Android builders summit
The Android media framework

Author: Bert Van Dam & Poornachandra Kallare
Date: 22 April 2014
Usage models

• Use the framework: MediaPlayer
  – `android.media.MediaPlayer`
  – Framework manages
    • Demuxing
    • Decoding
    • AV synchronization
    • AV rendering

• DIY: the application manages
  – Demuxing: `android.media.mediaExtractor`
  – Decoding: `android.media.MediaCodec`
  – Video rendering: `android.media.MediaCodec`
  – Audio rendering: `android.media.AudioTrack`
MediaPlayer usage model

• The easy way: instantiate **VideoView**
  – Creates the MediaPlayer for you
  – Exports similar API to MediaPlayer

• The slightly more complicated way
  – Application creates **SurfaceView**
  – Application creates **MediaPlayer**
  – MediaPlayer.setSurface(surface)
Which media players exist

- **Built-in players**
  - AwesomePlayer (default player selected)
  - NuPlayer (Apple HLS)
  - SonivoxPlayer (midi files)
  - testPlayer

- **Extra player factories can be registered**

- **Every player provides same interface**
  - frameworks/av/include/media/MediaPlayerInterface.h
Architecture

**Application**
- `android.media.MediaPlayer`
  
**JNI**
- `frameworks/base/media/java/android/media/MediaPlayer.java`
- `frameworks/base/media/jni/android_media_MediaPlayer.cpp`
- `frameworks/av/media/libmedia/mediaplayer.cpp`

**Binder**
- `frameworks/av/media/libmediaplayerservice/MediaPlayerService.cpp`
- `frameworks/av/media/libmediaplayerservice/MediaPlayerFactory.cpp`

**Media service**
- `frameworks/av/media/libmediaplayerservice/nuplayer/NuPlayerDriver.cpp`
- `frameworks/av/media/libmediaplayerservice/StagefrightPlayer.cpp`
- `frameworks/av/media/libstagefright/AwesomePlayer.cpp`
Player creation (simplified)

(1) mp = new MediaPlayer();

native_setup(new WeakReference<MediaPlayer>(this));

sp<MediaPlayer> mp = new MediaPlayer();

Object initialization
mAudioSessionId = AudioSystem::newAudioSessionId();
AudioSystem::acquireAudioSessionId(mAudioSessionId);

Nothing much happened yet ...
Player creation (simplified)

Which player handles this URL???
Player creation factory

```cpp
player_type MediaPlayerFactory::getDefaultPlayerType() {
    char value[PROPERTY_VALUE_MAX];
    if (property_get("media.stagefright.use-nuplayer", value, NULL) && (!strcasecmp("1", value) || !strcasecmp("true", value))) {
        return NU_PLAYER;
    }
    return STAGEFRIGHT_PLAYER;
}
```

```cpp
class NuPlayerFactory : public MediaPlayerFactory::IFactory {
    public:
    virtual float scoreFactory(const sp<IMediaPlayer>& client, const char* url, float curScore) {
        static const float kOurScore = 0.8;
        if (kOurScore <= curScore)
            return 0.0;
        if (!strncasecmp("http://", url, 7) || !strncasecmp("https://", url, 8)) {
            size_t len = strlen(url);
            if (len >= 5 && !strcasecmp(".m3u8", &url[len - 5])) {
                return kOurScore;
            }
            if (strstr(url, "m3u8")) {
                return kOurScore;
            }
        }
        if (!strncasecmp("rtsp://", url, 7)) {
            return 0.0;
        }
    }
}
```

```cpp
class SonivoxPlayerFactory : public MediaPlayerFactory::IFactory {
    public:
    virtual float scoreFactory(const sp<IMediaPlayer>& client, const char* url, float curScore) {
        static const float kOurScore = 0.4;
    }
```
AwesomePlayer

• Building blocks
  – OMX-IL
    • http://www.khronos.org/openmax/il/
    • Standardized interface for accessing streaming components
    • Google provides set of SW decoders
    • SOC suppliers provide HW accelerated decoders

  – MediaExtractors
    • frameworks/av/media/libstagefright/
    • Classes capable of demuxing specific container formats (MP3Extractor, MPEG4Extractor, MatroskaExtractor, …)
    • Allow extraction of audio, video, subtitle tracks

  – Audioflinger, surfaceflinger for rendering
OMX-IL - principles

Used by Stagefright players
OMX-IL – Android integration
OMX-IL – example config file

```xml
<MediaCodecs>
  <Decoders>
    <MediaCodec name="OMX.google.mp3.decoder" type="audio/mpeg" />
    <MediaCodec name="OMX.google.amrnb.decoder" type="audio/3gpp" />
    <MediaCodec name="OMX.google.amrwb.decoder" type="audio/amr-wb" />
    <MediaCodec name="OMX.google.aac.decoder" type="audio/mp4a-latm" />
    <MediaCodec name="OMX.google.g711.alaw.decoder" type="audio/g711-alaw" />
    <MediaCodec name="OMX.google.g711.mlaw.decoder" type="audio/g711-mlaw" />
    <MediaCodec name="OMX.google.vorbis.decoder" type="audio/vorbis" />
    <MediaCodec name="OMX.google.mpeg4.decoder" type="video/mp4v-es" />
    <MediaCodec name="OMX.google.h263.decoder" type="video/3gpp" />
    <MediaCodec name="OMX.google.h264.decoder" type="video/avc" />
    <MediaCodec name="OMX.google.vpx.decoder" type="video/x-vnd.on2.vp8" />
  </Decoders>

  <Encoders>
    <MediaCodec name="OMX.google.aac.encoder" type="audio/mp4a-latm" />
    <MediaCodec name="OMX.google.amrnb.encoder" type="audio/3gpp" />
    <MediaCodec name="OMX.google.amrwb.encoder" type="audio/amr-wb" />
    <MediaCodec name="OMX.google.h263.encoder" type="video/3gpp" />
    <MediaCodec name="OMX.google.h264.encoder" type="video/avc" />
    <MediaCodec name="OMX.google.mpeg4.encoder" type="video/mp4v-es" />
    <MediaCodec name="OMX.google.flac.encoder" type="audio/flac" />
  </Encoders>
</MediaCodecs>
```
MediaPlayer.prepare

mConnectingDataSource = HTTPBase::Create;
mConnectingDataSource->connect(URL);
mCachedSource = new NuCachedSource2();
dataSource = mCachedSource;
creates a ChromiumHttpclient

Go through the cache from here onwards

Wait for 192 KB of data in the cache
Datasource->sniff();
extractor = MediaExtractor::Create(MIME, datasource);
Calculate bitrate of stream through extractor
Select first video and audio stream as default
initVideoDecoder()
  mVideoSource = OMXCodec::Create();
  mVideoSource->start();
initAudioDecoder()
  mAudioSource = OMXCodec::Create();
  mAudioSource->start();
Create and start the video decoder
Create and start the audio decoder

Continue buffering
Notify Prepared state when highwatermark is reached

Detect the MIME type of the stream
Create the extractor

AwesomePlayer.cpp
RegisterSniffer_1(SniffMPEG4);
RegisterSniffer_1(SniffMatroska);
RegisterSniffer_1(SniffOgg);
RegisterSniffer_1(SniffWAV);
RegisterSniffer_1(SniffFLAC);
RegisterSniffer_1(SniffAMR);
RegisterSniffer_1(SniffMPEG2TS);
RegisterSniffer_1(SniffMP3);
RegisterSniffer_1(SniffAAC);
RegisterSniffer_1(SniffMPEG2PS);
RegisterSniffer_1(SniffWVM);

MediaPlayer is now ready to start playback
Decoding is not yet happening at this stage!!!
Status after prepare

MediaSource API instances
- mVideoTrack
- mAudioTrack

Mediaplayer

OMXCodec (video)
OMXCodec (audio)

FileSource (local files)

NuCachedSource2 (network files)

ChromiumHttpClient (fetches data over IP)

Mediaplayer

buffer
MediaPlayer.start

```cpp
void Application::startAudioPlayer_l()
{
    // Start the audio player
    mAudioPlayer = new AudioPlayer();
    mAudioPlayer->setSource(mAudioSource);
    mTimeSource = mAudioPlayer;
    startAudioPlayer_l();
    mTextDriver->start();
    initRenderer_l();
    Start video event generation
    Render buffers after applying AV sync logic

    // Audio track used as timing reference
    // Starts the audio player
    // Start subtitle player
    // Initialize the rendering path (based on SW/HW codec)
    // Loop of video events with A/V sync logic
}```
Status after start

- ChromiumHttpQuoteClient (fetches data over IP)
- NuCachedSource2 (network files)
- MediaExtractor
- FileSource (local files)
- OMXCodec (video)
- OMXCodec (audio)
- AudioPlayer
- AwesomePlayer
- SurfaceFlinger
- AudioFlinger
- AwesomeRenderer
- nativeWindow

Video data pulled by timed events
Audio data pulled by sink through callback

MediaSource API instances:
- mVideoTrack
- mAudioTrack

Audio data pulled by sink through callback
Audio data through callback
Track selection

– MediaPlayer. getTrackInfo
  • Returns list of tracks

<table>
<thead>
<tr>
<th>Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>MEDIA_TRACK_TYPE_AUDIO</td>
</tr>
<tr>
<td>int</td>
<td>MEDIA_TRACK_TYPE_TIMEDTEXT</td>
</tr>
<tr>
<td>int</td>
<td>MEDIA_TRACK_TYPE_UNKNOWN</td>
</tr>
<tr>
<td>int</td>
<td>MEDIA_TRACK_TYPE_VIDEO</td>
</tr>
</tbody>
</table>

– MediaPlayer. selectTrack(idx)
  • Maps to MediaExtractor
  • Select audio, video or subtitle track
Subtitle handling

• Limited formats supported
  – SRT, 3GPP

• Both embedded and external files
  – `addTimedTextSource` to add external file
  – `MediaPlayer.getTrackInfo` returns both internal and external subtitle tracks

• Player takes care of syncing to playback time
  – `TimedText` notifications raised at correct time
Subtitle rendering

To render the timed text, applications need to do the following:

- Implement the `MediaPlayer.OnTimedTextListener` interface
- Register the `MediaPlayer.OnTimedTextListener` callback on a `MediaPlayer` object that is used for playback
- When a `onTimedText` callback is received, do the following:
  - call `getText()` to get the characters for rendering
  - call `getBounds()` to get the text rendering area/region

Simple `TextView` can be used to render
The DIY model

- **android.media.MediaCodecList**
  - Returns supported formats
  - Based on config.xml file explained before

- **android.media.MediaCodec**
  - Is basically an abstraction of OMX-IL
  - Application juggles buffers to and from component

- Application acts as the player in this case
  - Responsible for rendering + AV sync
The DIY model – typical setup

Create SurfaceView (for rendering video)
Create AudioTrack (for rendering audio)
Create MediaExtractor (alternatively have your own system for ES retrieval)
  -> query tracks
  -> selectTrack(audio track idx)
  -> selectTrack(video track idx)
  -> getTrackFormat(idx)
Create MediaCodec for audio and for video
Configure MediaCodecs as per formats detected above, and start them

while (1) on thread 1
{
  extr.readSampleData
  extr.getSampleTrackIndex // determine if it’s the audio or video
  extr.getSampleTime // presentation time
  audio/videodec.queueInputBuffer
}

while(1) on thread 2
{
  audio/videodec.dequeueOutputBuffer
  audiotrack.write for audio – videodec.releaseOutputBuffer for video
}
Classic DRM Framework

Classic DRM Framework

• The Android DRM framework is implemented in two architectural layers
  • A DRM framework API exposed to applications via Dalvik/Java.
    • Application/DRM specific handling for license acquisition, etc.
  • A native code DRM manager
    • Implements the DRM framework
    • Exposes an interface for DRM plugins (agents) to handle rights management and decryption for various DRM schemes.

• The interface for plugin developers is listed and documented in DrmEngineBase.h.
  • Identical to the Java DRM Framework API (DrmManagerClient).

• On the device, the DRM plugins are located in “/vendor/lib/drm” or in “/system/lib/drm”.

• DRM Plugins work with media framework for content decryption
Prepare Redux – Classic DRM

Example

AwesomePlayer.cpp

mConnectingDataSource = HTTPBase::Create;
mConnectingDataSource->connect(URL);
mCachedSource = new NuCachedSource2();
dataSource = mCachedSource;

creates a ChromiumHttpClient

Go through the cache from here onwards

AwesomePlayer.cpp

Wait for 192 KB of data in the cache
Datasource->sniff();
extractor = MediaExtractor::Create(MIME, datasource);
Calculate bitrate of stream through extractor
Select first video and audio stream as default

Detect the MIME type of the stream
Create the extractor

Create and start the video decoder
Create and start the audio decoder

RegisterSniffer(SniffDRM)

There is a media extractor instance for DRM called DrmExtractor. DrmExtractor implements SniffDRM
Status after prepare – Classic DRM

AwesomePlayer

MediaSource API instances
• mVideoSource
• mAudioSource

OMXCodec (video)

OMXCodec (audio)

MediaSource API instances
• mVideoSource
• mAudioTrack

MediaSource API instances
• mVideoTrack
• mAudioTrack

DrmSource

DrmPlugins

OriginalExtractor

DrmExtractor

NuCachedSource2 (network files)

FileSource (local files)

ChromiumHttpClient (fetches data over IP)
DRM with media codec

• Applications using mediacodec can also use DRM
  • Example: MPEG DASH CENC
  • Using MediaCrypto and MediaDRM
• MediaDRM provides application API to
  • Provision DRM clients
  • Generate DRM/content specific challenges
  • Download licenses/keys
  • Generate a session ID that can be used to create media crypto objects
• MediaCrypto object obtained from MediaDRM can then be used with mediacodec
  • Submit to media codec using
    \texttt{public final void queueSecureInputBuffer (int index, int offset,}
    \texttt{MediaCodec.CryptoInfo info, long presentationTimeUs, int flags)}
• Internally uses a plugin framework
  • Not the same plugins as used in classic DRM!
  • Different set of plugins with different API
DRM with Mediacodec

http://developer.android.com/reference/android/media/MediaDrm.html
Media framework changes

- Audio track selection improvements
  - Improve runtime audio track changes

- Trickmodes
  - Android only supports Seek
  - I-Frame based trickmodes, DLNA compliancy (x1/2, x1/4)

- Adaptive streaming added (DASH, …)

- Subtitle gaps
  - Add SAMI, SUB, external TTML, …

- DRM extensions
  - PlayReady, WMDRM, Marlin
TV inputs

Extra player taking care of TV inputs (tuner, extensions)

class TvPlayerFactory : public MediaPlayerFactory::IFactory {
public:
    virtual float scoreFactory(const sp<IMediaPlayer>& client,
                                const char* url,
                                float curScore)
    {
        static const float kOurScore = 2.0;

        if (kOurScore < curScore)
            return 0.0;

        if (!strcasecmp("tv://", url, 5))
        {
            return kOurScore;
        }
        return 0.0;
    }

class TvPlayerFactory : public MediaPlayerFactory::IFactory {
public:
    virtual float scoreFactory(const sp<IMediaPlayer>& client,
                                const char* url,
                                float curScore)
    {
        static const float kOurScore = 2.0;

        if (kOurScore < curScore)
            return 0.0;

        if (!strcasecmp("tv://", url, 5))
        {
            return kOurScore;
        }
        return 0.0;
    }

    virtual sp<MediaPlayerBase> createPlayer() {
        ALOGV(" create TvPlayerBase");
        return new TvPlayerBase();
    }
};