Implementing JAIN-SLEE on the JBoss AS
The Mobicents Open SLEE Project

M. Ranganathan and Francesco Moggia

NIST Advanced Networking Technologies Division
Gaithersburg MD
http://www.antd.nist.gov

University Of Genoa
http://www.dist.unige.it

Speaker Intro

- M. Ranganathan
  - ✓ Computer Scientist Advanced Networking Technologies Division, N.I.S.T.
  - ✓ Co-Spec Lead for JSR 32 JAIN-SIP
  - ✓ Development lead for Mobicents
  - ✓ Member of JSR 240 E.G.
- Francesco Moggia
  - ✓ PhD Student at University of Genoa
  - ✓ Guest Researcher at NIST
  - ✓ Core developer on Mobicents.

Talk Overview

- IP Telephony: more than telephony over IP
  - ✓ Services – the key differentiator.
  - ✓ Concrete Example: SIP services.
  - ✓ Converged Services
- The requirements of IP telephony services motivate a new container architecture:
  - ✓ What are the requirements of such services?
  - ✓ Why does EJB + Signaling Stack not adequately address these requirements?
  - ✓ What motivates the need for a new service architecture?

Talk Overview

- Implementing the JAIN-SLEE spec on JBoss:
  - ✓ Quick SLEE Demonstration : A SIP Proxy Server
  - ✓ Key JBoss AS components used in the implementation.
- Future Work
  - ✓ SLEE Enhancements for Security and Resource Control.
  - ✓ SLEE Enhancements for emergency signaling: The Emergency Call Control Scenario.

IP Telephony In the Large

- There's two parts to IP Telephony:
  - ✓ Call setup (signaling) and media.
  - ✓ Signaling is where the Network Intelligence (services) reside.
- This talk will focus on Signaling and Services

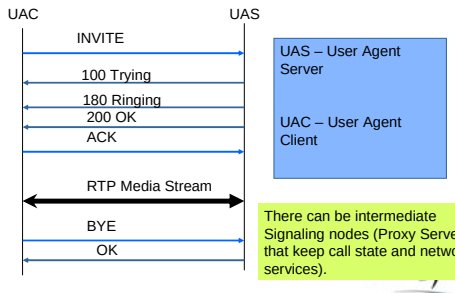
IP Telephony In the Large

- VOIP is everywhere!
  - ✓ Free or cheap voice is a commodity.
  - ✓ Services is the differentiator – the way to make revenue.
- VOIP affords flexibility
  - ✓ New classes of services become possible.
  - ✓ Converged services which combine VOIP and web services.

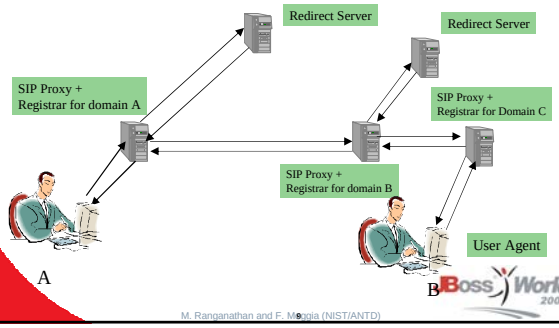
Signaling and Services

- In order to set up a call the two end-points (IP Phones) exchange messages.
- SIGNALING refers to the messages that are required to set up the call.
- SIGNALING is interesting because Services reside in the Signaling Plane.
- SIP is the dominant standard for call setup.
- We will motivate the requirements using SIP as an example.
  - ✓ SLEE is SIGNALING PROTOCOL AGNOSTIC.

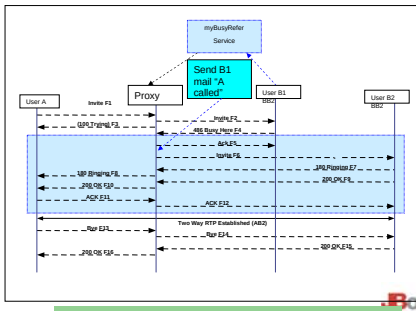
Motivating the Requirements
Example Simple SIP Call Flow



Motivating the Requirements
A Typical SIP Enabled Network



Custom SIP Service



Some Typical Services

- Call Hold.
- Consultation Hold
- Music On Hold.
- Unattended Transfer.
- Attended Transfer.
- Call Forwarding Unconditional.
- Call Forwarding - Busy.
- Call Forwarding - No Answer.
- 3-way Conference….
- See IETF Draft “SIP Service Examples”

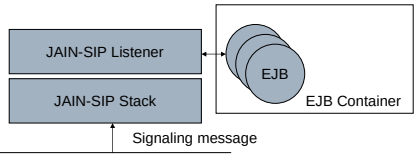
Motivating the Requirements
Service Structure

- Each signaling message triggers a fragment of code to run on the server.
- Each triggered fragment of code runs for a finite amount of time.
  - ✓ Services are event oriented, asynchronous and distributed.
  - ✓ Low latency event delivery requirements for scalability.

An Architecture for Building Services

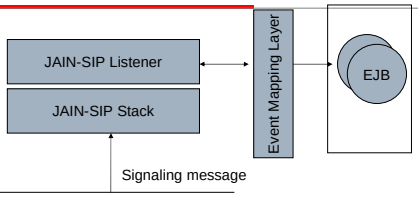
- Components are good.
  - ✓ But I am preaching to the choir!
  - ✓ We need a component oriented event driven service platform
- Need high reliability and failure resilience
- Transactions are good
  - ✓ Simplifies the task of building resilient applications.
  - ✓ But I am preaching again!
- So we need a component oriented transaction supporting, event driven platform.

A Possible architecture for Building Signaling Services



Tightly Coupled Listener
Constrains distribution.
Object management is under application control
Application Complexity
High Latency
Persistent state is stored in an EJB.

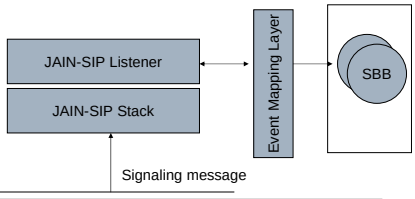
Refinement 1 Lets Add an Event Mapping Layer



Gets rid of tight coupling between stack and EJB
Still high latency and does not do much for structuring services.
EJB is tuned to the needs of Enterprise-oriented applications.
Heavy weight transactions (O(1) second latency)
Data objects with long persistent lifetimes.
IP Telephony services are asynchronous (triggered by one way messages).
EJBs are typically used with synchronous invocations (Round trip)

Refinement 2 : Lets Replace the EJB

EJB offers a nice component model.
Lets keep the cool stuff about the EJB model and toss the rest out.



- •Replace EJB with a lighter weight component - "SBB"
- •Event driven (one way messages)

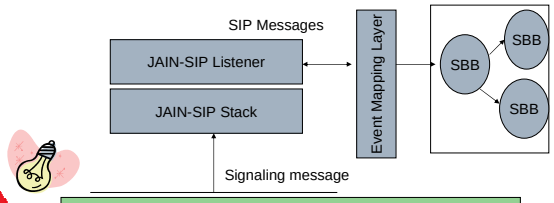
What about execution order of the SBBs?

Services and SBBs

- Services are compositional
- Each compositional block is an SBB.
  - ✓ SBB: Event Driven Service Building Block
- SBBs fire in response to events
- SBBs send each other events.
- Order of SBB execution is important
  - ✓ Otherwise outcome of composition is non-deterministic.

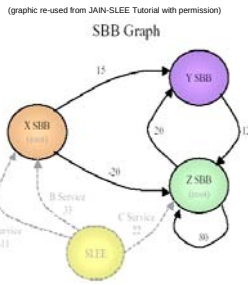
Refinement 3: Lets group and order the SBBs

Lets group SBBs and define a means for specifying execution order



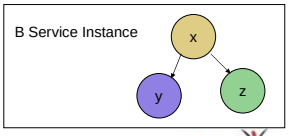
A Service is a group of related SBBs.
Deployment descriptor allows us to specify execution order of SBBs.

Service instantiation

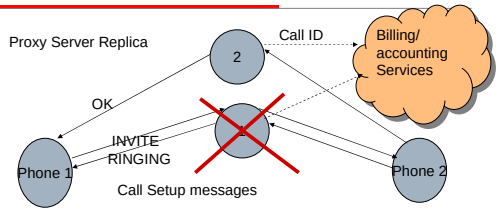


These are the SBBs known to the SLEE
This picture is known at Deployment time (through the deployment descriptors).

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



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 (Transaction ID) that server 1 was handling
- TID is transitory (only lasts for the setup of the call)

Replication and Persistence

- Replicated State:
  - ✓ Replicated state (instance variables) of SBBs including Call state, Presence state
  - ✓ Compositional state of the service
  - ✓ Mapping between SBBs and events
- Persistent state that can survive restarts:
  - ✓ Account information.
  - ✓ User service configuration information persistent.
  - ✓ Service State

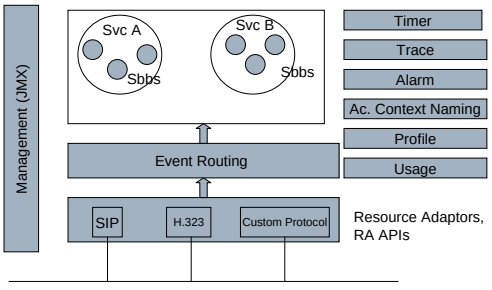
Summing it up: What is the SLEE ?

- JSR 22 (spec leads Open Cloud and Sun).
- Crafted for the needs of Communications service platforms
  - ✓ Highly Available
  - ✓ Scalable
  - ✓ Distributed
- Supports standard JMX Management Interfaces
- Supports standard facilities (timer, trace, usage, alarm)
- Point of integration for multiple protocols and components:
  - ✓ Events and components are strongly typed using java interfaces.
  - ✓ A single container can support multiple protocols

Summing it up: Why Invent the SLEE?

- Need to support Asynchronous invocations.
  - ✓ EJBs are typically synchronous
- SLEE is designed for fine grained short lived objects that are typically replicated in memory.
  - ✓ SLEE objects are replicated in memory.
  - ✓ SLEE transactions are light weight.
  - ✓ SLEE manages transaction boundaries.

Simplified JAIN-SLEE Architecture

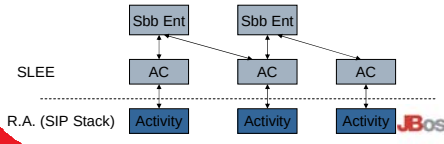


SLEE Building Blocks

- Event Type
  - ✓ Typed event model.
  - ✓ Resource adaptors generate events
- SBB
  - ✓ Fundamental building block (like an EJB).
  - ✓ SBBs can communicate by firing events ACIs
- Service
  - ✓ Management Artifact
  - ✓ Contains information for initial event processing
- Profile Specification
  - ✓ Provisioned data for management of services.
- Usage parameters interfaces
- JMX management clients

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.



Some Details

- SBBS are auto attached to ACIs
  - ✓ Declarative Event Subscription Model.
  - ✓ Does not allow SBBs to directly register themselves as *Listeners* for a resource. Indirection allows for distributed architecture.
  - ✓ The deployment descriptor indicates what events are of interest to root SBB of a service.
  - ✓ SLEE takes care of instantiation, attachment and routing events to the SBB.

SLEE Programming Model

- Component model shares many common concepts with EJB.
  - ✓ But it is different from the EJB model.
- Specialized component model for event driven applications.
  - ✓ Event names are mapped to method invocations.
  - ✓ Primary key is not directly visible to the application.
  - ✓ “Bean” creation and deletion is automatic and event triggered.

JMS vs. SLEE

- SLEE uses publish-subscribe model like JMS so why not just use it?
  - ✓ Impedance mismatch.
  - ✓ SLEE messages are supposed to be processed in 10-100 ms. JMS messages could take anywhere from seconds to days. Results in different implementation strategies.
  - ✓ The “Topics” are not known a-priori here.
  - ✓ JMS drags in baggage that we don’t want.

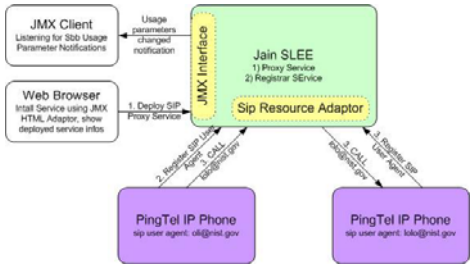
JMS vs. SLEE

- JMS does not have some of the features that we do want:
  - ✓ Endpoints talk SIP not JMS messages. We would need to encapsulate SIP in JMS.
  - ✓ JMS does not have event triggered object creation. SLEE will create SBBs in response to message arrivals.
  - ✓ JMS does not have built in event triggered garbage collection. SLEE will pool the resources for a service after the event processing is complete.

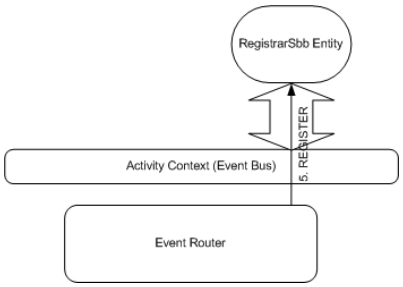
The Mobicents Project

- Purpose – to build an open source experimental SLEE implementation.
- Project is housed at http://www.mobicents.org
- Development Lead: M. Ranganathan (NIST)
- Core Contributors : Francesco Moggia, Tim Fox, Jean Deruelle.
- Significant contributions to date: Vodafone R&D (Ralf Siedow), Lucent Technologies (team lead Buddy Bright), Emil Ivov, Jean-Noel Gadreau.
- Interest (with potential contributions) from : PT Innovaco, TI Labs.
- JBoss Technical Advisor: Ivelin Ivanov.
- An active project with a growing list of contributors!

Demo 1



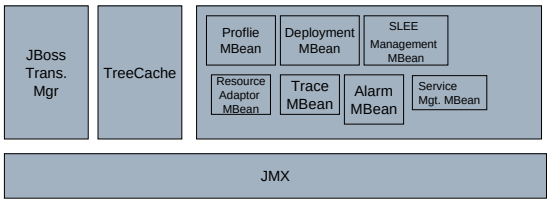
Demo 2



SLEE as a JBoss Service

- JAIN SLEE is not a J2EE Spec. However utilizes many J2EE components like JMX, Transactions.
- JBoss AS is an application building platform that provides many facilities that we use.
- Some useful JBoss services and tools:
  - ✓ JBoss Cache, JMX, JNDI, JavaAssist, JBoss Clustering.

SLEE as a JBoss Service

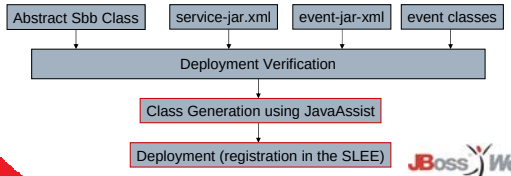


Deployment

- The SLEE specification defines the SBB Interface.
  - ✓ Some methods of the SBB must be abstract.
- The SLEE deployment tool generates the SBB concrete class by implementing the SBB abstract class.

Using JavaAssist for 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.

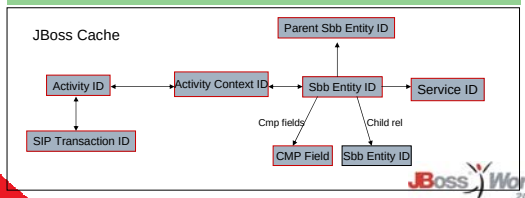


Transactions, Replication and Caching

- Activities are SLEE Representation of Events.
  - ✓ An Activity is associated with an Activity Context (AC)
  - ✓ The SLEE event delivery model is transactional – each event handler method runs in its own transaction.
  - ✓ State propagation can happen at transactional boundaries.
  - ✓ Transactions simplify design.

Cached and Replicated Structures

- For failover handling ACs need to be replicated on Tx boundaries
  - ✓ ACs are visible to applications and represent a transitory event bus.
  - ✓ ACs, their attachment to SBBs and their state need to be replicated for failover handling.
  - ✓ JBoss Cache is very handy for this –provides transactional distributed caching.



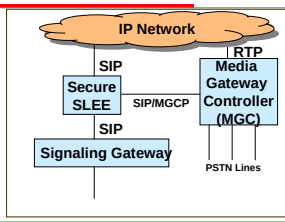
JMX and Management

- We utilize JBoss AS JMX implementation.
- The SLEE Specification relies on JMX to standardize the management interface
- A Service is a management artifact in the SLEE.
- The SLEE exports a SLEE Management MBean.
- Each Facility exports an MBean.
- Each Service exports a Service MBean
- The Service Management MBean may be accessed through the SLEE MBean and used to control the service.

SLEE Extensions: Security

- NIST Research project Survivable IP Telephony Service Platform
- SLEE currently does not have a security model.
  - ✓ Assumes a secure environment.
  - ✓ SLEE Operator trusts SBB writers.
  - ✓ SLEE Operator cannot place deployment time roles and constraints for SBB providers.
- We are adding a security layer to the SLEE

Use Case Scenario

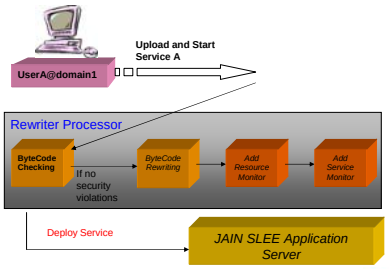


When an emergency call setup request (ie. GETS SIP INVITE) arrives at the proxy, we want to be able to:
Identify the call as having been originated by an emergency response worker.
Allow the invoked Service to access the resources it needs to setup the requested call.
Access any privileged resources it needs.

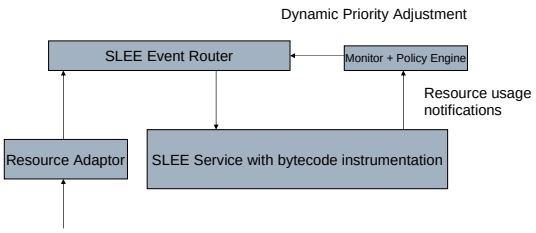
Uploadable SLEE Services

- NIST Research Project.
- Install and monitor un-trusted components in the SLEE
- Control the resources that these components use.
- Extend the Usage Parameters model for CPU and Memory.

Uploadable SLEE Services



Resource Constrained Services



Reference Material

- JAIN-SLEE Specification, white papers and tutorials
  - ✓ http://jainslee.org/slee/slee.html
- JAIN-SLEE Principles
  - ✓ http://java.sun.com/products/jain/article_slee_principles.html
- SIP RFC 3261
  - ✓ http://www.faqs.org/rfcs/rfc3261.html

Acknowledgement

- The JAIN-SLEE Specification is lead by Sun Microsystems and Open Cloud.
- Material from the JAIN-SLEE tutorial is reused in this presentation with permission.
- Mobicents is supported in part by the NIST Advanced Networking Technologies Division, JBoss and others.