Subject: ***Design Specification for stream player on opera*** **Date:** **8/10/2016**

From: **Hawking Wang**
+86 532 557-55479
wangxinzhi@hisense.com

1

TABLE OF CONTENTS

1. Technical Introduction

1.1 Purpose

This document is intended to provide a design document of player on opera.

1.2 Scope

The document will focus on inter-working of MediaBackend, PlayerImpl and vendor player. And define the interface for vendor to implement.

1.3 Feature Description

This part depicts the high level description of this feature. Usually needs to introduce some standard, requirements and related scenarios.

1.4 Acronyms

Acronym	Definition
ES	Elementary Streams
PTS	Presentation Time Stamp
MSE	Media Stream Extension
DRM	Digital Rights Management
TEE	Trusted Execution Environments

2. Document Overview

2.1 Reason for Reissue

This is the second version of the document.

Revision	Date	Author	Description
0.1	Aug 1st, 2016	Hawking Wang	Created
0.2	Aug 8, 2016	Hawking Wang	Revised for comments of reviewers
0.3	Aug 10, 2016	Hawking Wang	Revised for comments of reviewers

2.2 Assumed Reader Knowledge

The reader should be familiar with:

1. UVA Backend of Opera
2. Basic knowledge of player

2.3 Open Issues

For some cases, the decrypted data should be stored at trust zone (TEE), which is prohibit from accessing by user. So the flow of injecting data will be different. But we hasn't known the use case of the vendor API, this part will be completed in future.

3. Design Constraints

The URL player (with encryption content) will implemented by vendor, while accessed by PlayerImplMedia

4. Design Strategy

It will reuse the current prekilby(v3) architecture as many as possible. There three types of stream for opera, MSE_STREAMING, OPERA_STREAMING and BACKEND_STREAMING. We mainly focused on the first one. The second one is optional, we don't support it now. The most work of BACKEND_STREAMING will within vendor player.

5. Design Description

5.1 Design Overview

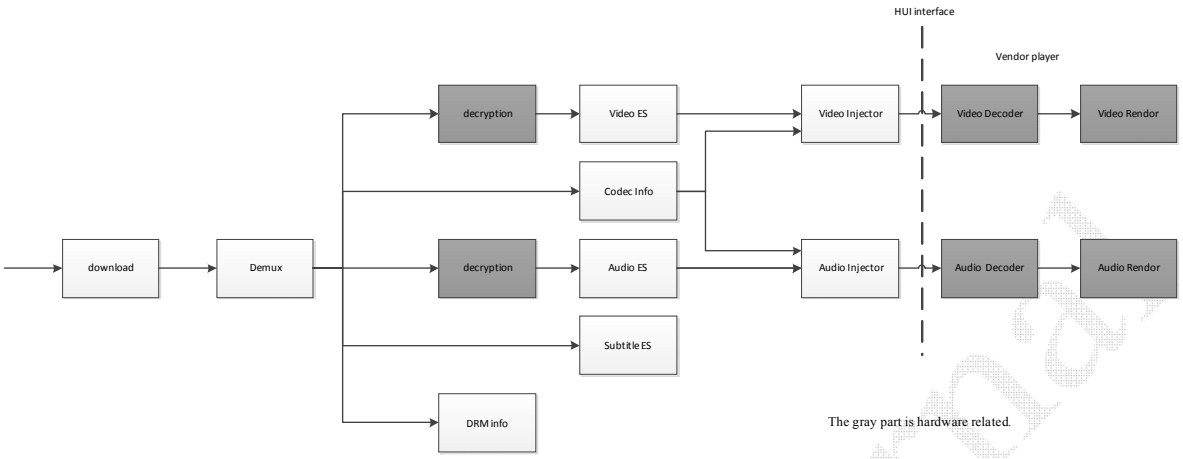

Figure 5.1.1: Data flow

For MSE_STREAMING type, only gray part is implemented by vendor. For BACKEND_STREAMING type, all are implemented by vendor.

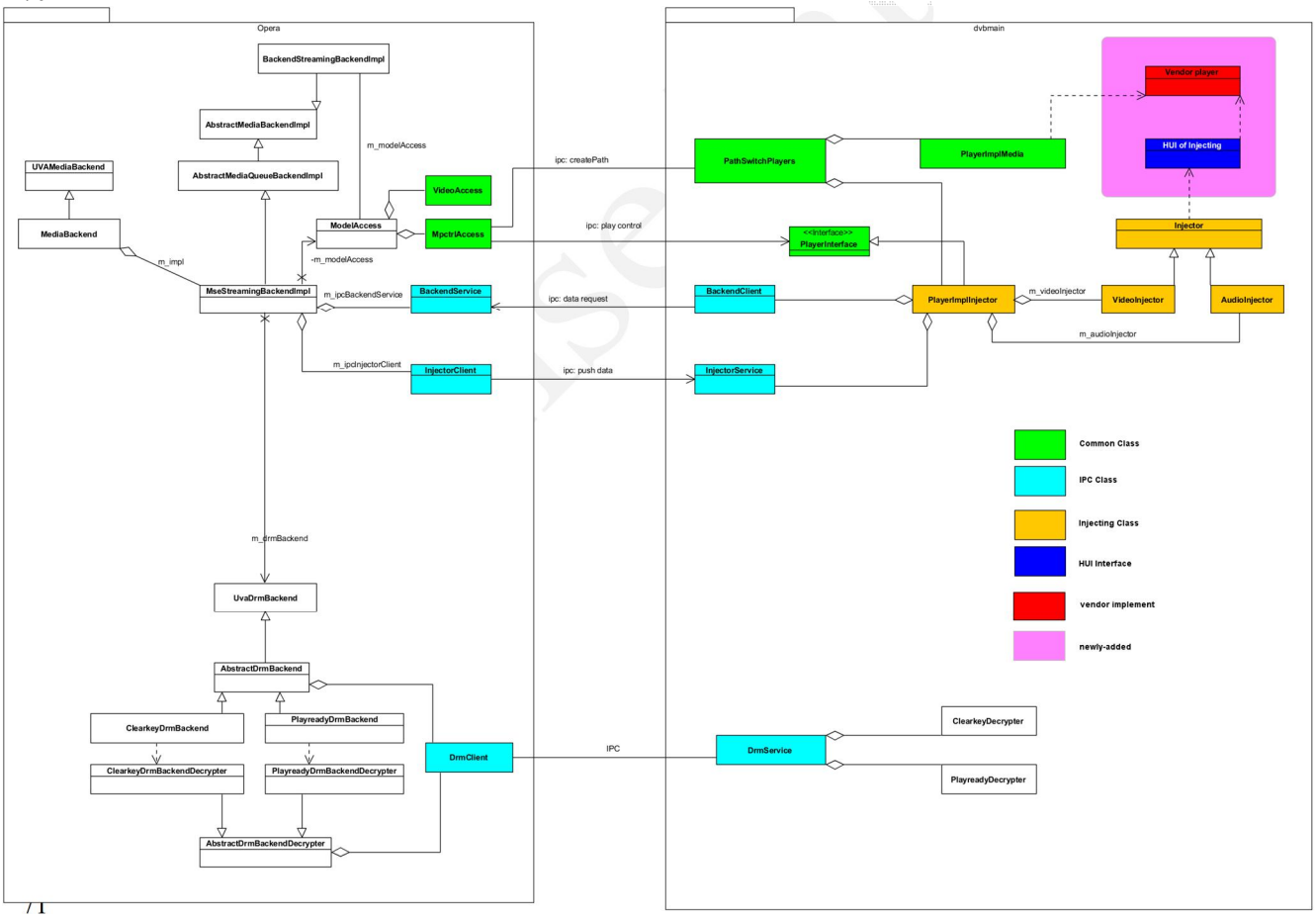

Figure 5.1.2: Class diagram

5.1.1 MSE_STREAMING

The opera is responsible for downloading and demuxing, the MediaBackend is responsible for inter working with DrmBackend. The PlayerImplInjector got ES data via shared memory. The data is decrypted in MediaBackend, so the PlayerImpl always got clear stream. When setup up, the dvbmain create related path to PlayerImpl_Injector, and handle related conflict with other source (DTV decoder is conflicted with MediaPlayer) The PlayerImpl_Injector is nder vamp-path control just like normal source. The only difference between physical source and stream is where the data coming. So all control command and event callback is routed via MpCtrl/biz.

5.1.1.1 Overview of HUI interface

It's common for video and audio, determined only by the open type.

```
HUI_U32 HUI_StreamInjectInit(void);

HUI_U32 HUI_StreamInjectDeInit(void);

HUI_U32 HUI_StreamInjectGetCapability(HUI_StreamInjectCapbility_s *pCapability);

HUI_U32 HUI_StreamInjectOpen(HUI_StreamInjectOpenParam_s *pOpenParam, HUI_StreamInjectHandle_t *pInjectHandle);

HUI_U32 HUI_StreamInjectClose(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectRegisterCallback(HUI_StreamInjectHandle_t injHandle, HUI_StreamInjectCallback_f callback, void
*pUserData);

HUI_U32 HUI_StreamInjectUnRegisterCallback(HUI_StreamInjectHandle_t injHandle, HUI_StreamInjectCallback_f callback);

HUI_U32 HUI_StreamInjectSetCodec(HUI_StreamInjectHandle_t injHandle, HUI_StreamInjectCodec_u);

HUI_U32 HUI_StreamInjectStart(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectStop(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectPause(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectResume(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectGetFreeSize(HUI_StreamInjectHandle_t injHandle, HUI_U32 *pInjBufFreeSizeByte);

HUI_U32 HUI_StreamInjectWriteData(HUI_StreamInjectHandle_t injHandle, void *pData, HUI_U32 length, HUI_U64 pPtsInMs);

HUI_U32 HUI_StreamInjectGetPts(HUI_StreamInjectHandle_t injHandle, HUI_U64 *pPtsInMs);

HUI_U32 HUI_StreamInjectNotifyEOS(HUI_StreamInjectHandle_t injHandle);

HUI_U32 HUI_StreamInjectFlush(HUI_ StreamInjectHandle _t injHandle);

HUI_U32 HUI_StreamInjectSetDisplayRect(HUI_StreamInjectHandle_t injHandle, HUI_Rectangle_t *pSrect);
```

5.1.1.2 Init and Feed data

To setup vendor player of Video/Audio, the PlayerImplInjector provide the stream type and codec type:

```
typedef enum
{
  STREAM_INJECT_STREAM_TYPE_MIN = 0,
  STREAM_INJECT_STREAM_TYPE_VPES = STREAM_INJECT_STREAM_TYPE_MIN, //Optional
  STREAM_INJECT_STREAM_TYPE_APES, //Optional
  STREAM_INJECT_STREAM_TYPE_VES, //Mandatory
  STREAM_INJECT_STREAM_TYPE_AES, //Mandatory
  STREAM_INJECT_STREAM_TYPE_MAX,
  INJECT_STREAM_TYPE_INVALID = -1,
}HUI_StreamInjectStreamType_e;
```

```
typedef enum
{
  HUI_STREAMINJECT_CODEC_UNKNOWN,
  HUI_STREAMINJECT_CODEC_VIDEO_H264 = (1 << 0),
  HUI_STREAMINJECT_CODEC_VIDEO_H265 = (1 << 1),
  HUI_STREAMINJECT_CODEC_VIDEO_VC1 = (1 << 2),
  HUI_STREAMINJECT_CODEC_VIDEO_MPEG2 = (1 << 3),
  HUI_STREAMINJECT_CODEC_VIDEO_MPEG4 = (1 << 4),
  HUI_STREAMINJECT_CODEC_VIDEO_THEORA = (1 << 5),
  HUI_STREAMINJECT_CODEC_VIDEO_VP8 = (1 << 6),
  HUI_STREAMINJECT_CODEC_VIDEO_VP9 = (1 << 7),
} HUI_StreamInjectVideoCodec_e;

typedef enum
{
  HUI_STREAMINJECT_CODEC_UNKNOWN,
  HUI_STREAMINJECT_CODEC_AUDIO_AAC = (1 << 0),
  HUI_STREAMINJECT_CODEC_AUDIO_MP3 = (1 << 1),
  HUI_STREAMINJECT_CODEC_AUDIO_PCM = (1 << 2),
  HUI_STREAMINJECT_CODEC_AUDIO_VORBIS = (1 << 3),
  HUI_STREAMINJECT_CODEC_AUDIO_OPUS = (1 << 4),
  HUI_STREAMINJECT_CODEC_AUDIO_EAC3 = (1 << 5),
  HUI_STREAMINJECT_CODEC_AUDIO_AC3 = (1 << 6),
  HUI_STREAMINJECT_CODEC_AUDIO_DTS = (1 << 7),
} HUI_StreamInjectAudioCodec_e;
```

After setup up, the vendor player need enough buffered data before playing.
The PlayerImplInjector should know the total size and current free size of buffer in vendor player.

```
HUI_U32 HUI_StreamInjectGetCapability(HUI_StreamInjectCapbility_s *pCapability);
HUI_U32 HUI_StreamInjectGetFreeSize(HUI_StreamInjectHandle_t injHandle, HUI_U32 *pInjBufFreeSizeByte);
```

It hold two thresholds, one is to determine the vendor player has enough data to play and another to determine need of feed new data.

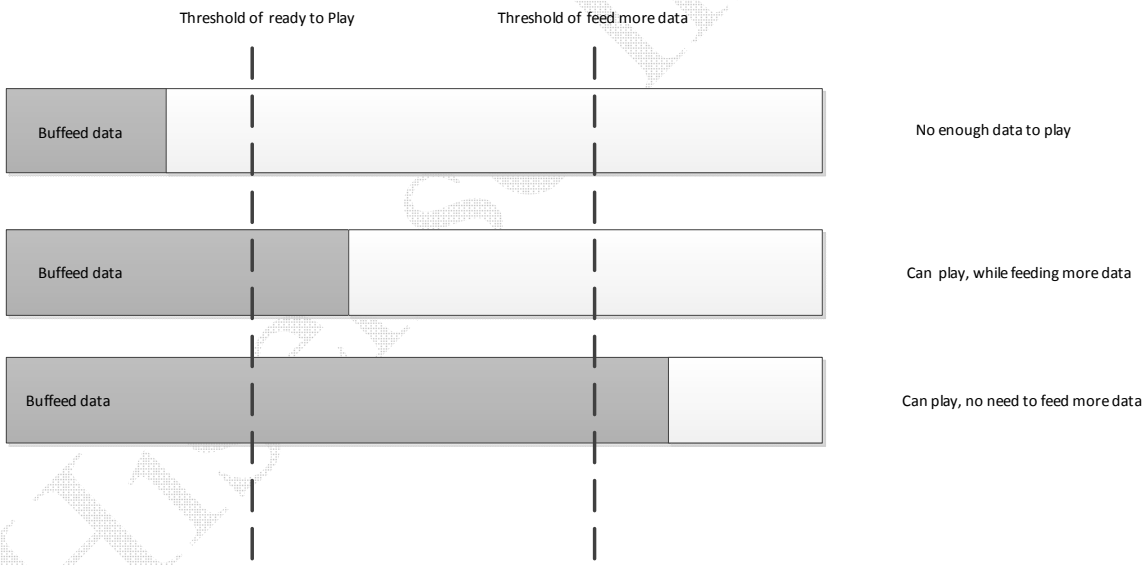

Figure 5.1.1.2: threshold of buffer

If there are no enough data to play, data query event will be send out. It's triggered by PlayerImplInjector via BackendClient. Then the MediaBackend forward the request to opera with NewMSEDataEvent message.

The opera call MediaBackend's interface named writeData with buffer address, length, codec and PTS info.
Then MediaBackend forward the data to PlayerImplInjector with InjectorClient/InjectorService.
PlayerImplInjector then feed data to vendor player via HUI interface:

```
HUI_U32 HUI_StreamInjectGetFreeSize(HUI_StreamInjectHandle_t injHandle, HUI_U32 *pInjBufFreeSizeByte);
HUI_U32 HUI_StreamInjectWriteData(HUI_StreamInjectHandle_t injHandle, void *pData, HUI_U32 length, HUI_U64 pPtsInMs);
```

Video info event post by common interface which can be handled by VideoAccess in BackEnd.

5.1.1.3 The seek process

Beside of the description on opera's document, the vendor player should be paused. When data arrived at new position, clean up the previous buffered data, fill with new data then resume.

5.1.1.4 AV sync

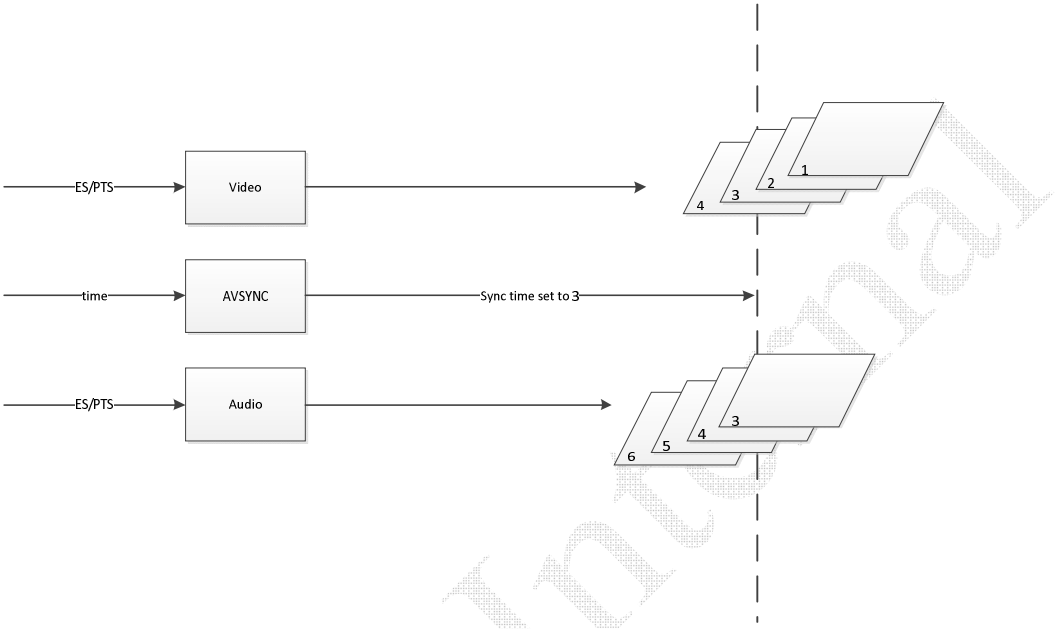

Figure 5.1.1.4: AV sync

The vendor player should has the capability of syncing video and audio stream.

Interface :

```
HUI_U32 HUI_StreamInjectAVSyncInit();

HUI_U32 HUI_StreamInjectAVSyncDeInit();

HUI_U32 HUI_StreamInjectAVSyncOpen(HUI_StreamInjectAvSyncHandle_t *pInjAvSyncHandle);

HUI_U32 HUI_StreamInjectAVSyncClose(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle);

HUI_U32 HUI_StreamInjectAVSyncTimerSetTime(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle, HUI_U64 timeInMs);

HUI_U32 HUI_StreamInjectAVSyncTimerGetTime(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle, HUI_U64 *pTimeInMs);

HUI_U32 HUI_StreamInjectAVSyncTimerPause(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle);

HUI_U32 HUI_StreamInjectAVSyncTimerUnPause(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle);

HUI_U32 HUI_StreamInjectAVSyncSetAudioLatency(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle, HUI_U64 timeInMs);

HUI_U32 HUI_StreamInjectAVSyncGetAudioLatency(HUI_StreamInjectAvSyncHandle_t injAvSyncHandle, HUI_U64 *pTimeInMs);
```

5.1.1.5 Sequence digrams

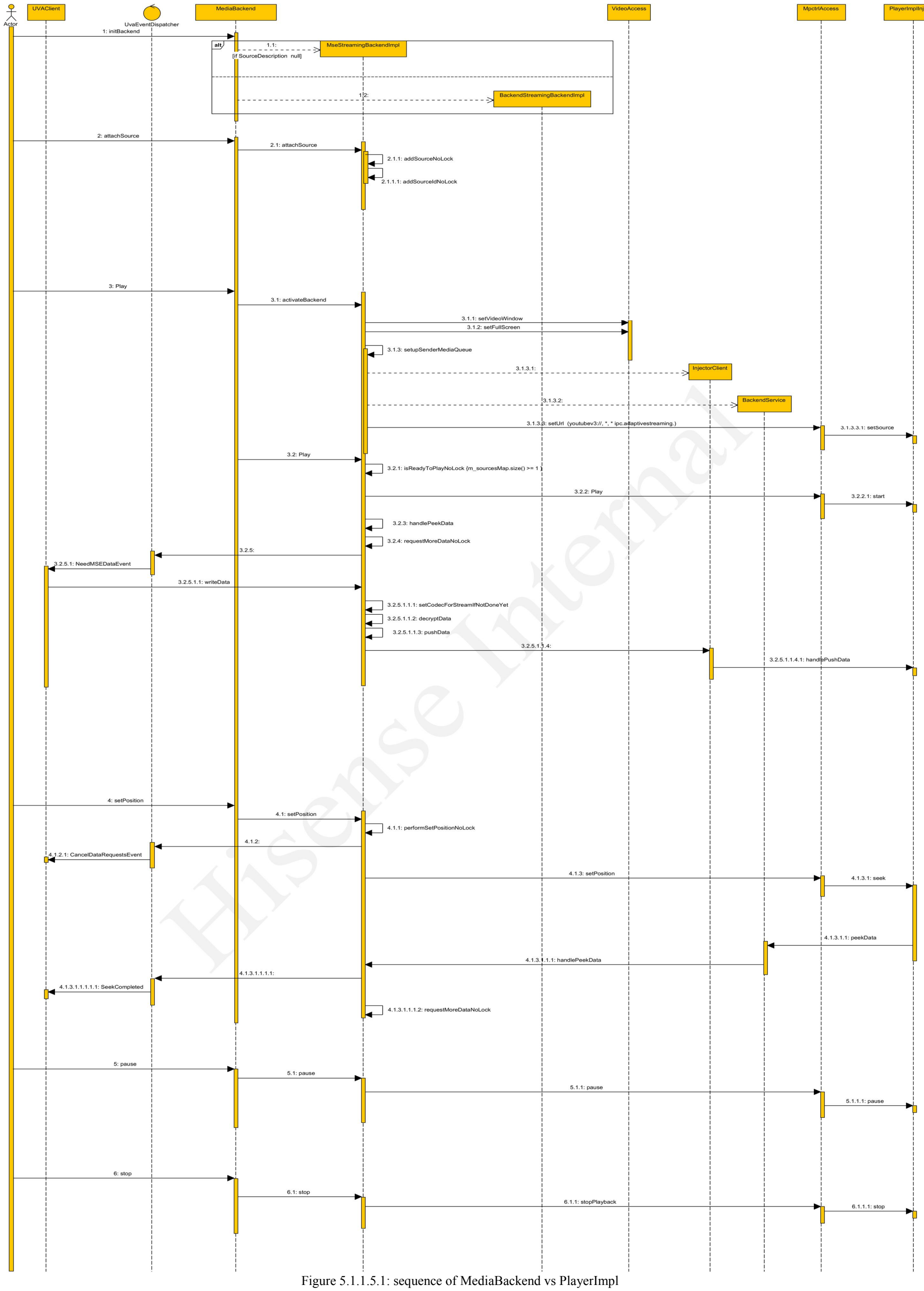

Figure 5.1.1.5.1: sequence of MediaBackend vs PlayerImpl

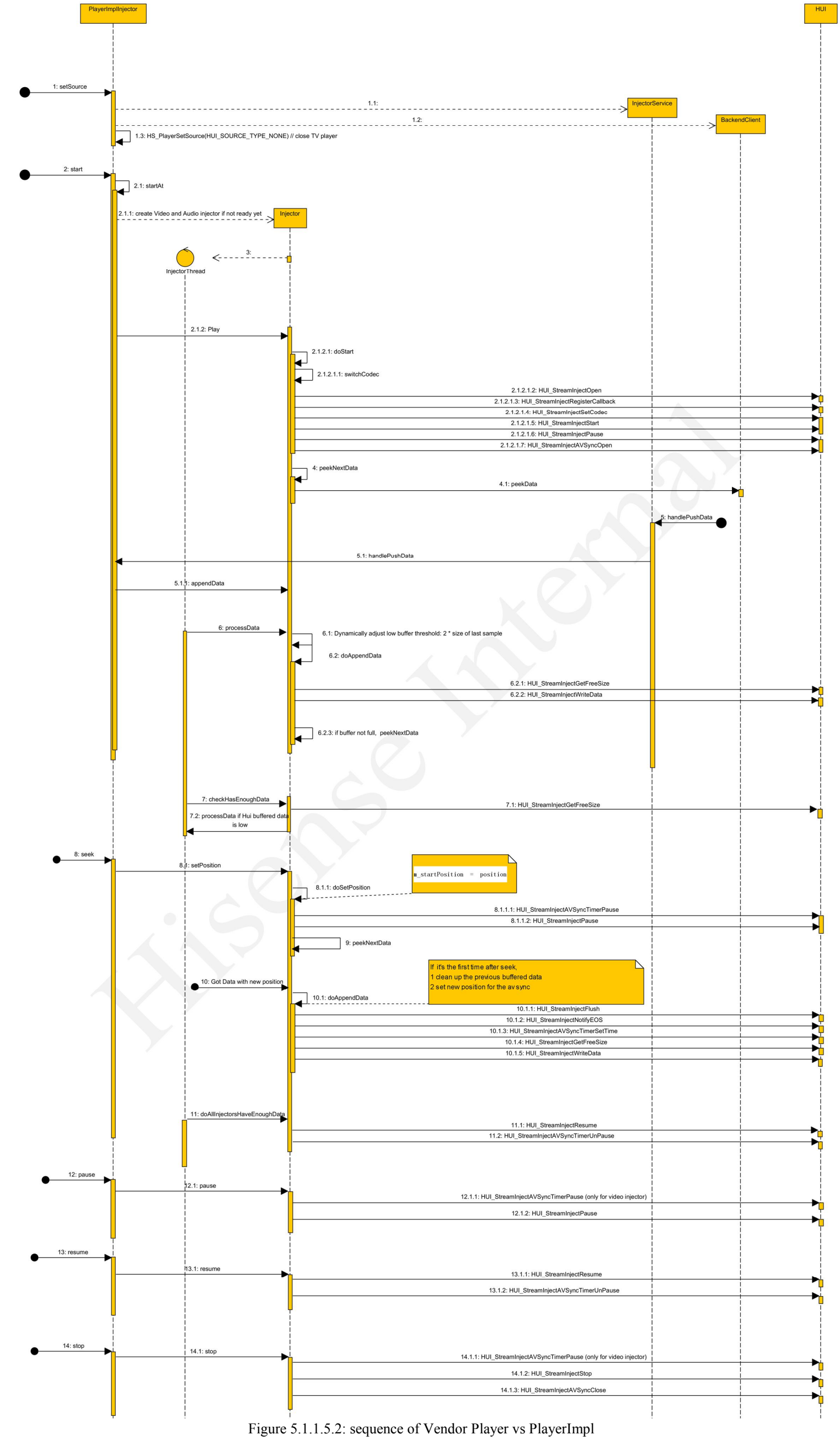

Figure 5.1.1.5.2: sequence of Vendor Player vs PlayerImpl

5.1.2 BACKEND_STREAMING

As we will finally implement the downloading and demuxing with gstreamer at future. So we prefer to use solution 5.1.2.2 as a whole solution at this phase to save effort.

5.1.2.1 Current PreKilby solution

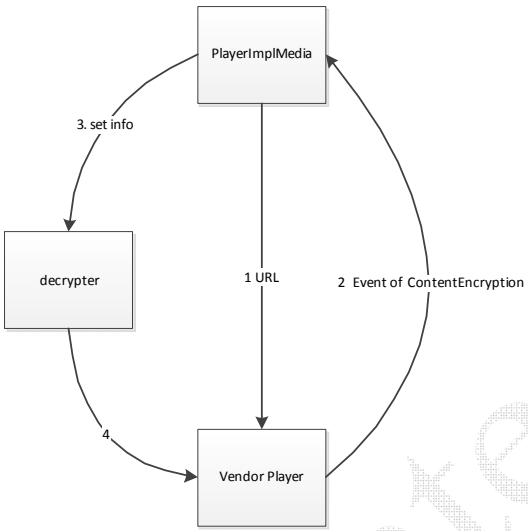

Figure 5.1.2.1: PreKilby solution of BACKEND_STREAMING

The vendor player has the capability to download, demux and decode. If the content is encrypted, it will report the related info to PlayerImplMedia. Then the PlayerImplMedia use same decrypter as DrmBackend to decrypt the content. The data will not pass to PlayerImplMedia.
Only the playready is supported now.

5.1.2.2 Fully vendor solution.

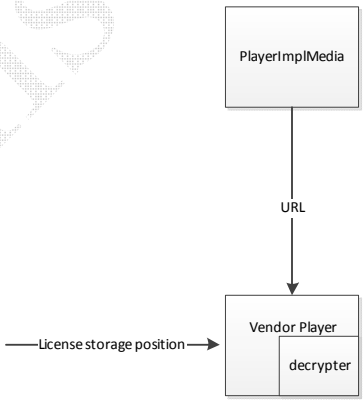

Figure 5.1.2.2: fully vendor solution of BACKEND_STREAMING

Implemented fully by vendor with fixed position of license key storage. Control command and event call back is passed by common player interface.

5.1.3 The mapping between HUI and player interface defined by Jamdeo team.

Note：The HUI interface is mainly focused on ES injector which is used by MSE_STREAMING, so there are no “set video/audio/subtitle tracks” api.

	HUI	Jamdeo
1	HUI_StreamInjectInit	

2	HUI_StreamInjectDeInit	
3	HUI_StreamInjectGetCapability	bufferSize
4	HUI_StreamInjectOpen	
5	HUI_StreamInjectClose	
6	HUI_StreamInjectRegisterCallback	
7	HUI_StreamInjectUnRegisterCallback	
8	HUI_StreamInjectSetCodec	
9	HUI_StreamInjectStart	play
10	HUI_StreamInjectStop	stop
11	HUI_StreamInjectPause	pause
12	HUI_StreamInjectResume	
13	HUI_StreamInjectGetFreeSize	dataRemaining
14	HUI_StreamInjectWriteData	writeStream
15	HUI_StreamInjectGetPts	
16	HUI_StreamInjectNotifyEOS	
17	HUI_StreamInjectFlush	
18	HUI_StreamInjectSetDisplayRect	setScreenPosition
19	HUI_StreamInjectAVSyncInit	
20	HUI_StreamInjectAVSyncDeInit	
21	HUI_StreamInjectAVSyncOpen	
22	HUI_StreamInjectAVSyncClose	
23	HUI_StreamInjectAVSyncTimerSetTime	setTime/ seek
24	HUI_StreamInjectAVSyncTimerGetTime	getPosition
25	HUI_StreamInjectAVSyncTimerPause	pause
26	HUI_StreamInjectAVSyncTimerUnPause	
27	HUI_StreamInjectAVSyncGetAudioLatency	
28	HUI_StreamInjectAVSyncSetAudioLatency	

5.1.4 Mapping between hs_mediaplayer_api and player interface defined by Jamdeo team.

Note: It's mainly for BACKEND_STREAMING case.

	Hs_mediaplayer_api	Jamdeo
1	IMediaPlayer_Play	play
2	IMediaPlayer_TimeSeek	seek
3	IMediaPlayer_GetPosition	getPosition
4	IMediaPlayer_Stop	stop
5	IMediaPlayer_Pause	pause
6	IMediaPlayer_SetPlaybackSpeed	setSpeed
7	IMediaPlayer_SetVideoWindow	setScreenPosition
8	IMediaPlayer_SetAudioTrackNo	setAudioTrack
9		setVideoTrack
10	IMediaPlayer_SetTxtTrackNo	setSubtitleTrack
CALL BACKS		
11	MEDIAPLAYER_EVENT_CUR_TIME_UPDATE	current position changed
12	MEDIAPLAYER_EVENT_TOTAL_TIME_UPDATE	duration has changed
13	MEDIAPLAYER_EVENT_PLAYER_STATUS_IDLE/PREPARING/PREPARED/PLAYING/PAUSED/STOP/RELEASING	state changed
14	Null , but has API to query the counts of these tracks	AV component changed (e.g., audio, video, or subtitle track)
15	Nul	AV component selected (e.g., audio, video, or subtitle track)

5.2 Data Design

5.3 UI Design

N/A

5.4 Application Design

N/A

5.5 Component and Functionality Configuration Design

NA

6. System Impact

NA

6.1 CPU Impact

NA

6.2 Memory Impact

NA

6.3 Others

NA

7. External Interfaces

NA

8. ACKNOWLEDGEMENTS

Thanks to all of project development team members and the reviewers for this design.