JBoss Communications JAIN SLEE Call Controller2 Example User Guide

by Eduardo Martins, Bartosz Baranowski, and Alexandre Mendonça

Preface v
1. Document Conventions v
1.1. Typographic Conventions v
1.2. Pull-quote Conventions vii
1.3. Notes and Warnings vii
2. Provide feedback to the authors! viii
1. Introduction to JBoss Communications JAIN SLEE Call Controller2 Example 1
2. Setup 3
2.1. Pre-Install Requirements and Prerequisites 3
2.1.1. Hardware Requirements 3
2.1.2. Software Prerequisites 3
2.2. JBoss Communications JAIN SLEE Call Controller2 Example Source Code 3
2.2.1. Release Source Code Building 3
2.2.2. Development Trunk Source Building 4
2.3. Installing JBoss Communications JAIN SLEE Call Controller2 Example 4
2.4. Uninstalling JBoss Communications JAIN SLEE Call Controller2 Example 4
3. Design Overview 7
3.1. Call Controll Profile 7
3.1.1. Profile default data 9
3.2. ACI variable aliasing and service composition 9
3.3. Call Blocking Service 12
3.4. Call Forwarding Service 13
3.5. Voice Mail Service 14
3.5.1. Box access 14
3.5.2. Message recording 16
3.5.3. Media path 16
4. Source Code Overview 19
4.1. Services descriptor source 19
4.2. SBB SLEE Facilities access 20
4.2.1. Initial event selector 22
4.3. Profile Specification Source 23
4.3.1. Profile descriptor 23
4.3.2. Profile interface and management 24
4.4. Blocking Service Source 27
4.4.1. Service root 27
4.4.2. Events handlers 27
4.4.3. User profile access 29
4.4.4. Call Blocking SBB descriptor 30
4.5. Forwarding Service Source 32
4.5.1. Service root 32
4.5.2. Events handlers 32
4.5.3. User status check 34
4.5.4. Call forwarding 35
4.5.5. User profile access 36

4.5.6. Call Forwarding SBB descriptor 37
4.6. Voice Mail Service Source 40
4.6.1. Service root 40
4.6.2. SIP Event handlers 40
4.6.3. MGCP Event handlers 45
4.6.4. MGCP signals 50
4.6.5. Voice Mail profile access 53
4.6.6. Voice Mail SBB descriptor 54
5. Running the Example 59
5.1. Configuration 59
6. Traces and Alarms 61
6.1. Tracers 61
6.2. Alarms 62
A. Revision History 63
Index 65

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
  public static void main(String args[])
      throws Exception
  {
    InitialContext iniCtx = new InitialContext();
    Object         ref    = iniCtx.lookup("EchoBean");
    EchoHome       home   = (EchoHome) ref;
    Echo           echo   = home.create();

    System.out.println("Created Echo");

    System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
  }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.



Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications JAIN SLEE Call Controller2 Example**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_CallController2_EXAMPLE_User_Guide

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to JBoss Communications JAIN SLEE Call Controller2 Example

This example is a JAIN SLEE application which acts as simple call center. Services acts as voicemail with call forwarding and blocking features.

Example ilustrate one of patterns to compose different services into working logical service.

It is not trivial example as it involves more than one protocol and builds logical abstraction over three different services.

Example provides usage of JSLEE facilities, `Activity Context Interface` variables and aliasing, SBB activity context interfaces

Setup

2.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

2.1.1. Hardware Requirements

The Example doesn't change the JBoss Communications JAIN SLEE Hardware Requirements, refer to JBoss Communications JAIN SLEE documentation for more information.

2.1.2. Software Prerequisites

The Example requires JBoss Communications JAIN SLEE properly set, with SIP RA , MGCP RA and sip-services deployed.

Sip-services must use profile with INVITE event as not initial.

Media server must run with enabled MGCP controller.

2.2. JBoss Communications JAIN SLEE Call Controller2 Example Source Code

This section provides instructions on how to obtain and build the Call Controller2 Example from source code.

2.2.1. Release Source Code Building

1. **Downloading the source code**



Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at *http://svnbook.red-bean.com*

Use SVN to checkout a specific release source, the base URL is ?, then add the specific release version, lets consider 2.3.0.FINAL.

```
[usr]$ svn co ?/2.3.0.FINAL slee-example-call-controller2-2.3.0.FINAL
```

2. **Building the source code**



Important

Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Maven to build the deployable unit binary.

```
[usr]$ cd slee-example-call-controller2-2.3.0.FINAL
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if JBoss Communications JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Enterprise Application Platform directory, then the deployable unit jar will also be deployed in the container.

2.2.2. Development Trunk Source Building

Similar process as for *Section 2.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is NOT AVAILABLE.

2.3. Installing JBoss Communications JAIN SLEE Call Controller2 Example

To install the Example simply execute provided ant script `build.xml` default target:

```
[usr]$ ant
```

The script will copy the Example's deployable unit jar to the `default` JBoss Communications JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=`.

2.4. Uninstalling JBoss Communications JAIN SLEE Call Controller2 Example

To uninstall the Example simply execute provided ant script `build.xml undeploy` target:

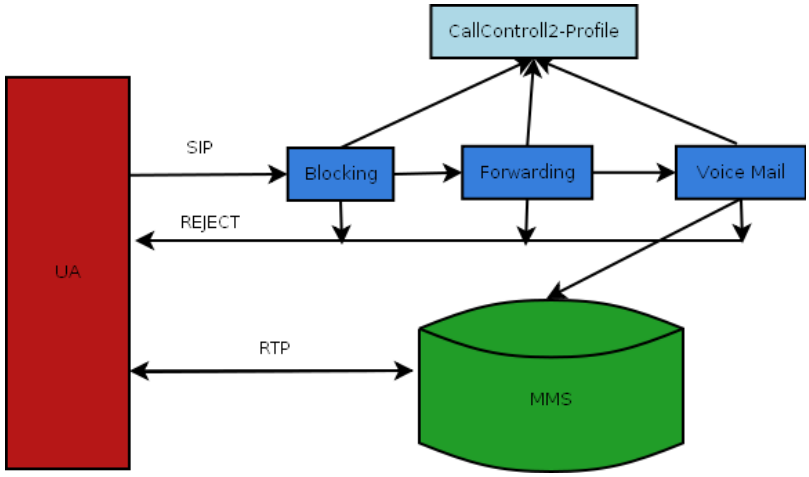
```
[usr]$ ant undeploy-all
```

The script will delete the Example's deployable unit jar from the `default` JBoss Communications JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.

Design Overview

Example is created by three independent services which compose logical whole. `CallBlocking`, `CallForwarding` and `VoiceMail` process incoming `INVITE` request in sequence and make decision based on certain configurable conditions.

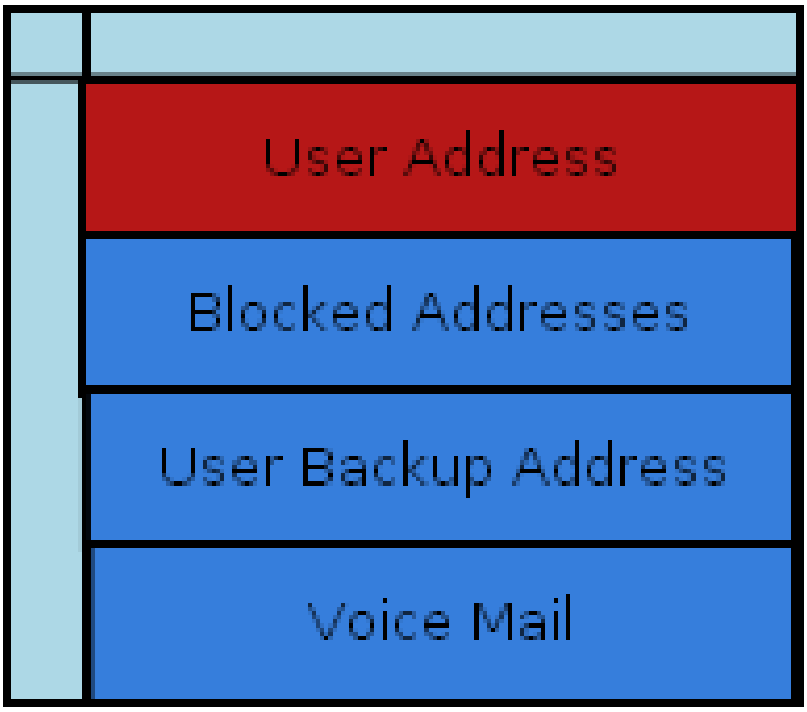
Top view of example components look as follows:



Call Controller2 Top overview of components

3.1. Call Controll Profile

Profile specification is JSLEE API to access non SBB data. It provides `call controller` services with required data to make decision how call should look like. Structure of profile data look as follows:



Call Controller2 Profile Specification

Table 3.1. Call Controller2 Profile attributes

Name	Type	Description
userAddress	javax.slee.Address	Determines AOR(address of record) of profile owner.
blockedAddresses	javax.slee.Address[]	Set of addresses blocked for profile owner. Calls originating from one of those should be blocked.
backupAddress	javax.slee.Address	Backup address to which call should be forwarded
voicemailState	boolean	Determines if user has enabled voice mail service, `true` indicates enabled voice mail.



Note

Addresses stored in profiles are AORs - that is logical addresses of user present in `From` and `To` headers for instance. Example AOR : sip:newbie@mobicents.org..

3.1.1. Profile default data

Profile table for Call Controller2 is "Controller"

Table 3.2. Call Controller2 Profile default values

Profile Name	User Address	Blocked Addresses	Backup Address	Voice Mail State
torosvi	SIP:sip:torosvi@ b127.0.0.1	{SIP:sip:mobicents@ 127.0.0.1, SIP:sip:hugo@ 127.0.0.1}	null	true
mobicents	SIP:sip:mobicents@ 127.0.0.1	null	null	false
victor	SIP:sip:victor@ 127.0.0.1	null	SIP:sip:torosvi@ 127.0.0.1	false
vhros2	SIP:sip:vhros2@ 127.0.0.1	null	null	true
vmail	SIP:sip:vmail@ 127.0.0.1	null	null	true



Important

Spaces where introduced in `address type field` column values, to correctly render the table. Please remove them when using copy/paste.



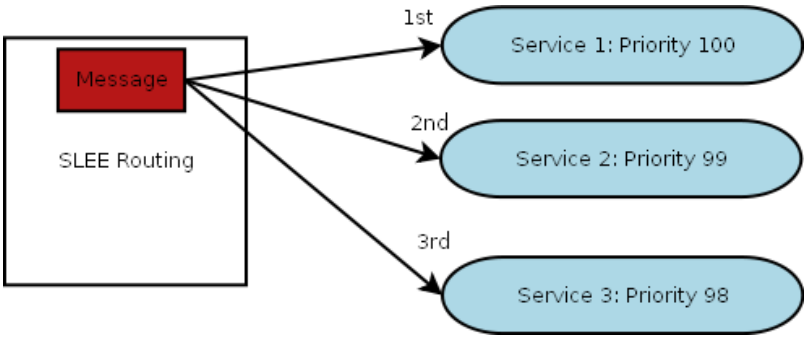
Note

Prefix `SIP:` comes from JSLEE addressing scheme.

3.2. ACI variable aliasing and service composition

Call control services act independently from each other. New services can be inserted into chain without modifying source code of existing pieces.

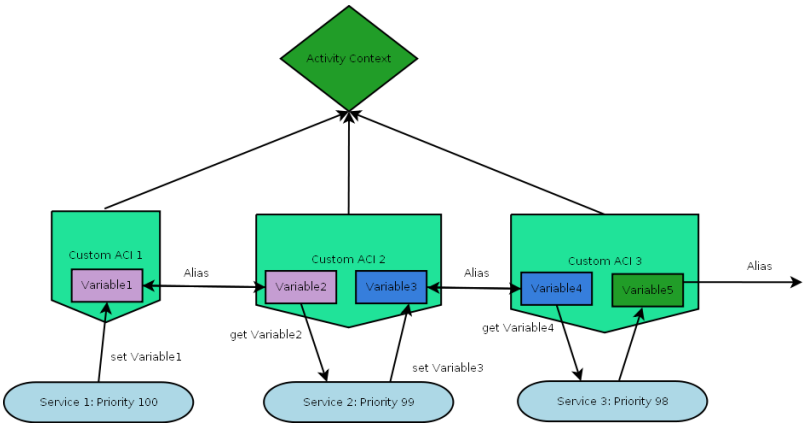
Chain of services is built with service priority feature of JSLEE:



Call Controller2 Message routing

Service priority determines order by which services are chosen to receive the same event. However JLSEE services are independent entities, that is, each has separate state associated with incoming message. To indicate that event has been consumed and must not be processed anymore services need to share information. This is done with ACI variable aliasing. It allows to share data stored in ACI variable

Aliasing scheme used by this example look as follows:



Call Controller2 Aliasing overview



Important

Aliased ACI variables must be of same, serializable type. However alias name must be distinct



Note

There is no limitation on how many variables and ACIs pairs can be aliased, ie. three ACI can share the same variable.

Aliases are created on custom ACI variables. Custom ACI and aliases are defined in sbb-jar.xml as follow:

```
<sbb>
   <description />
   <sbb-name>CallBlockingSbb</sbb-name>
   <sbb-vendor>org.mobicents</sbb-vendor>
   <sbb-version>0.1</sbb-version>



   <sbb-classes>
      ...
      <sbb-activity-context-interface>
         <sbb-activity-context-interface-name>

org.mobicents.slee.examples.callcontrol.blocking.CallBlockingSbbActivityContextInterface
         </sbb-activity-context-interface-name>
      </sbb-activity-context-interface>
   </sbb-classes>

    ....

   <activity-context-attribute-alias>
      <attribute-alias-name>inviteFilteredByCallBlocking</attribute-alias-name>
      <sbb-activity-context-attribute-name>filteredByMe</sbb-activity-context-attribute-name>
   </activity-context-attribute-alias>

</sbb>
<sbb>
   <description />
   <sbb-name>CallForwardingSbb</sbb-name>
   <sbb-vendor>org.mobicents</sbb-vendor>
   <sbb-version>0.1</sbb-version>


   <sbb-classes>
      <sbb-abstract-class>
         ...
      <sbb-activity-context-interface>
         <sbb-activity-context-interface-name>

org.mobicents.slee.examples.callcontrol.forwarding.CallForwardingSbbActivityContextInterface
         </sbb-activity-context-interface-name>
      </sbb-activity-context-interface>
```

```
        </sbb-classes>


        <activity-context-attribute-alias>
            <attribute-alias-name>inviteFilteredByCallBlocking</attribute-alias-name>
              <sbb-activity-context-attribute-name>filteredByAncestor</sbb-activity-context-attribute-
name>
        </activity-context-attribute-alias>
        <activity-context-attribute-alias>
            <attribute-alias-name>inviteFilteredByCallForwarding</attribute-alias-name>
            <sbb-activity-context-attribute-name>filteredByMe</sbb-activity-context-attribute-name>
        </activity-context-attribute-alias>

    </sbb>
```

Source definition of ACI and variable accessors look as follows:

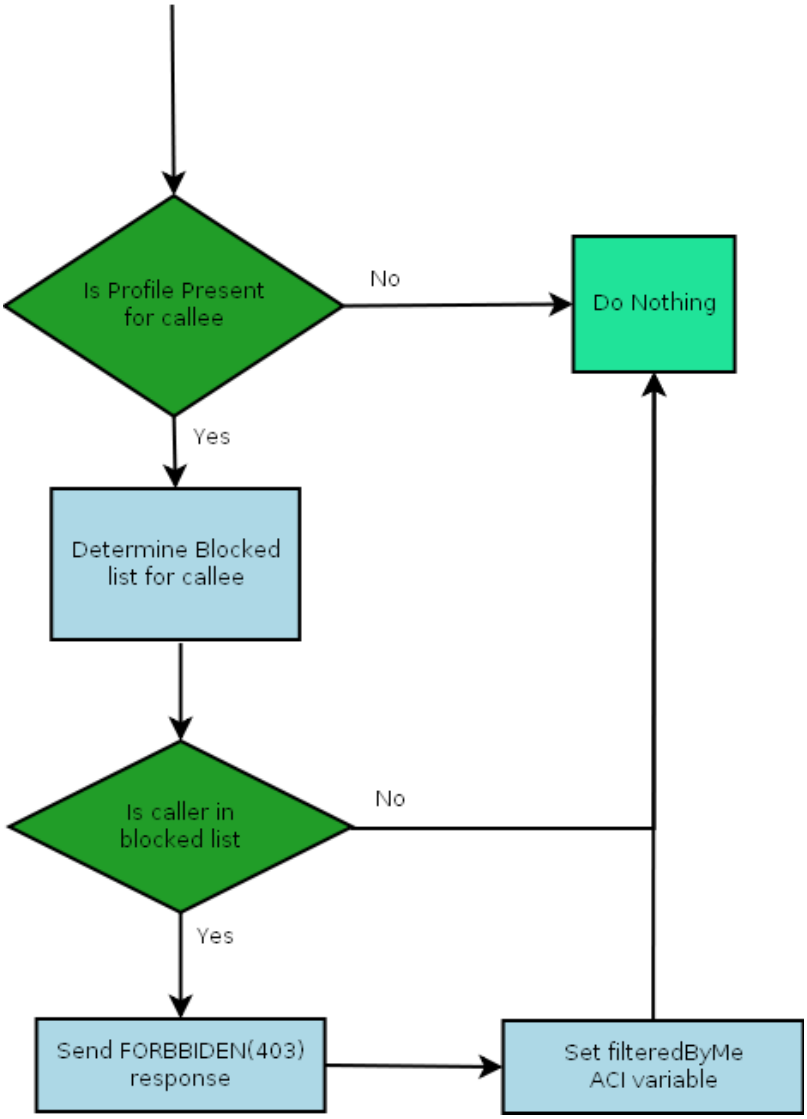
```
public interface CallBlockingSbbActivityContextInterface
    extends javax.slee.ActivityContextInterface {
    public void setFilteredByMe(boolean val);
    public boolean getFilteredByMe();
}
```

```
public interface CallForwardingSbbActivityContextInterface
    extends javax.slee.ActivityContextInterface {
    public boolean getFilteredByAncestor();
    public void setFilteredByMe(boolean val);
    public boolean getFilteredByMe();
}
```

3.3. Call Blocking Service

Call blocking service purpose is to block incoming calls based on data stored in call controll2 profile. It is first service to receive incoming SIP INVITE request.

Flow diagram for call blocking look as follows:



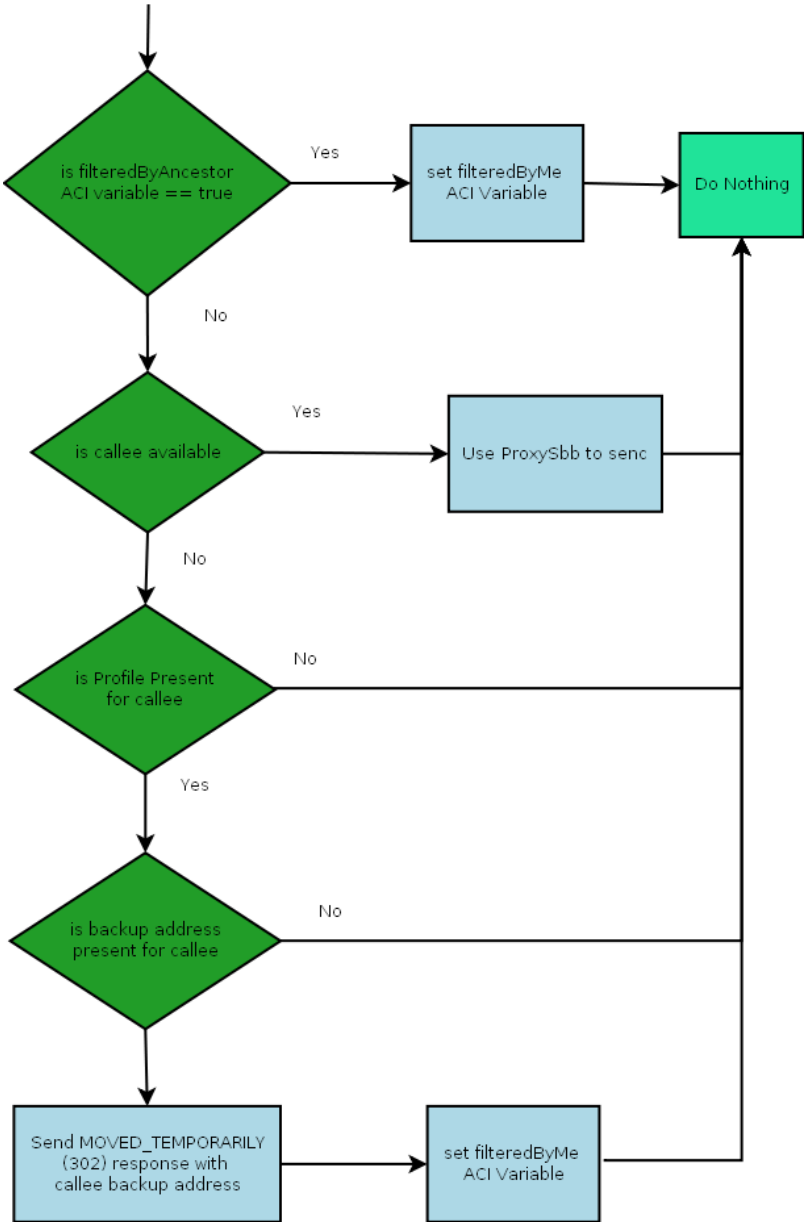
Call Controller2 Blocking Service flow

3.4. Call Forwarding Service

Call forwarding purpose is to simply forward call to callees backup address in case main is not available.

It is second service to receive incoming SIP INVITE request. Call forwarding service relies on sip-services proxy to route messages to proper destination UA.

Flow diagram for call forwarding look as follows:



Call Controller2 Forwarding Service flow

3.5. Voice Mail Service

Voice mail purpose is to store voice message and play it back when callee requests it.

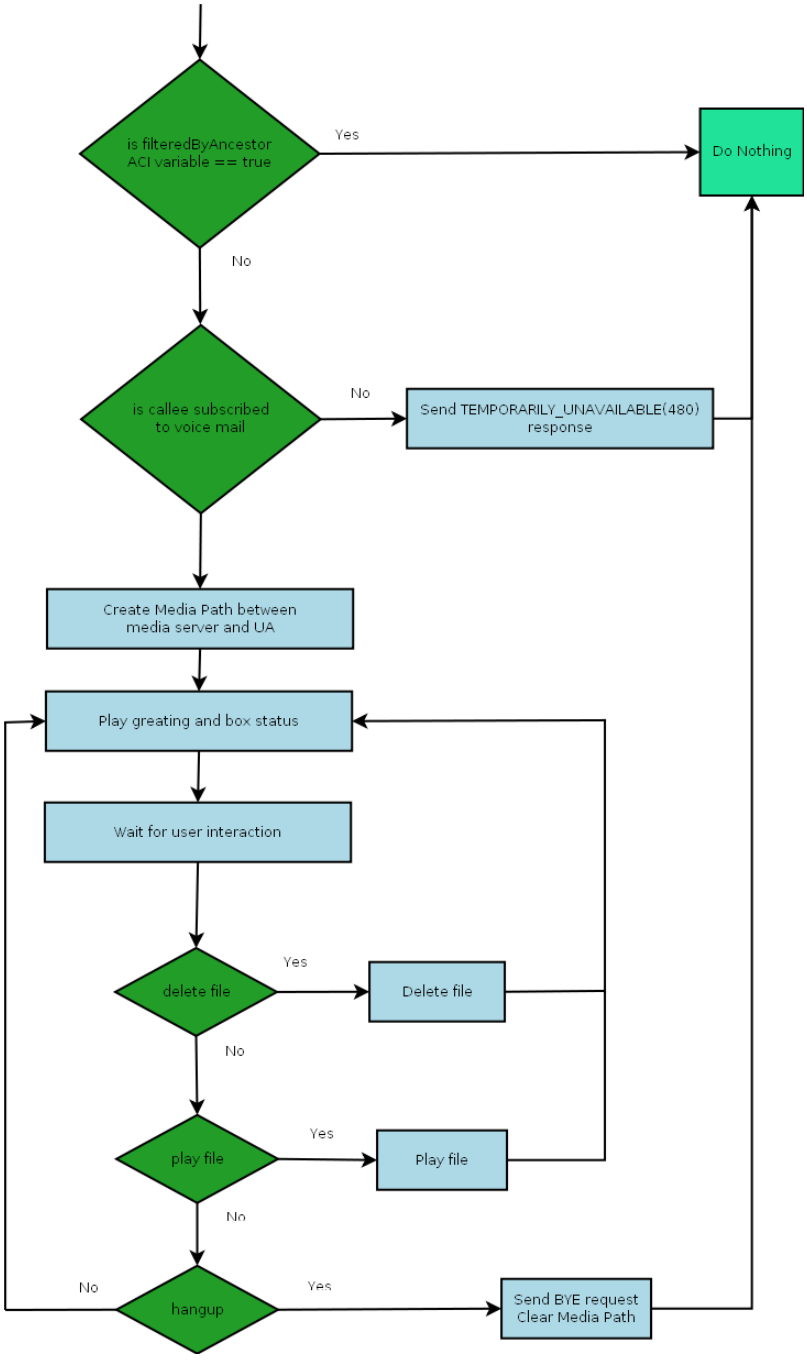
It is the last to receive incoming SIP INVITE request.

Voice mail service is most complicated service of Call Controller2. It carries on task of communicating with SIP UA and media server. That includes SDP negotiation procedures and media server primitives handling.

3.5.1. Box access

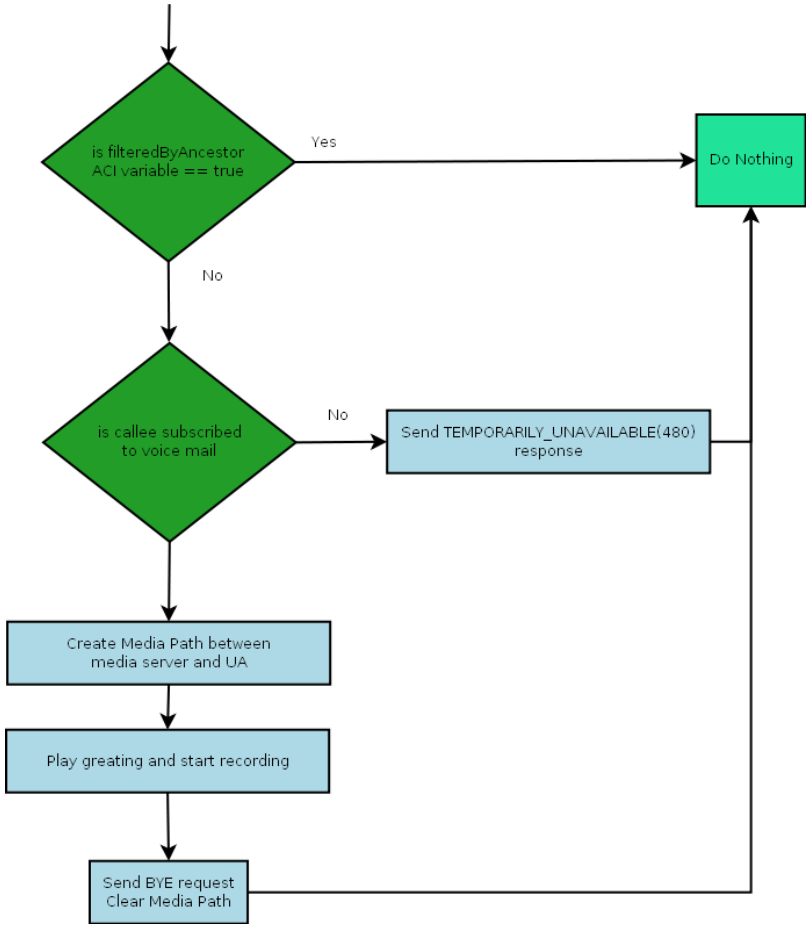
Users accesse voice mail box by calling `sip:vmail@mobicents.org`.

Flow diagram for voice mail look as follows:



Call Controller2 Voice Mail Service playback flow

3.5.2. Message recording



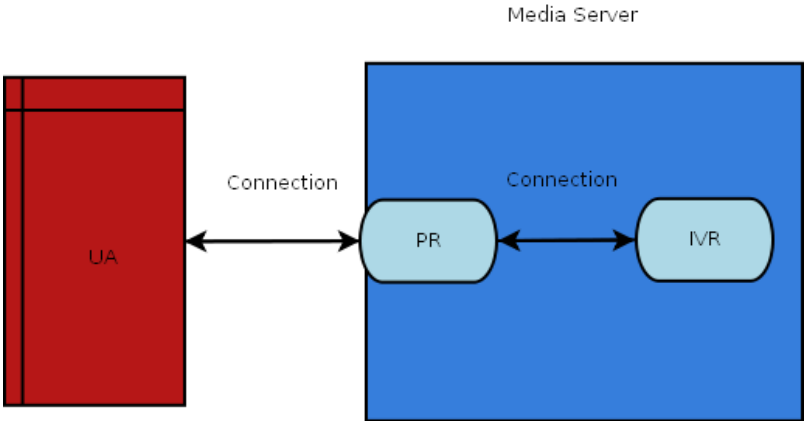
Call Controller2 Voice Mail Service record flow

3.5.3. Media path

Voice mail must create media session and establish media path in media server in order to perform its function.

Media path is created in media server with `connections` and `endpoints` (for explanation of acronyms and capabilities please refer to `Mobicents Media Server Documentation`)

Path is composed as follows:



Call Controller2 Media Path

For details of creating media session please refer to *RFC4566* [http://tools.ietf.org/html/rfc4566]

Source Code Overview



Important

To obtain the example's complete source code please refer to *Section 2.2, "JBoss Communications JAIN SLEE Call Controller2 Example Source Code"*.

Chapter *Chapter 3, Design Overview* explains top level view of example. This chapter explains how components perform their tasks. For more detailed explanation of JSLEE related source code and xml descriptors, please refer to simpler examples, like `sip-wakeup`

4.1. Services descriptor source

Each components of Call Controller2 runs as independent service. Each service is defined as follows(in order of priority):

- Call blocking service

```
<service-xml>
   <service>
      <description/>
      <service-name>CallBlockingService</service-name>
      <service-vendor>org.mobicents</service-vendor>
      <service-version>0.1</service-version>
      <root-sbb>
         <description/>
         <sbb-name>CallBlockingSbb</sbb-name>
         <sbb-vendor>org.mobicents</sbb-vendor>
         <sbb-version>0.1</sbb-version>
      </root-sbb>
      <default-priority>100</default-priority>
   </service>
</service-xml>
```

- Call forwarding

```
<service-xml>
   <service>
      <description/>
      <service-name>CallForwardingService</service-name>
      <service-vendor>org.mobicents</service-vendor>
      <service-version>0.1</service-version>
      <root-sbb>
         <description/>
         <sbb-name>CallForwardingSbb</sbb-name>
         <sbb-vendor>org.mobicents</sbb-vendor>
         <sbb-version>0.1</sbb-version>
      </root-sbb>
      <default-priority>0</default-priority>
   </service>
</service-xml>
```

- Voice mail

```
<service-xml>
   <service>
      <description/>
      <service-name>VoiceMailService</service-name>
      <service-vendor>org.mobicents</service-vendor>
      <service-version>0.1</service-version>
      <root-sbb>
         <description/>
         <sbb-name>VoiceMailSbb</sbb-name>
         <sbb-vendor>org.mobicents</sbb-vendor>
         <sbb-version>0.1</sbb-version>
      </root-sbb>
      <default-priority>-50</default-priority>
   </service>
</service-xml>
```

4.2. SBB SLEE Facilities access

Call Controller2 SBBs access JSLEE facilities in the same way. Code to perform this tasks is generic and gathered in super class for all Call Controller2 SBB classes.

It does following:

- retrieve JSLEE facilities and SIP RA interfaces

```
public void setSbbContext(SbbContext context) {
    this.sbbContext = context;
    try {
        //"If NamingException is thrown check jmx-console -> JNDIView ->
        // list or sbb-jar (check entity-bindning) for proper JNDI path!!!!"
        Context myEnv = (Context) new InitialContext().lookup("java:comp/env");
        // Getting JAIN SIP Resource Adaptor interfaces
        fp = (SleeSipProvider) myEnv.lookup("slee/resources/jainsip/1.2/provider");
        // To create Address objects from a particular implementation of JAIN SIP
        addressFactory = fp.getAddressFactory();
          // To create Request and Response messages from a particular implementation of
JAIN SIP
        messageFactory = fp.getMessageFactory();
        // To allow SBB entities to interrogate the profile database to find
        // profiles that match a selection criteria
        profileFacility = (ProfileFacility) myEnv.lookup("slee/facilities/profile");

    } catch (NamingException ne){
        log.error("COULD NOT LOCATE RESOURCE IN JNDI: Check JNDI TREE or entity-
binding for proper path!!!", ne);
    }
}
```

- provide accessors to facilities with simple getters
- provide profile CMP interface lookup

```
protected CallControlProfileCMP lookup(javax.slee.Address address) {
    String profileTableName = new String();
    //ProfileID profileID = null;
    CallControlProfileCMP profile = null;

    try {
        profileTableName = "CallControl";

        ProfileTable table = getProfileFacility().getProfileTable(profileTableName);
```

```
            ProfileLocalObject plo=table.findProfileByAttribute("userAddress", address);
            profile = (CallControlProfileCMP) plo;


        }catch (NullPointerException  e) {
            log.error("Exception using the getProfileByIndexedAttribute method", e);
        }catch (UnrecognizedProfileTableNameException e) {
                                    log.error("Exception in getting the Profile Specification in
getControllerProfileCMP(profileID):" +
                    "The ProfileID object does not identify a Profile Table created from the Profile
Specification", e);
        }catch (TransactionRolledbackLocalException e) {
            log.error(e.getMessage(), e);
        }catch (FacilityException e) {
            log.error(e.getMessage(), e);
        }

        return profile;
    }
```

4.2.1. Initial event selector

Common code provides all SBBs with initial event selector call back method. This method is called by JSLEE for each event declared as initial. It assigns unique name to service instance handling call. For descriptor declaration please refer to SBB event handler description.

```
public InitialEventSelector callIDSelect(InitialEventSelector ies) {
    Object event = ies.getEvent();
    String callID = null;

    if (event instanceof RequestEvent) {
        // If request event, the convergence name to callId
        Request request = ((RequestEvent) event).getRequest();
        callID = ((CallIdHeader) request.getHeader(CallIdHeader.NAME)).getCallId();
    }

    ies.setCustomName(callID);
    return ies;
}
```

4.3. Profile Specification Source

Call Controller2 profile specification is not complicated. It is designed only as simple service data container. It does not perform any other tasks.

Profile specification descriptor for Call Controller2 declares following values:

- Profile ID - that is name,vendor and version.
- profile CMP interface
- profile abstract class

4.3.1. Profile descriptor

Descriptor look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE profile-spec-jar PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE Profile Specification 1.1//EN" "http://java.sun.com/dtd/slee-profile-spec-jar_1_1.dtd">
<profile-spec-jar>
   <profile-spec>
      <description />
      <profile-spec-name>CallControlProfileCMP</profile-spec-name>
      <profile-spec-vendor>org.mobicents</profile-spec-vendor>
      <profile-spec-version>0.1</profile-spec-version>

      <profile-classes>
         <profile-cmp-interface>
            <profile-cmp-interface-name>
               org.mobicents.slee.examples.callcontrol.profile.CallControlProfileCMP
            </profile-cmp-interface-name>
            <!-- <profile-index unique="True">userAddress</profile-index>  -->
            <cmp-field>
            <cmp-field-name>userAddress</cmp-field-name>
               <index-hint query-operator="equals"/>
            </cmp-field>
         </profile-cmp-interface>
         <profile-abstract-class>
            <profile-abstract-class-name>
               org.mobicents.slee.examples.callcontrol.profile.CallControlProfileManagementImpl
            </profile-abstract-class-name>
         </profile-abstract-class>
      </profile-classes>
```

```
    </profile-spec>
</profile-spec-jar>
```

4.3.2. Profile interface and management

Profile data can be accessed and modified with profile CMP interface. It is interface exposed to JSLEE components. It is defined as follows:

```
package org.mobicents.slee.examples.callcontrol.profile;

import javax.slee.Address;

public interface CallControlProfileCMP {
    // 'userAddress' CMP field setter
    public abstract void setUserAddress(Address value);
    // 'userAddress' CMP field getter
    public abstract Address getUserAddress();

    // 'blockedAddresses' CMP field setter
    public abstract void setBlockedAddresses(Address[] value);
    // 'blockedAddresses' CMP field getter
    public abstract Address[] getBlockedAddresses();

    // 'backupAddress' CMP field setter
    public abstract void setBackupAddress(Address value);
    // 'backupAddress' CMP field getter
    public abstract Address getBackupAddress();

    // 'voicemailState' CMP field setter
    public abstract void setVoicemailState(boolean value);
    // 'voicemailState' CMP field getter
    public abstract boolean getVoicemailState();
}
```

Profile abstract class implements custom logic to handle profile data and JSLEE callback methods for management operations. This class becomes part of profile object implementation. It allows to control some aspects of profile and its data.

Call Controller2 profile abstract class performs following operations:

- initialize profile with defailt values
- verify data constrains

It is defined as follows:

```
import javax.slee.Address;
import javax.slee.AddressPlan;
import javax.slee.CreateException;
import javax.slee.profile.Profile;
import javax.slee.profile.ProfileContext;

import javax.slee.profile.ProfileVerificationException;


public abstract class CallControlProfileManagementImpl implements Profile,
        CallControlProfileCMP {

    private ProfileContext profileCtx;

    /**
     * Initialize the profile with its default values.
     */
    public void profileInitialize() {
        setUserAddress(null);
        setBlockedAddresses(null);
        setBackupAddress(null);
        setVoicemailState(false);
    }

    public void profileLoad() {
    }

    public void profileStore() {
    }

    /**
     * Verify the profile's CMP field settings.
     *
     * @throws ProfileVerificationException
     *             if any CMP field contains an invalid value
     */
    public void profileVerify() throws ProfileVerificationException {
        // Verify Called User Address
```

```
        Address address = getUserAddress();
        if (address != null)
            verifyAddress(address);
        // Verify Blocked Addresses
        Address[] blockedAddresses = getBlockedAddresses();
        if (blockedAddresses != null) {
            for (int i = 0; i < blockedAddresses.length; i++) {
                if (blockedAddresses[i] != null)
                    verifyAddress(blockedAddresses[i]);
            }
        }
        // Verify Backup Address
        Address backupAddress = getBackupAddress();
        if (backupAddress != null)
            verifyAddress(backupAddress);
    }

    public void verifyAddress(Address address)
            throws ProfileVerificationException {
        // Check address plan
        if (address.getAddressPlan() != AddressPlan.SIP)
            throw new ProfileVerificationException("Address \"" + address
                    + "\" is not a SIP address");
        // Check URI scheme - must be sip: or sips:
        String uri = address.getAddressString().toLowerCase();
        if (!(uri.startsWith("sip:") || uri.startsWith("sips:")))
            throw new ProfileVerificationException("Address \"" + address
                    + "\" is not a SIP address");
    }

    public void profileActivate() {


    }

    public void profilePassivate() {


    }

    public void profilePostCreate() throws CreateException {


    }
```

```
    public void profileRemove() {

    }

    public void setProfileContext(ProfileContext ctx) {
        this.profileCtx = ctx;
    }

    public void unsetProfileContext() {
        this.profileCtx = null;
    }

}
```

4.4. Blocking Service Source

Blocking service is simplest of services in Call Controller2

It processes incoming SIP `INVITE` request and based on callee and caller denies or allows call to happen.

4.4.1. Service root

Root class of this service is `org.mobicents.slee.examples.callcontrol.blocking.CallBlockingSbb`

4.4.2. Events handlers

Call blocking service receives only one event: SIP INVITE request. Its responsibility is:

- extract callee AOR
- determine list of blocked users for callees AOR
- terminate call if caller AOR is blocked and set ACI variable to indicate that call has been handled

It is defined as follows:

```
    public void onInvite(javax.sip.RequestEvent event,
            CallBlockingSbbActivityContextInterface localAci) {
        Request request = event.getRequest();

        try {
```

```
            localAci.detach(this.getSbbLocalObject());

            FromHeader fromHeader = (FromHeader) request.getHeader(FromHeader.NAME);
            ToHeader toHeader = (ToHeader) request.getHeader(ToHeader.NAME);
            // From URI
            URI fromURI = fromHeader.getAddress().getURI();
            // To URI
            URI toURI = toHeader.getAddress().getURI();
            // In the Profile Table the port is not used
            ((SipURI)fromURI).removePort();
            ((SipURI)toURI).removePort();

            ArrayList targets = getBlockedArrayList(toURI.toString());

            if (targets != null) {
               // Cheking whether the caller is blocked by the called user
               for (int i = 0; i < targets.size(); i++) {
                  if ((targets.get(i).toString()).equalsIgnoreCase(fromURI.toString())) {
                     log.info("########## BLOCKING ADDRESS: " + targets.get(i));
                     log.info("########## BLOCKING FOR URI: " + toURI);
                     localAci.setFilteredByMe(true);
                     // Notifiying the client that the INVITE has been blocked
                     ServerTransaction stBlocking = (ServerTransaction) localAci.getActivity();
                     Response blockingResponse = getMessageFactory().createResponse(
                           Response.FORBIDDEN, request);
                     stBlocking.sendResponse(blockingResponse);
                  }
               }
            }

         } catch (TransactionRequiredLocalException e) {
            log.error(e.getMessage(), e);
         } catch (SLEEException e) {
            log.error(e.getMessage(), e);
         } catch (ParseException e) {
            log.error(e.getMessage(), e);
         } catch (SipException e) {
            log.error(e.getMessage(), e);
         } catch (InvalidArgumentException e) {
            log.error(e.getMessage(), e);
         }
      }
```

4.4.3. User profile access

Call Blocking SBB accesses examples profile in order to determine if callee has defined caller as blocked.

Profile is accessed in following way:

```
public abstract class CallBlockingSbb extends
    SubscriptionProfileSbb implements   javax.slee.Sbb
{

  ...
   /**
    * Attempt to find a list of Blocked Addresses (SIP URIs), but the method
    * returns null if the called user (sipAddress) does not block to any user.
    */
   private ArrayList getBlockedArrayList(String sipAddress) {
      //sipAddress is AOR: sip:newbie@mobicents.org
      ArrayList uris = null;
      CallControlProfileCMP profile = super.lookup(new Address(AddressPlan.SIP,
            sipAddress));
      if (profile != null) {
         Address[] addresses = profile.getBlockedAddresses();

         if (addresses != null) {
            uris = new ArrayList(addresses.length);

            for (int i = 0; i < addresses.length; i++) {
               String address = addresses[i].getAddressString();

               try {
                  SipURI uri = (SipURI) getAddressFactory().createURI(address);
                  uris.add(uri);

               } catch (ParseException e) {
                  log.error(e.getMessage(), e);
               }
            }
         }
      }

      return uris;
   }
```

```
}
```

4.4.4. Call Blocking SBB descriptor

Descriptor contains following definitions:

- sbb ID
- profile reference by profile ID
- sbb abstract class
- custom ACI
- ACI variable alias
- event handler with initial event selector
- resource adaptor binding

```
<sbb>
  <description />
  <sbb-name>CallBlockingSbb</sbb-name>
  <sbb-vendor>org.mobicents</sbb-vendor>
  <sbb-version>0.1</sbb-version>


  <profile-spec-ref>
    <profile-spec-name>CallControlProfileCMP</profile-spec-name>
    <profile-spec-vendor>org.mobicents</profile-spec-vendor>
    <profile-spec-version>0.1</profile-spec-version>
    <profile-spec-alias>CallControlProfile</profile-spec-alias>
  </profile-spec-ref>

  <sbb-classes>
    <sbb-abstract-class>
      <sbb-abstract-class-name>
        org.mobicents.slee.examples.callcontrol.blocking.CallBlockingSbb
      </sbb-abstract-class-name>
    </sbb-abstract-class>
    <sbb-activity-context-interface>
      <sbb-activity-context-interface-name>
```

```
org.mobicents.slee.examples.callcontrol.blocking.CallBlockingSbbActivityContextInterface
            </sbb-activity-context-interface-name>
        </sbb-activity-context-interface>
    </sbb-classes>



    <event event-direction="Receive" initial-event="True">
        <event-name>Invite</event-name>
        <event-type-ref>
            <event-type-name>javax.sip.message.Request.INVITE</event-type-name>
            <event-type-vendor>net.java.slee</event-type-vendor>
            <event-type-version>1.2</event-type-version>
        </event-type-ref>
        <initial-event-selector-method-name>callIDSelect</initial-event-selector-method-name>
    </event>

    <activity-context-attribute-alias>
        <attribute-alias-name>inviteFilteredByCallBlocking</attribute-alias-name>
        <sbb-activity-context-attribute-name>filteredByMe</sbb-activity-context-attribute-name>
    </activity-context-attribute-alias>

    <resource-adaptor-type-binding>
        <resource-adaptor-type-ref>
            <resource-adaptor-type-name>JAIN SIP</resource-adaptor-type-name>
            <resource-adaptor-type-vendor>javax.sip</resource-adaptor-type-vendor>
            <resource-adaptor-type-version>1.2</resource-adaptor-type-version>
        </resource-adaptor-type-ref>
          <activity-context-interface-factory-name>slee/resources/jainsip/1.2/acifactory</activity-
context-interface-factory-name>
        <resource-adaptor-entity-binding>
          <resource-adaptor-object-name>slee/resources/jainsip/1.2/provider</resource-adaptor-
object-name>
            <resource-adaptor-entity-link>SipRA</resource-adaptor-entity-link>
        </resource-adaptor-entity-binding>
    </resource-adaptor-type-binding>
  </sbb>
```

4.5. Forwarding Service Source

Forwarding service is simple illustration of call forwarding logic.

It processes incoming SIP `INVITE` request and based on callee availability permits call to proceed or redirects to backup address if it exists. Its a bit more complicated than *Section 4.4, "Blocking Service Source"* service as it:

- uses location service to determine availability of user
- uses proxy service to rely messages to proper target

4.5.1. Service root

Root class of this service is `org.mobicents.slee.examples.callcontrol.forwarding.CallForwardingSbb`

4.5.2. Events handlers

`CallForwardingSbb` performs all its tasks in `INVITE` event handler. That is:

- check if message has been already processed and possibly mark it as processed for services lower in chain
- extract user AOR and determine if user is online, if so use proxy service to deliver request
- if user is not online, attempt forward

Event handler code look as follows:

```
    public void onInvite(javax.sip.RequestEvent event
        , CallForwardingSbbActivityContextInterface localAci) {
        Request request;

        try {
            localAci.detach(this.getSbbLocalObject());

            if (localAci.getFilteredByAncestor()) {
                    log.info("########## CALL FORWARDING SBB: FILTERED BY ANCESTOR
##########");
                // Next in chain has to know that someone is looking after
                // message
                localAci.setFilteredByMe(true);
                // If it was not set, every change in the chain of services will
                // extort source change in service lower in chain...
                return;
            }
```

```
        request = event.getRequest();
        // ToHeader toHeader = (ToHeader) request.getHeader(ToHeader.NAME);
        // URI toURI = toHeader.getAddress().getURI();
        URI toURI = event.getRequest().getRequestURI();
        URI contactURI = isUserAvailable(toURI);
        if (contactURI != null) {
            // USER IS AVAILABLE
            localAci.setFilteredByMe(true);
            log.info("########## User " + toURI + " is available with contact " + contactURI);

            // Create proxy child SBB
            ChildRelation ProxyRelation = getJainSipProxySbb();
            SbbLocalObject ProxyChild = ProxyRelation.create();
            // Attach ProxyChild to the activity
            // Event router will pass this event to child SBB,
            // which in this case is the Proxy SBB. It will in turn proxy
            // the request to the callee.
            localAci.attach(ProxyChild);

            return;
        } else {
            log.info("########## User " + toURI + " is not available, not forwarding");
        }
    } catch (SipSendErrorResponseException e) {
        log.error(e.getMessage(), e);
    } catch (CreateException e) {
        log.error(e.getMessage(), e);
    }

    // IF WE GOT HERE IT MEANS THAT USER IS NOT AVAILABLE AND SBB HIGHER IN
    // CHAIN DID NOT FILTER INVITE.
    // WE HAVE TO FIND NEW ADDRESS... OR LEAVE INVITE TO BE PROCESSED BY
    // NEXT SBB IN CHAIN.
    Address add = forwardCall(event, localAci);

    if (add != null) {
        // INVITE WAS FORWARDED
        // let the next service in the chain know that the event was
        // processed here.
        localAci.setFilteredByMe(true);
    }

    // LET NEXT CHAINED SBB TAKE CARE OF INVITE.
```

```
        return;
    }
```

4.5.3. User status check

`CallForwardingSbb` depends on location service to check if user is online. Example checks entries in `registrar` for callees AOR. Registrar stores bindinings in form of mapping: AOR - {ContactAddress-BindingData,...}

Registrar entries are inspected in following way:

```
    private URI isUserAvailable(URI uri) throws SipSendErrorResponseException {
        String addressOfRecord = uri.toString();
        URI target = null;
        Map bindings = null;

        try {
            bindings = getLocationSbb().getBindings(addressOfRecord);

        } catch (LocationServiceException e) {
            log.error(e.getMessage(), e);
        } catch (TransactionRequiredLocalException e) {
            log.error(e.getMessage(), e);
        } catch (SLEEException e) {
            log.error(e.getMessage(), e);
        } catch (CreateException e) {
            log.error(e.getMessage(), e);
        }

        if (bindings != null & !bindings.isEmpty()) {
            Iterator it = bindings.values().iterator();

            while (it.hasNext()) {
                RegistrationBinding binding = (RegistrationBinding) it.next();
                log.info("########## BINDINGS: " + binding);
                ContactHeader header = null;
                try {
                    header = getHeaderFactory().createContactHeader(
                        getAddressFactory().createAddress(binding.getContactAddress()));
                } catch (ParseException e) {
                    log.error(e.getMessage(), e);
                }
```

```
            log.info("########## CONTACT HEADER: " + header);

            if (header == null) { // entry expired
                continue; // see if there are any more contacts...
            }

            Address na = header.getAddress();
            log.info("isUserAvailable Address: " + na);
            target = na.getURI();
            break;
        }

        if (target == null) {
            log.error("findLocalTarget: No contacts for " + addressOfRecord + " found.");
            throw new SipSendErrorResponseException("User temporarily unavailable",
                    Response.TEMPORARILY_UNAVAILABLE);
        }
    }

    return target;
}
```

4.5.4. Call forwarding

Call is beeing forwarded in case user is not logged in - that is `registrar` does not have binding for user AOR.

Forward operation is performed by means of SIP 3XX response class. Example sends response with code 302 and backup address as contact. It is done as follows:

```
protected Address forwardCall(javax.sip.RequestEvent event, ActivityContextInterface ac) {
    Address toAddress = null;
    Request request = event.getRequest();
    try {
        // Checking if the called user has any backup address
        ToHeader toHeader = (ToHeader) request.getHeader(ToHeader.NAME);
        String toURI = toHeader.getAddress().getURI().toString();
        Address backupAddress = getBackupAddress(toURI);

        if (backupAddress != null) {
            // Checking whether the called user has any backup address.
            toAddress = getAddressFactory().createAddress(backupAddress.toString());
```

```
            // Notifying the caller that the call has to be redirected
            ServerTransaction st = (ServerTransaction) ac.getActivity();
            ContactHeader contactHeader = getHeaderFactory()
                .createContactHeader(toAddress);
            Response response = getMessageFactory().createResponse(
                Response.MOVED_TEMPORARILY, request);
            response.setHeader(contactHeader);
            st.sendResponse(response);
            log.info("########## REQUEST FORWARDED: " + contactHeader.toString());
            // The Request-URI of the new request uses the value
            // of the Contact header field in the response
        }
    } catch (ParseException e) {
        log.error(e.getMessage(), e);
    } catch (TransactionRequiredLocalException e) {
        log.error(e.getMessage(), e);
    } catch (SLEEException e) {
        log.error(e.getMessage(), e);
    } catch (SipException e) {
        log.error(e.getMessage(), e);
    } catch (InvalidArgumentException e) {
        log.error(e.getMessage(), e);
    }
    return toAddress;
}
```

4.5.5. User profile access

Call Forwarding SBB accesses examples profile in order to determine if callee has defined backup address. Its done in following way:

```
private Address getBackupAddress(String sipAddress) {
    Address backupAddress = null;
    CallControlProfileCMP profile = this.lookup(
        new javax.slee.Address(AddressPlan.SIP, sipAddress));

        if (profile != null) {
            javax.slee.Address address = profile.getBackupAddress();

            if (address != null) {
                try {
```

```
                backupAddress = getAddressFactory()
                    .createAddress(address.getAddressString());
                } catch (ParseException e) {
                log.error(e.getMessage(), e);
            }
        }
    }

    return backupAddress;
}
```

4.5.6. Call Forwarding SBB descriptor

Descriptor contains following definitions:

- sbb ID
- profile reference by profile ID
- sbb abstract class
- child relation methods definition
- custom ACI
- ACI variable alias
- event handler with initial event selector
- resource adaptor binding

```
<sbb>
    <description />
    <sbb-name>CallForwardingSbb</sbb-name>
    <sbb-vendor>org.mobicents</sbb-vendor>
    <sbb-version>0.1</sbb-version>

    <sbb-ref>
        <sbb-name>ProxySbb</sbb-name>
        <sbb-vendor>mobicents</sbb-vendor>
        <sbb-version>1.1</sbb-version>
        <sbb-alias>JainSipProxySbb</sbb-alias>
    </sbb-ref>
    <sbb-ref>
```

```
        <sbb-name>LocationSbb</sbb-name>
        <sbb-vendor>org.mobicents</sbb-vendor>
        <sbb-version>1.2</sbb-version>
        <sbb-alias>LocationSbb</sbb-alias>
      </sbb-ref>
      <profile-spec-ref>
        <profile-spec-name>CallControlProfileCMP</profile-spec-name>
        <profile-spec-vendor>org.mobicents</profile-spec-vendor>
        <profile-spec-version>0.1</profile-spec-version>
        <profile-spec-alias>CallControlProfile</profile-spec-alias>
      </profile-spec-ref>

      <sbb-classes>
        <sbb-abstract-class>
          <sbb-abstract-class-name>
            org.mobicents.slee.examples.callcontrol.forwarding.CallForwardingSbb
          </sbb-abstract-class-name>

          <cmp-field>
            <cmp-field-name>locationSbbCMP</cmp-field-name>
          </cmp-field>

          <get-child-relation-method>
            <sbb-alias-ref>JainSipProxySbb</sbb-alias-ref>
            <get-child-relation-method-name>
              getJainSipProxySbb
            </get-child-relation-method-name>
            <default-priority>0</default-priority>
          </get-child-relation-method>
          <get-child-relation-method>
            <sbb-alias-ref>LocationSbb</sbb-alias-ref>
            <get-child-relation-method-name>
              getLocationSbbChildRelation
            </get-child-relation-method-name>
            <default-priority>0</default-priority>
          </get-child-relation-method>
        </sbb-abstract-class>
        <sbb-activity-context-interface>
          <sbb-activity-context-interface-name>

org.mobicents.slee.examples.callcontrol.forwarding.CallForwardingSbbActivityContextInterface
          </sbb-activity-context-interface-name>
        </sbb-activity-context-interface>
      </sbb-classes>
```

```
<address-profile-spec-alias-ref>
   CallControlProfile
</address-profile-spec-alias-ref>


<event event-direction="Receive" initial-event="True">
   <event-name>Invite</event-name>
   <event-type-ref>
         <event-type-name>javax.sip.message.Request.INVITE</event-type-name>
         <event-type-vendor>net.java.slee</event-type-vendor>
         <event-type-version>1.2</event-type-version>
   </event-type-ref>
   <initial-event-selector-method-name>
      callIDSelect
   </initial-event-selector-method-name>
</event>


<activity-context-attribute-alias>
   <attribute-alias-name>
      inviteFilteredByCallBlocking
   </attribute-alias-name>
   <sbb-activity-context-attribute-name>
      filteredByAncestor
   </sbb-activity-context-attribute-name>
</activity-context-attribute-alias>
<activity-context-attribute-alias>
   <attribute-alias-name>
      inviteFilteredByCallForwarding
   </attribute-alias-name>
   <sbb-activity-context-attribute-name>
      filteredByMe
   </sbb-activity-context-attribute-name>
</activity-context-attribute-alias>

<resource-adaptor-type-binding>
   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>
         JAIN SIP
      </resource-adaptor-type-name>
      <resource-adaptor-type-vendor>
         javax.sip
      </resource-adaptor-type-vendor>
      <resource-adaptor-type-version>
```

```
            1.2
          </resource-adaptor-type-version>
        </resource-adaptor-type-ref>
        <activity-context-interface-factory-name>
          slee/resources/jainsip/1.2/acifactory
        </activity-context-interface-factory-name>
        <resource-adaptor-entity-binding>
          <resource-adaptor-object-name>
            slee/resources/jainsip/1.2/provider
          </resource-adaptor-object-name>
          <resource-adaptor-entity-link>
            SipRA
          </resource-adaptor-entity-link>
        </resource-adaptor-entity-binding>
      </resource-adaptor-type-binding>
    </sbb>
```

4.6. Voice Mail Service Source

Voice mail service illustrates usage of MGCP protocol to control `Media Server`. It is capable of following:

- setting up media session
- receiving event notification from media server
- play audio
- record audio

4.6.1. Service root

Root class of this service is `org.mobicents.slee.examples.callcontrol.forwarding.VoiceMailSbb`

4.6.2. SIP Event handlers

SIP are more complicated as they more complicated tasks including MGCP signaling and SDP negotiation.

`VoiceMailSbb` defines two handlers for SIP.

4.6.2.1. INVITE

INVITE event handler performs following tasks:

- check if event has already been processed
- determine if messages will be recorded or played
- create dialog
- determine if target user has enabled voice mail(subscribed), if not, terminate call
- extract SDP from message and send MGCP CRCX reques. Request is sent with SDP to PR endpoint.

Handler is defined as follows:

```
public void onInvite(javax.sip.RequestEvent event,
        VoiceMailSbbActivityContextInterface localAci) {
    Response response;
    log.info("########## VOICE MAIL SBB: INVITE ##########");
    // Request
    Request request = event.getRequest();
    // Setting Request
    this.setInviteRequest(request);

    // Server Transaction
    ServerTransaction st = event.getServerTransaction();

    try {

        if (localAci.getFilteredByAncestor()) {
            log
                    .info("########## VOICE MAIL SBB: FILTERED BY ANCESTOR ##########");
            return;
        }

        // if we are calling to vmail this means we want to check our mail
        // box
        // sameUser = true
        boolean sameUser = sameUser(event);
        URI uri;

        if (sameUser) {
            // The user is the caller
            FromHeader fromHeader = (FromHeader) request
                    .getHeader(FromHeader.NAME);
            uri = fromHeader.getAddress().getURI();
        } else {
```

```
        // The user is the callee - we are calling someone else
        ToHeader toHeader = (ToHeader) request.getHeader(ToHeader.NAME);
        uri = toHeader.getAddress().getURI();
    }
    // In the Profile Table the port is not used
    ((SipURI) uri).removePort();

    // Responding to the user
    // To know whether the user has the Voice mail service enabled
    boolean isSubscriber = isSubscriber(uri.toString());

    if (isSubscriber) {

        // Formalities of sip, so we dont get retrans
        // Attaching to SIP Dialog activity
        Dialog dial = getSipFactoryProvider().getNewDialog(
                (Transaction) st);
        ActivityContextInterface dialogAci = sipACIF
                .getActivityContextInterface((DialogActivity) dial);

        // attach this SBB object to the Dialog activity to receive
        // subsequent events on this Dialog
        dialogAci.attach(this.getSbbLocalObject());

        // Notify caller that we're TRYING to reach voice mail. Just a
        // formality, we know we can go further than TRYING at this
        // point
        response = getMessageFactory().createResponse(Response.TRYING,
                request);
        st.sendResponse(response);

        // RINGING. Another formality of the SIP protocol.
        response = getMessageFactory().createResponse(Response.RINGING,
                request);
        st.sendResponse(response);

        String sdp = new String(event.getRequest().getRawContent());

        CallIdentifier callID = this.mgcpProvider
                .getUniqueCallIdentifier();
        // this is not requiered, but to be good MGCP citizen we will
        // obey mgcp call id rule.
        setCallIdentifier(callID);
```

```
            EndpointIdentifier endpointID = new EndpointIdentifier(
                    PRE_ENDPOINT_NAME, mmsBindAddress + ":"
                            + MGCP_PEER_PORT);
            CreateConnection createConnection = new CreateConnection(this,
                    callID, endpointID, ConnectionMode.SendRecv);
            try {
                createConnection
                        .setRemoteConnectionDescriptor(new ConnectionDescriptor(
                                sdp));
            } catch (ConflictingParameterException e) {
                e.printStackTrace();
            }
            int txID = mgcpProvider.getUniqueTransactionHandler();
            createConnection.setTransactionHandle(txID);

            MgcpConnectionActivity connectionActivity = null;
            try {
                connectionActivity = mgcpProvider.getConnectionActivity(
                        txID, endpointID);
                ActivityContextInterface epnAci = mgcpActivityContestInterfaceFactory
                        .getActivityContextInterface(connectionActivity);
                epnAci.attach(getSbbContext().getSbbLocalObject());

            } catch (FactoryException ex) {
                ex.printStackTrace();
            } catch (NullPointerException ex) {
                ex.printStackTrace();
            } catch (UnrecognizedActivityException ex) {
                ex.printStackTrace();
            }

            mgcpProvider
                    .sendMgcpEvents(new JainMgcpEvent[] { createConnection });
            log
                    .info("########## VOICE MAIL AVAILABLE FOR USER: sent PR CRCX request
##########");
        } else {
            // Voice Mail service disabled
            response = getMessageFactory().createResponse(
                    Response.TEMPORARILY_UNAVAILABLE, request);
            log.info("########## NO VOICE MAIL AVAILABLE FOR USER: "
                    + uri.toString());
            st.sendResponse(response);
        }
```

```
    } catch (TransactionRequiredLocalException e) {
        log.error(e.getMessage(), e);
    } catch (SLEEException e) {
        log.error(e.getMessage(), e);
    } catch (ParseException e) {
        log.error(e.getMessage(), e);
    } catch (SipException e) {
        log.error(e.getMessage(), e);
    } catch (InvalidArgumentException e) {
        log.error(e.getMessage(), e);
    } catch (NullPointerException e) {
        log.error(e.getMessage(), e);
    }
}
```

4.6.2.2. BYE

This event handler is different than `INVITE`. It is triggered by event fired as in-Dialog event, that is , its different class of events - indicated by different name, please refer to SIP `Resource Adaptor Guide`. `BYE` event handler performs following tasks:

- tear down media path
- flush media buffer
- free service

Handler is defined as follows:

```
public void onByeEvent(RequestEvent event, ActivityContextInterface aci) {
    log.info("########## VOICE MAIL SBB: BYE ##########");
    try {

        releaseState();

        // Sending the OK Response to the BYE Request received.
        byeRequestOkResponse(event);

    } catch (FactoryException e) {
        log.error(e.getMessage(), e);
    } catch (NullPointerException e) {
        log.error(e.getMessage(), e);
```

```
    }
  }
```

4.6.3. MGCP Event handlers

For details on MGCP protocol please refer to:

- *RFC3435* [http://tools.ietf.org/html/rfc3435]
- `mgcp-demo` documentation
- MGCP RA documentation

4.6.3.1. Create Connection (CRCX) Response

This event is received as answer to CRCX request. `Voice Mail` receives this in two cases:

- media server acknowledges creation of connection to PR endpoint
- media server acknowledges creation of connection between PR and IVR endpoints

It performs following task:

- disconnect UA in case of failure and clear state of media server.
- send OK(200) response to UA with SDP of PR endpoint connection please refer to *Section 3.5, "Voice Mail Service"* for overview of arrangement.
- create connection between PR and IVR endpoints
- start greeting playback if media path is estabilished

Handler is defined as follows:

```
public void onCreateConnectionResponse(CreateConnectionResponse event,
      ActivityContextInterface aci) {
   log.info("Receive CRCX response: " + event);

   ReturnCode status = event.getReturnCode();

   switch (status.getValue()) {

   case ReturnCode.TRANSACTION_EXECUTED_NORMALLY:
      log.info("Connection created properly.");
```

```
        break;
    default:
        ReturnCode rc = event.getReturnCode();
        log.error("CRCX failed. Value = " + rc.getValue() + " Comment = "
                + rc.getComment());

        sendServerError("Failed to create connection, code: "
                + event.getReturnCode(), Response.SERVER_INTERNAL_ERROR);

        return;
    }

    boolean startMailMedia = false;
    if (event.getSecondEndpointIdentifier() == null) {
        // this is response for PR creation
        // we have one connection activity, lets send another crcx

        // send OK with sdp
        DialogActivity da = getDialogActivity();
        ServerTransaction txn = getServerTransaction();
        if (txn == null) {
            log.error("SIP activity lost, close RTP connection");
            releaseState();
            return;
        }

        Request request = txn.getRequest();

        ContentTypeHeader contentType = null;
        try {
            contentType = getHeaderFactory().createContentTypeHeader("application", "sdp");
        } catch (ParseException ex) {
        }

        String localAddress = getSipFactoryProvider().getListeningPoints()[0].getIPAddress();
        int localPort = getSipFactoryProvider().getListeningPoints()[0].getPort();

        javax.sip.address.Address contactAddress = null;
        try {
            contactAddress = getAddressFactory().createAddress(
                "sip:" + localAddress + ":" + localPort);
        } catch (ParseException ex) {
            log.error(ex.getMessage(), ex);
        }
```

```
ContactHeader contact = getHeaderFactory()
    .createContactHeader(contactAddress);

Response response = null;
try {
    response = getMessageFactory().createResponse(
        Response.OK, request, contentType,
        event.getLocalConnectionDescriptor().toString().getBytes());
} catch (ParseException ex) {
}

response.setHeader(contact);
try {
    txn.sendResponse(response);
} catch (InvalidArgumentException ex) {
    log.error(ex.getMessage(), ex);
} catch (SipException ex) {
    log.error(ex.getMessage(), ex);
}
EndpointIdentifier endpointID = new EndpointIdentifier(
        IVR_ENDPOINT_NAME, mmsBindAddress + ":" + MGCP_PEER_PORT);
CreateConnection createConnection = new CreateConnection(this,
        getCallIdentifier(), endpointID, ConnectionMode.SendRecv);

int txID = mgcpProvider.getUniqueTransactionHandler();
createConnection.setTransactionHandle(txID);

// now set other end
try {
    createConnection.setSecondEndpointIdentifier(event
            .getSpecificEndpointIdentifier());
} catch (ConflictingParameterException e) {
    e.printStackTrace();
}

MgcpConnectionActivity connectionActivity = null;
try {
    connectionActivity = mgcpProvider.getConnectionActivity(txID,
            endpointID);
    ActivityContextInterface epnAci = mgcpActivityContestInterfaceFactory
            .getActivityContextInterface(connectionActivity);
    epnAci.attach(getSbbContext().getSbbLocalObject());
    // epnAci.attach(getParentCmp());
} catch (FactoryException ex) {
```

```
            ex.printStackTrace();
        } catch (NullPointerException ex) {
            ex.printStackTrace();
        } catch (UnrecognizedActivityException ex) {
            ex.printStackTrace();
        }

        mgcpProvider
                .sendMgcpEvents(new JainMgcpEvent[] { createConnection });

    } else {
        // this is last
        startMailMedia = true;

    }

    EndpointIdentifier eid = event.getSpecificEndpointIdentifier();
    log.info("Creating endpoint activity on: " + eid);
    MgcpEndpointActivity eActivity = mgcpProvider.getEndpointActivity(eid);
    ActivityContextInterface eAci = mgcpActivityContestInterfaceFactory
            .getActivityContextInterface(eActivity);
    eAci.attach(this.getSbbContext().getSbbLocalObject());

    if (startMailMedia) {
        startMailMedia();
    }

}
```

4.6.3.2. Notification Request Response

This event is received as response to notification request. It indicates result of signal request sent to media server.

It disconnect UA in case of failure.

Handler is defined as follows:

```
public void onNotificationRequestResponse(
    NotificationRequestResponse event, ActivityContextInterface aci) {
```

```
    ReturnCode status = event.getReturnCode();

    switch (status.getValue()) {
    case ReturnCode.TRANSACTION_EXECUTED_NORMALLY:
                    log.info("########## VOICE MAIL SBB: RQNT executed properly. TXID:
 "+event.getTransactionHandle()+" ##########");
        break;
    default:
        ReturnCode rc = event.getReturnCode();
                log.info("########## VOICE MAIL SBB: RQNT failed, terminating call. TXID:
 "+event.getTransactionHandle()+" ##########");
        sendByeRequest();

        break;
    }

}
```

4.6.3.3. Notify Request

This event is received as result of requested notification - `Voice Mail` requests to be notified on certain conditions.

Depending on observed event type, handler performs different action.

Handler is defined as follows:

```
public void onNotifyRequest(Notify event, ActivityContextInterface aci) {


    NotifyResponse response = new NotifyResponse(event.getSource(),
            ReturnCode.Transaction_Executed_Normally);
    response.setTransactionHandle(event.getTransactionHandle());
   log.info("########## VOICE MAIL SBB: Sending Notify response["+response+"] to ["+event+"]
["+event.getTransactionHandle()+"] ["+response.getTransactionHandle()+"]##########");

    mgcpProvider.sendMgcpEvents(new JainMgcpEvent[] { response });


    EventName[] observedEvents = event.getObservedEvents();

    for (EventName observedEvent : observedEvents) {
```

```
        switch (observedEvent.getEventIdentifier().intValue()) {
        case MgcpEvent.REPORT_ON_COMPLETION:
                                log.info("########## VOICE MAIL SBB: Signal completed, event
identifier["+observedEvent.getEventIdentifier()+"] ##########");

            if(observedEvent.getEventIdentifier().toString().equals("oc"))
            {
               onAnnouncementComplete();
            }

            break;
        case MgcpEvent.REPORT_FAILURE:
                                    log.info("########## VOICE MAIL SBB: Signal failed, event
identifier["+observedEvent.getEventIdentifier()+"] ##########");
            //releaseState();
            sendByeRequest();
            break;

        case MgcpEvent.DTMF_1:
            this.checkDtmfDigit("1");
            break;
        case MgcpEvent.DTMF_7:
            this.checkDtmfDigit("7");
            break;
        case MgcpEvent.DTMF_9:
            this.checkDtmfDigit("9");
            break;

        default:
                        log.info("########## VOICE MAIL SBB: Notify on unknown event, event

##########");
            break;
        }

    }

}
```

4.6.4. MGCP signals

`VoiceMailSbb` sends request to media server in order to trigger media play or record. It also requests to be notified on certain events:

- announcement completed
- DTMF digit has been pressed

It sends MGCP `Notification Request` with signals(play or record) to be applied and events to be detected.

Notification Request is built as follows:

```
public void sendRQNT(String audioFileUrl, boolean record, boolean detectDtmf) {
    MgcpEndpointActivity endpointActivity = getEndpointActivity("IVR");

    if (endpointActivity == null) {
        // bad practice
        throw new RuntimeException("There is no IVR endpoint activity");
    }
    MgcpConnectionActivity connectionActivity = getConnectionActivity(endpointActivity
            .getEndpointIdentifier());
    if (connectionActivity == null) {
        // bad practice
        throw new RuntimeException(
                "There is no IVR connection activity");
    }
    EndpointIdentifier endpointID = endpointActivity
            .getEndpointIdentifier();
    ConnectionIdentifier connectionID = new ConnectionIdentifier(
            connectionActivity.getConnectionIdentifier());
    NotificationRequest notificationRequest = new NotificationRequest(this,
            endpointID, mgcpProvider.getUniqueRequestIdentifier());
   RequestedAction[] actions = new RequestedAction[] { RequestedAction.NotifyImmediately };


    if (audioFileUrl != null) {
        EventName[] signalRequests = null;
        if (!record) {

            signalRequests = new EventName[] { new EventName(
                    PackageName.Announcement, MgcpEvent.ann
                            .withParm(audioFileUrl),connectionID) };
        } else {
            signalRequests = new EventName[] { new EventName(AUPackage.AU,
                    AUMgcpEvent.aupr.withParm(audioFileUrl), connectionID) };
        }
```

```
        notificationRequest.setSignalRequests(signalRequests);

        //add notification, in case dtmf part is not included
        RequestedEvent[] requestedEvents = {
                new RequestedEvent(new EventName(PackageName.Announcement
                    , MgcpEvent.oc,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Announcemen
                    t, MgcpEvent.of,connectionID), actions),
                 };

        notificationRequest.setRequestedEvents(requestedEvents);

    }

    if (detectDtmf) {



        // This has to be present, since MGCP states that new RQNT erases
        // previous set.
        RequestedEvent[] requestedEvents = {
                new RequestedEvent(new EventName(PackageName.Announcement
                    , MgcpEvent.oc,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Announcement
                    , MgcpEvent.of,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf0,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf1,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf2,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf3,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf4,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf5,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf6,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf7,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf8,connectionID), actions),
```

```
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmf9,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfA,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfB,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfC,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfD,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfStar,connectionID), actions),
                new RequestedEvent(new EventName(PackageName.Dtmf,
                        MgcpEvent.dtmfHash,connectionID), actions) };

            notificationRequest.setRequestedEvents(requestedEvents);
        }
        notificationRequest.setTransactionHandle(mgcpProvider
                .getUniqueTransactionHandler());

        NotifiedEntity notifiedEntity = new NotifiedEntity(JBOSS_BIND_ADDRESS,
                JBOSS_BIND_ADDRESS, MGCP_PORT);
        notificationRequest.setNotifiedEntity(notifiedEntity);

        // we can send empty RQNT, that is clean all req.
        mgcpProvider
                .sendMgcpEvents(new JainMgcpEvent[] { notificationRequest });

        log.info(" NotificationRequest sent: \n" + notificationRequest);
    }
```

4.6.5. Voice Mail profile access

Voice Mail SBB accesses examples profile in order to determine if callee has subscription to voice mail service.

Its done in following way:

```
    private boolean isSubscriber(String sipAddress) {
        boolean state = false;
        CallControlProfileCMP profile = lookup(new Address(AddressPlan.SIP,
```

```
            sipAddress));

    if (profile != null) {
        state = profile.getVoicemailState();
    }

    return state;
}
```

4.6.6. Voice Mail SBB descriptor

Descriptor contains following definitions:

- sbb ID
- profile reference by profile ID
- sbb abstract class
- cmp fields definition
- custom ACI
- ACI variable alias
- event handler for transactional INVITE
- event handler for in-Dialog BYE
- event handlers for MGCP events
- SBB environment entries
- resource adaptor bindings

```
<sbb>
  <description />
  <sbb-name>VoiceMailSbb</sbb-name>
  <sbb-vendor>org.mobicents</sbb-vendor>
  <sbb-version>0.1</sbb-version>

  <profile-spec-ref>
    <profile-spec-name>CallControlProfileCMP</profile-spec-name>
    <profile-spec-vendor>org.mobicents</profile-spec-vendor>
    <profile-spec-version>0.1</profile-spec-version>
```

```
        <profile-spec-alias>CallControlProfile</profile-spec-alias>
      </profile-spec-ref>
      <sbb-classes>
        <sbb-abstract-class>
          <sbb-abstract-class-name>
            org.mobicents.slee.examples.callcontrol.voicemail.VoiceMailSbb
          </sbb-abstract-class-name>
          <cmp-field>
            <cmp-field-name>inviteRequest</cmp-field-name>
          </cmp-field>
          <cmp-field>
            <cmp-field-name>callIdentifier</cmp-field-name>
          </cmp-field>
          <cmp-field>
            <cmp-field-name>sameUser</cmp-field-name>
          </cmp-field>
        </sbb-abstract-class>
        <sbb-activity-context-interface>
          <sbb-activity-context-interface-name>

org.mobicents.slee.examples.callcontrol.voicemail.VoiceMailSbbActivityContextInterface
          </sbb-activity-context-interface-name>
        </sbb-activity-context-interface>
      </sbb-classes>

      <event event-direction="Receive" initial-event="True">
        <event-name>Invite</event-name>
        <event-type-ref>
          <event-type-name>javax.sip.message.Request.INVITE</event-type-name>
          <event-type-vendor>net.java.slee</event-type-vendor>
          <event-type-version>1.2</event-type-version>
        </event-type-ref>
        <initial-event-selector-method-name>
          callIDSelect
        </initial-event-selector-method-name>
      </event>

      <event event-direction="Receive" initial-event="False">
        <event-name>ByeEvent</event-name>
        <event-type-ref>
          <event-type-name>javax.sip.Dialog.BYE</event-type-name>
          <event-type-vendor>net.java.slee</event-type-vendor>
          <event-type-version>1.2</event-type-version>
        </event-type-ref>
```

```
    </event>
    <!-- MGCP events  -->
    <event event-direction="Receive" initial-event="False">
      <event-name>NotificationRequestResponse</event-name>
      <event-type-ref>
        <event-type-name>
          net.java.slee.resource.mgcp.NOTIFICATION_REQUEST_RESPONSE
        </event-type-name>
        <event-type-vendor>net.java</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>

    <event event-direction="Receive" initial-event="False">
      <event-name>NotifyRequest</event-name>
      <event-type-ref>
        <event-type-name>
          net.java.slee.resource.mgcp.NOTIFY
        </event-type-name>
        <event-type-vendor>net.java</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
    <event event-direction="Receive" initial-event="False">
      <event-name>CreateConnectionResponse</event-name>
      <event-type-ref>
        <event-type-name>
          net.java.slee.resource.mgcp.CREATE_CONNECTION_RESPONSE
        </event-type-name>
        <event-type-vendor>net.java</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>


    <event event-direction="Receive" initial-event="False">
      <event-name>ActivityEndEvent</event-name>
      <event-type-ref>
        <event-type-name>javax.slee.ActivityEndEvent</event-type-name>
        <event-type-vendor>javax.slee</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
```

```
    <activity-context-attribute-alias>
      <attribute-alias-name>
        inviteFilteredByCallForwarding
      </attribute-alias-name>
      <sbb-activity-context-attribute-name>
        filteredByAncestor
      </sbb-activity-context-attribute-name>
    </activity-context-attribute-alias>
    <env-entry>
      <description>
        This is the path where the recorded files will reside.
        it is part of record/announce path. Full path is comined as follows:
        ${MOBICENTS_SLEE_EXAMPLE_CC2_RECORDINGS_HOME}/${filesRoute}.
      </description>
      <env-entry-name>filesRoute</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>${files.route}</env-entry-value>
    </env-entry>
    <env-entry>
      <description>This is the IP address of media server
       to which MGCP requests are sent</description>
      <env-entry-name>server.address</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>${server.address}</env-entry-value>
    </env-entry>
    <resource-adaptor-type-binding>
      <resource-adaptor-type-ref>
        <resource-adaptor-type-name>
          JAIN SIP
        </resource-adaptor-type-name>
        <resource-adaptor-type-vendor>
          javax.sip
        </resource-adaptor-type-vendor>
        <resource-adaptor-type-version>
          1.2
        </resource-adaptor-type-version>
      </resource-adaptor-type-ref>
      <activity-context-interface-factory-name>
        slee/resources/jainsip/1.2/acifactory
      </activity-context-interface-factory-name>
      <resource-adaptor-entity-binding>
        <resource-adaptor-object-name>
          slee/resources/jainsip/1.2/provider
        </resource-adaptor-object-name>
```

```
            <resource-adaptor-entity-link>
               SipRA
            </resource-adaptor-entity-link>
         </resource-adaptor-entity-binding>
      </resource-adaptor-type-binding>
      <resource-adaptor-type-binding>
         <resource-adaptor-type-ref>
            <resource-adaptor-type-name>
               jain-mgcp
            </resource-adaptor-type-name>
            <resource-adaptor-type-vendor>
               net.java
            </resource-adaptor-type-vendor>
            <resource-adaptor-type-version>
               2.0
            </resource-adaptor-type-version>
         </resource-adaptor-type-ref>
         <activity-context-interface-factory-name>
            slee/resources/jainmgcp/2.0/acifactory
         </activity-context-interface-factory-name>
         <resource-adaptor-entity-binding>
            <resource-adaptor-object-name>
               slee/resources/jainmgcp/2.0/provider
            </resource-adaptor-object-name>
            <resource-adaptor-entity-link>
               MGCPRA
            </resource-adaptor-entity-link>
         </resource-adaptor-entity-binding>
      </resource-adaptor-type-binding>
   </sbb>
```

Running the Example

The easiest way to run example is to:

- deploy example and dependencies(please follow: *Chapter 2, Setup*
- start media server
- set environment MOBICENTS_SLEE_EXAMPLE_CC2_RECORDINGS_HOME variable to point to absolute path to `media` directory of running media server. For instance: `g:/tmp/test/tools/media-server/media`
- start container

Setup two UAs, set proxy as ${jboss.bind.address}:5060 (should be 127.0.0.1:5060). To test:

- Blocking - make call either from "mobicents@127.0.0.1" or "hugo@127.0.0.1" to "torosvi@127.0.0.1"
- Forwarding - register UA as "torosvi@127.0.0.1" and make call from any non blocked addresses to "victor@127.0.0.1"
- Voice Mail - make call to "torosvi@127.0.0.1" while user is not registered, record message, register "torosvi@127.0.0.1" and call "vmail@127.0.0.1"

5.1. Configuration

Voice Mail SBB supports two environment variables which change how example behaves.

- server.address - changes adddress to which MGCP requests are sent
- filesRoute - changes prefix of directory where voice files are recorded

Traces and Alarms

6.1. Tracers

The example Application uses multiple JAIN SLEE 1.1 Tracer facility instances. Below is full list:

Table 6.1. Call Controller2 Tracer and Log Categories

Sbb	Tracer name	LOG4J category
SubscriptionProfileSbb	SubscriptionProfileSbb	javax.slee.SbbNotification[service= ServiceID[name= CallBlockingService ,vendor=org.mobicents, version=0.1], sbb=SbbID[name=CallBlockingSbb, vendor=mobicents, version=1.1]]. SubscriptionProfileSbb
SubscriptionProfileSbb	SubscriptionProfileSbb	javax.slee.SbbNotification[service= ServiceID[name= CallForwardingService,vendor=org.mobicents, version=0.1], sbb=SbbID[name=CallForwardingSbb, vendor=mobicents ,version=1.1]]. SubscriptionProfileSbb
SubscriptionProfileSbb	SubscriptionProfileSbb	javax.slee.SbbNotification[service= ServiceID[name= VoiceMailService, vendor=org.mobicents, version=0.1], sbb=SbbID[name=VoiceMailSbb, vendor=mobicents ,version=1.1]]. SubscriptionProfileSbb
CallBlockingSbb	CallBlockingSbb	javax.slee.SbbNotification[service= ServiceID[name= CallBlockingService, vendor=org.mobicents, version=0.1], sbb=SbbID[name=CallBlockingSbb, vendor=org.mobicents ,version=0.1]]. CallBlockingSbb
CallForwardingSbb	CallForwardingSbb	javax.slee.SbbNotification[service= ServiceID[name= CallForwardingService, vendor=org.mobicents, version=0.1], sbb=SbbID[name=CallForwardingSbb, vendor=org.mobicents ,version=0.1]]. CallForwardingSbb

Sbb	Tracer name	LOG4J category
VoiceMailSbb	VoiceMailSbb	javax.slee.SbbNotification[service= ServiceID[name=VoiceMailService, vendor=org.mobicents, version=0.1], sbb=SbbID[name=VoiceMailSbb, vendor=org.mobicents ,version=0.1]]. VoiceMailSbb



Important

Spaces where introduced in `LOG4J category` column values, to correctly render the table. Please remove them when using copy/paste.

6.2. Alarms

The example Application does not set JAIN SLEE Alarms.

Appendix A. Revision History

Revision History
Revision 1.0 Wed Feb 9 2010 BartoszBaranowski
Creation of the JBoss Communications JAIN SLEE Call Controller2 Example User Guide.

Index

F

feedback, viii