Voice-over-IP applications using open standards and open source software

Eric Cheung & Tom Smith
AT&T Labs - Research
Outline

- Background for SIP application development
- SIP Servlet containers
- E4SS : state-machine programming for converged applications
- KitCAT : Testing framework for converged applications
- SIP application composition
- Summary
Background for SIP application development
Voice over IP protocols are complex...

In this scenario, Alice gives up on the call before Bob answers (sends a 200 OK response). Alice sends a CANCEL (F3) since response had been received from Bob. If a 200 OK to the F3 crossed with the CANCEL, Alice would have sent an ACK then Bob in order to properly terminate the call.

Note that the CANCEL message is acknowledged with a 200 OK by hop basis, rather than end to end.

This scenario illustrates the race condition which occurs when the UA receives an Early message, CANCEL, while in the Moratorium. Alice sends a CANCEL and Bob sends a 200 OK response to the initial INVITE message at the same time. As described in the previous section, according to RFC 3261 [3], an INVITE server transaction is not supposed to be terminated by a 200 response, but this has been corrected in [7].

This section describes a case in which an INVITE server transaction is not terminated by a 200 response to the INVITE request. In
Recipe for growth of applications

**Software tools**
Frameworks, libraries, test tools

**Developer Community**
Code examples, tutorials, training, support, availability of developers

**Open standards**
Portability
Avoid being locked into a technology

**Wide choice of execution environments**
Choose best of breed (performance, operations, etc)

**Open Source**

**Higher level API’s**
Familiar programming model, hide complexity, lower learning curve
# Standards for VoIP applications

Many standards to address different needs:

<table>
<thead>
<tr>
<th><strong>Call Processing Language</strong></th>
<th><strong>Parlay/OSA, Parlay X</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- XML-based language intended for programming by end-users.</td>
<td></td>
</tr>
<tr>
<td>- For proxies and other network servers</td>
<td></td>
</tr>
<tr>
<td><strong>SIP Servlet</strong></td>
<td></td>
</tr>
<tr>
<td>- Similar to HTTP servlet</td>
<td></td>
</tr>
<tr>
<td>- Specific to SIP, requires intimate knowledge of SIP</td>
<td></td>
</tr>
<tr>
<td>- Integration with Java Enterprise Edition, built-in HTTP convergence</td>
<td></td>
</tr>
<tr>
<td><strong>JAIN SLEE</strong></td>
<td></td>
</tr>
<tr>
<td>- High level API, protocol independent</td>
<td></td>
</tr>
<tr>
<td>- Component model, event driven</td>
<td></td>
</tr>
<tr>
<td>- Service Building Blocks handle events</td>
<td></td>
</tr>
<tr>
<td>- Resource Adaptors interface with network</td>
<td></td>
</tr>
</tbody>
</table>
Why we chose SIP servlet

• Our group is working in a SIP-only environment
  – e.g. applications intended to run on native SIP application servers in the 3GPP IMS architecture
• With the SIP servlet API, applications have low-level control, down to message header level
• SIP servlet supports pipe-and-filter architecture for application composition
  – similar to the Distributed Feature Composition (DFC) architecture and IMS
• Stateless model lends itself to scalability and high availability
• Low-level API allows higher level frameworks to evolve
SIP servlet API evolution

- **2003**: Version 1.0 (Java Community Process JSR 116)
- **By 2005**, about 5 commercial container implementations
- Shortcomings identified, community demands update
- **Jan 2006**: New JSR 289 formed to work on next version
  - 25 companies in Expert Group, up from 8 in JSR 116
- **Aug 2008**: Version 1.1 Final Release

Main enhancements include:
- Standardized application selection and composition mechanism
- Enhanced integration with Java Enterprise Edition (Java EE)
- Seamless support for converged HTTP and SIP servlet applications
Roles in SIP servlet

- Deployer
- Application Developers
- SIP Servlet Container Vendor
Components in SIP servlet

**Application Router**
- Queried by container to select an application to handle an initial request
- No application logic, cannot modify or send SIP messages

**Applications**
- Provide call control or other application logic
- Each application acts as a SIP entity (proxy, B2BUA, UA, etc)
- Should be independent of other applications

**SIP Servlet Container**
- Send/receive SIP messages, handles low-level concerns such as retransmission
  - Maintains lifetime and states of AR and apps
  - Queries AR and dispatches messages to apps
SIP Servlet Containers
Availability of open source containers
SIP servlet containers

Commercial products
- Avaya, IBM, MicroMethod⁺, Oracle⁺, Voztelecom

Open source implementations
- Cipango
- **Mobicents⁺** (Red Hat)
  - On top of JBoss or tomcat
  - Supports clustering for high availability
- **SailFin⁺** (Ericsson/Sun)
  - On top of GlassFish
  - Supports clustering for high availability

⁺ Claimed JSR 289 compliance
Experience with SailFin

• Open source project started in May 2007
  – SIP servlet container contributed by Ericsson
  – GlassFish Java EE container from Sun Microsystems
• Active ongoing development (beta phase)
  – Version 1.0 alpha released in Aug 2008
  – Version 1.0 final release scheduled for end of 2008
  – Currently not yet certified for JSR 289, probably not ready for real deployments
• Fairly active and wide developer and user community
  – More than 80 posts/month on mailing lists
  – Developers very responsive to questions and bug reports
• Commercial distribution will be available
  – Sun announced ‘Sun GlassFish Communications Server’ based on SailFin
E4SS : a state-machine programming model for converged applications
Introduction to E4SS

- **ECharts for SIP Servlets** – open-source, state-machine-based framework for programming converged services
- **Changes programming abstraction** from servlet callbacks to state machines
- Based on ECharts state-machine programming language
  - Inspired by UML Statecharts, with a number of implementation-oriented extensions
  - Allows embedded, concurrent, dynamic machines
  - Supports parameterized state machines, great for reuse
E4SS Programming Model

- **Application programmer specifies state-machine logic**, coded in ECharts.
- The toolkit provides a master SIP servlet, which creates the specified FSM and dispatches received messages to appropriate ECharts ports.
- These messages drive FSM logic.
- Arbitrary Java code can be embedded in FSM, which allows **full use of SIP Servlet API**.
- Supports UAs, proxies, and back-to-back user agents (B2BUAs).
- Particularly well-suited for state-intensive nature of B2BUA applications.
Benefits of E4SS

- **Centralization of application logic**
- **Reusable feature framework** (see Greg Bond’s talk)
- ECharts port abstraction
- Embedded state machines
  - State machine re-use
  - Overriding default behavior
- First-class support for non-SIP events
  - Timed transitions
  - Convergence framework (e.g., HTTP)
- SIP message class hierarchy
- Automatic call termination handling
- Support for application composition
Developer support

- **appgen** - tool to create application skeleton
- **monitor** - SIP messages + ECharts events summary
- **echdoc** - graphical machine representation with interactive browsing
E4SS status

- Release 2.4
- Pre-release 2.5 has JSR289 support
- Used for dozens of applications --- industrial-scale teleconferencing service development underway
- Roadmap includes support for clustered environments
KitCAT – a Framework for Converged Application Testing
Introduction to KitCAT

- **Kit for Converged Application Testing** – Java-based testing framework supporting SIP, RTP, and HTTP
- Allows creation of JUnit-style test cases for converged applications
- **High-level call control primitives** to control test agents (e.g., call, answer, end), with options for low-level customization
- Ability to **send and receive RTP streams**
- Ability to **send DTMF** key sequences
- Uses HtmlUnit for **web testing**
- **Flexible and extensible assertion primitives** to test conditions on agent states and messages sent/received by agents.
Example code: call forwarding test case

```java
SipAgent alice = createAgent("alice");
SipAgent bob   = createAgent("bob");
SipAgent chuck = createAgent("chuck");
setCallForwardingViaWeb(bob, chuck);
alice.setProxy(sut);

alice.call(bob);
processSIP(2000);
assertThat(chuck, has(recvdRequest("INVITE")));
```
Example code: call forwarding test case

chuck.answer();
processSIP(2000);
assertThat(alice, is(connectedTo(chuck))); // SIP

ertestMediaFlow(alice, chuck);  // RTP

alice.end();

processSIP(2000);
assertThat(chuck, is(disconnected()));

assertThat(bob, is(idle()));

assertThat(callLogPage, shows(callFrom(alice)));
Sample KitCAT report

### Unit Test Results

Designed for use with JUnit and Ant.

Class `com.att.cantata.features.sequentialLocation.test.SequentialLocationTest`

<table>
<thead>
<tr>
<th>Name</th>
<th>Tests</th>
<th>Errors</th>
<th>Failures</th>
<th>Time(s)</th>
<th>Time Stamp</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>SequentialLocationTest</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>56.500</td>
<td>2008-10-16T14:20:54</td>
<td>moab.research.att.com</td>
</tr>
</tbody>
</table>

### Tests

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>testNoLocations</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>testAnswer</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>testAllLocationsFail</td>
<td>Failure</td>
<td>Expected: is not agent subscriber is in one of these states - Idle curr state = Idle &gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>junit.framework.AssertionFailedError:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expected: is not agent subscriber is in one of these states - Idle curr state = Idle got:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>==Current state = Idle &gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expected: is not agent subscriber is in one of these states - Idle curr state = Idle got:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>==Current state = Idle &gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>at com.att.cantata.features.sequentialLocation.test.SequentialLocationTest.testAllLocationsFail</td>
</tr>
<tr>
<td>testCancel</td>
<td>Success</td>
<td></td>
</tr>
</tbody>
</table>
KitCAT status

- Initial release expected 4Q 2008
- Will be available at echarts.org
- Create an account at echarts.org to receive release notifications.
Application Composition

open source DFC Application Router
DFC Application Router

- Simple Application Router implementation for illustration
- But still full-featured and powerful for real deployments
- An *address* subscribes to applications
  - Caller = From header
  - Callee = Request-URI
- Applications have precedence relationship
  - Governs ordering to manage feature interactions
- Static XML file for configuration
DFC AR Configuration - Subscription

```xml
<application-mapping>
  <terminating-region-mapping>
    <mapping>
      <address-pattern>sip:bob@.*</address-pattern>
      <app-name>/rerouteUponFailureTest</app-name>
      <app-name>/ccfTest</app-name>
      <app-name>/noAnswerTimeoutTest</app-name>
    </mapping>
  </terminating-region-mapping>
</application-mapping>
```

Call to Bob
DFC AR Configuration - Precedence

<precedence>
  <terminating-region>
    <ordering>
      <app-name>/noAnswerTimeoutTest</app-name>
      <app-name>/ccfTest</app-name>
      <app-name>/rerouteUponFailureTest</app-name>
    </ordering>
  </terminating-region>
</precedence>

Call to Bob

RR → CCF → NAT → Bob
Summary

- Open-source **SIP Servlet containers** provide a standard platform for development of converged VoIP applications.
- **E4SS** provides developers with a state-machine-based programming model with excellent possibilities for re-use.
- Modular applications can be assembled using the powerful and flexible **DFC Application Router**.
- **KitCAT** allows for automated functional testing of converged applications or service compositions.
For more information

- Visit:  
  - [http://echarts.org/](http://echarts.org/) for software, documentation, blog entries and more on E4SS, KitCAT, and DFC Application Router  
  - [http://sailfin.dev.java.net/](http://sailfin.dev.java.net/) for Sailfin  