IMS Communication Service Manual

Conferencing API Prototype Developer's Guide

For Java Developers

Table of content

1. Introduction

This document provides a guide for developers on how to use the conferencing API to develop multimedia conferencing services.

In this section, we first introduce briefly the background information on conferencing in IP Multimedia Subsystem (IMS). We then present an overview of the conferencing API. The organization of the document will also be introduced.

1.1 Multimedia conferencing and floor control

Multimedia conferencing can be defined as active exchange of multimedia data among several parties. It enables a rich set of multimedia services such as multiparty gaming, audio/videoconferencing and distance learning.

There are three main components in multimedia conferencing: signaling, media handling and conference control. Signaling establishes, modifies and terminates sessions. Media handling performs media processing, trans-coding and mixing. Conference control includes policy control, participant management and floor control.

Floor control manages the shared resources of the conference, such as audio, video and text messages. The purpose of floor control is to prevent conflict and ensures an optimized use of resources.

Three entities are involved in floor control: floor participant, floor chair and Floor Control Server (FCS). A floor participant is an entity that can request a floor. A floor chair is an entity that manages floors (i.e. grant, deny or revoke the floor). The FCS is the entity that maintains floor(s) status (e.g. which floors exists, who the floor chairs are, who hold the floors). There maybe no floor chairs. In that case, the FCS uses a non-chair controlled floor algorithm. An example is the First Come First Serve (FCFS) algorithm. In this document, we only cover FCFS algorithm.

The standardization of conferencing is still ongoing. So far 3GPP has specified requirements, basic conferencing architecture, and some conference sequences in IMS.

Figure 1 summarizes the 3GPP IMS conferencing architecture [1]. In the architecture, Session Initiation Protocol (SIP [2]) is used as signaling protocol and the Real-time Transport Protocol (RTP [3]) as media transport protocol. In the figure, the Media Resource Function Controller (MRFC) /Application Server (AS) acts as a centralized conference server. It handles conference control and signaling. The Media Resource Function Processor (MRFP) provides all the media related functions such as mixing and trans-coding.

The MRFP may also contain an FCS functional entity. In that case, the Binary Floor Control Protocol (BFCP [4]) is used between the FCS and the participants. A User Equipment (UE) is a conference participant that has the required conferencing functionality in the end-users' terminals. The Call Session Control Functions (CSCFs) in the figure are entities that proxy or route SIP messages.

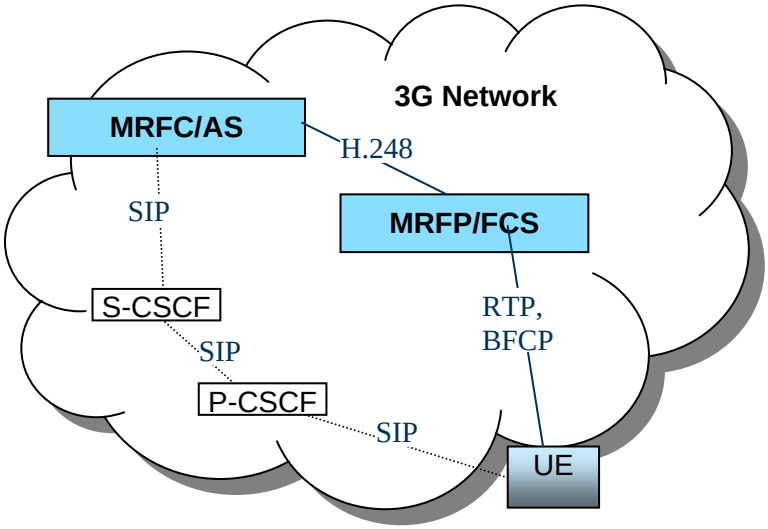

Figure 1: 3GPP conference architecture

Conferencing can be classified into three types: dial-out, dial-in and ad hoc. When conference participants are called by a centralized server, we call the conference dial-out conference. Unlike dial-out, dial-in specifies the conference that participants join by calling the conference server. Ad hoc conference is normally created by a participant and any other participants can join the conference. In this document, we only cover dial-out conference.

1.2 Conferencing API overview

As one part of the IMS Communication Service (CoSe) APIs, the conferencing API aims at opening up IMS conferencing capabilities. The API is at a high-level of abstraction and it facilitates the development of conferencing applications. In order to use this API, the developer does not need to be an expert in IMS domain. The following functionalities are provided in the current version of the API:

- Dial-out conferencing including:
    - Conference creation
    - Conference termination
    - Adding/removing participants
- FCFS based floor control with:
    - Different number of floor holders
    - Different number of floors
- Playing announcement to conference participants

1.3 How is this guide organized

In chapter 2, we present the overview architecture in which we introduce the relationship between the conference API and other APIs. We then describe the server side conferencing API that includes an introduction to the API, sequence diagrams and sample codes.

In chapter 3, we introduce a simple client side floor control API, which can be used with Ericsson IMS Client Platform (ICP).

2. Overall Architecture and Server Side API

In this chapter, we first present an overall architecture. Then we briefly introduce the JSR 309 implementation that we used for the media part of conferencing API implementation. After that we introduce the main interfaces of the server side conference API. In the last section, we describe how to use the API.

2.1 Overall architecture

Figure 2 presents the overall architecture that shows where the conference API is located. From the server side, there are three main components: the AS/MRFP, the FCS and the MRFP. The AS/MRFC hosts the emulated IMS network (i.e. via SDS), the conference application, and the conference API. The conference API is an IMS Communication Service (IMS CoSe) API. It relies on IMS Core API (which hides the complexity of JSR 289 SIP Servlet API) and a set of lower level floor control API, which provide basic session management control and floor control functionalities, respectively.
The communication between the AS and the MRFP is based on JSR 309 implementation. JSR 309 is a Media Server (MS) control API. It is a generic MS abstraction interface independent of MS control protocols, such as H.248 and SIP Media Server Control Markup Language (MSCML). It provides application developers multimedia capabilities such as mixing, interaction dialogs, and media playing and recording.

From the client side, the client application uses the client side floor control API, and it runs on ICP server.
In the figure, floor Control API in both client and server side exposes floor control capability in IMS network. The API can be used to manage the access to any shared resource in a conference (e.g. audio, video, slide bar presentation, and pointer). It allows applications to add floor to a conference, add/remove participants to/from an existing floor, and remove an existing floor from a conference.
The floor control API in the server side is used by the conference API as basis to provide high level floor control API.

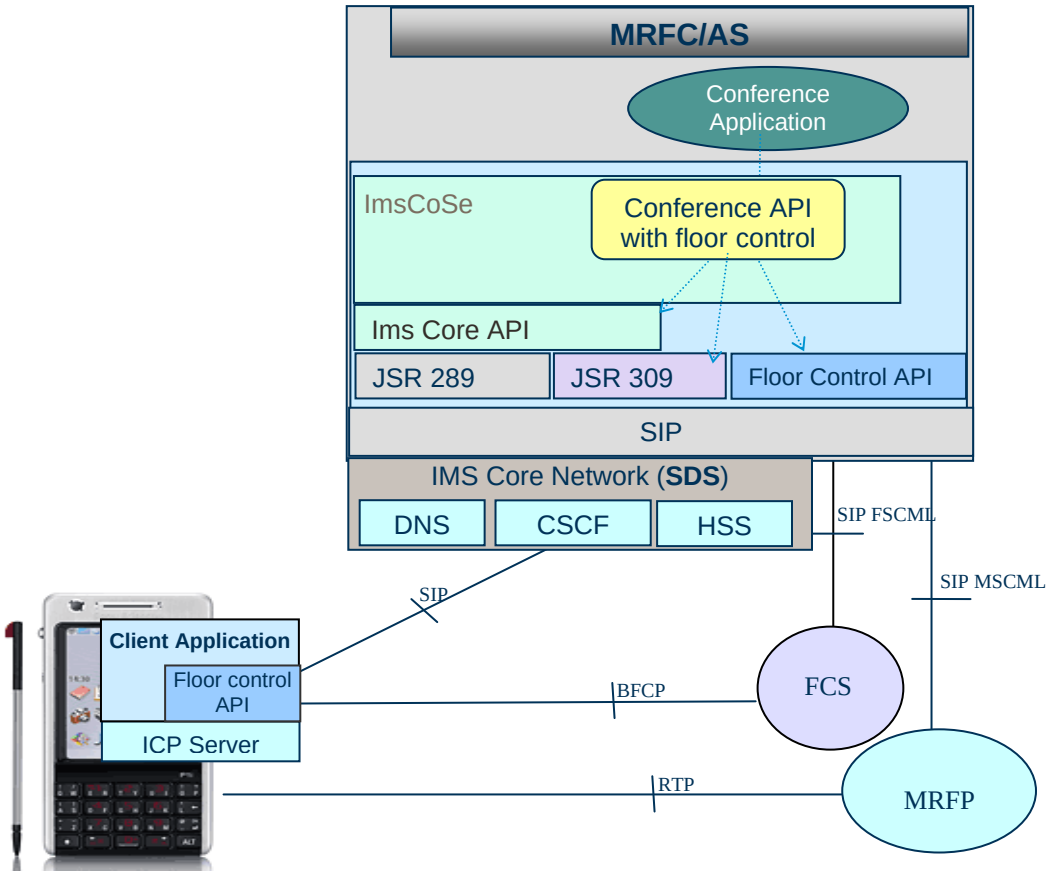

Figure 2: Global architecture

2.2 JSR 309 implementation

The current implementation of JSR 309 is an implementation of a subset of JSR 309 early draft review [5]. The implemented interfaces/classes are as follows.

- mscontrol package: MediaSessionFactory (createBasicMediaMixer, createMediaSession, createParameters), MediaSession (createNetworkConnection, createMediaMixer, createMediaGroup), Joinable (getJoinees, join, unjoin), JoinableContainer (addJoinableStream, getJoinableStreams).
- resource package: ResourcesContainer (getParameters, setParameters), MediaEvent, EventNotifier.
- networkconnection package.
- Mixer package: MediaMixer, BasicMediaMixer

- Mediagroup package: MediaGroup (getPlayer, getParameters, setParameters, join, release), PlayerEvent, Player

2.3 Server side main interfaces

Figure 3 shows a global view of the main conference API in the server side. The main interface is the '*ImsConference'*, which is responsible for creating and managing conferences (e.g. adding participant, removeing participant). The '*ImsConference'* extends the following interfaces: *ImsSignalingSessionManager*, *ImsMediaManager*, *ImsFloorControlManage* and *ImsSubConferenceManager*. *ImsSignalingSessionManager* and *ImsMediaManager* handle conference signaling (e.g. inviting a participant) and media (e.g. playing announcement to a participant) respectively. *ImsFloorControlManager* enables floor control management (e.g. creating floor for a given resource such as voice and adding participants to floor), within a conference. *ImsSubConferenceManager* is responsible for creating and managing sub-conferences. This interface has not been implemented in the current version.

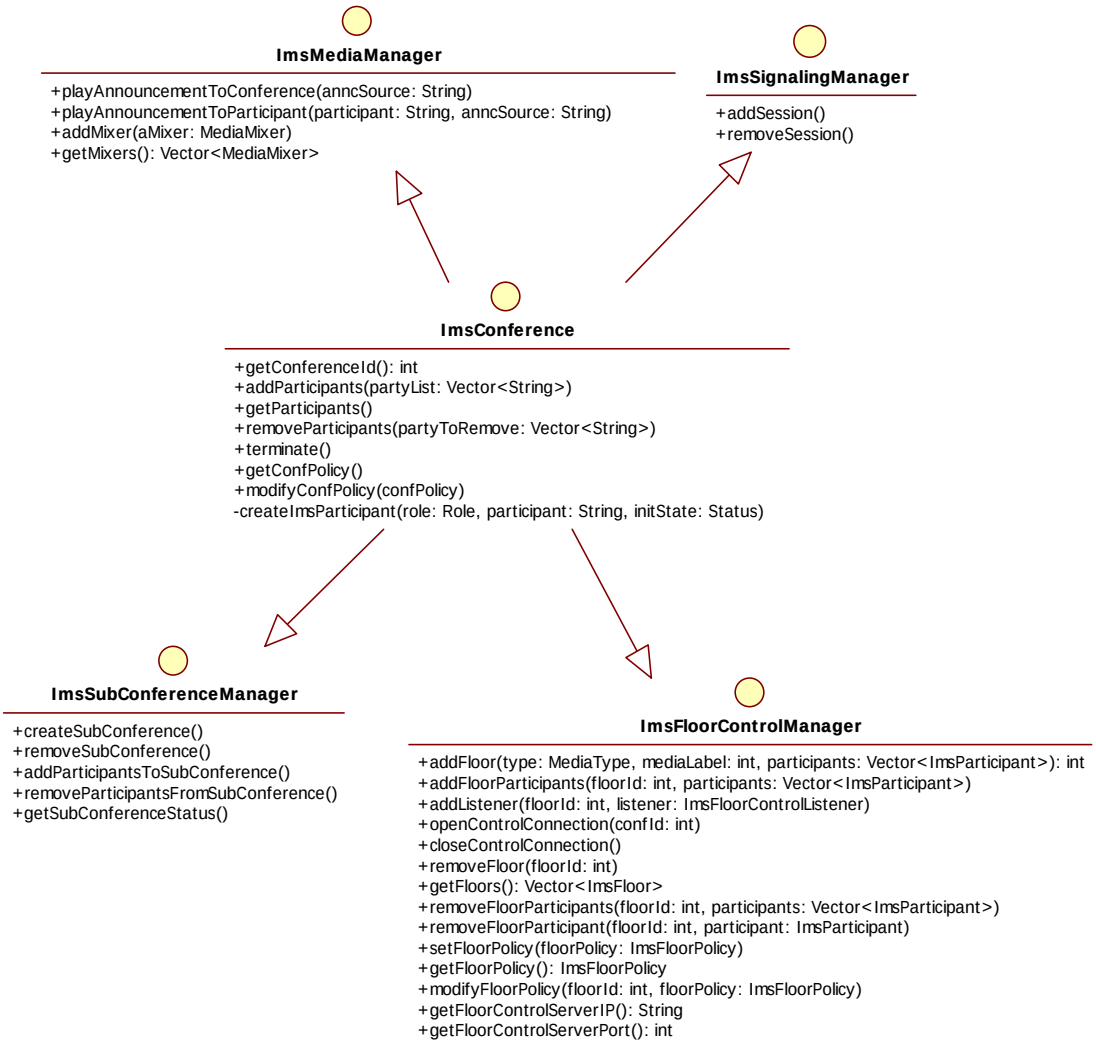

Figure 3: Main conferencing interfaces

2.4 Use cases and sample code

This section presents basic use cases which introduce examples of the functionalities provided by the conferencing API and how to use them. Each use case includes a sequence diagram that presents the different steps to follow, followed by sample code.

The next sub-section describes how to create a conference with floor control. The second sub-section discusses how to create a conference without floor control, and then add floor control to it. The third sub-section describes how to play announcement inside a conference.

2.4.1 Creating conference with floor control

Sequence diagram

Figure 4 shows the sequence diagram for creating a conference with floor control.

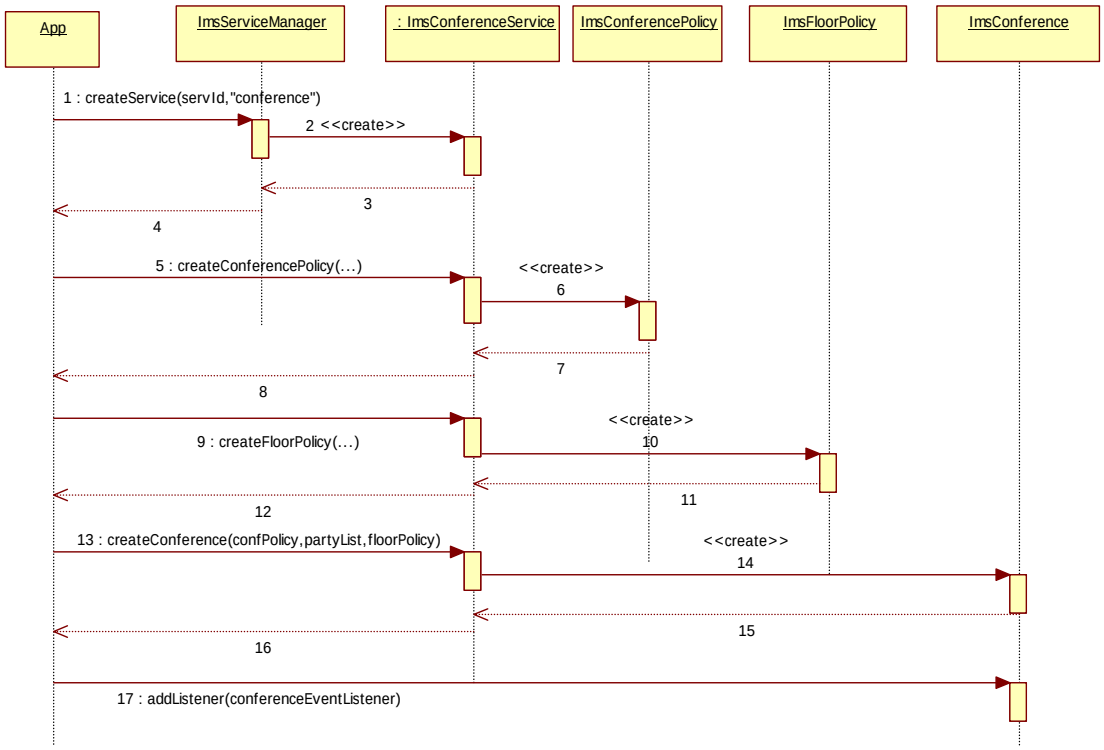

Figure 4: Create conference with floor control

The creation procedure can be described as follows:

1. The conference application gets an instance of the *ImsServiceManager*.
2. The conference application creates a new conferencing CoSe service, by calling `createService(…)` on the *ImsServiceManager* instance.
    a. The `"servId"` parameter specifies the conferencing service Id
    b. The conferencing service Id must be identical to the one specified in `%YOUR_Project_HOME%\src\main\webapp\WEB-INF\ims.xml`

```
<service-name name="Conferencing">
      <class>se.ericsson.ims.jee.conferencing
            .conference.ImsConferenceServiceImpl</class>
</service-name>
```

```
<service-id id="2663" register="false">
      <cscf>&lt;sip:192.168.1.152:5082&gt;</cscf>
</service-id>
```

In the example, the conference service id is ‘2663’.

3. The conference application creates an *ImsConferencePolicy*.
    a. The `startTime` parameter specifies the starting time of the conference. It is used to allow the creation of pre-arranged conferences. The current implementation only supports immediate creation of the conference (i.e. equivalent to `startTime` is now).
    b. The `duration` parameter specifies the maximum time duration of the conference, after which the conference is terminated automatically. The current implementation only supports explicit termination by the application (i.e. equivalent to `duration` is forever).
    c. The `maxParties` parameter specifies the maximum number of participants supported by the conference. It is used for resource reservation in the MRFP side.
    d. The `eventMethod` parameter specifies the notification method used by the conference to notify the clients about the conference related events (e.g. a new participant has joined the conference). There are two notification methods: notification by subscription (the client subscribes to a specific type of notification and by information (i.e. the client application does not need to subscribe but get notified when there is a change. The current implementation does not support conferencing event notification.
    e. The `mixingMode` parameter specifies the default mixing mode for the conference (e.g. full-duplex, half-duplex). (Not implemented).
    f. The `floorEnabled` parameter specifies whether the conference is floor enabled.
4. The conference application creates an *ImsFloorPolicy*. The *ImsFloorPolicy* specifies:
    a. The floor participants

b. The floor type (e.g. based on First Come First Serve algorithm, chair managed)
c. The floor chair if any
d. The maximum number of participants that can hold the floor at the same time (e.g. the maximum number of participants that can talk at the same time).
e. The maximum time a participant can hold a floor if another participant requests the same floor (i.e. the maximum time a participant can hold a floor before it is automatically revoked from him to give it to a new requestor).

5. The conference application creates a new conference, using the conference and floor policies created in the previous steps.

Sample code

```
ImsServiceManager manager = ImsServiceManager.getInstance();
ImsConferenceService confService = manager.createService(servId,
                                     "Conferencing");
ImsConferencePolicy confPolicy = confService.createConferencePolicy
                                   (startTime, duration,
                                   maxParties, eventMethod, mixingMode,
                                   floorEnabled);
ImsFloorPolicy floorPolicy = confService.createFloorPolicy(partyList,
                                    type, chair, maxHolder,
                                    maxHoldingTime);
ImsConference conference = confService.createConference(confPolicy,
                                    partyList, floorPolicy);
conference.addListener(confEventListener);
```

2.4.2 Creating conference and then adding floors

Sequence diagram

Figure 5 shows the sequence diagram for creating a conference without floor control and adding floor to it.

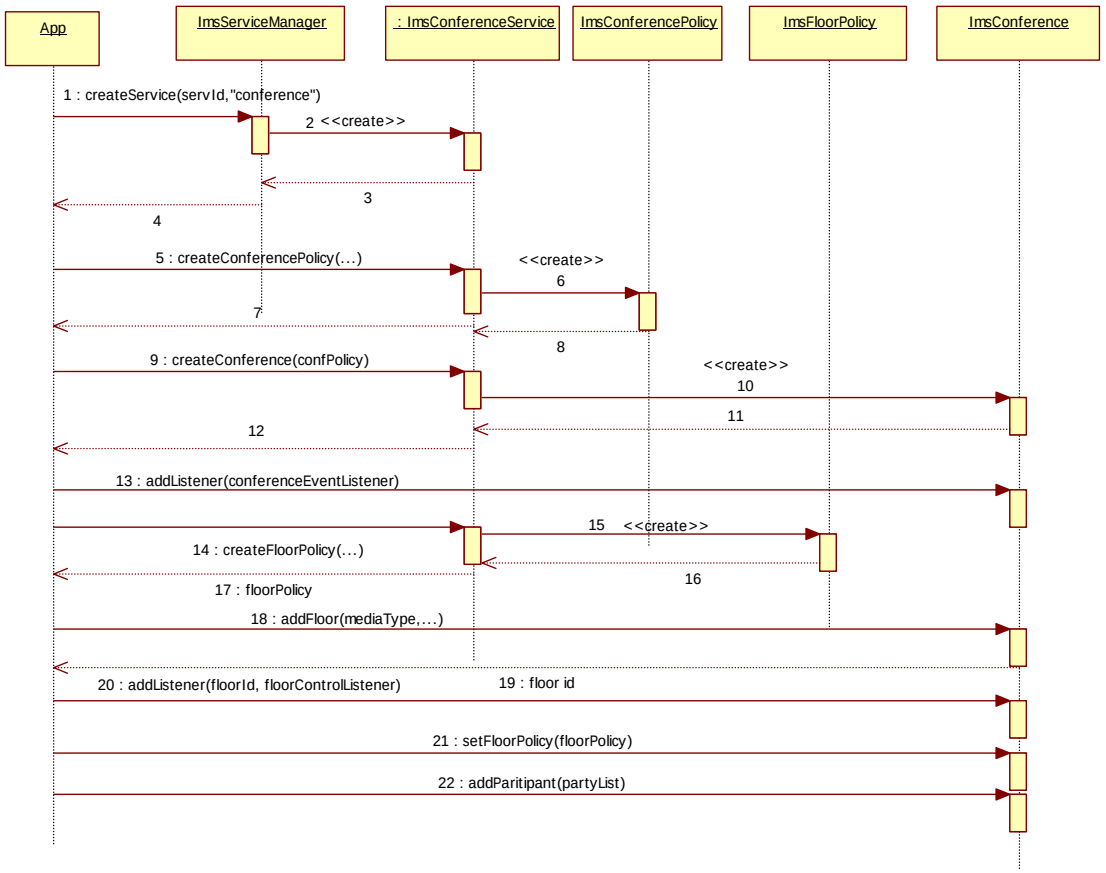

Figure 5: Creating conference without floor control and then adding floor to it

As in the previous use case, the conference application starts by getting an instance of the *ImsServiceManager* and creating a new conferencing CoSe service. After that, the application creates a new conference, while specifying the conference policy and the participants list. Then, it adds two floors (i.e. floor1 and floor2) to the conference. For each floor, the application specifies the associated floor policy, and specifies the conference as the listener to the floor events.

Sample code

```
ImsServiceManager manager = ImsServiceManager.getInstance();
ImsConferenceService confService = manager.createService(servId,
                                     "Conferencing");
ImsConferencePolicy confPolicy = confService.createConferencePolicy
                                   (startTime, duration,
                                   maxParties, eventMethod, mixingMode,
                                   floorEnabled);
conference = confService.createConference(policy);
conference.addListener(confEventListener);
```

```
//first floor
ImsFloorPolicy policy1 = service.createFloorPolicy(playersAddrList,
                    ImsFloor.FloorAlgorithm.FCFS, null,
                    maxHolders, 0);
floor1 = conference.addFloor(ImsMedia.MediaType.Audio, 1, null);
conference.addListener(floor1, conference);
conference.setFloorPolicy(floor1, policy1);

//second floor
ImsFloorPolicy policy2 = service.createFloorPolicy(
                    playersAddrList, ImsFloor.FloorAlgorithm.FCFS,
                    null, 1, 0);
floor2 = conference.addFloor(ImsMedia.MediaType.Object, 2, null);
conference.addListener(floor2, conference);
conference.setFloorPolicy(floor2, policy2);

conference.addParticipants(playersAddrList);
```

2.4.3 Playing announcement within a conference

To play announcement to a given participant within a conference, the conference application can call the function conference.playAnnouncementToParticipant() (see Figure 6). This function has two parameters: the sip address of the participant (e.g. alice@ericsson.com) and the file to play (e.g. welcome.wav). The file to plays should be stored in the MRFP beforehand.

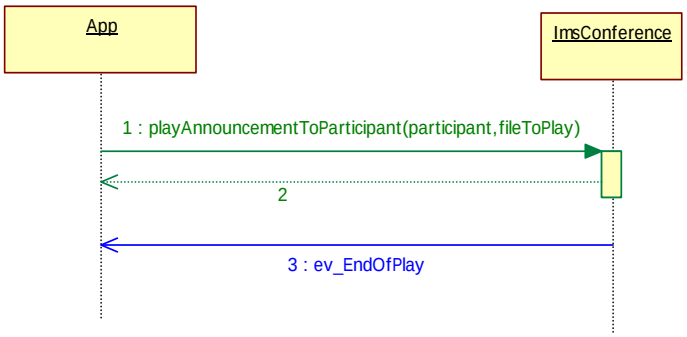

Figure 6: Play announcement to a participant

3. Client side API

This chapter presents how to use the floor control API at the client side to provide floor control functionalities for clients within an existing IMS conference.

The API allows an IMS conference client to:

- Establish floor control connection with the FCS.
- Create floor with associated resources.
- Request and release the floor.
- Terminate the floor control connection.

3.1 Floor control API overview

The com.ericsson.ims.client.floorcontrol package provides classes for all the floor control related functionalities at the client side. Table 1 provides an overview of the APIs in this package. Then in Fig. 7, we show relationship between these APIs.

Table 1: Floor control API Overview

Class	**Functions**
ImsClientFloorFactory	Creates the Ims Floor Session.
ImsFloorSession	Establishes a floor connection between the client and the FCS.
ImsFloorSessionListener	Enables the client application to receive floor control events from the FCS (e.g. floor accepted, granted, and released).
ImsFloor	Provides floor functionality for participants to request, release the floor, and also for adding and removing resources.
ImsFloorSessionEvent	Provides details for the floor event reports received by the client application (e.g. floor accepted, granted, released).
ImsResource	Represents the resource that is associated to the floor (e.g. voice, video and pointer).

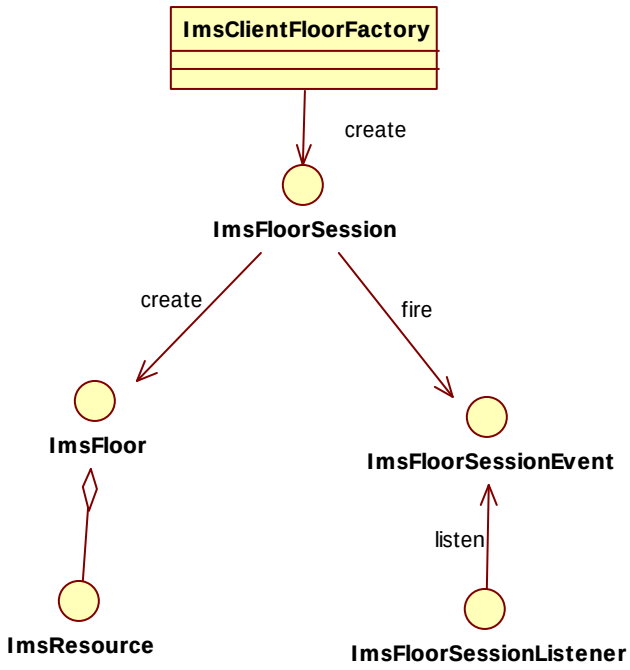

Figure 7: Floor control API class diagram

3.2 Using floor control API

This section provides information on how to use the floor control API for application development.

3.2.1 Creating ImsFloor

Sequence diagram

Fig.8 shows the sequence of creating ImsFloor.

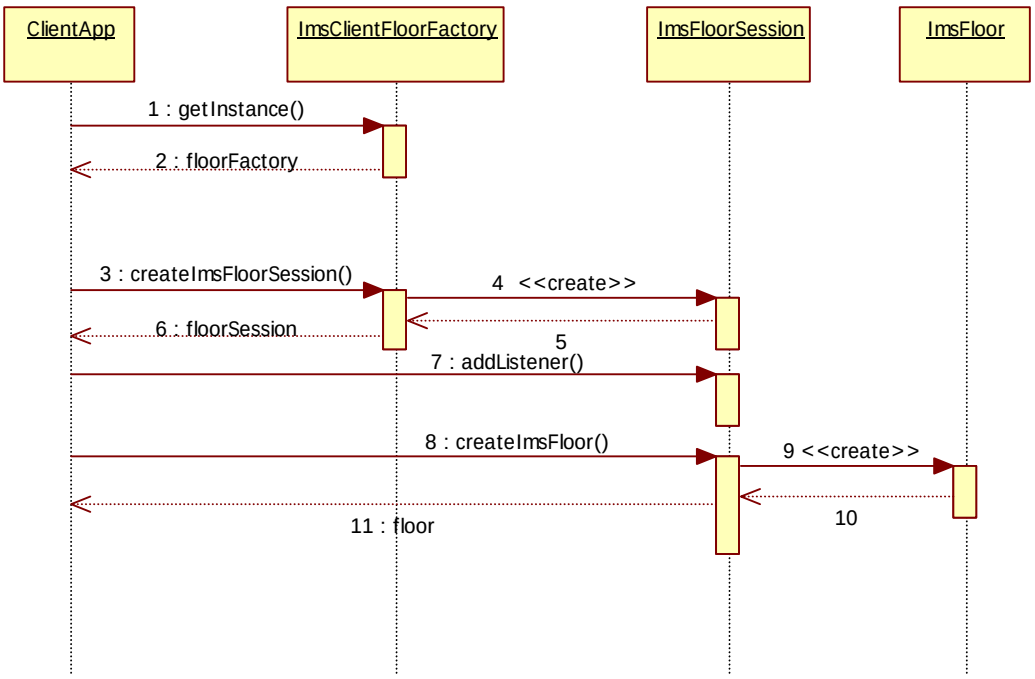

Figure 8: Creating ImsFloor

The procedure of creating ImsFloor can be described as follows:

1. The client application gets an instance of the ImsClientFloorFactory.
2. The client application creates an ImsFloorSession by calling createImsFloorSession on the ImsClientFloorFactory instance. The floor session represents the connection between the client application and the floor control server.
    - The client application should receive the floor control related information in the SDP body of the INVITE message. This includes the conference id, the user id, and the FCS IP address and port number. For more information on SDP floor control related information, refer to RFC 4583 [6].
3. The client application adds an ImsFloorSessionListener in order to listen to the floor events generated by the ImsFloorSession (e.g. to report events received from the FCS).
4. The client application creates an ImsFloor using the floor id, which is also received in the floor control related SDP attributes in the SIP INVITE message.

For more information on SDP floor control related information, refer to RFC 4583 [6].

Sample code

```
ImsClientFloorFactory floorFactory = ImsClientFloorFactory.getInstance();
ImsFloorSession floorSession = factory.createImsFloorSession(confId, userId,
                                                    fcsIP, fcsPort);
floorSession.addListener(imsFloorSessionListener);
ImsFloor voiceFloor = floorSession.createImsFloor(floorId);
```

3.2.2 ImsFloor manipulation

Floor participants can request and release floors. We will use a voice floor as example to show these operations in this section.

a) Request an ImsFloor

The voice floor created earlier (see section 3.2.1) can be requested using the following code:

```
int floorRequestId = voiceFloor.requestFloor();
```

The requestFloor() method is a synchronized method that returns an id associated to the floor request at the FCS side. This id can be used to release the floor or cancel the pending floor request in the future.

b) Release the ImsFloor

The client can release the granted voice floor or cancel the pending floor request using the following code:

```
voiceFloor.releaseFloor(floorRequestId);
```

3.2.3 Terminate the floor session

The floor session can be terminated using the code below.

```
floorSession.terminateSession();
```

The termination of the floor session releases the connection between the FCS and the client. It also removes all the floors with their associated resources from the client side.

3.3 Floor control API implementation and ICP

The current implementation of the client floor control API is based on ICP (The ICP related documentation is provided by the SDS package). The communication between the client application and the FCS is based on SIP MESSAGE request. The BFCP requests and responses are embedded in SIP MESSAGE body. We use the ICP IService interface to send and receive the SIP MESSAGE. Therefore, the IService instance created by the client application should be passed to the ImsFloorSessionImpl. This will enable the floor session to send the floor control messages. The following code shows how the IService instance can be passed to the ImsFloorSessionImpl:

```
((ImsFloorSessionImpl) floorSession).setService(service, profile.getIdentity());
//profile is an instance of the ICP IProfile
```

The floor control messages sent by the FCS are received by the ICP IServiceListener in the client side, through the processMessage() method. After the message is received, the analyseMessage()method, which is defined in ImsFloorSession implementation, should be called. This will trigger the processing of the floor control message. The following code shows the function.

```
public void processMessage(String from, String messageType, byte[] message,
                           int messageLength) {
((ImsFloorSessionImpl) floorSession).analyseMessage(message);
}
```

References

[1] TS 24.147, “Conferencing using the IP Multimedia (IM) Core Network (CN) subsystem”, Stage 3, Release 8, March 2008

[2] J. Rosenberg et al., “SIP: Session Initiation Protocol”, IETF RFC 3261, June 2002

[3] H. Schulzrinne et. al., “RTP: A Transport Protocol for Real-Time Applications”, IETF RFC3550, July, 2003

[4] G. Camarillo et al, “The Binary Floor Control Protocol (BFCP)”, RFC 4582, November 2006

[5] JSR 309 early draft review, Oct 2007, online at: http://jcp.org/aboutJava/communityprocess/edr/jsr309/index.html

[6] G.Camarillo, “Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams”, RFC 4583, Nov. 2006