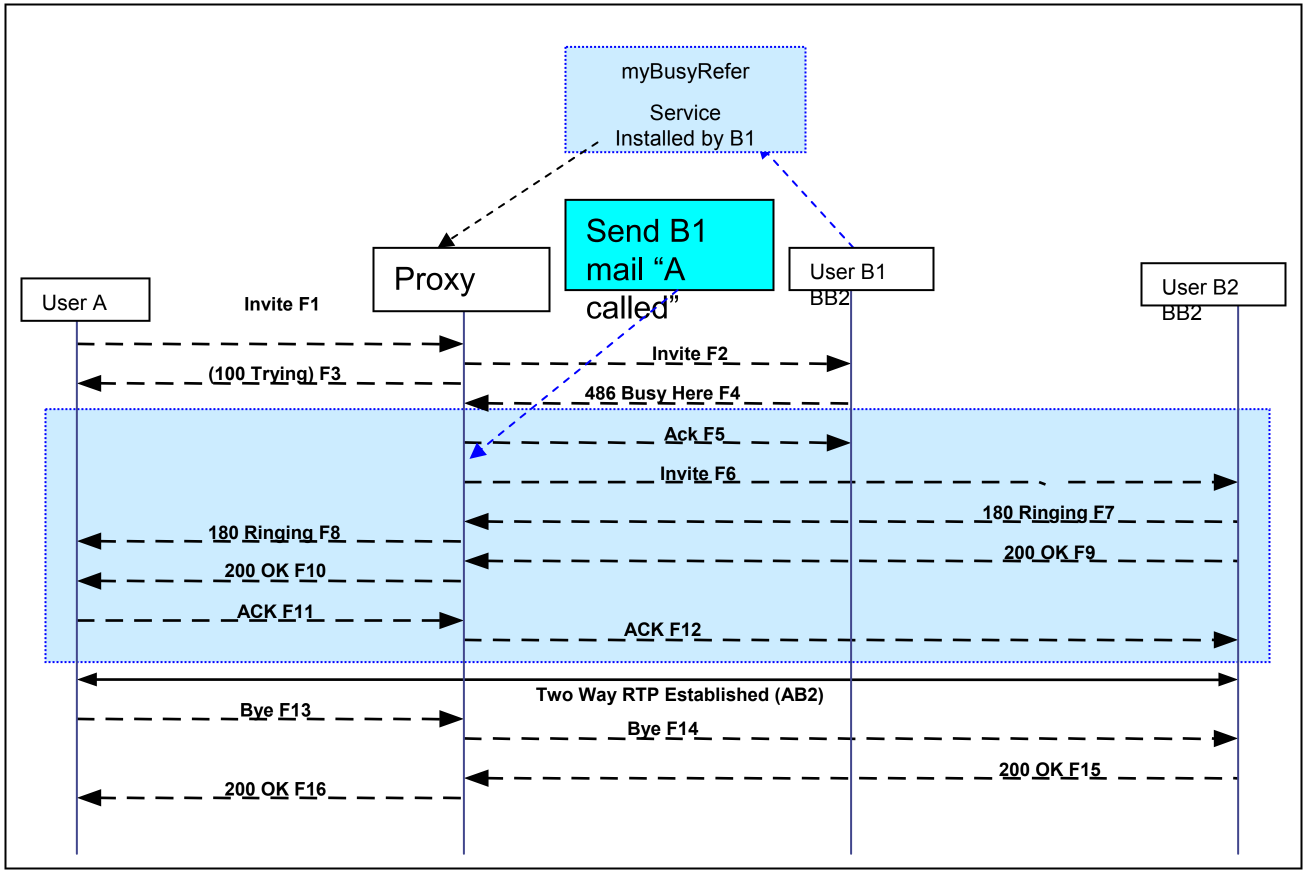
INFORMATION TECHNOLOGY LABORATORY

A walk on the wild side Converged Service

May want to use parts of the SIP message to invoke other API (eg. mail API) When busy signal comes in, take From and To headers and use them as mail from and to headers.

```
SIP/2.0 486 Busy Here
Call-ID: a3c321364fd5f1ffbcc206472e4a115e@129.6.55.82
CSeq: 1 INVITE
From: "The Master Blaster" <sip:BigGuy@here.com>;tag=12345
To: "The Little Blister" <sip:LittleGuy@there.com>
Via: SIP/2.0/TCP 129.6.55.82:5060;branch=z9hG4bK0958533d7e6d3c3c1cfdbf7d
Max-Forwards: 70
Contact: "Shootme" <sip:129.6.55.82:5070>
```

A Simple SIP Service Example



Threats: How can the server prevent spoofing? Need a composite security framework that spans both SIP and mail.

```
permission gov.nist.security.permissions.MailPermission “*.nist.gov”,
 “EventType = ResponseEvent, statusCode==486,
 transactionState==COMPLETED_STATE,
Mail.from==response.From, Mail.to==response.to”
```

Use Bytecode Rewriting to correlate outgoing mail headers
to sip headers and restrict behavior of service.
Requires NO changes to JAIN-SIP
You can also use this technique for ensuring completion for Listener
methods (bytecode allowances etc.)

NIST
National Institute of Standards and Technology

INFORMATION TECHNOLOGY LABORATORY

Future Wish List

- Incorporate a Security/Resource Framework into JAIN-SIP. Possible features of framework:
  - Support some of the DOS attack safety features currently not part of spec.
  - Restrict URLs that application may call / respond to.
  - Restrict ability to create Transaction/Dialog/Stateless behavior.

URLs of interest

- http://jain-sip.dev.java.net
- http://jain-sip-presence-proxy.dev.java.net
- http://sip-communicator.dev.java.net
- http://sip-for-me.dev.java.net
- http://jain-sip-applet-phone.dev.java.net
- http://jain-sip-3pcc.dev.java.net
- http://stun4j.dev.java.net
- http://jain-sip-appserver.dev.java.net
- http://www.jcp.org/aboutJava/communityprocess/final/jsr032/index2.html

Acknowledgements

- Notable contributions by:
  - Phelim O'Doherty (Co-spec lead from Sun)
  - Emil Ivov (Student at ULP)
  - Olivier Deruelle (Guest researcher at NIST)
  - Jean Deruelle (Guest researcher at NIST)
  - The nist-sip user community and early adopters.
- Stun support for RI is based on work by Emil Ivov.
- Thanks to the NIST Advanced Networking Technologies Division .
- Thanks to the US tax payer NIST SIP is in the public domain.