

The Mobicents Open Source SLEE Platform

http://www.mobicents.org

VON Fall 2005

- Ivelin Ivanov, JBoss, Inc.
- M. Ranganathan, NIST
- Francesco Moggia, IRIS Srl
- Phelim O'Doherty, BEA Systems, Inc.

What are we going to talk about?

- Mobicents Background
- JSLEE in the industry
- Mobicents Architecture
- Demo
- Roadmap

Mobicents Background

Ivelin Ivanov



In the beginning...

- First CVS commit in fall 2003
- Initial Goals
 - IM Portal bridging ICQ, MSN, Y!, Jabber
 - Message transformation, routing, broadcasting, command execution
- Challenges
 - Independent OSS projects for each IM protocol
 - Integration headache
 - Lack of unifying platform

Priorities revisited

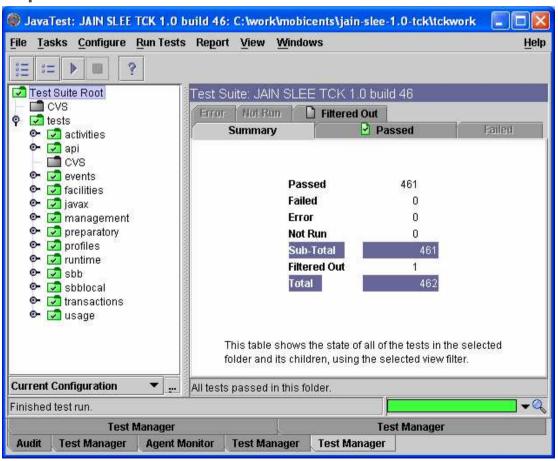
- Need for a solid foundation
- Standard way to add building blocks
- Ability to extend with VoIP services
- First option J2EE + JAIN SIP + Jabber + Y!...
 - Easy to demo, hard to extend, hopeless performance outlook
- Next option JSLEE
 - Looks like a duck…
 - Tough to comprehend, hard to implement
 - We'll byte the bullet anyway

1.0a milestone

- JAIN SLEE 1.0 Certification
- Rigorous and thorough spec

helps a lot

- Great TCK coverage
- Leverage JBoss 3.2.x

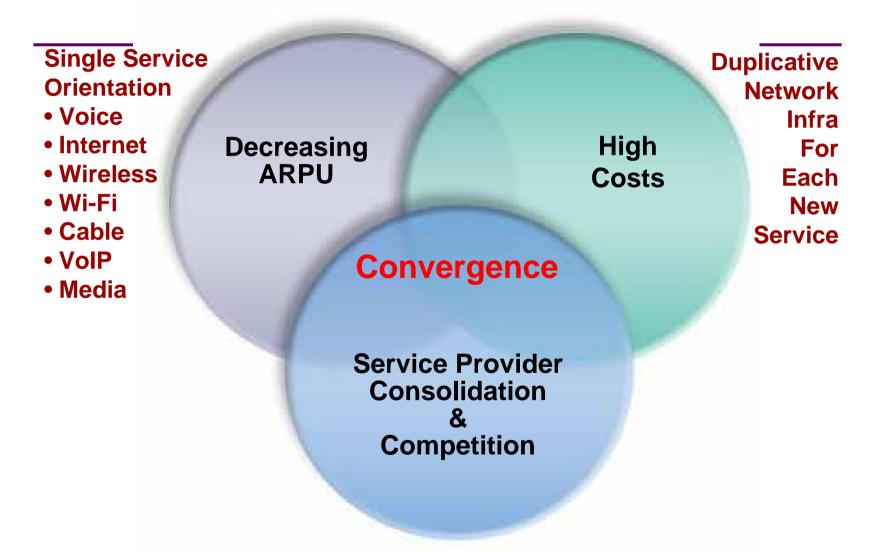


JAIN SLEE Industry Realization and Trends

Phelim O'Doherty

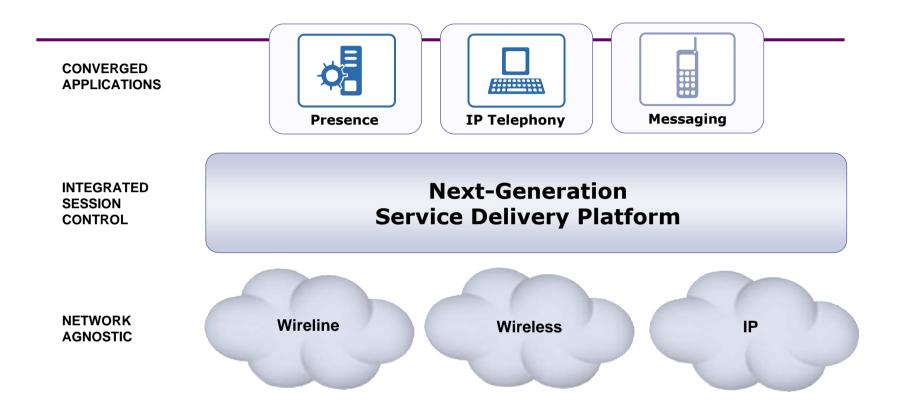


Telecom Industry Challenges



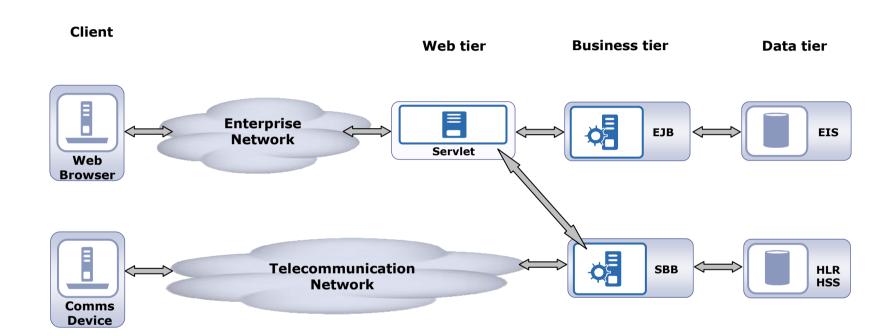
Consolidation of Networks, Bandwidth, Subscriber Base

Next-Generation Service Delivery Platform



- Evolution to an all-IP network
- Well-defined, standardized interfaces through 3GPP, OMA and IETF
- Based on SIP as standard communication protocol
- Common abstracted network service interfaces.

J2EE and JSLEE



JSLEE and Parlay

- Parlay defines an architecture and set of abstracted interfaces
 - No application server defined
- JSLEE is a standard event driven application server
 - Parlay App servers are typically built on proprietary SLEE's instead of J2EE in order to achieve performance
 - Vendors should build Parlay application servers on a standard JSLEE, enabling portability of services

JSLEE and SIP Servlet

- SIP Servlet satisfies the 'big bang' approach to Voip
 - All application development in SIP
 - Tightly integrated with web-tier
- JSLEE satisfies the 'migration' approach to Voip
 - Leverage existing IN investment
 - Single environment for IN evolution and IMS adoption
 - Enables multiple protocol development
- Both servers will co-exist each finding certain application niches
- Each will make the other successful
 - Same goal replace proprietary systems with standard application infrastructure

Benefits

- Traditional of-the-shelf software
 - Leverage IDE's for Service creation
- Co-location of services
 - Reduce latency and resource consumption
- Open programming environment
 - Tools can be leveraged by developers
 - Leverage the best of the enterprise patterns
 - Event driven design base
 - Profile interface to subscriber/service data

Barriers

- No large software provider
- NEP's resisting adoption
 - Spoils business model
- New technology
 - Will developers pick up programming model
 - Requires investment
- Bad press
 - ISV's competing for business trashing technologies
 - Competing architectures from different standards bodies

What changing in the industry?

- Operators reluctant to upgrade capacity from older black box solutions to new black-box solutions
 - NEP's are losing business they typically secured
- NEP's have in-house development of technology
 - Near tipping point
- Key operators are still applying pressure
 - Willingness to invest in standard based middleware
- Open source environment gaining traction
 - Communities speak volumes

Keep it real

July 5, 2005

Vodafone Spain Proves Software Portability for Programmable Network Services

MADRID, Spain (July 05, 2005)-While the telecommunications industry readies for the deployment of truly programmable network services that provides service portability in both IN and 3G/IMS, Vodafone Spain has already successfully used JAIN SLEE (Java) technology to deploy and run a real life commercial IN service on two different vendors platforms.

■ 17 March, 2005

Rogelio Martinez, head of the Centre of Competence for JSLEE technologies of Vodafone Spain, said:

"Ensuring the success of our MMS network is a business-critical priority for us. The NetSpira ECS-MMR provides a way to eliminate bottlenecks in our MMS network. NetSpira has proved once again that it is ahead of the game - anticipating the needs of mobile and cellular operators and developing must-have products." The trial used several value-added services running on the top of Open Cloud Rhino's 1.2, a JAIN™ SLEE standards compliant service logic execution environment for carrier grade implementations.

Mobicents Examples Architecture and Implementation

M. Ranganathan and Francesco Moggia

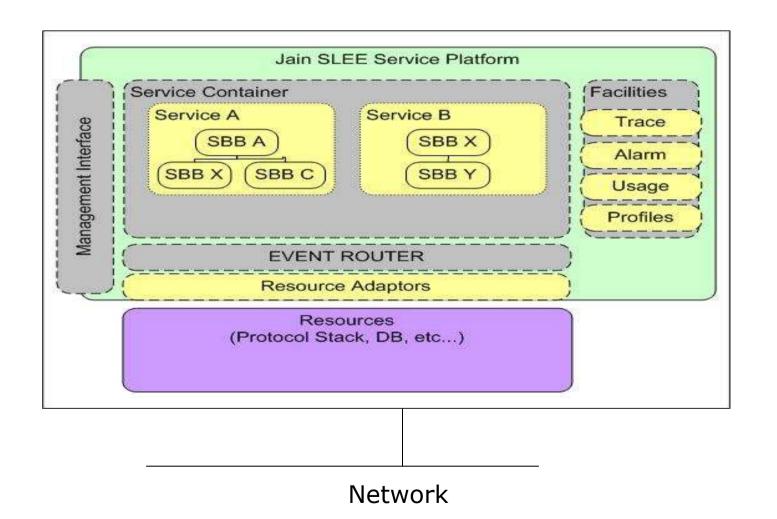




What is JSLEE - A more detailed answer

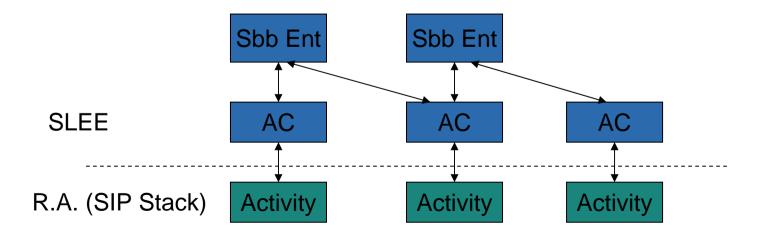
- JAIN SLEE is an event oriented application middleware. Its main job is to field events from external resources and deliver these events to application installed fragments of code.
- Is multi-protocol. Each protocol stack is a Resource with a corresponding Resource adaptor.
- Has the flavor of EJB but is oriented towards asynchronous low latency event oriented apps.
- Manages application objects, lifecycle, isolation, deployment, management, concurrency, persistence and failover needs.
- Provides the necessary facilities commonly needed by such applications (i.e. timers, management via JMX, profile, trace).

Simplified JAIN-SLEE Architecture



Some Details

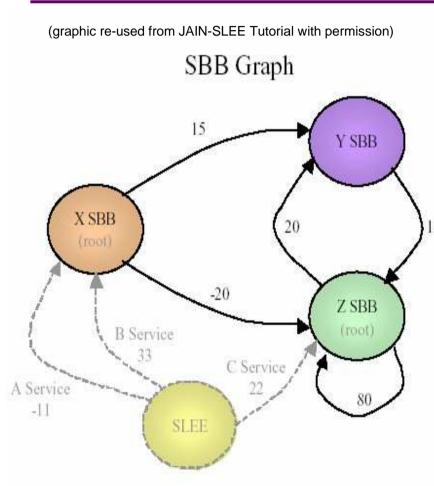
- The SLEE abstracts the notion of an Event bus and event triggered pieces of code (Called SBBs).
- The event bus is called an Activity Context (AC).
 - An Activity is a stream of related events.
 - One-to-one mapping between Activities (Resource Adaptor domain) and Activity Contexts (SLEE domain).
 - Activity Context is an Event Channel.



SLEE Building Blocks

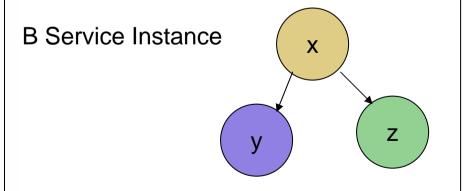
- Event Type
 - Typed event model.
 - Resource adaptors generate events
- SBB
 - Fundamental building block (like an EJB).
 - SBBs can communicate by firing events on ACs
 - SBBs receive events over AC's.
 - Event source can be Resource Adaptors, Facilities, Timers, Alarms...
- Service
 - Management Artifact
 - Contains information for initial event processing
 - Is instantiated by the SLEE upon receipt of appropriate message matching the Convergence Name of the Service.
- Profile Specification
 - Provisioned data for management of services.
- Usage parameters interfaces
- JMX management clients

Service instantiation



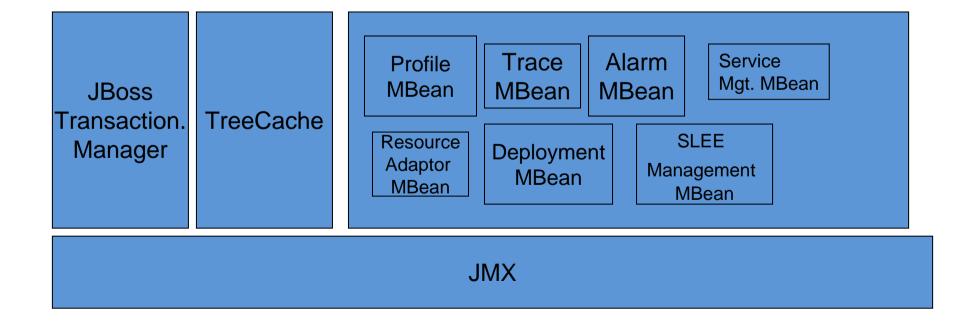
These are the SBBs known to the SLEE
This picture is known at Deployment time (through the deployment descriptors).

At run time, the service Instantiates a traversal of a Sub-graph of this sbb graph.



SLEE (JSR 22) as a JBoss Service

JBoss Microkernel architecture is a natural fit for building the SLEE

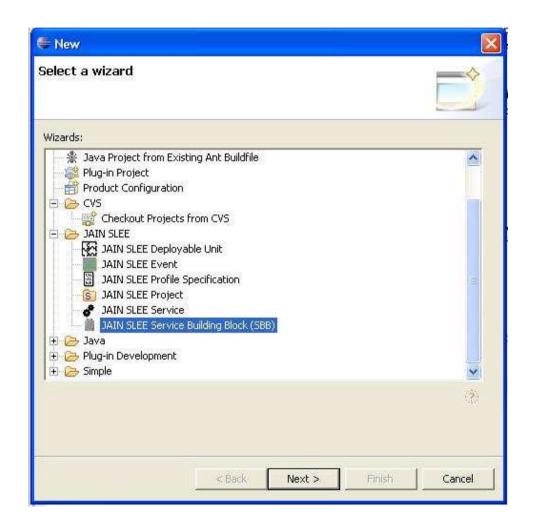


jboss-service.xml for Mobicents

```
<mbean code="org.mobicents.slee.container.management.jmx.SleeManagementMBeanImpl"
name="slee:service=SleeManagement">
    <depends optional-attribute-name="AlarmMBean">slee:name=AlarmMBean/depends>
    <depends optional-attribute-name="DeploymentMBean">
          slee:name=DeploymentMBean</depends>
    <depends optional-attribute-name=
          "ServiceManagementMBean">slee:name=ServiceManagementMBean</depends>
    <depends optional-attribute-name="TraceMBean">slee:name=TraceMBean/depends>
    <depends optional-attribute-name=</pre>
          "ProfileProvisioningMBean">slee:name=ProfileProvisioningMBean</depends>
    <depends optional-attribute-name=
          "ResourceAdaptorMBean">slee:name=ResourceAdaptorMBean</depends>
    <depends>slee:name=TransactionManagerMBean</depends>
    <depends>jboss.cache:service=TreeCache</depends>
    <depends>jboss.cache:service=DeploymentTreeCache</depends>
    <depends>jboss.cache:service=ProfileTreeCache</depends>
    <depends>slee:service=SleeProfileManager</depends>
    <depends>jboss.cache:service=RuntimeTreeCache</depends>
 </mbean>
```

EclipSLEE plug-in project

- EclipSLEE project on java.net
- Subproject of Mobicents
- OpenCloud Contribution
- The plug-in provides a user friendly service creation environment for JAIN SLEE.



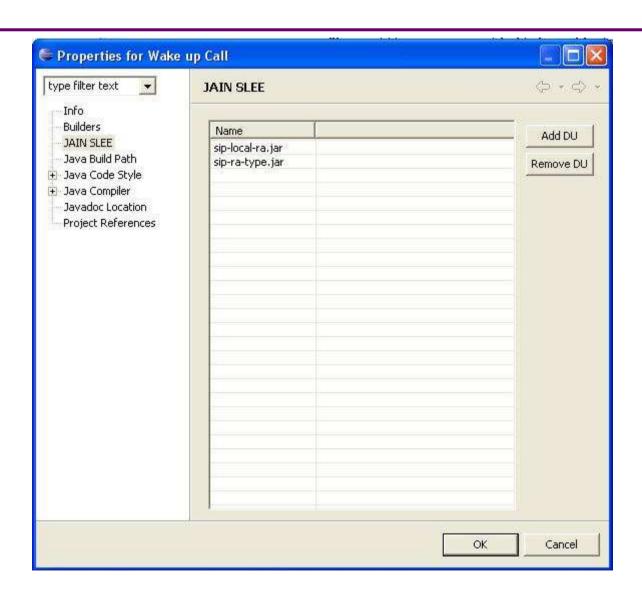
Minimal SBB xml & code

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sbb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE SBB 1.0//EN"</pre>
                        "http://java.sun.com/dtd/slee-sbb-jar 1 0.dtd">
<sbb-jar>
    <sbb>
        <description>This Sbb send a wake up call to the user.</description>
        (sbb-name>Wake Up Sbb</sbb-name>
        sbb-vendor>NIST</sbb-vendor>
        sbb-version>1.0</sbb-version>
        <sbb-classes>
           <sbb-abstract-class>
               <shh-abstract-class-nam</pre>
                   org.mobicents.slee.examples.wakeup.WakeUpSbb
                                               package org.mobicents.slee.examples.wakeup;
           </sbb-abstract-class>
        </shb-classes>
                                               import javax.slee.*;
    </shb>
</sbb-jar>
                                               public abstract class WakeUpSbb implements javax.slee.Sbb {
                                                   // TODO: Perform further operations if required in these methods.
                                                   public void setSbbContext(SbbContext context) { this.sbbContext = context; }
                                                   public void unsetSbbContext() { this.sbbContext = null; }
                                                   // TODO: Implement the lifecycle methods if required
                                                   public void sbbCreate() throws javax.slee.CreateException {}
                                                   public void sbbPostCreate() throws javax.slee.CreateException ()
                                                   public void sbbActivate() {}
                                                   public void sbbPassivate() {}
                                                   public void sbbRemove() {}
                                                   public void sbbLoad() {}
                                                   public void sbbStore() {}
                                                   public void sbbExceptionThrown(Exception exception,
                                                           Object event, ActivityContextInterface activity) {}
                                                   public void sbbRolledBack(RolledBackContext context) ()
                                                   protected SbbContext getSbbContext() {
                                                       return sbbContext;
                                                   private SbbContext sbbContext; // This SBB's SbbContext
```

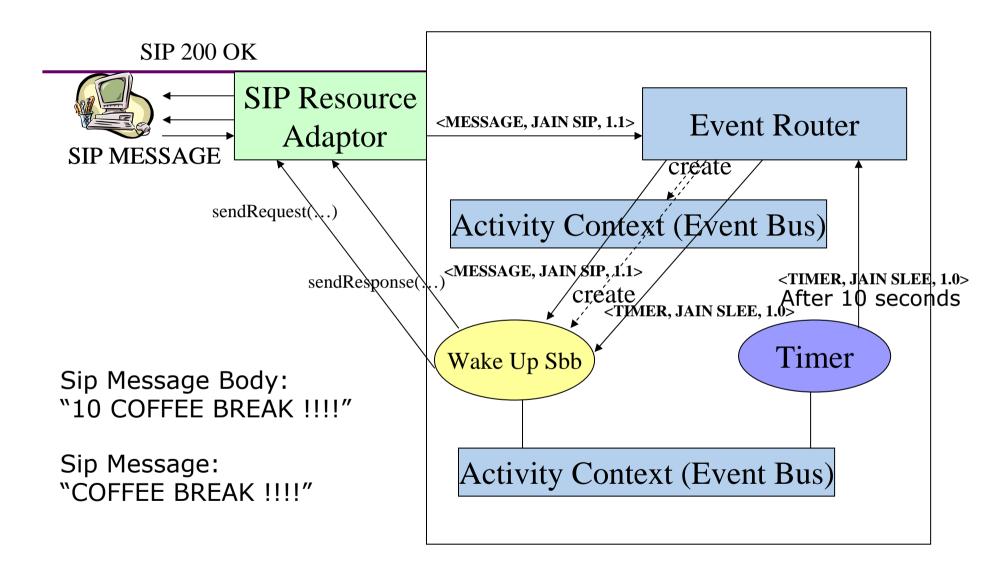
Service descriptor

```
<?xml version="1.0" encoding="utf-8"?>
K!DOCTYPE service-xml PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE Service 1.0//EN"
                             "http://java.sun.com/dtd/slee-service 1 0.dtd">
<service-xml>
    <service>
        <description>Wake up service</description>
        <service-name>Wake Up Service</service-name>
        <service-vendor>NIST</service-vendor>
        <service-version>1.0</service-version>
        <root-sbb>
            <description>This Sbb send a wake up call to the user.</description>
            <sbb-name>Wake Up Sbb</sbb-name>
            <sbb-vendor>NIST</sbb-vendor>
            <sbb-version>1.0</sbb-version>
        </re>
        <default-priority>0</default-priority>
    </service>
</service-xml>
```

Resource Adaptor inside the Plug-in



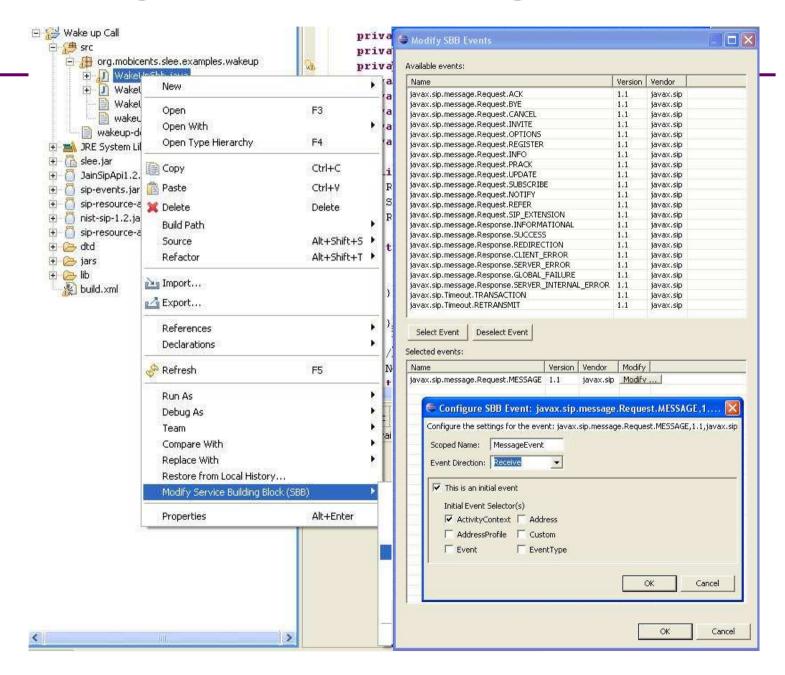
Wake Up Service



Resource adaptor & SBB

```
<re>ource-adaptor-type-binding>
    <resource-adaptor-type-ref>
          esource-adaptor-type-name>jain-sip</resource-adaptor-type-name>
          esource-adaptor-type-vendor>javax.sip</resource-adaptor-type-vendor>
          esource-adaptor-type-version>1.1</resource-adaptor-type-version>
    </resource-adaptor-type-ref>
    <activity-context-interface-factory-name>
        slee/resources/jainsip/1.1/acifactory
    <resource-adaptor-entity-binding>
                                                        Initialize the comonent
            slee/resources/jainsip/1.1/provider
                                                      ublic void setSbbContext(SbbContext context) {
        //resource-adaptor-object-name>
                                                         this.sbbContext = context;
        <re>ource-adaptor-entity-link></re>
                                                         try {
                 SipRA
                                                             Context myEnv = (Context) new InitialContext().lookup("java:comp/env");
        </resource-adaptor-entity-link>
                                                             // Storing Sbb Component ID
                                                             id = sbbContext.getSbb();
    </resource-adaptor-entity-binding>
</resource-adaptor-type-binding>
                                                             // Getting SLEE Factility
                                                             traceFacility = (TraceFacility) mvEnv.lookup("slee/facilities/trace");
                                                             timerFacility = (TimerFacility) myEnv.lookup("slee/facilities/timer");
                                                             alarmFacility = (AlarmFacility) myEnv.lookup("slee/facilities/alarm");
                                                             null&CIFactory = (Null&ctivityContextInterfaceFactory)myEnv.
                                                                            lookup("slee/nullactivity/activitycontextinterfacefactory");
                                                             nullActivityFactory = (NullActivityFactory)myEnv.lookup("slee/nullactivity/factory");
                                                             // Getting JAIN SIP Resource Adaptor interfaces
                                                             fp = (SipFactoryProvider) myEnv.loo.tup("slee/resources/jainsip/1.1/provider");
                                                             provider = fp.getSipProvider();
                                                             addressFactory = fp.getAddressFactory();
                                                             headerFactory = fp.getHeaderFactory();
                                                            messageFactory = fp.getMessageFactory();
                                                             acif = (SipActivityContextInterfaceFactory) myEnv
                                                                    lookur ("slee/resources/jainsip/1.1/acifactory");
                                                         } catch (NamingException ne) {
                                                             this.trace(Level.WARNING, "Exception During setSbbContext", ne);
```

Adding Events with the Plug-in



Event handlers

```
<event event-direction="Peceive" initial-event="True">
         <event-name>MessageEvent</event-name>
         <event-type-ref>
             <event-type-name>javax.sip.message.Request.MESSAGE</event-type-name>
             <event-type-vendor>javax.sip</event-type-vendor>
             <event-type-version>1.1</event-type-version>
         </event-type-ref>
        <initial-event-select variable="ActivityContext"/>
     </event>
     <event event-direction="Receive" initial-event="False">
         <event-name>TimerEvent</event-name>
         <event-type-rei>
             <event-type-name>javax.slee.facilities.TimerEvent</event-type-name>
             <event-type-vendor>javax.slee</event-type-vendor>
             <event-type-version>1.0</event-type-version>
         </event-type-ref>
     </event>
public abstract class WakeUpSbb implements javax.slee.Sbb {
   public void of MessageEvent javax.sip.RequestEvent event, ActivityContextInterface aci) {
   public void of TimerEvent TimerEvent event, ActivityContextInterface aci) { [
```

SIP MESSAGE event handler

public void onMessageEvent(javax.sip.RequestEvent event, ActivityContextInterface aci) {

```
Request request = event.getRequest();
try .
     // PARSING THE MESSAGE BODY
     String body = new String(request.getRawContent());
      int i = body.indexOf(" ");
      String timerValue = body.substring(0,i);
      int timer = Integer.parseInt(timerValue);
      String bodyMessage = body.substring(i+1);
      // SETTING VALUES ON THE ACTIVITY CONTEXT
      // USING THE SBB CUSTOM ACI
      WakeUnSbbActivitvContextInterface mvViewOfTimerBusACI =
     // SETTING THE TIMER BY USING THE VALUE
      // IN THE SIP MESSAGE BODY
      TimerOptions options = new TimerOptions();
      options.setPersistent(true);
      this.timerFacility.setTimer(timerBusACI,
              null.
              System.currentTimeMillis() +timer *1000,
              options);
```

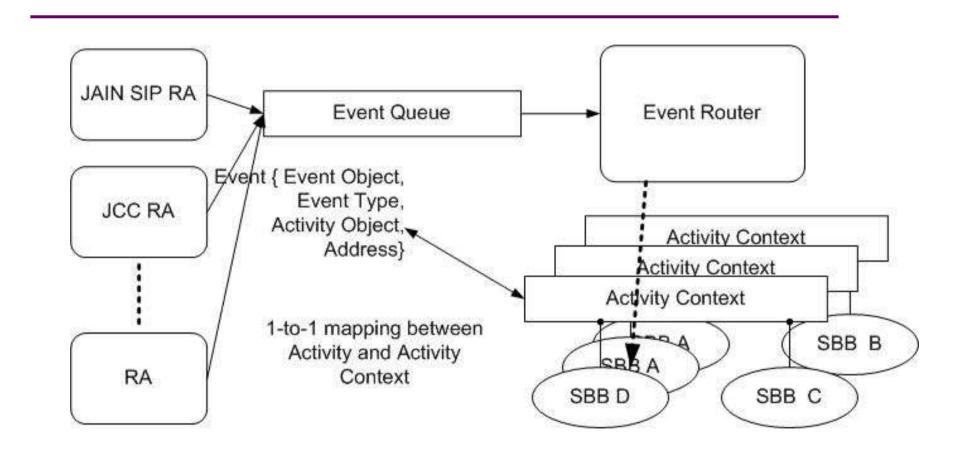
Timer Event Handler

```
public void onTimerEvent(TimerEvent event, ActivityContextInterface aci) {
    // RETRIEVING STORED VALUE FROM THE ACTIVITY CONTEXT INTERFACE
    WakeUpSbbActivityContextInterface myViewOfACI =
        this.asSbbActivityContextInterface(aci);
    Header contact = myViewOfACI.getContact();
    String body = myViewOfACI.getBody();

    // SENDING BACK THE WAKE UP CALL
    sendWakeUpCall(contact, body);
}
```

Some Implementation Details

Event Delivery



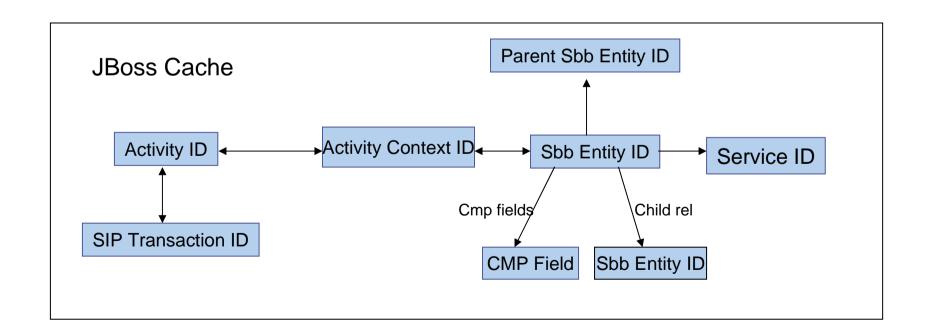
"Heart" of the SLEE is the Event Router. Each AC is an event queue.

Concurrency And Consistency

- SLEE spec is defined using a SERIALIZABLE consistency model.
- This implies that for concurrent event delivery the final state of the SLEE after consumption of these events should be achievable through some serial ordering of event delivery.
- Can be achieved using either optimistic or pessimistic concurrency control.
- Current version of Mobicents uses pessimistic concurrency control.
- Concurrency is at a per activity level.

Cached and Replicated Structures

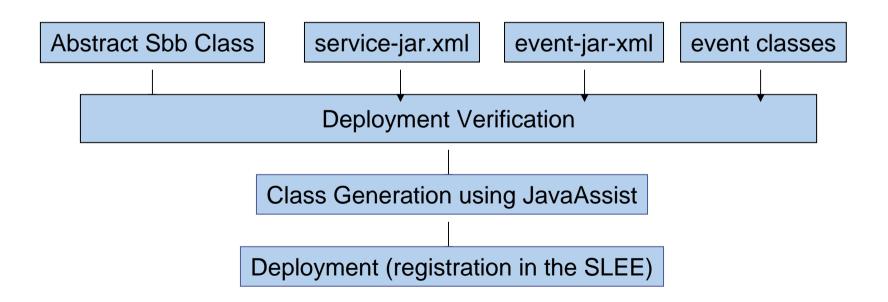
- ACs and Sbb Entities need to be replicated on Tx boundaries
 - ACs are visible to applications and represent a transitory event bus.
 - SBB Entities are the in memory representation of an SBB in use
 - JBoss Cache is very handy for this –provides transactional replication.



Deployment

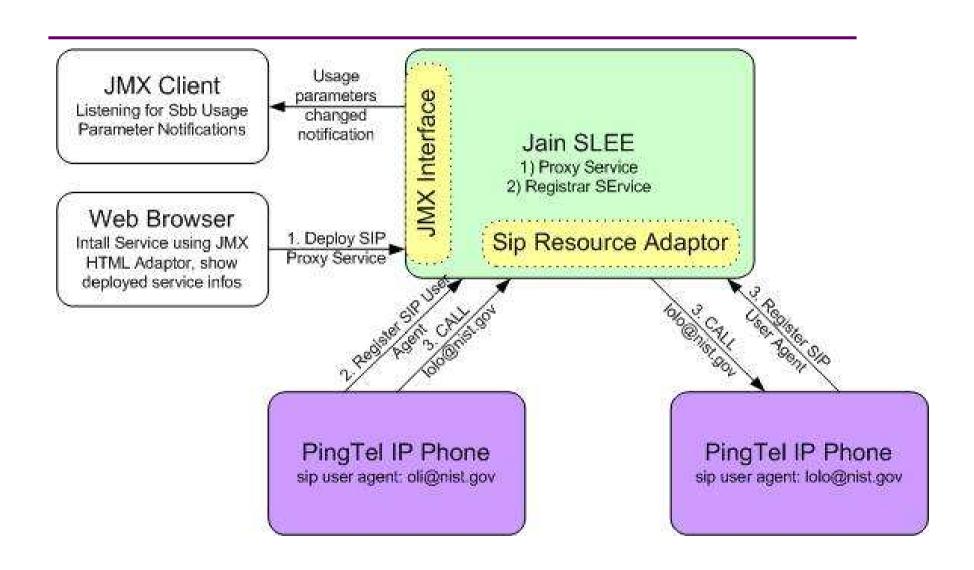
A deployable unit may contain Services, SBB jar files, Event jar files, Profile Specification jar files.

Each sbb has abstract methods for various operations: onXXX, CMP fields accessors. Profile CMP accessors, usage parameters. These are generated at deployment time to access container facilities for actually performing the operations.

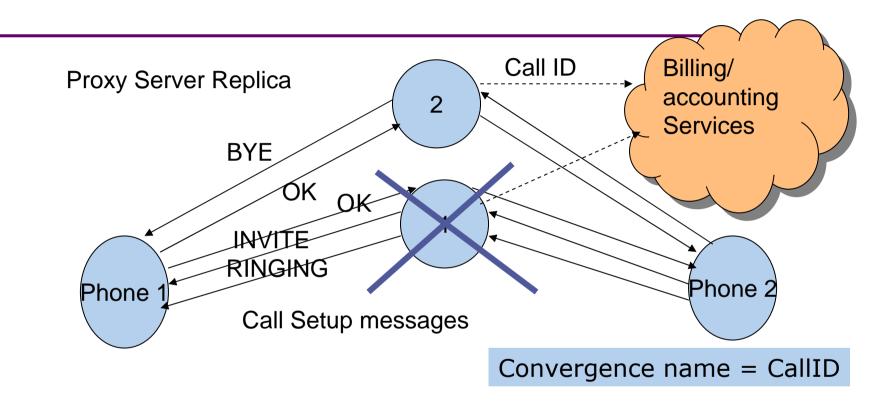


A More Sophisticated Example: SIP Proxy Server With Failover Support.

SIP Proxy as a SLEE Service



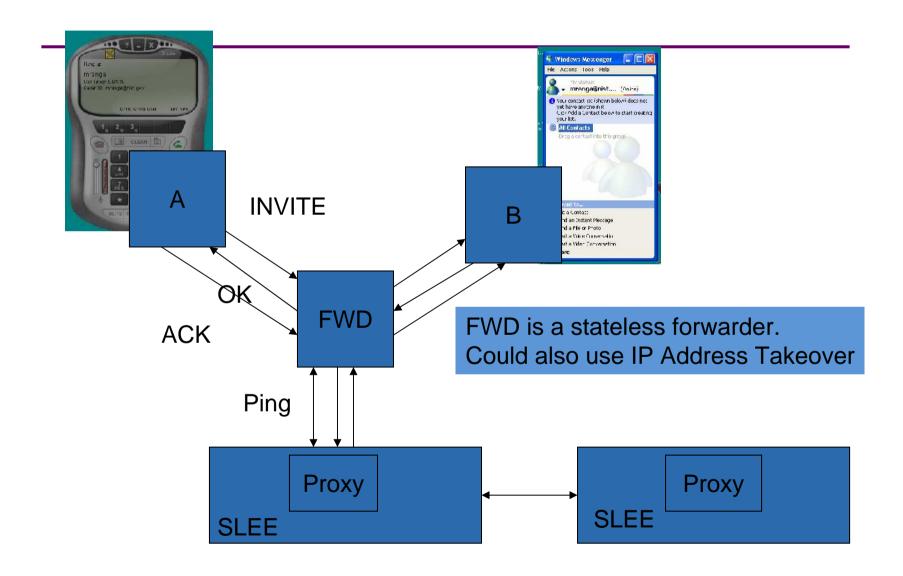
Replication and Persistence



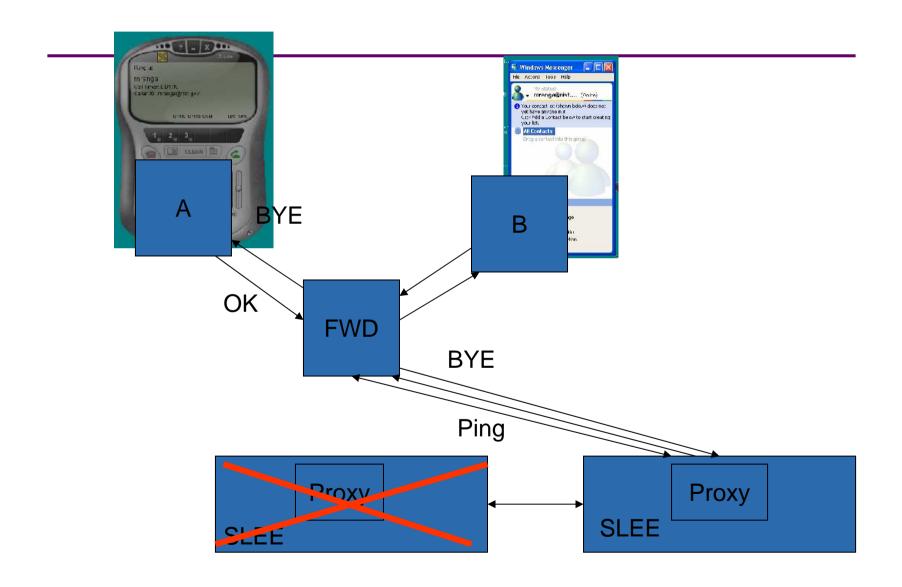
- Server 1 fails after call setup
- Server 2 takes over for Server 1
- •Server 2 needs to replicate the same service structure
- •Server 2 needs to know about the call (Call ID) that server 1 was handling and map it to the same Service

HA is still a work in progress

Demo Outline



Demo Outline (cont'd)



Demo Outline (cont'd)

- B registers with the proxy server.
- A calls B via the proxy.
- Proxy crashes replica takes over
- Hang up the call
 - Show that the Service survives the crash.
 - The initiator gets the BYE.
- Re initiate the call from A to B
 - Illustrates that the registrar survived the crash

Mobicents Community and Roadmap

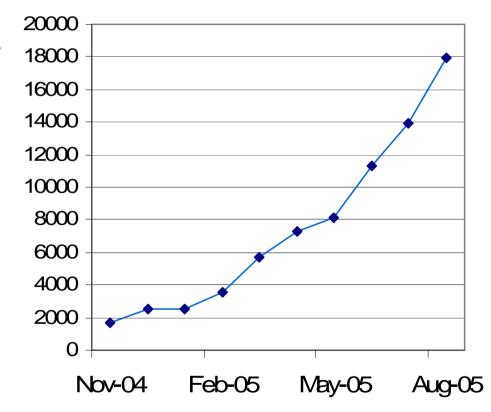
Ivelin Ivanov

Project Status

- ■1.0b1 release
- Beta quality
- More stable SLEE core
- Basic failover for SIP
- Much better performance ~ 15cps/CPU
- Profile persistence
- Auto deployment

What makes it possible?

- Committed Leadership
- Vibrant Community
 - 20+ registered developers
 - Daily forum discussions
 - Increasing Wiki Views



Mobicents Community

- NIST: Contributed SIP Stack, development of core platform
- University of Genoa: Core contributor.
- Luis Pasteur University:
 Continuous build and TCK runs
- Java.net: project hosting
- Portugal Telecom Innovacao: Asterisk RA, XMPP RA
- Lucent: Performance, profiling, HA

- Open Cloud: EclipSLEE, SIP RA
- Vodafone R&D: Examples, XMPP RA,
- <Tier 1 Telco>: Diameter RA
- <VolP vendor>: Parlay/Parlay-X
- ZyXEL: WiFi SIP Phone Prototype
- JBoss, Federation membership
- Several independent VoIP vendors

Technical Lead: M. Ranganathan Major Contributing Individuals: Francesco Moggia, Ivelin Ivanov, Jean Deruelle, Ralf Siedow, Buddy Bright, Leon Do, Marco Montiero, Luis Tiexeira, Tim Fox

Community feedback

Roadmap visibility and predictable timelines

- Most popular demand by far!
- Carrier grade quality
- Higher Availability
- Better performance 100+ cps
- More modules XMPP, Parlay, Skype, Asterisk, Diameter
- Sophisticated Administration GUI
- Tools for Rapid Service Development

What's next?

- Continue active development
 - HA, performance, new RAs
 - Enhance test framework
- Scale project coordination
 - Add new contributors
 - Preserve synergy and productivity
 - Ensure ongoing module integration
- Consider ideas for formal organization
 - Predictable roadmap
 - Measurable commitments by members
 - Professional support channels

