Interfacing Java to SIP
The story of the JAIN SIP API

Brian O’Neill
Technical Architect, Gestalt LLC
bone@alumni.brown.edu
mobile: 215.588.6024
**Goodspeed:** Well, I'm one of those fortunate people who like my job, Sir. Got my first chemistry set when I was seven, blew my eyebrows off, we never saw the cat again, been into it ever since.
Goal:
- A conversational understanding of SIP and a thorough working knowledge of the JAIN SIP API v1.1.

Definitions:
- **SIP** is the Session Initiation Protocol established by the IETF to create a new asynchronous, extensible communications protocol.
- **JAIN SIP** is the standardized Java interface to the Session Initiation Protocol for desktop and server applications.
SIP is powerful enough that:

- The computer savvy sixteen year old down the street can create a phone company in his parent’s basement on an old 486. And JAIN SIP enables him or her to use Java to do it.
Protocol lineage:
IP begat network. Networks begat TCP.
TCP begat the internet. The internet begat HTTP.
HTTP begat the web, which begat…

WARNING:
SIP is just a protocol, nothing more and nothing less.

Brian O’Neill (bone@alumni.brown.edu)
Network Architecture

- Think, “E-mail meets I.M.”
  - Inter-Enterprise
    - Instant Messaging
    - Buddy-Lists
    - VoIP
SIP Roles

- User Agents and their information
Applications receive events, and send messages.

Underneath the covers, the SIP stack implementation manages the processing and life-cycles of the requests, responses, transactions, and dialogs. It also manages the mappings between them.
A dialog is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request.

States:
Early, Confirmed, Completed, Terminated
The Life-Cycle of a Transaction

- Client Transactions

INVITE Transaction

Non-INVITE Transaction
The Life-Cycle of a Transaction

- Server Transactions

INVITE Transaction

Non-INVITE Transaction
### Anatomy of a SIP Message: Sniff, Sniff

<table>
<thead>
<tr>
<th>Method</th>
<th>Request URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Line:</td>
<td>INVITE sip:<a href="mailto:bob@biloxi.com">bob@biloxi.com</a> SIP/2.0</td>
</tr>
<tr>
<td>Headers:</td>
<td>Via: SIP/2.0/UDP</td>
</tr>
<tr>
<td></td>
<td>biloxi.com:5060;branch=z9hG4bKnashds7</td>
</tr>
<tr>
<td></td>
<td>Max-Forwards: 70</td>
</tr>
<tr>
<td></td>
<td>To: Bob <a href="">sip:bob@biloxi.com</a></td>
</tr>
<tr>
<td></td>
<td>From: Alice <a href="">sip:alice@biloxi.com</a>;tag=456248</td>
</tr>
<tr>
<td></td>
<td>Call-ID: 843817637684230@998sdasdh09</td>
</tr>
<tr>
<td></td>
<td>Cseq: 1826 REGISTER</td>
</tr>
<tr>
<td></td>
<td>Contact: <a href="">sip:bob@192.168.0.4</a></td>
</tr>
<tr>
<td></td>
<td>Expires: 7200</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 32</td>
</tr>
<tr>
<td>Content:</td>
<td>...message payload...</td>
</tr>
</tbody>
</table>
### Anatomy of a SIP Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Status Line:</th>
<th>SIP/2.0 200 OK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Headers:</strong></td>
<td>Via: SIP/2.0/UDP</td>
<td>foo.com;branch=z9hG4bKnashds8</td>
</tr>
<tr>
<td></td>
<td>To: Bob <a href="">sip:bob@foo.com</a>;tag=a6c85cf</td>
<td></td>
</tr>
<tr>
<td></td>
<td>From: Alice <a href="">sip:alice@bar.org.</a>;tag=6987</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Call-ID: a84b4c76e666710</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cseq: 314159 INVITE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact: <a href="">sip:bob@192.168.0.4</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content-Type: application/sdp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content-Length: 131</td>
<td></td>
</tr>
<tr>
<td><strong>Content:</strong></td>
<td>... SDP Information ...</td>
<td></td>
</tr>
</tbody>
</table>
SIP is the signaling component of the interaction, but does *not* get involved in the actual media.

- This is the job of the Session Description Protocol (SDP)
- What about Java support for the media? JMF?
Presence Models

- Edge vs. Aggregate
  - “Follow me”
  - Escalation
Invitations

alpha.com

alice@alpha.com

INVITE

200

ACK

SESSION

beta.com

bob@beta.com

INVITE

200

ACK

ACK

ACK

INVITE

200

200

200

INVITE

200

SBC

Brian O’Neill (bone@alumni.brown.edu)
Interface SipStack

All Known Implementing Classes:

*SipStackImpl*

public interface SipStack

This interface represents the management interface of a SIP stack implementing this specification and as such is the interface that defines the management/architectural view of the SIP stack. It defines the methods required to represent and provision a proprietary SIP protocol stack.

This SipStack interface defines the methods that are used by an application implementing the *SipListener* interface to control the architecture and setup of the SIP stack. These methods include:

- Creation/deletion of *SipProvider*'s that represent messaging objects that can be used by an application to send Request and Response messages statelessly or statefully via Client and Server transactions.
- Creation/deletion of *ListeningPoint*'s that represent different ports and transports that a SipProvider can use to send and receive messages.

Architecture:
This specification mandates a single SipStack object per IP Address. There is a one-to-many relationship between a SipStack and a SipProvider. There is a one-to-many relationship between a SipStack and a ListeningPoint.
Using JAIN SIP

- Suggests the proper architecture and level of abstraction for applications.
- Vendor independent application development.
- Third party customization with minimal support.
  - Documentations already written
  - Examples of use are plentiful
The Goals

- The Java-standard interface to a SIP signaling stack.
  - Standardizes the interface to the stack.
    (Customizable components)
  - Standardizes message, transaction, and dialog interfaces.
  - Standardizes events and event semantics.
  - Application portability - verified via the TCK.

- Designed for developers who require powerful and complete access to the SIP protocol, though simple enough that your grandmother can use it to build the robust voicemail server she has always wanted.

- JAIN SIP can be utilized in a user agent, proxy, registrar or imbedded into a service container.
RFC Driven

- Embodiment of the necessary entities.
- A definition of the important events in SIP.
- An event model providing application access to those events.
- A “customizable” level of control over the entities and their life-cycles.
  - stateful vs. stateless handling
  - retransmissions
  - routing mechanism
  - authentication mechanism
Responsibilities of JAIN SIP

- Provide methods to format SIP messages.
- The ability for an application to send and receive SIP messages.
- Parse incoming messages and enable application access to fields via a standardized Java interface.
- Invoke appropriate application handlers when protocol significant
  - Message arrivals and 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.
Application Responsibilities

- Application registers an implementation of the SipListener interface to interact with the SIP Stack.

- Application must register with the SipProvider for all messaging capabilities with the stack.
  - Application requests transactions for stateful messaging.
  - Application sends stateless messages.
  - Access stack objects (transactions and dialogs)

- Application receives messages from the stack as Events via the SipListener interface.
Event Model

- The architecture is developed for the J2SE environment therefore is event based utilizing the Listener/Provider event model.
  - There is a direct reference between the event provider and event consumer
  - Event consumer must register with the event provider
- Events encapsulate incoming Requests and Responses.
- Event Model is one way i.e. Application doesn’t send out events, it sends out messages.
- The event model is asynchronous in nature using transactional identifiers to correlate messages.
- The SipListener represents the event consumer and listens for incoming Events that encapsulate messages that may be responses to initiated dialogs or new incoming dialogs.
- The SipProvider is the event provider who recieves messages from the network and passes them to the application as events.
The packages

- **General package (javax.sip.*)**
  - Defines the architectural interfaces, the transaction and dialog interfaces and the event objects of the specification.

- **Address package (javax.sip.address.*)**
  - Address package contains a generic URI wrapper and defines SIP URI and Tel URIs interfaces.

- **Message package (javax.sip.message.*)**
  - Defines the interfaces necessary for the Request and Response messages.

- **Header packages (javax.sip.header.*)**
  - Header package defines interfaces for all the supported headers and extension headers.
Initialize Stack using SipFactory:

```java
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);
    }
}
```
try {
    SipURI requestURI = addressFactory.createSipURI
        (toUser, toSipAddress);

    // ... Create other headers
    Request request = messageFactory.createRequest
        (requestURI, Request.INVITE, callIdHeader,
         cSeqHeader, fromHeader, toHeader,
         viaHeaders, maxForwards);
}

Initialize Request using Factories:
Send outgoing messages:

try {
    // Create the client transaction
    ClientTransaction inviteTid = sipProvider.getNewClientTransaction(request);
    // send the request
    inviteTid.sendRequest();
}

Handle incoming messages as Events:

```java
try {
    public void processRequest(RequestEvent requestEvent) {
        Request request =
            requestEvent.getRequest();
        ServerTransaction st =
            requestEvent.getServerTransaction();
        // do request specific processing here
    }
}
```
SIP Extensions described in internet drafts and RFCs typically define:

- New SIP Methods
  - New dialog creating methods
- New SIP Headers.

JAIN SIP defines an extensible framework to support new headers standardized for SIP:

- New SIP methods can be set using the string method field of a request.
- An application informs the stack of dialog creating methods, by specifying the method name to the EXTENSION_METHOD property of the SipStack configuration.

Interface support:

- Defines a ExtensionHeader interface that contains the header name and header value attribute pair.
- Can be created and accessed by name.
Latest News (straight from the source)

- Jain-SIP 1.2 is out
  - The Spec and the RI both live on the NIST web site. The TCK is under development. See http://jain-sip.dev.java.net for details on how to get the source.

- The main new features are (in summary):
  1. Better support for forked invite processing at the user agent.
  2. Better support for forked subscribe notify processing.
  3. Dialog free operation of the stack resulting in much better performance for Proxy Servers.
  4. Explicit support for Prack
  5. Multiple IP addresses per stack thus allowing better support for multi-homed hosts.
The Gory Details – and potential remedies

- (and why the myphonegarage.com doesn’t always succeed)
NATs, Firewalls and Media, Oh My!
Session Border Controllers (SBC)

Media Server to the rescue.

Brian O’Neill (bone@alumni.brown.edu)
J2EE Revisited

- Fundamentals:
  - J2EE was designed for the web, not to be the underlying framework for a SOA.

- It encourages:
  - synchronous development
  - disparate application processing
  - use of HTTP as a transport
Service Logic Execution Environment

- From the makers of J2EE... comes SLEE.

- SLEE is an Event-Driven Architecture for developing loosely coupled high-throughput asynchronous applications.

  - Java Community Process
    - JSR 22: SLEE v1.0
      - Approved: July 1999
      - Went final: March, 2004
    - JSR 240: SLEE v1.1
      - Currently in Early Draft Review
SLEE Architecture

<table>
<thead>
<tr>
<th>Communications</th>
<th>Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invocations</strong></td>
<td><strong>Typically synchronous</strong></td>
</tr>
<tr>
<td>- Typically asynchronous</td>
<td></td>
</tr>
<tr>
<td>- Events such as protocol triggers</td>
<td>- Database, EAI systems</td>
</tr>
<tr>
<td>- Event occurrences mapped to method invocations</td>
<td>- RPC calls</td>
</tr>
<tr>
<td><strong>Event Granularity</strong></td>
<td><strong>Coarse-grained events</strong></td>
</tr>
<tr>
<td>- Fine-grained events</td>
<td></td>
</tr>
<tr>
<td>- High frequency</td>
<td>- Low frequency</td>
</tr>
<tr>
<td><strong>Components</strong></td>
<td><strong>Heavy-weight data access objects</strong></td>
</tr>
<tr>
<td>- Light-weight fine-grained objects</td>
<td></td>
</tr>
<tr>
<td>- Short transient lifetimes</td>
<td>- Long persistent lifetimes</td>
</tr>
<tr>
<td>- Rapid creation, deletion</td>
<td></td>
</tr>
<tr>
<td><strong>Data Sources</strong></td>
<td><strong>Back-end systems</strong></td>
</tr>
<tr>
<td>- Multiple data sources</td>
<td></td>
</tr>
<tr>
<td>- Location, context information</td>
<td>- Database servers</td>
</tr>
<tr>
<td>- Provisioned data, cached from master copy</td>
<td>- Definitive master copy</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td><strong>Database transactions</strong></td>
</tr>
<tr>
<td>- Light-weight transactions</td>
<td></td>
</tr>
<tr>
<td>- For state replication demarcation</td>
<td>- Slower completion and less frequent</td>
</tr>
<tr>
<td>- Faster completion and more frequent</td>
<td></td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td><strong>Database access intensive</strong></td>
</tr>
<tr>
<td>- Compute-intensive</td>
<td></td>
</tr>
<tr>
<td>- Main input and output are resource invocations, messages, events</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: High-level requirement comparison of event driven and enterprise systems.
A SLEE delivers Activity Objects (a.k.a. Events) to the SBB's as determined by a run-time configurable graph.

The edges of the graph determine priority of delivery.
Advantages

- Event producers are not aware of their consumers (true decoupling)
- A single consistent event distribution model (e.g. the order of event delivery to multiple event consumers, concurrency control semantics)
- Consolidated Management
- Shared Persistence
- Share Event Processing Information via Activity Contexts, by setting attributes (similar to what a shared session would be in J2EE)
- Timers, Tracing, Usages and Alarms can all be built into the infrastructure.
Dynamic Service-Oriented Architectures

Plan → Wait for Availability → Establish Connectivity → Execute Scenario

From months to minutes.
Still are challenges... one of which is Security.
Test-Driven Development and SIP

- http://www.cafesip.org/
Thank You

Shout-outs:
Phelim O'Dougherty, BEA  (formerly @ Sun)
M. Ranganathan, NIST

JSR 32
http://jcp.org/en/jsr/detail?id=32

Subscribe to:
http://archives.java.sun.com/jain-sip-interest.html
API DETAILS

- API DETAILS
JAIN SIP supports the SIP protocol functionality described in RFC 3261.

JAIN SIP supports the following SIP extensions:

- RFC 2976 allows for the carrying of session related control information that is generated during a session.
- RFC 3262 provide information on progress of the request processing.
- RFC 3265 the ability to request asynchronous notification of events.
- RFC 3311 allows the caller or callee to provide updated session information before a final response.
- RFC 3326 the ability to know why a SIP request was issued.
- RFC 3428 allows the transfer of Instant Messages.
- RFC 3515 requests that the recipient refer to a resource provided in the request.
SipStack Interface

- Created via the SipFactory.
- Initialized with javax.sip.* properties.
  - Retransmissions are optionally filtered.
  - Router functionality can optionally be assumed by the application.
  - Dialog-creating methods can be set as "extension methods"
  - Outbound proxy server can be set.
- Used to create the SipProvider
- Manages listening points and providers.
SipStack Properties

- **IP_ADDRESS**
  - Sets the IP Address of the SipStack. This property is mandatory.

- **STACK_NAME**
  - Sets a user friendly name to identify the underlying stack implementation. This property is mandatory.

- **OUTBOUND_PROXY**
  - Sets the outbound proxy of the SIP Stack.

- **ROUTER_PATH**
  - Sets the fully qualified classpath to the application supplied Router object that determines how to route messages before a dialog is established.

- **EXTENSION_METHODS**
  - This configuration value informs the underlying implementation of supported extension methods that create new dialog's.

- **RETRANSMISSION_FILTER**
  - A helper function for User Agents that enables the stack to handle retransmission of ACK Requests, 1XX and 2XX Responses to INVITE transactions for the application.
SipListener Interface

- Application implements this interface.
- This is the application's interface to events generated by the SIP stack.
- One per SipStack
- All SipProviders deliver events to the same listener.
- Receives requests from the provider to be handled either statefully or statelessly dependent on application logic.
- Receives responses from the provider.
- Receives timeout and error events from the provider.
SipProvider Interface

- Application registers its SipListener to the SipProvider.
- The application uses the provider to create Client and Server Transaction creation methods.
  - (stateful transmission and reception)
- Additionally, it can use the stack to send requests and responses statelessly (without transactions).
- Listening Point manipulation methods.
- One ListeningPoint per SipProvider

- Simply put, a provider monitors a “port”, delivers events occurring on that port to the application, and transmits messages.
Factories

- **SipFactory**
  - This interface defines methods to create new Stack objects and other factory objects.

- **AddressFactory**
  - This interface defines methods to create SipURI’s and TelURL’s.

- **HeaderFactory**
  - This interface defines methods to create new Headers objects.

- **MessageFactory**
  - This interface defines methods to create new Request and Response objects.
Headers and Addresses

- Headers and Address in the JAIN SIP API, are simple object representations of the entities described in the RFC. Which gives the application compile time checking, and logic-level sanitization for their application.

- JAIN SIP models each SIP header as a specific interface as opposed to having a single generic interface to handle all header information.
  - Each interface specifies the Headers acceptable parameters.
  - More explicit protocol support – parsing support for each header.

- JAIN SIP models each SIP address as a specific interface encapsulating the fields available in those respective entities.
There are two types of messages in SIP, which JAIN SIP defines as Interfaces:

- Request messages are sent from the client to server.
  - They contain a specific method type that identifies the type of Request.
  - A Request-URI which indicates the user or service to which this request is being addressed.

- Response messages are sent from server to client in response to a Request.
  - They contain a specific status code that identifies the type of Response.
  - A Request-URI which indicates the user or service to which this request is being addressed.
  - A reason phrase that is intended for the human user.

Messages may contain multiple Headers of the same type.

- The order of Headers of a given type within a message is significant.

A Message Body may contain a session description.

- JAIN SIP defines this format an Object which allows the body to be a String or an Object type defined the Session Description Protocol (SDP) JSR specification and also a byte array.
Transaction Support

- JAIN SIP standardizes the interface to the generic transactional model defined by the SIP protocol
  - JAIN SIP models both Client and Server Transactions as Interfaces.
- Transaction is created on incoming Request or may be created to send outgoing request.
  - When a Request is sent out statefully, application must request a ClientTransaction
  - When a new Request arrives, application determines whether to handle request via a ServerTransaction
  - When a Request in an existing dialog arrives the stack automatically associates it to a ServerTransaction
- When a response arrives, the Stack possibly associates a previously created ClientTransaction with the response
- Messages are passed to the SipProvider in order to generate a new transaction. This transaction can be used to send the message onto the network
- Implementation manages the association between Transactions and Dialogs.
Dialog Support

- A Dialog is a peer to peer association between communicating SIP endpoints.
  - The dialog represents a context in which to interpret SIP messages.
- Dialogs are never directly created by the Application.
  - Dialogs are established by Dialog creating Transactions (INVITE, SUBSCRIBE…) and are managed by the stack.
- Dialog deletion may be under application control.
  - Though not generally recommended.
- Dialogs are used to maintain data needed for further message transmissions within the dialog
  - Route Sets, Sequence Numbers, URI’s of the parties in the dialog.
- Dialogs have a state machine
  - Early, Confirmed, Completed and Terminated.
- Transactions may belong to a Dialog
  - Dialog state changes as a result of changes in Transaction State.
  - Access to dialog functionality from the transaction interface.
### The Upgrade

<table>
<thead>
<tr>
<th>JAIN SIP v1.0</th>
<th>JAIN SIP v1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>- RFC2543 Supported.</td>
<td>- RFC3261 Supported.</td>
</tr>
<tr>
<td>- J2SE 1.3 and above.</td>
<td>- J2SE 1.4 and above.</td>
</tr>
<tr>
<td>- Transactions referenced by long.</td>
<td>- Transaction interfaces defined.</td>
</tr>
<tr>
<td>- Transaction state is not visible to application.</td>
<td>- Transaction/Dialog state can be read by application.</td>
</tr>
<tr>
<td>- No explicit Dialog Support.</td>
<td>- Dialog interface defined and managed by stack.</td>
</tr>
<tr>
<td>- Stack Configuration not defined.</td>
<td>- Stack Configured with defined properties.</td>
</tr>
</tbody>
</table>
Profiles:
“The SLEE specification defines a generic provisioned data schema that is easy to use to define, provision, and access profiles. It includes interfaces for adding, removing, and modifying provisioned data. Typical provisioned data includes configuration data, such as per-subscriber data.”
- JAIN SLEE Principles, Phelim O'Dougherty

Management:
“To effectively manage application usage, services, event flow and resources it is necessary to monitor and measure the present performance, utilization and availability of these attributes, while estimating future such characteristics. ”
JAIN SLEE Principles, Phelim O'Dougherty