Deliverable 4.3.1

Project Title	**Next-Generation Hybrid Broadcast Broadband**
Project Acronym	**HBB-NEXT**
Call Identifier	**FP7-ICT-2011-7**
Starting Date	**01.10.2011**
End Date	**31.03.2014**
Contract no.	**287848**
Deliverable no.	**4.3.1**
Deliverable Name	**EVALUATION: Intermediate Middleware Software Components for Content Synchronization**
Work package	**4**
Nature	**Report**
Dissemination	**Public**
Author	**Christian Köbel (Editor), Christopher Köhnen, Nils Hellhund, Bastian Zeller (THM), Ray van Brandenburg, Arjen Veenhuizen (TNO), Ranina Renz, Michael Probst (IRT), Björn Stockleben (RBB)**
Contributors	
Due Date	**31.05.13**
Actual Delivery Date	**07.06.13**

Table of Contents

Executive Summary

This deliverable contributes inter-device synchronization (multi-screen) components to the overall HBB-NEXT system architecture and completes the previously introduced inter-media (multi-stream) synchronization components, described in deliverable D4.2 [4]. In the course of the document, advanced media synchronization services are presented. A joint synchronization scheme has been developed by HBB-NEXT WP4 partners, and has been ported to different hardware and software platforms. The selection of platforms reflects the current state-of-the art in consumer devices. For later evaluation, WP4 partners have developed a set of demonstrators.

Requirements and the Synchronization Approach

Before explaining the synchronization method in detail, requirements to enable these methods are discussed. To limit the scope of scenarios, all media devices can be considered to be grouped at the same location and in the same sub-network. This is commonly the case in a living room or in a home in general. A home-wide WiFi access point provides mobility to the end-devices. Device and media service discovery in a local network, and the resulting session establishment and management are essential tasks in a multi-device environment. In our approach, the used service discovery is able to detect devices and their respective shared services. This is done in advance, before media synchronization.. It also covers the case that several synchronization masters are present, to which devices can connect to. Once all devices have been registered, their system clocks are synchronized in the network. To enable a faultless, synchronized playout of digital media content on all involved devices, timing information is constantly exchanged. This information can be described as a mapping between the timeline of the media content and the wallclock timeline of the regarding device. Using extrapolation, receivers of this information are then able to calculate the target playout time for all subsequent video frames and audio samples. In the presented approach, a “synchronization master” is defined, which provides timing information to “slave” devices. Slave devices will use a seek method to control the play position of the played out media stream, in order to keep in sync with other devices and the master content timeline.

In order to prevent incrementing clock drift and possible playback interruptions, synchronization messages (containing timing information) are proactively signalled. The applied inter-device synchronization solution is simple and does not cause much network overhead, due to small and infrequent messages, also for the sake of portability to other platforms and APIs.

The Hardware and Software Components

Tablets and Set-Top-Boxes (STB) have been considered as hardware platform categories. As a STB platform, a Broadcom STB has been modified: Novel synchronization features have been directly included in the Inaris[1] DVB/HbbTV middleware for STBs. Inaris already includes state-of-the-art DVB and HbbTV receiver features. The HBB-NEXT extensions to the HbbTV Receiver module support both inter-media and inter-device synchronization. It now additionally features the fully integrated extraction of the DVB broadcast timeline, which was introduced in the previous deliverable D4.2. The module returns the current tick value of the broadcast stream in milliseconds, which is later on used to construct a joint content timeline for locally connected media devices. Also, Picture –in-Picture rendering of video/broadcast and video/mp4 in parallel is now possible on the STB; a requirement especially developed for the HBB-NEXT project. Since Android is one of the most popular mobile platforms, an Android synchronization app has been developed. It is placed inside Androids Java Layer. The app includes a user interface, methods to connect with the current synchronization master device and access to the media player control components of Android. It supports both master and slave functionality for coordination of multi-device synchronization. To demonstrate the inter-media synchronization abilities of the app, a Picture-in-Picture sign language interpreter, different languages (audio tracks), audio description and subtitles can be played out in a synchronized fashion. An application for iOS devices, such as the Apple iPad or iPhone, has been created with similar intentions. As a requirement for this app, a complete video renderer has been developed, with direct control over iOS' video decoding and playback timing modules.

[1] http://www.tara-systems.de/inaris-dvb-middleware.html

Using such a low-level renderer offers great advantages. It is possible to achieve very tight synchronization and enables to control the playout of video frames with millisecond accuracy. The possibility to allow synchronization not only between devices of the same type, but also between different platforms was a pervasive design criterion in the development process of the tablet and STB software modules. Additionally, the Gstreamer synchronization framework [17] has been improved, which works as an alternative to the synchronization framework on the STB. Gstreamer now includes service discovery and the aforementioned inter device synchronization algorithm. In its role as a STB, it automatically serves as a synchronization master in the network. This is due to the fact, that the GStreamer framework is primarily used to render a broadcast DVB stream, which by definition does not support seeking of live DVB-media. Firstly, Gstreamer now supports the processing of the broadcast timeline, which is nested in a DVB MPEG2 Transport Stream (see deliverable D4.2 [4]). Secondly, Gstreamer is able to distribute the timing information from the broadcast timeline to subscribed clients, just as a regular STB would do.

The Demonstrators

The developed demos cover different aspects of inter-device synchronization. The first demo treats the synchronized playout of different content formats on multiple end-devices, using a tailored settings application. As one of the main efforts of WP4 partners in this deliverable, the concept and design of this application have been developed, as well as closely related user studies. The technical adaption of the settings application to an HbbTV environment has been also handled. The application has a strong focus on accessibility. The offered services are synchronized sign language interpretation video, extra audio description / alternative language track and multi-language subtitling. All these services can be combined with the regular DVB broadcast signal and other on-demand IP video sources in various ways, all in a multi-device-ready media environment. A custom GUI helps to configure device-dependent playout options. Another demo enables precise inter-media synchronization for 3D content. In this demo, two video streams (one for the left eye, one for the right eye) from hybrid sources are harmonized on a Broadcom set-top-box, to create a single 3D video playout. The FascinatE demo [15] includes powerful inter-device features, combining content of two heterogeneous media sources. It is based on a combination of the Gstreamer framework and a set of iOS devices.

It allows end-users to interactively view and navigate around an ultra-high resolution video panorama of a live football game. The main screen hosts the regular broadcast DVB streams, whereas companion devices (e.g. iPad or iPhone) can be used to zoom in and navigate around the ultra-high resolution video panorama, completely in sync with the main DVB stream shown on the TV. In this demo, both source combinations of IP-IP and DVB-IP are considered for inter-media and inter-device synchronization.

1. Introduction

Today, users in the digital media world have adopted often more than one digital end-device to their daily lives. The heterogeneous device landscape spreads across countless digital TV sets, set-top boxes, smartphones and tablets, along with major operating systems like iOS or Android. Users already have high expectations on the quality of each single device (in terms of manufacturing resolution, weight, speed, and compatibility). But now it's up to broadcasters, content providers and the gadget industry to seamlessly integrate all of the user's consumer devices, in order to create a unifying media experience across all involved screens, speakers and input devices. Part of this integration is the synchronized playout of content: Nowadays, consumers already tend to use one or more media devices in front of a TV. Extending the TV experience to additional devices requires a robust and fast synchronization method, which allows media content to be consumed on multiple devices simultaneously.

The presented deliverable *D4.3.1: Intermediate Middleware Software Components for Content Synchronization* conquers this challenge by creating next-generation solutions for one of the most important aspects on multi-device media interaction: flexible synchronization of audio, video and metadata (e.g., subtitles). The deliverable documents the development of novel media sync components and their synthesis, leading to the design and realization of a set of demos. Each of the carefully selected demos covers different aspects relevant for multi-device usage, such as multi-screen video playout or second screen companionship. In these demos, an innovative mixture of different services is included, based on both inter-device and inter-media synchronization. To mention only two, an extra sign language interpreter video can be played out synchronously on a tablet, which enriches the media experience and improves the accessibility of the media environment. A highly accurate synchronized playout of 3D content represents the ultimate inter-media synchronization scenario, as the playout of video streams for the left and right eye needs to be constantly exact on video frame level.

The basic structure of this document is similar to the previous WP4 deliverable D4.2 [4] in the sense that it contains a two-step approach to documenting the work done in WP4. The first part (equivalent to Section 4) describes the approach in general, including algorithms and conceptual details (such as message formats). The second part (equivalent to Sections 5 and 6) describes the technical implementations, such as the used hardware, content, and demo assembly. Thus, the current deliverable is structured as follows: Following the introduction, Section 2 provides an overview on related work. Section 3 depicts the general HBB-NEXT framework, and inter-relates the deliverable with the overall architecture. Section 4 describes the joint multi-device synchronization approach, developed in conjunction by the WP4 partners. Several related aspects, such as the clock sync on end-devices are covered here. Section 5 describes the required components device-wise. Installation instructions for the device-related software components are also included in Section 5. Section 6 deals with the demos, which include and deploy the devices introduced in Section 5. Section 7 concludes the document.

2. Related Work

This Section briefly outlines selected approaches which have been published since the release of the HBB-NEXT deliverable D4.1 [3]. The set of included papers contains relevant technological aspects of recent media synchronization solutions. On the implementation side, specifics of the Android platform are mentioned as well.

Chen et. al. [12] deal with the aspects of media processing and playout performance of the mobile Android OS for digital media-ready end-devices (smartphones and tablets). Media processing capability is an important aspect, closely related to Section 5.3, which describes the characteristics of the custom Android application. Furthermore, the group treats the issue that DVB decoding and playout causes performance problems on Android. This is because the Android media framework does not natively support DVB as a protocol. Nevertheless, this is an important feature in a media environment where all end-devices shall work together effortlessly. Thus, the group has developed a solution to playout DVB-T content, by also considering multi-core and other hardware capacities of the device. First of all the group has identified how flows of DVB-T data are exactly passing the entire Android system and especially the media framework, using a TV tuner dongle. Later on, they modified the media framework to their needs. Additionally, a suitable driver for the DVB-T tuner devices was included in the Linux kernel. To evaluate their implementation, the group has tested the FPS rate and the related video buffer behaviour of different videos. As a conclusion, DVB is in general not suitable for low processor frequencies as it has a negative influence on real-time decoding speed. To solve this issue, modifications on the media framework are necessary. Details on the required changes can be found in the paper.

Moriyasu et. al. [14] have designed a multi-device synchronization testbed, specialized on synchronization of groups of user devices. Their target scenario is to connect groups accessing the same Video on demand (VoD) source via the Internet. Evaluation has been done on a PC based testbed. Additionally, they allow groups to use communication features at the same time to enrich the media experience. Just as the later described system, the group relies on the internal synchronization of the system wallclock of each involved synchronization member / end-device.

This is based on a host-client system, whereby clients adapt their wallclock to the one from the host. Therefore, SNTP [6] is used, a simplified version of NTP [5]. The NTP protocol is used in the presented approach as well, and explained in the course of this deliverable. Although they target an inter-destination scheme, their approach can be generally adapted to the local inter-device synchronization case. Once all wallclocks are synchronized, synchronization members exchange small messages to control the playout position of the current VoD stream on all devices. Different interactions between clients are defined, to coordinate the three commands play, pause and seek. To enable this, messages are exchanged in an ad-hoc / peer-to-peer fashion. The protocol introduced by Moriyasu et. al. considers the current latencies of the different clients and as an additional step, the hardware resources of the involved PCs.

The paper released by Köhnen et. al. [11] describes a WP4-internal work and is a contribution to high precision hybrid inter-media synchronization. The technical solution is tailored for HbbTV, enabling frame accurate synchronization on a single end-device for media from hybrid sources, like DVB and IP. Besides the approach itself, the work describes a testbed with hybrid sources and performed measurements. To enable hybrid inter-media synchronization, an absolute time code is needed for each media, as well as a reference between them. In the testbed, the broadcast signal is the main video source; therefore the absolute timeline needed for synchronization on the client side is inserted by the DVB broadcaster / DVB media encoder. On the IP-content side, an MPEG-DASH [21] ready source has been included. On broadcaster side, stuffing packets (Null packets) are identified in the MPEG-2 TS transport stream. Those Null packets in the transport stream closely placed to an I-frame, are replaced by a so called Broadcast Timeline packet, which contains the absolute timecode value of the related I-frame (based on the Presentation Time Stamp (PTS) [7] of the I-frame). So the timeline is already included in the transport stream before it is sent via the satellite transponder. On the client side, the timeline decoding is trivial and simple for implementers. The client’s DVB stack receives the custom time line and can use the absolute time codes. A second media streamed via IP (MPEG-DASH) is synchronized to the timeline during playback. In the same way, subtitles are synchronized with the DVB stream.

It can be claimed for our solution, that in a TS stream, the earlier a subsequent stuffing packet appears after a Packetized Elementary Stream (PES) header [7] containing an I-frame, the more accurate the synchronization process will be afterwards. The second article on inter-media synchronization by Beloqui et. al. [13] is also based on an HbbTV system. As a consumer scenario, the group has chosen to combine a live IP-TV channel with a broadcast radio channel. In accordance with the previously mentioned HBB-NEXT paper [11], different media streams need to have access to the same wall-clock on the receiver side, where both hybrid streams come together. In short, the group's approach aims to create a reference between the wallclock of the receiver device and the PTS timestamps of the DVB stream, by including PTS values in the DVB EIT [7] as an extra field.

3. HBB-NEXT System Architecture

3.1. Introduction

This section introduces the high-level system architecture that has been provided by WP6 and shows where WP4 contributes to it.

3.2. Common HBB-NEXT Architecture

One of the major objectives of HBB-NEXT is to define an open framework for the next generation of hybrid broadcast internet (HBI or HBB) services. Building on top of deployed standards like HbbTV, HBB-NEXT defines a set of extensions and open APIs to enable services that are more advanced and business cases. Please refer to D2.4 for more details on business models.

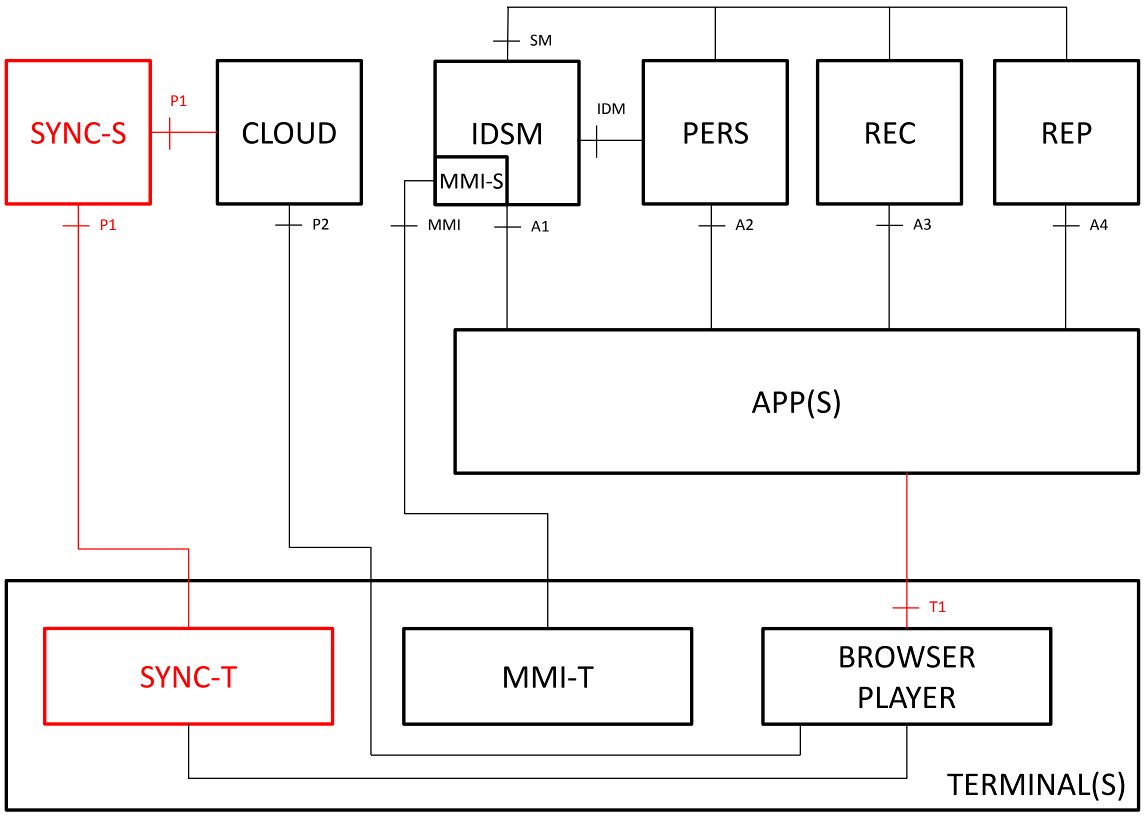


Figure 1: Inter Device Synchronization in the HBB-NEXT system architecture

Figure 1 shows the system architecture of the HBB-NEXT platform as developed in WP6. The diagram consists of enablers provided by WP3/4/5, which reside either on the internet (CLOUD) or on the terminal, and public interfaces to the HBB-NEXT applications (APPS). Interfaces in the diagram can be distinguished into APIs and protocols. APIs of internet-based enablers are potentially exposed by one or multiple service provider, named A1, A2 etc. The API that is exposed to applications on the terminal is named T1. HBB-NEXT also defines open protocols for the advanced synchronization features. These protocol interfaces are named P1 and P2. APIs between enablers are labelled with the name of the sub module that exposes it, MMI, IDM and SM.

3.3. WP4 and Inter-Device Synchronization

WP4 contributes to the system architecture the enablers for inter-media (multi-stream) synchronization, inter-device (multi-screen) synchronization and cloud offloading. This deliverable focuses on the inter-device synchronization. The enablers and their interfaces, relevant for inter-device synchronization are highlighted in red.

The synchronization enabler is split into a terminal part SYNC-T and a service provider part SYNC-S. Protocols and signaling are defined at the interface P1 (see Figure 1). These protocols include mechanisms to synchronize the clocks at multiple devices, streaming protocols and content timelines for a synchronized presentation.

At the terminal, the sync enabler exposes functionality to HbbTV applications at the interface T1. This includes starting streams and joining sessions in which content presentation is synchronized. The detailed description of these interfaces can be found in chapter 4.2.

4. Inter-Device Synchronization

4.1. Introduction

The project partners investigated in the research field of technical inter-device synchronization in the background of a media environment. Unlike previous developments within HBB-NEXT in the field of inter-media synchronization, in which two independent approaches (see HBB-NEXT Deliverable D4.2 [4])have been investigated in, findings picked up for inter-device synchronization have been merged into one single synchronization scheme, which has been implemented on various heterogeneous platforms. The general synchronization method however is consequently the same on all involved platforms: Timing information is constantly exchanged between devices, so devices which attempt to play out on-demand digital media content in a synchronized manner, will control the play out position and play out rate of this media in order to keep all media streams in sync.

The approach covers initial device discovery in local networks and subsequent connection establishment and management for further exchange of synchronization messages. A light-weight set of synchronization messages has been designed to carry necessary information for inter-device media synchronization.

The following sections provide the reader with a detailed look on the various aspects of the synchronization process.

4.2. Approach

4.2.1. Introduction

The essence of inter-device synchronization is not that different from other forms of synchronization; the basic process is making sure a certain content element (e.g. a video frame or audio sample) is played out at a certain point in time. In other words, matching the content timeline to the wallclock timeline (Figure 2).

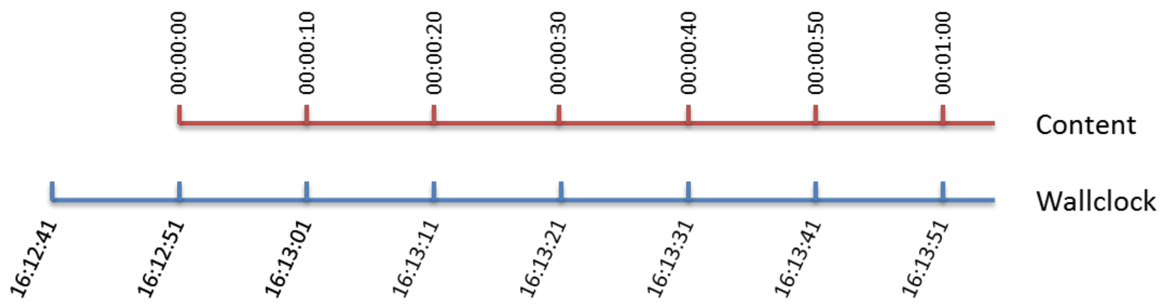


Figure 2: Wallclock timeline

The main difference between inter-media synchronization and inter-device synchronization is that with inter-device synchronization the to-be-synchronized content elements are played out on two separate devices, each with its own media playout engine. The basics of inter-device synchronization are therefore communicating how a certain device has mapped the content timeline to the wallclock timeline and replicating this mapping on other devices.

In an ideal world, where clock drift is not a problem and playback can continue uninterrupted, all that is necessary for two devices to synchronize their playout is to communicate one tuple containing the mapping between the content timeline and the wallclock timeline at one moment in time (i.e. contentTime@wallclockTime, compare Figure 3). Using extrapolation, devices are then able to calculate the target playout time for all other video frames and audio samples that make up the content that is to be synchronized. In actual applications, due to clock drift and possible playback interruptions (e.g. due to network issues), it is necessary to exchange such synchronization messages more frequently.

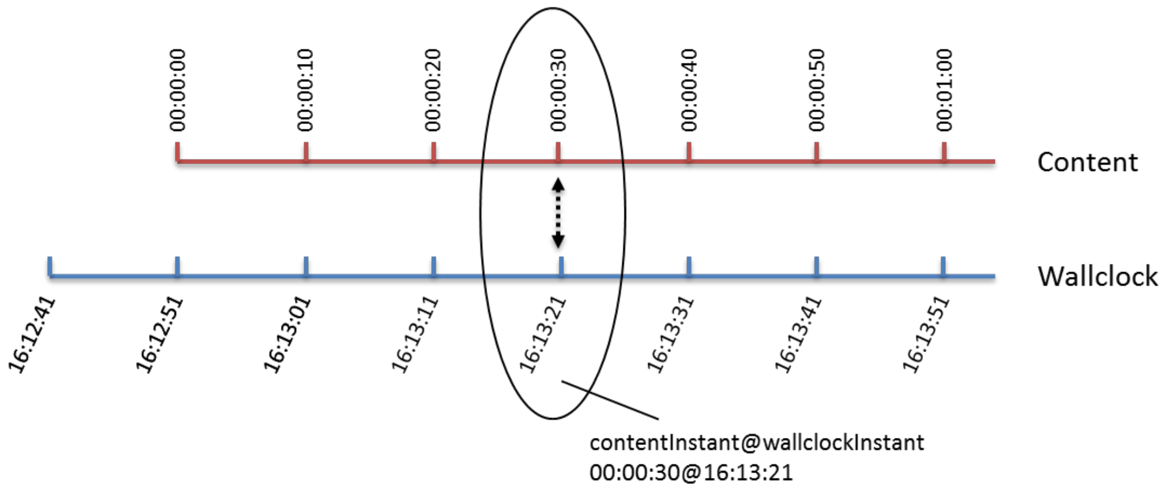


Figure 3: Content time and wallclock time

Within the HBB-Next project, it has been decided to develop a relatively simple and low overhead inter-device synchronization solution. A wide variety of different devices can be expected to be used for inter-device synchronization with HbbTV (e.g. STBs, TVs, smartphones, tablets, etc.). Furthermore, these devices do not share a common platform or set of APIs. For these reasons, it is important that the developed inter-device synchronization solution is relatively simple to implement and does not require platform-specific or complex protocols that might not be available on all platforms. For this reason, most of the elements of inter-device synchronization presented in this document are based on well-established and widely implemented protocols.

The decision was made to go for a master-slave type of inter-device synchronization mechanism, with one device in any given synchronization group playing the role of synchronization master. This main purpose of the synchronization master is to send synchronization updates to all devices making up the synchronization group. Despite the fact that in the HbbTV context, the TV or STB plays a central role, the chosen inter-device synchronization solution allows for any device in the synchronization group to play the role of synchronization master. Section 4.2.4 discusses in more detail the various roles in the developed inter-device synchronization solution, and also discusses the actual exchange of synchronization messages and their format.

In practical situations, especially with a heterogeneous and diverse set of devices making up a synchronization group, it cannot be assumed that the wallclocks of these devices are synchronized. Section 4.2.2 discusses this problem and provides a number of solutions.

Section 4.2.3 finally discusses various options for service and device discovery.

4.2.2. Clock Synchronization

One very important aspect of inter-device synchronization is clock synchronization. As exchanged sync-information refers to a moment in time, say a snapshot from the system time, the system clock on all synchronization peers needs to operate on the same time. A couple of options are available to achieve clock synchronization over network. The most popular kind is clock synchronization via the Network Time Protocol [5]. Alternatives are e.g., the Precision Time Protocol or other proprietary/custom protocols.

The Precision Time Protocol (PTP [10]) is a high-accuracy protocol for synchronization of clocks via network, which achieves accuracy in the sub-microsecond range. PTP operates on a peer-to-peer mode without the need for further infrastructure. It offers a negotiation process with the election of a grandmaster, say a reference clock host, as a result. After election, peers will connect directly to it and synchronize their clocks based on a master-slave model.

NTP is a widely known server centric networking protocol for clock synchronization with easy accessible implementations existing on a huge number of platforms. The actual clock synchronization algorithm operates on exchanging timestamps, computing time offsets according to round-trip delay times between server and client and adjusting the system time. However, in case of asymmetry in nominal delay on incoming and outgoing routes between client and server, calculated time offsets have systematic bias of half the difference between the forward and backward travel times [8]. Nevertheless, especially in local networks, NTP achieves clock offsets within a few milliseconds range or also with very good network conditions, offsets of several hundred microseconds [9]. NTP operates on a client/server model. The need for an available NTP server infrastructure may be a disadvantage for peer-to-peer synchronization models.

One major reason for choosing NTP is its popularity and wide-spread availability on client-devices. It is available on most Linux systems nowadays and can also be found on already existing network infrastructure, like routers in home networks. Other protocols like PTP might achieve higher accuracy on time synchronization over network, yet inter-device media-synchronization requires only frame level synchronicity, with drifts of about 20 milliseconds. With NTPs ability to achieve synchronicity of very few milliseconds, it offers a more than sufficient performance for inter-device synchronization.

4.2.3. Device/Service Discovery and Pairing

Synchronization is considered a media service, offered via network. Hence if synchronization peers want to establish a synchronization session, a step called 'Service discovery' needs to be executed in advance. Service discovery allows automatic detection of devices and their respective shared services in networks. A number of different popular approaches and protocols are available, SSDP [18], mDNS [19]or DHCP [20] to name a few. All vary in complexity, network layers and scope of application. For the given prototypes, two methods were elaborated and integrated. On one side, the peer-to-peer and mDNS protocol is being used. It serves as easy implementable solution and operates on exchange of multicast and broadcast UDP packets in local networks. On the other side, a proprietary QR-code will be integrated in further versions of the prototype. Here we deal with a user-friendly, infrastructure supported device discovery, with the ability to establish and maintain sessions, as well as exchange of messages, via an intermediate server. Discovery and device pairing is executed explicitly, by actively scanning a QR code on a mobile device, and not happening in the background.

mDNS/Bonjour

In most inter-device synchronization scenarios, all devices making up a synchronization group will be at the same location, and part of the same network. An example of such a scenario is a user synchronizing his tablet and/or his smartphone with his TV. In the case where all devices are part of the same IP subnet (i.e. location on the same local WiFi network), a very easy to use and simple method to perform device and service discovery is the combination of mDNS and DNS-SD (together also known under the Bonjour[2] name).

By using Bonjour devices can automatically broadcast their willingness and ability to perform inter-device synchronization. As an example, once an STB is turned on, it can start broadcasting a DNS record indicating that it is willing to synchronize with other devices. Any other device on the same subnet that is actively searching for devices to synchronize with will receive the advertisement. After resolving the DNS record back to an IP address, the second device can connect to the STB and the synchronization mechanism can start. In its most basic form, device discovery and pairing via Bonjour does not require any user input, and can be a fully automated process. Of course, in the situation where multiple synchronization groups are available on the same subnet (e.g. two STBs tuned to two different channels), it would require input from a user to decide which group (and device) to join.

The downside of using Bonjour is that it only works with devices on the same subnet. Since this will probably be the case in many of the inter-device synchronization scenarios envisioned by the HBB-Next project, it will not work in all circumstances. Apart from inter-destination synchronization use cases, in which devices will obviously not be in the same subnet, another limitation is that Bonjour does not work for devices that connect to the internet over e.g. a 3G network (a smartphone which has WiFi turned off). In these scenarios, another device discovery and pairing mechanism, such as the QR code-based mechanism explained in the next section, is more suitable.

[2] https://developer.apple.com/bonjour/

Implementation Considerations

The inter-device synchronization solution described in this document is advertised with the '_hbbInterDeviceSync._udp.' service name. This means that any device that implements the synchronization master mechanism, described in section 4.2.4, is broadcasting the '_hbbInterDeviceSync._udp.' service. At the same time, any device that wants to join an existing synchronization group will search for the same '_hbbInterDeviceSync._udp.' service name.

In some use cases more information is necessary to be able to decide whether to join a given synchronization master (and thus group). For example, more than one synchronization master might be advertising the inter-device synchronization service (e.g. the user has two STBs in his home). For this purpose, the Bonjour device name property can be used. By advertising the device name and showing this name prominently in the UI, it should be immediately obvious to the user which device it is. An example device name could for example be 'Living Room STB'.

It should be noted that the particular content item that is being watched by the synchronization master is explicitly not being broadcasted as part of the service discovery and pairing process but part of the message exchange that follows (see Section 4.2.4.5). It is assumed that based on the device name the user has enough information to be able to decide which synchronization master to join.

The QR-Code based '2nd Screen framework'

IRT has developed a framework which enables independent integration of second screens in HbbTV applications. In a HbbTV application running on a set-top-box, a QR code will popup which can be scanned by any device supporting a bar-code-scanner (e.g., Android/iOS tablets). The 2nd screen framework provides functional components on a server which can be accessed via JavaScript API. An API call to that server will generate a session – this session identifier is contained in the information within the QR-code. Via that session, all connected devices, which receive a unique identifier each, can push messages to each other. Substantial API queries for sending messages and polling the server for new incoming messages are available.

4.2.4. Synchronization Protocol

4.2.4.1. General

After successful discovery and pairing, synchronization peers know IP addresses and port numbers of the involved sync services and will establish a connection. The proposed synchronization model operates on the packed oriented protocol UDP (User Datagram Protocol). Each synchronization client maintains an open UDP port, thus every client can act as server as well. All subsequent synchronization messages will be transported via this port.

The approach implements a master-slave model (analog to Publish-Subscribe patterns). Clients will send a subscribe message to the synchronization master and receive sync information. On the server side, a subscription remains in a soft state, in this case, in form of a timeout value, until client-subscriptions expire. The synchronization host will notify clients with synchronization information in regular intervals, e.g. every 5 seconds. If a specific timeout expired, clients will no longer be notified unless they re-subscribe to refresh their timeout values.

4.2.4.2. Message Format

The proposed message format is lightweight and offers a clearly laid out functionality. A message is a text-encoded list with an arbitrary number of lines. Each line contains one tuple of information, delimited by a colon.

See the sample of a line:

```
<PARAMATER_KEY>: <VALUE>\r\n
```

Each line is terminated with the ASCII codes for carriage-return and line-feed ('\r\n'). Basically the message format resembles the structure of an HTTP-header. It provides high-readability for debugging processes and is easy to en- or decode.

4.2.4.3. Message Types

A number of fixed message parameters have been designed. In the Listing 1 below the number of designed default parameters are listed. Although all parameters are string based, the string will still be subject of a specific format:

```
typedef enum MESSAGE_TYPE {
MESSAGE_TYPE_SYNC = 0,
MESSAGE_TYPE_PAUSE = 1,
MESSAGE_TYPE_JOIN = 2,
MESSAGE_TYPE_QUIT = 3,
MESSAGE_TYPE_DROP = 4,
MESSAGE_TYPE_FAIL = 5,
} MESSAGE_TYPE
```

Listing 1: Message-Types

For basic synchronization, only two types of messages need to be exchanged, which are marked yellow in the listing above. As previously mentioned, base of the synchronization scheme is a client subscribing a media-session by sending the particular joining message to the host. This message is represented by the MESSAGE_TYPE_JOIN. The message does not require any further parameters, except for an optional Device name or similar information. QUIT serves as the correspondent message-type to actively leave a synchronization session and keep the host from sending further sync-packets to the client. A synchronization client will send a FAIL message, in case he is not able to synchronize (e.g. through bad network conditions), and finally a host can issue DROP towards a client to terminate this connection.

The messages SYNC AND PAUSE will manage media-playback. SYNC contains timestamp information for synchronization. Typically, a host will issue a SYNC message when a client joins a session, and also regularly to keep all clients synchronized. PAUSE is designed to manage media-playback in a distributed control manner.

4.2.4.4. Message Parameters

A couple of fixed message parameters have been designed. But in general, a line and text-based protocol like this can support an arbitrary number of custom parameters, which can be attached to the message for custom reasons. In Table 1, a number of designed default parameters are listed. Although all parameters are string based, the string will still be subject of a specific format:

Parameter Name	Format	Description
DEVICE_ID	String	Device Name / ID
PLAYPOSITION	Unsigned Integer	Number of milliseconds that passed since playback of the media event started
TIMESTAMP	YYYY/MM/DD;HH:MM:SS:sss	Absolute Wallclock date-time
MEDIA	URL	Address of media-file/stream to be played
MIME-TYPE	String	Internet-Media-Type
MESSAGE-TYPE	Enum	See 4.2.4.3
SESSION_ID	String	Media-Sessions can be identified via this parameter
TIMEOUT	Integer	Amount of seconds that indicates the period after which a client gets dropped from the sync session
NTP-SERVER	String	Address of NTP-server

Table 1: Message Parameters

4.2.4.5. Message Flow

The message flow has been kept as simple as possible. Basically, the synchronization host will provide all connected synchronization clients with synchronization information in a given interval. In case of a client joining the session, another single event-based synchronization message will be distributed, so the client does not have to wait for the next interval period to pass. Clients remain in a soft-state and will automatically drop from the session after a specific interval unless they re-send their JOIN message to refresh the timeout. Below, find a sample of a short media session with one host and two clients. To keep the image simple, only the message type and a message number in brackets are given. Find the messages in Table 2 for a more detailed look.

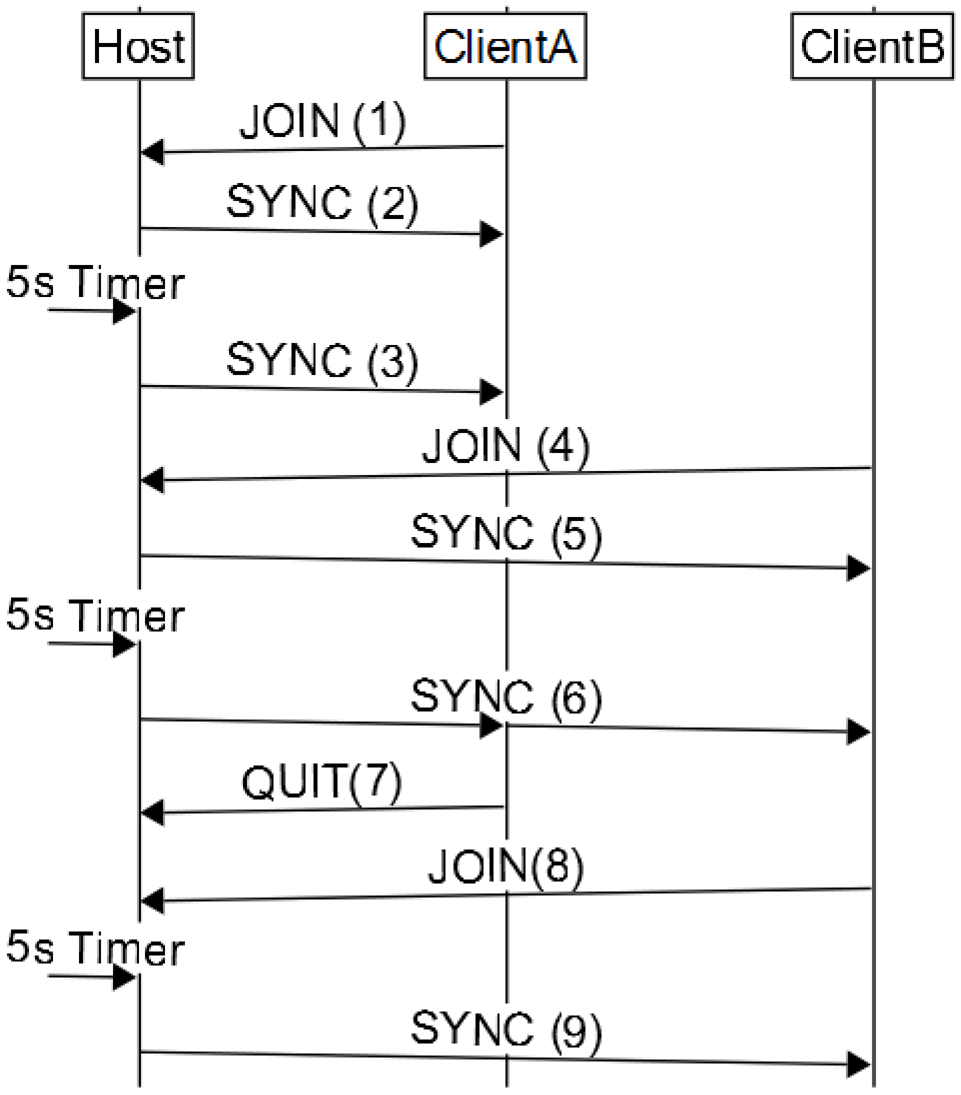


Figure 4 : Message Sequence Chart of a synchronization session

Message No	Message	Description
1	MESSAGE_TYPE: JOIN DEVICE_ID: ClientA	ClientA subscribes the current media Session
2	MESSAGE_TYPE: PLAY DEVICE_ID: HOST TIMEOUT: 300 PLAYPOSITION: 3000 TIMESTAMP: 01/01/1970;17:32:59:148 MEDIA: http://server.com/video.mp4	Host replies instantly with a synchronization message, containing all necessary information for sync and a timeout value

Message No	Message	Description
3	MESSAGE_TYPE: PLAY DEVICE_ID: HOST TIMEOUT: 295 PLAYPOSITION: 8000 TIMESTAMP: 01/01/1970;17:33:04:148 MEDIA: http://server.com/video.mp4	After 5 seconds, the next sync info gets distributed to keep clients in sync, timestamps and timeouts changed
4	MESSAGE_TYPE: JOIN DEVICE_ID: ClientB	ClientB subscribes the current media session...
5	MESSAGE_TYPE: PLAY DEVICE_ID: HOST TIMEOUT: 300 PLAYPOSITION: 9132 TIMESTAMP: 01/01/1970;17:33:05:280 MEDIA: http://server.com/video.mp4	...and when a client joins, the server replies with current media information
6	MESSAGE_TYPE: PLAY DEVICE_ID: HOST TIMEOUT: <client timeout> PLAYPOSITION: 13000 TIMESTAMP: 01/01/1970;17:33:09:148 MEDIA: http://server.com/video.mp4	The next regular interval hits, and both connected peers receive a sync message, but with different timeouts.
7	MESSAGE_TYPE: QUIT DEVICE_ID: ClientA	ClientA decides to drop from the session...
8	MESSAGE_TYPE: JOIN DEVICE_ID: ClientB	..while ClientB decides to refresh the session timeout and remain in the session
9	MESSAGE_TYPE: PLAY DEVICE_ID: HOST TIMEOUT: 300 PLAYPOSITION: 18000 TIMESTAMP: 01/01/1970;17:33:14:148 MEDIA: http://server.com/video.mp4	And as the next interval hits, only ClientB will receive sync messages, yet with the timeout restored to 300 seconds

Table 2: Tabular listing of messages within the synchronization session

4.3. Conclusions

To make inter-device synchronization possible for a wide spectrum of devices, information about play position and wallclock time must be exchanged between devices.

To make this possible the project partners decided to use existing, well established protocols, which are available on usual second-screen devices like mobile phones or tables.

Thus different devices have different time settings, their wallclock must be synchronized. To compute the devices local offset to a master clock, NTP is used. A master clock device is available on almost every home network in form of the local router.

Mobile device clocks are inclined to drift very fast, so synchronization information must be exchanged periodically.

To have devices indicate their ability for synchronization on local networks, mDNS is used. It is easily accessible from all used devices and enables automatic service discovery and joining. For more complex scenarios with multiple synchronization groups in local networks or inter-destination synchronization, the QR-Code framework can be used.

Finally the actual exchange of synchronization information is achieved by a light-weight UDP based protocol containing fundamental messages for media playback.

5. Device Implementations

5.1. Introduction

After treating the general concept and theoretical functionalities of inter-device synchronization in Section 4, this Section elaborates on the developed implementations on each end-device. Representative Subsections are included for the different device categories, each treated separately: Set-Top-Box (STB), Android tablet / smartphone, Apple iOS tablet / smartphone. In the case of inter-media synchronization in HBB-NEXT, concrete software modules for the used end-devices are explained in each Subsection. Additional Subsections treat the powerful Gstreamer Synchronization Framework and the related implementations in HBB-NEXT, as well as components developed to enable synchronization in the cloud.

Later on, Section 6 will describe the composition and interplay of the devices in multiple, concrete demo setups.

5.2. Set-Top-Box

HBB-NEXT partners are using a Broadcom STB in combination with sources of the Inaris DVB/HbbTV middleware. This allows implementing HbbTV extensions and synchronization features on a real hybrid STB device.

5.2.1. Placement of Application in System Architecture (App in OS/API)

Based on the Linux OS, "Inaris" is a modular middleware solution for DVB reception and recording, including popular features like HbbTV reception, EPG, Teletext and Subtitling. Based on long-term expertise in European DVB and sophisticated design and test processes, Inaris DVB Middleware has been developed to provide a flexible, sustainable and robust software solution to manufacturers of DVB receiver products. Additionally, the HbbTV Receiver, the Teletext Decoder and the DVB Subtitle Decoder are available as additional modules to the Inaris DVB Middleware. The Hardware platform is based on Broadcom's SoC BCM7342, being the core chip of the set-top-box.

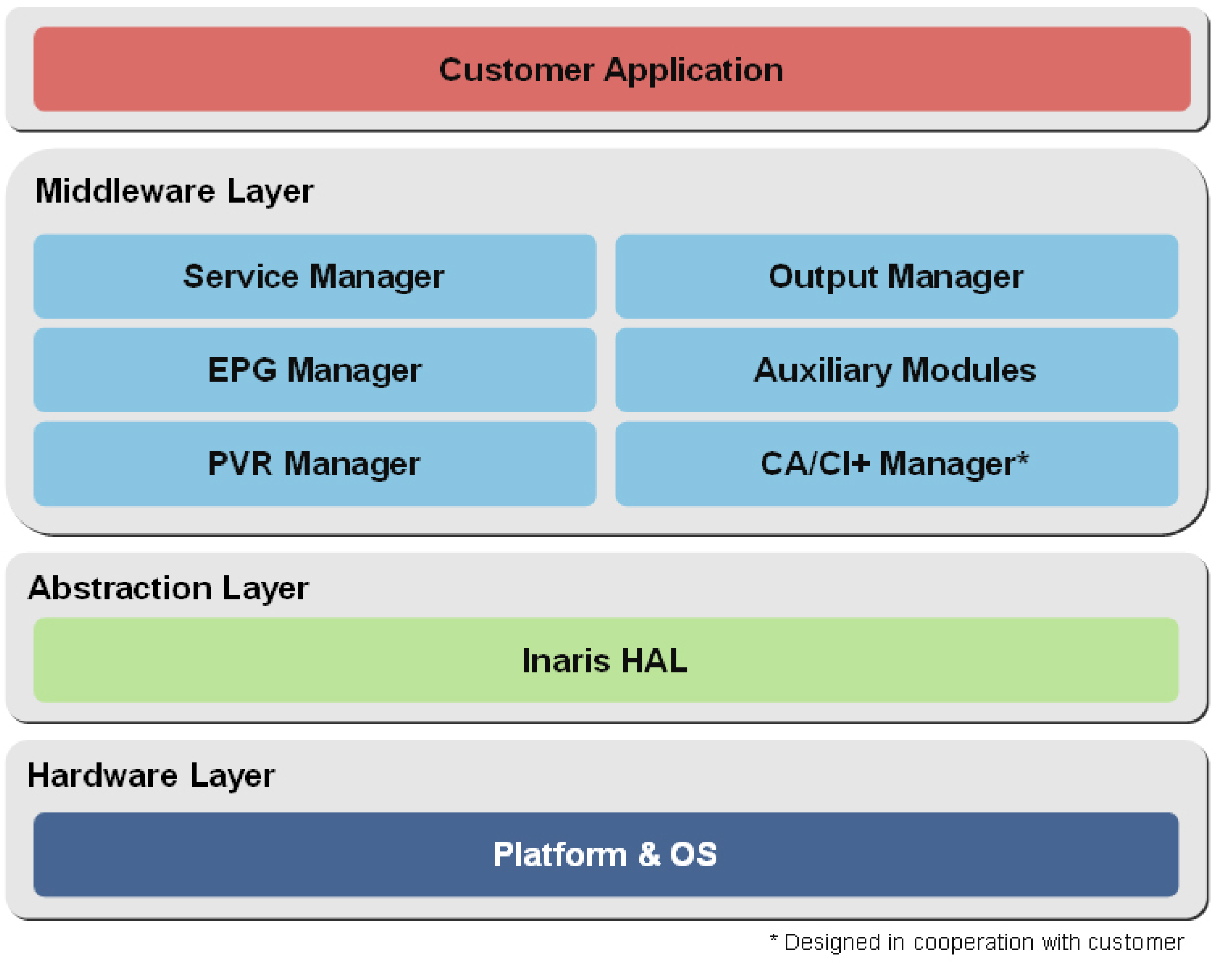


Figure 5: Inaris DVB Middleware Architectural Overview

As depicted in Figure 5, the Platform & OS environment is a custom Linux distribution, delivered by Broadcom. It contains the kernel, hardware modules, libraries and user space tools to control and access the hardware features of the STB.

The Inaris HAL component is a software module to provide a hardware abstraction for the required multimedia and OS functionality to the Middleware Layer.

The Inaris Middleware Layer implements the DVB functionality to access, process and display the DVB content. As additional middleware components, the HbbTV Receiver provides all functionality to run HbbTV applications. As browser module the Opera Embedded Browser is used.

The Customer Application layer is the user front end. Here, a version created with the Embedded Wizard is used.

The HBB-NEXT applications run in the browser, same as HbbTV applications do.

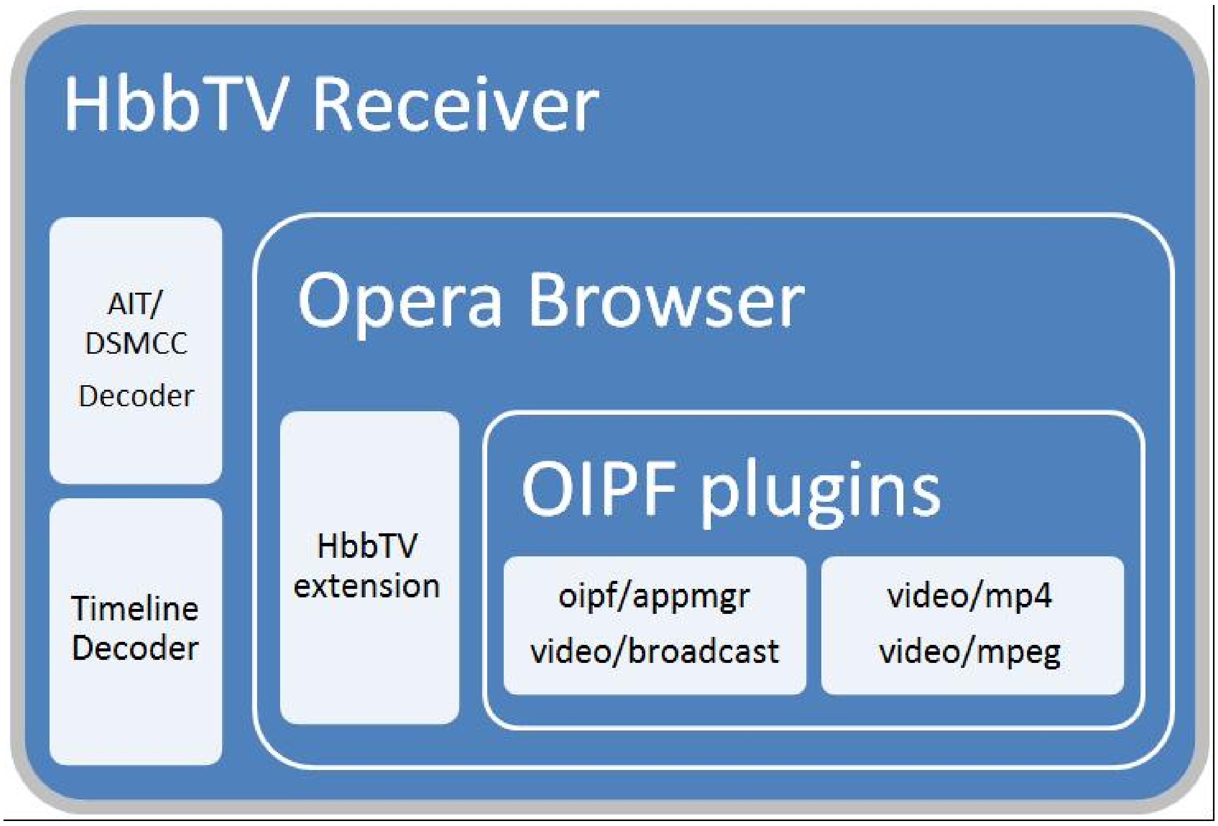


Figure 6: Inaris HbbTV Receiver Architecture

The HBB-NEXT API extensions were implemented into the video/broadcast and the video/mp4 plugin objects.

5.2.2. Application Architecture

The HBB-NEXT extensions to the HbbTV Receiver module are mainly implemented within the video/broadcast module. As depicted in Figure 7, the API OIPF API has been extended to provide a new defined eventPlayPosition property. Once called, it returns the current timeline tick value in milliseconds.

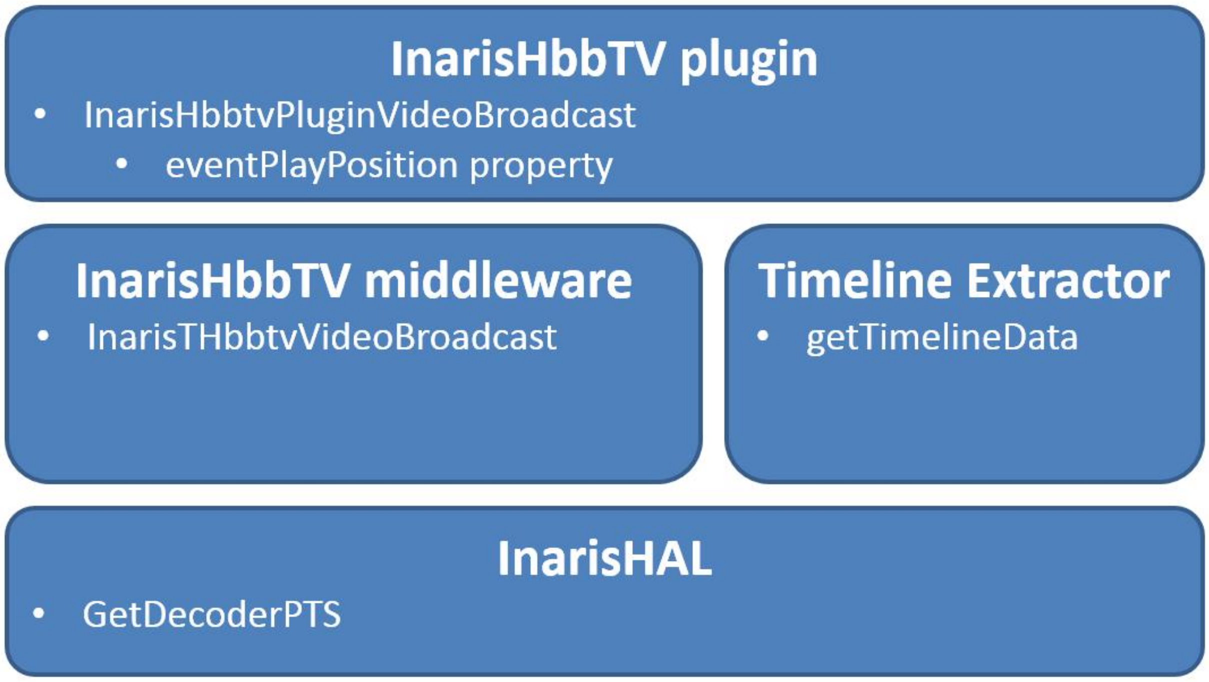


Figure 7: Inaris HbbTV plugin Architecture

The timeline tick is retrieved and interpolated by the Inaris HbbTV middleware from the Timeline Extractor module. The interpolation and video synchronization of timeline ticks is performed using the decoder's PTS values. The last timeline packet's PTS is compared to the current decoder's PTS value. The difference value is used to calculate the corresponding correct timeline tick value.

For enabling the Picture in Picture (PiP) feature for inter-media video synchronization, the video/mp4 plugin was modified to accept a sink="pip" attribute in the HTML object definition. This triggers the video object to not disable the video/broadcast and to render windowless in parallel.

5.2.3. Functionality

The extensions to the Inaris HbbTV Receiver provide the functionality of inter-media and inter-device synchronization. These requires the features of DVB Timeline decoding, Picture –in-Picture rendering of video/broadcast and video/mp4 in parallel.

5.2.4. Installation/Setup

The extensions are all fully integrated into the HBB-NEXT-branch of the Inaris Middleware sources. Once the binaries and libraries are installed, the STB is enabled to provide all functionality to any HbbTV/HBB-NEXT application, running in the browser. No further setup is needed. The implementation provides advanced HbbTV API to HbbTV browser applications.

5.3. Android Tablet

5.3.1. Placement of Application in System Architecture (App in OS/API)

The App is placed inside Androids Java Layer, as shown in Figure 8. From there it makes use of the Android SDK Modules *MediaPlayer* and *AudioManager*.

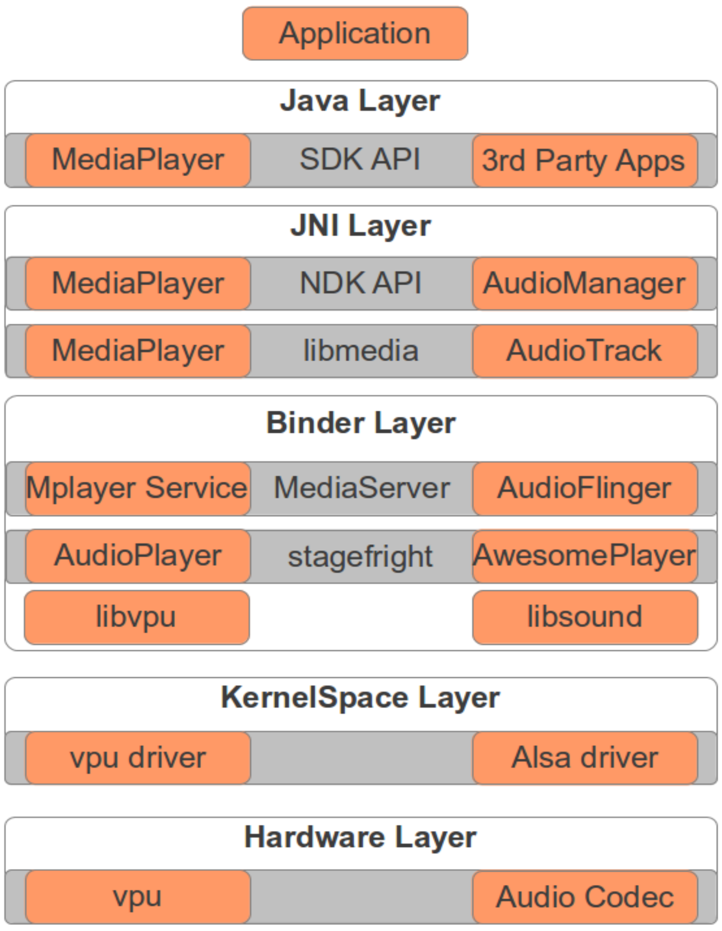


Figure 8: Placement of App in Android System Architecture

These modules are wrappers for the corresponding modules from Androids NDK. These Modules are provided in native machine-code and have direct control over lower layer frameworks like the *libmedia* framework. This framework called *stagefright* is Androids new playback engine since Android 2.0. It provides codecs, parsers, muxes and encoders for all common media formats, which are capsulated inside *AwesomePlayer* for video data or *AudioPlayer* for audio data.

libvpu and *libsound* provide access to specific hardware drivers needed for media decoding and playout.

The *KernelSpace* Layer provides drivers to communicate with different hardware modules like *VideoProcessingUnit* and *AudioCodes* which are located inside Hardware Layer.

Each layer uses only functionality provided by lower layers.

5.3.2. Application Architecture

The Application is separated into 4 Main modules, *UserInterface, MediaPlayerContainer MediaControlAP*I and *SynchronizationControl*. This is depicted in Figure 9.

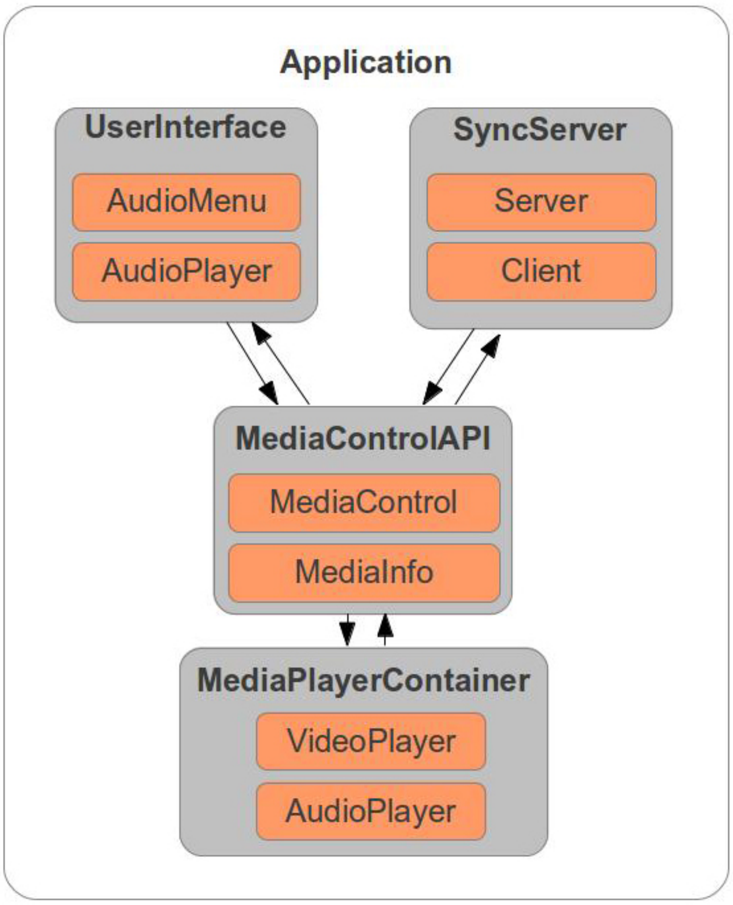


Figure 9: Interplay of Application Components

The *MediaPlayerContainer* module provides functionality for playing out common media formats from any location.

In case of multiple media files being played (video together with sign language interpreter, video with different audio language), multiple instances of *MediaPlayer* are linked together to play synchronous.

As the *MediaCodec* from a lower Layer is only available once, Callbacks into the Application (OnSeekComplete, OnPrepared, etc.) come from one asynchronous thread and any action in these callbacks should take as less amount of time as possible and must so be performed from inside a different thread. This makes it possible to have multiple instances playing out frame accurate without tweaking lower layers of the android framework.

The *MediaControlAPI* module gives direct high level control to media files. Common controls like PLAY, PAUSE, STOP or SEEK are provided to control all *MediaPlayer* instances hosted inside *MediaPlayerContainer*. These controls are located inside the *MediaControl* Module. It also provides information about all *MediaPlayer* instances at once, to read for example the synchronized *PlayPosition* of linked media files.

The *MediaControlAPI* is meant to be used by upper layer modules like *SyncronizationServer* or *UserInterface*.

The *UserInterface* provides control-elements for handling Synchronization and selection of media content. All HBB-NEXT settings are performed inside the module to give functionality of HBB-NEXT-Settings-APP in tablet-conform design rules. It makes direct use of the *MediaControlAPI* to control playout.

The Synchronization Control module handles device discovery and acts as Server and Client for media synchronization.

When acting as Synchronization-Master-Device (eg. Server), it provides functionality for adding devices to a synchronization session and periodically informing all clients about the actual *PlayPosition*, provided by the *MediaInformation* module.

When acting as Synchronization-Slave-Device (eg. Client), it controls the playout of local media when receiving Synchronization-Information *from MediaControlAPI*.

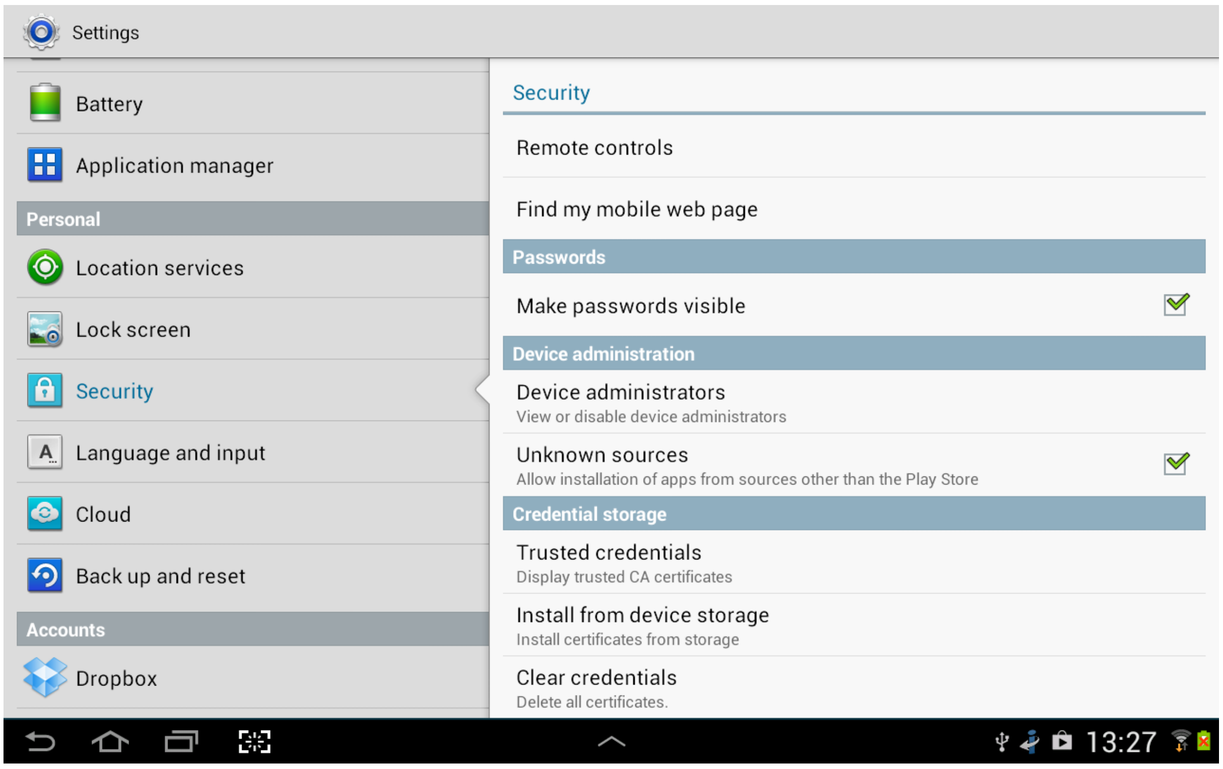
5.3.3. Functionality

The Android Application provides functionality for playing media-files, as well as hosting or joining Synchronization-Sessions inside the HBB-NEXT Framework.

The main functionality is mirrored from HBB-NEXT-Settings Application. The application can act as a master or slave device in an inter-device synchronization session, as well as providing inter-media synchronization for Sign language Interpreter (PIP), different languages (audio tracks), audio description or subtitles.

5.3.5. Installation/Setup

The Application is provided as .apk file, an Android specific Application Format. It must be made sure, that the regarding Android device allows installing applications from unknown sources. Figure 10 depicts how to enable this setting in the Android OS.



Settings

Battery
Application manager
Personal
Location services
Lock screen
Security
Language and input
Cloud
Back up and reset
Accounts
Dropbox

Security
Remote controls
Find my mobile web page
Passwords
Make passwords visible ✓
Device administration
Device administrators
View or disable device administrators
Unknown sources ✓
Allow installation of apps from sources other than the Play Store
Credential storage
Trusted credentials
Display trusted CA certificates
Install from device storage
Install certificates from storage
Clear credentials
Delete all certificates.

13:27

Figure 10: Required Setting for Installation (Example Device)

The media content for the application must be located in /sdcard/hbbtv/media

5.4. iOS Tablet

As part of the development process, the full inter-device synchronization solution has also been implemented on iOS, to allow iPhones and iPads to be synchronized either with each other or with one of the other platforms discussed in this chapter.

5.4.1. Placement of Application in System Architecture (App in OS/API)

In general, when developing a synchronization solution for an embedded platform, be it a tablet, smartphone or STB, the most difficult aspect is getting access to the lower level APIs and hardware that are necessary in order to have the required level of control of the video playback. iOS is no different in that regard.

Getting a video to playback on iOS is easy to achieve using the high level APIs, and seeking within that video relatively accurately is quite simple to achieve as well. However, such high level APIs do not provide enough control and accuracy to allow for frame accurate synchronization that some of the HBB-Next use cases call for. In order to really have frame-accurate control over video playback, one has to use the low level Core Media APIs (see below for an overview of the iOS media stack).

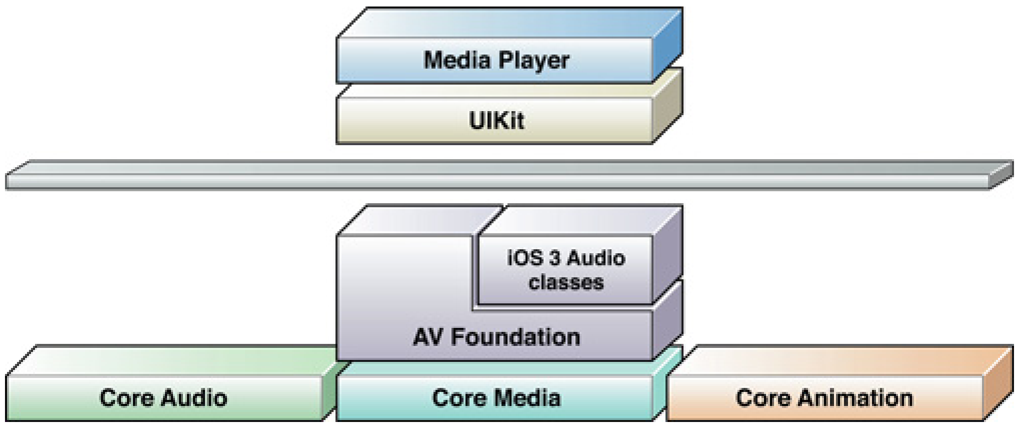


Figure 11: iOS Media API Layers

Using the Core Media APIs (specifically, the associated Core Video API), one can basically create a complete video renderer, with direct control over the video decoding and playback timing. Using such a renderer, it is possible to achieve very tight synchronization resulting and the possibility of controlling the playout of video frames with millisecond accuracy.

5.4.2. Application Architecture

The diagram below shows an architecture overview of the current iOS application.

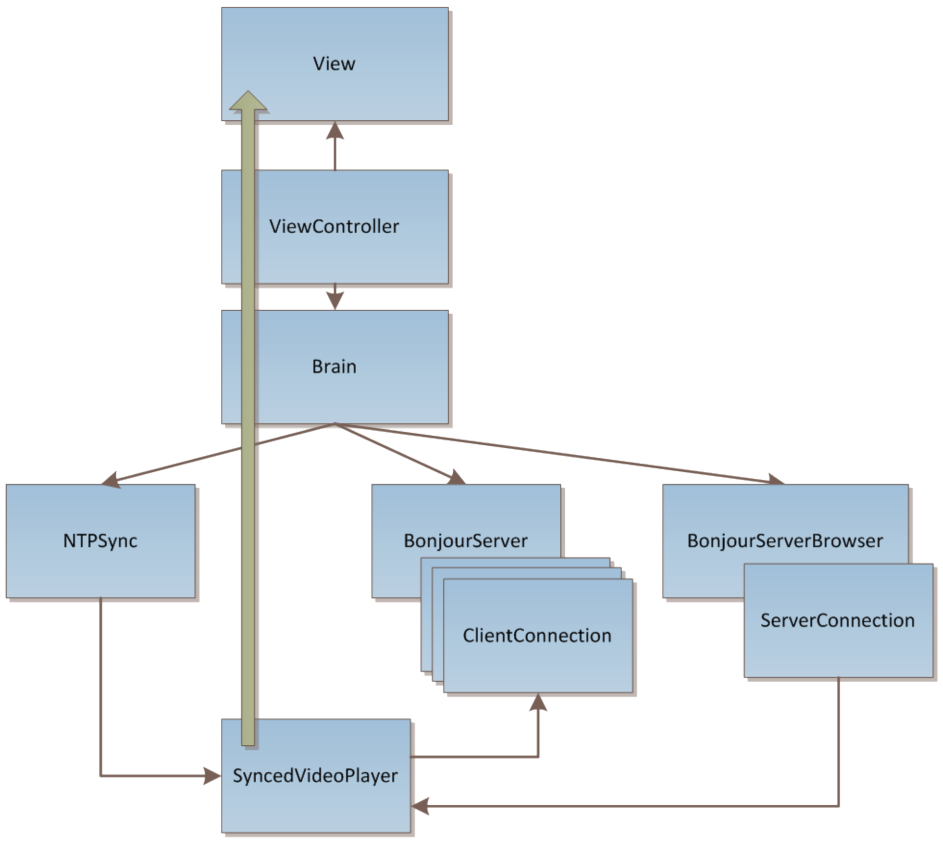


Figure 12: Architecture diagram of iOS inter-device synchronization app

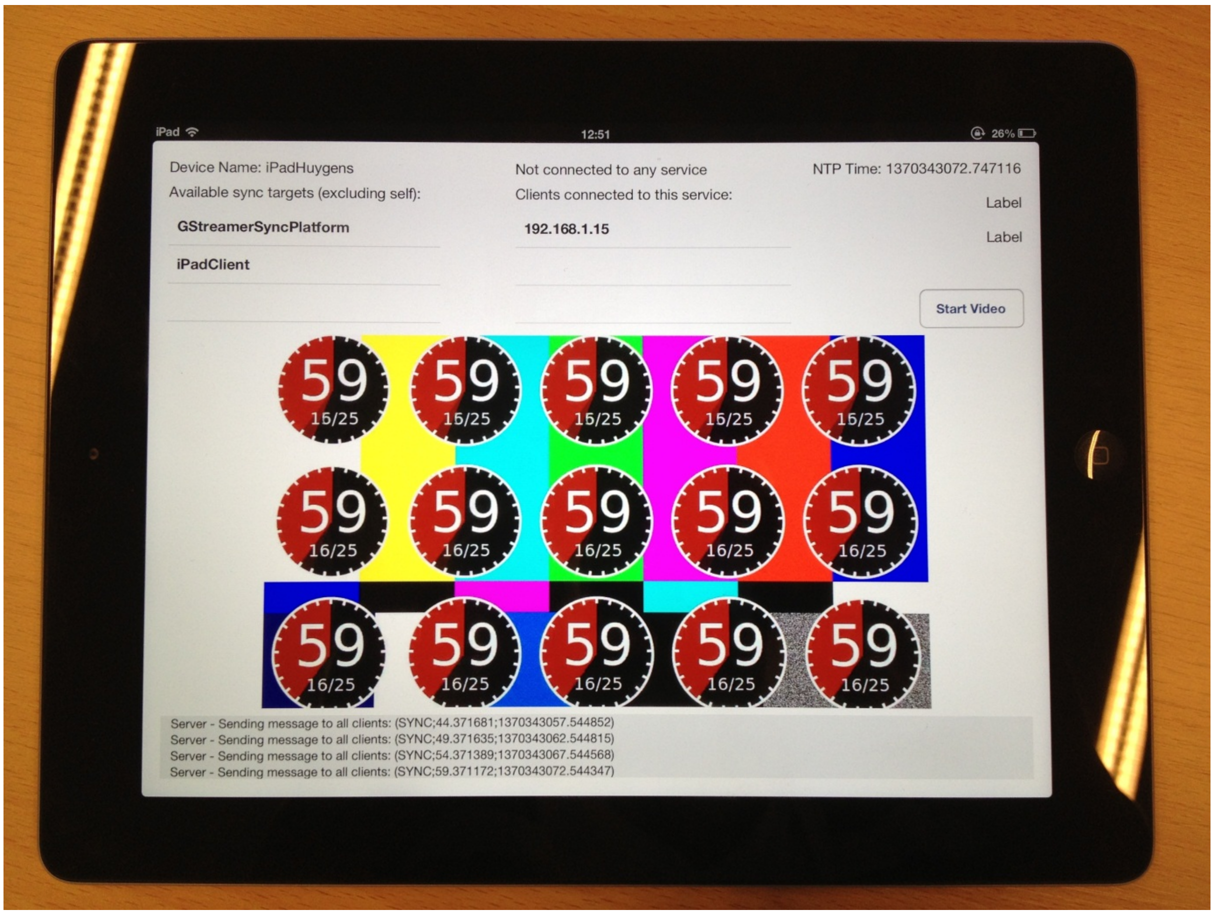
The *SyncedVideoPlayer* object in the diagram refers to the newly developer video renderer object, which is able to synchronize a video with frame-level accuracy. The *SyncedVideoPlayer* object communicates with a number of other objects:

- *NTPSync*: The module responsible for synchronizing the wallclock. Since on iOS it is not possible to set the system clock from within an App, this module keeps track of the offset between the iOS system clock and NTP time. This offset is then communicated back to the *SyncedVideoPlayer*, which performs playout based on the system clock but corrects for the offset with NTP.
- *BonjourServer*: The module responsible for the synchronization master mechanism. This includes advertising its availability via Bonjour and accepting incoming UDP packets from connected clients. It also makes sure that each connected client is periodically receiving synchronization updates (via the *ClientConnection* instances).

- *BonjourServerBrowser*: The module responsible for searching for available synchronization masters and connecting with a single master. This includes the client-side Bonjour procedures. Once connected to a synchronization master, this module receives synchronization updates and communicates them back to the *SyncedVideoPlayer*.
- *Brain*: The main model of the App. It is responsible for orchestrating communication between the various modules and is the interface to the app's *ViewController* (and thus it's view). Once the *SyncedVideoPlayer* has a new video frame available, the Brain forwards it to the *ViewController* which in turn allows the View to show it on the display.

5.4.3. Functionality

The Figure 13 shows an impression of the current proof-of-concept iOS synchronization demo. The top third of the interface can be used to select another device to perform inter-device synchronization with (in this particular case, another iPad and the GStreamer Synchronization testbed). It also shows a list of devices currently connected to the current iPad. On the top-right of the screen it shows the current NTP time, which is used in the synchronization process. The video being shown on the screen is generated test content that is particularly suitable for testing synchronization accuracy with.



Device Name: iPadHuygens
Available sync targets (excluding self):
GStreamerSyncPlatform
iPadClient

Not connected to any service
Clients connected to this service:
192.168.1.15

NTP Time: 1370343072.747116

```
Server - Sending message to all clients: (SYNC;44.371681;1370343057.544852)
Server - Sending message to all clients: (SYNC;49.371635;1370343062.544815)
Server - Sending message to all clients: (SYNC;54.371389;1370343067.544568)
Server - Sending message to all clients: (SYNC;59.371172;1370343072.544347)
```

Figure 13: Proof-of-concept version of iOS inter-device synchronization app

In Figure 14, shown below, an earlier version of the iPad application is synchronized with the GStreamer Synchronization testbed. As one can see, the inter-device synchronized iPad in this case allows a user to experience a football match from two different angles simultaneously. The main screen shows a zoomed-out top-down view on the football pitch while the iPad shows a zoom-in angled version.

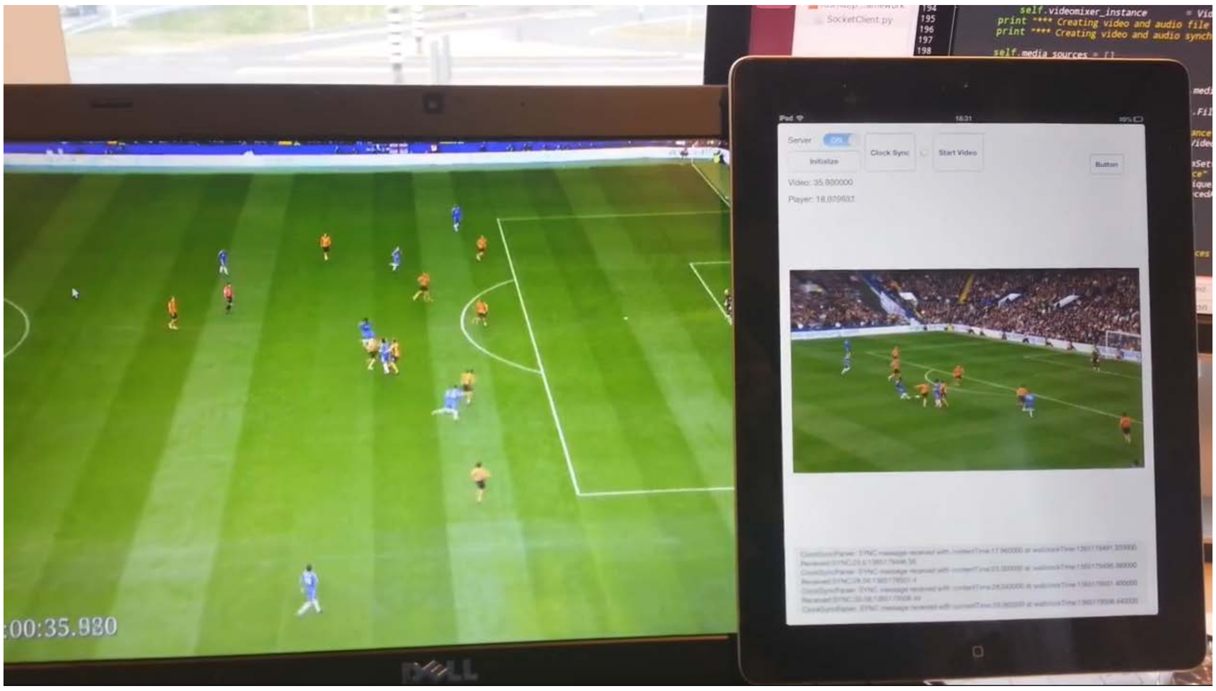


Figure 14: Synchronization between GStreamer Synchronization Testbed and iOS inter-device synchronization app

5.4.4. Installation/Setup

Due to the way provisioning of iOS devices works, the application bundle included in the software package can unfortunately not be installed directly on an iPad, since the iPad first needs to be provisioned with the correct Apple Developer's certificate. For a test version of the developed iOS inter-device synchronization application, please contact the authors of this document.

5.5. GStreamer Synchronization Framework

The GStreamer synchronization framework [17], as already discussed in detail in D4.2 [4], has been improved and extended to allow for inter device synchronization.

5.5.1. Placement of Application in System Architecture (LinuxOS/API)

The Linux based GStreamer synchronization framework is placed as a general purpose alternative to the set top box, without the constraints imposed by a set top box, like limited processing capabilities, future-proof, driver limitations, etc. The platform allows to experiment with state of the art multimedia technologies (like MPEG-DASH, Web Sockets, mDNS, etc.) while providing full flexibility at the same time. Furthermore, the open source GStreamer framework is supported by the Broadcom set top box platform.

A shortcoming of the framework is that it is not compatible with any HBB(-NEXT) user interaction APIs. It does not support HTML5 or the like. Therefore, the framework is focused on beyond state-of-the-art hybrid multi-device media synchronization.

The GStreamer Synchronization framework is compatible with a number of APIs and HBB-Next frame-accurate synchronization related enablers.

- mDNS based device discovery
- Partial implementation of the TS 102 823 - V1.1.1 specification for TS Timeline support
- UDP and TCP based inter device clock synchronization algorithm
- Inter media synchronization (e.g. used for showing 3D content or picture-in-picture rendering, see section 2.3)

5.5.2. Application Architecture

The GStreamer synchronization framework is a modular system which allows for frame accurate synchronization of heterogeneous media on a single device, and frame accurate synchronization in a multi device environment. From an architectural point of view, the system platform runs on a common off-the-shelf laptop, but could also run on a set top box (with modifications).

5.5.3. Functionality

The framework utilizes many of the build-in GStreamer elements (refer to Appendix A in D4.2 [4]). Besides these common elements, a number of custom elements have been developed to achieve frame accurate synchronization, and inter-device communication. These elements are shortly discussed in the subsequent sections.

5.5.3.1. SyncModule element

The SyncModule element enables PCR/PTS based inter media synchronization on a single device. For a complete discussion of the SyncModule, its implementation, capabilities and limitations, refer to section 3.3.2.4.1 in D4.2 [4].

5.5.3.2. Timeline converter element

This element handles timeline and PCR/PTS based media synchronization, and as such, is an improved SyncModule element as described in the previous section. The functionality of the timeline converter is twofold, depending on whether it is running in master or slave mode. In master mode (the left part of the figure below), it is distributing the clock information as specified by the timeline data. In this case, the “Content” input consists out of the video elementary stream embedded in for example the DVB-S transport stream. The “Sync data” input contains the elementary stream which holds the timeline information. In slave mode (the right part of the figure below), the timeline converter element receives this timeline information and adjusts the internal clocks of the passing content to match to the synchronization data it receives. This sync data is communicated within the pipeline in case of inter-media synchronization. In case of inter-device synchronization, the information is shared using different means (see the SyncComm element as discussed in the next Section.

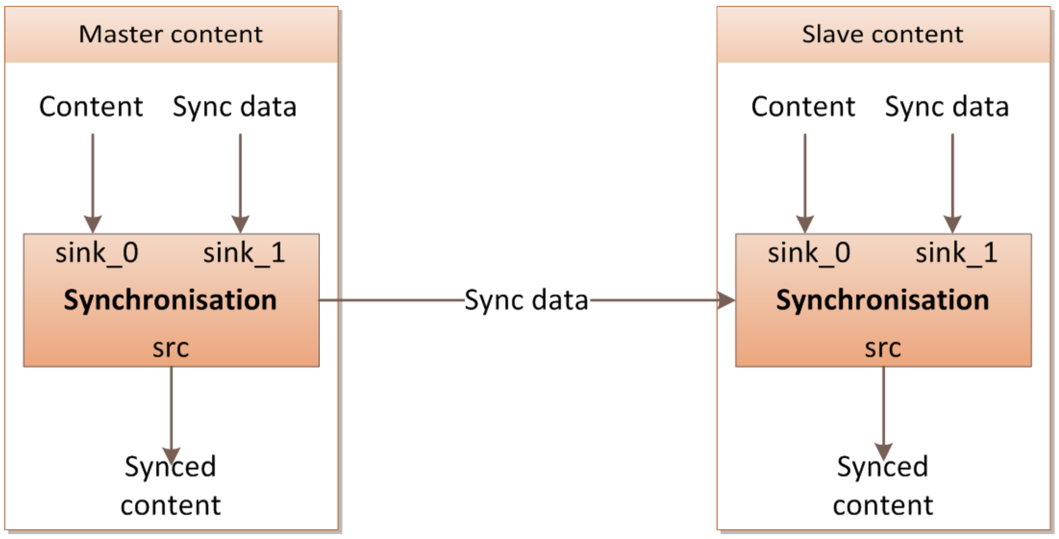


Figure 15. The timeline convert element provides both PCR/PTS and Timeline based inter-media synchronization.

5.5.3.3. SyncComm element

The SyncComm element facilitates inter-device synchronization. Currently, the GStreamer framework is considered to be rendering the master stream to which all other streams adhere too. This can be assumed, since the role of the GStreamer framework would primarily be to render a broadcast DVB stream which by definition does not support seeking (unless buffers or time shift functionality is in place). Other devices would synchronize their on demand IP streams to that master stream.

The SyncComm module announces the presence of the GStreamer framework and the source it is rendering to the local network. Furthermore, it allows other devices to subscribe to the synchronization service it is sharing. By emitting frame accurate clock and content time information (see Section 4.2.2 for an in depth explanation), other devices are able to synchronize on this media stream.

Announcements are broadcasted using mDNS/Bonjour. Synchronization information is shared using a more traditional low latency UDP or TCP port. This is specified in more detail in Section 4.2.3).

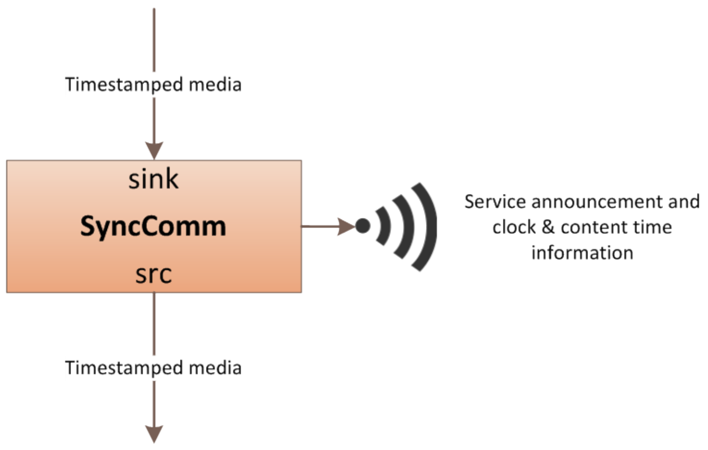


Figure 16: The SyncComm element facilitates communication to other devices.

The SyncComm module is operating in pass-through mode, e.g. timestamped media is passed and only timing information is extracted and communicated. Since it is multithreaded, this results in minimal overhead and processing delays.

5.5.3.4. High level abstraction in GStreamer: master blocks

GStreamer utilizes the concept of pipelines for rendering media. A pipeline consists of one or multiple elements which have a certain function (e.g. demultiplex, decode, crop or resize a video). Refer to Deliverable 4.2 [4] for a more in depth description of pipelines. Although elegant, managing a pipeline with many functions and/or multiple media files becomes increasingly more complex (due to the large amount of elements required). Creating and managing such a complex pipeline is cumbersome and prone to errors. Furthermore, many of the elements are repetitive, e.g. for video the same string of elements is required to demultiplex, decode and scale the video data. Therefore, a high level easy-to-use framework has been developed in Python providing simple interfaces to common sets of GStreamer elements. A set of common GStreamer elements is defined as a master block. This concept was already discussed in D4.2 [4], but since then this concept has been greatly simplified, further enhancing the power of this versatile approach to pipeline construction.

Previously, a number of master blocks were defined. These blocks are discussed in D4.2 [4], but are listed again for the convenience of the reader.

#	Master block	Description
1	HLSSource	Load and demultiplex an HLS transport stream
2	DASHSource	Load and demultiplex a MPEG DASH transport stream
3	LocalTSSource	Load and demultiplex a local transport stream
4	LocalTSSource	Load and demultiplex a local transport stream
5	LocalMP4Source	Load and demultiplex a local MP4 file
6	SyncedVideo	Decode and synchronize video data
7	SyncedAudio	Decode and synchronize audio data
8	MP3Encoder	Encodes and packages an audio stream into MP3 format
9	X264Encoder	Encodes and packages a video stream into x264 format
10	AACEncoder	Encodes an audio stream into AAC format and packages into an MP4 container
11	TheoraEnc	Encodes and packages a video stream into Theora format
12	VorbisEnc	Encodes and packages an audio stream into Vorbis format
13	MKVMux	Uses the Matroska multiplexer to multiplex audio and video
14	OGGMux	Uses the OGG multiplexer to multiplex audio and video and sent the resulting video stream to a TCP socket.
15	RTPBin	Packages one or multiple media streams into an RTP stream.
16	VideoMixer	Overlays multiple videos into one output video stream
17	UDPSink	Stream encoded audio or video data to an IP address and port.
18	VideoSink	Renders the video signal on the local display
19	AudioSink	Renders the audio signal on the local sound card

Table 3: Previously implemented master blocks.

As one can notice, a number of even higher level functionality can be distinguished. Sources (1 … 5), synchronization modules (6, 7), encoders (8 … 12), multiplexers (13 … 15), and sinks (16 … 19).

In the current version of the framework, master blocks have been defined on the level of the aforementioned key functions, namely:

- Source
- Processing
- Synchronization
- Encoding
- Sink (also known as output)

For example, a source master block has been created which can be fed virtually any type of media stream ranging from local transport stream to DVB-S source or MPEG DASH [21] playlist. Its auto-sensing feature automatically selects the proper underlying GStreamer element(s) which are required. The same holds for the other master blocks. This way, as simple as plug and play with minimal configuration overhead, one can create powerful pipelines while having full control if required. The primary features of the current master blocks are detailed in the Table 4.

#	Master block	Description
1	Source	Any local media file, HTTP Live Streaming playlist, MPEG-DASH playlist, DVB-S stream. The source could contain audio, video or both.
2	Processing	Buffer, demultiplex, decode (and position and scale if applicable) media stream
3	Synchronization	Perform synchronization, communicate with additional devices, announces sync service, handles both PCR/PTS based and timeline based synchronization.
4	Encoding	Optional encoding and multiplexing of resulting stream, useful when one wants to stream to a remote host or save to disk.
5	Sink	Renders to local display, file, TCP port, RTP stream or other output format. Can be audio, video or both.

Table 4: The master blocks given in Table 1 have been refactored to a small number of master blocks.

An example pipeline, using these master blocks, is given in Figure 17 below. In this example, a media source (e.g. a live broadcast via DVB-S) is processed by the GStreamer synchronization framework and displayed on a regular television. Next, a mobile device (e.g. a tablet) enters the room and announces itself on the local network. Since the GStreamer framework is also announcing itself, both devices are able to pair to each other and communicate content specific information. The tablet is instructed to play an on demand IP stream showing the same content as the broadcast but from an alternative camera angle. By exchanging clock and content time information, the tablet can play the alternative camera feed frame accurately synchronized to the DVB-S stream on the TV.

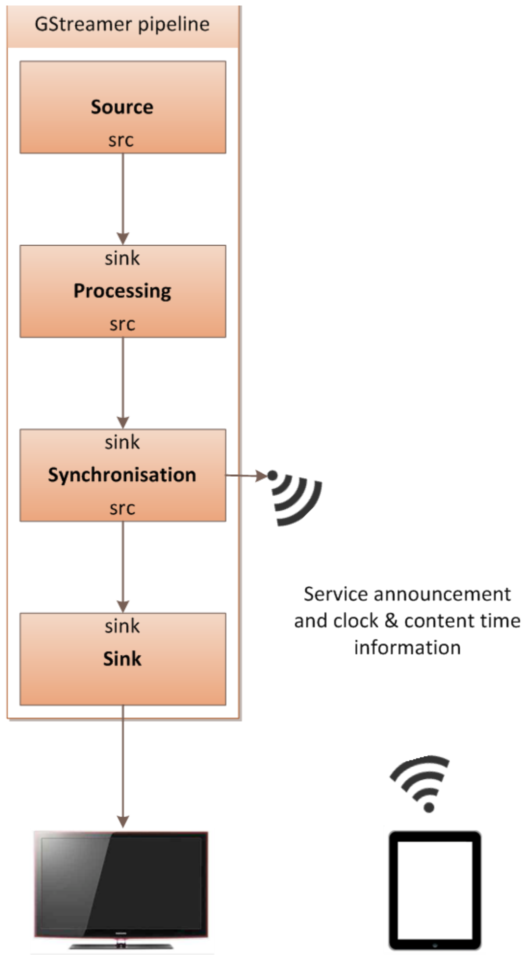


Figure 17: Simplified overview of inter device synchronization, synchronizing the GStreamer framework to a tablet device.

5.5.4. Installation/Setup

The GStreamer synchronization framework has a number of dependencies. These dependencies are listed in the following table:

Component	**Version**	**Notes**
Ubuntu	12.04	The 64 bit version is recommended, but the platform is known to be compatible with the 32 bit version as well. Furthermore, 12.10 can be used as well.
Python	2.7.3	Needs the following libraries: numpy-27, gst-python, pybonjour, autobahn, twisted
GStreamer	0.10.35	Needs the following libraries: gst-plugins-base-0.10.35, gst-plugins-good-0.10.30, gst-plugins-bad-0.10.18, gst-plugins-ugly-0.10.22. Everything compiled from git as of January 21, 2013.
FFmpeg	Current	The most up to date version of FFmpeg is recommended.
x264lib	Current	The most up to date version of x264 encoder is recommended.

Table 5: Software Dependencies of Gstreamer

Furthermore, it needs a fully functioning DVB-S receiver in case one wants to synchronize DVB-S content. By default the framework supports any DVB-S receiver as long as it is supported by the host operating system. GStreamer supports Apple's HTTP Live Streaming (HLS) and MPEG DASH [21], although support is still in an alpha stage. Adding DASH support is optional and requires significant effort. Since the specification, capabilities and installation of this element is continuously updated and changed, the exact installation details for this element are out of scope of this report. For a good start on adding MPEG DASH support to GStreamer, refer to the regarding Google Code repository[3]. HLS is supported by default.

[3] https://code.google.com/p/mpeg-dash-gstreamer/wiki/DASH

5.6. Synchronization in the Cloud

The synchronization component has not yet been integrated in the cloud due to a number of issues in the cloud offloading architecture. Furthermore, the current implementation of the synchronization module does not fit into the cloud architecture very well since it requires over the top inter-element communication which does not fit with the distributed architecture of the cloud. It is proposed that in a future release of the synchronization element, this communication is performed in band using for example Event signals[4].

5.7. Conclusions

The Synchronization features have been implemented on various platforms and types of devices. Beside an HbbTV set-top-box, also two tablet platforms are supported and a cloud tool. The modern, open-source Gstreamer framework has been tested as an alternative to the STB. This shows the feasibility and practicability of the developed HBB-NEXT synchronization algorithms and protocols. The inter-device features have been tested for interoperability on all devices and their use within the demonstrators, presented in Chapter 6, proofs the underlying synchronizations algorithms and protocols, described in Chapter 4.

[4] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/section-events-definitions.html

6. Demonstrators

6.1. Introduction

In the previous Section, the purpose of each device and software framework was explained, as well as the novel software modules. The scope of functionalities and capabilities of each developed component within HBB-NEXT should be clear now. This Section will deal with various demo installations. Each demo depicts a unique usage and composition of devices, with novel features in the field of inter-device media synchronization.

For the upcoming IBC conference in September 2013, several TV channels have been designed, whereby each channel represents a single demo and contains the regarding (HbbTV) application (e.g., a 3D channel, see Section 6.3). The purpose of each channel is highlighted in a single context. Note that some demos for IBC will cover features which are not directly related to inter-device media synchronization. Therefore, the set of features has been limited in order to fit the scope of this deliverable.

6.2. Demo 1: Settings App

The settings application as designed by WP2 offers accessibility services via a unified interface (see Figure 18). Currently these services are synchronized sign language interpretation video and multi-language subtitling. The rendering for both services is customizable by the user according to individual preferences and needs. As live subtitles, content from the flagship news magazine rbb aktuell is used. Sign language video samples are taken from the ARD Tagesschau news magazine.

The service can be supported by a second device. The device connection is initiated via the HBB-NEXT QR-Code based 2nd screen Framework. The user opens a personalized link on the second device (usually a tablet PC) that is provided via a QR code displayed on the TV device screen. The service provides the following cases for media synchronization:

- IP subtitles synchronized to broadcast video on TV
- IP sign language video synchronized to broadcast video on TV
- IP audio description on connected device synchronized to broadcast video on TV

- Inter-Device Synchronization of DVB-Video on TV to On-Demand Video on connected device
- Inter-Device Synchronization of DVB-Video on TV to alternate audio-streams on connected device (e.g. audio description)

The application UI especially comprises the following features:

- Individual positioning and scaling of sign language video (see Figure 19)
- Configuration of visual rendering options for subtitles (see Figure 20)
- Configuration of visual options for subtitles
- Native tablet-themed Android GUI on the connected device, corresponding to the GUI on the TV

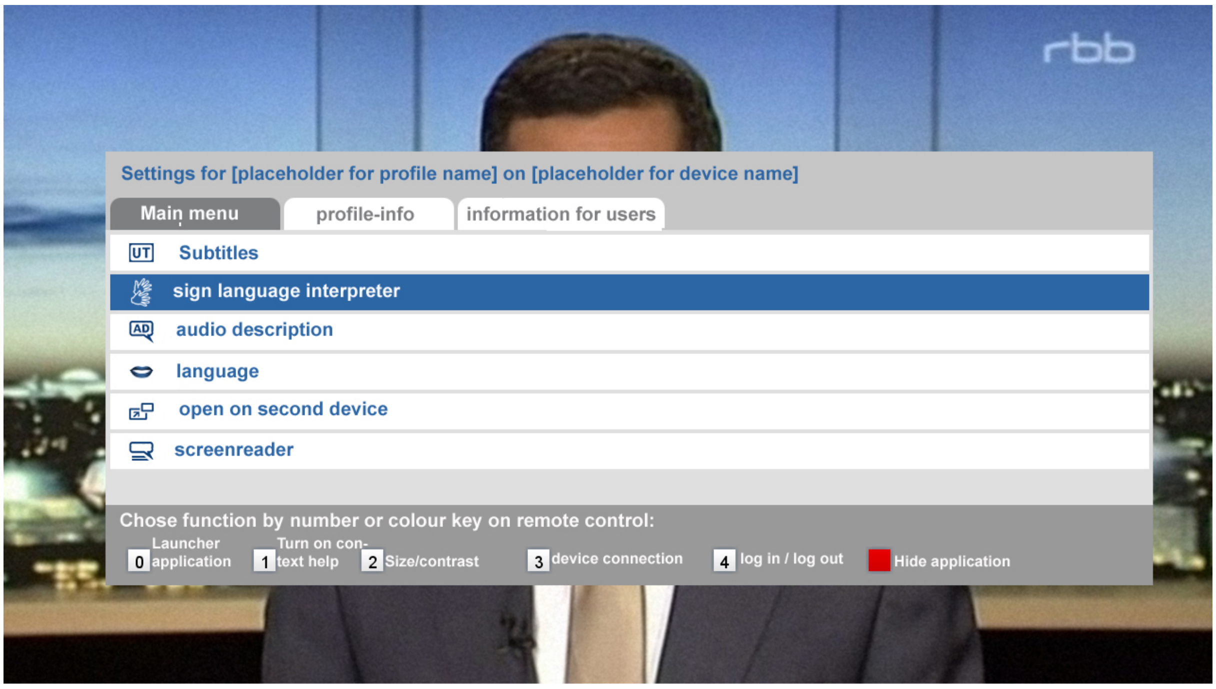


Figure 18: Main menu of settings application

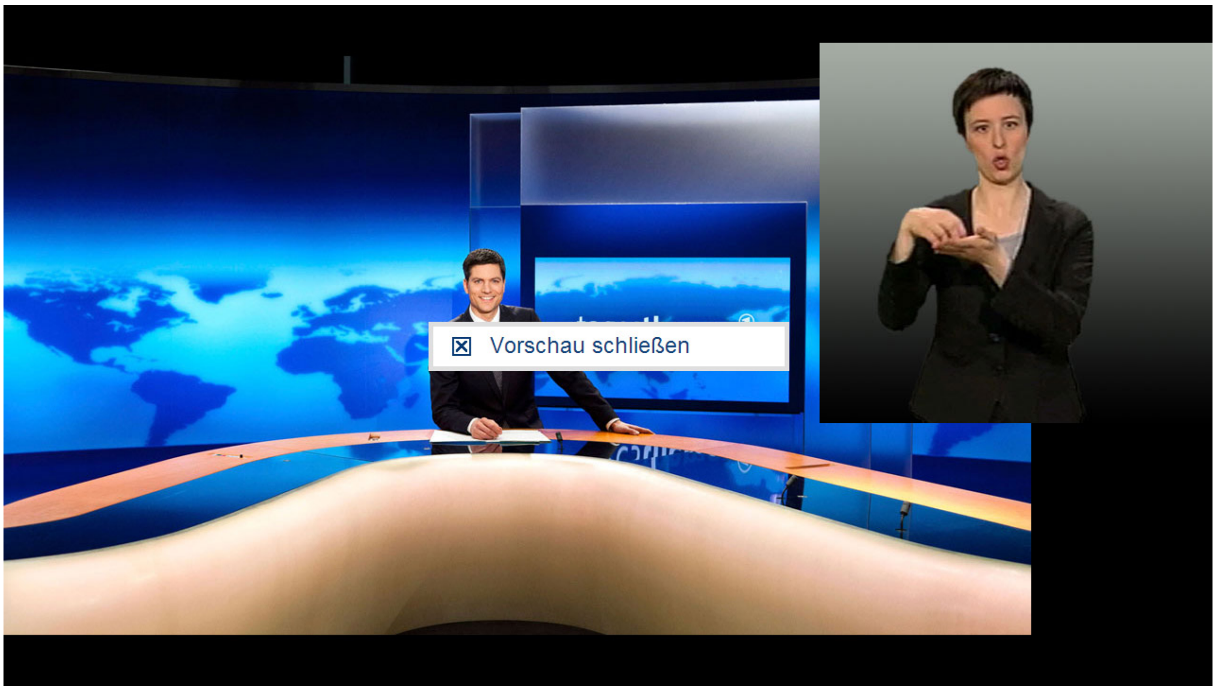
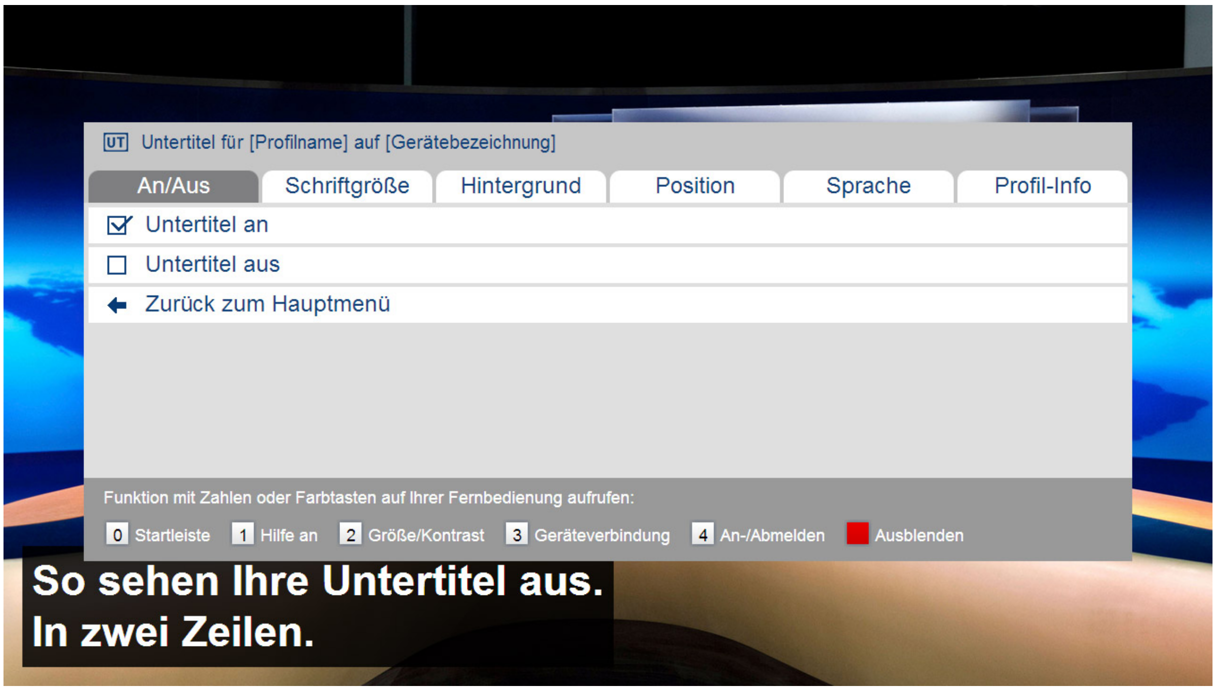


Figure 19: Synchronized sign-language service via IP



So sehen Ihre Untertitel aus.
In zwei Zeilen.

Figure 20: Configuration of synchronized subtitles via IP

Please refer to the appendix in Section 9 of this document to see all related WP4 requirements from HBB-NEXT Deliverable D2.2 [2].

6.3. Demo 2: 3D Content

The 3D demo is a new demonstrator, which was not planned from the beginning. The innovation and development process in WP4 revealed that the mechanisms and functionality for inter-media synchronization perfectly fit for also been used to enable hybrid 3D presentation. Also, the presented 3D demo for hybrid sources depicts the perfect application to demonstrate highly accurate inter-media synchronization, and was therefore included in the selection of IBC demos.

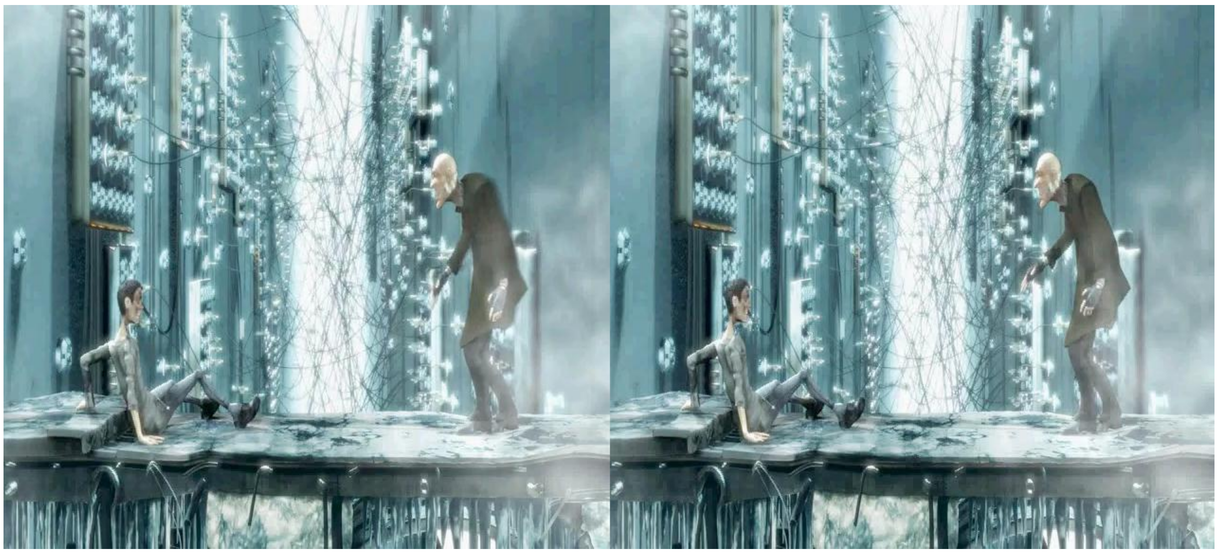


Figure 21: 3D Demo Screenshot

As shown in Figure 21 the demo application displays the 3D video in Half-Side-By-Side (Half-SBS) configuration, which can be rendered in 3D from 3D-enabled TVs. For this demo two different versions are available. One presents IP-IP inter-media synchronization, the other DVB-IP synchronization. The first loads the videos from the left and right from different files and therefor from different locations. The latter displays the left eye video from DVB-S tuner and the other from an IP source. As IP sources in this demo HTPP progressive download videos in H.264/AVC encoding are used. The DVB-S broadcast uses DVB video in MPEG-2 encoding. This requires the device to have the hardware decoders running with synchronised clocks, to avoid drifting of play speed.

The content shown in this demo was taken and re-encoded from the “Elephants Dream”[5] open source movie project, released under the Creative Commons license.

[5] http://orange.blender.org/blog/elephants-dream-in-stereoscopic-3d/

The user interface of this HBB-NEXT demo application is limited to display the movie in 3D. The control is limited to the colour buttons:

- RED: seekTo(0)
- GREEN: start left video
- YELLOW: start right (pip) video
- BLUE: sync left/right video

Additionally, the device should be able to render Half-Side-By-Side content to 3D output. Installation and setup of this application only require an HbbTV conformant Web Server and 3D content, prepared and encoded for hybrid playout. To enable this demonstrator the set-top-box device's enhanced inter-device synchronization implementation is used, as described in Section 5.2.

Please refer to the appendix in Section 9 of this document to see all related WP4 requirements from HBB-NEXT Deliverable D2.2 [2].

6.4. Demo 3: FascinatE

The FascinatE demonstrator shows inter device synchronization of two heterogeneous media sources. Within the EU FP7 project FascinatE [15] a capture, production and delivery system capable of supporting interaction, such as pan/tilt/zoom (PTZ) navigation, with immersive media has been developed by a consortium of 11 European partners from the broadcast, film, telecoms and academic sectors. This system allows end-users to interactively view and navigate around an ultra-high resolution video panorama showing a live event. The output is adapted to the end-user's device, ranging from a mobile handset to an immersive panoramic display.

Ultra high resolution (7k x 2k) content has been captured, in combination with a regular HD broadcast feed. The ultra-high resolution content is delivered to the end user using the IP-based streaming technology called "tiled streaming" [16]. Tiled streaming has been developed within the FascinatE project and allows for the delivery of that part of the content that the user is interested in, adapted to his display device (e.g. a tablet or HD-TV). Additionally, the user can interact with this panorama by panning, tilting and zooming.

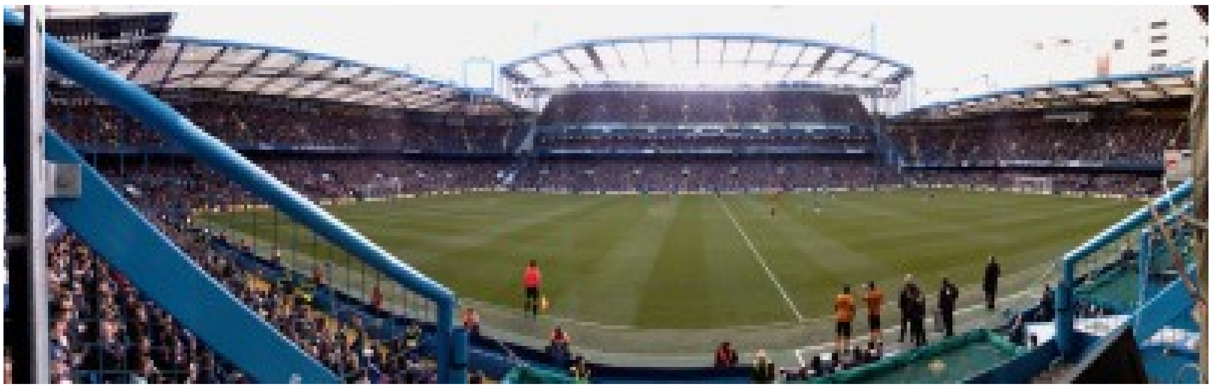


Figure 22: An example of an ultra-high (7k x 2k) resolution video capture.

Besides the ultra-high resolution panorama, the regular broadcast feed is broadcasted via for example DVB-S. In this demonstrator, both video feeds are synchronized using HBB-NEXT synchronization technology as discussed in detail in D4.2 [4]. Using the inter-device synchronization algorithms as detailed in Section 4.2.4, frame accurate synchronization is achieved between the FascinatE platform (rendering the interactive panorama on a tablet) and GStreamer synchronization framework [17] (rendering the broadcast camera on a regular HD-TV). Media displayed on both devices are frame-accurately synchronized. This is required, since when no frame accurate synchronization is achieved, differences in the two video feeds can be distinguished, which would be annoying for the end user and ruin the immersive experience. Refer to Figure 24 for an overview of this setup. The simplified architecture is given in Figure 23.

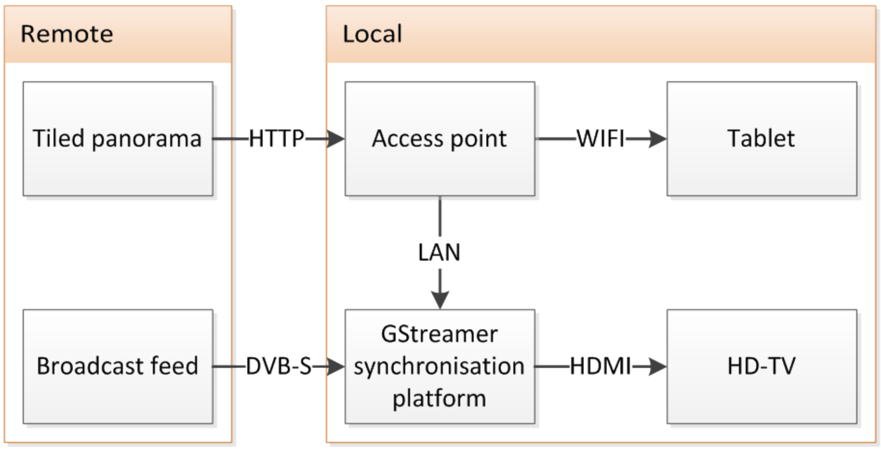


Figure 23: Simplified overview of the Fascinate demonstrator.

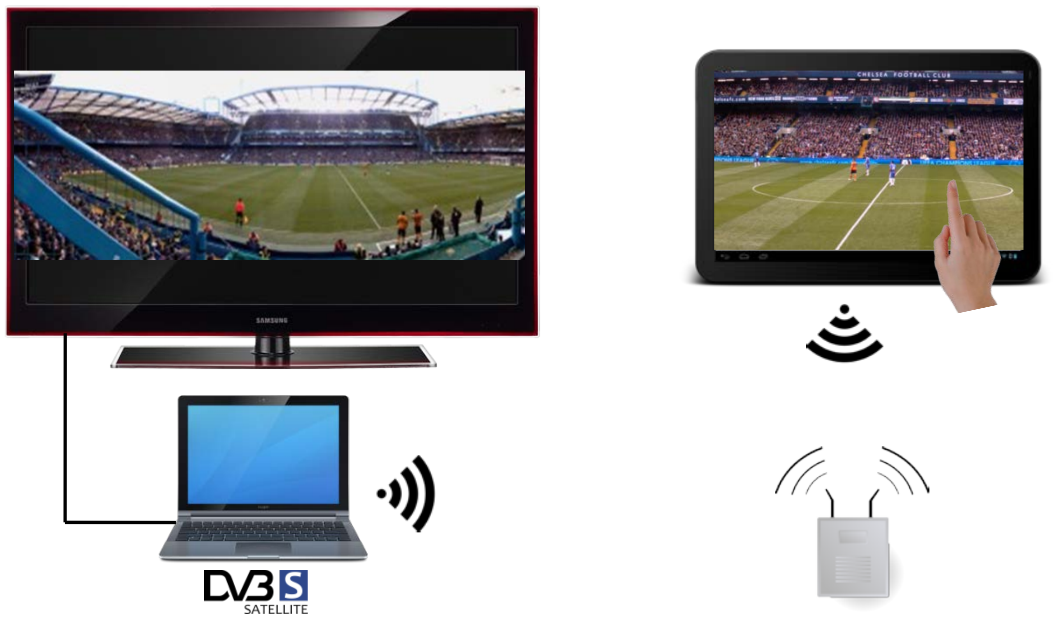


Figure 24: System overview of the HBB-Next FascinatE demonstrator showing inter-device synchronization between a tablet and a TV

In this specific demonstrator, on-demand pre-recorded content is used since the use of live Ultra-HD content would be difficult and very expensive to obtain. The concepts of this demonstrator, however, are applicable to live events as well.

This demonstrator clearly shows the synergy between the two EU FP7 programs, in which two separate technologies complement each other very well and enable new, immersive use cases. It combines interactive high resolution video interaction with frame accurate synchronization of heterogeneous media sources amongst different devices within a local network.

Future work will focus on merging this demonstrator with the set top box and Android based tablets. Furthermore, content management needs to be implemented linking the different content stream to each other within, for example, one session.

The demonstrator requires the following hardware components:

- Laptop
- GStreamer synchronization platform
- USB DVB-S receiver
- iPad 2 or newer with HBB Next synchronization application and iOs 6 or newer

- WiFi access point
- HD-TV connected via HDMI
- DVB-S modulator
- Web server hosting the segmented panorama content

The laptop and iPad must be connected to each other via the local area network and must reside in the same subnet. Furthermore, the DVB modulator must broadcast the FascinatE broadcast camera feed. The webserver could run on the same laptop, but could also be hosted on a more remote server (e.g. a CDN).

Please refer to the appendix in Section 9 of this document to see all related WP4 requirements from HBB-NEXT Deliverable D2.2 [2].

6.5. Conclusions

Three different demos have been presented. The settings app demo is focused on accessibility features in a multi-device HbbTV environment. It enables a glimpse to a future, where several present end-devices allow hearing or vision impaired media consumers to experience digital media with fewer barriers, tailored to their needs. The second demo, showing 3D content from hybrid sources highlights the effectiveness of the developed inter-media synchronization methods in HBB-NEXT. As 3D features are getting more important on the market, broadcasters may gain a powerful platform with this demo: Picture a scenario where regular customers receive a 2D HD broadcast stream via DVB. Now premium customers could receive the second video stream, which would enable a 3D playout on a compatible HD-TV, via a third-party, IP-based source. This premium feature might be an attractive way to increase revenues of a broadcaster, without consuming extra bandwidth of the DVB transponder. The third presented demo describes the successful fusion between HBB-NEXT WP4 developments and technology from the EU FP7 project FascinatE. Just as the 3D demo, the FascinatE demo also target to enrich the DVB consumer sector.

By allowing users to access selected parts of a main event which runs on the main screen, and visualize custom elements on their end-devices in a synchronous fashion, the demo takes huge steps towards a media world, where every consumer experiences digital media as an individual.

7. Summary and Outlook

Multi-device synchronization capability is a highly relevant aspect for next-generation media environments. For the presented deliverable, novel approaches to build the technical base for multi-device synchronization in an HbbTV environment have been developed. Research and development on both popular mobile OS platforms iOS and Android, as well as on a commercial STB and on the open-source Gstreamer framework has been conducted. As a practical result, demos based on these platforms have been forged, which shall be presented at the IBC conference in 2013. This deliverable documents the conceptual phase, as well as the assembly process of these demos. Furthermore, it is a result of the merging process of contributions from all WP4 partners, to create a single, light-weight synchronization scheme, which is applied in a heterogeneous end-device landscape.

The elaborated technology allows for vast enhancements of the set of already included features. Evaluations are not included in this deliverable. The different demos now need to be tested under different conditions, to match real-life digital media environments in common households. Therefore, the impact of varying network conditions (changing throughput, packet delay and jitter values) on the synchronization accuracy of each demo, respectively end-device, needs to be tested. To realize this, different uniform measurement methods need to be identified. Another aspect is compatibility. So far, the presented apps run on designated tablet models. It has to be evaluated, if they also run flawlessly on different models using the same operation system. Although not crucial in the current phase (design and first evaluation), customer acceptance can be already an important sign post for the next improvements of the inter-device synchronization software. The IBC conference in 2013 is a perfect playground for interested customers to test the new technology, and their feedback will be valuable for the course of the WP4 developments.

8. References

[1] HBB-NEXT Deliverable D2.1 - Usage Scenarios and Use Cases, http://www.hbb-next.eu/index.php/documents

[2] HBB-NEXT Deliverable D2.2 - System-, Service-, and User Requirements, http://www.hbb-next.eu/index.php/documents

[3] HBB-NEXT Deliverable D4.1 - ANALYSIS: Cloud-Based Services and Service/Content Synchronisation, http://www.hbb-next.eu/index.php/documents

[4] HBB-NEXT Deliverable D4.2 - DESIGN AND PROTOCOL: Middleware Components Content Synchronisation/Cloud Service Offloading, , http://www.hbb-next.eu/index.php/documents

[5] Network Time Protocol Version 4: Protocol and Algorithms Specification

[6] Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI

[7] Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems, ETSI EN 300 468 V1.11.1 (2010-04)

[8] Gotoh, T; Imamura, K; Kaneko, A (2002). "Improvement of NTP time offset under the asymmetric network with double packets method"

[9] Executive Summary: Computer Network Time Synchronization, http://www.eecis.udel.edu/~mills/exec.html

[10] Precision Time Protocol

[11] C. Köhnen, C. Köbel, and N. Hellhund, "A DVB/IP streaming testbed for hybrid digital media content synchronization," in 2012 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin), 2012, pp. 136–140.

[12] T.-M. Chen, L.-J. Lin, H.-L. Chen, K.-C. Liu, C.-L. Su, and C.-Y. Cho, "A DVB-T Implementation for Android Stagefright on a Heterogeneous Multi-core Platform," in 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012, pp. 112–118.

[13] L. Beloqui Yuste, S. M. Saleh Al-Majeed, H. Melvin, and M. Fleury, “Effective synchronization of Hybrid Broadcast and Broadband TV,” in 2012 IEEE International Conference on Consumer Electronics (ICCE), 2012, pp. 160–161.

[14] S. Moriyasu, K. Tajima, K. Ohshima, and M. Terada, “Group synchronization method with fast response time for VoD services,” in 2012 International Conference on Information Networking (ICOIN), 2012, pp. 182–187.

[15] EU FP7 project FascinatE, http://www.fascinate-project.eu

[16] Omar Niamut, Jean-Francois Macq, Martin Prins, Ray Van Brandenburg, Nico Verzijp, “Towards Scalable And Interactive Delivery of Immersive Media”, NEM Summit 2012 Proceedings, October 2012.

[17] GStreamer Open Source Multimedia Framework, http://GStreamer.freedesktop.org/

[18] Simple Service Discovery Protocol/1.0 - Operating without an Arbiter, IETF Internet Draft, October 1999

[19] Multicast DNS – IETF RFC 6762, February 2013

[20] Dynamic Host Configuration Protocol – Network Working Group RFC, March 1997

[21] ISO/IEC JTC 1/SC 29, Information technology MPEG systems technologies Part 6: Dynamic adaptive streaming over HTTP (DASH), 2011-01

9. Appendix

Each demonstrator presented in Section 6 fulfills several WP4 requirements described in the HBB-NEXT Deliverable D2.2 [2]. The following Subsections list these requirements.

9.1. General Link to Requirements

The demos cover all requirements related to the A/V synchronization enabler but system requirements YR 2.01 – Transfer media sessions and YR 2.02 – Time-Shifting. These requirements have been reassessed in their importance due to different reasons: An explicit transfer of media sessions in the sense that the complete session state is transferred to a connected device is a very generic use case. The synchronization scenario realized in demo 1 (“settings app”) allows mirroring the TV content along with synchronized content and comes close to fulfilling this requirement. Yet technical limitations as well as usability considerations on the connected device prevent a direct copy of the session when more than two sources shall be synchronized and rendered. As for the Time-Shifting, a decision had to be taken regarding allocation of development resources. Though a useful feature as described in Scenario 1 in HBB-NEXT deliverable D2.1 [1], providing synchronized services for live-broadcasts in real-time as demonstrated in demo 3 was considered the primary research challenge.

Contribution to Generic Requirements

The Generic Requirements as formulated in D2.2 [2] broadly categorize the requirements by the aspect in which they provide value to the user. It can be noted that the demonstrations described in this deliverable especially contribute to *Ensure Accessibility & Usability* by providing all synchronization solutions related to this Generic Requirement. Further the synchronization part of *Enable Hybrid Content* is supported by the developed technologies. A matching of the synchronization requirements of *Enable Seamless Device Integration* is as well positive with the exception of the already mentioned YR 2.01 and YR 2.02. It can be argued that YR 2.01 is in fact validated by demo 1, yet it remains a chance that certain enhancements to the original scenarios from D2.1 [1] would require a more versatile implementation.

9.2. Requirements of Demonstrator 1: Settings App

Demonstrator 1 especially focuses on the Generic Requirements *Ensure Accessibility & Usability* and *Enable Seamless Device Integration*. The detailed requirements are as follows:

Service Requirements independent from enablers

SR 00.02b EPG supported content,

SR 00.03a IP content,

SR 00.03b IP content,

SR 00.04b Content Listing adaptation,

SR 00.05d Playing A/V content,

SR 00.06 IR controller,

SR 00.07 (Dis-) Connection via user interface,

SR 00.18 Share content

Service Requirements related to the A/V Content Synchronization enabler

SR 02.01 PiP Service Option,

SR 02.02 Sync Service User Interface,

SR 02.03 IP Server functionality,

SR 02.04 Service based Timeline information generation,

SR 02.05 Timeline multiplexing

User Requirements independent from enablers

UR 00.01 Second Screen Locations

System Requirements independent from enablers

YR 00.05c Broadcast-Related apps,

YR 00.05d Playing A/V content,

YR 00.06 IR receiver, YR 00.07 Connection via WAN / LAN,

YR 00.08a Detection through,

YR 00.08b Non-local device detection,

YR 00.08c UPNP,

YR 00.08d Periodical scanning,

YR 00.10 Platforms,

YR 00.12 Autostart app on TV,

YR 00.13 Profile device independence,

YR 00.14 Profile device linkage.

System Requirements of A/V Content Synchronization Enabler

YR 02.03 Distributed Playback-Control,

YR 02.04 Decouple synchronicity between clients,

YR 02.05 Continuous check of synchronicity,

YR 02.06 Synchronous command execution,

YR 02.08 Multiple Parallel Decoding,

YR 02.09 Protocols,

YR 02.10 Render Picture in Picture,

YR 02.11 Synchronize two contents,

YR 02.12 Timeline extraction,

YR 02.13 Network request of subtitle information,

YR 02.14 Subtitle Rendering, YR 02.15 Timeline Based Subtitle Synchronization,

YR 02.16 Synchronization Functionality,

YR 02.17 Synchronize Clocks,

YR 02.18 Communication with Synchronization Servers,

YR 02.19 Set-Top-Box Remote Control Functionality,

YR 02.20 Synchronization mechanics,

YR 02.21 Transcoding Control Unit,

YR 02.22 Synchronization of subtitles,

YR 02.23 Inter-Destination Synchronization,

YR 02.24 Distributed-Playback Control,

YR 02.25 Multi-Domain Synchronization,

YR 02.26 Clock-Synchronization,

YR 02.27 Inter-Destination Synchronization of DVB and IPTV streams,

YR 02.28 IDMS Playback Management

9.3. Requirements of Demonstrator 2: 3D Content

The demonstrator implements the following requirements:

Service and system requirements for A/V Content Synchronization enabler

SR 02.01 PiP service option,

YR 02.12 Timeline extraction,

YR 02.10 Render picture in picture,

YR 02.11 Synchronize two contents,

YR 02.15 Subtitle synchronization

Service and system requirements independent from enablers

SR 00.05d Playing A/V content,

YR 00.05a Additional content on first screen

9.4. Requirements of Demonstrator 1: Settings App

This demonstrator concentrates on the Generic Requirements Enable Hybrid Content and Enable Seamless Device Integration. In detail it implements the following requirements:

System requirements for A/V Content Synchronization enabler

YR 02.03 Distributed playback control,

YR 02.05 Continuous check of synchronicity,

YR 02.06 Synchronous command execution,

YR 02.09 Protocols, YR 02.10 Render picture in picture,

YR 02.11 Synchronize two contents,

YR 02.12 Timeline extraction,

YR 02.16 Synchronization functionality,

YR 02.17 Synchronize clocks,

YR 02.18 Communication with synchronization servers,

YR 02.20 Synchronization mechanics,

YR 02.22 Synchronization of subtitles,

YR 02.24 Playback control,

YR 02.26 Clock synchronization

Service requirements independent from enablers

SR 00.05d Playing A/V content

System requirements independent from enablers

YR 00.04c Content adaption,

YR 00.05b Additional content on second screen,

YR 00.05d Playing A/V content,

YR 00.08a Detection through LAN,

YR 00.08d Periodical scanning,

YR 00.10 Platforms

System requirements for Cloud Service Offloading enabler

YR 04.09 Stream into elementary streams.