Directorate General for Communications Networks, Content and Technology

Innovation Action

ICT-687655

D2.1 System Architecture

Due date of deliverable: 31 March 2016

Actual submission date: 10 May 2016

Resubmitted with minor changes: 14 July 2017

Start date of project: 1 December 2015 Duration: 36 months

Lead contractor for this deliverable: **Cisco**

Version: **14 July 2017**

Confidentiality status: **Public**

Abstract

This document describes the system architecture being developed by the 2-IMMERSE project. This architecture is designed to enable the four multi-screen service prototypes that will be delivered through the project. The System Architecture is layered as a set of platform services, a client application architecture and production architecture. The system architecture is a work in progress; it will evolve both as we refine it and specify it in more detail, and as we deliver each of the multi-screen service prototypes through the project.

Target audience

This is a public deliverable and could be read by anyone with an interest in the system architecture being developed by the 2-IMMERSE project. As this is inherently technical in nature, we assume the audience is technically literate with a good grasp of television and Internet technologies in particular. We have included a Technology Overview section that summarises a range of technologies that are potentially applicable within the project. This document will be read by the Project Consortium as it defines the system architecture that will be adopted and evolved throughout the project.

Impressum

Full project title: 2-IMMERSE

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.1 System Architecture

Editor: James Walker, Cisco

Workpackage Leader: James Walker, Cisco

Project Co-ordinator: Helene Waters, BBC

Project Leader: Phil Stenton, BBC

This project is co-funded by the European Union through the ICT programme under Horizon2020.

Executive Summary

This document describes the 2-IMMERSE system architecture. The system architecture is a work in progress. We expect that the architecture will evolve both as we refine it and specify it in more detail (for example through detailed component interface specifications) and as we address the expanding scope of the four multi-screen service prototypes through the project. Within the project, we will keep this document updated to reflect this development.

As work on the architecture started before the D4.1 Prototype Service Descriptions deliverable was available, we have adopted a set of assumed Core Technical Requirements to enable us to proceed; these are presented in this document. Subsequently, with the release of D4.1 we have been able to address requirements emerging from the service prototype descriptions. We have included the set of prioritised user stories from the initial service prototype (Watching Theatre at Home) in this document.

We have taken a ‘layered’ approach to documenting our system architecture in order to maximise clarity and maintain an appropriate ‘separation of concerns’:

- The platform is defined as a set of services which support applications running on client devices. In defining these services we have described the service functionality, key interfaces and technology choices where they have been made.
- The client application architecture defines a common HTML and JavaScript environment for the Distributed Media Application components, and the underlying application that manages their lifecycle and presentation. It also details how this is supported on the various devices that participate in the system.
- The production architecture is defined at a high level; however, we note that a detailed, generalised production architecture is difficult to create, and specific production architecture will be determined for each service prototype.

The document also includes a comprehensive Technology Overview section that summarises a range of technologies that are potentially applicable to the 2-IMMERSE project.

List of Authors

Mark Lomas - BBC

Rajiv Ramdhany - BBC

Andy Gower - BT

Ian Kegel - BT

Jonathan Rennison - BT

Martin Trimby - BT

Doug Williams - BT

Ian Wray - ChyronHego

James Walker - Cisco (also editor)

Pablo Cesar - CWI

Jack Jansen - CWI

Michael Probst - IRT

Christoph Ziegler - IRT

John Wyver - Illuminations

Reviewers

Ian Kegel - BT

Table of contents

Abbreviations and Definitions

AVC	Advanced Video Coding (H.264/MPEG-4 Part 10 AVC)
Box	A metaphor for users sharing an experience (i.e. in the same session) but in different physical locations (contexts), based on the concept of the theatre box.
Context	One or more connected devices collaborating together to present a media experience. Each context has a contextId unique to its session. There can be many contexts on a single LAN, but a device can only be a member of one context at a time. Devices belonging to the same context must be able to discover each other using DIAL. Devices can join or leave a context at any time.
Correlation	The relationship between two timelines specified as a pair of timestamps (one from each timeline) and a speed value
CSA	Companion Screen Application
CENC	Common Encryption
CSS-CII	DVB-CSS Content Identification & other Information protocol
CSS-TS	DVB-CSS Timeline Synchronisation protocol
CSS-WC	DVB-CSS WallClock Synchronisation protocol
DASH	Dynamic Adaptive Streaming over HTTP
DMApp	Distributed Media Application – a set of software components that can be flexibly distributed across a number of participating multi-screen devices.
DMApp Component	A software component that renders media object(s), or supports viewer interactions.
DVB-CSS	DVB's Companion Screen and Streams specification
Experience	The experience of consuming a Distributed Media Application across multiple participating devices in a context.
Experience Timeline	Time since the start of an experience. The progress of time during an experience and the time positions when 'activities' such as the playback of a media object are scheduled to happen.
HbbTV	Hybrid broadcast broadband TV
HEVC	High Efficiency Video Coding (H.265/MPEG-H Part 2 HEVC)
HLS	HTTP Live Streaming
Lobby	A construct that allows groups of users who are members of the same session (but potentially different contexts) to come together and communicate. A session can have many lobbies or "rooms". Users can join or leave a lobby at any time and new lobbies can be created and destroyed

	at any time.
Media Composition Protocol	A protocol for describing how media objects / feeds should be composited together over time to produce a presentation optimised for a particular presentation device.
Media Object Timeline / DMApp Component Timeline	The playback progress of a media object. This can be reported as time since start of the media playback in seconds. Alternatively, presentation timestamps signalled in the media stream e.g. PTS or TEMI in broadcast streams can be reported.
Media Objects	Media streams or assets that comprise an experience. Includes today's typical programme elements (main A/V, audio description, subtitles etc.), but would also include clean feeds, audio commentary, auxiliary camera feeds, metadata feeds, images, graphics, A/V clips (e.g. highlights / replays).
MSAS	Media Synchronisation Application Server – a server entity in HbbTV2.0 (and DVB-CSS) stacks that collects current playback timestamps from a number of synchronisation clients and synchronises them.
Session	One or more contexts that are synchronised together into a shared media experience and presented simultaneously across sites. Contexts can join and leave a session at any time. Each session has a globally unique sessionId.
Synchronisation Timeline	A selected timeline to which a DMApp component will align itself e.g. the experience timeline. A correlation between the Synchronisation Timeline enables conversion of time values from the DMApp component timeline and the Synchronisation Timeline.
Synchronisation Server	A server entity that collects the current timeline position from the Synchronisation Timeline and distributes this timestamp to synchronisation clients connected to it.
Synchronisation Client	A client entity that receives Synchronisation Timeline updates from the Synchronisation Server and synchronises to the expected timeline position of its own media playback based on the update.
Timeline	The notion of progress of time or media playback progress. A timeline may have its own time representation i.e. a tick rate (ticks per second) and a speed (speed at which the timeline progresses e.g. 1.0, 2.0, etc.).
Timeline Updates	Intermittent presentation timestamps indicating progress along the timeline. Specified in the units of the timeline. Usually specified as a pair of timestamps, the second timestamp representing the WallClock time when the presentation timestamp was read.
TLS	Transport Layer Security
TS	Transport Stream
User	Someone consuming the media experience within a given context and session. A user has a userId, obtained by logging into one or more devices belonging to a context

UX Engine	**User Experience Engine - orchestrates the distributed multi-screen experience, managing each of the participating client devices, and adapting the presentation to the user’s environment, their participating devices and preferences.**
WallClock	**A shared clock representing a common notion of time by all entities enabling or participating in an experience.**

1 Use Case Summary

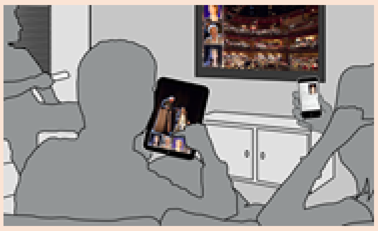
The 2-IMMERSE project will develop four service innovation prototypes of multi-screen entertainment experiences. Unlike existing services, the content layout and compositions will be orchestrated across the available screens and an object-based production approach will enable the experiences to be immersive, personalised and efficiently delivered.

Descriptions of these service prototypes and user stories derived from them form the basis of the requirements for the system architecture. These are documented in project deliverable D4.1. The service prototype descriptions are at different levels of maturity, which reflects that the trials of the service prototypes are planned at different points within the project.

A brief summary of the service prototypes is included below for reference. For the Watching Theatre at Home service prototype, which is the first to be trialled, a set of user stories has been generated. These are included in Annex A - Watching Theatre at Home User Stories, also for reference. These include a set of agreed priorities. User stories have not yet been generated for the other service prototypes but will be in due course.

Service Prototype Summaries

Watching Theatre At Home



This service innovation prototype is called **Theatre at Home** because it offers an enhanced social experience for users in a domestic context to watch a live or “as live” broadcast of a theatre performance. The user will have a second screen device that can access synchronized information streams directly from the provider of the broadcast and from the web through social media applications including Twitter but which can also, at times, feature audio and video chat with others who are watching.

The service innovation prototype will enable a user to watch a theatre production, shot with multiple cameras, as either a live or an ‘as live’ experience. Viewers will be able to contribute to and monitor different forms of feedback throughout the performance, and to discuss it with others who are watching at the same time, either in a different room or in a different home.

Owner: John Wyver (Illuminations) **Rights Originator**: Royal Shakespeare Company

Watching Theatre At School

This service innovation is called Theatre in School. This service enables pupils in schools across the country to watch a filmed performance of a play performed by the Royal Shakespeare company. Pupils are able to augment the main filmed presentation of a play with access to related supporting content and experiences to help them deepen their understanding of the play. This related content may include a synchronised transcript of the play, character summaries, short films featuring the talent in the play and even live communication session with the actors and other creative talent associated with the production.

Owner: John Wyver (Illuminations) **Rights Originator**: Royal Shakespeare Company

Watching MotoGP at Home

This service innovation will provide a user with a personalised experiences that can be controlled to suit a viewer's interests/experience with the sport. It will allow video footage and telemetry data to be displayed on a mixture of a large TV and on smaller personal screens. The trials with consumers will take place in multiple sites. Research insights will be captured from device/service instrumentation and follow-up qualitative questionnaires and interviews with trialists. We also plan to carry out VIP demos that could be held both at the track and at other VIP locations (BT Centre, BBC, Cisco, etc.).

The trial will focus on the Great Britain MotoGP race (September in 2017).

Owner: Andy Gower (BT) **Rights Originator**: Dorna Motor Sports

Watching Football In A Pub

This service innovation relates to an experience designed to suit UK city centre pubs

showing sport. It will mix large screen viewing with opportunities to access content and interactive experiences that may be playful and promotional on personal screens. We anticipate a system capable of supporting a diverse range of experiences centred, ultimately, on a single sport event but that finds a way to encourage and promote business within the pub through promotions and possibly competitions.

The trial will be centred on the Emirates FA Cup Final that will be held in May 2018.

Owner: Martin Trimby (BT) **Rights Originator**: The Football Association

2 Requirements

2.1 Overview

As WP2 has started to consider the 2-IMMERSE system architecture, WP4 has been defining the trial scenarios in parallel, with D4.1 only becoming available for a short while prior to this deliverable. We have defined a set of Core Technical Requirements to allow us to develop the architecture. These Core Technical Requirements are largely based on the vision outlined in the 2-IMMERSE project proposal and are described in more detail in the following section.

Now that D4.1 is available, the architecture and core requirements will be reviewed against the details of each of the four trial scenarios as they are defined at this early stage of the project. WP2 will evolve the architecture to reflect the requirements of each of the 2-IMMERSE field trials as they emerge.

In the original 2-IMMERSE project proposal, we said:

"2-IMMERSE will develop an extensible, standards based delivery platform based on re-usable components that will accelerate the development of new immersive multi-screen experiences, accelerate the take-up of the HbbTV2.0 standard and contribute towards its evolution"

From this statement, we can define some guiding principles for how we approach the architecture and development of the platform:

1. Sustainable production of live multi-device experiences – i.e. a cost-effective means to make multi-device experiences in volume (with re-use rather than expensive 'one-offs').
2. Integrate with existing TV broadcasting services, using appropriate standards and practise to produce an industrial strength solution.
3. Accelerate uptake of the HbbTV2.0 and contribute towards its evolution.
4. Leave something that's plausible as a foundation for others to build on. Our solution has to be extensible, has to use open source and potentially be open sourced, and it has to be made accessible to developers.
5. Object-based broadcasting approach – Media is captured and delivered as objects.
6. Designed to work at scale - It has to work at scale if we want broadcasters to adopt this technology

Extensibility and re-use are particularly important; the ability for us to add new services to the platform as the successive service prototypes require additional features, or, for broadcasters to use and extend the platform after the project is finished is essential. Similarly, for experience producers to be able to reuse DMApps and DMApp components is essential for sustainable productions.

Architecturally, we are adopting a 'micro-services' approach. This is comprised of:

1. Services that are: *"Small, focused and doing one thing very well"*
2. *The supporting ecosystem and authoring capability for new micro services*

Both are extensible, scalable and following industry best practices, will give us a very clear separation of concerns between our platform services and the supporting infrastructure.

Although we are very early in the development cycle, we aspire to the following principles:

- Early integration
- Continuous deployment
- Enterprise level software
- Focus on security, scalability, robustness and maintenance

2.2 Core Technical Requirements

The following requirements are considered to be essential to the 2-IMMERSE architecture. They have been inferred from one or more of the four trial scenarios as defined at the beginning of the project. They are all considered to be essential for the architecture and are deliberately expressed in a solution-neutral way.

1. **Association of multiple connected devices** (clients) in a home/school/pub environment, with detection of device features (discovery and launch).

   A multi-screen environment is central to the 2-IMMERSE system, and a key challenge for the project is to enable experiences to be adapted to arbitrary arrangements of connected devices. Discovery of such devices and features is one of the first steps towards setting up an experience.

2. **Delivery, decoding and rendering of multiple media streams** on any client in the environment.

   The processes associated with carrying heterogeneous media streams to a client and decoding them are fundamental to a multi-screen experience. It should be noted that each client may have a different capacity (for example bandwidth and processing resources) to decode media streams, and this will need to be taken into account.

3. **Composition of media** in arbitrary and dynamic layouts/presentations.

   Flexible composition is an essential enabler for interactive, personalised and adaptive multi-screen experiences. The 2-IMMERSE system must enable control of both spatial and temporal media composition.

4. **Synchronised presentation of media** between multiple clients in one or more environments.

   Some 2-IMMERSE use cases describe shared experiences in which media streams must be synchronised between multiple devices in the same environment (intra-location synchronisation) and between devices in multiple environments (inter-location synchronisation).

5. **Lobby/chat room** to allow clients to meet during an experience.

   Some 2-IMMERSE use cases describe experiences which are shared between groups of people in different locations. A mechanism is therefore required to enable people to join a virtual group as part of their experience. A lobby or chat room is the standard mechanism for achieving this.

6. **Management of user identities** to register and authorise access to experiences and enable presence information to be shared.

   2-IMMERSE trials are likely be provided over public networks and therefore need a mechanism to control access to multi-screen experiences. Individual users (or possibly households) will need to be identified if experiences are to be personalised or shared between groups in different locations. Identification is also an important key against which interactions and system behaviour can be recorded in order to analyse how experiences are being used.

7. **Real-time audio and video communication** between multiple home environments.

   Some 2-IMMERSE use cases describe how participants in an experience which is shared between multiple locations have the ability to see and hear each other during part of the experience. Real-time communication places additional requirements on clients involved in a multi-screen experience, including low latency transmission, local capture of audio and video streams and echo control.

8. **Tools to enable production personnel to have live control** over aspects of composition in home/school/pub environments.

9. **User interface on one or more clients in the environments** to interact with and control aspects of the experience, which responds to the devices available in the environment.

   A key feature of the 2-IMMERSE system is the ability for control of the multi-screen experience to be shared between its production team and individual users. Both groups must therefore have sufficient controls to make changes during playout. The type and design of the controls is prescribed when the experience is created, and on the client side their composition must be adapted to the client environment.

10. **Tools and/or data formats to author a multi-screen experience** in terms of layouts, events and interactions.

    2-IMMERSE multi-screen experiences will require a new approach to authoring which is independent of any specific configuration of devices in the user environment. As a minimum this will require new data formats to describe the elements of an experience and the rules for how they can be assembled and interacted with. The project also anticipates the need for new tools to support this process which reduce the barriers to entry for creative professionals as well as the impact on the time and cost for creating an experience.

11. **Each system component (especially each client) logs key aspects of its behaviour** and these logs are aggregated.

    Recording user and system behaviour is an important requirement for the 2-IMMERSE system because it will provide essential insight into how experiences are being used. By building integrated multi-screen experiences from the ground up, the project has a unique opportunity to understand how user attention moves between devices.

12. **Offline analysis** of client behaviour logs after the event.

    2-IMMERSE must provide a mechanism for offline analysis of logs. This does not need to be part of the 2-IMMERSE system itself and could potentially be achieved by enabling logs to be imported into third-party tools or platforms.

13. **Monitoring of key aspects of the system during operation** with option to aggregate and feed back into the experience.

    In addition to recording logs for offline analysis, some 2-IMMERSE use cases describe how data relating to user interactions and viewing behaviour may be presented within the experience (for example within a pub environment). The system must therefore make this data available within both the authoring and delivery/composition processes.

3 System Architecture - Overview

We have taken a 'layered' approach to documenting our system architecture, in order to maximise clarity, and maintain an appropriate 'separation of concerns'.

Our layers are defined as follows:

- Platform Architecture – the high-level architecture of the 2-IMMERSE platform.
- Service Architecture – the services that comprise the platform.
- Application Architecture – how our client applications are architected.
- Production Architecture – how the production capabilities are architected.

The sections that follow describe each of these architecture layers in turn in detail.

4 Platform Architecture

Figure 1 below shows our end-to-end platform architecture at a very high level. This shows a basic flow from production through to platform services, with clients accessing those platform services to present experiences to users. The Application Architecture, Platform Services and Production Architecture are described in detail in the next sections of this document.

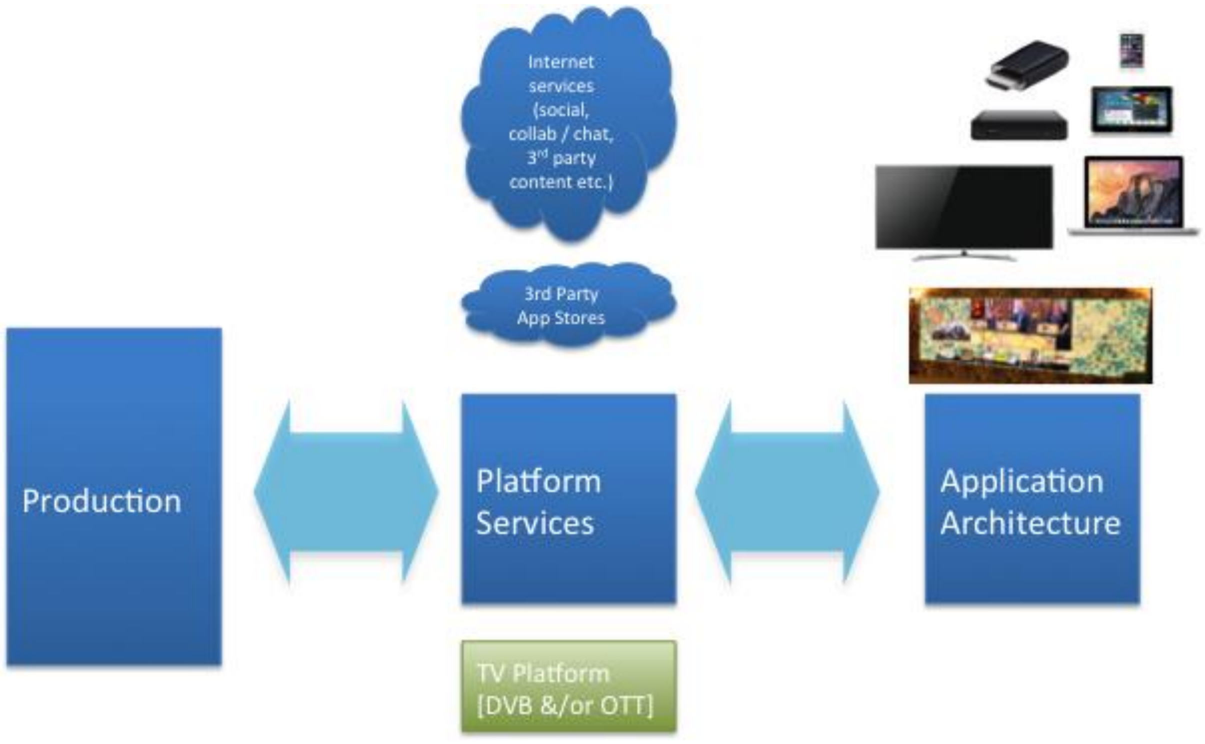

Internet services (social, collab / chat, 3rd party content etc.)
3rd Party App Stores
Production
Platform Services
Application Architecture
TV Platform [DVB &/or OTT]

Figure 1: High-Level Platform Architecture

Figure 1 shows some example 3rd party Internet services as being separate to the Platform; these would typically be used by DMApp components to provide content, data and features specific to these components, but not core system functionality. Examples might include social networks, collaboration / real time communications, live data feeds etc.

We also separate third-party app stores e.g. Apple App store (iOS), Google Play Store (Android), and HbbTV vendor App Stores, as although not considered a core part of the platform, they are a necessary existing mechanism through which client applications will be made available to end users.

The underlying TV Platform is also shown separately; we assume we will be building on an existing TV Platform that is capable of delivering live and on-demand content. We have included this in our service descriptions since it should be considered core, and the production architectures will still need to be able to deliver content through the TV Platform.

Before describing the client application architecture and platform services in detail, we will introduce or recap some important concepts and terms:

- **Experience** – The experience of consuming a Distributed Media Application across multiple participating devices in a context.
- **Context -** One or more connected devices collaborating together to present a media experience. Each context has a contextId unique to its session. There can be many contexts on a single LAN, but a device can only be a member of one context at a time. Devices belonging to the same context must be able to discover each other using DIAL. Devices can join or leave a context at any time.
- **User Experience (UX) Engine -** orchestrates the distributed multi-screen experience, managing each of the participating client devices, and adapting the presentation to the user's environment, their participating devices and preferences. This has been decomposed into two services; timeline and layout.
- **Media Composition Protocol** – a protocol for describing how media objects / feeds should be composited together over time to produce a presentation optimised for a particular presentation device.

5 Application Architecture

5.1 High-level Requirements

2-IMMERSE multi-screen entertainment experiences are composed of many applications configured to work together to deliver the look and feel of a single application. 2-IMMERSE calls this collection a Distributed Media Application, or DMApp.

The application architecture must address the following requirements:

1. Development of experiences must be sustainable.
2. Experiences must be tailored to the content being delivered.
3. The presentation should be consistent on all devices.
4. Functionality must be available on multiple device types.

5.1.1 Sustainable Application Development

Producers need be able to create experiences quickly and efficiently for a large range of programmes if the multi-screen format is to succeed. DMApp development can be made sustainable by:

1. Exploiting commonality between programmes (i.e. genres)
2. Adopting a library of reusable templates and components that can be re-skinned to capitalise on prior investment
3. Adopting technology that allows an application to be written once, but deployed to multiple devices
4. Adopting data-driven components that can be reconfigured for use in different contexts
5. Deferring production decisions to audiences and smart layout engines to reduce the number of device permutations and configurations to author for and test

Perhaps the most impactful way to reduce the cost of developing an experience is to license components from 3rd party developers and to foster a community of open source components. This might be the only option in circumstances where there is limited budget for developing multi-screen experiences in-house.

5.1.2 Presentation

Consistent appearance of components across all devices has a bearing on the perceived quality of experience. Moving a component of functionality from one device to another should preserve its visual appearance. This is difficult without a common renderer running on all devices and it also implies migration of state.

HbbTV 2.0 provides an application environment based on the open web standards of HTML5, CSS3 and JavaScript. This enables developers to author applications once and use them across a range of devices. It provides a basis for migrating functionality from one screen to the next, whilst preserving appearance.

5.1.3 Multi-device Support

Companion screen devices aren't themselves HbbTV 2.0 terminals but they can run environments that are equivalent using web browsers or native applications that wrap web browser functionality.[1]

Using web technology to construct distributed media experiences reduces the number of operating systems to support, however abstractions such as the Television Application Layer (TAL) may still be required to address minor incompatibilities between devices.

Homogenising the application run-time environment by using web technology and exposing low-level media APIs via JavaScript simplifies the task of authoring, testing and deploying a distributed media experience. It provides a cross-platform 'Media OS' against which 2-IMMERSE experiences are authored.

5.2 Application Stack

Devices participating in a 2-IMMERSE experience must implement the stack shown below.

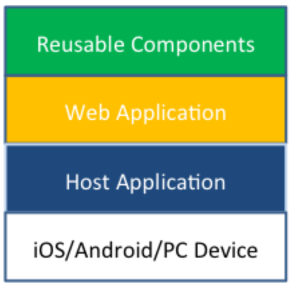


Figure 2: Abstract Application Stack

A distributed media experience is composed of several native and non-native applications running on multiple devices.

5.2.1 Web Applications and Reusable Components

Web applications and reusable components deliver the core user experience. Both are written using HTML5, CSS3 and JavaScript. Reusable components deliver individual features or act as containers that aggregate other components together. Components also leverage other units of reuse such as JavaScript libraries and templates. The Web Application hosts these components and is also responsible for their life cycle. W3C Web Components are a candidate technology for reusable components although other choices are available[2].

[1] See http://www.oipf.tv/web-spec/volume5a.html for minimum browser requirements for HbbTV 2.0.

[2] See https://www.youtube.com/watch?v=5sETJs2_jwo for Netflix's solution that swaps HTML and CSS for primitive boxes and pure JavaScript.

5.2.1.1 Component Life Cycle

Users may personalise their setup by deselecting components or enabling new ones during the experience. The active set of components is also influenced by companion devices joining and leaving the experience and by layout changes triggered by the broadcaster.

For example, a user may choose to enable a component from within the virtual lobby of the ‘theatre at home’ use case in response to a recommendation from a friend received by voice chat. The introduction of this new component may also cause layout changes on other devices.

The active component set can be manually configured from any companion screen device to personalise the experience. A reusable multi-device configuration component is therefore required.

Administrators may choose to limit the availability of components to certain times or restrict access through authentication and DRM schemes.

5.2.2 Host Application

The Host Application provides a common runtime environment across all devices and is responsible for hosting web applications. The Host Application is a native application written using a wrapper technology such as Cordova or Titanium, or a web browser (running native plugins). On an HbbTV 2.0 terminal device, the host application is an implementation of the HbbTV 2.0 profile. The host application’s job is to provide a consistent web development platform on all devices and to expose platform APIs via JavaScript bindings. An HbbTV 2.0 terminal has an application manager that is responsible for launching TV applications and managing their life cycle. The 2-IMMERSE application stack features a similar component for managing web application life cycle on companion devices.

5.2.3 Launcher Application

A device may also optionally run an HbbTV companion screen launcher service to permit HbbTV 2.0 terminals to launch companion screen applications. The Launcher Application is a native application provided by the television manufacturer. Part of the process of launching a companion screen from an HbbTV 2.0 master terminal is proprietary and the manufacturer may elect to support a limited number of companion platforms. This is a consideration to take into account when bootstrapping a 2-IMMERSE experience.

5.3 Device Roles

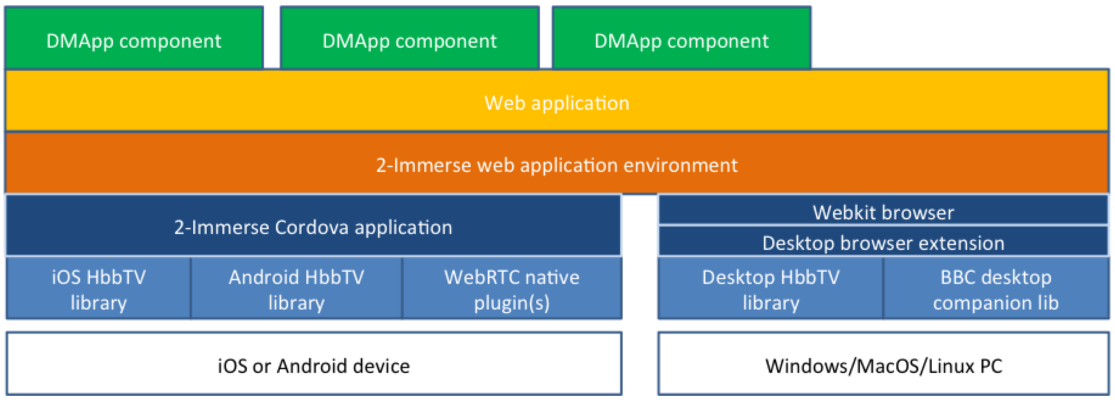
Devices have specific roles within a 2-IMMERSE experience and this gives rise to differences in the application stack specification for each of those devices.

Device Role	**Description**
Mobile & Desktop Companion	HbbTV 2.0 CSS running on Android and iOS phones/tablets and desktop PC/laptops. The desktop companion can be used for integration testing.
HbbTV Master	HbbTV 2.0 Television (due to market Q2 2016) acting as a master device.

Device Role	Description
Emulated HbbTV Master	Placeholder HbbTV 2.0 STB for the development and testing of extended features until HbbTV 2.0 devices are readily available. This stack represents the HbbTV 2.0 profile subset used by 2-IMMERSE experiences and may include features classified as optional in the HbbTV 2.0 specification.
Headless Companion	A headless companion that permits devices lacking a screen to participate in 2-IMMERSE experiences. This is useful for automated testing where it is more practical to spin-up a headless companion instance than a physical device with screens. A headless companion can be run on IoT devices in the home as part of an integrated experience. One example is an IoT light bulb that is dimmed to warn viewers that a programme is about to start, simulating a cinema or theatre experience more closely. The headless companion stack can also be run on embedded devices, servers or in continuous integration environments.

It might be useful to consider a games console application stack too. Games consoles have powerful graphics processing capabilities that make them suitable for hosting a local composition service or the UX engine, including layout and timeline services.

5.3.1 Mobile & Desktop Companion Device Stacks

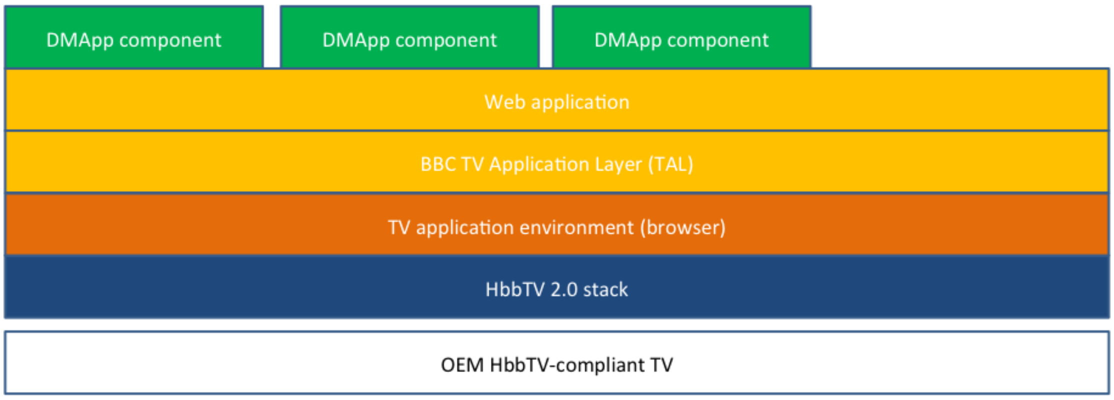


On mobile devices, the 2-IMMERSE Cordova application exposes DVB CSS (CII, WC, TS), DIAL, App2App and WebRTC via JavaScript APIs. The underlying functionality is provided by Cordova plugins written natively for iOS and Android. The BBC iOS companion library and IRT Android companion library will be utilised for this purpose.

On desktop devices, the BBC Chrome extension for companion apps will provide media player control and HbbTV discovery in a browser. DVB CSS (CII, WC, TS) and App2App APIs will be exposed to JavaScript via further desktop browser extensions.

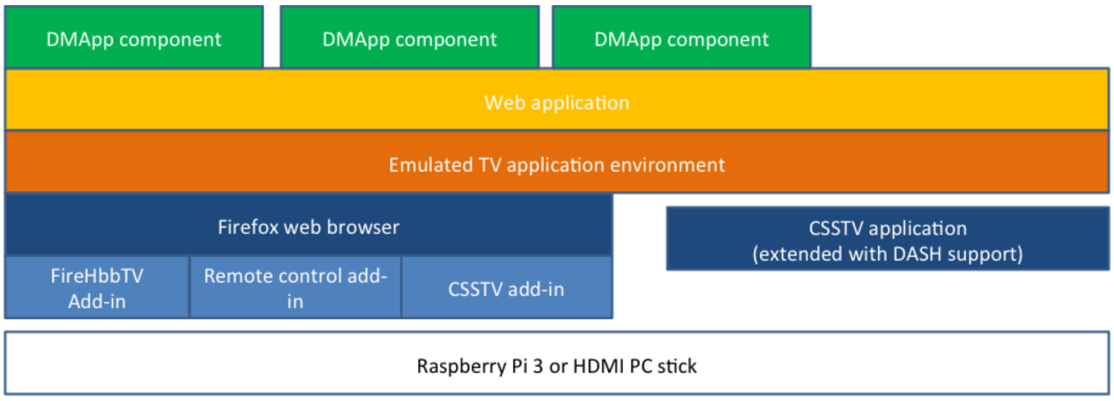
On continuous integration servers, a replacement browser driver can be used to help run automated tests.

5.3.2 HbbTV Master Device Stack



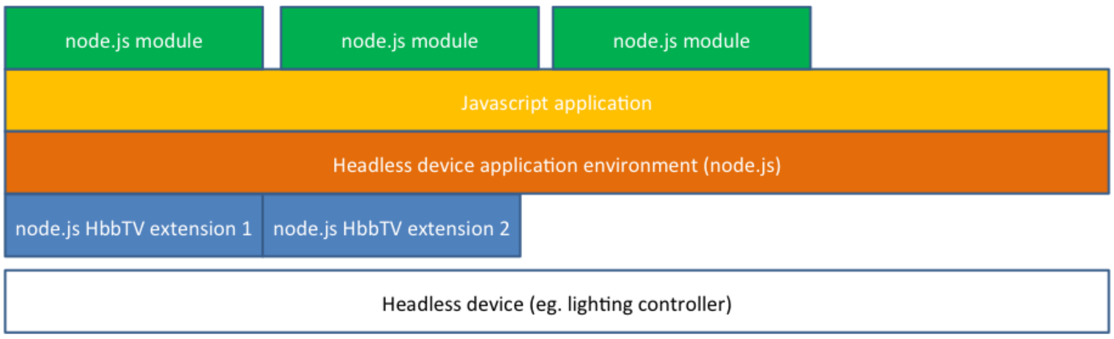
The HbbTV master stack will utilise the BBC TAL library (or equivalent) because not all HbbTV 2.0 devices on the market will provide identical functionality. For example, there are optional parts to the HbbTV 2.0 profile and there will be implementation differences and firmware bugs from one manufacturer to the next.

5.3.3 Emulated HbbTV Master Device Stack



The emulated HbbTV master device may be a small form-factor PC, such as a Raspberry Pi or HDMI stick, running a virtual machine or Docker image to emulate HbbTV 2.0 terminal features. The stack can play MPEG-2 transport streams using the BBC's CSSTV environment, a C++ gstreamer-based prototype extended to include support for DVB-DASH playback. The application environment is provided via a browser such as FireFox or Chromium leveraging a FireHbbTV-like approach to run HbbTV master applications in the browser. An infrared remote control add-in will provide the browser with user interactions and the BBC's "CSSTV in browser" prototype will be leveraged to provide a proxy for UDP traffic and JavaScript support for media synchronisation.

5.3.4 Headless Companion Device Stack


node.js module
node.js module
node.js module
Javascript application
Headless device application environment (node.js)
node.js HbbTV extension 1
node.js HbbTV extension 2
Headless device (eg. lighting controller)

DVB CSS (CII, WC, TS), DIAL & App2App APIs are exposed to node.js via extensions. One such example is Fraunhofer "hbbtv": a module for node.js. It implements the DIAL protocol and the extensions defined in the HbbTV 2.0 specification. The module can emulate both the DIAL client (companion screen) and the DIAL server (HbbTV device).

Node.js modules (as opposed to native extensions) can be used to modularise applications into components and to implement the functionality of the headless device application environment.

5.4 Web Application Architecture

Components provide much of the functionality associated with a web application, but there are architectural requirements that must be satisfied by every web application regardless of which components are activated. The following table highlights common architectural concerns associated with web application development that the application architecture must address in order to support DMApps.

Single-page application (SPA) infrastructure	SPAs offer a more native-app-like experience for the user. Single page apps are distinguished by their ability to redraw any part of the UI without requiring a server round-trip to retrieve HTML. Caching is also used to minimize server round trips. (See http://singlepageappbook.com/goal.html)
Separation of concerns	Ensuring separation of concerns by implementing the request processing logic and application logic separately from the user interface. Choosing a pattern such as MVC or MVVM to simplify event-driven programming of user interfaces
Request Processing	RESTful service calls between the browser and the server are asynchronous. The application architecture and user experience must be designed to handle asynchronous requests
View management	Views observe model changes and redraw the UI automatically using an event change system that listens to notifications from models. Views also

	manage internal view state
Navigation	A consistent navigation structure for the application that's decoupled from application logic
Session state management	The web application must deal with what to store, where to store it, and how long information will be kept. The application must store state (both locally and remotely, when possible) so that users can pick up wherever they left off
Service clients	Client libraries are required by the web application in order to invoke service APIs. They hide the client from details of the communication protocols used to transfer data to/from the server
Connection/disconnection management	"Offline-first" infrastructure is required because network connections can be unreliable or slow. This principle also improves responsiveness of the application's user experience
Component life cycle management	A more specific requirement for 2Immerse web applications is the ability to instantiate/kill reusable component instance(s), hide/show and enable/disable components and refresh or upgrade components.
Logging	Designing an effective logging and instrumentation strategy is important for the security and reliability of the application. Typical events that are logged in a web application include state transitions, component activation, performance, latency, bandwidth, user interactions and analytics.

5.5 Application Architecture Technology Choices

Certain technology choices and decisions remain outstanding pending further investigation. In particular:

1. Choice of web browser wrapper technology (Cordova, PhoneGap, Titanium)
2. Choice of component technology (Web Components, Polymer, X-tag, pure JavaScript)
3. Scene management (Virtual DOM, React-art, Gibbon)
4. Technology choices for native HbbTV/CSS web browser and web view plugins.
5. Choice of state management framework
6. Whether a game application stack is required

5.6 Deployment Considerations

The various application components and layers identified in this section have different deployment models associated with them, and as we develop each of these, and each of the trial experiences, we will need to manage their deployment carefully. This topic is given further consideration in Experience Deployment.

6 Service Architecture

The table below summarises the core services that comprise the 2-IMMERSE platform, and where they could be deployed (in-home or cloud). Note that in-home here refers to the local environment of the participating devices, so for the non-domestic trials this would represent In-school or In-pub.

Service	Description	In-Home	Cloud	Scale
Service Registry	Service discovery	●	●	
Device Discovery (inc. DIAL client and server)	Device discovery and communications	●		
UX Engine a) Timeline b) Layout c) Server-based Composition	DMApp orchestration, adaptation and composition	●	●	per household per household per device / per composition permutations
Timeline Synchronisation a) In Home b) Between Home	Multi-screen content synchronisation	●	●	per household per session
Content Protection/Licensing Service	Media content protection		●	
Identity Management and Authentication	Identity management and authentication		●	
Session (lobby) Service	User group discovery and management		●	
Call Server (SIP)	Real time communications services		●	
Logging	System activity monitoring		●	
Analytics	Platform usage and performance insights		●	
Origin Server/CDN	Media object and DMApp component distribution		●	
TV Platform	Live and on-demand TV content distribution		●	

Table 1: Core Services

In addition to these core services, additional services will be required on a trial specific basis; for example, backend services may be required to support particular DMApp components that are part of the trial experience, or, specific services may be required as part of the production workflow for that trial.

As noted in the table above, for the UX Engine and Sync services, we can see two envisaged deployment options; one where these services are deployed in-home (for example running on the TV device), and alternatively with these services running in the cloud. These two models are shown in Figure 3 and

Figure 4 below. We do not envisage supporting both models in a single trial, but will likely decide on a particular deployment model on a trial specific basis.

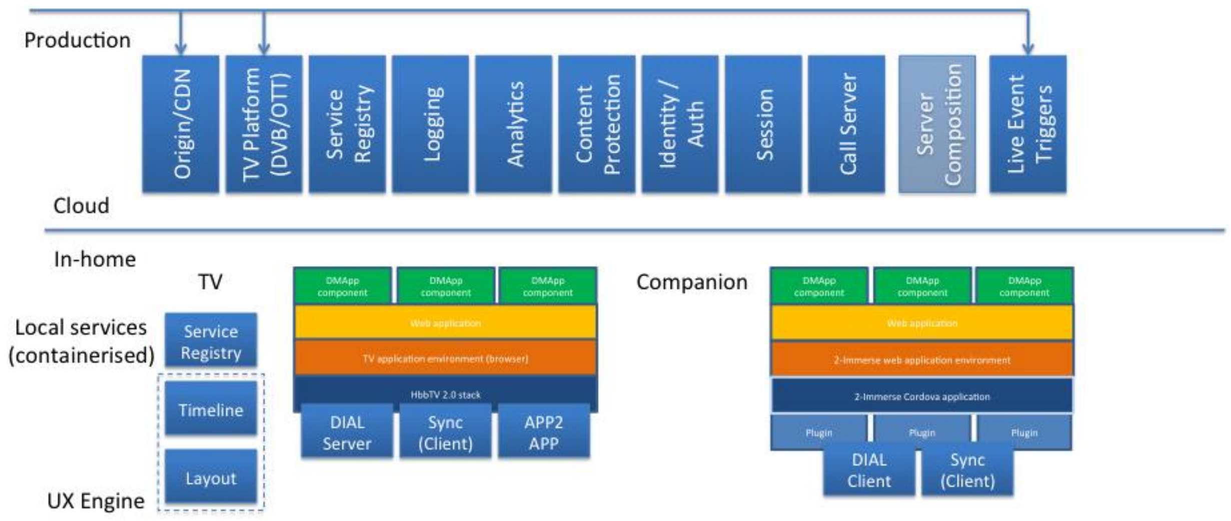


Figure 3: Service / Client Deployment - in-home UX Engine Services

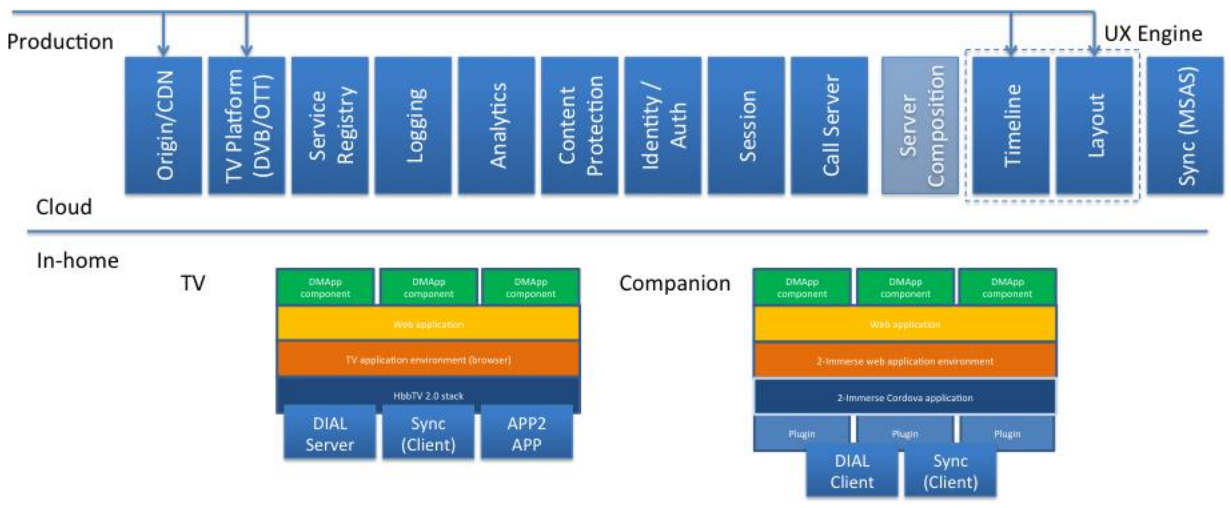


Figure 4: Service / Client Deployment - cloud UX Engine Services

6.1 Service Descriptions

The sections that follow define services, provide descriptions of their functionality and interfaces, and where appropriate identify selected technology.

6.1.1 Service Registry

Client devices and applications that need to use platform services, need to be able to establish their availability, and where they are hosted. A typical mechanism for enabling this is a service registry, which is deployed to a well-known location (the address of which would typically be part of the application configuration). As the platform services are provisioned, they register with the service registry so that they are discoverable by the applications wishing to use them.

Where we may have services deployed in-home or in the cloud, our service registry solution needs to be able to accommodate this efficiently.

Selected Technologies

There are a number of widely available and open source implementations of service registry (Consul, Spring Cloud, ZooKeeper et al.), some of which will be integrated into a particular cloud PaaS (platform as a service); so it may be that selection of a PaaS will drive the use of a particular service registry.

The particular solution we will adopt is still to be determined.

6.1.2 Device Discovery

The device discovery service enables applications, e.g. running on companion screen devices, to detect available TV sets and STBs and services they support, e.g. sync service, in local networks. Additionally, the service allows applications to be launched on the discovered device. The service also provides a way to check whether an application is already running on a discovered device, via application to application (App2App) communication.

Device Discovery Architecture Overview

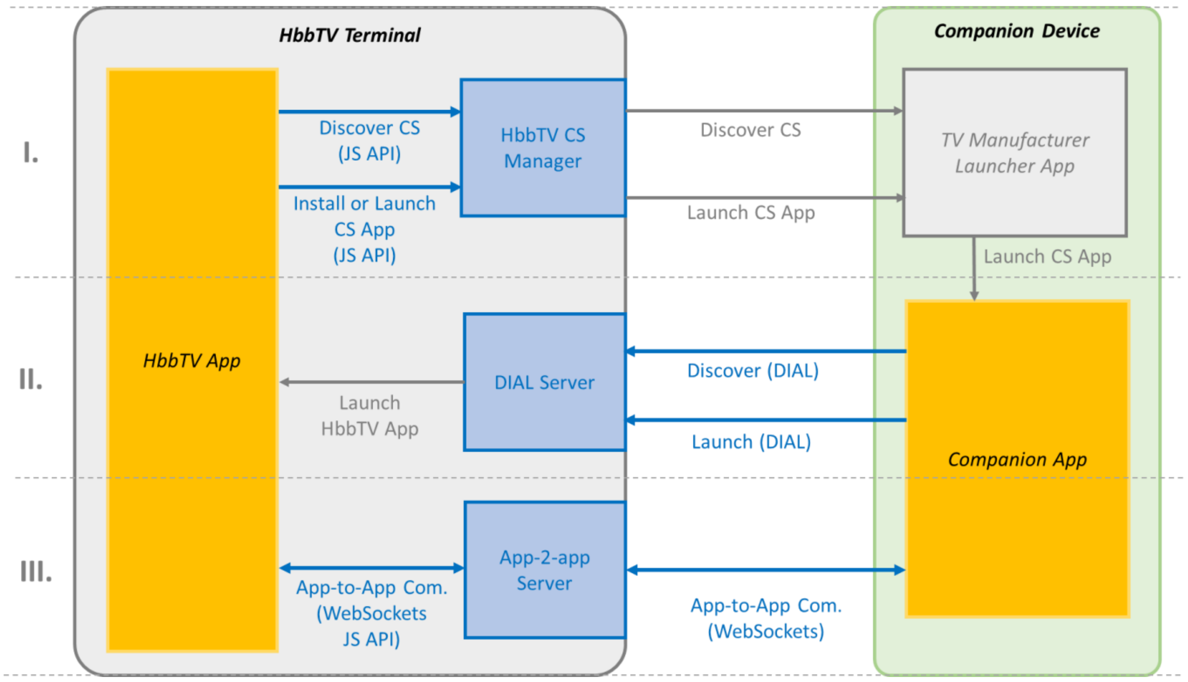


Figure 5: Device Discovery Architecture Overview Diagram

At the time of writing it is assumed 2-IMMERSE will adapt the protocols as for Discovery, Launch and App2App Communication as defined in the HbbTV 2.0 specification. Figure 5 provides a high-level overview of the specified components, the information exchanged between these components and the protocols used to transfer the information between components. The diagram uses the following notation:

- ***Boxes with round corners*** denote devices (i.e. HbbTV terminal and Companion Device).
- ***Boxes with straight corners*** denote software components (e.g. App2App Server, DIAL Server, Broadcaster Companion App).
- ***Arrows*** indicate the direction of information flow. Labels indicate what information is exchanged. In those cases where the information flow is within the scope of the HbbTV 2.0 specification (blue arrows), text in brackets provide details on the interface or protocol used to transfer the information.
- ***Blue colour*** indicates that the respective interface and protocol as well as the behaviour of the respective component is specified in the HbbTV standard.
- ***Grey colour*** indicates that the specification and implementation of the respective components, protocols or interfaces is under the authority of the TV manufacturer. It is out of scope of the HbbTV 2.0 specification.
- ***Green colour*** indicates that indicates that the specification and implementation is under the authority of the Companion Screen manufacturer.
- ***Orange colour*** labels those components who are under the authority of the provider of the user experience (e.g. broadcaster).
- ***Dashed lines*** divide the diagram into three areas, each labelled with a roman number.

Area I. covers the elements of the Discovery Architecture that deal with companion-screen interaction initiated by an HbbTV application. Major components are the HbbTV CS Manager on the HbbTV terminal side and the TV Manufacturer Launcher App on the companion screen side. The HbbTV CS Manager exposes a JavaScript API that allows the HbbTV Application to initiate the Discovery of companion screens and subsequently request the launch of a web-based or native app on the respective device. In case a desired native app is not available on the targeted companion device yet, the HbbTV application can request the installation of that native app. It is important to note that the protocols for Discovery of companion screens and the Launch of Apps on these devices is out of scope of the HbbTV 2.0 specification, but under the authority of the TV manufacturer. To be able to Discover users' devices on the local network it as crucial that they have the TV Manufacturer Launcher App installed on their companion screen device.

Area II. covers components, interfaces and protocols that allow Companion Screen Apps to discover HbbTV terminals and to launch HbbTV apps on these devices. For this purpose HbbTV 2.0 adapts the DIAL protocol (http://www.dial-multiscreen.org/). For discovery, DIAL references SSDP (simple service discovery protocol) which is part of the UPnP stack. To discover an HbbTV terminal, Companion Screens send a UDP message to multicast address 239.255.255.250. Terminals willing to connect respond with a Notify message. The message's header contains the UPnP Device Description URL, which is used to retrieve the REST endpoint of the DIAL service. To launch an HbbTV App on the HbbTV terminal the companion app sends a HTTP POST request to this endpoint containing an XML AIT. Via HTTP GET the Companion Application can retrieve the endpoints for app-to-app communication (see Area III.) and companion screen synchronisation.

Area III. covers those parts that enable bi-directional exchange of messages between Companion Screen Applications and HbbTV Applications. Messaging is done via the WebSocket protocol. For this purpose the HbbTV terminal provides a WebSocket-Server (App2App Server) that provides endpoints for the HbbTV application and Companion Screen Applications.

Interfaces

- Query device: Listens for local device discovery queries, responds to issuer with location of device description.
- Get device description: Returns description of the device including supported services:
    - Application launch
    - Local synchronisation
    - app2app communication
- Launch application: Launches an application on the device.
    - Launch is approved by either checking a whitelist or getting user approval
    - Returns status, whether launch was successful, denied by user or an error like application could not be loaded.
- Open app2app connection: Open a connection to an application running on the device. Returns a connection, which gets paired with an application if that is already running or later on when it gets started on the device. A pairing completed message is send afterwards.

6.1.3 Timeline

The timeline service is responsible for managing the timeline of an experience as it is presented over a set of participating devices (i.e. a context). It uses authored timeline metadata and optionally live triggers to determine what media objects / DMApp Components are available for potential presentation as the timeline of the experience progresses.

As the current set of media objects / DMApp Components changes (either through reaching an event in the timeline metadata, or on receiving an external event trigger), the timeline service will send an updated component list to the corresponding layout service instance.

The format for timeline metadata is still to be specified, but will need to describe the changing availability of media objects and DMApp Components that comprise an experience over the duration of that experience.

This service could either be deployed in the cloud (an instance for each context), or within a client device.

The timeline service has the following interfaces

- Load experience (content id)
- Ingest timeline metadata (content id, URL)
- Sync
- Event Trigger
- Component list

6.1.4 Layout

The layout service is responsible for managing and optimising the presentation of a set of DMApp Components across a set of participating devices (i.e. a context). Given a set of media objects / DMApp Components, authored layout requirements, user preferences, and the set of participating devices and their capabilities, the layout service will determine an optimum layout of components for that configuration. It may be that the layout cannot accommodate presentation of all available components concurrently.

The service instance maintains a model of the participating devices in the environment, and their capabilities e.g. video: screen size, resolution, colour depth, audio: number of channels, interaction: touch etc.

The layout requirements will specify for each media object/DMApp component, layout constraints such as min/max size, audio capability, interaction support, and whether the user can over-ride these constraints. Some of these constraints may be expressed relative to other components (priority, position, etc.).

The grammar expressing layout requirements is still to be specified.

Layout changes can be triggered by a number of events:

- On receiving an updated component list from the layout service
- On client devices joining or leaving a context (through a manage context call).
- On receiving a manage component call from a client device application.

As such layout updates need to be pushed to all participating clients (for example, by a mechanism such as WebSockets).

The data format describing layout that is pushed to clients will be logically related to the Media Composition Protocol defined in the Sever-Based Composition service, since the layout format will essentially describe the composition of DMApp components on a device display, however the format of this data is still to be specified.

This service could either be deployed in the cloud (an instance for each context), or within a client device.

The layout service has the following interfaces:

- Manage context (Create / Join / Leave) (capabilities)
- Component List (a list of the currently available components)
- Manage component (Hide/Show/Move/Clone) - subject to role/permissions
- Load experience (content id)
- Ingest layout requirements (content id, URL)
- Layout (metadata describing the current layout of components across all participating devices)

6.1.5 Server-Based Composition

The composition service provides on-the-fly compositing of media feeds on client devices or via dedicated servers for client devices with limited bandwidth, performance or battery life. Examples of composition include:

- Picture-in-picture
- Cropping and scaling
- Video tiling
- Animation
- Info-graphic rendering
- Camera/VT transitions (cross-fade, fade to white/black, straight cut)
- Visual effects
- Audio mixing
- Rendering of textual overlays e.g. subtitles

Responsive multi-screen layouts require clean feeds to be composited after broadcast in order to adapt to the changing client ecosystem of users, apps and devices. The composition service runs downstream from production and can be located:

- At the origin/head-end as a permutation cache
- In the cloud
- On a networked device local to the experience
- On a client device

The composition service is capable of generating outputs that are tailor-made for the resolution, colour depth and bandwidth requirements of each client device. Lightweight compositing operations such as overlaying menus will be performed on client devices such as STBs, televisions, tablets and smartphones. These client devices still have the separate task of synchronising the playback of composited media feeds generated by the composition service.

A more detailed description of the composition service is given in Annex C.

Interfaces

The composition service has interfaces for:

- Configuration (e.g. secure licensing server, CDN origin server, media feeds)
- Media Composition Protocol
- Exchanging DRM keys and authentication
- Remote control and monitoring
- Life-cycle management

The Media Composition Protocol itself is part of the composition service's interface.

6.1.6 Timeline Synchronisation

A multi-device synchronised experience consists of a number of media objects that will be presented on separate participating devices (multi-device presentation is managed by the Layout and Timeline services, see Section 6.1.4), at times specified on the experience timeline. These media objects are manifested in the experience by DMApp components and can be discrete such as images, infographics or continuous such as live/on-demand audio/video streams.

The Timeline Synchronisation Service enables DMApp components on devices participating in an experience to synchronise to a source of timing information (a timeline) representing the progress of the experience. In an *intra-location* synchronisation scenario, companion devices synchronise their content playback to a master device e.g. a TV playing a broadcast/IP-delivered stream; all devices residing on the same network. In this particular context, the experience timeline (also called the **synchronisation timeline**) is the timeline of the master device's content e.g. the timeline of a TV programme. Provided mappings from the Synchronisation timeline and the DMApp media objects' timelines are available, the companion devices can synchronise their DMApp components to the master TV.

In another scenario (*inter-location* sync), devices at different locations synchronise their content playback as part of a distributed synchronised experience. In this inter-location synchronisation configuration, the synchronisation timeline can be one of the following:

1) the timeline of an elected master device,
2) a mutually-agreed timeline, achieved through timing contributions[3] by independent peers, or
3) a timeline set by a central coordinator e.g. the experience timeline as set by the Timeline Service

[3] Although, it is possible for a mutually-agreed timeline to be achieved independently through consensus by individual peers through rounds of state update, we refer here to a centralised service that receives timeline progress updates from all peers and computes a reference timeline position that all peers should adhere to.

The Timeline Synchronisation Service provides Synchronisation as a Service (a SaaS) to devices, allowing them to subscribe to synchronisation-timeline progress updates. Based on these timeline progress updates, the devices can decide on how to adapt their playback to achieve synchrony.

In particular, a *Synchronisation Server* (a Timeline Synchronisation Service provider) and a *Synchronisation Client* (a service consumer e.g. devices in the experience) have distinct responsibilities in achieving the synchronised experience.

A Synchronisation Server drives the experience by collecting/distributing timeline updates and making decisions about the reference timeline position of Synchronisation Clients (SC) after every round of synchronisation. It essentially performs the same functions as the MSAS (Media Synchronisation Application Server) component in an HbbTV 2.0 terminal; in fact, in intra-location synchronisation scenarios, we may assume that the Synchronisation Server *is* the MSAS.

In more details, the Synchronisation Server performs the following:

- Collects timeline positions[4] of DMApp components from the individual devices (the synchronisation clients, SCs)
- Calculates delay differences between the media playout of the synchronisation clients, and creating Control Timestamps (a reference timestamp) based on this. The Control Timestamp is usually a position on the Synchronisation Timeline.
- Distributes Control Timestamps to SCs to suggest the timing of presentation that each SC should align to in order to achieve synchronised timing of presentation across all SCs.
- Optionally receives Correlation Timestamps from the Timeline Service, and uses these to translate the timeline of the media objects and Control Timestamps.

Synchronisation Clients, on the other hand, perform the following functions:

- Provide periodic timeline updates of their media playout as Actual, Earliest and Latest - Presentation Timestamps.
    - The earliest and latest presentation timestamps represent the time range of the media that is currently loaded in the player's buffer
    - The timestamps may be converted to positions on the synchronisation timeline
- Receiving control timestamps from the Timeline Synchronisation Service and adapting their media playout to reflect this changed relationship between their content timeline and the WallClock.
- Optionally receiving Correlation Timestamps from the Timeline Service, and using these to translate the timeline of the Control timestamp to its content timeline.
- If performing the master role, provide content identification updates to SCs to inform them about the content/programme it is playing.

All presentation timestamps reported by Synchronisation Clients are read with respect to a common time clock; they actually represent a pair of time values read at the same time (time on content timeline, time of common clock). To this end, all actors in the model share a common time reference via a shared WallClock[5]. This is achieved by each device maintaining a local WallClock instance synchronised to a master WallClock using clock synchronisation techniques. An overview of the WallClock service is given in Annex B.

[4] Actual presentation timestamps, Earliest Presentation Timestamps and Latest Presentation Timestamps from DMApp components (Synchronisation Clients) are actually sent to the Synchronisation Service

[5] The WallClock is a software clock at each device/service that is kept synchronised to the WallClock instance on the synchronisation master terminal via time synchronisation schemes (e.g. CSS-WC, NTP, etc).

We illustrate the roles of the Synchronisation Service and its clients by presenting an abstract model of their operation in Figure 6. The Timeline Service, the Timeline Synchronisation Service and its clients are shown here as abstract components; no assumptions are made as to where they run (on same devices or on the local network or in the cloud).

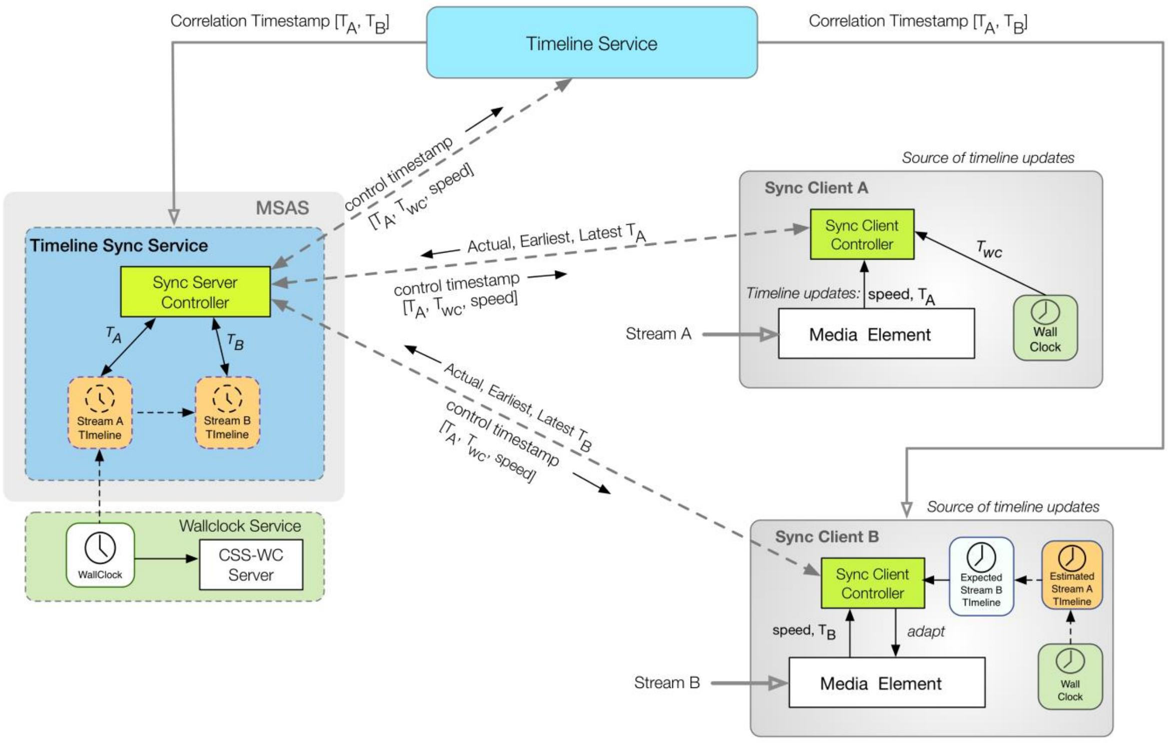

Correlation Timestamp [T_A, T_B]
Timeline Service
Correlation Timestamp [T_A, T_B]
control timestamp [T_A, T_{WC}, speed]
MSAS
Timeline Sync Service
Sync Server Controller
T_A
T_B
Stream A Timeline
Stream B Timeline
Wallclock Service
WallClock
CSS-WC Server
Actual, Earliest, Latest T_A
control timestamp [T_A, T_{WC}, speed]
Source of timeline updates
Sync Client A
Sync Client Controller
T_{WC}
Timeline updates: speed, T_A
Stream A
Media Element
Wall Clock
Actual, Earliest, Latest T_B
control timestamp [T_A, T_{WC}, speed]
Source of timeline updates
Sync Client B
Sync Client Controller
Expected Stream B Timeline
Estimated Stream A Timeline
speed, T_B
adapt
Stream B
Media Element
Wall Clock

Figure 6: Abstract Timeline Sync Model – Sync Client B synchronising to Sync Client A

All presentation timestamps reported by Synchronisation Clients are read w.r.t. to a common time clock; they actually represent a pair of time values read at the same time (time on content timeline, time of common clock). To this end, all actors in the model share a common time reference via a shared WallClock. This is achieved by each device maintaining a local WallClock instance synchronised to a master WallClock using clock synchronisation techniques. An overview of the WallClock service is given in Annex B.

In Figure 6, two Synchronisation Clients A and B play distinct video streams StreamA and StreamB respectively as part of an authored experience (retrieved by the Timeline Service). Sync Client B needs to synchronise to Sync Client A; Sync Client A is the master device. Both Sync Clients A and B share a synchronised WallClock with the Timeline Synchronisation Service. The clock synchronisation is achieved using the WallClock service. Each Sync Client sends timeline updates to the Timeline Synchronisation Service to indicate the current position on its timeline. The timeline update contains the Actual Presentation Timestamp, the WallClock time when the Actual Presentation Timestamp was read and Earliest/Latest available Presentation Timestamp of content being played.

Based on timeline updates from Sync Clients A and B, the Timeline Synchronisation Service can build an estimate of each client’s content timeline (dotted clock outlines in Timeline Sync Service in Figure 6). Using a correlation (T_A, T_B) from the Timeline Service, the Timeline Synchronisation Service can map time values from each timeline to the other. Based on the Actual Presentation Timestamps, Earliest Presentation Timestamps and Latest Presentation Timestamps from each client, the Timeline

Synchronisation Service suggests a presentation timing for each client. Client B can then adjust its playback to this new presentation timing.

If the timestamps from B indicate that it may be hard to catch up with A, then the Timeline Synchronisation Service may instruct client A to decrease its presentation speed by sending it a control timestamp as well. The Timeline Service itself functions as a Synchronisation Client if it assumes the role of the synchronisation master (this is outlined in more detail in Annex B).

The Timeline Synchronisation Service provides the following interfaces:

- *Enable Synchronisation* - establish a timeline synchronisation service instance per experience and make Timeline Synchronisation interface endpoint known to clients)
- *Timeline Synchronisation interface* - for master timeline negotiation, timeline update collection/re-distribution, computed reference time position distribution
- *Content Identification & Other Information* (for content identifier and service-endpoints dissemination to clients)
- *Rebase Experience Timeline Correlation*– (update experience timeline/ WallClock correlations if post-production latencies are introduced)

Selected Technologies

There are a number of alternative technologies and approaches to achieving media synchronisation, each delivering different synchronisation accuracy and convergence delays. The following solutions are favoured candidates to achieving WallClock Synchronisation and Timeline Synchronisation in both intra-location and inter-location synchronisation configurations. They have been selected for their synchronisation accuracy and the suitability for both local and distributed deployment configurations. A local deployment configuration will allow independent instantiations of our platform (e.g. for demos) and will leverage local-network performance to deliver more accurate synchronisation. Other reasons are their availability as open standards (obviates tie-ups with proprietary solutions) and the existence of relatively mature implementations.

Timeline Synchronisation	DVB-CSS (HbbTV 2.0 Stack) for intra-location Sync, cloud-based DVB-CSS variant for inter-location sync
WallClock Synchronisation	DVB-CSS WallClock Synchronisation Protocol, W3C Web Timing API
Content Identification Service	DVB-CSS CII protocol

A more detailed explanation of how to achieve both inter- and intra-location synchronisation based on the technology selections is provided in Media Synchronisation (Annex B). In particular, Annex B describes how the actors and interactions in our model are mapped into existing DVB-CSS component roles and protocols.

Available implementations:

- Synchronisation Client on mobile devices: BBC’s iOS Sync Library, IRT’s Android Sync Library
- Synchronisation Client on desktop browsers: BBC’s JS Sync Library

- Timeline Synchronisation Server/Synchronisation Client on TV: BBC's DVB-CSS TV Emulator

6.1.7 Content Protection and Licensing

Media content protection is where access (playback and/or download) to media content is limited to users/accounts which are currently suitably authorised, in such a way that access cannot be easily propagated to other users, or to the current user once any authorisation has expired or been revoked.

The requirement that access cannot be propagated and is limited to authorised usage imposes constraints on the authorisation model and additional technical requirements. This is because the end-user of the media (the person(s) viewing/consuming it) is also an adversary who could otherwise consume the media data and then replay it to non-authorised users, or consume it by a means that is not authorised. Policy and technology to implement these restrictions is referred to as Digital Rights Management (DRM).

Technical mechanisms to do this include:

- Contacting a remote license or key server when media is about to be accessed, to check whether and how content can be consumed, and/or what keys are required to decrypt it.
- Transport encryption: encryption/protection of data being transmitted against passive or active snooping.
- Encryption at rest: encryption of data (temporarily) stored in user controlled/owned devices.
- Obfuscation of keys, credentials and/or decryption mechanisms in software.
- Obfuscation of keys, credentials and/or decryption mechanisms in hardware.
- Obfuscation of and/or technical barriers around hardware paths which carry unencrypted media or key data.

Which mechanisms are used and how is a trade-off between the level of protection required, hardware availability/requirements, client platform support, technical cost/difficulty, and media distribution platform. Protection mechanisms implemented in software are simpler to implement and more portable however offer less protection than hardware-based mechanisms. This is because the level of competence, cost and difficulty required to read or modify content protection software is orders of magnitude less than that required to reverse-engineer or modify content protection hardware.

Current commercial hardware platforms for consuming content, including a significant subset of TVs, tablets and smartphones, and a smaller subset of PC/laptop type devices, include hardware protection mechanisms. However, development platforms such as might be used to emulate a HbbTV 2.0 TV, are unlikely to include any hardware support for DRM/protected content playback. Therefore, any media content to be used in a trial with development devices which needs to be protected, will only be able to be protected using software-based mechanisms. The mechanism by which content is protected would need to be agreed with the relevant rights holders/owners and may limit the choice of what content to use.

Possible content protection types which could be used in a trial using development devices include:

1. No encryption at all, only authorisation.
   This is probably insufficient, as it is vulnerable to trivial snooping.
   This does not require a content protection service.
2. Transport encryption (TLS) with authorisation.
   This may be sufficient as it prevents trivial recovery of the media by snooping.
   However this requires that the media distribution platform (typically a CDN) implement validation of authorisation credentials and TLS for media download. This is unlikely to scale well or be compatible with current media CDN platforms.
   This requires that the media distribution platform uses the identity management and

authentication service for access control, but does not otherwise require a content protection specific service.

3. Encryption of media data, with keys stored within an unencrypted part of the media data.
   This imposes a slight inconvenience on an attacker, who would be able to decrypt all items of protected content even if not originally authorised.
   A standardised implementation of this is using ClearKey with MPEG-DASH CENC.
   This does not require a content protection service.
4. Encryption of media data, with pre-shared key(s).
   An attacker who extracted the pre-shared key(s) from the executable would be able to decrypt past and future items of protected content, even if not currently authorised.
   This does not require a content protection service.
5. Encryption of media data, with key(s) retrieved separately using transport encryption and authorisation.
   This is the mechanism used for protected BT Sport HLS content on tablets/smartphones, see section 9.5.1.6.
   An attacker would need to be authorised, to extract the key(s) used to decrypt the content, and these key(s) would not allow decryption of other items of content.
   In the case of BT Sport HLS, a key server is contacted for each individual segment (10s chunk of media) during playback. This imposes scalability requirements on any key server implementation used in a trial. These scalability requirements could be reduced by sharing keys between multiple media segments, or by using the same key for the entirety of the item of content.
   This requires a content protection service, which acts as a key server and uses the identity management and authentication service to validate user credentials.

Of the possible content types listed above, only type 5 requires the addition of a specific service for content protection. This service would act as a key server.

Selected Technologies

MPEG-DASH CENC is a standardised mechanism to encrypt fragmented MP4 media files as used in MPEG-DASH using a common key(s), and optionally include proprietary metadata for one or more DRM schemes to be able to independently decrypt the content. This could be used as the media format for options 3 to 5 above, optionally including a proprietary marker to label any choice of scheme in options 4 and 5.

In a production system where protected content does not need to be consumed on development devices without hardware support for protected content, MPEG-DASH CENC could be used with one or hardware-based commercial DRM systems. Commonly used examples include: PlayReady, Marlin, Widevine, FairPlay and Adobe.

6.1.8 Identity Management and Authentication

Components of the system including the lobby, social/communications functionality and access control for media content may require that users (individual persons or groups of persons) can be identified as being associated with an account. Users could be invited to create an account and enter their identity details and credentials to authorise access to that account, at first use, or in the case of a trial, may be allocated an account in advance.

Authentication is used within the architecture for:

- An entity to prove its identity to other entities by proving that it has the required authorisation credentials.
- An entity to be authorised or unauthorised to access user data, media content, services, or perform other actions.
- An entity to assign or revoke authorisation credentials to/from an entity.

The authentication architecture contains a number of entity types:

- Users (individual persons or groups of persons)
- Accounts
- Client devices
- Client applications
- 2-IMMERSE platforms
- 3rd party platforms

Possible authentication scenarios include:

1. A user signs into an account using a client application. Authorisation for that account is stored on the client device for future use by that client application.
2. A user returns to a client application that they have previously authorised to access an account, the client application signs into the account without requiring any credentials to be input.
3. A user revokes authorisation credentials previously assigned to a client device and/or client application.
4. A user creates, deletes, accesses or modifies an account. The authentication model may require that some operations such as these are privileged and require inputting credentials again or inputting separate credentials, which are not then stored on the client device.
5. A user creates, deletes or modifies a subsidiary account associated with a primary account. The subsidiary account may share some subset of the capabilities or access rights of the primary account. This may be useful for child accounts with some form of parental access limitations, for example.
6. Account identity and/or credentials for a 3rd party platform are associated with an account on a 2-IMMERSE platform, or vice versa.
7. A 3rd party platform accesses or modifies an account or associated data on a 2-IMMERSE platform, or vice versa.
8. A user uses a client application/device to play an item of protected media content. Authorisation may involve authorisation/communication with platforms and/or may require that the user, account, client application and/or client device is identified/authorised.
9. A user uses a client application to connect to or control another client device. This may require that the user input suitable credentials, if not already stored on the client device, and/or that the user performs a confirmation on the device being connected to or controlled. This may vary depending on the locations of the devices, for example less authentication may be required if both devices are present on the same LAN.

10. A user uses a client device to create, delete or join a limited-access shared group which requires credentials to access. A possible example of this is a protected chat where only persons who have been given credentials out of band are permitted to join.
11. A client application or client device securely connects to a platform (2-IMMERSE or otherwise) and each verifies and/or authenticates that the other end of the connection is the (type of) entity which is expected.
12. A client application or client device downloads a software update. Platforms such as iOS and Android already have a comprehensive framework for this, but on other platforms if secured updates are required it may be necessary to use a secured connection (typically TLS) and/or to use signed update packages.

OAuth is an authentication framework which could be used for authenticating client devices/applications. OAuth is a standardised general framework for authentication using tokens. It includes support for varying access types on a per-token basis and authentication flows such as entering the username and password on a different application or device than the application to be authorised. (See Section 9.5, Authentication and Security).

Authentication that the other end of a network connection is the (type of) entity which is expected can be done using certificates, typically using TLS.

3rd party platforms which client applications/devices and/or 2-IMMERSE platforms may require authentication to interact with might include:

- BT (BT Sport), for access to sports media content/data.
- Dorna, if access to MotoGP data or media content is not routed via BT.
- Pub-specific platforms, either within the pub premises or a pub-specific remote service.
- Theatre-specific platforms.
- Education-specific platforms.
- Social media platforms, if client applications or a 2-IMMERSE platform are to interact on social media on a user's behalf.
- VOIP/real-time comms platforms.

BT authentication appears to be SAML (Security Association Markup Language) based, however this does not require any other parts of the system to also use this authentication data format.

Authentication and management of user credentials, and generation of authentication tokens, for all parts of the 2-IMMERSE platform is encapsulated within a single identity management and authentication service. This allows Single Sign On (SSO) within the 2-IMMERSE platform. Users use the same account and credentials to access all parts of the platform (subject to user/account entitlements).

The particular identity and authentication functionality required is likely to be specific to the user-stories of each prototype service trial, however these different scenarios will be implemented by means of a single authentication service which provides identity and credential management functionality common to all use cases.

6.1.9 Session (Lobby) and Call Services

The session (lobby) service allows peers to join a named group to discover and communicate with each other. It acts as an introductory service based on people's names as opposed to IP addresses. Peers can join and leave the lobby at any time and everyone in the lobby is notified when this happens. It's possible to join more than one lobby concurrently. The lobby doesn't exist in any form prior to the first peer joining and ceases to exist after the last peer has left. It is possible to request a list of the peers that are in the lobby. This is useful for late joiners who have missed all prior join notifications.

6.1.9.1 Broadcasting

The lobby service can broadcast user-defined messages on behalf of a peer to all other peers in the lobby. This is useful for signalling changes in the state of a shared experience and to synchronise actions amongst the peers without having to establish separate peer-to-peer connections. An example would be an application-defined message instructing each peer to begin playing a media stream.

6.1.9.2 Meta-data

Meta-data can be passed to the lobby when a peer joins, but the lobby service does not impose a schema on it. For example, the data can describe a person's name or the data can be application defined. This meta-data is automatically distributed to all peers via join notifications.

6.1.9.3 Signalling

The lobby service implements signalling for WebRTC applications via WebSockets and XHR. Signalling allows initiation of peer-to-peer sessions by exchanging control messages that initialise or close communication and report errors. Peers can use the lobby's signalling mechanism to exchange ICE candidates obtained from STUN/TURN servers. ICE candidates are the public IP addresses and ports that peers should use to communicate with each other and are the result of establishing NAT punch-through or relay. Signalling is also used to negotiate codecs, video resolutions and communication protocols via Session Description Protocol (SDP). Transport type can also be negotiated as reliable (TCP-like) or unreliable (UDP-like).

Once signalling is complete, peers can chat directly with one another using real-time video, audio and text messages, courtesy of WebRTC. Peers can also exchange arbitrary binary payloads using application-defined protocols.

6.1.9.4 Hosting

The lobby service is cloud hosted and uses secure web sockets and HTTPS XHR to communicate with peers. It leverages WebRTC's security for peer-to-peer communications.

6.1.9.5 Lobby Architecture Overview

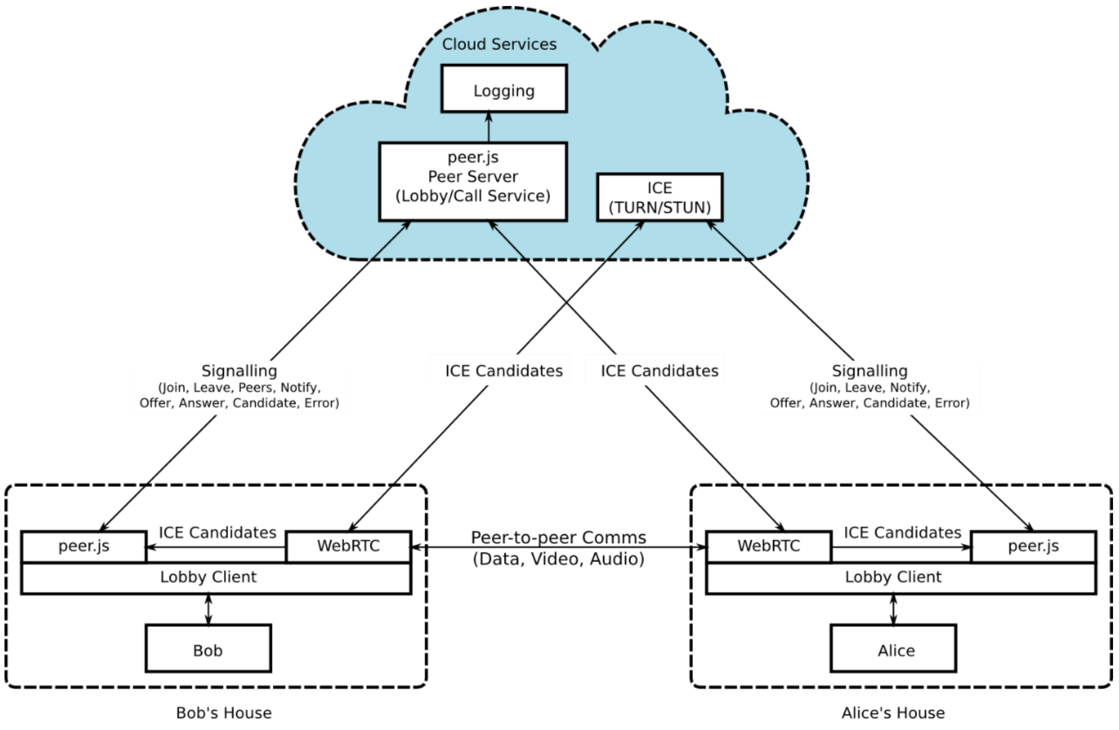


Figure 7: Lobby Architecture

6.1.9.6 Selected Technologies

- Telepresence: WebRTC
- Adapter.js: A WebRTC adapter shim (https://github.com/webrtc/adapter)
- Peer.js: Call service (http://peerjs.com/)
- PeerServer: A server for PeerJS (https://github.com/peers/peerjs-server)

6.1.10 Logging

The logging service provides a consistent mechanism for monitoring all aspects of system activity which developers and producers consider to be important (see discussion in Section 7.7, Testing, Monitoring and Analytics).

Activities to be logged may include:

- User interactions with all devices in the client environment for the duration of a production session.
- Interactions between components in the production environment (such as video servers, metadata and graphics feeds).
- Interactions between devices in the client environment to discover and launch apps, and to synchronise media objects between devices.

- The request and delivery of media objects and streams.
- The transfer of layout information from the Layout Service to either a cloud compositor or devices in the client environment.
- The authentication of users and client devices whenever this is required by application logic.
- Communication sessions set up between client devices in different locations, mediated by the Lobby.

For the purposes of 2-IMMERSE trials, the scope of a consistent set of logs will be restricted to a **production session**, which refers to the up-time of the prototype 2-IMMERSE platform during an individual trial event, such as a theatre play, MotoGP race or football match. The logging service must be started before all other services and will be the last service to be shut down. It may also be started independently of a production session to enable developers and producers to read and analyse log data.

The logging service will operate on the following principles:

- Logs are created by the majority of other system components, which are each responsible for transmitting their logs to the logging service.
- The logging service acts as a log aggregator to ingest, store and index log data. It will provide ingested log data to the analytics service, which can be used to present and analyse its data.
- All components which create logs should use a Wall Clock service which is synchronised with the Sync Service as their reference for log timestamps to ensure that ordering is correctly preserved by the log service.
- If logs are aggregated from external 'black box' components, they should be annotated appropriately if synchronised clocks cannot be guaranteed.
- Log transactions should not noticeably impact the performance of the originating component and should be executed as soon as possible after event being logged.
- For simplicity and reliability, logs will be made available to the server during every production session using one of two methods:
    - **Instantaneous transmission:** Critical log data is transmitted to the log server at the time the logged event takes place. This should be restricted to essential data and meeting real-time system monitoring requirements - such as reporting errors, understanding system load and key success criteria during the experience (e.g. have all the users logged in?).
    - **Store and forward:** Non-critical log data will be stored locally by the component or service creating the log. This data must then be uploaded to the log server in a one-time transaction before the component or service terminates. Non-volatile local storage will be used where available to reduce the risk of the stored log data being lost if the component or service running on it crashes.
- Given the relatively short length of a production session, these two options provide a pragmatic approach which should remove the complexity of joining incremental batches of log events and also the risk of losing critical log data in the event of an individual component or service failure.

6.1.10.1 Selected Technologies

There are a wide variety of log analysis tools available, but the following three options seem to be the most appropriate candidates for the 2-IMMERSE logging service:

- **Logstash and Elasticsearch**, both components from the Elastic Stack (https://www.elastic.co/products). Logstash provides a flexible, open source data collection pipeline, while Elasticsearch provides storage, plus indexing and analytics functions.
- **Splunk Light** (http://www.splunk.com/en_us/products/splunk-light.html) provides log search and analysis for small IT environments and is available free of charge if it indexes less than 20GB of logs per day.
- **A DIY solution:** Given the relatively small volumes of data involved per production session, it may just be sufficient to write a simple log server which aggregates atomic events delivered via HTTP transactions into a database. Log files stored by other components or services could then be manually combined after the event.

6.1.11 Analytics

The Analytics service is likely to be of greatest value as an offline service – i.e. to be used for post-hoc analysis of data collected and aggregated by the logging service during a production session. It is feasible that developers or producers might make limited use of analytics during an event, but in a field trial environment this is likely to consist of simple queries to confirm the correct operation of the system, perhaps presented as a live graphical 'dashboard'.

The following table lists a potential set of usage scenarios for the analytics service:

Usage Scenario	**Live**	**Offline**
Measuring system performance by comparing aggregated logs and calculating latencies and throughputs.	Yes	Yes
Extracting basic statistics about the audience for a particular production session, including details of their home environment and the group dynamics of those participating in the experience.	Yes	Yes
Pattern mining user behaviour sequences, perhaps comparing against stereotypes estimated by authors. This could indicate: how behaviour differed from what was expected which parts of the experience were most and least popular when users were confused or frustrated with an experience		Yes
Examining the impact of system errors and degraded performance on user behaviour.		Yes
Evaluating whether companion devices increased user engagement with the experience, and which combinations were most effective at doing so.		Yes

Table 2 -Usage scenarios for the analytics service

6.1.11.1 Selected Technologies

As described in Section 7.7: Testing, Monitoring and Analytics, there is a massive choice of tools available for data analytics. While the volumes of data generated by a production session will not be very large by modern standards, they can still take advantage of tools designed for analytics at scale. The following are likely candidates, and in principle any or all of these could be used for offline analytics given that there are no dependencies with the rest of the system beyond access to data stored by the log server.

1. **Kibana from the Elastic Stack** (https://www.elastic.co/products) is an open source analytics and visualisation platform designed to connect directly to Elasticsearch.
2. **R Studio and Shiny** (https://www.rstudio.com). R Studio is a popular open source development environment for the powerful statistical language, R. Shiny is a web framework for building interactive visualisations (such as dashboards) using R.
3. **Tableau** (http://www.tableau.com/products/desktop) is a paid-for visual analytics tool which supports complex visual analytics on local or remote data sources.
4. **RapidMiner** (https://rapidminer.com/) is an open source and paid-for predictive analytics platform which offers a graphical plug-and-play approach to the implementation of machine learning techniques.

6.1.12 Origin Server / CDN

In our architecture we assume the availability of a CDN to efficiently serve media objects and DMApp Components to client devices. The CDN shall contain an origin server operating as the source of truth for all content and shall be capable of serving all the content available on the CDN. As the geographical spread of consuming clients increases in distance from the origin server, the CDN should utilise edge servers to minimise the physical distance required to deliver content to clients. Standard CDNs use the DNS resolver's IP address to perform a geographic lookup to select a delivery server (origin or edge) closest to the client. An edge server selection policy based upon closest geography positioning offers the best delivery performance.

It is assumed that delivery performance takes priority within the server selection algorithm for 2-IMMERSE, combining both geographical distance and load balancing factors to maximise performance. This is preferred to other policies which may factor the value of specific content and distribute availability across the CDN based upon relative hosting costs.

The complexity of the CDN can evolve with the size of the 2-IMMERSE client population size and geographic spread. In the early PoC phases, it may well be sufficient to minimise costs and host a CDN containing just the origin server. Although, it is assumed that contributing partners to the 2-IMMERSE project already operate CDNs to deliver content to connected devices (i.e. OTT video delivery to companion devices). We should be considering leveraging these based upon availability and costs.

Interfaces

The CDN should be addressed through a domain name used in specific content URLs resolvable through the CDN web-service APIs accessed by the client. The selection of the delivery server (origin or edge) within the CDN is performed as part of the DNS resolution process.

The origin server will host an API to manage content hosted as the source of truth within the CDN.

The edge server instances will host an API for configuration of the content caching policies adopted as part of its role within the CDN.

It is assumed that these management APIs will be hosted by a web service.

6.1.13 TV Platform

In our architecture we assume the availability of a TV platform that can support linear and on-demand AV distribution. Linear distribution could include DVB Terrestrial / Satellite, IP Multicast, or OTT live streams (e.g. HLS or DASH). For on-demand distribution we assume OTT (e.g. HLS or DASH).

The TV platform is expected to provide the following services as required to support AV distribution of the primary content for consumption on client devices (STB, Smart TV, Companion Devices):

- Control plane (catalogue, identity, offer, security, recommendations, messaging, etc.)
- Data plane (origination, acquisition, distribution)

Data Plane

The acquisition, origination and distribution of the AV assets played out as part of the TV platform’s linear or VOD services. Origin AV assets are ingested, stored, encoded, transcoded, packaged and distributed within a media preparation process required by TV platform’s delivery networks (i.e. Satellite, Terrestrial, Cable, OTT IP ABR).

The consuming clients have been integrated to interface with the AV delivery network, this integration ranges from middleware stacks operating in dedicated STBs through to applications running on agnostic connected devices such as SmartTVs and companion devices.

Control Plane

The components providing the user experience services required to consume the TV platform’s linear or VOD services. The following shortlists relevant components for a typical Pay-TV platform:

- User Management: Operated within Pay-TV platforms, used to manage subscription accounts, profiles and identity. 2-IMMERSE elements (such as the Identity Management and Authentication Service) may need to interface with an existing user management service to facilitate unified identity and single sign-in experience for users.
- Security: Operated within Pay-TV platforms, used to control access to channels/content based upon subscription entitlements. Interfacing with the TV platform security service maybe required when providing 2-IMMERSE experiences which use protected media.
- Catalogue Management: Ingesting, storing, indexing and distributing content metadata, used in client UIs to present EPGs. Interfacing with the Catalogue management service maybe required to map 2-IMMERSE media objects with the TV platform catalogue entities.

Interfaces to the TV Platform control plane are expected to be proprietary and require a custom integration layer.

Selected Technologies

For the purposes of our trial platforms, the project is likely to adopt a simpler TV platform from both a data and control plane perspective, since we will be developing our own emulated HbbTV device / stack. For the data plane we are likely to restrict linear distribution to OTT live or on-demand streams in order to simplify the emulated HbbTV device / stack.

7 Production Architecture

A detailed, generalised production architecture is difficult to create, since we are looking to extend existing production workflows and these existing workflows for the different trial scenarios are already quite different. There is however commonality in what needs to be created through the production process and delivered to the platform.

In this section we will describe in general terms the content and data flows from the production process to the platform. We will then describe the existing production workflows for each of the scenarios, and note what additional content and data flows will be needed to support the scenario as defined in D4.1 (noting that the various scenarios are described with varying levels of detail, reflecting the scenario maturity). Finally we cover the topic of Testing, Monitoring and Analytics.

There are likely to be a number of ‘production services’ required to deliver the additional content and data flows e.g. moderation / conformance, info-graphics generation, push notifications (triggers) etc., however at this stage it is difficult to define these and we will define them on trial-specific basis.

7.1 Production Content and Data Flows

7.1.1 Timeline Description

The timeline description describes the changing availability of media objects and DMApp Components that comprise an experience over the duration of that experience.

The format / syntax of this data is to be specified, but will likely be defined as JSON or XML.

7.1.2 Layout Requirements

The layout requirements specify a priority for each media object/DMApp component, and, for each media object/DMApp component layout constraints such as:

- min/max size
- audio capability
- interaction support
- and, whether the user can over-ride these constraints.

Some of these constraints may be expressed relative to other components (priority, position etc.).

The format / syntax of this data is to be specified, but will likely be defined as JSON or XML.

7.1.3 Media Objects

The set of media streams or assets that comprise an experience. These includes typical programme elements (main audio/video, audio description, subtitles etc.), but would also include the constituent elements that are composited / and mixed to create these programme elements, such as clean feeds, audio commentary, auxiliary camera feeds, metadata feeds, images, graphics, A/V clips (e.g. highlights / replays).

7.1.4 DMApp Components

As described in section 5.2.1: Web Applications and Reusable Components, DMApp components are reusable components written using HTML5, CSS3 and JavaScript.

7.1.5 Native Companion Applications

As described in section 5.2.2: Host Application, these are native host applications for the common 2-IMMERSE run time web application and DMApp components, which will likely be developed using a wrapper technology such as Cordova or Titanium.

7.1.6 HbbTV Applications

As described in section 5.2.1: Web Applications and Reusable Components, HbbTV Applications are web applications written using HTML5, CSS3 and JavaScript.

7.1.7 Live Trigger Events

For live productions, live trigger events will be created to trigger presentation changes in the immersive experience (these triggers could change the set of DMApp components being presented, or could trigger a change of presentation within a DMApp component). These trigger events are received by the Timeline Service.

7.2 Prototype Service 1 – Watching Theatre at home

7.2.1 Existing Workflow

Six Sony 4K HDC 4300 cameras, which are located in the auditorium and are controlled by human operators, provide continuous live image feeds along rigged cables to a nearby OB4K unit operating with a XAVC Intra codec. Two of the cameras are on fixed pedestals and three others are mounted on dollies which run on short tracks in areas within the auditorium from which seats have been removed.

The framings, lens adjustments and camera moves for each of these cameras have been scripted in advance by the screen director and rehearsed during two earlier camera rehearsals. Both the live mix of these rehearsals and the outputs of each camera have been recorded, composited into a single feed with a main image of the mix and the individual feeds arrayed around this in a “L” configuration. This composite is studied by the screen director and the camera and sound teams to refine the camera script before the live performance.

During the performance the operators receive additional instructions and communicate with the screen director and script supervisor who, with the vision mixer, are monitoring the live feeds and selecting moment by moment, and broadly in accordance with the camera script (but with occasional improvised variations), the shots in the master ‘mixed' feed. As the image feeds are coming into the scanner they are being monitored by engineers and balanced for to ensure consistency of colour and intensity from shot to shot.

Two additional Sony 4K HDC 4300 cameras are dedicated to providing shots of the host and interviewees before the performance begins and during the interval.

In parallel with the image feeds is the live mixing of a 5.1 surround audio track created, again in accordance with a previously created script that has been tested during two rehearsals, from up to 156 separate feeds from inside the theatre. These feeds come from individual radio mics mounted in the wigs or costumes of each of the actors, from mics placed in the band room where the music is played live by up to 8 musicians, from mics hung in the auditorium to pick up audience response, and also from pre-recorded special effects and music. The synchronous 5.1 audio mix is fed to the scanner where it is married with the image track before output to the satellite uplink.

Live English subtitles are created, using pre-set files, are also inserted into the signal on a separate channel so that the cinemas taking it via a downlink can opt to include these in the projection or not. Only a small number of European cinemas take advantage of this option.

Three Sony PWS 4500 media streamers, with a total of 8TB internal storage, are used both to play out short pre-recorded elements that are used as context and marketing before the performance and during the interval. These machines also record the master feed and audio, as well as continuous feeds from three of the performances cameras. These 'isolated' feeds are used to provide alternative shots and editing coverage during the small adjustments made between the live presentation and the release of a final recording for DCP mastering, DVD production and potential broadcast and other exploitation.

Throughout this process there is a system of private audio channels in operation connecting, variously, the team in the scanner, the sound crew (including two radio mic operators in the auditorium), the camera operators and the stage management team who are running the show, including giving audio cues for lighting, sound and automation changes.

The married image mix and 5.1 audio is delayed by 65 seconds in an EVS machine (this is so that the production can be classified as a "film" for purposes of UK tax relief funding; there is no intention of intervening in the feed in any way) and then output to a satellite uplink truck which transmits the signal to three channels across two satellites: Intelsat 10-02 (used for download by most UK cinemas, Picturehouse and Cineworld cinemas, and used by most of the European cinemas who take a live feed), and Intelsat 905 (used by most of the multiplex theatres in the UK as well as some UK independents). The cinemas take the feeds by their local downlinks, using dishes mounted on their roofs, and use their in-theatre digital projection systems to present the feed on their screens.

After the live performance, on subsequent days, some 'tidying-up' work is done to both image and sound, including some additional grading, before a DCP (digital cinema package) master file is created, from which DCPs are made for courier distribution to cinemas abroad where the production will be shown "as live" on dates up to two months later. This DCP master file is also distributed via an Aspera link so that duplication facilities in some countries can create DCPs locally.

This "tidied-up" file is also the one used for DVD and Blu-ray mastering and for further down-stream exploitation, as well as for archiving by the theatre company. Some of the rehearsal material and certain of the "iso" feeds are also archived.

7.2.2 Additional Content and Data Flows

The additional content and data flows identified through the user stories in Annex A are as follows:

- Relevant text, image, audio and video resources about the play, the production, the cast and crew (Introductory, Informed or Expert levels)
- Synchronised information and commentary in the form of image and text (Introductory, Informed or Expert levels)
- Static wide shot of the stage
- Live and interactive 360-degree video and audio from the foyer (TBD)

7.3 Prototype Service 2 – Theatre in School

7.3.1 Existing Workflow

This will be the same production workflow that has been developed to support Prototype Service 1 – Watching Theatre at Home.

7.3.2 Additional Content and Data Flows

This is still to be determined but is likely to be additional elements to support teaching and the educational context of this Prototype Service. It may also require the presentation of the content in an episodic fashion; since it is unlikely the entire play will be consumed in a single session due to school timetabling (amongst other constraints).

7.4 Prototype Service 3 – MotoGP at home

7.4.1 Existing Workflow

The trial will use the available video and data feeds provided through the existing MotoGP rights deal agreed with BT Sport. These rights include access to 9 live video feeds; clips and highlights of qualifying session and races; real time data (timing, track positions and circuit maps and rider standings); and editorial content, such as news stories, picture galleries and social interaction with the riders, teams BT Sport presentation and commentary teams.

Content type	Content description
Video	Live video streams of each MotoGP, Moto2 and Moto3 qualifying session and race, including: • Live BT Sport programming • Feed 1: On Board Camera 1 • Feed 2: On Board Camera 2 • Feed 3: On Board Camera 3 • Feed 4: On Board Camera 4 • Feed 5: Helicopter Feed (race day only) • Feed 6: Live Timing • Feed 7: Live Tracking • Feed 8: Highlights/Clips
On demand replays (geo-blocked to UK)	Provision of full replays of MotoGP, Moto2 and Moto3 qualifying sessions and races.
Clips (geo-blocked to the UK)	Access to clips and highlights of each MotoGP, Moto2 and Moto3 qualifying session and races.
Real-time Data	• Timing • Tracking visualized on circuit maps. • Rider standings.
Editorial	• News stories. • Pictures galleries. • Social interaction with riders, teams, BT Sport presentation and commentary team.

7.4.2 Additional Content and Data Flows

Production requirements for the 2017 trial will be agreed with Dorna, BT Sport and North One Production during the course of this year. We will further develop relationships with these stakeholders at the Silverstone MotoGP race in early September 2016. This will also enable the project team to further develop our understanding of current production workflows.

A range of options being considered for the MotoGP trial are outlined in D4.1.

7.5 Prototype Service 4 – Watching Football in a Pub

7.5.1 Existing Workflow

The trial will use the available video and data feeds provided through the existing FA Cup rights deal agreed with BT Sport. These rights include live video coverage of matches, pre-match, in-play and post-match data feeds (including match events and team and player performance data); and editorial content, such as news stories, picture galleries and social interaction with the managers, players and broadcast team.

Content type	Content description
Video	Live 'dirty' video feed via DTT delivery on SKY Platform to BT Sport Business subscribers • Live BT Sport programming • Additional L Bar overlay option
Clips	Access to in-game selected* highlight clips via: • BBC & BT Sport Websites • BBC & BT Sport Mobile Apps. • BBC & BT Sport social media accounts. *Exact media rights restrictions TBC via BT Sport.
Real-time Data	BT Sport & BBC* receive the following football xml feeds from Opta to power a range of live production and digital services: • F1 – Fixtures & Results (non-live) • F2 – Match Preview (form, head to head etc.) • F3 – Standings (live league tables) • F7 – key match events/stats (goals, sub, cards) • F9- Live detailed match stats (full player and team stats) • F13 – Automated text commentary • F24 –Live detailed positional stats (used for touchmaps, average formation graphics etc.) • F25 – EVS compatible log feeds • F26 – Live scores • F30 – Cumulative season statistics (team and player) • F40 – Squads • F50 – Live event alerts • Penalty Data – penalty history by player for BT Sport's live Premier League matches (xls • format). *Based on BT Sport Opta Agreement 2015/16, no visibility of BBC Opta agreement.

Content type	Content description
Editorial	• News stories. • Pictures galleries. • Social interaction with players, managers, BT Sport presentation and commentary team.

7.5.2 Additional Content and Data Flows

Production requirements for the 2018 trial will be agreed with BT Sport production, BT Sport Business and additional stakeholders during 2017. We are currently developing relationships with these stakeholders on both production (FA Cup Final 2016) and commercial (Pub of the Future) drivers for developing a collaborative customer showcase environment at the BT Production Hub. This will also enable the project team to further develop our understanding of current stakeholder production workflows and commercial drivers.

A range of options being considered for the Football in Pubs trial are outlined in D4.1

7.6 Media formats used by 2-IMMERSE Production Associates

The following section provides the specifications of the media formats used by 2-IMMERSE production associates, and hence indicates the formats in which content is likely to be received for use in 2-IMMERSE trials and demonstrations.

Disclaimer: The information below may be incomplete and is subject to change. Details should be checked before implementation of each trial or demo.

7.6.1 BT Sport HD Production[6]

	File-based	**Live**
Picture format	1920 x 1080 50 interlaced fields per second	1920 x 1080 (16:9 aspect ratio) 25 frames per second, delivered as 50 interlaced fields per second.
Container/Interface	MXF OP1a file conforming to AMWA specification AS-11 v1.1. The AS-11 file must use the ‘UK DPP shim specifications’ that describe exactly how the file must be constructed to meet DPP requirements.	*Preferred:* 1.485 Gb/s HD-SDI connection, SMPTE 292M SD-SDI alternatives exist using compression.

[6] Reference: DPP (Digital Production Partnership) Technical Delivery Standards for BT Sport (BT Sport 4.0, 15/10/2013) http://dpp-assets.s3.amazonaws.com/wp-content/uploads/specs/bt/TechnicalDeliveryStandardsBTSport.pdf

	File-based	Live
Video Compression	AVC Intra codec - 100Mbps High 4:2:2 Intra Profile, Level 4.1	*Preferred:* None. *Alternatives include:* JPEG2000 - 140 Mbps H.264/MPEG-4 Part 10 AVC, Long GOP 4:2:2 – 45 Mbps H.262/MPEG-2 Part 2, Long GOP, 4:2:2 – 60 Mbps
Audio Compression	PCM 48kHz/24bit sampled Uncompressed 4 or 16 tracks	48kHz/24bit sampled Stereo mixed programme on tracks 1&2 (uncompressed) Stereo international sound on tracks 3&4 (uncompressed) Surround mix Dolby E encoded on tracks 5&6 Surround international Dolby E encoded on tracks 7&8

Table 3 -BT Sport HD Delivery Standards

7.6.2 BT Sport UHD Production[7]

	File-based	Live
Picture format	3840 x 2160 50 frames per second progressive	To be confirmed
Container/Interface	MXF OP1a file conforming to AMWA specification AS-11 X1.	To be confirmed
Video Compression	Sony XAVC 4K profile (Long GOP) – 300Mbps 4:2:0/8bit colour	To be confirmed
Audio Compression	AES3 48kHz/24bit sampled Uncompressed Must include both stereo and 5.1 sound tracks.	To be confirmed

Table 4 - BT Sport UHD Delivery Standards

[7] Reference: DPP (Digital Production Partnership) Technical Delivery Standards Supplement for Delivery of UHD Programmes (UK 1.0, 26/1/2016)

http://dpp-assets.s3.amazonaws.com/wp-content/uploads/2016/01/TechnicalDeliverySupplementUHDDPP.pdf

7.6.3 BT Sport Delivery

Linear broadcast content is delivered to the Set-Top Box via IP multicast. The content is encrypted and requires hardware support to decrypt, and hence 2-IMMERSE will not be able to make use of these streams.

Linear broadcast content is delivered to the BT Sport App via HTTP adaptive streaming using both the HLS and Smooth Streaming formats. These are also encrypted, but decryption does not require hardware support and hence 2-IMMERSE clients would be able to consume them.

	BT Sport App (iOS, Android and Web)
Picture formats	480 x 270p 200.0 Kbit/s 480 x 270p 400.0 Kbit/s 640 x 360p 600.0 Kbit/s 640 x 360p 1.2 Mbit/s 960 x 540p 1.8 Mbit/s 960 x 540p 2.4 Mbit/s 1280 x 720p 3.5 Mbit/s 25 frames per second
Container	HLS (MPEG-2 TS) and Smooth Streaming (fragmented MP4)
Video Compression	H.264/MPEG-4 Part 10 AVC (Constrained baseline, main or high profile, depending on rate).
Audio Compression	AAC (HE-AAC) 48kHz sampled 2-channel (stereo) 94 kb/s

Table 5 - BT Sport Adaptive Streaming Delivery

7.6.4 Royal Shakespeare Company Delivery

	File-based
Picture format	3840 x 2160 50 frames per second progressive
Container/Interface	MXF OP-1a
Video Compression	Sony XAVC
Audio Compression	Uncompressed 5.1

Table 6 - RSC Delivery Formats

7.7 Testing, Monitoring and Analytics

7.7.1 Overview

As for Distributed Media Applications, this is a broad area and one that represents new ground in the context of object-based media production for multi-screen experiences. We acknowledge that testing, monitoring and analytics are already well-known disciplines in the context of traditional TV programme production and software application development, for example, and an overview of these is given in Testing and Validation. However, for multi-screen experiences new challenges exist:

- Testing: A key objective of 2-IMMERSE is to develop experiences whose presentation is flexible across single or multiple screens in response to the devices available and the preferences of audience members. At the authoring stage, creative professionals will need to test how their production adapts to different audience scenarios. By sampling a variety of scenarios, they will need to gain confidence that the experience will deliver a consistent quality even though they will never be able to test every permutation themselves. Equally important will be to test the transition between different scenarios and the balance between automated changes and user interactions. Furthermore, scalability testing could attempt to predict, using typical audience profiles, resource demands (such as network bandwidth and client/server processing) over time during an experience.

  This definition of testing is not synonymous with carrying out trials with real audiences: it is more about providing professionals with the tools they need to iterate their designs, although this could potentially extend to user focus groups.
- Monitoring: IPTV service monitoring today is largely focused upon atomic events at the client side, such as accessing a linear channel or playing a piece of VOD content, and metrics aggregated from across the delivery chain, such as the number of concurrent streams through a network pipe, or the number of errors of different types recorded over a period of time. However, we believe that creative professionals may additionally need to monitor the precise way in which audiences are consuming their experiences. This monitoring could be driven by event logs transmitted live by distributed media applications that allow audience members' journeys through an experience to be recorded. These logs would complement existing data relating to server loading or content delivery errors, for example, and over time could be used to anticipate behaviour and hence make more effective use of end-to-end resources.
- Analytics: The growth of Data Science as a discipline and a burgeoning range of tools to support data processing and analytics means that many content service providers are increasing the amount of analysis they perform on data captured by their platforms. 2-IMMERSE multi-screen experiences will enable highly-granular data to be captured, including how the audience is composed, how devices are used in the home environment and how audience attention changes during an event. By monitoring a fully-integrated multi-screen environment for the first time, this data is likely to provide a much fuller picture than that available from most existing platforms, and as such there is great potential for analytics tools to derive new insights after the event. Analysis may also be required to support live monitoring to help the delivery of an experience.

The following sections provide a non-exhaustive list of existing technologies and technical methods which may contribute to meeting these new challenges.

7.7.2 Testing

- Simulation tools and test harnesses
- Unit testing and test automation (e.g. various JavaScript unit testing frameworks)
- Tools and techniques used by game developers

7.7.3 Monitoring

- JavaScript logging frameworks (e.g. log4js and others)
- Enterprise-grade IPTV DoE monitoring platforms such as Conviva
- QoS/QoE monitoring for real-time communications (e.g. single-ended using RTCP, full reference/no reference models)
- Monitoring using software/hardware probes to measure a customer or network environment.
- Event/messaging architectures (again!).

7.7.4 Analytics

- Relational DBMS platforms (from MySQL to Oracle) and associated tools – may be necessary for integration with legacy components in the broadcast production workflow.
- Enterprise-grade log ingestion and real-time dashboard tools such as Splunk.
- Big Data tools as a scalable and flexible approach for ingesting and analysing semi-structured data (Hadoop, Pig, Hive, Spark, R to name but a few + commercial tools such as Tableau and RapidMiner).
- Machine learning methods including clustering, decision trees and time series analysis to support audience analysis and behaviour prediction.

8 Requirements – Architecture Mapping

This section provides a high-level mapping between the requirements that have been defined in section 2.2 and the architecture as currently defined.

Requirement	Architecture component
1. **Association of multiple connected devices** (**clients**) in a home/school/pub environment, with detection of device features (discovery and launch).	The association of multiple devices is managed by the layout service through a context. Devices will declare their capabilities to the Layout Service on joining a context. Devices discover each other using the Device Discovery Service (using HbbTV 2.0 Discovery, Launch and App2App Communication protocols.
2. **Delivery, decoding and rendering of multiple media streams** on any client in the environment.	We will aim to adopt codecs and formats that are supported across all our client devices. The Layout Service will model device capabilities and take these into account when determining placement of media streams onto these devices.
3. **Composition of media** in arbitrary and dynamic layouts/presentations.	The Timeline Service and Layout Service will deliver this capability; giving temporal and spatial control of media composition respectively
4. **Synchronised presentation of media** between multiple clients in one or more environments.	The Timeline Synchronisation Service will deliver this capability for intra and inter home scenarios. This will be based on DVB-CSS
5. **Lobby/chat room** to allow clients to meet during an experience.	The Session (Lobby) Service will deliver this capability
6. **Management of user identities** to register and authorise access to experiences and enable presence information to be shared.	The Authentication service will deliver this capability although we have get to select a particular technology / implementation for this service
7. **Real-time audio and video communication** between multiple home environments.	The Call service will deliver this capability. We are adopting WebRTC for this service.
8. **Tools to enable production personnel to have live control** over aspects of composition in home/school/pub environments.	As noted in section 7, a detailed, generalised production architecture has not yet been defined, but clearly enabling these tools is a key requirement of this architecture
9. **User interface on one or more clients in the environments** to interact with and control aspects of the experience, which responds to the devices available in the environment.	These control and interaction elements of the client user interface will be implemented as DMApp Components which will be able to interact with the Layout & Timeline Services to

Requirement	Architecture component
	control aspects of the experience.
10. **Tools and/or data formats to author a multi-screen experience** in terms of layouts, events and interactions.	The data formats to describe the elements of an experience and the rules for how they can be assembled and interacted with are yet to be defined. The production architecture to enable tools to support creation of these formats is also still to be defined
11. **Each system component (especially each client) logs key aspects of its behaviour** and these logs are aggregated.	The Logging Service will deliver this capability. The Analytics Service will enable insight into system component behaviour and interactions, and at a higher level how users are using the platform.
12. **Offline analysis** of client behaviour logs after the event.	The Analytics Service will deliver this capability.
13. **Monitoring of key aspects of the system during operation** with option to aggregate and feed back into the experience.	For user interactions and viewing behaviour to be presented within the experience in near real-time, a specific service may be required. This service would aggregate that data from participating clients (and the DMApp components running on them), and make it available for presentation.

9 Technology Overview

This section of the document gives a brief overview of a number of candidate technologies that have been identified as potentially relevant to the 2-IMMERSE project. We have grouped these into a number of categories: system setup, distributed media applications, layout and composition, device and content synchronisation, authentication and security, production, and media and metadata delivery. For each technology, there is a goal, overview, pros and cons (from the perspective of applicability within 2-IMMERSE), and links to further information and implementations.

Table 7 below gives a summary of the technologies covered in this section, and indicates their likely adoption within the project.

Technology	Summary	Will be adopted	Under consideration	Unlikely to be adopted
DIAL	Device Discovery & app launch	●		
HbbTV 2.0 Device Discovery	Device Discovery & app launch	●		
Second-Screen Framework	Device Discovery & app launch for HbbTV 1.x		●	
W3C Presentation API	Multiscreen presentation for web-apps			●
W3C Remote Playback API	Browser media remote control			●
UPnP Multiscreen	Device Discovery & app launch			●
Meteor	JavaScript web/mobile/desktop app platform		●	
Web Components	Re-usable web components		●	
Television Application Layer (TAL)	TV HTML application portability layer		●	
WebRTC	Browser-based Real-Time Communications			
Cascading Style Sheets	Document presentation style sheet language	●		
HTML5	Web mark-up language	●		
SMIL	Synchronized Multimedia Integration Language			●
HbbTV browser profile	HbbTV browser profile	●		
Browser Media Composition	Media object composition client		●	
HbbTV / DVB CSS synchronization	Frame accurate companion app sync	●		
Content Synchronisation Events	Application / content sync		●	

Technology	Summary	Will be adopted	Under consideration	Unlikely to be adopted
Ad Insertion/Ad Replacement	HbbTV 2.0 features to support Ad Insertion/Ad Replacement		●	
Access Control	Access control/authentication for access to data, services and media	●		
Cross Platform Authentication	Securely associate a connected media device with an online user account		●	
Object-based Production Tooling	Protocol to enable production platforms to capture and distribute editorial decisions to clients		●	
TRACAB	Sports venue moving object 3D live position capture platform		●	
Virtual Placement	Track and place virtual graphics into live video		●	
Adaptive Streaming	HTTP video content delivery	●		
HbbTV Media Formats	HbbTV media formats	●		
Dolby AC-4	Multi-channel audio codec and container format, supporting object-based audio		●	
360 degree / Immersive Video	360 degree / immersive Video		●	

Table 7 -Technology Overview Summary

9.1 System Setup

9.1.1 DIAL

9.1.1.1 Goal

DIAL stands for **DI**scovery **A**nd **L**aunch. It is a protocol which enables second-screen devices (smart phones, tablets or similar devices) to find first-screen devices (these are TVs, Blu-ray players, set-top-boxes, or similar devices) on the local network and to launch applications on available first-screen devices.

9.1.1.2 Overview

DIAL defines two protocol components and a client-server architecture. The protocol components are the DIAL Service Discovery and the DIAL REST Service. DIAL Service Discovery allows DIAL clients discovering DIAL servers on the local network. DIAL clients are typically implemented by the application running on the second-screen device. DIAL servers are typically implemented by the first-screen device. DIAL is used by a number of services; in DIAL terms these are called applications. They are registered at the DIAL registry (http://www.dial-multiscreen.org/dial-registry/) and include prominent examples like YouTube or the BBC iPlayer. HbbTV has also registered a DIAL application, for further details please refer to the next section.

The DIAL Service Discovery is borrowed from SSDP (simple service discovery protocol), which is part of the UPnP stack (UPnP Device Architecture 1.1, 15 October 2008, http://upnp.org/sdcps-and-certification/standards/device-architecture-documents/). DIAL Service Discovery supplements SSDP by defining a new search target "`urn:dial-multiscreen-org:service:dial:1`" and adding a new response header to the HTTP request for UPnP device description.

The DIAL REST Service allows DIAL clients requesting the DIAL server device to launch and stop applications. The DIAL REST Service is accessed using HTTP.

9.1.1.3 Pros

- Platform agnostic: DIAL Service Discovery and the DIAL REST Service can be implemented on any OS.
- Open: Accessed to the specification is free and there are no license fees on its use.
- Slim: The specification document is 30 pages long.
- Low implementation effort: Implementations can be shared across different DIAL applications

9.1.1.4 Cons

- No stable browser support: There are no stable APIs defined yet, that enable browser apps to send or trigger UDP packages, which is essential for SSDP. However, a proposal is currently discussed in a W3C community group (Presentation API).

9.1.1.5 More Info

- Webpage: http://www.dial-multiscreen.org
- Specification: http://www.dial-multiscreen.org/dial-protocol-specification/DIAL-2ndScreenProtocol-1.7.2.pdf?attredirects=0&d=1

9.1.2 Device Discovery in HbbTV 2.0

9.1.2.1 Goal

HbbTV 2.0 provides facilities for device discovery and application launch to HbbTV devices (this may be TVs, set-top boxes or similar devices) and companion-screen devices (these are tablets, smartphone and similar devices).

9.1.2.2 Overview

The HbbTV 2.0 specification requires HbbTV devices to provide a DIAL server that allows companion screen devices (mobile connected device, like tablet or smartphones) to launch arbitrary HbbTV applications. For this purpose, HbbTV 2.0 defines a new DIAL application name "HbbTV", which can be accessed via the DIAL REST Service. Moreover, HbbTV 2.0 defines a JavaScript API, which allows HbbTV applications discovering companion-screen devices on the local network and enables HbbTV applications to launch an application on the companion screen. The protocol to the companion device is not specified and relies on companion screen applications provided by the TV manufacturers.

9.1.2.3 Pros

- Open specification: Stable specification available and recognised by all major TV and STB vendors.
- Two-way application launch.
- Prototype terminals available: There is a high interest by the manufacturers to implement the CS features.
- Launch of browser-based or native apps: HbbTV terminals can request the launch of web pages in the companion screens Web browser. They can also request the launch of native applications (e.g. Android or iOS apps) on the companion screen.
- Request installation of native apps: HbbTV terminals can request the installation of apps that are not yet installed on the companion-screen device.
- Development tools: There are Libraries available that ease the development of HbbTV 2.0 companion screen experiences, e.g.:
  - HbbCast: https://gitlab-ext.irt.de/christophziegler/HbbtvDialPlayer/tree/master
  - node-hbbtv: https://github.com/fraunhoferfokus/node-hbbtv
- Additional APIs for:
  - App-to-app communication: HbbTV 2.0 specification requests terminals to implement a WebSocket server, which can be used by applications running the HbbTV terminal and the companion screen for exchanging arbitrary messages. This can be used for example to send control commands from the companion screen to a video player app on the TV.
  - Inter-device synchronisation: HbbTV 2.0 provides facilities for the synchronisation of media content across devices (cf. section 9.4.1).

9.1.2.4 Cons

- Discovery of companion-screen devices is not specified in HbbTV and depends on either manufacturers app using a proprietary protocol, or the availability of a published protocol by the manufacturer.

9.1.2.5 More Info

- Web page of the HbbTV consortium: http://hbbtv.org/
- HbbTV 2.0 specification: http://www.etsi.org/deliver/etsi_ts%5C102700_102799%5C102796%5C01.03.01_60%5Cts_102796v010301p.pdf

9.1.3 Second-Screen Framework

9.1.3.1 Goal

The Second-Screen Framework, developed in the EU FP7 project FIcontent, aims at providing facilities for companion screen (CS) discovery, CS application launch and app-to-app communication to HbbTV 1.x devices.

9.1.3.2 Overview

The Core of the Second-Screen Framework is a Web service. Discovery, or rather pairing of TVs and companion devices, is handled by means of a QR code, which displayed on the TV screen and is than scanned by the companion device. Pairing information is stored by the Web service. This allows reactivating connections for later user sessions. So users need to scan the QR code only once per device pair. Clients (HbbTV or companion app) interact with the framework via a JavaScript API exposed by JavaScript libraries, which they obtain from the Web service.

9.1.3.3 Pros

- App-to-app communication: Applications running on TV and companion device can exchange messages via the frameworks Web service.
- Network agnostic: TV and companion device do not need to be in the same local network.
- Interoperability: The system builds upon open Web standards that are supported by HbbTV 1.x terminals. Apps that make use of this system can target a large number of devices that are already available in people's households. In Germany 16 million households own a HbbTV-ready TV set, that is connected to the internet.
- Proven quality: The system is already used in on-air services of German broadcast network ARD, e.g. VoD portal ARD Mediathek and electronic programme-guide application ARD EPG.
- Flexible for extensions: Software is maintained by project partner IRT. Extensions (e.g. support for native applications on Android and iOS or support for multiple companions being connected to a TV) possible if needed.

9.1.3.4 Cons

- User experience: Requests additional user action for connection setup (scan QR-Code)
- Supported platforms: Currently only web-based companion apps supported
- Multi-user support: Currently a TV can only be connected to one companion device and one companion device can only be connected to one TV device.

9.1.3.5 More Info

- Web page including documentation and example applications at: http://lab.mediafi.org/discover-secondscreenframework-overview.html

- Video that explains the concept and that shows sample applications: https://www.youtube.com/watch?v=k0y58ah0yrI

9.1.4 W3C Presentation API

9.1.4.1 Goal

The W3C Presentation API aims at empowering web applications to make use of presentation displays like projectors or connected TVs. For example, this allows devices with limited screen sizes to display content to a larger audience on a projector in a conference room. The specification considers two use cases: the 1-UA (one user agent) case and the 2-UA case. In the 1-UA case, the presentation is controlled and rendered by the same user agent. One could think of a device that has an additional display connected via HDMI. In the 2-UA case, the control and rendering are handled by two different user agents. An example could be a mobile device used for browsing a catch-up content portal and a Miracast-ready HDMI device plugged into a large display for presenting the content.

9.1.4.2 Overview

The W3C specification defines a JavaScript interface exposing features needed to implement above-mentioned use cases. This includes facilities to gather information on available display devices and to exchange messages between devices.

9.1.4.3 Pros

- App-to-app communication
- App launch
- Implementations available:
    - Crosswalk: Crosswalk is a Web app runtime for mobile devices. The project has been initiated by Intel.
    - FAMIUM: Chromium derivate developed by Fraunhofer FOKUS with Presentation API support.

9.1.4.4 Cons

- Availability: The Presentation API is not yet available in standard Web browsers.
- Interoperability: The API exposes different features to web applications including discovery and app-to-app communication. However, the specification does not define how to implement these features. The API can only make its way, if there are enabling protocols with broad acceptance across devices of different vendors. However, the HbbTV 2.0 companion screen protocols could be one solution.
- Maturity: The specification is a working draft.

9.1.4.5 More Info

- Specification available at: https://www.w3.org/TR/presentation-api/
- Use-case and requirements available at: https://github.com/w3c/presentation-api/blob/gh-pages/uc-req.md
- A tutorial on how to use the Presentation API in Crosswalk applications: https://software.intel.com/en-us/html5/hub/blogs/presentation-api-tutorial
- The Crosswalk project's webpage: https://crosswalk-project.org/
- Webpage of the FAMIUM Presentation API project: https://gitlab.fokus.fraunhofer.de/famium/famium-webscreens/wikis/home

9.1.5 W3C Remote Playback API

9.1.5.1 Goal

The Remote Playback API aims at enabling controlling remote playback of media from a webpage.

9.1.5.2 Overview

The API extends the HTMLMediaElement and equips it with interfaces for querying the availability of remote devices that are able to playback the desired content and with interfaces to control the playback on presenting devices.

9.1.5.3 Pros

- Cast solution: Provides a full feature solution to implement cast scenarios (launch and control video presentation on another device) in Web browsers.

9.1.5.4 Cons

- Availability: The Playback API is neither available in standard Web browsers nor are reference implementations available.
- Interoperability: The API exposes different features to web applications including discovery and app-to-app communication. However, the specification does not define how to implement these features. The API can only make its way, if there are enabling protocols with broad acceptance across devices of different vendors.
- Maturity: The specification is a working draft.
- Limited set of use-cases: Use-cases that require the exchange of control commands other than those controlling the playback of media cannot be realised without additional technology.

9.1.5.5 More Info

- API specification available at: https://w3c.github.io/remote-playback/
- Use-Cases and Requirements at: https://github.com/w3c/remote-playback/blob/gh-pages/use-cases.md

9.1.6 UPnP Multiscreen

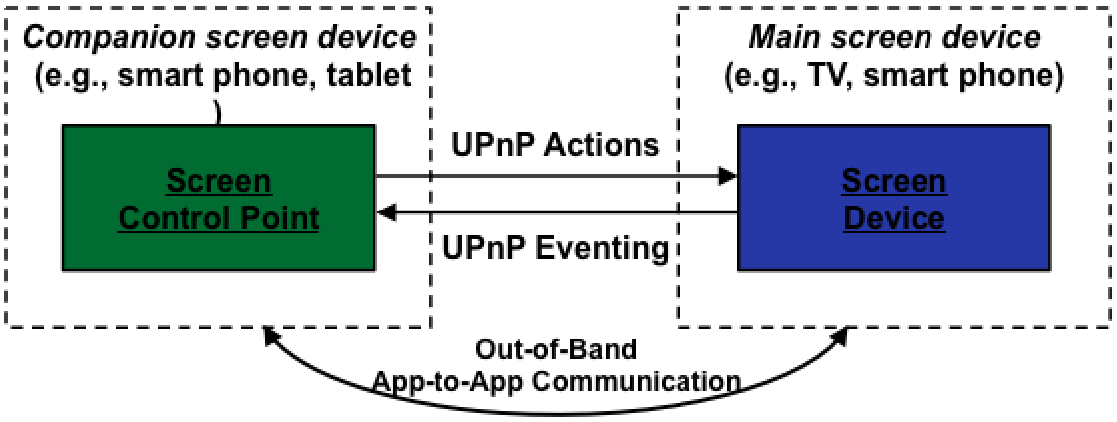
9.1.6.1 Goal

Discovery, Launch and App-to-app Communication for multiscreen applications using an open UPnP standardised approach (as opposed to proprietary approaches, some of which build on parts of UPnP)

9.1.6.2 Overview

The UPnP multi-screen solution enables coordinated interaction between a dynamic set of screen devices, addressing use cases for synchronisation, and screen passing (moving content between devices).

The UPnP multi-screen architecture defines a “screen device” and a “screen control point”. Any device can have either or both of these features. A screen device defines actions and state variables that can be controlled. A screen control point launches those actions and can read and set the state variables. The UPnP device architecture provides device discovery and notification when devices disappear, enabling a dynamic system for interaction between multiple screens.



The UPnP multi-screen solution does not prescribe any particular application design. You can design an application that does all communication via UPnP, or one that just uses UPnP communication to set up the screens and load the application, and then communicates out-of-band.

9.1.6.3 Pros

- Open Standard

9.1.6.4 Cons

- Seems to largely replicate DIAL functionality
- Not clear the adoption is as broad as DIAL

9.1.6.5 More Info

- http://openconnectivity.org/upnp/specifications

9.2 Distributed Media Applications

A technology overview for 'Distributed Media Applications' is challenging because it's not a single technology per se and we are covering brand new ground. Below are some examples of application-layer technologies and architectures that would be useful, some of which could be pushed into the platform layer if considered fundamental enough, and vice versa.

9.2.1 Overview of application-layer technologies

WP2 is concerned with producing API specifications for a platform, however sustainable development of DMApps also requires a high-level *application framework* and a *service architecture* sat on top of the underlying platform architecture. Listed below are some of the things we might need to consider and some candidate technologies. The following research questions to help frame requirements:

1. How can application functionality be decomposed and assigned to separate application components, potentially on different devices?
2. How can application content and functionality be authored for separate application components?
3. How can user interfaces be distributed across many devices to many users?

9.2.1.1 Application Framework

- Distributed User Interfaces (DEMIPLAT, Mediascape, Web-components)
- Cross-platform development tools (Cordova, Appcelerator Titanium, TAL, Unity, Xamarin)
- Application layer network protocols (peer-to-peer and/or client-server)
- Compositing architecture and protocols (in the cloud, on a home server, on a client device)
- Fat-clients and thin-clients (VNC, RemoteDesktop, MediaRecorder + WebRTC)
- Application event/messaging (publish-subscribe)
- Application component frameworks (entity/component/event)
- Reactive frameworks (Meteor – also isomorphic)

9.2.1.2 Service Architecture

- Distributed service-oriented architecture (Microservices, MVC, Multi-tier, Microservice hybrid)
- Service authoring framework (Node.js)
- Event/messaging architecture
- Local caching architecture (proxies, file stores, varnish)
- Storage architecture (state management, local database)
- Scalable cloud-hosted services

There are many other things we could add to this list, but most of the listed items are fundamental to constructing a distributed media experience that is decomposed into a number of apps, running on different user devices.

9.2.1.3 Further Information

- Xamarin:
  - https://xamarin.com/download-it?_bt=101035044668&_bk=xamarin&_bm=e&gclid=CNbkzPfnyMoCFRKNGwodl2oBlQ
- Appcelerator Titanium:
  - http://www.appcelerator.com/mobile-app-development-products/
- Cordova and PhoneGap:
  - https://cordova.apache.org/
  - http://phonegap.com/
- Mediascape multi-device applications:
  - http://mediascapeproject.eu/outcomes.php
- Microservices / Service Oriented Architectures:
  - http://martinfowler.com/articles/microservices.html
  - http://www.infoq.com/news/2014/03/microservices-soa

9.2.2 Meteor

9.2.2.1 Goal

Meteor is an open source application platform for building reactive apps for iOS, Android, and the web, entirely in JavaScript.

9.2.2.2 Overview

Meteor is a series of open source projects (MIT license) that form a JavaScript app platform for 'full stack' development on mobile and web. It is an isomorphic JavaScript framework meaning that it uses the same language on both the client (web browser) and server (via node.js). This reduces the training and skillsets required to create an app end to end.

Meteor is also a reactive framework. Reactive programming updates variables automatically on both client and server sides without extra work by the programmer e.g. updating a client web page without requiring the client to press Refresh. Changes from any client immediately appear on everyone's screen.

It integrates with MongoDB and uses the Distributed Data Protocol and a publish-subscribe pattern to automatically propagate data changes to clients without requiring the developer to write any synchronization code. On the client, Meteor depends on jQuery and can be used with any JavaScript UI widget library.

Meteor's mission is to transition the web from a "dumb terminal" model that is based on serving HTML, to a client/server model that is based on exchanging data.

Meteor can build a native wrapper for web apps and publish them to the Google Play Store or iOS App Store with just a few commands.

9.2.2.3 Pros

- Would provide 2-IMMERSE with a way to rapidly create apps that run in a distributed multi-device environment and share state.
- It's cross platform, which would different 2-IMMERSE devices interact with one another.
- It's full-stack and scopes 2-IMMERSE 's service layer.
- Use a single language (JavaScript) and API client and server side.
- Provides an application deployment mechanism.
- Could possibly be used in conjunction with TAL.

9.2.2.4 Cons

- Meteor ecosystem is still evolving.
- Expects you to use a MongoDB like interface on client and server, although that's not necessarily a bad thing.

9.2.2.5 More Info

- Official site: https://www.meteor.com/
- Meteor on GitHub: https://github.com/meteor/meteor
- Blog post (old now and conclusions are out of date, but useful for understanding the fundamentals): http://blog.jasoncrawford.org/meteor-demystified

9.2.3 Web Components

9.2.3.1 Goal

Express distributed media applications as a collection of reusable components that can be tested in isolation.

9.2.3.2 Overview

Web components are a standards-driven approach to developing re-usable chunks of web pages. They assist with modern web application architecture by ensuring apps are structured into logical modules that can be reused anywhere.

Web components allow you to define custom elements that extend the vocabulary of HTML and deliver sophisticated user interfaces. They were introduced in 2011. All major browsers have started implementation of the technologies needed to run web components natively. While browser vendors are still working on native implementations, a polyfill exists to make web components available to developers already.

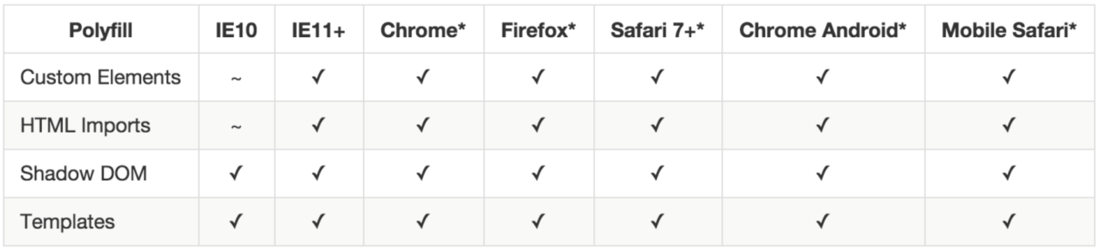
The following table summarises the W3C standards to make up Web Components. (See: http://webcomponents.org/)

9.2.3.3 Suite of W3C standards

Custom Elements	Allow authors to define their own custom tags.
Shadow DOM	Provides encapsulation by hiding DOM sub-trees and CSS styles under shadow roots.
Templates	Reusable fragments of HTML mark-up that remain inert until instantiated.
HTML Imports	A way to include and reuse HTML documents via other HTML documents.

9.2.3.4 Browser Support

Chrome and Opera natively support all web component features, but the webcomponent.js polyfill gives feature parity across all major browsers.



Polyfill	IE10	IE11+	Chrome*	Firefox*	Safari 7+*	Chrome Android*	Mobile Safari*
Custom Elements	~	✓	✓	✓	✓	✓	✓
HTML Imports	~	✓	✓	✓	✓	✓	✓
Shadow DOM	✓	✓	✓	✓	✓	✓	✓
Templates	✓	✓	✓	✓	✓	✓	✓

(Source: https://github.com/webcomponents/webcomponentsjs)

* Indicates the current version of the browser

The polyfill may work in older browsers however they may require additional polyfills (such as classList) to be used.

9.2.3.5 Pros

- Prior to custom elements, there was no standard way to define a component. Every framework, such as Angular, Ember or Backbone, invents its own mechanism. This results in fragmentation: components built using different frameworks do not interoperate with each other. Web components provide a way to define UI elements that can be used in any application, regardless of the framework it's written with. In addition, these custom UI elements work like built-in HTML elements, so they can be used with any HTML framework.
- Custom elements can be used to define completely new HTML elements or to extend the functionality of existing HTML elements.
- The style of custom elements can be encapsulated inside a shadow DOM boundary, preventing external CSS rules affecting their appearance.
- Web components themselves can be implemented in terms of other frameworks such as React, Handlebars and Bootstrap.
- Web Components define subsystems that are simple to test in isolation.
- They have small external interfaces and can be refactored easily.
- Each HTML import typically defines a single element that is testable from the command line with a headless browser.

9.2.3.6 Cons

- Not verified as operable on embedded HbbTV2 profiles
- Can require a very large polyfill
- Polymer makes authoring easier, but introduces extra layers (platform.js, polymer.js)

9.2.3.7 More Info

- W3C Custom Elements specification (http://w3c.github.io/webcomponents/spec/custom/)
- W3C HTML Imports specification (http://w3c.github.io/webcomponents/spec/imports/)
- Template specification (HTML living standard) (https://html.spec.whatwg.org/multipage/scripting.html#the-template-element)
- W3C Shadow DOM specification (http://w3c.github.io/webcomponents/spec/shadow/)

9.2.3.8 Supporting Technologies

Web Component Polyfill	webcomponent.js is a polyfill for all major browsers: https://github.com/WebComponents/webcomponentsjs
Packaging	Vulcanize reduces an HTML file and its dependent

	HTML imports into one file. https://github.com/polymer/vulcanize
Catalogues/libraries of web component	Custom Elements: a catalogue of pre-written custom elements (web components). https://customelements.io/ Polymer Element Catalogue: prebuilt Polymer elements. https://elements.polymer-project.org/
Web component development environment	Polymer: a <declarative> syntax for defining custom elements that replaces the shadow DOM polyfill with a lightweight shim, uses a data-binding system, and significantly reduces code size. https://www.polymer-project.org/1.0/

The Polymer library provides a declarative syntax that makes it simpler to define custom elements. It adds templates, two-way data binding and property observation to help build powerful, reusable elements. An element built with Polymer looks and works just like any other HTML element.

9.2.4 Television Application Layer (TAL)

9.2.4.1 Goal

The goal of TAL is to simplify TV application development whilst increasing the reach of TV applications. It allows you to write an application once, and be confident that it can be deployed to all HTML-based TV devices.

9.2.4.2 Overview

The TV Application Layer (TAL) is an open source library written in JavaScript/HTML/CSS for building Connected TV applications. Today all of the BBC's HTML-based TV applications are built using TAL and UKTV and Arqiva have also adopted it.

TAL works by abstracting differences in device behaviour, which is important when TV software cannot be updated. It provides common APIs for areas that differ the most from manufacturer to manufacturer.

TAL also provides application layer functionality such as widgets, components, focus management, events, navigation, carousels and data binding. TAL handles approximately 300 uniquely different devices including TVs, Blu-ray players, YouView, BT TV, Games Consoles and even Roku (NowTV). It supports HbbTV 1.0/1.5, HTML5 and Samsung Maple compatible devices.

There are hundreds of different devices in the marketplace and they all use slightly different technologies to achieve the same result (despite standardisation efforts).

9.2.4.3 Pros

- Open source.
- Abstracts differences in devices.
- Write once, run on many devices.
- The bulk of your development can be done on a desktop browser that is built on the same origins as the TV browsers.
- TAL interacts with Broadcast APIs, HLS streaming APIs etc.

9.2.4.4 Cons

- Provides a lowest common denominator solution
- Doesn’t yet support the new HBBTVv2 features

9.2.4.5 More info

- Open source project on GitHub
    - http://fmtvp.github.io/tal/getting-started/introducing-tal.html

TAL exists because there are HTML5 standards, but no one is enforcing them. Compare this to DVB where you must conform to their specification in order to get a logo on your box.

Desktop browsers behave differently from each other, however the differences between TV web browsers are much more significant and extensive. TV web browsers are in no way similar to Google Chrome with its large numbers of contributors and active support community. TV web browsers are bug-ridden snap-shots made by small teams with tiny maintenance budgets.

HTML is also evolving, which means that TVs certified a year ago are no longer fully compliant. HTML5 isn't just a single specification, it implements a huge range of specifications and it would be a massive challenge to ensure they all conformed fully.

WebKit is a popular choice for TVs, consoles and set-top boxes. It's split into two principle libraries, WebCore (all the common stuff) and a separate library that implements platform specific rendering, media playback, audio, user input etc. Each TV manufacturer must re-implement the WebKit backend under a tight deadline for its custom hardware. Even if WebCore is fully standards compliant, there are differences in this lower layer (interpretations, bugs, different device capabilities etc.). Ultimately, the TV manufacturer's goal is to launch their product in time for Christmas, so they do the bare minimum to support apps and quickly move onto something else.

9.2.5 WebRTC

9.2.5.1 Goal

WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple JavaScript APIs.

9.2.5.2 Overview

WebRTC enables RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allows them all to communicate via a common set of protocols. The WebRTC initiative is a project supported by Google, Mozilla and Opera, amongst others.

WebRTC offers web application developers the ability to write real-time multimedia applications without requiring plugins, downloads or installs.

This is the technology underpinning Google Hangouts and provides transmission and synchronised presentation of AV and user defined data over peer-to-peer connections. It uses a SIP (Session Initiation Protocol) handshake to instigate peer connections (offer, answer, hang-up).

Support for H.264/MPEG-4 Part 10 AVC and VP8 is mandated by the specification. VP9 is also supported. It uses SCTP (Stream Control Transport Protocol), a message-based protocol with optional flow control and congestion control algorithms that are superior to TCP, giving better utilisation of bandwidth. It can be configured for reliable or unreliable data transmission between peers and most optimisation work has centred on mobile devices and wireless connections.

WebRTC uses ICE/STUN/TURN to traverse NATs and establish connections.

SDP files (Session Description Protocol) are exchanged between peers containing information about what codecs to use, network connection information for the direct peer-to-peer route and security keys to use. The SDP is sent via user defined signalling from one side to the other, but you can use any protocol for this. You can send it via JSON over a WebSocket for example.

9.2.5.3 Pros

- Supported in over 1.5 billion devices worldwide.
- iOS/Android native WebRTC clients available and supported in Chrome, Firefox, Opera and soon Edge. Works on Chromecast too.
- Possible technology for compositing remotely (in conjunction with MediaRecorder API).
- Facilitates screen sharing, peer-to-peer comms, telepresence and distributing media from a central DVB receiver.
- Possible 2-IMMERSE technology for a thin-client (dumb client) approach.
- A WebRTC gateway such as Janus could provide a bridge between DVB and client devices for 2-IMMERSE
- Provides secure comms between devices out of the box which will be useful for 2-IMMERSE use cases in public places
- 2-IMMERSE could use RTC DataChannel for non-AV content such as vehicle telemetry with low-latency.

9.2.5.4 Cons

- Not part of the HBBTVv2 spec
- Not seen applied directly to broadcast TV content, although people have recognised this as a use case.

9.2.5.5 More Info

- Official site: https://webrtc.org/
- Good summary of core components: https://webrtc.org/architecture/
- Spec: https://www.w3.org/TR/webrtc/
- Live demos can be accessed at: https://webrtc.github.io/samples
- MediaRecorder: https://www.w3.org/TR/mediastream-recording/
- WebRTC Gateway (Janus): https://janus.conf.meetecho.com/

9.3 Layout and Composition

9.3.1 Spatial

A wide range of documents formats (content models) support the notion of layout; that is the ability to define the size and position of elements within the document, whether that document represents content for printed media, screens or 3d spaces (e.g. PDF, HTML+CSS, SVG, Collada). Similarly, rendering APIs (typically used by rendering clients for these content models) all have a coordinate space that honours the layout (size and position) of media as it is rendered (e.g. OpenGL, OpenVG etc.)

9.3.1.1 CSS (Cascading Style Sheets)

9.3.1.1.1 Goal

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colours, spacing) to Web documents.

9.3.1.1.2 Overview

The latest version of CSS (CSS3) is modular set of specifications for styling web documents and content. At the core of CSS is a visual formatting model where elements in the DOM tree correspond to boxes, according to a defined box model. The layout of these boxes is determined by a layout engine according to (amongst other things) box dimensions and type, and their positioning scheme, be that the normally-flowed, floated, or absolute. Boxes can be overlapped (using z-indices) and transformed in two and three dimensions. CSS units can be absolute (pixels) or relative to another property (em/rem, percentages et al.)

CSS also includes a number of features to help with responsive layouts; that is screen layouts that can adapt to the capabilities of the presentation device. These include

- Media queries
- Flexbox
- Grid Layout

9.3.1.1.3 Pros

- As a web technology it is generally well supported in browsers, across a wide range of devices.

- A well understood (if sometimes quirky) way of describing 2D box layout (which can be generalised for non-HTML content).

9.3.1.1.4 **Cons**

- Differing levels of maturity and platform/ browser support for particular CSS modules and features.

9.3.1.1.5 **More Info**

- https://www.w3.org/Style/CSS/
- https://www.w3.org/TR/css-2015/ gives a good overview on the current status of various CSS modules

9.3.2 Temporal

9.3.2.1 HTML5

9.3.2.1.1 **Goal**

Standard web components, such as the audio, video and track elements, transparently provide intra-media and inter-media sync, as well as shared media control via the MediaController object.

9.3.2.1.2 **Overview**

When using HTML5, the <video> element allows full-fledged media players to be embedded into web pages. This element takes a media file as an input, specified via its *src* attribute. This element is self-sufficient to provide intra-media and inter-media sync (between audio and video), which means that it internally and transparently handles the aspects related to de-multiplexing, decoding, buffering and temporal presentation of media. Similarly, the <audio> element is used for playing audio files.

Playback control (e.g., play, pause, seeking, volume…) can be dynamically set by users through the controls of the media player (by setting its controls attribute to true). Each media (<audio>, <video>) element can also have a MediaController, which is an object that coordinates a synchronized playback of multiple media elements. By default, a media element has no MediaController. An implicit MediaController can be assigned using the mediagroup content attribute. An explicit MediaController can be assigned directly using the controller attribute. Media elements linked to a MediaController are slaved to it. The playback control (e.g., play, pause, seek…), rate and volume of each of the media elements slaved to a MediaController are shared.

Besides, the <track> element can be used as a child of the media (<audio> and <video>) element. It is used to specify (external) subtitles, caption files or other timed-text data that must be presented together with the audio and video data. The input files (in WebVTT format) include a list of items, called cues, each of which contains a start and end time (relative to the media playback position). Synchronization between audio-video and timed-text data is automatically provided, even when dynamic changes in the playback position are issued.

No <track> element can be defined without the prior definition of an associated parent <video> or <audio> element.

Synchronization across devices can be achieved by monitoring the playback position of the media element on each device (by reading its currentTime attribute), exchanging this information via WebSockets, comparing playback time differences between the involved elements, and finally adjusting either the playback position (by setting the currentTime attribute) or the playback rate (by setting the PlaybackRate attribute) during the required time interval.

9.3.2.1.3 **Pros**

- Cross-platform, device, and browser support.
- In combination with WebRTC it can provide audio-visual chat channels for social TV applications.

9.3.2.1.4 **Cons**

- Limited access and control to low level (i.e., stream-level, buffering, codec-level…) information
- Different behaviour in different (versions of) browsers

9.3.2.1.5 **More info**

- W3C HTML5 API https://www.w3.org/TR/html5/
-

9.3.2.2 HTMLTimingObject

9.3.2.2.1 **Goal**

Recent activities within W3C aim at decoupling dependences between media elements, enabling linear composition, sequencing and synchronization across devices with the definition of a new HTMLTimingObject.

9.3.2.2.2 **Overview**

A new Community Group (CG), called Multi-Device Timing CG (MTCG), aims at providing standard mechanisms for a better support for Web-based linear composition, both in single-device and multi-device scenarios. Linear composition refers to the ability to construct advanced linear media presentations by coordinating the playback of independent timing-sensitive Web components. It is also possible to dynamically load and remove individual components during playback (e.g., as a feature of the storyline, as a reaction to the user input, as a reaction of bandwidth limitations…). Similarly, dynamic linear composition is relevant in multi-device scenarios, as the same or different (but related) media content can be simultaneously presented either on companion devices or on geographically distributed devices (e.g., in Social TV). It would also involve dynamically loading and removing the proper linear components when the involved devices join and leave.

To achieve it, the MTCG proposes the definition of a new HTMLTimingObject as basis for Web-based linear composition. The HTMLTimingObject is essentially an advanced stop-watch, wrapping around the system clock, whose value changes predictably in time (if it is not reset). It also supports any velocity or acceleration, and may jump to any position on the timeline. The value of the HTMLTimingObject may be queried for its value at any time.

In single-device scenarios, the idea is to provide linear composition by replacing the use of the MediaController as the director, but using the HTMLTimingObject instead. Accordingly, each of the involved media elements has to equally interface with the HTMLTimingObject, monitor it, and perform the appropriate reactions whenever it pauses, resumes, jumps or speeds up. The involved media elements may also enforce control over the shared timing object, by requesting it to pause, resume, etc.

In multi-device scenarios, linear composition can be achieved by connecting the involved HTMLTimingObjects on each device to the same shared and external timing service provider (by specifying a valid URL for the source attribute of the HTMLTimingObject). This way, the involved HTMLTimingObjects on each device perform as a local representation of a shared online timing mechanism.

Another objective of the W3C MTCG is to define a new version of the <track> element, removing any dependences with the <video> or <audio> parent element. The intention is that this new standalone version of the <track> element can also independently and directly connect with the HTMLTimingObject, without being a sub-element of the media element anymore. This way, linear composition can be provided by keeping the independency between the involved elements. Indeed, the elements do not communicate directly between themselves (they can even be unaware of the existence of other media elements), but only indirectly through a shared timing object.

9.3.2.2.3 **Pros**

- Cross-platform, device, and browser support
- In combination with WebRTC it can provide audio-visual chat channels for social TV applications

9.3.2.2.4 **Cons**

- Limited access and control to low level (i.e., stream-level, buffering, codec-level…) information
- Different behaviour in different (versions of) browsers
- It is not a standard

9.3.2.2.5 **More info**

- W3C Timing Object Draft: http://webtiming.github.io/timingobject/

9.3.2.3 SMIL

9.3.2.3.1 **Goal**

SMIL (Synchronized Multimedia Integration Language) is an XML language for creating, delivering and playing back multimedia presentations.

9.3.2.3.2 **Overview**

SMIL 3.0 is a very large modular standard (66 modules), with the intention that application areas can pick-and-choose modules to define a language (“Profile”) suitable to their needs. 5 such profiles are defined in the standard, ranging from smilText profile (for only specifying temporal behaviour of things like subtitles) to Language Profile (full-blown multimedia composition in time and space). SVG animation also imports some of the SMIL modules to define its temporal characteristics.

In general, SMIL does not deal with media encoding and transport. Media items are generally referred to by URL, with SMIL specifying how and when the media items should be combined into a presentation.

The modules fall into a number of broad categories:

- Timing and synchronisation
- Spatial layout
- Content selection
- Linking and interactivity
- Visual effects (transitions, animation)
- Dynamic (programmatic) modifications

Of these categories, the first is the most interesting for this overview. The main timing model is a hierarchical tree model: individual media items are grouped to play back in parallel, sequentially or based on user interaction or external events. There is fine-grained control to loop individual items or start and stop them at specific intervals. SMIL also provides control over how synchronisation between group members is done: group members can each have their own timeline or a specific group member can control the timelines of other group members, with the possibility to specify constraints for how much the timelines can slip.

Groups can then be treated as media items again and combined with other groups or media items to build a complete presentation.

Content selection allows selecting media item (or group) playback based on all sorts of environmental and user parameters: preferred language, disabilities, screen size, available bandwidth, etc. Timing and synchronisation will adapt to selected content.

9.3.2.3.3 **Pros**

- The timing and synchronisation model is very complete
- Content selection model is simple and effective
- Modularity allows to pick and match whatever functionality is needed

9.3.2.3.4 **Cons**

- Very few full implementations are available
- Complete timing model can be daunting to implement
- Spatial layout model is rather static

- Interaction with other web technologies (especially JavaScript) can be difficult
- The SMIL Language Profile was designed to be writeable by (professional) humans, similar to HTML. This can be seen as both an advantage and a drawback, the drawback being that some of the “convenience” features lead to semantics that are difficult to understand.

9.3.2.3.5 **More Info**

- https://www.w3.org/TR/SMIL/ is the standard itself.
- http://ambulantplayer.org is a complete open source implementation and has sample documents and links to further information.

9.3.3 Composition

9.3.3.1 HbbTV browser profile

At the time of writing, the HbbTV specification has three published versions, the browser profile is identical for 1.0 and 1.5 and got a major update for version 2.0.

The browser profile of 1.0/1.5 is defined by CEA-2014-A also known as CE-HTML. It is based on XHTML 1.0 which is equivalent with HTML4. The CSS profile is CSS TV 1.0 derived from CSS 2.1. The scripting environment is ECMA script 3 (ECMA-262 Edition 3) with DOM 2 support.

An informal reference guide can be found at

- http://www.oipf.tv/docs/OIPF-T2-R1_DAE_Reference_Guide_v1_0-2010-03-11.pdf

Important other features and constraints of HbbTV 1.X are:

- The browser window is full screen on 16 by 9 panels and has a logical resolution of 1280px by 720px
- HTML pages have to use the HbbTV mime type: application/vnd.hbbtv.xhtml+xml
- HbbTV applications can be broadcast related or broadcast independent, the former are controlled, i.e. signalled, by a broadcast channel. The latter are not. An application lifecycle defines when an application must be started and when it must be stopped. See chapter 6 of the HbbTV spec for further details.
- Access to broadcast resources is restricted to broadcast related applications
- Broadcast-related applications can be delivered via broadcast (DSMCC carousel) or HTTP
- Cookies and XML HTTP requests are limited to the same origin policy. CORS is not supported.
- Cookies are not supported for apps delivered via DSMCC carousel.
- Scripts, images, etc. may be loaded from other origins.

HbbTV 2.0 updates the browser profile. HTML5 as well as CSS3 are supported. A list of CSS modules, APIs, etc. can be found in the Web Standards TV profile which is the normative reference used in HbbTV:

http://www.oipf.tv/docs/OIPF-T1-R2-Specification-Volume5a-Web-Standards-TV-Profile-v2_3-2014-01-24.pdf

Compared to the constraints from HbbTV 1 above, the following changes:

- Standard mime types for HTML pages are supported
- CORS (cross origin resource sharing) is supported with XHR2
- The Web Storage API allows to store local information for applications delivered via DSMCC
- Pages can be standard HTML or XHTML, as both is supported by HTML5

HbbTV currently works on a minor update of the specification that will include support for higher screen resolutions.

9.3.3.2 Media Composition in the browser

9.3.3.2.1 Goal

Provides solution for compositing of object-based broadcasts on client devices.

9.3.3.2.2 Overview

BBC R&D have open-sourced their work on compositing media in the browser. This works looks at sequencing and applying effects to videos and other media sources in real-time within the browser. It also has provision for modifying the effects and sequences of media in real-time to create dynamic and interactive content. This work was used to create a number of demos including an object based weather broadcast that adapts the content to users' preferences.

Further work in this area will look to define a media composition protocol that will provide a standard interface for higher-level composition engines to control a media rendering engine. A composition engine could be something as complex as a variable length story rendering system, or as simple as a static description of the constituent parts of a program (like a traditional edit decision list). The media rendering engine could be a browser based renderer, offline back-end server renderer, or even a screen-less smart-radio device.

In conjunction with this work BBC R&D will also be looking to build a more flexible graph-based media processing and sequencing library for the browser.

9.3.3.2.3 Pros

- Would provide 2-IMMERSE with an object-based media compositing solution.
- Higher level abstraction over complex multimedia processing operations.
- Allows 2-IMMERSE to dynamically modify compositing decisions as required by its use cases.
- Built using concepts from IP Studio

9.3.3.2.4 Cons

- High-level abstractions can limit the flexibility of the final experience.

- Technology currently focused on rendering media in the browser with server-side compositing work just commencing.
- Not all work is publically available yet.
- Requires device capable of at least dual AV decode and WebGL for post effects and camera transitions.

9.3.3.2.5 **More Info**

- HTML5-Video-Compositor blog post: http://www.bbc.co.uk/rd/blog/2015-11-open-source-html5-video-compositor
- HTML5-Video-Compositor: https://github.com/bbc/html5-video-compositor
- Forecaster: http://www.bbc.co.uk/rd/blog/2015-11-forecaster-our-experimental-object-based-weather-forecast
- Variable length Radio: http://www.bbc.co.uk/taster/projects/responsive-radio

9.4 Device and content synchronisation

9.4.1 HbbTV / DVB CSS synchronization

9.4.1.1 Goal

Enable companion applications that are frame-accurate synchronized to a TV show or advert on broadcast and on-demand TV services in a home environment.

9.4.1.2 Overview

The TV acts as a server, the companion as a client. Communication is via a LAN (e.g. home Wi-Fi) using a combination of WebSocket and UDP. DVB CSS defines:

- Content ID scheme for DVB broadcast and IP delivered services.
- How timelines can be carried in DVB broadcasts and derived from DASH streams.
- Network protocols to carry timeline position and content ID where TV is the server (CSS-CII, CSS-WC, CSS-TS)

HbbTV 2.0 profiles DVB CSS for use in connected TVs. It defines functions for enabling/disabling the server side from HbbTV apps (HTML+JS apps that run on the TV).

The following are defined by DVB CSS but are not profiled for use in HbbTV 2.0:

- Protocol for notifying companions of “events” in a broadcast/DASH stream (CSS-TE)
- Network discovery mechanism (CSS-DA) … HbbTV defines its own.

DVB CSS also defines a metadata format and protocol between companion and cloud servers for “resolving” broadcast identifiers and timelines to companion content (CSS-MRS) in a flexible delivery platform independent way. This does not involve the TV.

DVB CSS and HbbTV 2.0 are open ETSI standards created by the DVB Project and HbbTV Association.

9.4.1.3 Pros

- Being implemented in HbbTV 2.0 compliant TVs by major manufacturers.
- Single mechanism supporting both broadcast and on-demand delivered content.
- Frame accurate sync.
- Not dependent on cloud based services or proprietary back-ends.
- Client side can be mostly (but not completely) implemented in a browser.
- Does not rely on detecting audio, so suitable for a noisy pub environment.
- Strong expertise available (original spec authors).
- Some tools, code and other resources available (see end).
- HbbTV 2.0 also brings other useful building blocks:
    - Discovery of the TV on the LAN (DIAL)
    - A TV application environment (HTML+JS)
    - Triggering launching apps on the TV (DIAL)
    - Communication channel between TV apps and companion (“app2app” comms)

9.4.1.4 Cons

- “Real TVs” not yet available. Prototype TVs likely this year (2016). BBC prototypes with limited capabilities and/or immature.
- Currently limited library support for companion applications (BBC and IRT efforts).
- Requires LAN connection (e.g. Wi-Fi) with the TV.
- Pure browser based for clients not possible (requires UDP for one protocol) but Apache Cordova will word (UDP done natively, HTML+JS for everything else).
- TVs not required to support >10 clients (but implementations can choose to).

9.4.1.5 More Info

Specifications:

DVB CSS	ETSI TS 103 286 v1.1.1 parts 1 (overview) and 2 (data model and protocols)
HbbTV 2.0	ETSI TS 102 796 v1.3.1

Open source tools/code/resources:

- Client and server reference implementations (protocols only) https://www.github.com/BBC/pydvbcss
- Tools for calibration to achieve frame accuracy https://www.github.com/BBC/dvbcss-synctiming

- Broadcast timeline generation (GPAC)
  https://gpac.wp.mines-telecom.fr/

Closed source tools/code/resources:

- BBC iOS companion library
  (implements protocols and media player control and HbbTV TV discovery)
- BBC “CSSTV” TV
  (C + gstreamer-based prototype, plays MPEG-2 TS, no TV app environment)
- BBC “CSSTV in browser” prototype
  (python proxy server & JS library in browser, syncs against <video> or <audio>)
- BBC Chrome extension & library for companion apps for Win/Mac/Linux
  (implements protocols and media player control and HbbTV TV discovery in browser)

9.4.2 Content Synchronisation Events

9.4.2.1 Goal

Interactive applications often require synchronisation to video content. For on-demand clips, usually the current play position reported by the platform is sufficient. For broadcast services and live streams, it is a bit more complex, as the play position does not reflect a particular position like a frame in the service or stream. Therefore, platforms like HbbTV define events that can be embedded in broadcast or live streams to enable applications to synchronise video.

9.4.2.2 Overview

DSMCC stream events, to be more precise do-it-now stream events, have been included in HbbTV since version 1.0. Stream events are inserted at the head end as MPEG-2 sections which are not synchronised to the PCR of the broadcast service but by their position in the stream. At the receiver side a stream event is delivered to an application that has registered for this event at the time when it arrives. Use cases are

- notifications in ad breaks which are hints to microsites that provide the user more detailed information on the product,
- application rendered subtitles, that give the user more flexibility how subtitles are rendered, e.g. position, size, font
- timed overlays, e.g. in quiz shows

The stream events are only available in broadcast, but DASH and particularly the DVB DASH profile which is used in HbbTV version 2 adds support for DASH events, which include application layer events that behave very similar to stream events.

HbbTV 2.0 uses the timeline concept from DVB CSS for inter device synchronisation and for multi-stream synchronisation, e.g. video from broadcast with audio from broadband. Applications can make use of those timelines for timed overlays, by retrieving the current value of the timeline. The following timelines are supported:

- PTS: MPEG-2 presentation timestamps

- TEMI: A timeline carried in MPEG-2 TS headers
- DASH PR: The DASH timeline relative to the start of the period.
- MP4 CT: the composition time of MP4 (ISOBMFF) files

For retrieving the current value on these timelines as well as for play positions reported by media objects, HbbTV 2.0 defines a few testable performance parameters as for accuracy, where in the decoding chain the value is measured and minimum granularity (play position).

9.4.2.3 Pros

- HbbTV 2.0 standardizes app sync for live and on demand.
- Deployments for DSMCC stream events available. Stable technology since DVB MHP.

9.4.2.4 Cons

- Synchronisation timelines and DVB DASH events are rather new technology. Deployments unknown.

9.4.2.5 More Info

- Specifications:
  - See chapter 13.11 of ETSI 102 796 1.3.1 (HbbTV 2.0) http://www.etsi.org/deliver/etsi_ts%5C102700_102799%5C102796%5C01.03.01_60%5Cts_102796v010301p.pdf
  - See chapter 7.2.4 of ETSI 102 796 for DSMCC stream events

9.4.3 Ad Insertion/Ad Replacement

9.4.3.1 Goal

As targeted advertising has become standard on webpages, broadcasters are looking for solutions on connected TVs. HbbTV 2.0 includes updates that enable broadcasters to insert or exchange content on an individual basis. While advertisement is the main driver for these features, it could be used for other purposes as well. One example are regional variants of a broadcast channel, which differ only for a daily regional news show or magazine.

9.4.3.2 Overview

HbbTV 2.0 defines two mechanisms to realize ad-insertion. One is based on MPEG-DASH, the second one defines the resource management of multiple media clips in a way, that an application can play a sequence of media clips without larger gaps.

The resource management defined in HbbTV 2.0 for the HTML5 media element allows pre-buffering of a second media clip while another media is playing even if the device only supports one media presenting at any time. In former versions of HbbTV the media object supports queuing of media clips, but did not require pre-buffering. The informal annex J of HbbTV 2.0 describes the basic concept behind this.

MPEG-DASH includes features to dynamically include content from different sources like an advertisement server. In short this is based on dynamic updates of the so-called media presentation description and the use of the W3C XLINK mechanism. First an update introduces a new DASH period, which is a placeholder and does not include any references to media. The xlink mechanisms lets the device resolve the actual content of the DASH period with the ad server.

With ad replacement we refer to the use case of replacing ad breaks of a broadcast channel with ad clips delivered via broadband. One possible option to do this with HbbTV is the combination of stream events or timelines to signal the ad break start, and the optimized resource management of HbbTV 2.0.

9.4.3.3 Pros

- Open specification: Stable specification available and recognised by all major TV and STB vendors.

9.4.3.4 Cons

- Availability of HbbTV 2.0 prototypes
- Though MPEG-DASH is gaining some momentum, features like xlink, multiple periods and DASH events are not widely implemented.

9.4.3.5 More Info

- HbbTV 2.0 specification: http://www.etsi.org/deliver/etsi_ts%5C102700_102799%5C102796%5C01.03.01_60%5Cts_102796v010301p.pdf
- DVB DASH specification: https://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.01.01_60/ts_103285v010101p.pdf

9.5 Authentication and Security

9.5.1 Access Control

9.5.1.1 Goals

1. Provide a means of access control/authentication to allow only authorised users to access or perform actions to personal data, personal devices or otherwise restricted content/data.
2. Enable access on TV and/or second screen devices to protected media and data content where required.

9.5.1.2 Overview

Areas which may need to be authenticated / secured may include:

- Discovery and interaction with the TV device from the second screen device and vice-versa. This falls within the scope of HbbTV/DVB-CSS device discovery and interaction. The security model for discovery in HbbTV/DVB-CSS is that the devices must be visible on the same local area network (LAN), and that the devices may prompt the user for approval where necessary.
- Centralised account and personal data management
  Users may have account(s) on non-local systems which may store personal data. These account(s) could be associated with subscription information where the user has purchased or is otherwise entitled to access to content/services.
  A "user" could be defined as a person, collection of person(s), or household/LAN. An individual device or household/LAN may then be required to support multiple users.

Authentication and security technologies:

TLS (typically in the form of HTTPS) should be used for secure communication between devices and non-local systems.

Typically, users log into accounts using a username and password. Once a user has done this for the first time on a particular device, that device could remember the account credentials for future use.
On TV devices, apps running within a HbbTV/browser environment could use browser-provided local storage APIs, such as cookies and web local storage to store any credentials.
On second screen devices there are a number of mechanisms for secured app-local data storage.

Storing the password itself on the device may be problematic, as if the device is compromised so is the account. A common solution is to use session tokens instead; this is where a token unique to a particular log in session is stored on the device; this is often implemented using HTTP cookies. Either the user can enter the username and password at first log in, or in the case of a TV device paired to a second screen, the first log in could take place on the second screen, and any session tokens could be transmitted to the TV device and optionally stored there.

OAuth is a standardised general framework for authentication using tokens. It includes support for varying access types on a per-token basis and authentication flows such as entering the username and password on a different application or device than the application to be authorised.

9.5.1.3 Content Access

For rights or commercial reasons, some content may be required to be delivered by a protected mechanism and/or require authentication to access. This may be less of an issue for the case of a trial using non-live content, however.

Existing sports media feeds from BT and Dorna which may be useful within the project are protected by a number of mechanisms (see more info).

For as-live demos, live streams from the above sources could be recorded, and then streamed out later as if they were live. As the content is no longer live, using reduced or no content protection may be feasible, (subject to agreement of the content/rights owners, etc.). BT Research & Technology already has some software for recording live HLS protected or unprotected streams and playing them out as-live with no/less protection for demo purposes.

9.5.1.4 Pros

- Storing passwords on devices: This is very easy to implement.
- Using session tokens/cookies as persistent credentials: The case where the log in using the username and password is done from same device as where the session cookie is stored and used is well understood and standardised, and easy to implement.
- Using OAuth: This supports any practical token-based authentication flow.

9.5.1.5 Cons

- Storing passwords on devices: Not very secure, though this may not matter for a prototype/trial.
- Using session tokens/cookies as persistent credentials: Performing the login on the second screen and passing cookies to the TV is somewhat non-standard and may be problematic; using some other form of token may not be an issue though. Passing the password to the TV to do the login there may be insecure, depending on the transport used.
- Using OAuth: This is quite a large and complex specification with many options, such that different implementations are not automatically compatible with each other, which may be overkill for a prototype/trial implementation.
- Storing personal data and account details requires additional safeguards, and requires the implementation of additional components.

9.5.1.6 More Info

- OAuth - IETF RFC6749 – There are a large number of existing implementations

Existing sports media feeds:

- BT: live multicast services (BT Sports).
  These are DRM protected and can only be consumed on production YouView STBs (not dev boxes).
- BT: live services to tablets, Chromecast, etc. (BT Sports).
  These do not require any special hardware to access, only credentials for a valid BT account with BT Sports enabled.
  These are available using HLS and SmoothStreaming (used via Silverlight on PC).
- BT: non-live clips (football and MotoGP) and catch-up BT Sports to tablets, etc.
  Presumably similar to BT’s live services to tablets (to be confirmed).
- Dorna (MotoGP): Additional live video streams from bike onboard cameras, helicam, etc.
  These are not protected and don’t require authentication. HLS format.
- Dorna (MotoGP): Live telemetry and race data.
  Awaiting info from Dorna (to be confirmed).

9.5.2 Cross Platform Authentication

9.5.2.1 Goal

Cross Platform Authentication (CPA) provides a way of securely associating an Internet-connected media device with an online user account, to enable delivery of personalised services to the device.

9.5.2.2 Overview

Cross Platform Authentication (CPA) provides a way of securely associating an internet-connected media device with an online user account, to enable delivery of personalised services to the device, for example, media recommendations, bookmarking, and pause/resume of media playback between devices. In its first version, CPA focuses on hybrid (i.e., broadcast and internet) radios, but could also be applied to connected TVs or set-top boxes.

The protocol is based on OAuth 2.0 but adapted to the characteristics of media devices, and to the needs of broadcasters. The authentication flow is based on the draft OAuth 2.0 device profile.

The goal of CPA is to grant a client permission to make authenticated API calls to a service provider on behalf of an end user. The authorization provider manages client identities, and the link between the client and the end user's identity (which is obtained from an identity provider).

9.5.2.3 Pros

- Makes it easy for end users of connected media devices with limited interaction capabilities to authenticate with online accounts for personalised services.
- Enables single sign-on across all of a broadcasters' services.
- Open source implementations of Authorization provider, Service provider and a range of clients (browser, Android, iOS)

9.5.2.4 Cons

- May be less useful in an environment where devices can discover one another and establish direct communication

9.5.2.5 More Info

- http://www.bbc.co.uk/rd/blog/2014-09-cross-platform-authentication
- https://tech.ebu.ch/cpa
- http://ebu.io/project/cpa (include links to open source projects)

9.6 Production

9.6.1 Object-based Production Tooling

9.6.1.1 Overview

The BBC is developing a general-purpose protocol that allows production platforms to capture and distribute editorial decisions to clients in a standard way. 2-IMMERSE multi-device experiences are object-based in nature and require editorial decisions and clean feeds to be captured and distributed to client devices and intermediary composition services.

9.6.1.1.1 Background

Edit decisions are traditionally expressed using a standardised AAF (Advanced Authoring Format) - an interchange format for migrating data between software suites. AAF can describe video edits (e.g. cross-fades and camera switching) but audio support is basic. The general expectation is that audio is edited in a different package utilising another interchange format called OMF (Open Media Framework).

Apart from handling audio separately, the biggest problem with these formats is that they are not built around live production. If you want to do a live production, you have use a 'growing file' that is appended with new content during the production. This is problematic if you want to capture multiple edit decisions using distributed workflows because the approach is aimed at one editor using one disk drive on a single workstation.

The BBC’s approach to object-based production utilises 'Operational Transformation' (OT), a lock-free, non-blocking technique that stops edit response time from being sensitive to networking latencies. (https://en.wikipedia.org/wiki/Operational_transformation). As a result, OT is suitable for implementing collaboration features such as group editing in the Web/Internet context.

All edit decisions are stored as key-frames that describe a state change on a timeline in a networked database. Issues of conflict and overlap resolution in multi-user workflows that arise from large batches of changes can be avoided by describing each event as a single atomic state change and by applying operational transformations to the insert/delete/undo operations.

9.6.1.1.2 Example

In a traditional AAF file, a simple camera transition from clip#1 to clip#2 is described as follows:

```
Clip#1 (time0 to time2)

Clip#2 (time2 to time4)
```

The operational transform approach describes only the salient events that make this camera transition possible. This description has more in common with animation key-frames. For example:

```
Time0: clip#1

Time2: clip#2

Time3: null
```

In this example, rather than using a time/duration description, which could lead to conflict resolution when the duration of two edits overlap, the simplest description of state at a given time is used to keep changes as small as possible. Operational transforms are then used to transform incoming updates to ensure consistent ordering of these timeline state changes. Updates to the timeline are described as atomic operations, for example:

```
insert(time0, clip#1)
insert(time2, clip#2)
insert(time4, null)
```

9.6.1.2 IP Studio Data Model

IP Studio is a BBC R&D project that aims to use commodity hardware and IP networks to reduce the cost of production. The data model it uses for capturing edits and describing media compositions is derived from the operational transformations approach. It uses domain knowledge to organise state changes into layers to further limit conflicts and allow different craft roles to work together collaboratively. Interoperability with legacy formats will eventually be provided via AAF exporters and importers.

In a production composition, the choice of media source over time is captured in a layer called the “media sequence” timeline. The video processing effects applied to the current media source are described by the “processor sequence” timeline and the connectivity graph of the video processors at any given time is captured as a state block in the “connection sequence” timeline. Finally, a “control sequence” timeline captures the parameter state changes that drive processor sequences, such as the degree of cross-fade. In each case, the data captured is the smallest atomic payload of state information possible that describes the event.

The resulting composition is represented by thousands of JSON blobs representing each state change submitted to the IP Studio composition database. The model is efficient enough to be distributed directly to client devices for client-side compositing and flexible enough to be recomposed on the fly for personalised presentations, both of which are requirements for 2-IMMERSE multi-device experiences.

9.6.1.3 BBC R&D Primer

Primer is a prototype production tool developed by BBC R&D that allows the operator to perform camera cuts, reframe existing cameras and edit shot decisions of live footage. The software also has basic support for book-ending programmes with intros and outros, thus forming a minimum viable product. The pipeline is object-based from end-to-end, capturing all edit decisions and preserving every camera feed for compositing later in the cloud or on audience devices.

Primer is written in HTML and JavaScript and can synchronise playback of multiple live DASH feeds. It provides a master feed that shows the output of the vision mixing process to the operator. It was important to understand how technologies used for distribution and playback of media over the Internet such as HTML5 and DASH could also be used at the authoring stage to streamline workflows and give producers an accurate sense of what the final experience would be. A key challenge for 2-IMMERSE multi-device experiences is retaining a degree of editorial control over resulting compositions. Primer as a platform breaks free from traditional editing by permitting multi-device heuristics to be authored, such as those conforming to a negotiated set of digital rights.

Recently, Primer has been integrated with IP Studio allowing edit decisions to be stored using the IP Studio data model.

Together Primer and IP Studio’s Media Composition Protocol are candidate technologies for helping to author 2-IMMERSE multi-device experiences.

9.6.1.4 Pros

- The IP Studio data model is an industry-supported effort that is rapidly gaining traction.
- The Media Composition Protocol provides a solution for describing composition operations required by 2-IMMERSE.
- Supports collaborative workflows.

9.6.1.5 Cons

- Few implementations of the IP Studio data model and Media Composition Protocol currently exist.
- Technology is still relatively immature.

9.6.1.6 More Info

- BBC R&D Primer blog page: http://www.bbc.co.uk/rd/projects/nearly-live-production
- AAF ASSOCIATION SPECIFICATION (Advanced Authoring Format (AAF) Edit Protocol): http://www.amwa.tv/downloads/specifications/aafeditprotocol.pdf
- BBC R&D Object Broadcasting: Nuts & Bolts: http://www.bbc.co.uk/rd/blog/2013-08-object-broadcasting-nuts-bolts

9.6.2 TRACAB

9.6.2.1 Goal

TRACAB is a production platform located within sports venues for capturing the live positions of all moving objects in 3D space within its field of view.

9.6.2.2 Overview

TRACAB is a camera-based technology deployed on venue at arena-based sports events. It consists of an array of cameras mounted into two discrete camera boxes, with each box usually containing 3 cameras. The array of cameras in each of the boxes are positioned so that the whole field of play is covered collectively by each box, forming a stitched panorama of the whole area. The resultant two panoramic videos are then processed by the TRACAB computer vision algorithms to determine the physical position of each object to be tracked in 3D space.

The system operator on venue is responsible for assigning identities to sports players so that the system knows which player is which. Once assigned, each player's position and the ball is then automatically tracked and calculated 25 times per second (for every frame of video) at an accuracy of around 5-10cm.

The resultant data-feed output from TRACAB can take several different forms, depending upon the target application needs. It can either be supplied in an extracted format, where specific derived metrics have been calculated based upon agree methodologies. Example metrics include cumulative distances run, current speed etc. An additional RAW positional feed is also available that provides the basic x, y, z coordinates of where the player/ball is at that specific moment in time.

In both cases, the data-feed is provided "live". The latency of the system between a player/ball moving and the data leaving the TRACAB system for other recipient applications is normally in the region of 2-3 frames of delay.

Additionally, due to certain events taking place within certain sports, then the identity of players may be temporarily lost, such as when they leave the pitch or when many bodies group together (e.g. during a goal celebration). To alleviate the provision of the correct identities of players in such scenarios, the system also provides a delayed secondary data feed of a fixed 15 second delay. This gives the TRACAB operator a window of 15 seconds within which to reassign the correct identities of player after they are lost. The 15 second delayed feed is therefore more accurate than the live feed is in terms of player identities during the whole of a match.

9.6.2.3 Pros

- Can be used in multiple arena-based sports
- Non-intrusive method of capturing tracking data which does not interfere with the sport
- Semi-automated operation and consistent accuracy
- Already deployed on many of the top sports leagues in the world
- Data can power a large variety of new innovation concepts in content delivery for both live and post-produced content

9.6.2.4 Cons

- Requires an operator on venue to operate the system
- Panoramic video is low quality as it is not designed as an output

9.6.2.5 More Info

- Official site: http://www.chyronhego.com/tracab

9.6.3 Virtual Placement

9.6.3.1 Goal

Virtual Placement is a software tool used within broadcast productions to track the movement of cameras and to enable the placement of virtual graphics into live video for adding sports enhancements, graphical elements and for virtual advertising.

9.6.3.2 Overview

Virtual Placement is a software application that implements a range of advanced computer vision algorithms and techniques to track the Pan. Tilt, Zoom (PTZ) movement of cameras and then place graphical elements into the live video content. Traditionally, this has only been possible to achieve by fitting broadcast cameras with special hardware encoders and calibrated lenses, which is both costly and technically complex. Virtual Placement achieves the same results through software and without any special hardware on the cameras. Therefore, any raw video has the possibility for virtual or augmented graphics to be placed into a scene.

Depending upon the type of video content as well as the type of graphics elements and enhancements to be placed into the scene, Virtual Placement offers several different placement methodologies to achieve the best end result. These are as follows:

- Pixel Tracking – This base module recognizes the pixels used in a scene to determine camera PTZ. This methodology can only place 2D graphical elements into the 2D video.
- Anchor Tracking – This module uses screenshots taken from video as the reference to where and when to place graphical elements. In dirty-feed video sources consisting of multiple camera angles, this methodology allows
- Line Tracking – This module is designed for stadium based sports where the lines used in the field of play of the sport are used by the system to track camera movement. Examples are for sports such as Football, Tennis, Rugby, NFL etc.). This methodology enables an understanding of 3D space and perspective from the singular perspective of the 2D video.
- Scene Tracking – This module is designed for scenarios where an understanding of the 3D perspective is required but there are no lines or other known geometry in the scene as reference.

In all cases, Virtual Placement requires a calibration process to be performed in advance of being used on the live video. Depending on the content and the module of the software used, this calibration process can take up to 15 minutes of preparation time. Additionally, the calibration may need to be updated dynamically during live production usage to adjust for changing lighting conditions in a scene.

9.6.3.3 Pros

- Non-intrusive method to cameras
- Less complex in setup than using camera encoders
- More cost-effective than using camera encoders
- Can be used on any video content
- Very effective for dealing with video content from unstable camera footage with vibrations
- Can assign different graphical content to different destination clients

9.6.3.4 Cons

- Can only be used on fixed position camera content
- Cannot place 3D objects into 3D scenes

9.6.3.5 More Info

- Official site: http://www.chyronhego.com/virtual-graphics/virtual-placement

9.7 Media and Metadata Delivery

9.7.1 Adaptive Streaming

9.7.1.1 Goal

Adaptive streaming over HTTP provides a reliable, cost-effective means of delivering continuous and long-form video over the Internet. It allows a receiver to adapt the bit rate of the media to the current network conditions in order to maintain uninterrupted playback at the best possible quality. It offers compatibility with large-scale HTTP caching infrastructure to support delivery to large audiences

- HLS (HTTP Live Streaming) is Apple’s adaptive bitrate Internet media streaming technology.
- Smooth Streaming is Microsoft’s adaptive bitrate Internet media streaming technology.
- MPEG-DASH is an open standard to enable adaptive bitrate Internet media streaming.

9.7.1.2 Overview

Error! Reference source not found. provides a summary of key features of these three adaptive streaming technologies, including their manifest types, storage structure, late binding capability and encryption options.

Late Binding: Audio and Video is held separately on the server. Different combinations of audio and video can be requested by each client; this allows many custom combinations while using minimal server space. The audio and video is bound together by the client, late in the process.

Common Encryption (CENC): The media is encrypted using a content key. This content key is in turn protected by multiple different Digital Rights Management (DRM) systems. The resulting media collection is now protected, and accessible by multiple devices using different DRM systems; increasing availability whilst maintaining the same number of files on the server.

Adaptive System	Origins	Used by	Manifest Type	Media stored as	Late Binding	Encryption
HLS	A very early streaming system that was developed by Apple. Still in use today and being extended to allow more features	Mandated by Apple for any streaming app that uses cellular connectivity. Global adoption, available on almost all devices.	Hierarchical files. Explicitly listing every segment	Large collections of separate segments, stored in TS containers.	Not possible, all possible combinations must be created and held by the server.	Typically the manifest is un-encrypted, and all media segments are fully encrypted.
Smooth Streaming	A propriety system developed by Microsoft. Powerful feature set, originally locked to Microsoft systems.	The enhanced security offered makes smooth streaming very suitable for premium VoD. i.e. Netflix	Single, template based file. Manifest describes a single media presentation.	Each representation held as a single large file, audio and video held separately. Each large file is internally fragmented into fMP4.	Client device can request different combinations of audio and video from the server, binding them together as needed.	Encryption can be detailed in either the manifest or within the fragment The header portion of each fragmer left un-encrypted to assist file handling.
DASH	Developed as the final solution, combining the best parts of Smooth Streaming and HLS. Owned by the MPEG Group, released as an open standard.	Not currently in widespread use, but due to its standardisation, many systems are looking to converge on DASH. i.e. YouView.	Single, template based file. The manifest can describe a playlist, allowing pre-roll and advert insertion.	Large collections of separate fragments, stored in fMP4 containers.	Client device can request different combinations of audio and video from the server, binding them together as needed.	Header portion of each fragment left un-encrypted. Manifest holds information for decryption. Supports multiple DRM systems referencing a single encryption key. Common Encryption (CENC)

Table 8 - Adaptive Streaming Technologies

9.7.1.3 More info

HLS:

- https://datatracker.ietf.org/doc/draft-pantos-http-live-streaming/

DASH:

- ISO/IEC DIS 23009-1.2 Dynamic adaptive streaming over HTTP (DASH)
- http://dashif.org/
- http://dashif.org/software/

9.7.2 Media formats supported by HbbTV

The main intention of the HbbTV A/V profile is to reuse codecs used in broadcast services. That means H.264/MPEG-4 Part 10 AVC is the main video codec and AAC the main audio codec. Other codecs are supported if they are for broadcast, e.g. Dolby AC3. Availability of extra codecs is signalled in the XML capabilities that are available through a JavaScript API.

As delivery protocol, HTTP progressive download and, since HbbTV 1.5, MPEG-DASH are available. For progressive download either MPEG-2 Transport Stream or the MPEG-4 file formats may be used as system format. The MPEG-DASH profile for HbbTV 2.0 is DVB DASH. This is backwards compatible and includes the HbbTV 1.5 profile.

https://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.01.01_60/ts_103285v010101p.pdf

The following sections show the codec details and supported video resolutions.

9.7.2.1 A/V codecs

System	**Max bitrate**	**Video**	**Audio**	**Mime-type**
MPEG-2/DVB transport stream	8 Mbit/s	H.264/MPEG-4 Part 10 AVC HD/SD H.265/MPEG-H Part 2 HEVC 8-bit, 10-bit, UHD **	MPEG-4/AAC Dolby AC3 * Dolby E-AC3 **	video/mpeg
MPEG-4 file formats				video/mp4
MPEG-DASH	12 Mbit/s if UHD is not supported 26 Mbit/s if UHD is supported			application/dash+xml
MPEG-4 file formats	-	-		audio/mp4
(RAW)	-	-	MPEG-1 layer 3	audio/mpeg

Table 9 HbbTV A/V Codec Support

* HbbTV 1.0 if supported for broadcast
** HbbTV 2.0 if supported for broadcast

9.7.2.2 Video resolutions supported by HbbTV 1.0/1.5/2.0

Standard	HbbTV 1.0/1.5				
	HbbTV 2.0				
Coding	H.264/MPEG-4 Part 10 AVC		H.265/MPEG-H Part 2 HEVC		AVC/HEVC
Protocol	Progressive MP4 /TS/DASH (HbbTV 1.5)		Progressive MP4 / DASH	Progressive MP4 / DASH	DASH SD/HD/UHD
Label	AVC_SD_25	AVC_HD_25	HEVC_HD_25	HEVC_UHD_25	
Reference	TS 101154 1.9.1 TS 101154 1.11.1 5.6.2.3	TS 101154 1.9.1 TS 101154 1.11.1 5.7.1.4	TS 101154 2.2.1 15.14.2.2	TS 101154 2.2.1 15.14.3.2	TS 103285 1.1.1 10.3
192 x 108					p25/p50
320 x 180					p25/p50
384 x 216					p25/p50
480 x 270					p25/p50
352 x 288	p25/i25				i25
512 x 288					p25/p50
640 x 360					p25/p50
704 x 396					p25/p50
720 x 404					p25/p50
768 x 432					p25/p50
852 x 480					p25/p50
960 x 540			p25/p50		p25/p50
352 x 576	p25/i25				
480 x 576	p25/i25				
544 x 576	p25/i25				i25
720 x 576	p25/i25				i25
704 x 576					i25
1024 x 576					p25/p50
640 x 720		p25/i25/p50			
960 x 720		p25/i25/p50	p25/p50		
1280 x 720		p25/i25/p50	p25/p50		p25/p50
1600 x 900			p25/p50		p25/p50
960 x 1080		p25/i25/p50			

1280 x 1080		p25/i25/p50			
1440 x 1080		p25/i25/p50	p25/i25/p50		
1920 x 1080		p25/i25/p50	p25/i25/p50		p25/i25/p50
2560 x 1440				p25/p50	p25/p50
3200 x 1800				p25/p50	p25/p50
2880 x 2160				p25/p50	
3840 x 2160				p25/p50	p25/p50

Table 10 - HbbTV Supported Video Resolutions

9.7.3 Dolby AC-4

9.7.3.1 Goal

A multi-channel audio codec and container format with special features which will enable the scalable and efficient delivery of object-based, personalised audio and metadata to a variety of devices.

9.7.3.2 Overview

Dolby AC-4 is a product developed by Dolby Laboratories to succeed the existing multi-channel codec marketed as Dolby Digital Plus. The AC-4 codec itself is being standardised by ETSI (TS 103 190 Parts I and II) and the DVB (TS 101 154). It is also included in the ATSC 3.0 specification.

In terms of audio coding, AC-4 improves on Dolby Digital Plus (DD+) by allowing up to 10 independent streams to be encoded (versus 8 streams with DD+) and with approximately 50% better compression. This equates to roughly 160kbps for a high quality 5.1 stream, or 384kbps for a fully-immersive experience.

AC-4 introduces the concept of an audio I-Frame which, when aligned with the equivalent video I-Frame, ensures clean cuts between different streams.

Probably the most interesting feature of AC-4 is its flexible and scalable container format. Each AC-4 frame contains a Table of Contents which enables multiple ‘presentations’ to be defined, each of which define a combination of available audio and metadata streams. This means that a variety of different personalised audio presentations can be generated from a single AC-4 stream.

AC-4 also defines an Enhanced Metadata Format (EMDF) which can be used to generate metadata-only streams which are included within presentations and synchronised with corresponding audio streams. Third parties could potentially define their own sub-schemas which could be recognised by decoders in specific devices: for example, in principle EMDF could be used to carry subtitle information or even triggers to launch applications or content.

For Dolby, the main purpose of AC-4 is to deliver immersive, object-based audio using its Dolby Atmos technology. AC-4 supports spatial coding of up to 17 spatial groups on top of the 10 core channels described above. These can be rendered to any speaker configuration by choosing an appropriate presentation authored within the AC-4 stream.

AC-4 is a new format and support is not yet widespread (January 2016). Unlike one traditional approach of locating the DD+ decoded in an AV Receiver or Soundbar, the preferred approach is for AC-4 to be decoded in the Set-Top Box or client device, which will also provide appropriate controls for personalisation. While hardware support is expected during the lifetime of 2-IMMERSE, a mobile app SDK is expected during early 2016.

9.7.3.3 Pros

- Flexible container format designed for object-based audio.
- Built-in support for personalisation through selectable ‘presentations’.
- EMDF provides opportunities to add functionality and could be used to support multi-screen experiences where alternatives aren’t available (e.g. in a VOD stream?).
- Standardised and will be implemented by major hardware vendors in the near future.
- TV service providers including BT are exploring how they will support object-based audio (Dolby Atmos) for both linear TV and VOD. This can be done to a limited extent using DD+, but AC-4 will be expected later.
- Dolby are keen to support 2-IMMERSE R&D work involving AC-4.

9.7.3.4 Cons

- Hardware implementations not yet available – although they will be during the lifetime of 2-IMMERSE. The first SDK to be available will be only for mobile OS.
- AC-4 tools and encoders are proprietary and will need to be acquired (possibly for free) and learned.
- Browser support will not be available by default – we will need to check whether the SDK provides plugin support.
- Object-based audio demands some changes to the production environment, and for live production this may mean upgrading existing mezzanine formats such as Dolby E to their higher-capacity equivalent (such as Dolby ED2).

9.7.3.4.1 More Info

- AC-4: ETSI TS 103 190 Parts I and II
- DVB TS 101 154 v13 and v14
- The BT team have further literature direct from Dolby which can probably be shared if we inform them first.
- Atmos/AC-4 SDK for mobile – available Spring 2016.

9.7.4 360 degree / Immersive Video

9.7.4.1 Goal

The best entertainment emotionally engages the audience and presents content in a way that provides full immersion in the experience. Whilst this can mean using large screen TVs, new opportunities associated with the emergence of consumer centric immersive virtual reality video and computer generated content presented on tablets and head mounted displays that fill the viewer's entire field-of-view, are being released offering a new immersive experience.

The virtual reality 'hype' of being within a Star Trek style 'Holodeck' where viewers inhabit the same space as the movie they are watching may still be some way off but VR entertainment experiences could be about to move from niche markets into mainstream consumers' homes

9.7.4.2 Overview

Capture

An immersive video is the recording of a real-world scene where the view in every direction is recorded at the same time. The term 'every direction' can of course be limited to a panoramic view with less 'width' than that needed for a full 360-degree view.

Multiple cameras arranged in a circle or arc, with their FOV overlapping, produce a series of images that once stitched together produce an immersive image which can then be further processed to create an immersive video experience.

BT Research & Technology have experimented with two camera rigs to create 360 degree video assets. Costs of the above rigs are approximately £2000 each.

Camera / capture systems are being developed and released with increasing frequency to improve the capture process – of note in this space are Ricoh with their Theta range of cameras - https://theta360.com/uk/, Giroptic - http://www.giroptic.com/ - has a consumer 360 camera and Nokia Technologies are launching a 'high-end' solution with the Nokia OZO - https://ozo.nokia.com/. Google, in collaboration with GoPro, are launching a 16-camera array (using 'synchronised' GoPro Hero cameras) called the Odyssey - https://gopro.com/odyssey.

Processing

Typically, once captured the multiple video sources need to be processed and 'stitched' together to produce a single 360-degree view – commonly this is an off-line process producing VoD content, (as opposed to live video), and is a single stitched 'spherical' video presented as an equi-rectangular video.

Due to the nature of stitching the multiple video sources the resulting spherical videos are relatively high resolution – commonly a UHD-esque 3840x1920 pixel resolution encoded as a high data rate H.264/MPEG-4 Part 10 AVC MP4 encapsulated video file. It should be remembered that when being viewed the user is only presented with a small area of the full video to fill their field-of-view. Whilst the full video may be a 'UHD' resolution the viewer is looking at a 'less than' HD part of the video. Capture resolutions as well as output resolutions are likely to increase beyond UHD/4K with a resultant increase in power required for processing these videos.

Software solutions for off-line processing are available from Kolor within their Autopano suite - http://www.kolor.com/ - and from VideoStitch - http://www.video-stitch.com/ - amongst others.

Google's software solution, working solely with the Odyssey camera rig, moves the processing and stitching of the separate videos into the 'cloud' using much of the same 'backend' as YouTube.

Real-time, live, capture and processing is increasingly available – although still in early production stages. High-end solutions from NextVR - http://www.nextvr.com/ - have been used for high-profile sports events using proprietary and modified broadcast equipment. VideoStitch has a 'consumer' product, VahanaVR - http://www.video-stitch.com/live-vr/ - that uses commercially available capture cards and GoPro style camera rigs, combined with fast CPU and GPU processing to provide live 360-degree video.

Delivery

YouTube has supported upload, storage and playback of 360 degree video, (https://www.youtube.com/results?search_query=360+video), since March 2015 and this support covers all playback options, desktop and app on all platforms. Roll-out of 'split-screen' playback view allowing for 3D stereoscopic video is also beginning, (available on Android platform as of January 2016). Facebook also supports 360 degree videos in News Feeds across multiple platforms.

There are a number of dedicated players for 360 degree content available across desktop and mobile platforms, (e.g. the Kolor Eyes player - http://www.kolor.com/kolor-eyes/), offering playback of web-hosted content, (either streaming video or download-and-play), or locally stored video files.

Due to the video resolutions used, (and end-user display processing), delivery of 360 degree video is at the higher end of current platform requirements – the best experience for YouTube hosted videos is by selecting the 2160p display mode and on a 'superfast' broadband network.

Solutions involving adaptive video streaming and 'tiled' streaming are being explored, (Facebook is now offering 'Dynamic streaming' of 360 videos, storing all of such content at multiple resolutions and only displaying the best quality at the area of the video being currently looked at, the remainder being lower resolution automatically swapped when the view changes - http://www.engadget.com/2016/02/21/facebook-dynamic-streaming-gear-vr/).

Consumer viewing

The recent technology developments in the field of consumer-orientated virtual reality have revolved around displays, video and graphics processing, motion sensing, motion tracking and the creation of 'immersive' content. Coming together in the form of Head Mounted Displays, (HMD's), these use near-eye screens and lenses to fill the user's field of view and include sensors to translate head movements into changes in POV within the content being watched.

Leading 'high-end' display solutions are from Oculus, (https://www.oculus.com/en-us/rift/), and HTC, (http://www.htcvive.com/uk/).

Google introduced a much lower cost 'entry-level' display solution, utilising smartphones for the display and POV-changing motion sensors, the Google Cardboard - https://www.google.com/get/cardboard/ - typically costing as little as £3.

Samsung have gained a lead in the consumer market with their 'first to market' solution based around their Galaxy smartphone hardware. The Samsung Galaxy GearVR (http://www.samsung.com/global/galaxy/wearables/gear-vr/) primarily uses a smartphone for the processing and display of immersive content but with higher quality materials and lenses than those found in Google's cardboard. Samsung, in collaboration with Oculus, provide a curated and high-quality CGI app store for GearVR immersive content discovery and showcases.

Whilst using some form of HMD results in the best immersive experience 360 degree video content displayed on a tablet or smartphone – as a 'window on another world' – offers a lower 'barrier to entry' and can be equally compelling. Using the KolorEyes content player in tablet mode as an example - https://itunes.apple.com/gb/app/kolor-eyes-360-video-player/id551037018?mt=8 and https://play.google.com/store/apps/details?id=fi.finwe.koloreyesandroid&hl=en_GB

9.7.4.3 Pros

- 360 degree video provides a highly immersive viewing environment to content consumers – a feeling of being there.
- Capture, processing and delivery systems are in place and share features and requirements of existing IP video delivery.
- A wide range of viewing options are available –fully immersive HMD's, smartphone based solutions of various quality, desktop PC, tablet and smartphone options.
- High 'wow' factor

9.7.4.4 Cons

- Primary focus for much of the resurgent VR industry is on games and CGI content, not 360 degree video
- POV is interactive but camera position is static or 'captured' at content capture time
- Capture and initial processing resolution may be high but typically results in a relatively low 'perceived' resolution to the end user
- The current 'best' immersive media experiences are socially isolating and all-encompassing (using a HMD and headphones)

10 Conclusions and Next Steps

In this document we have presented the 2-IMMERSE system architecture. As noted in the Executive Summary, this is work in progress. We expect that the architecture will evolve both as we refine it and specify it in more detail (for example through detailed component interface specifications), and as we address the expanding scope of the four multi-screen service prototypes through the project. Within the project we will keep this document updated to reflect this development.

We have taken a ‘layered’ approach to documenting our system architecture, in order to maximise clarity, and maintain an appropriate ‘separation of concerns’:

- The platform is defined as a set of services, which support the client devices applications. In defining these services, we have described the service functionality, key interfaces, and technology choices where they have been made.
- The client application architecture defines a common HTML and JavaScript environment for the Distributed Media Application components, and the underlying application that manages their lifecycle and presentation. It also details how this is supported on the various devices that participate in the system.
- The production architecture is defined at a high level; however, we note that a detailed, generalised production architecture is difficult to create, and specific production architecture will be determined for each trial.

The next step in developing the system architecture is to specify the component interface specifications, which will be published in project deliverable D2.2 Platform-Component Interface Specifications. Alongside the development of these interface specifications we will develop skeleton implementations of these components to validate the interface specifications. These will then be further developed to support the functionality required for the first multi-screen service prototype (Watching Theatre at Home).

Annex A Watching Theatre at Home User Stories

The user stories in the table below describe functionality required by the 2-IMMERSE system for the Watching Theatre At Home service prototype. These are described from the point of view of two actors; the user at home and the producer. Originally published in D4.1, this version has been augmented with agreed basic priorities, and implementation notes, which capture current thinking on how described functionality might be implemented. Both were the outcome of project discussions.

N.B. These user stories and notes make reference to a 'box'; this is the metaphor for users sharing an experience (i.e. in the same session) but in different physical locations (contexts), based on the concept of the theatre box.

As a user at home	As a producer	Priority	Implementation
HT001 **Before the live performance of play on TV I want to use Facebook and Twitter to make it clear to my friends and others online that I will be watching the play at home later, so that I can later invite some of them to watch the play with me.**	HT101 Before the performance I want to be able to alert users of the system about this new production, its casting and its distinctive features, so as to be able to attract users.	Essential	Trivial – doesn't require additional implementation in 2-IMMERSE
HT002 **Before the live performance I want to learn from Facebook and Twitter who else is planning to watch so that I can choose who to invite to watch with me.**		Essential	Trivial – doesn't require additional implementation in 2-IMMERSE
HT003 **Before the live performance I want to be able invite to people to watch with me so as to share the experience of watching the performance with my friends or others.**	HT102 Before the performance I want to be able to send personalised invitations to those who have used the system before to encourage them to participate in this new experience. This is so that I can maximise the number of users and revenue.	Essential	Needs a registration process and the ability to associate users with boxes (using the Authentication & session (Lobby) services) Need to sort out the process for getting our apps on the appropriate app stores and directing people to download them. Box guests can be edited until a specific time before the show. We will mandate that all trialists have at least a TV – so box participants will need to be firmed up a while before the show.
HT004		Essential	

As a user at home	As a producer	Priority	Implementation
Before the live performance I want to be able to receive invitations to watch with others, and to accept or decline these, so that I can control with whom I will be watching the performance.			
HT005 **Before the performance I want to be able to use my credit card or PayPal to purchase access to the production and its enhanced features, so as to be able to participate in the experience.**	HT103 Before the performance I want to be able to process payments via credit cards and PayPal so as to develop a revenue stream for the production.	Not needed for trial	
HT005b **Before, during and after the performance I want to see and be seen, hear and be heard, by others within the group who I have chosen, and to be able to exchange private and group text messages within this group, so that we can enjoy each other's company with the performance as a focus, and to exchange ideas and reactions prompted by the play.**		Essential	Box of one is possible. Need to limit the maximum number of people who can be in a box. Needs to be at least 6. (lobby service requirements) WebRTC will enable this, although audio is a major problem for implementation and will need to be addressed.
HT006 **Before, during and after the performance I want to be able to access relevant text, image, audio and video resources about the play, the production, the cast and crew, made available by the producer so that my experience and appreciation of the broadcast can be enriched and made more compelling.**	HT104 Before, during and after the performance I want to be able to provide relevant text, image, audio and video resources about the play and production before, during and after the broadcast. This is both to provide a rich, compelling experience for the home user and to add value so as to differentiate my media offering from those of competitors.	Essential	Relatively trivial to implement as DMApp components, made available at appropriate time by the timeline service.
HT007 **Before the performance I want to be able to access live and interactive 360-degree video and audio from the foyer of the theatre, so as to feel that I am part of the communal experience of watching the play with a physical audience.**	HT105 Before the performance I want to make available a 360 live video feed from the foyer of the theatre so that a home user can access content that mirrors the experience of arriving at a theatre and enhances the anticipation and sense of event enjoyed by those	Not needed for trial.	Too complex given the potential benefits in the trial.

As a user at home	As a producer	Priority	Implementation
	attending a physical theatre.		
HT008 **Before the performance, or at any point during it, I want to be able to personalize my access to the resources available to me (in, for example, HT006, HT016 and HT020) in order to set their address to *Introductory*, *Informed* or *Expert*. This is so that I can receive materials that will best enhance my experience and understanding of the production.**	HT106 Before the performance, or at any point during it, I want to be able to facilitate personalized access to the resources that I am making available (in, for example, HT006, HT016 and HT020) so that I can offer materials that will best enhance my user's experience and understanding of the production.	Desirable.	A lot more content will need to be produced. Implementation within the architecture should be feasible (either as multiple DMApp component versions, or as single components that adapt their presentation according to selected expertise level - TBD).
HT009 **During the broadcast I want to be able to view the various media streams as below on both the main screen in the room and on one or more second-screen devices, and to be able easily to switch these, so as to achieve control over the viewing experience.**	HT107 During the broadcast I want to be able to produce, and control the available options for displaying the theatre play within the home so that I can provide the most satisfying communal simulation of theatre-going for the user – and in this way attract her to revisit the experience in the future.	Essential.	Users will be able to control and personalize layout. We may not be able to have complete flexibility to show all streams on all screens due to device limitations (e.g. if we can't get cloud composition ready in time). Note that a media stream here could be information content as well as streaming video.
HT010 **During the broadcast I want to be able to see a graphical display of how many other home viewers are watching at any moment during the broadcast, so that I can appreciate and enjoy being part of a simultaneous communal experience.**	HT108 During and after the broadcast I want to be able to access detailed analytics about those who are watching, where, on what devices and for how long, so that I can understand the behaviour of the users and – potentially – refine future offerings to make them more attractive.	Essential	A good first implementation of the logging and analytics services. We may want to fake the data during our first trial to give the illusion of a larger audience.
HT011 **During the broadcast I want to be able to rate on a scale of 1 to 10 my current assessment of the production, so that I can express my developing responses and feel that I am contributing to a communal assessment.**		Not needed for trial.	
HT012 **During the broadcast I want to be**		Not needed for trial.	

As a user at home	As a producer	Priority	Implementation
able to see an aggregated total of the ratings of all those who are watching simultaneously, so as to monitor and assess the responses of the audience and to measure my own responses against the broader view.			
HT013 **During the broadcast I want to be able to choose to view either the broadcast mix coverage of the production being created by the screen director or to view a static wide shot of the stage or both. This choice will allow me to experience the broadcast in a "purer", less mediated or narrativised form.**	HT109 During the broadcast I want to be able to offer the full mix as created by the screen director and a static wide-shot of the whole stage, so as to allow users to toggle between these and so achieve a more inclusive understanding of the production.	Essential.	This is a specific expression of HT009 which relates to video only.
HT014 **During the broadcast I want to be able to view synchronised sub-titles for the production, either on a second-screen device or overlaid on the main performance feed. If I am hard-of-hearing I want to do this to enjoy the broadcast fully; and if my hearing is good I may want to do this if I find the language of the playwright (e.g. Shakespeare) unfamiliar and a bar to achieving a satisfying understanding.**	HT110 During the broadcast I want to be able to offer synchronised sub-titles so as to enhance the experience for the user.	Essential for subtitles only.	Subtitles only to be rendered on the TV, not on the second screen device. Note IRT's existing work on subtitling MPEG-DASH streams. We will need to understand how to obtain and process the subtitle feed from RSC.
HT015 **During the broadcast if I have restricted sight I want to be able to access synchronised audio description for the production, so as to understand and appreciate fully what is being shown.**	HT111 During the broadcast I want to be able to offer audio description so as to enhance the experience for the user.	Not needed for trial.	
HT016 **During the broadcast I want to be able to access synchronised information and commentary in the form of image and text created by the producer, so that these elements can enhance my viewing experience, deepening my engagement and understanding.**	HT112 During the broadcast I want to be able to offer synchronised information and commentary (created in a cost-effective manner before the production) so as to enhance the experience for the user.	Essential.	Implementation within the architecture should be feasible using the Timeline Synchronisation service.
HT017 **During the performance I want to**		Not needed for trial.	

As a user at home	As a producer	Priority	Implementation
be able to use Twitter from my second-screen device so as to contribute to an unfolding discussion of the production and to view similar contributions by others. This is so that I can feel engaged in an active and developing discussion, which may be light-hearted or serious, of the production.			
HT018 **During the performance I want to be able to access synchronised subtitles (cf HT013) and/or synchronized text commentary (cf HT015) and/or comments via Twitter (cf HT017) on either my second-screen device or as overlays on the main screen or as elements on the main screen with the main image inset. This is so that my viewing experience can be as flexible and as responsive as possible.**		Essential	Roundup of previous requirements.
HT019 **At scheduled moments during the performance (such as towards the ends of scenes) I want to be able to express my approval of the production and have that approval aggregated with that of others who are watching and displayed as audio (such as applause) or in a visual form. This is so that I can feel I am participating in the full social and communal experience of experiencing a play.**	HT113 At scheduled moments during the performance I want to be able to feedback the approval expressed via HT019 to the performers, either in an audio or visual form, so as to provide them with an understanding of those who are watching and their appreciation of the show.	Not needed for trial.	
HT020 **During and after the broadcast I want to be able to contribute text comments about the production, so that I can share my knowledge and responses, and I want these comments - after moderation by the producer – to be preserved in a layered structure that be accessed synchronously with the “as live” video on demand offering of the production.**	HT114 During and after the broadcast I want to be able to moderate the text comments about the production, so that I can control these comments and create a lasting version that can continue to be enhanced.	Not needed for trial.	John envisages that this scenario will be implemented within the Theatre in Schools trial.
HT021	HT115	Not needed	

As a user at home	As a producer	Priority	Implementation
At any point after the live broadcast I want to be able to access an "as live" recording of the broadcast with the functionality of many of the URs already specified. This will allow me – if I have missed the live broadcast - to recreate many of its elements at a time that is convenient to me.	At any point after the live broadcast I want to be able to provide to users an "as live" recording of the broadcast with the functionality of many of the URs already specified. This will allow me to further maximize the number of users and the revenue for the production.	for trial.	
HT020b **After the broadcast I want to be able to offer feedback to the producer about any and all aspects of the production, and to know that feedback has been communicated, so that I can feel I have an involvement in the shared experience of the production.**	HT116 After the broadcast I want to be able to receive feedback from users about any and all aspects of the production, to acknowledge their comments, and to communicate individually and collectively with them both about this production and those in the future.	Not needed for trial.	

Annex B Intra-Location and Inter-Location Media Synchronisation with DVB-CSS

A multi-device synchronised experience consists of a number of media objects that will be presented on separate participating at times specified on the experience timeline. These media objects are manifested in the experience by DMApp components and can be discrete such as images, infographics or continuous such as live/on-demand audio/video streams.

In this section, we firstly outline how timeline relationships are specified to facilitate the conversion of timestamps from one timeline to another (for example, between the experience timeline and the media objects' timelines). An overview of the WallClock Synchronisation service is provided next. The possible configurations of Timeline Synchronisation components for intra-home, inter-home sync scenarios are covered in the sections that follow. The possibility of hybrid scenarios where concurrent intra-home sync experience merge into an inter-home sync experience is also covered.

B.1 Timeline Correlations

A correlation between two timelines specifies a deterministic linear monotonic relationship between them, such that there is a one-to-one mapping between time values on each timeline. It is expressed as a pair of corresponding timestamps (from each timeline) and a speed constant. The relationship between the experience timeline and the timeline of a media object in the experience can be expressed thus:

$$[T_{Exp}, T_{Content}, \text{Speed}]$$

A time value on the media timeline $T_{Content}$ and the corresponding time value T_{Exp} on the parent timeline represent the same moment in time. The speed is a number that controls how rapidly time moves on the media timeline as time moves on the parent timeline. The timestamps need not be expressed in terms of the same units; the timelines can have different tick rates. For instance, T_{Exp} can be in milliseconds whilst $T_{Content}$ is in PTS ticks (for an MPEG-2 TS video stream).

For an on-demand video stream, for example, the correlation is relatively simple e.g. $[T_{Exp}, 0]$ – at time T_{Exp} on the experience timeline, play video stream from start position. The notion of a start time position is somewhat different if the media object is a live or time-shifted video stream. The timing information signalled in the stream need not have time-zero as the starting position for a programme (e.g. PTS in MPEG-2 TS); the counter used for injecting timing may even rollover.

For live video streams, this correlation mapping may only be known at production time.

B.2 WallClock Synchronisation

To realise a multi-device synchronised experience, each companion device must report the progress of its own media timeline to a master device (as in DVB-CSS) which then decides on a reference timestamp that all companions will adapt their playback to. However, the timestamp will be out-of-date when it arrives at the companion due to network and processing delays. To account for latencies in network communication between master terminals (e.g. an HbbTV) and their companions, the timestamps exchanged make reference to a shared Wall Clock.

The DVB-CSS Wall Clock synchronisation protocol (CSS-WC) uses clock synchronisation techniques to establish a best effort approximation of the TV's Wall Clock at the companion device. The clock synchronisation technique is based on the client-server mode of operation in NTP. The TV is assumed to have some kind of local system clock known as the Wall Clock. The companion uses the clock synchronisation protocol to create its own local model of the TV's Wall Clock.

The WallClock Service performs the same function as the WallClock synchronisation mechanism in DVB-CSS but has to accommodate, in addition to site-local setups (TV and Companion Devices on same local network), situations where devices are distributed.

Site-Local Clock Synchronisation

For site-local clock synchronisation, the WallClock service uses the CSS-WC protocol to establish the common reference time between the TV and the companion devices. The WallClock service runs on the TV device (this comes for free on HbbTV 2.0 televisions) and timestamp exchanges in the CSS-WC protocol are carried out over UDP for low-latency communication (avoiding the SYN-ACK handshake and congestion control delays of TCP).

The clock synchronisation model is a best-effort estimate. It enables the companion to estimate device, at any given moment, what time value the TV's Wall Clock will have. The companion will create its model of the TV's WallClock by determining the relationship between its own local system clock and the Wall Clock of the TV.

System clocks can drift over time; no two crystal oscillators are perfectly identical. Periodic timestamp exchanges with the WallClock service are therefore required by each companion device to adjust their WallClock offset.

Distributed Clock Synchronisation

In the distributed configuration, the WallClock Service runs on a cloud platform and WallClock service clients perform the timestamp-exchanges with the service via WebSockets instead of UDP. A WebSocket-based variant of the CSS-WC protocol is employed for carrying the timestamped request/response messages.

The reason for this is that port restrictions across firewalls may prevent the CSS-WC UDP messages from travelling from one home network to another via the internet. The question then arises as to why do this instead of relying on the system clock in each device being synchronised using NTP?

The problem with relying on NTP is that for many classes of application, there is no ability to ask the host operating system whether it's clock is synchronised and how accurate it is. There is no control within the application over which NTP server the host device synchronises to. Also, more accurate results are achieved (in principle) if there are fewer network hops involved.

It is likely that using a WebSockets-based protocol for clock synchronisation will achieve lower sync-accuracy results than CSS-WC UDP-based protocol. If the sync-accuracy exceeds the perceptible thresholds of asynchrony for inter-home experiences, then other options such as using UDP-based CSS-WC across sites or using a different protocol for clock synchronisation altogether such as W3C Timing Object will be explored.

B.3 Timeline Synchronisation Service

The DVB specification for Companion Screen and Streams (DVB-CSS, hereafter) defines the necessary concepts, functional roles, overall architecture and abstractions to deliver up to frame-accurate CS experiences bridging DVB-based broadcast services and companion content. HbbTV 2.0 adopts the same architectural abstractions, interfaces and protocols as DVB-CSS for synchronisation between a master TV terminal and one or more slave CS applications.

The Timeline Synchronisation Service performs similar functions as the Media Synchronisation Application Server (MSAS) specified in DVB-CSS (DVB Bluebook A167-2, Section 4.2.2).

Timeline Service as a Synchronisation Client

The most basic operation of the Timeline Synchronisation Service is to forward timeline updates from one device (the synchronisation master terminal) to other devices (the synchronisation slaves). A rudimentary implementation of the service need not even require presentation timestamps from all devices to compute a reference presentation timestamp that all devices can comfortably achieve. Only presentation timestamps from the synchronisation master is necessary.

In fact, Sync Client A need not be a device at all. It can be anything that can 'play' a timeline and periodically provide timeline updates using a synchronised WallClock as a time reference. As the Timeline Service manages an experience timeline, it is can feasibly itself be a source of timeline updates to devices participating in the experience. This is illustrated by Figure 8.

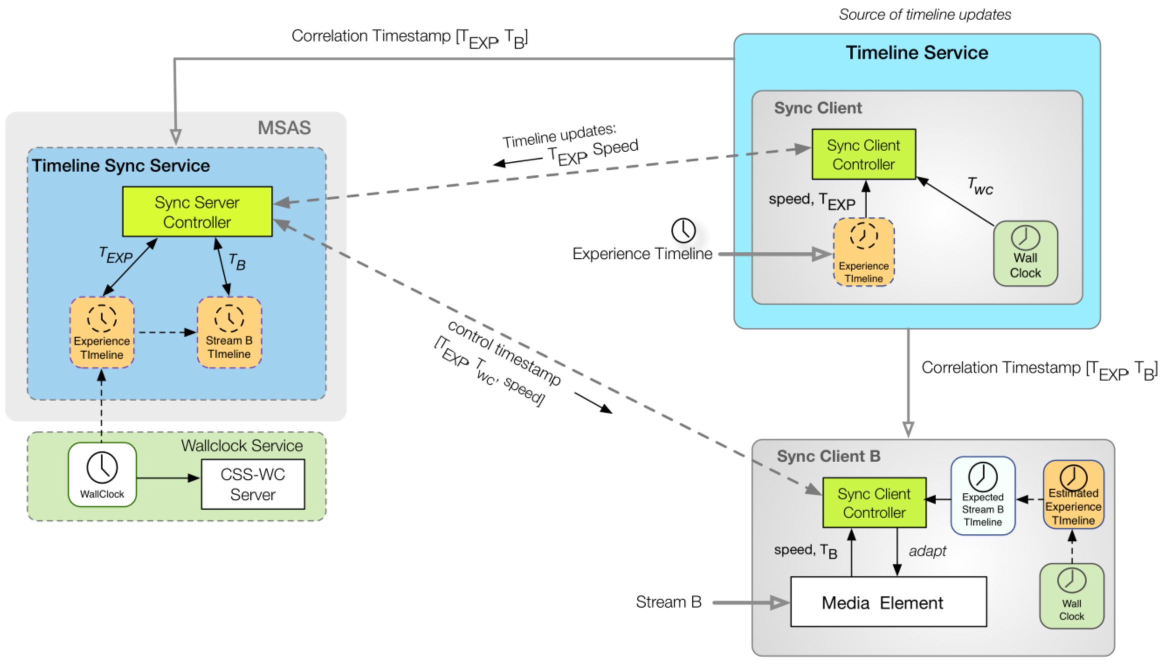


Figure 8: Timeline Service as a Synchronisation Client

B.4 Reusing DVB-CSS Interfaces

In the abstract depiction of the Timeline Synchronisation Service in Figure 6, the service, its clients and the timeline service are assumed to be runnable anywhere i.e. they can be co-located on the same device, run on devices within the same local network or run as a cloud service.

An HbbTV 2.0 stack is an instance of the model where both the Timeline Synchronisation Service and the Synchronisation Client are co-located in a television. Similarly, a cloud-based synchronisation solution is an instance of the model where the Timeline Synchronisation Service runs as a SaaS in the cloud.

Where inter-process/network communication is required between the Timeline Sync Service and a Sync Client, the interfaces defined in DVB-CSS can be used:

- CSS-WC (for clock synchronisation)
- CSS-TS (for timeline updates) and possibly

- CSS-CII (for content id and WC/TS endpoints dissemination)

The caveat for using the CSS-WC interface is that it was designed for local-network operation and relies on UDP for transferring message payloads. For distributed clock synchronisation, a variant of the CSS-WC protocol based on W3C Web Sockets may be used or alternatively, the W3C Web Timing API can be used to perform WallClock synchronisation. With the second option (W3C Web Timing API), the CSS-WC in an HbbTV 2.0 device can still be used on the home network for WallClock synchronisation with companion devices; cross-home WallClock Synchronisation is achieved using W3C Web Timing API and Timing Objects.

This approach ensures that for intra-home inter-device synchronisation, HbbTV 2.0 devices are used with their existing media sync-protocol stack. For inter-home synchronisation, the same HbbTV 2.0 stack can be used but augmented with a W3C Web Timing API client to perform distributed clock synchronisation.

The interfaces used for communication between the Timeline Sync/WallClock services and Synchronisation Clients are summarised in Figure 9.

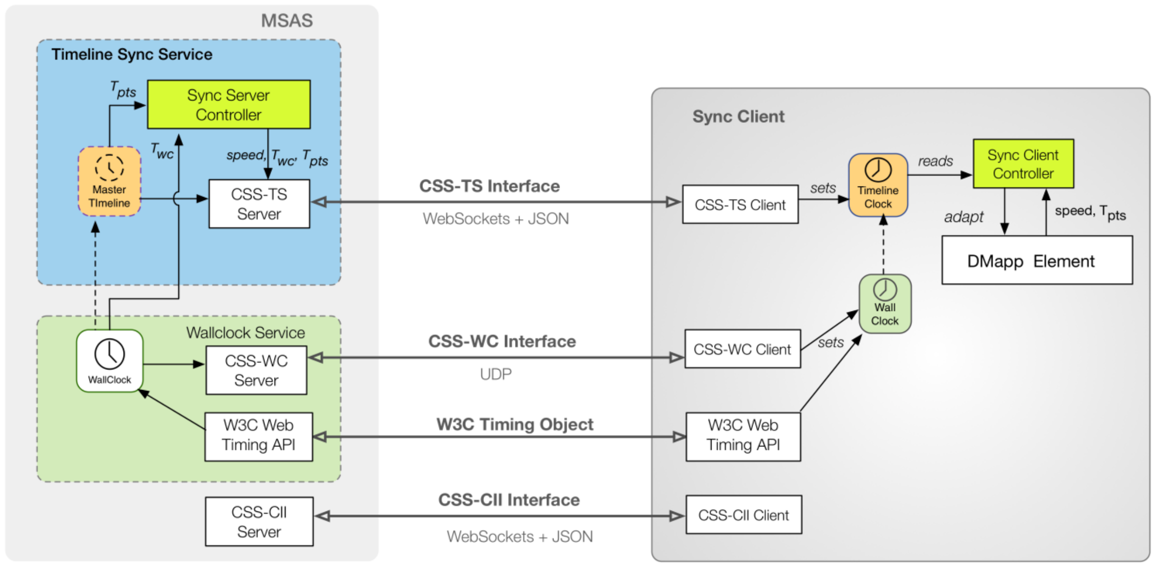


Figure 9: Communication interfaces between the Timeline Sync/WallClock services and Sync Clients

B.5 Site-Local/Intra-Home Synchronisation

For scenarios requiring local synchronisation (i.e. intra-home inter-device synchronisation), a configuration of our model is used where both the Timeline Synchronisation service and the WallClock service run on the master device. This configuration is illustrated in Figure 10.

Both services are part of the HbbTV 2.0 stack and will therefore be available on HbbTV 2.0 compliant televisions. These services are launched when inter-device synchronisation is enabled on the TV. Upon launch, service endpoints (CSS-WC server and CSS-TS server) are created for the WallClock Service and the Timeline Sync Service respectively.

In addition to the Timeline Synchronisation service and the WallClock service, a Content Id & other Information Service (CSS-CII server) is also present in the HbbTV 2.0 media synchronisation stack. The purpose of this service is to advertise content identifiers and WC/TS interface endpoints to companion screen applications. A Synchronisation Client is also available as part of the HbbTV stack to report content timeline positions to the Timeline Synchronisation service when the television plays a media stream.

The companion screen application is effectively a Synchronisation Client. It is provided a CSS-CII endpoint URL after the DIAL-based TV- discovery process and can then proceed to create a WebSocket connection with the CSS-CII server. Through this CSS-CII protocol connection, the Companion Screen App is informed of the content id of the content currently playing on the TV and the CSS-WC/CSS-TS endpoints URLs[8]. Using these, the companion can *i)* initiate WallClock synchronisation using the CSS-WC protocol, and *ii)* connect to the Timeline Sync Service's CSS-TS server to receive timeline updates[9].

The timeline updates enable the Companion Screen App to update its estimate of the TV's timeline. Using a correlation between the TV's content timeline (Stream A) and the companion content timeline (Stream B), the companion can determine the expected position on its timeline at the current WallClock time. It can then proceed to adapt the playback of its content (Stream B) to reflect this updated position (seek in, speed up or slow down the media playback).

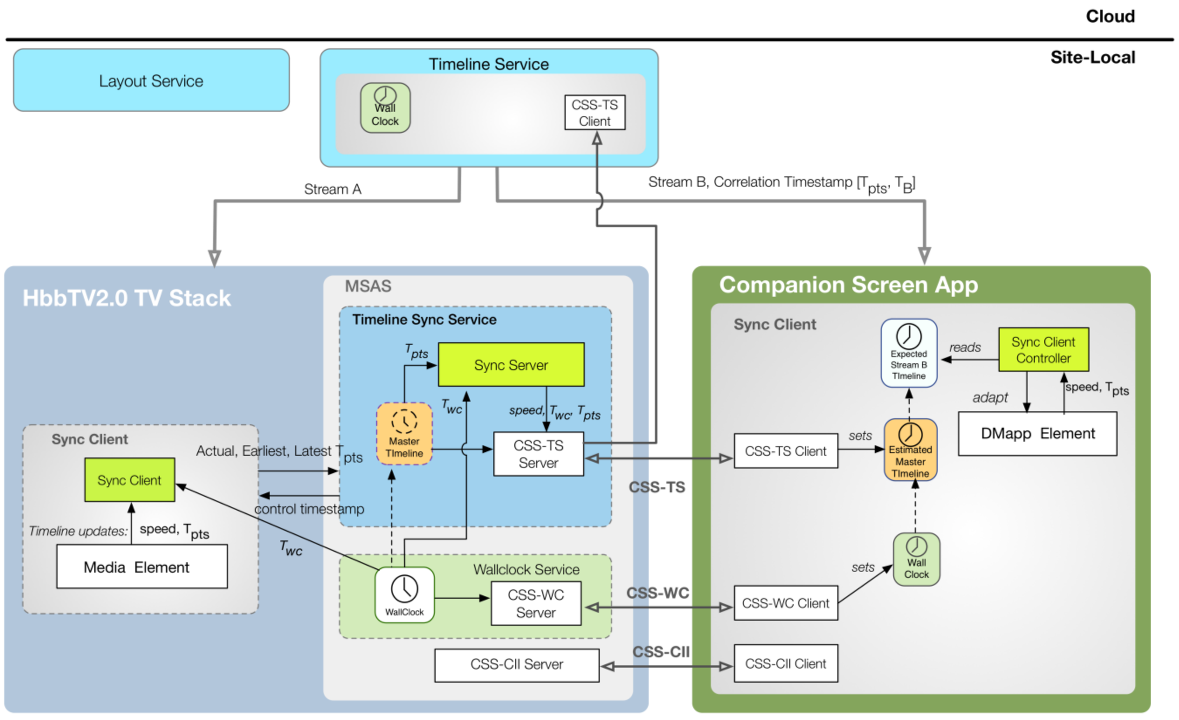


Figure 10: Timeline Sync Service and WallClock Service deployment for intra-home sync

The Layout Service and Timeline Service are shown in Figure 10 as site-local deployments. This is for illustrative purposes only; they may run on the TV device, a set top box or in the cloud.

In either case, the Timeline service needs to maintain an awareness of the experience timeline progress. This is because it is notified of newly available media objects/DMApp Components during the experience and needs to instruct the Layout service to make a decision on where to present these

[8] The companion is also informed about available timelines it can use to synchronise against.

[9] The synchronisation client selects a timeline to receive updates about.

media objects/DMApp Components. The Timeline service achieves this by instantiating a CSS-TS client endpoint to receive timeline updates from the television.

B.6 Inter-Home Synchronisation

For inter-location synchronisation scenarios such as distributed home sync, a configuration of our abstract model is selected such that timeline synchronisation is centralised as a service running in the cloud. In this configuration, as shown in Figure 11, the Timeline Synchronisation Service, the WallClock service along with the Timeline and Layout services, are all deployed as part of the UXEngine running in the cloud.

The Timeline Service retrieves and manages an experience timeline; it can therefore be used as a source of timeline updates as the experience progresses. The Timeline Service in Figure 11, therefore, also includes the functionality of a Synchronisation Client. The Timeline Service enables synchronisation as a service to be available for a particular experience by launching the Timeline Synchronisation Service and passing to it experience metadata such as Correlation Timestamps, context identifiers, etc. The Correlation Timestamps ([T_{exp}, T_B] in Figure 11) enable the Timeline Synchronisation Service to map time values from the experience timeline to the media objects timeline bn(media object played by the devices).

Once launched, the Timeline Synchronisation Service can receive experience timeline updates from the Timeline Service and forward these updates to any devices connected to it.

The cloud-based WallClock service enables devices at different locations (e.g. different homes) to establish a common WallClock. It uses a mechanism such as the W3C Web Timing API to achieve Clock Synchronisation.

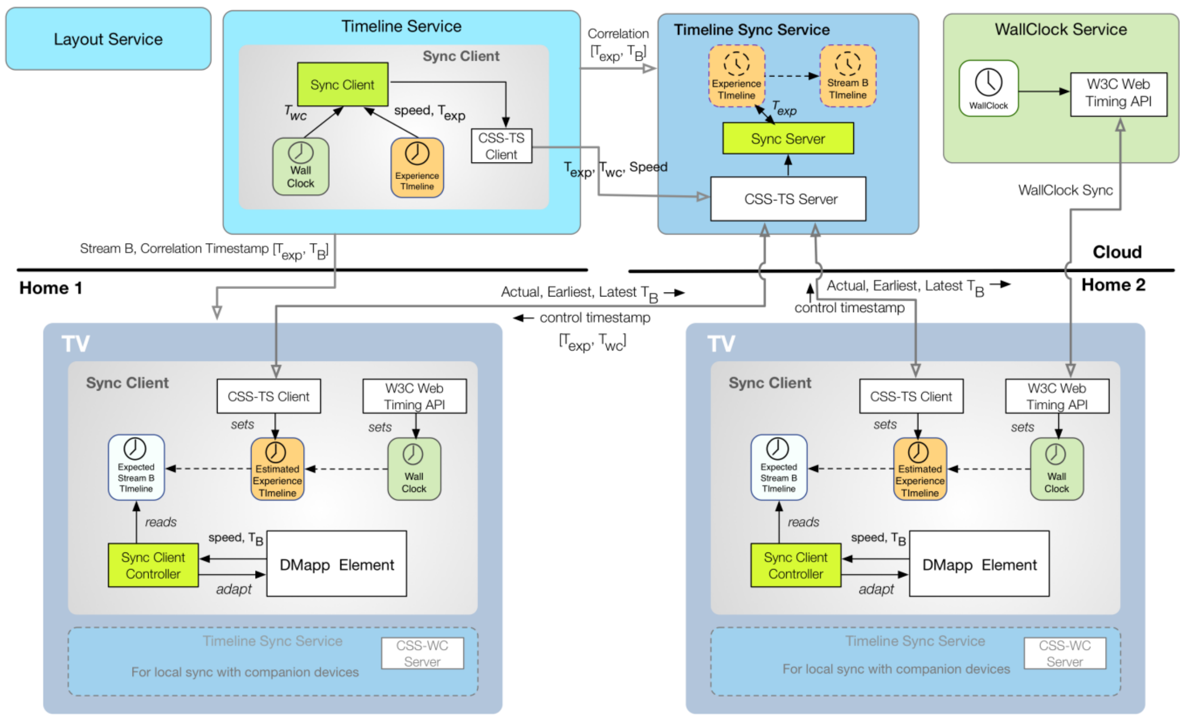


Figure 11: Cloud-based service deployment for inter-home sync

In this inter-home synchronisation configuration, the HbbTV terminals in each home are essentially Synchronisation Clients. The component driving the distributed experience is the Timeline Service via the experience timeline i.e. the Timeline Service is the synchronisation master and the HbbTV terminals are the slaves[10].

The HbbTV terminal in the home will be informed of the Timeline Synchronisation Service and WallClock Service URLs by the Timeline Service during the experience bootstrapping process. Using the WallClock Service URL, the HbbTV terminal can initiate the WallClock synchronisation process using the W3C Web Timing API. To receive timeline updates (control timestamp in Figure 11) from the synchronisation master (the Timeline Service), the HbbTV terminal brokers a connection with the CSS-TS server of the Timeline Sync Service. Based on the received control timestamp, the HbbTV terminal in each home can update its estimate of the experience timeline (i.e. the correlation of the experience timeline with the WallClock). Using Correlation Timestamps from the Timeline service, it can map the current position on the experience timeline to the expected current position on its content timeline. By comparing the expected content timeline position to its actual content timeline position, the HbbTV terminal can determine how to adapt its playback to move to the expected position.

Although, the HbbTV devices in the inter-home synchronisation scenario are slaves to the Timeline Service, they can be synchronisation master in their own home dominion. Thus if companion devices want to synchronise to an HbbTV already participating in inter-home sync, a local Timeline Sync Service (shown as greyed out service in HbbTV device in Figure 11) can be started on each TV to mimic the intra-home sync configuration described in the previous section.

B.7 Intra-Home Synchronised Experiences Merging into an Inter-Home Synchronised Experience

It is possible for separate intra-home synchronised experiences to merge (after having already started) into one inter-home synchronised experience. This could for example represent a scenario where two separate Theatre-in-the-home experiences are launched at different locations. At each location, a television device plays an on-demand video stream showing the performance and additional content appearing on companion devices are synchronised with the TV. After a while, the persons in the two concurrent experiences decide to merge their experience into a common one.

For the user journey to be plausible, it assumed that

1) the televisions play on-demand video streams,

2) they have the capability to seek within the stream and

3) the timeline positions of the TV's media stream at each location differ only by a relatively small amount.

When the separate intra-home synchronised experiences have been launched, then the deployment configuration at each location may be site-local. i.e. at each location, a configuration of components as shown in Figure 10 can be envisaged. The synchronisation master in each home is the Synchronisation Client running on the TV (part of HbbTV 2.0 stack).

As explained in the intra-home sync section, inter-device synchronisation on the local network is achieved through the DVB-CSS suite of protocols. Each TV runs a site-local Timeline Synchronisation Service to which companion devices connect to receive timeline updates (control

[10] The functionality for HbbTV terminals to function as synchronisation slaves is optional.

timestamps). The Timeline Service also runs a CSS-TS protocol client to be informed of timeline updates sent by the TV.

When both intra-home synchronisation experiences are merged into a single inter-home experience, an additional Timeline Synchronisation Service instance will be instantiated in the cloud-based UXEngine. A hierarchical configuration of Timeline Sync services is obtained where the Timeline Sync service instances in the homes (although still synchronisation masters in their own dominion) are now slaves to the cloud-based Timeline Sync service instance.

As shown in Figure 12, each site-local Timeline Service receives timeline updates from the TV's Timeline Sync service instance, specifying the time position the TV is at. The Timeline Service will forward these presentation timestamps to the cloud-based Timeline Sync service. After having received TV presentation timestamps from all homes, the cloud-based Timeline Sync service computes a presentation timestamp that all TVs can achieve. The suggested presentation timestamp is communicated back to each location-specific Timeline Service which will instruct the TV to align itself to this new correlation.

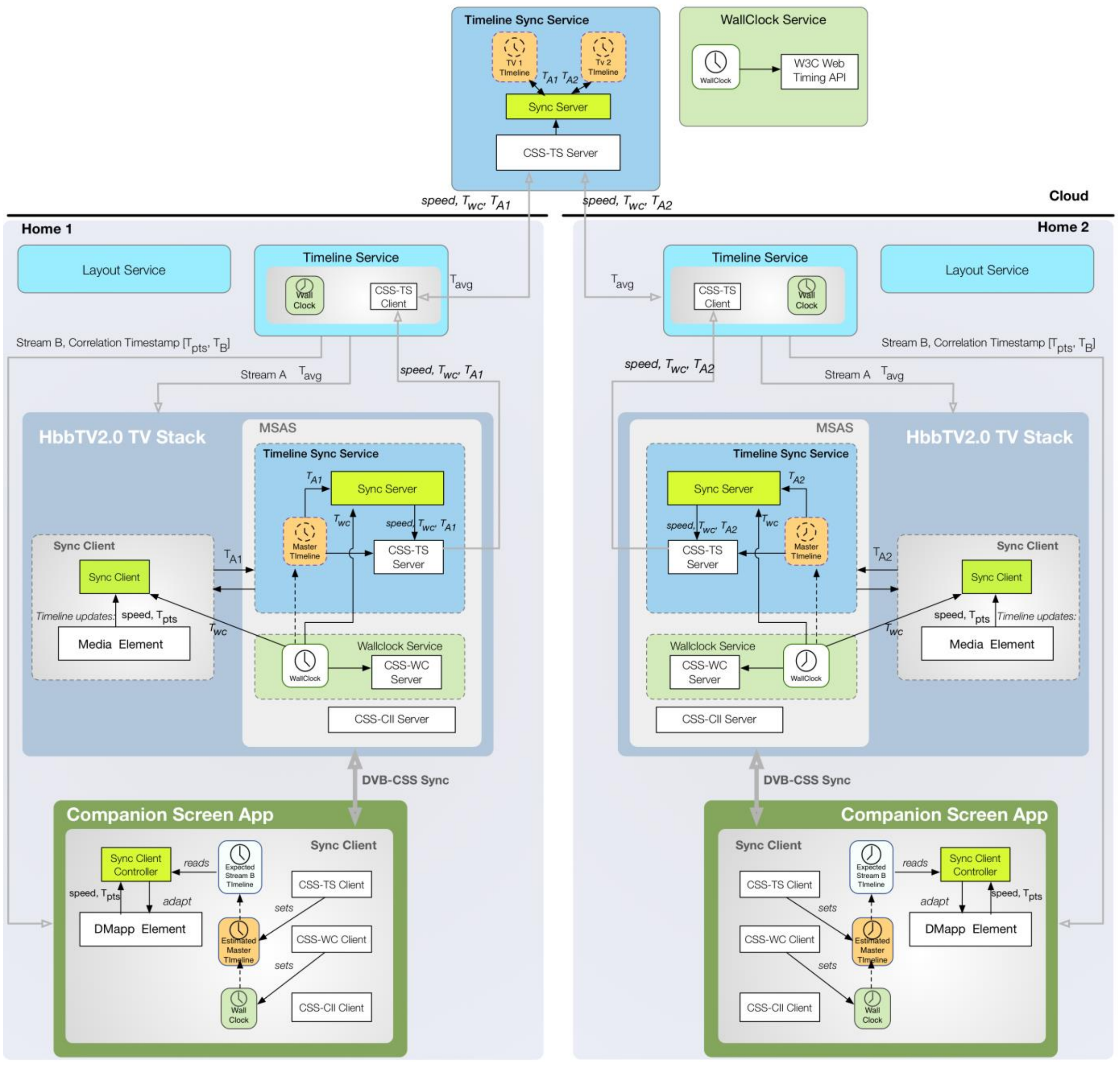


Figure 12: Concurrent intra-home sync experiences merged into an inter-home sync experience

Annex C Server-based Media Composition

C.1 Goal

"Deliver a superb presentation that doesn't require the broadcaster to author bespoke outputs."

C.2 Example compositions

Commonly used compositions are pre-generated using a production instance of the composition service and uploaded to a content delivery network. This reduces the amount of dynamic compositing performed on client devices.

C.2.1 Example #1

Description: Video wall and picture-in-picture overlay showing replays

Characteristics: Video tiling, cropping, scaling, overlay

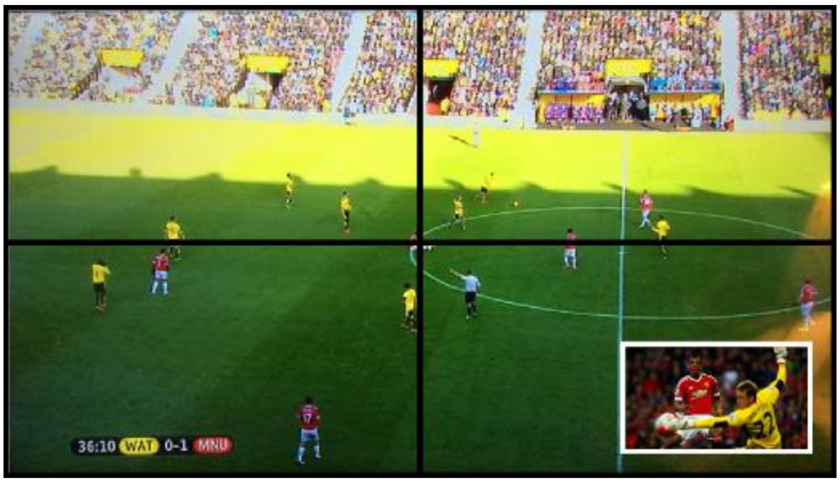


Figure 13: Example #1

C.2.2 Example #2

Description: Four alternative cameras on a tablet and info graphic overlay

Characteristics: Multiple feed aggregation, scaling, alpha-blended 2D overlay

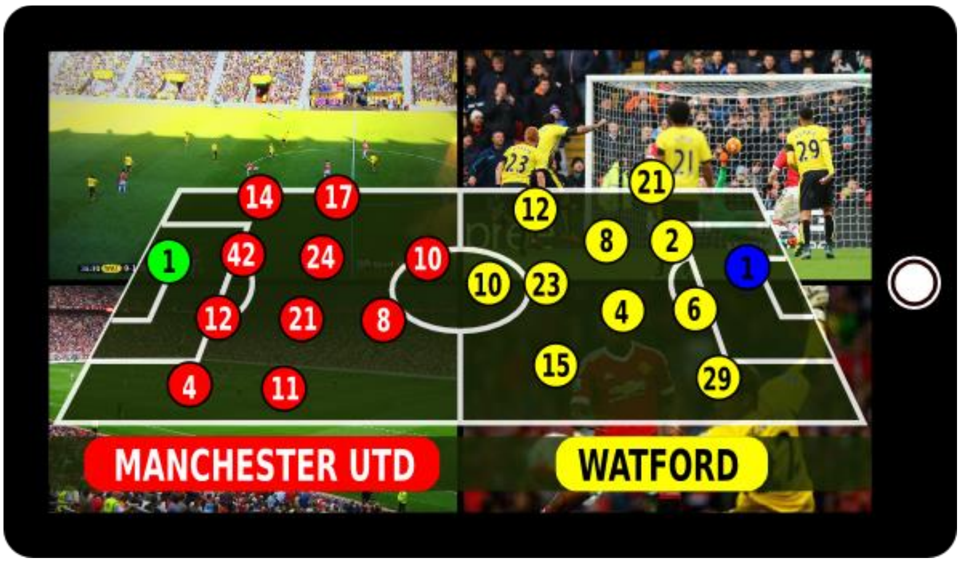


Figure 14: Example #2

C.2.3 Example #3

Description: Wide-angle camera with curated footage overlay and info graphics

Characteristics: Scaling, overlay, alpha-blended overlay

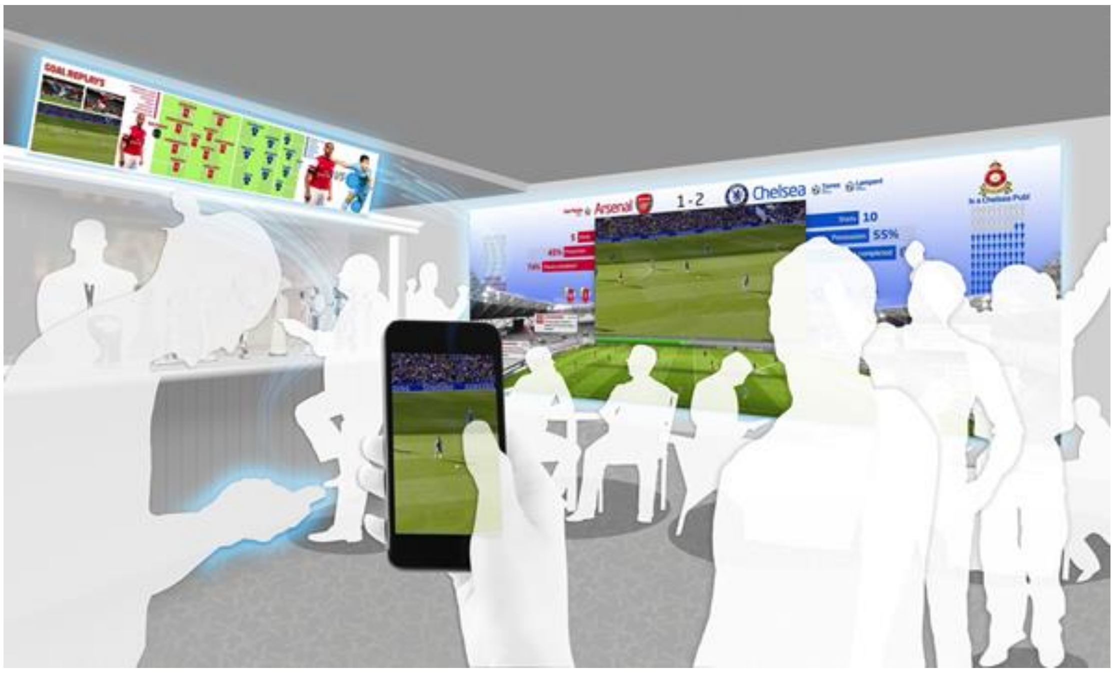


Figure 15: Example #3

C.2.4 Further examples

- Live reframing and compositing of preferred cameras angles

- Split screen presentation
- Stretching content across two or more screens
- Choice of commentary, choice of telemetry feeds and stats presented using overlays, picture-in-picture or on different screens.
- "My stats screen", "My replay screen", “My TMO" screen, "My wide-angle shot screen", "Chelsea stats screen", "MUFC screen" (representing a semantic abstraction over physical screens)
- Cross fading or overlaying adverts and other filler content during breaks in action
- Slow motion (bullet-time) action replays
- Compositing other compositions together

C.3 Functional description

The experience engine is responsible for combining timeline events, broadcast signals and layout information into low-level compositing instructions that are sent to the composition service via a media composition protocol. This protocol describes which input feeds to use, how to combine them over time and which output feeds to create.

The composition service will fetch feeds, decrypt, decode, synchronise, composite, post-process and present the result to a screen. Separately hosted composition services will also encode, encrypt and deliver the result over a network.

The composition service is capable of generating outputs that are tailor-made to the resolution, colour depth and bandwidth requirements of each client device, as described by the media composition protocol.

The experience engine will only generate compositing instructions that conform to usage rights previously negotiated with the media provider, such as the context in which a clean feed can be used and for how long. The composition service is responsible for encrypting the resulting composited streams before they are uploaded to a content delivery network or distributed via a real-time protocol. The service has a digital rights management (DRM) component to manage all the keys.

C.4 Functional Flow

The following list outlines the steps performed by the composition service in response to edit decisions and user interactions that affect layout and presentation.

1. Parse media composition protocol (MCP)
2. Determine which object-based media feeds to request
3. Receive, decrypt and decode each feed
4. Synchronise playback of each feed to the MCP timeline.
5. Perform compositing operations such as alpha blending, scaling and cropping.
6. Layout and render 2D/3D info-graphics.
7. Optionally re-encode and re-encrypt composited outputs for distribution.

The compositor has much in common with a Multipoint Control Unit (MCU) used in video conferencing apps (http://whatis.techtarget.com/definition/multipoint-control-unit-MCU) and is capable of aggregating multiple feeds together into a single presentation to generate the experience and assist with delivery.

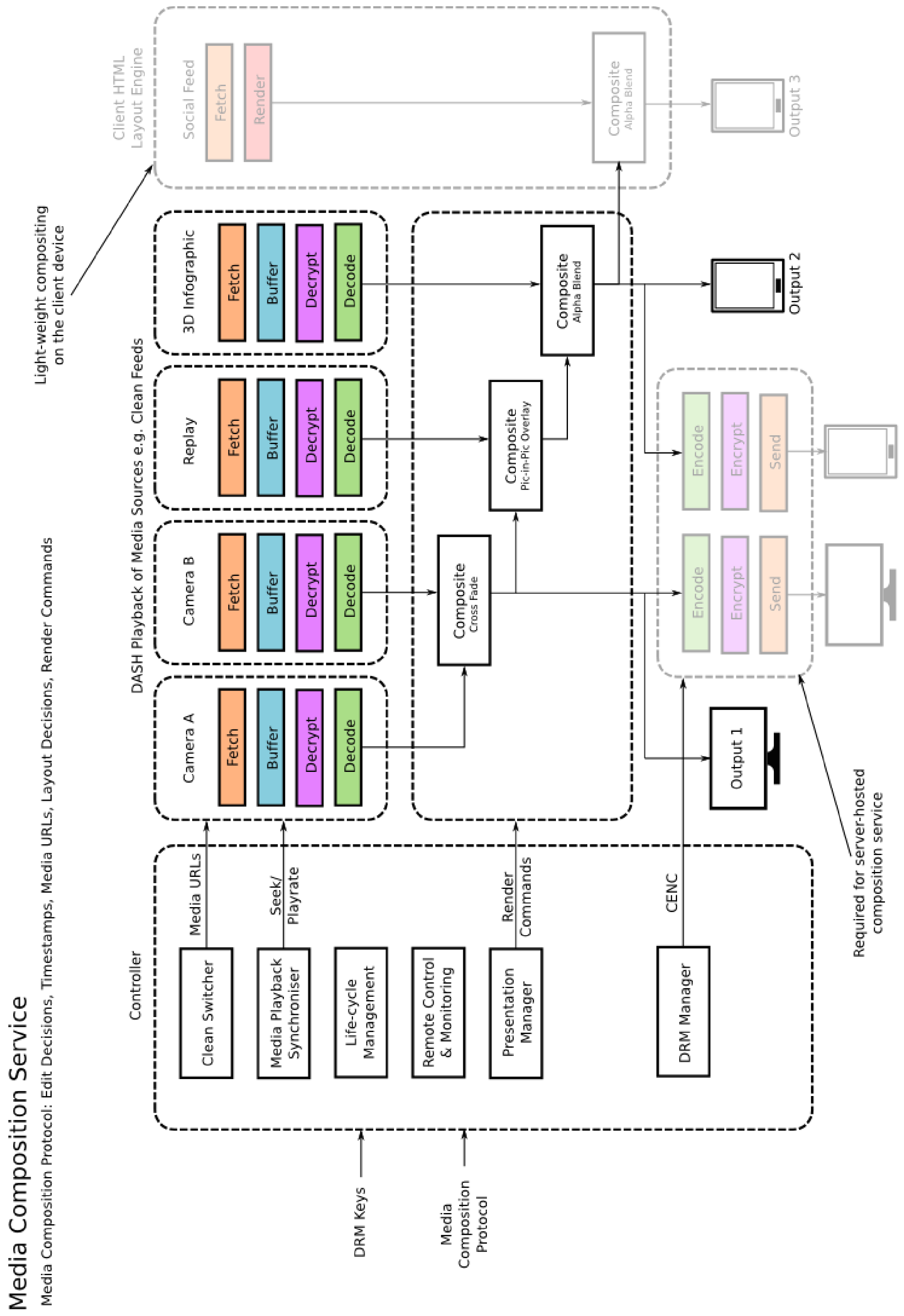

Media Composition Service
Media Composition Protocol: Edit Decisions, Timestamps, Media URLs, Layout Decisions, Render Commands
Light-weight compositing on the client device
Client HTML Layout Engine
Social Feed
Fetch
Render
Composite Alpha Blend
Output 3
DASH Playback of Media Sources e.g. Clean Feeds
Camera A
Camera B
Replay
3D Infographic
Fetch
Buffer
Decrypt
Decode
Composite Cross Fade
Composite Pic-in-Pic Overlay
Composite Alpha Blend
Output 2
Output 1
Encode
Encrypt
Send
Required for server-hosted composition service
Controller
Clean Switcher
Media Playback Synchroniser
Life-cycle Management
Remote Control & Monitoring
Presentation Manager
DRM Manager
Media URLs
Seek/ Playrate
Render Commands
CENC
DRM Keys
Media Composition Protocol

Figure 16: Media Composition Service

C.5 Composition Latency

The composition service is responsible for pre-emptively downloading audio and video assets in order to prime the decoding and compositing pipeline. It must attempt to minimise this warm-up latency in the interests of a responsive user experience. Network hosted composition services will incur greater latency due to media encode and delivery overhead.

Animated layout changes composited remotely will appear slick and smooth, but there may be considerable latency (in the order of 4 seconds) before the client can render them. Anything that involves a fundamental reconfiguration of the compositing pipeline as the result of a user interaction will incur significant latency. *[Note that one way of mitigating this is to utilise RTP for delivery of composited media feeds from the composition service.]*

Instantaneous feedback is required for most user interactions and this will be achieved by presenting the user with a spinner or progress bar until composited media feeds become available.

C.6 Compositing Devices

2-IMMERSE experiences will be presented on client devices with a large range of capabilities. Some devices are capable of compositing object-based media; some are incapable due to limited memory, battery life and compute resources. Compositing is the process of combining object-based broadcast feeds into a number of different presentations and this requires a powerful compositing solution that’s able to meet the demand of multiple UHD outputs.

HbbTV 2.0 mandates support for only one video/audio being decoded at time. A manufacturer can choose to go beyond that and support more than one, and the specifications are designed to accommodate that, but it is safe to assume that it is unlikely any will implement that in the near future.

The HbbTV specification describes how the TV must provide a browser environment and TV tuner with certain minimum capabilities. So, for example, picture-in-picture functionality provided as part of the TV's user interface does not necessarily imply that that dual-decoding capability is made available to HbbTV applications.

C.6.1 Local Versus Remote Compositing

A high-end PC or STB is required to run the composition service on the same network as companion screens and televisions. This device must be relatively powerful and will need to be installed in homes, schools and pubs.

Compositing in the cloud has the advantage of being readily scalable and upgradable. A paid-for subscription service built around the provision of object-based experiences driven by cloud-based compositing is a new source of revenue. During downtime between televised events, there is no ROI from a dedicated STB solution whereas with cloud hosting, costs are only incurred when the service is used.

As an example, Gaikai (https://www.gaikai.com/) has pushed PlayStation hardware into the cloud and uses a video streaming mechanism to deliver interactive game play straight to thin-clients on a per user basis. They have built a subscription model that allows users to play a PlayStation game within seconds of pressing a button, without downloading or installing anything and on a device that was never designed to do it.

C.6.2 Thin Clients

Both the local and remote compositors have the benefit that client devices can be thinner or simpler, therefore catering for a greater range of legacy devices and future devices. This also reduces the need for a framework like TAL (Television Application Layer). This is beneficial because TAL delivers a lowest common denominator feature set in order to abstract away differences in devices.

C.7 Interfaces

When hosted as a network service, the composition interface is accessed via a web server supporting REST end-points, cross origin resource sharing (CORS) and HTTPS. The resulting composited media is pulled via HTTPS or pushed via SRTP, RTMP or other real-time protocol.

For a composition service running on the client device, the interface is accessed provided by a JavaScript library in the web browser. The composited outputs are available as WebGL or Canvas contexts.

C.8 Selected Technology

Transport	HTTPS, SRTP or RTMP
Encryption/Decryption and licensing	Common Encryption (CENC) and OMA DRM 2.0 (as used by mp4box)
Decoding	libav, MSE and EME
Encoding	libav, FFMPEG, MP4BOX, DVB-DASH
File serving	nginx, CDN
Client-side composition	BBC R&D HTML5 Media Compositor (WebGL, DASH)
2D info graphic overlays	HTML5/CSS3
HW accelerated compositing	WebGL, OpenGL

Annex D Experience Deployment

Deployment involves moving software and content to a place where it can be used when it's needed. In the context of 2-IMMERSE experiences, this involves deploying:

- Media content and meta-data
- Web components
- Web applications/sites
- 2-IMMERSE native applications
- TV applications
- Manufacturer CSS launcher applications
- Provisioned services (locally and in the cloud)

The mechanics of deploying an application depends on the user journey taken to launch an experience and whether the application is launched from the companion screen or the TV.

D.1 2-IMMERSE Native Application Deployment

A native companion screen app (host to the web application) can be:

1. Manually downloaded and installed from an app store
2. Installed from a TV prompt over DIAL (with user confirmation)
3. Installed semi-automatically via a DevOps provider such as HockeyApp

The 2-IMMERSE native companion application must be installed in order to join the distributed media experience. During development and user testing a DevOps provider such as HockeyApp (http://hockeyapp.net/) will be used to deploy beta versions, collect live crash reports, get feedback from real users and analyse test coverage. Upon release, it is expected that the native application will be uploaded to native app stores.

Application versions will be promoted to a release repository branch. This will trigger the continuous integration servers to build and test the apps in the cloud on multiple devices, before being made available via HockeyApp (or similar).

Users will be prompted by DIAL on the TV or CSS device if the 2-IMMERSE native application needs to be installed.

D.2 HbbTV 2.0 TV/CSS Application Deployment

The 2-IMMERSE native application has an application manager. This application manager is equivalent to HbbTV 2.0's application manager in its responsibilities. The management of web components and presentation is done at a higher level within the web application itself in coordination with the UXE.

D.3 HTML Television Application Deployment

HTML TV applications can be deployed:

1. Automatically via the DVB signal (Broadcast Carousel Delivery)
2. Manually from an internet TV portal (TV app store)
3. Via a URL specified by a companion screen
4. Via a URL specified in the DVB signal (or DVB-DASH event)

D.4 Broadcast-Dependent Application Deployment

Broadcast-related autostart applications are usually associated with a broadcast channel and triggered by an event on that channel. Broadcast-dependent applications can start as soon as the channel is selected or on receipt of an Application Information Table (AIT) update (usually co-incident with the start of an event). The application can be launched immediately; as is generally the case with radio stations, or the user can be prompted to open the application (such as via the red button).

An AIT transmitted in the DVB signal can carry the URL of an HbbTV 2.0 application for IP delivery. Alternatively, the application can be transmitted within the DVB signal, a process called 'Broadcast Carousel Delivery'. In general, applications are loaded via the broadband connection from standard HTTP servers. As an alternative, AIT signal can be delivered by DVB-DASH events.

The TV will receive and analyse the AIT, using the information it contains to download the application using an IP or Broadcast Carousel delivery method, finally displaying it via the HbbTV 2.0 compliant web browser. A DVB multiplexer is required to create a DVB AV signal that also carries an AIT.

(See http://www.hbbtv-developer.com/site/wiki/index.php/Setting_up_a_HbbTV_environment)

The HbbTV Application Manager uses the AIT to control the lifecycle of an application whereas the browser is responsible for presenting and executing it.

(See section 5.3.3 of the HbbTV 2.0 specification: https://www.hbbtv.org/wp-content/uploads/2015/07/HbbTV_specification_2_0.pdf)

D.5 Broadcast-Independent Application Deployment

Broadcast-independent applications are started via a running application or an Internet TV Portal. An Internet TV Portal is an application that provides a type of start page where broadcast-independent applications are filtered and offered in an appropriate and useful way to the end user. The Internet TV Portal may be opened by pressing a dedicated Internet TV Button on the remote control unit.

For all 2-IMMERSE use cases, it is valid for a user or administrator to want to launch a broadcast independent HbbTV 2.0 application on an HbbTV 2.0 terminal from a companion screen application. In this instance, DIAL expects the HbbTV 2.0 terminal application to be pre-installed by the user from a TV app store. This requires HbbTV 2.0 applications to be deployed to the manufacturer's app stores. (See: https://www.fokus.fraunhofer.de/go/fokusmegastore)

(See: http://www.hbbtv-developer.com/site/wiki/index.php/Starting_HbbTV_Development)

D.6 HTML Companion Application Deployment

HTML Companion applications can be deployed via:

1. URLs from a HbbTV 2.0 master device
2. URLs from another companion device
3. URLs from the UXE (timeline & layout services)

Launching a companion screen application from an HbbTV 2.0 television requires the companion device to have the application pre-installed and a manufacturer specific HbbTV 2.0 CSS launcher service must be running on the companion to listen for launch requests. This is a complicated setup for users participating in the 2-IMMERSE public trials. To improve the user experience, it is better if the 2-IMMERSE experience is already running; therefore companion devices will only require application discovery and app-2-app communication. Users will still have to install and run the 2-IMMERSE native companion application in order to join the distributed media experience.

D.7 Content Deployment

2-IMMERSE web applications, components and media objects are deployed to a CDN. This makes it possible to push upgrades out to devices. Meta-data and media objects can also be distributed via DVB signalling in DVB T/S/C streams.

To configure a multi-device experience, a curated set of web applications and components must be loaded onto each device. To achieve this, devices must be running the 2-IMMERSE native application and have subscribed to events governing application life cycle.

D.8 Web Application Life Cycle

Features will be activated and deactivated during different phases of the experience. For example, in the case of the theatre, different functionality is required in the show versus the interval. Consequently, application components have a life cycle that is controlled automatically by the broadcast and by users as a result of enabling or disabling components, or because of the introduction of new devices to the environment.

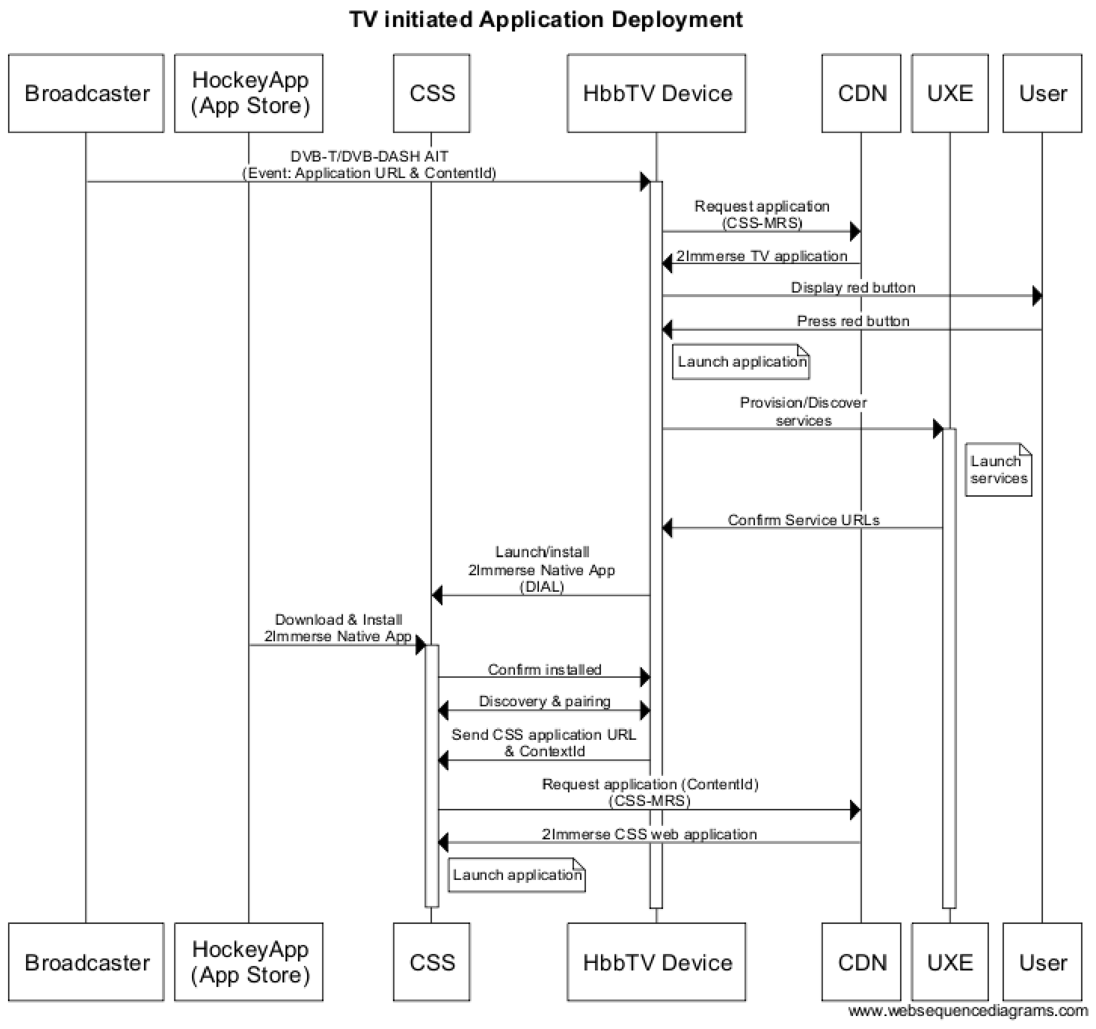
Application life cycle events include:

- Pre-emptive load (install)
- Activate/enable (run)
- De-active/disable (stop)

Life cycle events must be subscribed to by the 2-IMMERSE native application wrapper and used to manage the life cycle of the web application. Similarly, the web application must subscribe to events that control the life cycle of web components.

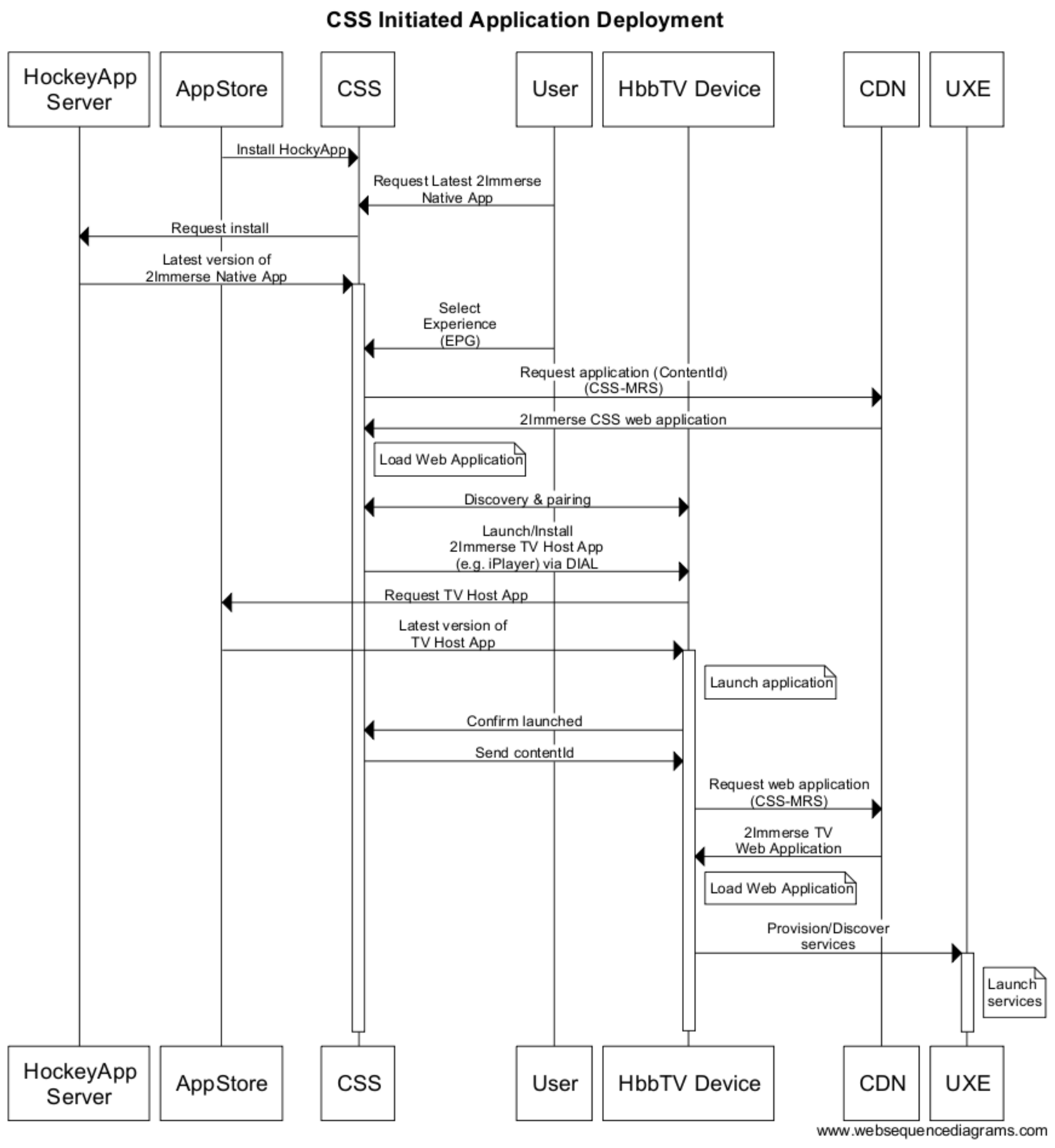
D.9 TV Versus CSS Initiated Deployment

The following sequence diagrams illustrate two possible application deployment scenarios out of many possible combinations.


TV initiated Application Deployment
Broadcaster
HockeyApp (App Store)
CSS
HbbTV Device
CDN
UXE
User
DVB-T/DVB-DASH AIT (Event: Application URL & ContentId)
Request application (CSS-MRS)
2Immerse TV application
Display red button
Press red button
Launch application
Provision/Discover services
Launch services
Confirm Service URLs
Launch/install 2Immerse Native App (DIAL)
Download & Install 2Immerse Native App
Confirm installed
Discovery & pairing
Send CSS application URL & ContextId
Request application (ContentId) (CSS-MRS)
2Immerse CSS web application
Launch application
www.websequencediagrams.com

A web application is firstly deployed to the HbbTV 2.0 master device. This web application communicates with the layout engine and therefore indirectly controls the deployment of web content (including functionality) to companion devices. It may also instigate the deployment/provisioning of cloud-based or local services

There is provision in the HbbTV 2.0 spec for routing events/triggers from the DVB-DASH or DVB-T stream to companion devices. Companion devices can also fetch the content directly or be instructed to fetch the content by the user experience engine.


CSS Initiated Application Deployment
HockeyApp Server
AppStore
CSS
User
HbbTV Device
CDN
UXE
Install HockyApp
Request Latest 2Immerse Native App
Request install
Latest version of 2Immerse Native App
Select Experience (EPG)
Request application (ContentId) (CSS-MRS)
2Immerse CSS web application
Load Web Application
Discovery & pairing
Launch/Install 2Immerse TV Host App (e.g. iPlayer) via DIAL
Request TV Host App
Latest version of TV Host App
Launch application
Confirm launched
Send contentId
Request web application (CSS-MRS)
2Immerse TV Web Application
Load Web Application
Provision/Discover services
Launch services
www.websequencediagrams.com

D.10 Upgrades

A 2-IMMERSE device ecosystem may cache web applications and web components, preventing the latest versions from running. Layers in the application stack must subscribe to upgrade events to ensure the last version of the software is running.

Annex E Testing and Validation

Testing and validating the correctness of a large distributed system as a unit presents a significant challenge due to complexity, hidden state and a large surface area to test. To make testing more manageable, individual features, modules and network nodes should first be tested individually, and then the overall system tested as a distributed collection of nodes.

The items of content/user experience applications that run on the system also need to be tested to ensure that they have been implemented correctly and that there are not any unexpected interactions with the wider system.

E.1 System testing

Testing the system itself (rather than the content/user experiences which run on it) might include:

- Unit/module-level testing
- Web page/UI validation (e.g. ensuring that given a test input all expected elements are present and physically on screen, for a range of device types).
- User-input simulation testing
- Media/video rendering testing (e.g. checking that the result of a media rendering operation of fixed test input is pixel identical to a pre-prepared expected test output).
- Testing of distributed join/leave/discovery semantics.
- Testing for graceful handling of unexpected conditions and failure paths.
- Testing for potentially problematic distributed interactions even if they don’t result in a failure or incorrect output (e.g. ordering issues).

Test coverage can be measured by automated tools; this can be useful for identifying areas which need further testing or are unused. Metrics which may be useful might include:

- Line coverage
- Branch coverage
- API interface coverage

Particularly for branch and line coverage, 100% coverage is not usually necessary due to diminishing returns.

Depending on language and platform it may be possible to run tests and/or build the test program with increased instrumentation to detect potentially problematic conditions which may otherwise pass unnoticed. This is particularly the case if any parts of the system are implemented in native code (e.g. LLVM/gcc sanitisers).

Frameworks and tooling for testing varies depending on language and platform. There are a large number of JavaScript testing libraries to choose from.

To aid testing and debugging of the system, it may be useful to include API endpoints specifically for the purpose of observing otherwise hidden state during testing or for directly initialising a test environment where the state would not otherwise be directly controllable. Generally, such test interfaces should not be present in deployed builds.

It may also be useful to include invariant testing or other code-level assertions too expensive for deployed builds, in test builds.

It may be useful in some test cases for logging output (possibly including test-build only debug logging) to be directed to the test harness and used as part of the output under test; a possible case is where intermediary state which is useful to test would otherwise have been lost.

API endpoints which are accessible to the public Internet should as much as possible be resistant to attempts to subvert the system by supplying invalid input. It may be useful to run tools such as fuzzers, possibly in combination with build instrumentation as described above, on such API endpoints to check for unexpected failure modes or output types which may indicate a security issue.

Similarly, any API endpoints which handle authentication should be tested to ensure that invalid credentials are properly rejected without leaking undue information by side channels (e.g. partial success or timing).

E.2 Content/user experience application testing

Assuming that a content/user experience application is purely declarative, a subset of testing can be performed statically. Any format used to describe the application should have a schema defined (whether formal or informal) which can be used to automatically validate the format of the application. In the case where the format defines layouts, transitions and so on, it should be possible to validate that all coordinate, sizes, etc. lie within pre-determined bounds, and that all URLs and other references refer to valid, existing resources. Non-static testing would include running the application on a test-instance of the system, or a subset thereof, and validating that for a representative combination of states and device types, the outputs are as expected, or at least in an expected range, and that no unexpected failure conditions occur.