Directorate General for Communications Networks, Content and Technology

Innovation Action

ICT-687655

D2.5 - Distributed Media Application Platform and Multi-Screen Experience Components: Description of Final Release

Due date of deliverable: 30 September 2018

Actual submission date: December 2018

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

Lead contractor for this deliverable: **Cisco**

Version: Final

Confidentiality status: **Public**

Abstract

This document describes the final release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's four service prototypes. At the end of the project, this release represents the most mature and robust implementation of the platform, components and tools with the functionality required to deliver all of the service prototypes, and in particular the three which have been completed during the final year of the project: *Football Fanzone*, *Football at Home* and *Theatre in Schools*.

Target audience

This is a public deliverable and could be read by anyone with an interest in the details of the platform, service prototypes and production tools 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.

Impressum

Full project title: 2-IMMERSE

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.5 Distributed Media Application Platform and Multi-Screen Experience Components: Description of Final Release

Editor: Ian Kegel (BT)

Workpackage Leader: James Walker, Cisco

Technical Project Leader: Mark Lomas, BBC

Project Co-ordinator: Matthew Scarth, BBC

This project is co-funded by the European Union through the Horizon 2020 programme.

Executive Summary

The final release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools remains a practical implementation of the system architecture defined in project deliverable D2.1 (1), and the platform component interfaces defined in project deliverable D2.2 (1).

It represents the most mature and robust implementation of the platform, components and tools with the functionality required to deliver all of the service prototypes, and in particular the three which have been completed during the final year of the project: *Football Fanzone*, *Football at Home* and *Theatre in Schools*.

The key features and highlights of the final release are:

- Development of the Football and Theatre in Schools DMApp implementations, with their corresponding requirements for new DMApp Components and updates to timeline, layout and Client API features. These updates include:
    - Timeline service support for live object-based production
    - More precise control of component priorities and constraints within the Layout service
    - More efficient Client API interfaces and improved input documents for configuration
- The Unified Launcher, a single all-purpose onboarding implementation that supports the diverse requirements of all the 2-IMMERSE service prototypes, including managing multiple device roles and the ability to launch an experience on multiple communal devices. This comprises:
    - A new Android Unified Launcher host application
    - An Android Unified Launcher configuration application (for development)
    - New TV Emulator firmware
    - A new Unified Launcher web application
- A complete inventory of the 37 DMApp Components which have been developed during the course of the project, some providing core functionality which has been re-used across multiple DMApps, others providing specific functionality which meet the more precise requirements of a genre, but which could still be re-used within that genre. Newly-developed components include:
    - *Content Browser*, which enables the creation of more complex user interfaces with reduced developer effort requirements. It supports templates for reusing components across different UI views and is used extensively in the Theatre in Schools DMApp.
    - *Prime*, a generic graphics overlay component used for realising any kind of non-interactive on-screen graphics. Its scene representation parser is compatible with ChyronHego Prime broadcast graphics authoring software, which means 2-IMMERSE graphics can re-use existing broadcast graphics or vice versa.
- A standalone implementation of the 2-IMMERSE platform, which enables the 2-IMMERSE service prototypes to be demonstrated without requiring a connection to the cloud-hosted Rancher-managed instances of the platform.
- The evolution of the production toolset to provide three distinct tools:
    - For live production:
        - A *triggering tool* to prepare events before they are shown
        - A *trigger launcher* to allow exact control over when an event is shown
    - A new *pre-production tool* which is intended to help create the initial storyline of an experience, assembling all media, DMApp components and story elements that are available before the live event happens.
- A description of work being carried out on platform sustainability to enable the 2-IMMERSE platform and the 2-IMMERSE open-source software release to be more easily set up and used by project partners and third parties after the end of the project.

- A description of three targeted platform evaluations carried out to measure the effectiveness of core functionality of the platform which it was not possible to test within the scope of the service prototypes. These evaluations were focused on:
  - Using real HbbTV2.0 devices
  - Cloud-based synchronisation accuracy
  - Network bandwidth orchestration effectiveness

At the end of the 2-IMMERSE project, the Final Release demonstrates that the project has delivered a fully-featured, extensible platform to support the end-to-end production, delivery and consumption of multi-screen experiences.

List of Authors

Mark Lomas – BBC

Rajiv Ramdhany - BBC

Ian Kegel – BT (editor)

Jonathan Rennison - BT

Stefan Fjellsten - ChyronHego

Tal Maoz – Cisco

Aviva Vaknin - Cisco

Jack Jansen – CWI

Thomas Röggla - CWI

Michael Probst – IRT

Florian Bachmann - IRT

Reviewers

Tal Maoz – Cisco

Aviva Vaknin - Cisco

Table of contents

Glossary of terms

Term/acronym	***Definition/explanation***
Experience	2-IMMERSE is developing, using its own platform, four innovative service prototypes of multi-screen entertainment 'experiences'. Unlike existing services, the content layout and compositions are orchestrated across the available screens and an object based broadcasting approach is used for efficient, dynamic, high quality, synchronized content distribution and rendering.
Distributed Media Application (DMApp)	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.
Distributed Media Application (DMApp) Component	In 2-IMMERSE, re-usable components are assembled within a Distributed Media Application (DMApp) to create coherent multi-screen experiences.
CI/CD	Continuous Integration and Continuous Delivery
Context	2-IMMERSE defines a 'context' as 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 (e.g. a home network). Devices belonging to the same context must be able to discover each other using the DIAL protocol. Devices can join or leave a context at any time.
IPTV - Internet Protocol television	Internet Protocol television (IPTV) is a major concept within the 2-IMMERSE platform as it allows the delivery of next generation experience levels using an object-based broadcasting (OBB) approach to content delivery. However, we made sure that our client stack is compatible with traditional linear broadcast using the HbbTV 2 standard.

1 Introduction

This document describes the final release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's four service prototypes. At the end of the project, this release represents the most mature and robust implementation of the platform, components and tools with the functionality required to deliver all of the service prototypes, and in particular the three which have been completed during the final year of the project: *Football Fanzone*, *Football at Home* and *Theatre in Schools*.

The description of the final release was originally defined as two separate written reports: D2.5 (Distributed Media Application Platform: Description of Final Release), and D5.3 (Multi-Screen Experience Components: Description of Final Release). To make the content easier to read and navigate, the complete description is now provided in this document, D2.5. Deliverable D5.3 now contains a series of videos which show the 2-IMMERSE Platform, Components and Production Tools in action in the context of the service prototypes completed during the final year.

The final release remains a practical implementation of the system architecture defined in project deliverable D2.1 (2), and the platform component interfaces defined in project deliverable D2.2 (1). While significant improvements have been made to some platform services, the development focus has largely been on the client middleware, applications and production tools to support the requirements of the two Football service prototypes (which share many features) and the Theatre in Schools service prototype. These have also necessitated the development of several new Multi-screen Experience Components.

The deliverable is structured as follows:

- **Introduction** - introduces the final release of the Distributed Media Application Platform, and explains how the rest of the deliverable is structured.
- **Requirements and Development Milestones** – A summary of the additional technical requirements for the platform, DMApp and components arising from the three new service prototypes, and the technical development milestones undertaken to deliver them.
- **Platform Infrastructure** – describes updates to the infrastructure deployed to support the 2-IMMERSE service platform for the final release.
- **Platform Services** – describes updates to the core platform services developed and deployed for the final release.
- **Client Application** – describes updates to the client application stacks developed and deployed for the Football and Theatre in Schools service prototypes.
- **Multi-Screen Experience Components** – describes all of the Multi-Screen Experience Components that have been developed within the 2-IMMERSE project, including those required for the Football and Theatre in Schools service prototypes.
- **Production Tools** – describes updates to the Production Tools developed to support the authoring of multi-screen experiences and the live end-to-end testing of the *Football at Home* service prototype in particular.
- **Platform Sustainability** – describes updates being made to the platform and client applications to support their sustained use after the end of the project.
- **Platform Evaluation** – describes the technical evaluation of important platform and client application functionality which was beyond the scope of the trials of the service prototypes.

2 Requirements and Development Milestones

The purpose of this section is to summarise how the technical development of the Final Release of the 2-IMMERSE platform and multi-screen experience components was managed, in terms of defining requirements and planning milestones appropriate to the service prototypes and the nature of the trials and tests undertaken to evaluate them.

As was demonstrated within deliverable D2.4 (3), which described the second release of the 2-IMMERSE platform and multi-screen experience components, the MotoGP service prototype drove many significant technical requirements to enable the platform to support the delivery of an 'as-live' sports experience within a home environment. A different approach was applied for the development of the two Football service prototypes because the first objective was to achieve a live end-to-end test involving a representative production environment. The opportunity to work closely with BT Sport and their broadcast production suppliers at three live events in Wembley Stadium necessitated a more agile approach in which requirements and development milestones for both client and production infrastructure were adapted as the team learned more about the production environment and the equipment and facilities available. Deliverable D4.6 (4) describes the live testing activities in more detail. Subsequently, a fully-featured 'as-live' Football at Home service prototype was developed based on the version completed for the live tests. At the same time, functionality was added to the client software and some platform services to enable the Football Fanzone experience to be demonstrated across multiple communal screens.

2.1 Football at Home interaction requirements

Deliverable D4.4 (5) defined the high-level user interaction requirements for the Football at Home service prototype, which can be summarised as follows:

1. **A 'TV experience'**: Build on the existing match director's production feed as the centre of the experience on the main TV, while providing a suite of enhanced features across all screens with which viewers can (as a group, or individually) assemble different ways of watching the same match.
2. **Live Additional Views:** A variety of ISO video feeds are available to the viewer to supplement the main match feed on the TV. These video feeds can be displayed, picture-in-picture, on the main TV or tablet, or as a full screen or part of a split screen on any of the companion screens.
3. **Alternative Commentary:** One or more alternative audio commentary feeds can be selected by the viewer, and switched at any time during the match. Each of these commentary audio options is still combined with the broadcast video feed provided by the match director.
4. **Object-based interactive graphics:** The match director and the graphics team are still responsible for triggering graphics, but the decision to render them (or not) and the decision about to which screens they should be rendered is influenced by the viewer's current multi-screen setup. For those viewers wishing to engage further with the information, the graphic is also interactive, allowing it to be expanded to provide greater depth of information on the current playing XI and the other substitutes available to the manager.
5. **Virtual Graphics:** A combination of player tracking data and object based virtual graphics rendered on the companion app allows viewers to personalise a live view of the game with virtual graphics that highlight and track the specific elements of the game that interest the viewer.
6. **In-Game Replays:** Combining the features of interactive match graphics, flexible screen layout and a server-based replay service allows viewers to access on-demand replays across a multi-screen system.

2.2 Football at Home live testing development milestones

Delivery of the technical developments required for a live end-to-end object-based broadcasting test during the FA Cup Final at Wembley in May 2018 was managed through a sequence of technical development milestones. As shown in Table 1 below, these were classified as 'Client' or 'Production' milestones and were addressed in parallel as far as possible. The table also shows how prioritisation and the availability of broadcast resources meant that some milestones were de-scoped or postponed to the 'as-live' version.

	Client Milestone Description	**Production Milestone Description**	**Date signed off**
C1	**DMApp Foundation** First version of Football DMApp uses VOD content to show synchronised match playback across TV and multiple companions, reusing onboarding process from MotoGP.		16/3/2018
C2	**Discrete Production GFX Rendering Component – first version** New DMApp Component developed and tested which renders arbitrary GFX packages developed using the ChyronHego Prime authoring tool. Component capabilities will be limited to those required for Football trial, rather than full Prime feature set.		22/4/2018
C3	**Enable dynamic Production GFX** Animations are parameterised so that we can insert arbitrary team names, colours, logos and player shots.		12/5/2018
P4		**Updated design for Live Triggering tool** New button-based UI design developed by Martin in conjunction with Jie and Tom.	11/4/2018
P5		**Live Triggering Tool for Football Production GFX** Updated tool enables triggering of Production GFX live within DMApp.	22/4/2018
P6		**Live video uplink and DASH encoding solution** Solution for contributing synchronised live camera feeds (clean feed plus ISOs and additional cameras) out of match location to cloud-based distribution. Includes live DASH encoding – and appropriate content protection.	22/4/2018
P7		**Synchronisation architecture for live football** Sync architecture for live video, audio and data confirmed. Includes handling of	22/4/2018

	Client Milestone Description	Production Milestone Description	Date signed off
		correlation timestamps/MPEG TEMI (if used)/synchronisation in client. Agreement from BT TSO broadcast team that it will work.	
P8		**Synchronisation Service Deployed** Sync service components deployed, running and tested within 2-IMMERSE Test environment.	De-scoped
P9		**Integrate Sync Client** Sync client Javascript library integrated with Client API. Relevant Sync client functionality implemented within Timeline Service.	De-scoped
P10		**Integrate EVS feeds** Obtain access to EVS feeds from BT Sport and integrate with DMApp, enabling access to multi-angle match replays. Understand content protection and DASH encoding issues.	12/3/2018
P12		**Production GFX Rehearsals** Using an audio/video recording of the match director during a live match, rehearse using the Live Triggering tool to trigger production GFX.	11/5/2018
P13		**Test Commentator Recording** Carry out test audio/video recording of one or more commentators to confirm angles and acceptability.	Not available
P14		**Live player tracking test** Test getting Tracab player tracking data into the DMApp via Data Playback Service.	19/5/2018
P15		**Live camera tracking test** Test getting the (synchronised) camera parameters into the Data Playback Service.	19/5/2018
P16		**Player cam streams available** Player cams automatically captured by Nikon robotic cameras and made available as streams. Review quality of virtual player camera outputs and decide on requirement for additional fixed cameras.	Not available

	Client Milestone Description	Production Milestone Description	Date signed off
C17	**Add ScoreClock Menu Component** The ScoreClock Menu is toggled by tapping on the Score Clock on each companion device. It presents four menu options: 1. Match Overview (key events) – the default 2. Stats 3. Lineups 4. Replays (more detail) The content of all four are driven by live data from the Data Playback Service.		6/4/2018
C18	**Add Broadcast Menu Component** The Broadcast Menu is toggled by tapping on the BT Sport bug on each companion device. It presents four menu options: 1. Cameras (preview/TV?) 2. Commentary – option to have local audio on companion; sync with commentary cam selection 3. Screens/Templates (companion?) – personalised to team 4. Virtual – different camera, perhaps different UI		12/5/2018
P19		**Optional elements in Timeline document** Change to Timeline document structure to enable optional elements – eg. different graphics triggers in multiple commentaries.	De-scoped
P20		**Add lightweight triggering UI** Lightweight UI is available for commentators to trigger a subset of production GFX (eg. for in-match promos).	De-scoped
C22	**Enable optional interactive broadcast graphics on companion** When this option is enabled for a particular companion, production graphics shown on the TV are mirrored on the companion. If the graphic is tapped while being displayed, the ScoreClock Menu is shown with contextually-relevant information.		18/5/2018

	Client Milestone Description	Production Milestone Description	Date signed off
C23	**Enable reconfiguration of TV and Companion layout** Once the Broadcast Menu (or the ScoreClock Menu for a replay) has been used to present Picture-in-Picture content on a companion device, the position of the PiP on the screen can be changed interactively between a number of fixed options. It is also possible to configure PiP location on the TV.		19/5/2018
C24	**Virtual GFX rendering on companion** Virtual GFX can be displayed on any companion when selected from the Broadcast Menu.		De-scoped
C26	**Add notifications and prompts** Pop-up notifications and prompts are displayed on the TV and companion devices when specified within the live triggering tool. When tapped on the companion, the relevant menu is shown with contextually-relevant information.		Postponed
C27	**Add pre-match experience** This will include the dirty feed, with companion interaction limited to VOD and stats.		Postponed
C28	**Add post-match experience** This will include the dirty feed, with companion interaction limited to replays, but with no Broadcast Menu.		Postponed
C31	**Support for fan-out from editable timeline to multiple contexts** Live DMApp can be viewed on multiple clients in addition to the Trigger Tool preview. Propagation of common start-time into clients. Restart / late joining enables fast-forwarding		19/5/2018

Table 1: Football at Home live testing development milestones

2.3 Football at Home and Football Fanzone ‘as-live’ development milestones

Delivery of the necessary updates to the client and platform software to create ‘as-live’ service prototypes for Football at Home and Football Fanzone was managed through a further set of technical development milestones which were agreed following the final live end-to-end test.

	Milestone Description	Date signed off
1	**Unified multi-DMApp Android APK** A single APK is used to launch different DMApps. It may also have a configuration app which enables it to be switched between the Edge, Test and Production environments.	30/8/2018
2	**Updated Match (ScoreClock) Menu Component** The updated Menu is toggled by tapping on the Score Clock on each companion device. It presents five menu options: 1. Match Overview (Data) 2. Match Stats (Data) 3. Home Team Lineup (Data) 4. Away Team Lineup (Data) 5. Replays (Video Clips)	1/8/2018
3	**Updated Broadcast Menu Component** The Broadcast Menu is toggled by tapping on the BT Sport bug on each companion device. It presents four menu options: 1. Live Cameras (Video Streams) 2. Multi-Screen Templates (Layout) 3. Custom Popups (Non Match Data) 4. Audio component (mix between commentary and non-commentary version)	1/8/2018
4	**TV Control Component** The TV Control component at the top of the screen provides the following functions on the tablet companion device only. 1. View current TV content 2. Show/Hide TV PiPs 3. Remove TV PiPs 4. Remove TV Replays 5. Scale PiPs between small and large (independently)	1/8/2018
5	**Fanzone Manager Component** The Fanzone Manager component enables the Fanzone experience to be configured and controlled.	7/9/2018
6	**Updated Match GFX** An agreed set of Match GFX are completed 1. A limited set of new GFX authored in PRIME 2. Existing GFX updated so they ALL include animations	28/8/2018
6b	**Chyron Prime component optimisation** 1. GFX components all support scaling across different screen sizes 2. Performance optimisation (inc. Match GFX run in OpenGL on companion devices)	28/8/2018
8	**Enable reconfiguration of TV and Companion layout** Different video layouts can be presented on TV and companion devices, specifically: 1. Full Screen (1 video) Layout on any of the 3 screens (TV, tablet and Phone) 2. Full Screen + PiP layout (3 videos) on TV and Tablet 3. Dual Screen layout (2 videos) on TV and Tablet Also, a single timeline event (Name Super Yellow) can be rendered in different ways based on video stream content (context) and layout (position).	21/8/2018
9	**Add preset TV and Companion layout configurations** These include pre-configured definitions of 'Chelsea mode' and 'Man Utd mode'. Once these layouts are invoked, they can be adjusted using the existing interactive controls.	20/7/2018

	Milestone Description	Date signed off
10	**Updated entire match timeline to include additional supporting GFX** Timeline for Football at Home experience includes GFX pop-ups for non-match alerts for goals, BBC news alerts etc, possibly inserted using (offline) triggering tool.	7/9/2018
11	**Highlight reel timeline with navigation** Highlight reel (short) timeline which enables navigation around the timeline to key content points to best demo features.	7/9/2018
12	**Onboarding process for Football Fanzone experience** The onboarding process enables multiple additional communal devices (TVs) to join a context, and for devices to be designated as 'primary' or 'auxiliary'.	30/8/2018
13	**Updated Layout and Timeline for Football Fanzone experience** New timeline and layout documents for Fanzone demo created and tested with new Layout Service features, plus Fanzone manager component.	7/9/2018
14	**Add pre-match experience** This will include the dirty feed, with companion interaction limited to VOD and stats.	28/8/2018
15	**Add post-match experience** This will include the dirty feed, with companion interaction limited to replays, but with no Broadcast Menu.	28/8/2018
16	**Experience working on phone** To include different GFX scaling and different menu handling	7/9/2018

Table 2: Football at Home and Football Fanzone 'as-live' development milestones

2.4 Theatre in Schools requirements and development milestones

The Theatre in Schools service prototype placed very different requirements on the 2-IMMERSE platform, namely to demonstrate how digital content derived from a filmed theatre play can be orchestrated on multiple screens in a classroom environment in order to improve learning outcomes for children. The need for close engagement with the content partner, Donmar Warehouse, in the iterative development of the experience once again meant that requirements and milestones were adapted as development progressed. The high-level interaction requirements at the beginning of the development period were identified as follows:

1. **Bookmarking and replay**
    a. Investigating scenes
    b. Creating playlists
    c. Annotating video
2. **Story and sequencing**
    a. Story exploration
    b. Story fragments
    c. Creating sequences
3. **Feedback and responses**
    a. Synchronised responses
    b. Live interaction
    c. Visual overlays

4. **Performing and remixing**
    a. Creative expression
    b. Manipulating audio
    c. Remixing content

The majority of technical features for Theatre in Schools were realised within the client application, with only a small amount of orchestration by the platform services, which meant that a smaller development team was needed, and a simpler list of development milestones (listed in Table 3 below). A template-based approach enabled the significant amounts of re-use between lesson applications. The detailed design of the Theatre in Schools service prototype is described in deliverable D3.5 (6). It should be noted that the third lesson application which was proposed for Theatre in Schools (Profile Creator) was not developed.

	Milestone Description	**Date signed off**
1	**Content Browser DMApp component** First version of Content Browser DMApp component, including tech sample based on one UX element of the Storyboard Builder Make or Share exercises.	6/8/2018
2	**Bookmarked Timeline DMApp Component** First version of Bookmarked Timeline component, enabling interactive bookmarks to be associated with a video player and used to start video playback from specific timecodes.	12/7/2018
3	**Foundation DMApp for Storyboard Builder** First version of Storyboard Builder DMApp, only implementing the Watch and Make exercises.	7/9/2018
4	**Countdown Timer GFX DMApp Component** A component which enables a countdown timer to be configured and displayed on both TV and tablets. **Milestone deprecated as functionality was included within Content Browser component.**	(1/10/2018)
5	**Complete DMApp for Storyboard Builder** Completed version of Storyboard Builder DMApp, including all exercises and transitions between them.	1/10/2018
6	**Emotion GFX DMApp Component** Component which enables emotion GFX to be overlaid on video during Script Detective Share phase.	8/11/2018
7	**Word Cloud GFX DMApp Component** Component which displays a word cloud overlaid on video.	De-scoped
8	**Foundation DMApp for Script Detective** First version of Device Detector DMApp, only implementing the Make and Share exercises.	25/10/2018
9	**Complete DMApp for Script Detective** Completed version of Script Detective DMApp, including all exercises and transitions between them.	8/11/2018
10	**Character Relationship GFX DMApp Component** Component which displays character relationships overlaid on video.	De-scoped

	Milestone Description	Date signed off
11	**Foundation DMApp for Profile Creator** First version of Profile Creator DMApp, only implementing the Make and Share exercises.	De-scoped
12	**Simple Setup web app** Simple app which enables a teacher to customise a lesson, for example changing content references and timecodes.	De-scoped
13	**Complete DMApp for Profile Creator** Completed version of Profile Creator DMApp, including all exercises and transitions between them.	De-scoped

Table 3: Theatre in Schools development milestones

3 Platform Infrastructure

This section of the deliverable summarises updates made to the infrastructure deployed to support the Final Release of the 2-IMMERSE service platform.

3.1 Rancher Platform

The following improvements and additions were made to the Rancher platform this year:

- load balancer updates
- added Tyk and live-origin gateways
- added cloud sync session controller and NTQ broker services
- service upgrades
- updated cloud-sync microservices
- added timeline observer
- Rancher Catalog version update
- general improvements of the load balancer configuration scripts

Rancher was reliable, fairly easy to navigate, and provided good utilities to set up and view what is happening in the services stacks.

3.2 Origin Server

The Tyk gateway was added in front of the live origin server; the gateway uses the Auth service to validate users before allowing access to the origin server.

3.3 CI/CD and Docker Registry

We added some hooks in Rancher to trigger image upgrades, and added rules into Gitlab to call the hooks once a new version is built so Rancher automatically deploys onto the Edge stack.

Otherwise the CI/CD set in place worked well.

3.4 Standalone Implementation

In the final year of the project, it was desirable to demonstrate the 2-IMMERSE service prototypes without requiring a connection to the cloud-hosted Rancher-managed instances of the platform. The approach here has been to have a standalone host that emulates the platform, with minimal changes to the NUC/emulator that connects to it. The technical setup is shown in Figure 1 below.

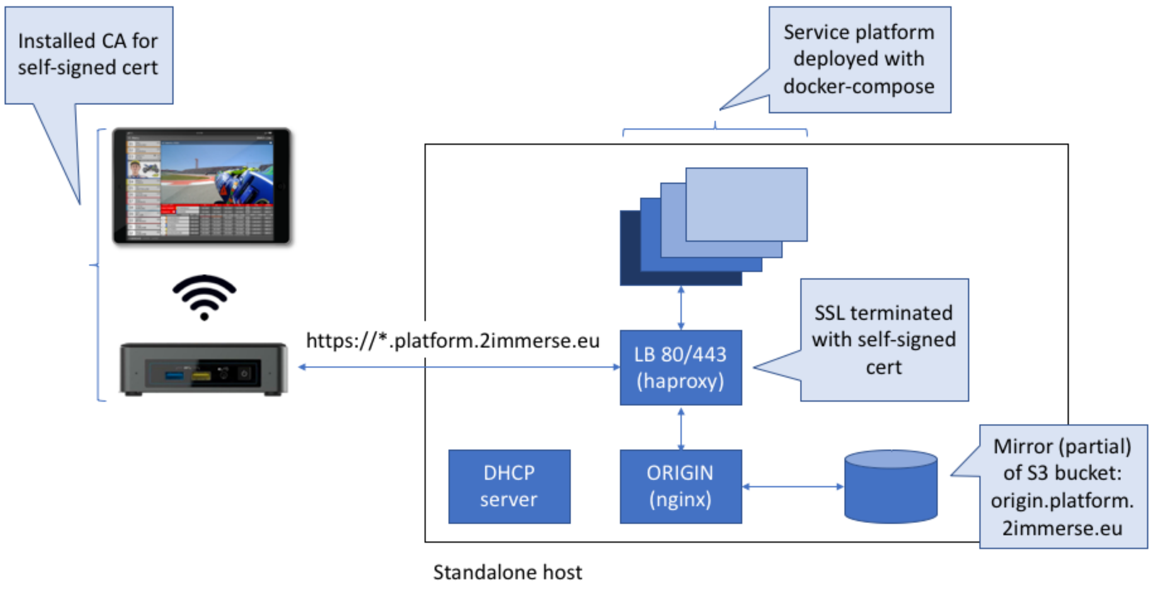

Installed CA for self-signed cert
Service platform deployed with docker-compose
https://*.platform.2immerse.eu
LB 80/443 (haproxy)
SSL terminated with self-signed cert
DHCP server
ORIGIN (nginx)
Mirror (partial) of S3 bucket: origin.platform.2immerse.eu
Standalone host

Figure 1: Standalone implementation of the 2-IMMERSE platform

We use a self-signed certificate since it is not good practise to extract the certificates from the platform and use them in local versions (indeed if they were compromised they could be revoked by the Certification Authority). However, by using the same domain we avoid the complexity of having to change any hard-coded domain URLs in the DMApp documents and components.

We use *docker-compose* to deploy the standalone service stack. We defined a *docker-compose yaml* script with the rules for deploying each of the services. It includes image versions to deploy as well as dependency rules so that the services are deployed, and brought down, in dependency sequence. Running the script will deploy the entire 2-IMMERSE service stack.

The services brought up by *docker-compose* also include:

1. a local version of DynamoDB to replace the instance running on AWS,
2. an Nginx Docker container that hosts locally mirrored content from the origin server in place of the AWS S3 backed origin server, and
3. an HAProxy Docker container that routes incoming service & origin URLs to the appropriate Docker container / port, as well as provides SSL termination using a self-signed certificate for the domain platform.2immerse.eu (which mirrors what is deployed for production in Rancher).

4 Platform Services

This section of the deliverable summarises updates and noteworthy information relating to platform services deployed to support the Final Release of the 2-IMMERSE service platform.

4.1 Timeline

The following changes have been made to the Timeline service (in addition to general bug fixes):

1. **Initial temporal positioning**: to cater for live broadcasts, where viewers may “tune in” while the programme is already underway, the timeline document may have to be fast-forwarded, skipping over media items that have already been shown and also removed. Other media items may have to be shown, but in turn fast-forwarded to the correct playout position.
2. **Live temporal positioning**: for demonstration purposes (and potentially for allowing timeline navigation in on-demand playback of DMApps, even though that is not pursued in the project) it must be possible to set the playback clock to a specific value. The Timeline service has been adapted to compute which media items and DMApp components should be visible at that time during playout, and send the right activation and deactivation commands to the clients.
3. **Fragment insertion and deletion**: in the previous release (described in deliverable D2.4 (3)) the implementation of inserting and deleting XML fragments in the timeline document was only partially done: only for fragments that were in the future. To cater for pre-production editing and also to solve some issues with live viewing the implementation was completed and events can now also be inserted in the past and “now”.
4. **Websocket updates**: the document update API was only usable via a REST API previously. This does not scale well for live broadcasts (where the Timeline service instances of potentially millions of viewers have to be updated at the same time), therefore a websockets-based API has been added. This allows a broadcast mechanism to be used to update many timeline service instances at the same time.
5. **Repeat element**: as an addition to the *<tl:par>* and *<tl:seq>* structuring elements a *<tl:repeat>* has been added. This element repeatedly plays back its child element, restarting it when it finishes.
6. **Python 3 port**: to prepare for the platform being usable after the end of the project the Timeline service code has been ported to Python 3, because Python 2.7 support is scheduled to terminate in early 2019 (7). This included making all string handling Unicode, and using a different web application framework (*Flask* instead of the older *web.py* which is not Python 3 compatible).

4.2 Layout

There were several major additions to the Layout service:

1. **Device Grouping**: Added groups to the device hierarchy. Users may now define groups of devices over which the engine will generate a layout; groups may be comprised of communal and/or personal device types. If no groups are specified, the default grouping, communal and personal are used.
2. **Vertical Centre Anchor**: until this version, components could be anchored to the sides of the region, we now added the ability to specify vertically-centred components.
3. **Set Priority API:** the API was extended to allow specification of the override of component priorities per device type, device group, or single device. Priorities overrides are set according to those specifications exactly, with no side effects.

4. **Video/Audio component counting:** the layout API now allows specification of the maximum number of audio or video components. Component constraints define whether it is a video and/or audio component, and the layout engine ensures that no more than the maximum such components will be laid out on any individual device.
5. **Aesthetics:** algorithm improvements try to spread the components evenly across the participating communal devices, as best as possible.

4.3 Shared State

There have been no changes made to this service. The MotoGP, Football and Theatre in Schools DMApps do not use this service as there is no sharing of state between contexts in this experience, and this service carries unnecessary overhead for sharing of state between components in the same context.

4.4 Logging and Monitoring

The logging and monitoring infrastructure did not require any updates or changes, however periodic maintenance was required – for example to clean up the ElasticSearch data store.

4.5 Synchronisation Service

Deliverable D2.4 (3) explained how the project had developed a new synchronisation model which could address a diverse set of possible synchronised distributed media experiences, and it provide an architectural overview of the cloud-based Synchronisation Service.

This service has now been integrated within the 2-IMMERSE platform and evaluated independently of the service prototypes developed during the final year of the project. Please see Section 9.2 for a full description of this evaluation.

4.6 Authentication

As described in Section 5.5.3.8, the authentication service was modified to permit requesters to specify the deviceId when linking a device to a user account, as opposed to having the authentication service generate a unique identifier each time. This helped reduce the number of different sources for device identifiers and assisted with the on-boarding implementation. Other minor changes included improved service discovery for the MongoDB service and a bug fix for bypassing token validation when creating new user sign-ins.

4.7 Data Playback

Two additional data sets where added to the data playback concept, player tracking data and camera pose tracking data. Both these data set are relatively large and progresses in the same way video does, with new information for each frame of time tied to the video. This differs compared to the MotoGP timing data which happened more occasionally with less data. We decided to use an approach conceptually more like DASH-streaming for this kind of information, creating chunks of data on the production site and transferring these directly to the CDN. From a consuming point of view the data will requested from the CDN, buffered, re-constructed and played out (rendered) in sync with video.

A short explanation of these data types:

- **Player tracking.** The football trial took place on a stadium equipped with a ChyronHego TRACAB optical player tracking system. This will output in real-time, a 3D-position of every player, referees and ball 25 times a second. This information can, for example, be used to derive statistics and representations of the game in a live tactics board overlay.

- **Camera pose.** For the football trial we set up ChyronHego Virtual Placement software to capture the main camera. The software's normal usage is to insert augmented graphics in to the broadcast video. It this case we used it only to capture and relay the analysed camera pose information (which is necessary to create virtual or augmented graphics). From this analysis we get information such as camera position, pan, tilt, zoom and lens distortion for every video frame. The usage for this information is to create virtual graphics directly in the client.

4.8 Docker-Hive

Docker-Hive provides a Docker image for running distributed integration tests for some of the 2-IMMERSE platform services, in a series of test suites. The test suite uses Chakram and Mocha to test service interactions. By default, test results are reported using the Mocha reporter.

When run in hive mode, a number of test clients can be spun up and run in parallel, supporting basic load test and scalability testing for the platform services.

These test suites cover main functionality of the layout API and engine and are used for regression testing when changes are made to the layout service.

We added sections 07 to test the grouping and 08 for regression testing of the MotoGP service prototype. We also updated the tests to include some new functionality. We also updated the existing suites to account for API / behavioural changes in the layout service as described above.

The currently implemented test suites are:

01 – layout context APIs

02 – layout DMApp APIs

03 – websocket service

04 – layout packer algorithm

05 – layout dynamic algorithm

06 – layout constraints API

07 – layout using device groupings

08 – MotoGP experience simulation

10 - Auth Service APIs

The Docker-hive also includes simple web UI to allow launching of specific test suites against specific platform instances (edge, test, production).

5 Client Application Stack

This section of the deliverable describes changes made to the client application stack for the Final Release.

5.1 Overview

The descriptions of the constituent client software parts that comprise the application stack to run a DMApp remain the same as for previous releases and are summarised in Table 4 for reference.

	Constituent part	***Content***	***Hosting***
7	Timeline events	Timeline document or live-broadcast events	CDN, streamed or broadcast
6	DMApp Components	Video player, leaderboard, animation etc.	CDN
5	**DMApp SPA**	Genre- or programme- specific application comprised of style sheet/theme, images, supplementary content, responsive layout, glue-code and a Layout Requirements document.	CDN
4	**Onboarding SPA** *Launches the DMApp SPA and implements features common to all DMApps*	Genre- or programme- independent application that includes sign-in/up, EPG, discover/join/create contexts, accept/send invites, box office, broadcaster-specific styling and layout (e.g. BT, BBC, Sky etc.).	CDN
3	**Bootstrapping Application** *Launches the Onboarding SPA.*	A simple web page embedded into the native Cordova application, used to redirect the browser to the CDN-hosted Onboarding SPA.	Embedded in Cordova app.
2	Common support libraries and resources	e.g. client-api	CDN
1	**TV Emulator or Cordova Application** *The DMApp Operating System. Contains the bootstrapping application.*	DVB CSS, DIAL, Cordova extensions.	App store

Table 4: DMApp Client Application Stack

In the sections of the document that follow we describe developments in the various parts of the client application stack to deliver the Football at Home, Football Fanzone and Theatre in Schools service prototypes:

- Football DMApp Implementation
- Theatre in Schools DMApp Implementation
- Client API updates

- HbbTV2.0 Emulator updates
- Updates to the onboarding implementation

5.2 Football DMApp Implementation

The Football at Home and Football Fanzone DMApps are structured in a similar way to the MotoGP DMApp as described in deliverable D2.4 (3), however there have been a number of additions and changes to support additional and differing functionality. It is composed of the elements listed below:

- **Client input documents**
  There is a client input document for each of:
  - Running as the main TV.
  - Running as a companion device.
  - Running as a FanZone non-main TV.
  - Running as the FanZone controller on a companion device.
  - Common definitions for running on the main TV and running as a FanZone non-main TV. This document is referenced by each of these input documents.

  These client input documents are used to launch the DMApp on their respective device type.
  The client input documents include:
  - General configuration of the client-api
  - A URL of the HTML document to include.
  - A reference to the DMApp control component. The main TV client input document includes DMApp control component configuration parameters including references to DMApp configuration files.
  - Declarations of layout regions and their properties.
  - Options to enable additional debugging and development features when the DMApp is being run for development or testing.
  - The main TV client input document includes URLs for the timeline and layout documents, and options to select alternative timeline documents.
  - The main TV client input document also includes options: whether or not the TV is running in FanZone mode, and whether or not a local video capture device should be used (for production tool previewing).
- **Timeline documents**
  Timeline documents are loaded into the Timeline service at Context and DMApp initialisation. These include a list of all components which are instantiated during operation of the DMApp, and the temporal relations between components.
  The DMApp currently includes the following timeline documents:
  - A minimalistic timeline with added event definitions suitable for live production use with the Timeline Authoring Tool.
  - An authored timeline generated live during the FA Cup Final at Wembley Stadium, using the Timeline Authoring Tool from the minimalistic timeline with added event definitions (above).
  - A manually authored timeline document which is based on the authored timeline generated document (above). This document has been manually refined to a greater extent than would be possible during live production.
    This is the default timeline and the one which is used for user-testing and trials. This timeline includes all of the components required for a user-facing experience
- **Layout document**
  This includes layout constraints for all components listed in the timeline documents.

A single layout document is used for all timeline documents, for both TV and companion device operation.

- **HTML document**
  There is one HTML document for running as a TV, and one for running as a companion device.
  The HTML documents include the Document Object Model (DOM) elements which are used for layout regions and for user interface backgrounds.
  The HTML document elements are styled by the CSS documents.
- **CSS documents**
  There is one CSS document for running as a TV, one for running as a companion, and one for common definitions on both TV and companion.
  The CSS documents define the sizes, positions, backgrounds, and other properties of document elements, including those used for layout regions. The CSS documents also define common definitions used by DMApp components within the DMApp, including font and other styling properties; this is done by means of CSS rules and CSS variable definitions.
- **Assets**
  Assets such as icons, fonts, animations, Opta data, and other static files are uploaded to fixed URLs on the origin server where they can be referenced by Football components as required.
- **Media**
  Media assets (video and audio) are uploaded to fixed URLs on the origin server where they can be referenced by Football components, configuration files and timeline documents as required.
- **DMApp control component**
  The DMApp control component is an invisible DMApp component which is referenced directly in the client input document and is loaded before context and DMApp are created or joined.
  The DMApp control component is the same for both the TV and companion devices, however it changes its behaviour depending on it whether it detects that the client is operating as a TV or companion device.
  On all devices, the DMApp control component handles:
    - Loading and parsing configuration files, and making the configuration available to other DMApp components.
      To avoid configuration mismatches, configuration files are loaded only on the main TV control component, which distributes the parsed configuration to the other control components.
    - Loading an authentication component which shows a username and password dialog, if authentication information is not already available.

  On all devices except the FanZone Controller companion, the DMApp control component handles:
    - Management of local Picture-in-Picture state and presentation.
    - Providing DMApp-level debugging interfaces for testing and troubleshooting.
    - Managements of replays and media stream selection, selection of and transitions between dual/single video layouts, and display of wipe/transition animations.
    - Instantiation and control of locally-controlled graphics, including: team formations, and notification popups.

  On the main TV and on FanZone TVs, the DMApp control component handles:
    - Adjusting the layout to fill the screen regardless of size/resolution (this is required for 4K TV support).

- Hiding graphical component regions when the TV Graphics Disabled mode is enabled.

On companion devices, the DMApp control component handles:

- Detecting whether the current device should be considered a phone or a tablet, and adjusting the layout as necessary.
- Setting the show/hide state of menu overlay components in response to user interaction.

On the FanZone Controller companion device, the DMApp control component handles:

- Display of a user interface to select which FanZone preset to use.
  This sets a shared variable which is read by DMApp control component instances running on the FanZone TV devices.
- Changing the layout and playing media according to the current value of the FanZone preset shared variable, the current device's FanZone screen ID, and the FanZone configuration, see below.

- **Configuration files**
  Some aspects of the Football DMApp are controlled by separate configuration files. These files are loaded by the DMApp control component on the main TV.
  These files include:
  - Events listing
    This includes: a list of all replay events, and a list of all VOD clips. Each list item includes the media URLs, event time, event type, team association, and descriptive text.
    This file also includes timing information for when to display pre-defined notification popups.
  - Stats listing
    This includes values for the statistics: attempts, shots on target, corners, fouls, and possession, for each of the two teams, sampled at a regular interval during the match.
    This data is in a separate file because it was not included in the Opta data recorded during the live event.
  - FanZone configuration
    This defines the media configuration for each of the FanZone screen IDs, for each of the four FanZone presets.
    The media configuration includes: which broadcast stream to show, whether or not to display two videos streams dual-screen, whether each of the two Picture-in-Pictures (PiPs) should be shown.
  - Timeline navigation configuration
    This configures the timeline navigation component, which is used to be able to seek to key points in the event, or to automatically run a highlight reel. This is useful for demos as it is no longer necessary to wait a significant amount of time until key moments in the event occur.
    The configuration includes: layout templates, bookmarks, and highlight reels.
    Layout templates define: which Picture-in-Pictures are enabled on the TV and companion devices and at what size/scale and which video stream is shown on companion devices.
    Bookmarks specify a time, a layout template and a display name. Clicking the bookmark in the timeline navigation component seeks the DMApp to that point and adjusts the layout according to the layout template.
    Highlight reels specify an array of clips, each of which specifies: a time, a duration, and a layout template. Clicking the highlight reel in the timeline navigation

components automatically executes each clip in sequence, in the same way as for bookmarks, above.

- **Football-specific DMApp components**
  The Football DMApp includes a number of Football specific components, these are described in detail in Section 6.
- **ChyronHego Prime animations**
  The majority of the non-interactive graphical overlays used in the DMApp use the Prime component. This is a generic DMApp component which executes graphical overlays and animations produced by a corresponding dedicated graphics/animation authoring tool. This is described in Section 6.

5.2.1 Football data

Football data as provided by Opta is in the form of file which describes the current state of an event, this file is in a documented XML format. During a live event, this file is regularly updated. Opta data clients poll the file and apply any observed changes as necessary.

During the FA Cup Final event at Wembley, each update to the Opta data file for the event which changed the file contents was reformatted as JSON and recorded as a new file, with the current time encoded into the file name. An additional JSON file was also created which lists all of the data file names.

The DMApp control component includes parameters which specify the URL of the list file, above, and the mapping between the timestamps encoded into the file names, and the DMApp/timeline timestamps. This mapping is not required to be continuous. These parameters are set for the main TV device only. The manually authored FA Cup Final VOD timeline is discontinuous with respect to the Opta data file timestamps because half time was removed from the recorded media and therefore from the DMApp/timeline.

The main TV DMApp control component maps the current DMApp clock time into the same time domain as is encoded in the Opta data file names as per the component parameters, and selects the most recent data file with a timestamp less than or equal to that value. The data file is loaded dynamically and its contents are stored into a shared variable. This selection re-occurs as the current DMApp clock time changes.

Other components including the score clock menu running on the companion devices update their displayed contents based on the current value of the shared variable.

5.2.2 Deployment

For testing purposes three copies of the Football DMApp (excluding audio and video media) are deployed to the origin server.

These are labelled as edge, test and production. These correspond to 3 branches in the Football git repository: master (the default branch), test and production respectively. New features are generally deployed to edge first, for testing, and if the results of testing are satisfactory they are also deployed to test. When the feature is suitable for production (use by external users), it is also deployed to production.

As assets, DMApp components, and other deployed items have fixed URLs, the build and deployment scripts adjust these URLs to use different directories when building and deploying to edge and test. The service URLs referenced in the client input documents are also adjusted when deploying to edge and test, to default to using the corresponding edge and test deployments of the services.

To avoid accidental deployments to production and test, the build and deployment scripts include a number of checks of the current state of the Football git repository both locally and relative to the copy on the server before permitting a deployment to production and test.

5.3 Theatre in Schools DMApp Implementation

For the fourth and last trial, Theatre in Schools (TiS), two DMApps have been developed, each of them meant to be used during a full lesson for a class of students:

- StoryBuilder
- Script Detective

The TiS DMApps are different to for example the Football DMApp as they make use of a new DMApp component, the ContentBrowser, which enables an alternative way to create 2-IMMERSE applications, especially for applications that are made of a lot of interactive elements. While in previous apps each overlay graphic is a separate DMApp component, the ContentBrowser can combine multiple elements in a single component.

In the TiS scenario, there is one communal device, the TV served by an Intel NUC, and tablets for the teacher and up to 6 student groups.

A TiS DMApp consists of a single ContentBrowser component for each of the involved devices, which are spawned by simple timeline and layout documents through client configuration documents as in previous DMApps. Any data that is generated through interactions by students and the teacher is only kept and shared locally via the app2app service, unless network logging is enabled for debugging purposes.

A TiS DMApp consists of the following documents:

- **Client Input documents:**
  There is a client input for each of:
    - the TV
    - tablets independent of whether used as teacher or student tablet

  The input documents contain
    - configuration of the client API
    - link to the TiS control component
    - structure and initial values of local and shared signals (shared data)
        - metadata of videos used
        - static text used during the lesson
        - data storing the state (e.g. phase) of the lesson
        - data generated by students during the lesson

  The DMApp loads a single file which is constructed from multiple input documents:
    - client.yaml and client-companion.yaml contain config of the client API and references to other input documents
    - all files in the folder 'signals-config' define the base signal structure and reference additional files for individual phases of a lesson
    - each phase (watch, make, share) may reference a 'bookmarks.yaml' and a 'strings.yaml' file which hold metadata for media content used and static text. Those files are contained in sub folders for each of the phases containing the HTML templates (see below)
- **Timeline and Layout documents**
    - Simple documents are contained to initialize the content browser component
- **Assets**
    - Icons, thumbs and still images

- **CSS**
  - One CSS file with common definitions for TV and tablets
  - One with individual definitions for TV
  - One with definitions for students and teacher tablets
- **DMAppControl component**
  The DMApp control component is an invisible DMApp component which is referenced directly in the client input document and is loaded before context and DMApp are created or joined.
  The DMApp control component is the same for both the TV and companion devices, however it changes its behaviour depending on it whether it detects that the client is operating as a TV or companion device.
  The two Theatre in Schools DMApps have two different but similar DMApp control components, they will be described together below.
  On all devices, the DMApp control components handle:
  - Adjusting the layout to fill the screen regardless of size/resolution (this is required for 4K TV support, and to support different companion devices sizes).
  - Loading an authentication component which shows a username and password dialog, if authentication information is not already available.
  - Creating, configuring and managing bookmark timeline component instances, and their associated state. The details of this vary significantly between DMApps.
  - Merging, pre-processing and re-formatting of various signal values into formats which can be more directly inserted into ContentBrowser component templates. The details of this vary significantly between DMApps.
  - Providing debugging interfaces to enable inspection and modification of the DMApp state.

  On the TV device, the DMApp control components handle:
  - Automatic assignment of new companion devices into a vacant student group.

  On companion devices, the DMApp control components handle:
  - Adjusting the layout to compensate for any on-screen keyboard which is opened on Android devices.
  - Changing the app to a full-screen configuration, on Android devices.
- **HTML templates and documents**
  - The templates in the common folder mainly contain those meant to be reused across devices and the different phases of the session like the top, bottom and left bars.
  - There is one folder for each phase with
    - one shared main document, named like the phase, e.g. WatchDisplay.html, that links to the specific documents below depending on the selected role during the configuration phase
    - one document or more documents for each of the roles TV, teacher and student, depending on complexity of a phase
    - a strings.yaml as described above
    - a bookmarks.yaml as described above, during the share phase the one from make is reused
  - The templates make use of the signals provided through the client API to alter the state of the session, for example the shared signal “signals.shared.phases.current” can have the values ‘config’, ‘watch’, ‘make’, and ‘share’. The teacher can control it by changing its value through the left bar menu. Conditional loading (a feature of the vuejs framework integrated in the ContentBrowser) of sub templates based on the value of a signal, which part of the DMApp gets visible.

- If one device crashes during the lesson, e.g. a student closes the app on the tablet accidentally, the state is recovered when the app is restarted by exchanging signal data with the other devices.

5.3.1 Deployment

The same concept as described for the Football DMApp above is used.

5.4 Client API

5.4.1 DMApp Component Interface

DMApp Components are a way to encapsulate functionality and user interface elements in discrete entities which are individually specified and controllable by the Layout Service.

A DMApp Component is a JavaScript object which as a minimum meets a defined and documented JavaScript interface. This interface does not require the use of any specific library or style to create a DMApp Component, but is instead designed to ensure flexibility of implementation, and support simple conversion or wrapping of existing 3rd party functionality into DMApp Components.

Additional features have been added to the DMApp Component Interface, and minor changes made to existing areas which were present in release 1, however these changes and additions have been made with backwards compatibility in mind.

Existing DMApp Components designed for releases 1 and 2 of the interface continue to work with release 3 and with intermediary unreleased versions.

Notable changes to the DMApp Component interface are listed below:

- Additional creation options/modes for child DMApp components, including whether the child component should inherit the parent's start/stop times, layout configuration, visibility blocked state and soft-stopped state.
- Changes to the component start/stop timing model in the case where the current time is changed in the negative direction, in particular in the case where the time is changed to be before the component start time.
- DMApp component configuration states received from the timeline service via the layout service and the associated instances at the client now include a revision number. This revision number is included in status update transactions to the timeline service via the layout service, and can be read by the component instance. This revision number is used to distinguish between different instances of DMApp components with the same name in the case where a component is destroyed and later re-created, and prevent mis-routing of update messages. This can occur when the timeline is advanced in a negative direction.
- Add universal component parameters to acquire reference count signals when presentable.

Other client-api functionality and interfaces not attached to the DMApp Component interfaces are available for use by DMApp Components which choose to use them. Notable changes and additions to these interfaces and functionality are listed below:

- The interfaces for observable values/variables (referred to in the client documentation as "signals") have been extended. New functionality includes: reference-count signals aggregated over the set of all devices, per-signal configurable equality/change detection, methods to enumerate existing signals, methods to simplify and increase the efficiency of partial writes/updates to signal values, methods to simplify chaining/aggregation of local reference-count and block-count type signals.

- Add interfaces to enable DMApp components to monitor the presence of other DMApp components by name on the same device, including top-level components, and child components.
- DMApp components can now send timeline event notifications directly to the timeline service without routing them via the layout service. This is useful for child DMApp components because they cannot contact the layout service using their own component ID.
- DMApp components can now acquire additional top-level names in the app2app message namespace. This is useful for child components and those with non-static component IDs to expose statically-named RPC endpoints to other components.

5.4.2 Client Input Document Changes and Improvements

Client input documents can be used by the client-api to load and configure a DMApp. Using client input documents removes the need to develop custom loader pages for each DMApp and reduces the overhead associated with developing a DMApp.

Notable changes and additions to the document format are listed below:

- Layout region definitions can now include coordinate transformations in the dimensions reported to the layout service and in the component coordinates within the region received from the layout service.
- Input documents now include an option base URL field, such that input documents using relative URLs can be easily served from a different URL.
- Addition of a field to indicate the mode in which this input document is being launched by the authoring tool.
- Input documents can now indicate that a variation override is to be applied by default.
- Input documents can now unconditionally include other input documents, by URL.
- Add a companion option to disable joining the context/DMApp via the layout service, but still join and participate in local app2app communication.
- Add a mechanism to arbitrarily set the value of named signals according to schedules defined in the document, relative to the default clock.
- Support for setting auxiliary information for discovery advertisement ad discovery filtering, see section 5.4.3.

5.4.3 Other Client API Improvements

In addition to functionality and interfaces which are exposed to and/or useful for DMApp Components, other improvements were made to support non-component development and usage, and improve non-component specific functionality. Notable changes and additions include:

- Multiple DMApp component status change notifications to the layout service are now batched and transmitted using a single HTTP request where possible to improve efficiency and reduce unnecessary layout recalculations.
- The discovery process now includes support for attaching arbitrary auxiliary information to discovery advertisements/broadcasts, and using this additional information for filtering of the set of discovered devices. This is used to prevent the discovery and joining process from connecting devices which are running unrelated and incompatible DMApp instances.
- Improved support for playback and synchronisation of live MPEG DASH media streams in the media player components.
- Addition of further modes/options, status reporting and signalling to the media player component.

- Addition of support for local media capture devices/hardware to the media player component.
- Client support for synchronisation using the Cloud Sync service.
- Client support for the Unified Launcher discovery and launch mechanism.
- Additional debugging commands and interfaces to facilitate easier and more rapid troubleshooting and development.

5.5 Onboarding

5.5.1 Introduction

The Theatre at Home service trial used a bespoke “rendezvous” mechanism to support inter-home synchronisation and experience launch, whilst the MotoGP-At-Home service trial used pairing codes to demonstrate cross-network support, with experience launch orchestrated via the Authentication service.

Whilst Football Fanzone and Theatre in Schools service trials have a more streamlined on-boarding user experience, they require a more advanced implementation strategy. A revised user experience design satisfying the requirements of these service trials is documented in detail in deliverable D3.4 (8).

The following sections discuss the updated implementation; a generalised on-boarding process and unified launcher application.

5.5.2 Unified Launcher

Building bespoke on-boarding user journeys for each service trial was not a sustainable approach. We decided to consolidate the different implementations into a single all-purpose on-boarding implementation that could support the diverse requirements of all three remaining service trials. This became known as the ‘Unified Launcher’.

The goal was to address sustainability and to optimise the workflow to make it easier to demonstrate the full range of 2-IMMERSE experience prototypes without complex device setup and versioning issues. The resulting on-boarding implementation is comprised of four components that enable any 2-IMMERSE multi-screen experience to be configured and launched:

1. Android Unified Launcher host application
2. Android Unified Launcher configuration application (for development)
3. TV Emulator firmware
4. Unified Launcher web application

The following sections discuss the implementations of each of these Unified Launcher components.

5.5.2.1 Android Unified Launcher host application

The Unified Launcher host application for Android is a derivative of the original MotoGP host application but it resides in its own repository and is fully data-driven. It is independent of any specific service trial implementation.

The host application’s job is to download and run the Unified Launcher web application from the CDN within a WebView. The URL and service environment settings to use (edge, test, production) are loaded from a local config file written out by the Android Unified Launcher configuration application.

This permits the Android Unified Launcher to be used unmodified in the future by any hosted instance of the 2-IMMERSE platform. It is the Android platform's counterpart to the 2-IMMERSE TV Emulator Firmware which fulfils a similar role for TVs.

5.5.2.2 Android Unified Launcher configuration application (for development)

The configuration application is stored in the same repository as the Android Unified Launcher host application. It stores the Unified Launcher web application URL and environment settings in a file that's shared with the Android Unified Launcher application. It provides the user with a page of configuration settings which includes:

1. The URL of a hosted JSON configuration file, known as the 'Service Pre-sets File'. This file contains a list of 2-IMMERSE REST end-points for each of the platform services, grouped by hosted platform instance.
2. The hosted platform instance to use.
3. Debug settings

5.5.2.3 TV Emulator firmware

The TV emulator firmware has been described in detail in Annex B of deliverable D2.4 (3). Improvements have been made to the firmware to make it easier to demonstrate multi-screen experiences in environments where lots of TV Emulators are deployed and to support for the new Unified Launcher on-boarding design.

5.5.2.3.1 TV Emulator Name

The user now has the ability to configure the Wi-Fi ESSID name and DIAL Unique Device Name (UDN) advertised by the TV Emulator, which have been made consistent. Making the ESSID and UDN the same reduces confusion, as does the ability to choose a user-friendly name, especially in environments where multiple TV Emulators are deployed. The TV Emulator name is generally used as a hint to differentiate between TVs that are positioned in different locations around a venue, so that they can be assigned specific Device Roles by the Unified Launcher web application.

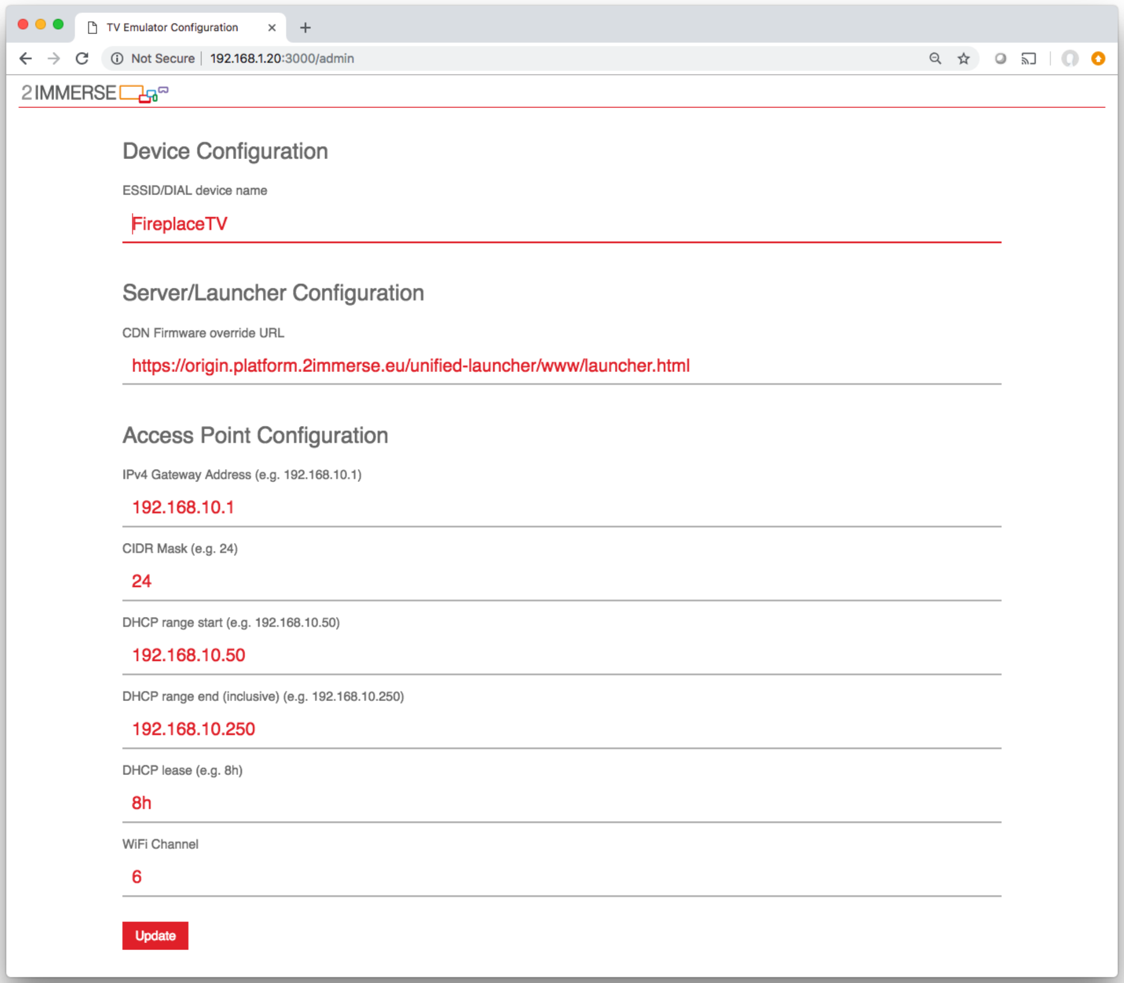
5.5.2.3.2 Service Environment

The original firmware allowed the user to specify the location of the 'Service Pre-sets File' and the name of the environment that experiences should use. With the switch to a Unified Launcher, the 'Service Pre-sets File' and the environment are now dictated by the companion device that launched the experience. These settings are included in the launch data payload sent to the DIAL server of the TV emulator. Therefore, this setting has been removed from the TV Emulator firmware as it is no longer needed.

5.5.2.3.3 Access Point Configuration

A challenge faced with the introduction of stand-alone platform instances was that sometimes the DHCP ranges assigned by the routers used in demo configurations would conflict with the default range used by the TV Emulator firmware, which was used by the firmware's captive portal and internet connection sharing functionality.

It is not always feasible to re-configure routers at a venue, so the TV Emulator firmware was modified to allow the access point to be reconfigured. Users were given control over the gateway address, CIDR mask, DHCP range start/end, DHCP lease time and Wi-Fi channel. The latter was done to address issues of demonstrating 2-IMMERSE experiences in noisy Wi-Fi environments such as IBC 2018. Figure 2 shows a screen-grab of the admin page.


TV Emulator Configuration
Not Secure | 192.168.1.20:3000/admin
2IMMERSE

Device Configuration

ESSID/DIAL device name: FireplaceTV

Server/Launcher Configuration

CDN Firmware override URL: https://origin.platform.2immerse.eu/unified-launcher/www/launcher.html

Access Point Configuration

IPv4 Gateway Address (e.g. 192.168.10.1): 192.168.10.1

CIDR Mask (e.g. 24): 24

DHCP range start (e.g. 192.168.10.50): 192.168.10.50

DHCP range end (inclusive) (e.g. 192.168.10.250): 192.168.10.250

DHCP lease (e.g. 8h): 8h

WiFi Channel: 6

Figure 2: New TV Emulator Administration Settings

5.5.2.3.4 Proxying of DIAL launch payloads

Previously, application launch had been implemented entirely by the Unified Launcher web application running in the TV Emulator's web browser. Launch was handled using the 2-IMMERSE cloud-hosted platform services. Launch payloads were exchanged between devices by the Authentication service.

For the Unified Launcher, DIAL was chosen in preference in order to streamline simultaneous launch on multiple TVs and to avoid having to enter separate pairing codes for each TV. The DIAL server on the TV Emulator firmware was modified to forward the payload that described the programme being launched to the Unified Launcher web application, which was running in Chromium.

The Unified Launcher web application maintains a persistent websocket connection to a local web server running on the TV emulator in order to be notified about system events such as changes in network connectivity. The DIAL server uses the local web server to proxy the launch payload to Chromium by sending it an HTTP POST request with the launch payload in the body. The local web server has a route set up to forward the payload using a websocket message to Chromium.

5.5.2.3.5 Upgrades and Bug Fixes

Other fixes included improved network stack management and upgrades to drivers, system components and work to ensure the latest version of Chromium was being used to achieve consistency with Chromium on desktop PC development environments.

5.5.3 Unified Launcher Web Application

Changes to generalise the launcher and to support on-boarding for the Football Fanzone and Theatre in Schools service trials were mostly made to the Unified Launcher web application hosted on the CDN. This sections describes each of the major changes.

5.5.3.1 Electronic Programme Guide

The web application was modified to be able to load an Electronic Programme Guide (EPG) so that the user could choose which multi-screen programme to launch. The EPG data model is shown below in Figure 3.

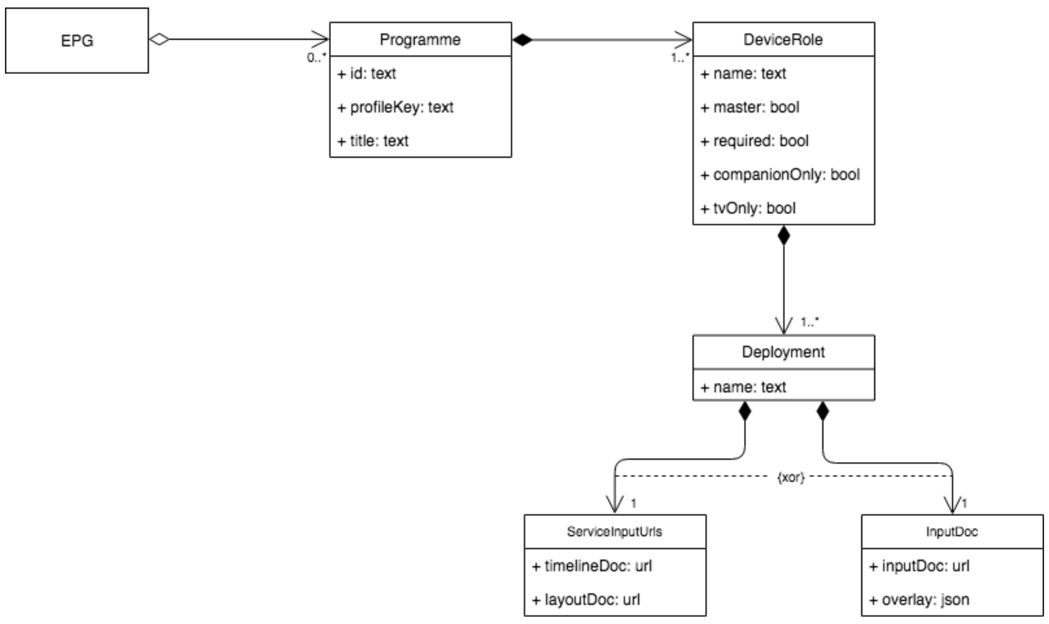


Figure 3: Electronic Programme Guide (EPG) Class Diagram

The EPG data model contains a list of Programme descriptions that are used to populate the web application's content selector. Each Programme defines one or more DeviceRoles which the user can assign to specific TVs or companion devices enrolled into a multi-screen experience. A DeviceRole dictates the initial configuration of a programme on a specific device. Properties of the DeviceRole are used to validate the configuration before launch. For example, all experiences require a 'Primary Master' TV role to be configured due to the master-slave design of HbbTV2.0. A DeviceRole also has a user-friendly name, which is chosen by content authors according to the genre of programme. This name is eventually passed onto the DMApp where is can be used as a hint by the application logic to orchestrate the experience.

A DeviceRole defines a Deployment for each hosted 2-IMMERSE environment such as 'edge', 'test' and 'production'. The Deployment contains the necessary information for the web application to launch the programme in that environment. It comes in the form of either a Client API InputDocument fragment or a pair of timeline and layout document URLs (known as a

ServiceInputUrls pair). An InputDocument fragment is consumed directly by the Client API during initialisation. Typically, there is one InputDocument URL for each device role, but it is possible to use the same InputDocument as a template and override settings using the JSON overlay property.

A copy of the Deployment description is sent as the launch payload to the DIAL server by the companion application to start an HbbTV2.0 application running on the TV Emulator.

5.5.3.2 User Profile Preferences

The Unified Launcher web application stores and retrieves user preferences in the profile of the signed-in user. This includes all DeviceRole assignments on a per-programme basis. The Programme's profileKey attribute (see Figure 3) is a unique key under which the preferences are stored in the user's profile. The profileKey can be shared by multiple Programmes. This is useful if there are multiple episodes of the same Programme and the user wishes to retain the same preferences for each. It saves unnecessary re-configuration.

5.5.3.3 Live EPG Updates

A list of live broadcasts hosted by the 2-IMMERSE platform were requested periodically from the 'editor' service and used to update the EPG. The live session description response included the URL of an InputDocument which was used to generate a Deployment description for the Programme. This allows users to launch live and on-demand content from the same application via a single EPG.

5.5.3.4 DIAL-based launch (Master TV Device)

The launch process adopted by the new on-boarding design follows the DIAL specification (9). The sequence for using a companion device to remotely launch a DMApp on a master TV is shown in the sequence diagram in Figure 4.

Firstly, the Unified Launcher web application running on the companion device performs DIAL discovery to populate the user interface with available TV devices. The user can assign each device a DeviceRole for a given programme. When that programme is launched, the web application iterates over each device, starting with the device enrolled as the master and repeats the DIAL launch protocol.

Whilst the protocol is well defined, how the applications themselves are loaded, run and terminated on an HbbTV2.0 terminal is outside the scope of the DIAL specification. In the 2-IMMERSE TV emulator firmware, a shell script is run by the DIAL server to forward the launch payload to the TV's Unified Launcher web application already running in Chromium.

The TV needs to obtain an access token from the 2-IMMERSE authentication service corresponding to its designated DIAL Unique Device Name (UDN). The access token allows the DMApp to communicate with the 2-IMMERSE platform services securely. This is achieved by periodically polling the 2-IMMERSE Authentication server and waiting in a pending state until an access token is granted.

Meanwhile, the Unified Launcher web application running on the companion device registers the UDN of each TV device with the 2-IMMERSE Authentication service to grant it an access token and access to 2-IMMERSE platform services under the credentials of the signed-in user. When the Unified Launcher web application running on the TV emulator obtains an access token, it proceeds to load and run the DMApp.

As a final step, the Unified Launcher web application on the TV advertises the fact that it is now running a DMApp by sending an HTTP POST request to its own local DIAL server.

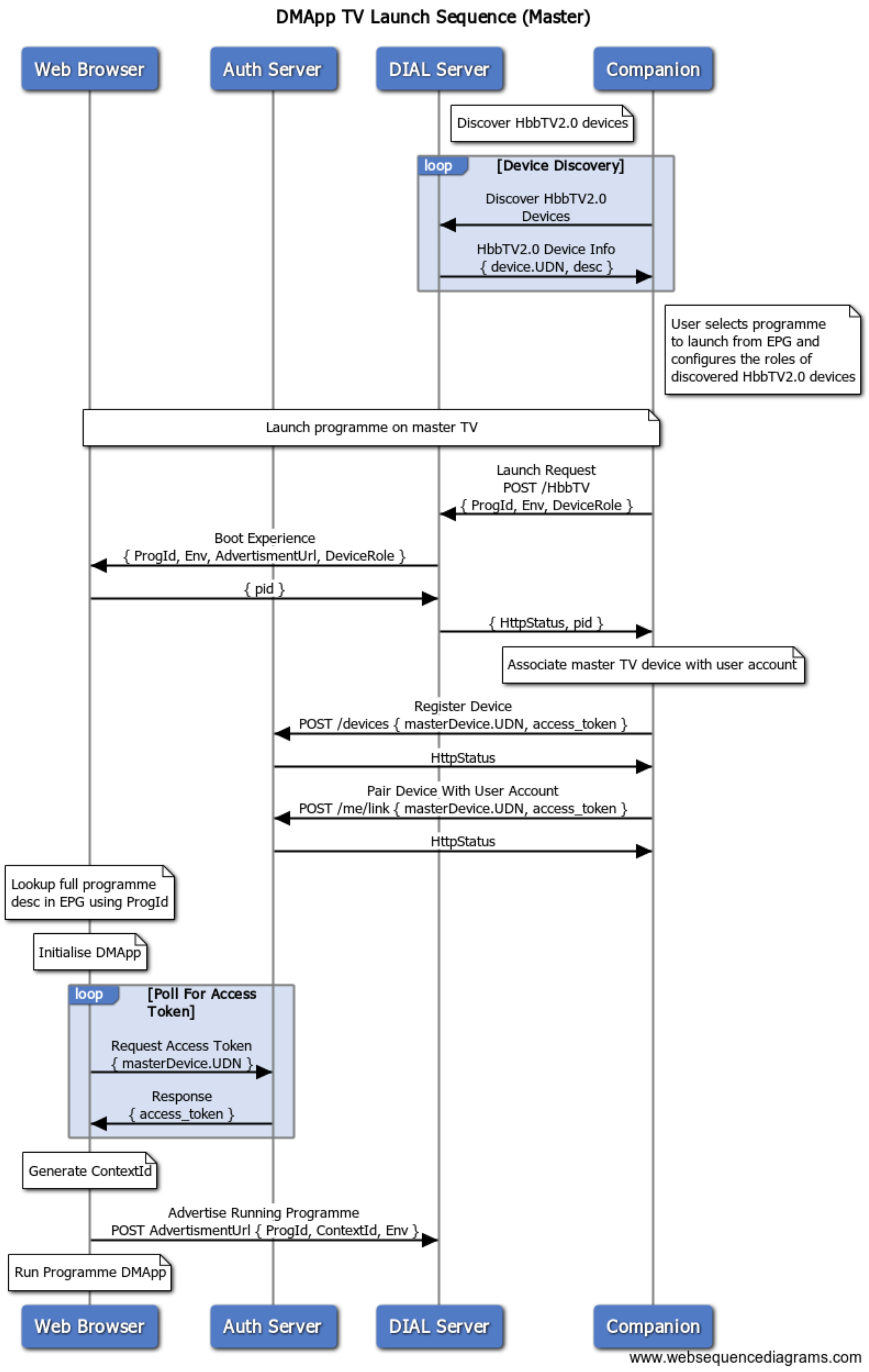

DMApp TV Launch Sequence (Master)
Web Browser
Auth Server
DIAL Server
Companion
Discover HbbTV2.0 devices
loop [Device Discovery]
Discover HbbTV2.0 Devices
HbbTV2.0 Device Info { device.UDN, desc }
User selects programme to launch from EPG and configures the roles of discovered HbbTV2.0 devices
Launch programme on master TV
Launch Request POST /HbbTV { ProgId, Env, DeviceRole }
Boot Experience { ProgId, Env, AdvertismentUrl, DeviceRole }
{ pid }
{ HttpStatus, pid }
Associate master TV device with user account
Register Device POST /devices { masterDevice.UDN, access_token }
HttpStatus
Pair Device With User Account POST /me/link { masterDevice.UDN, access_token }
HttpStatus
Lookup full programme desc in EPG using ProgId
Initialise DMApp
loop [Poll For Access Token]
Request Access Token { masterDevice.UDN }
Response { access_token }
Generate ContextId
Advertise Running Programme POST AdvertismentUrl { ProgId, ContextId, Env }
Run Programme DMApp
www.websequencediagrams.com

Figure 4: Sequence for using a companion device to remotely launch a DMApp on a master TV

5.5.3.5 DIAL-based Join for Slave TV Devices

The sequence for using a companion device to remotely join a DMApp on a slave TV is similar to that of launching on a master TV as shown in the sequence diagram in **Error! Reference source not ound.**.

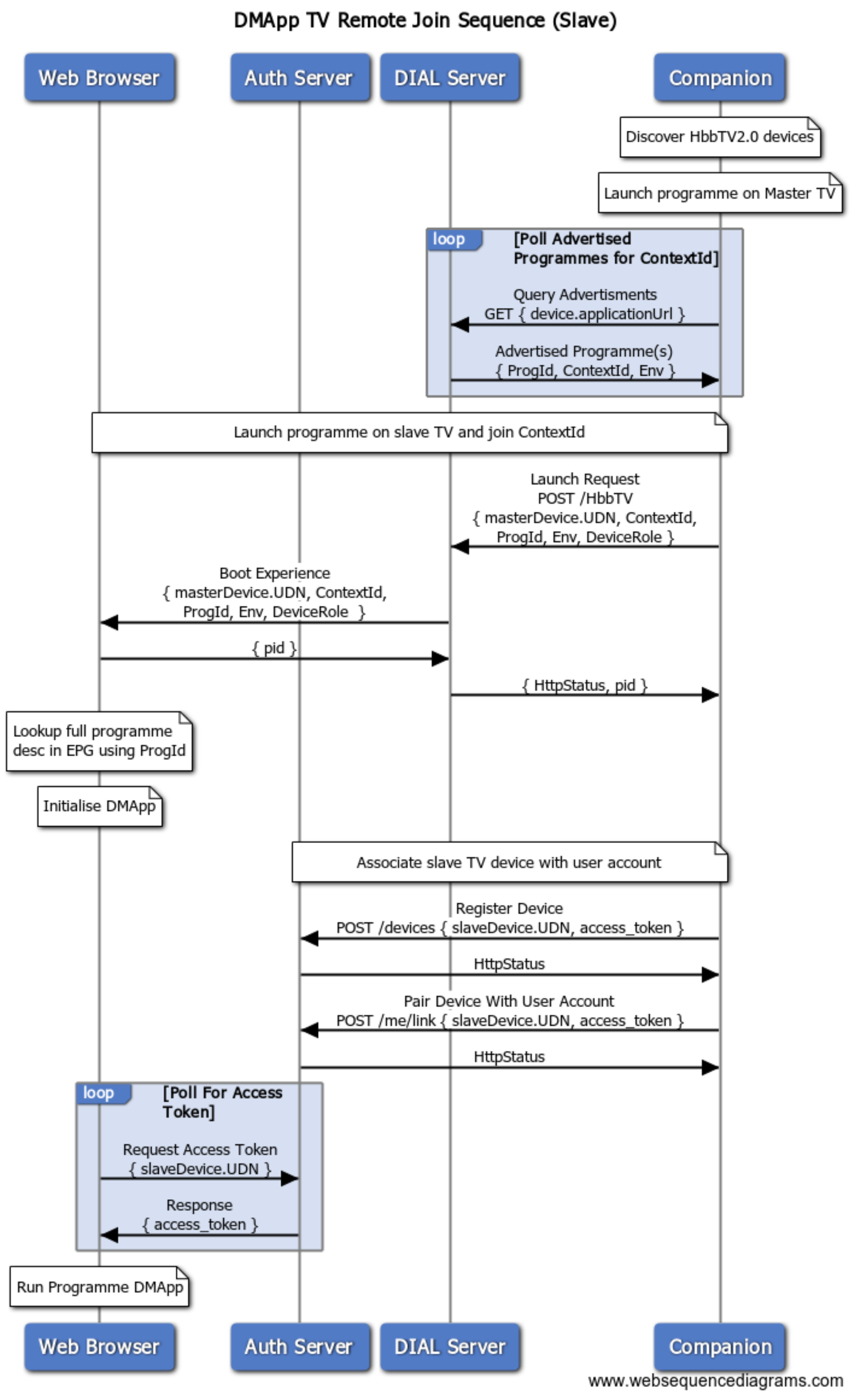

DMApp TV Remote Join Sequence (Slave)
Web Browser
Auth Server
DIAL Server
Companion
Discover HbbTV2.0 devices
Launch programme on Master TV
loop
[Poll Advertised Programmes for ContextId]
Query Advertisments GET { device.applicationUrl }
Advertised Programme(s) { ProgId, ContextId, Env }
Launch programme on slave TV and join ContextId
Launch Request POST /HbbTV { masterDevice.UDN, ContextId, ProgId, Env, DeviceRole }
Boot Experience { masterDevice.UDN, ContextId, ProgId, Env, DeviceRole }
{ pid }
{ HttpStatus, pid }
Lookup full programme desc in EPG using ProgId
Initialise DMApp
Associate slave TV device with user account
Register Device POST /devices { slaveDevice.UDN, access_token }
HttpStatus
Pair Device With User Account POST /me/link { slaveDevice.UDN, access_token }
HttpStatus
loop
[Poll For Access Token]
Request Access Token { slaveDevice.UDN }
Response { access_token }
Run Programme DMApp
Web Browser
Auth Server
DIAL Server
Companion
www.websequencediagrams.com

Figure 5: Sequence for using a companion device to remotely join a slave TV

The DMApp on the master TV is always launched first so that it can advertise the contextId that uniquely identifies the instance of the Programme. The Unified Launcher web application running on the companion device polls the DIAL server of the master TV until it advertises this contextId. The contextId and the UDN identifying the master TV are forwarded in the launch payload to other slave TVs. This is the information required to join an existing multi-screen context and is the same information required by a companion device wishing to join the context. Note that only the master TV makes an advertisement to its DIAL server. This ensures the same experience isn't enumerated multiple times during DIAL discovery.

5.5.3.6 DIAL-based Join for companion devices

The process for joining a companion device is much simpler than remotely joining a slave TV, but it still uses the DIAL server advertisement process as shown in Figure 6.

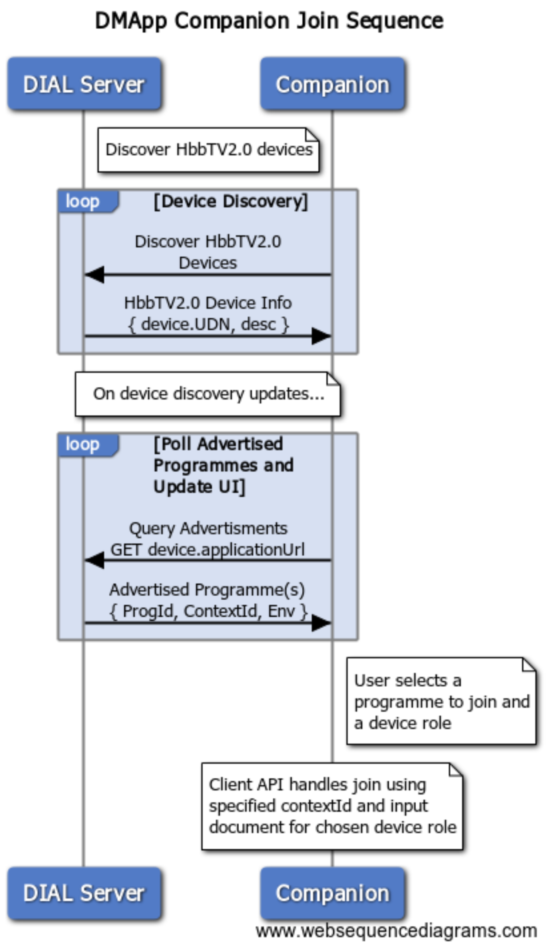

DMApp Companion Join Sequence
DIAL Server
Companion
Discover HbbTV2.0 devices
loop
[Device Discovery]
Discover HbbTV2.0 Devices
HbbTV2.0 Device Info { device.UDN, desc }
On device discovery updates...
loop
[Poll Advertised Programmes and Update UI]
Query Advertisments GET device.applicationUrl
Advertised Programme(s) { ProgId, ContextId, Env }
User selects a programme to join and a device role
Client API handles join using specified contextId and input document for chosen device role
DIAL Server
Companion
www.websequencediagrams.com

Figure 6 Companion Join Sequence

The main addition is the ability to specify a DeviceRole for the companion device. This allowed the DMApp to differentiate between the Teacher's and the Student's tablets in the Theatre-In-Schools service trial and between the Fanzone manager's tablet and devices of patrons of the Football Fanzone service trial.

5.5.3.7 Terminating applications on multiple remote TVs

In addition to launching and joining experiences on remote TVs, it's important to be able to terminate them again. This is especially important if the master TV is directed to launch a different Programme because other Slave TVs will be orphaned. The Unified Launcher web application provides both Join and Stop options for the running programmes it discovers.

Stopping an experience involves iterating over all devices that were assigned DeviceRoles for the given experience instance and making an HTTP DELETE request to their DIAL servers.

5.5.3.8 Authentication Service & Device Identifiers

Prior to the revised on-boarding design, there were four different device identifiers in use by the 2-IMMERSE platform:

1. Wi-Fi ESSID
2. DIAL UDN
3. Client API device identifier
4. Authentication service device identifier (for linking devices with user accounts)

The Unified Launcher has consolidated the use of device identifiers from four down to just using the DIAL UDN device identifier. This allows the companion device, Authentication service and TV emulator to negotiate secure application launch and join.

There was a minor change to the Authentication service device registration to allow the requester to specify a user-defined deviceId and a small change to the initialisation of the Client API to pass the DIAL UDN in as the deviceId it should use.

5.5.4 On-boarding Summary

Migration to a reusable Unified Launcher implementation has helped support the requirements of Football Fanzone and Theatre in Schools service prototypes, but perhaps the biggest benefit has been improved platform sustainability, maintainability and usability.

Pairing codes were stripped out as a simplification, but would need to be re-introduced to support the requirements of DMApp contexts spanning multiple networks. This is needed for inter-home synchronisation in the Theatre at Home service trial. In addition, context and device discovery that spans multiple networks would require devices to advertise themselves in named lobbies, as was the case with Theatre at Home.

This hybrid approach is featured in the on-boarding user experience design in work package 3 deliverable D3.4 - User Interaction Design: the development of generic components & features.

6 Multi-Screen Experience (DMApp) Components

This section provides a full alphabetical list of DMApp Components implemented for the 2-IMMERSE platform at the time of the Final Release, including those developed for the Theatre at Home, MotoGP, Football at Home, Football Fanzone and Theatre in Schools DMApps. Table 5 on the following pages describes each component and provides comments on how they were used in specific DMApps, building on the information already provided in deliverables D2.3 (10) and D2.4 (3).

It will be seen that several components contain core functionality which in some cases has been re-used across multiple DMApps, or could be re-used in the future. Some components provide specific functionality which meet the more precise requirements of a genre, be it filmed theatre or live sports, and as such were developed for only one or two DMApps within 2-IMMERSE. However, in most cases their visual appearance could be reconfigured to enable re-use within their genre – for example for different football matches or motorsports events.

Name	Description	Comments
Adobe Animate	This is a generic component which supports the playback and control of a JavaScript-based animation exported from Adobe Animate.	14 MotoGP-specific animations were developed using Adobe Animate and derived from this component. The information presented in many of these components is determined by live data provided by the Data Spooler component. These components are designed to momentarily provide additional information overlaid on the main race video at specific times during the race, as determined by the pre-authored timeline.
Article	This can be used to present a range of additional content, including cast/creatives bios. Content is authored using a simple markdown format and the viewing position within the article can be synchronized between multiple instances of the component.	In the Theatre at Home DMApp, Article components on both the TV Emulator and companion device are used to present a wide range of additional content. During the show, this is restricted to the companion only.
Article Controls	This enables the user to interact with content presented in the Article component.	In the Theatre at Home DMApp, the control options provide the ability to scroll up and down within the Article, and is only available on the companion.
Bookmarked Timeline	This component controls the playback of a separate media player component instance according to a configured list of subranges of the video (clips). This component has an optional user interface; however it can be remotely controlled via an RPC interface whether or not this is enabled. Users (via the optional user interface or the RPC interface) can queue or unqueue subranges of the video as defined in the component configuration, and these are then played in	This is used invisibly in the Theatre in Schools DMApp. The user interface is not used.

Name	Description	Comments
	sequence.	
Component Switcher	This provides a UI to enable different parts of the experience (and hence DMApp Components) to be selected. It is responsive to the device on which it is running and can be 'collapsed' so that it occupies minimal screen space when it is not needed.	This is used in the Theatre at Home DMApp to allow content to be selected for presentation in the Image and Article components only.
Content Browser	This component integrates the UI framework VUE.JS into the existing client-api infrastructure. This component enables the creation of more complex user interfaces with reduced developer effort requirements. The ContentBrowser supports templates for reusing components across different UI views. The signal handling of the client-api that is used to synchronise data models within an application's context (i.e. the various communal and personal devices) is connected to VUE.JS data binding. Thus the state or content of a UI component can be controlled simply by changing a signal from somewhere in the context. ContentBrowser templates can embed other DMApp components, e.g. the video components, and the ContentBrowser provides a mechanism to send events to the timeline service.	This is used extensively in the Theatre in Schools DMApp. Detailed documentation on the ContentBrowser and tech samples can be found in the dmapp-components repository under ./components/content-browser/README.md
Football Broadcast Menu	This presents a menu user interface which the user can interact with to control various aspects of the Football DMApp. This is primarily used to control the selection of video stream content.	This is presented on companion devices in the Football at Home and Football Fanzone DMApps.
Football Goal Info	This component presents a notification that a goal has been	This is presented on TV and companion devices in the Football at

Name	Description	Comments
Popup	scored in a separate concurrently running football match.	Home and Football Fanzone DMApps.
Football Score-clock Menu	This presents a menu user interface which the user can interact with to control various aspects of the Football DMApp. This is primarily used to display statistics/information, and select replays.	This is presented on companion devices in Football at Home and Football Fanzone DMApps.
Football Timeline Navigation	This presents a menu user interface which can be used to seek within the Football DMApp timeline and adjust the Football DMApp layout according to pre-defined bookmarks.	This is presented on companion devices in the Football at Home and Football Fanzone DMApps and is primarily used to facilitate demonstrations.
Football TV Control Menu	This presents a menu user interface which the user can interact to remove or control Picture-in-Picture or replay instances currently running on the main TV.	This is presented on companion devices in the Football at Home and Football Fanzone DMApps.
Google Analytics	This component aggregates user interaction events generated by other DMApp Components and sends them to Google Analytics.	This is a non-displaying component used in the MotoGP DMApp.
HTML Snippet	This presents formatted text-based content on the TV emulator or companion device.	This is used in the Theatre at Home and MotoGP DMApps. Within MotoGP, for example, it is used for static text overlays such as the “MotoGP™” box and replay/event titles.
Image	This presents a static image on the TV emulator or companion device.	This is used in the Theatre at Home and MotoGP DMApps for static graphical overlays, for example channel and brand logos.
IoT Data Fetcher	This component enables live data to be received from the Data Playback Service.	This is a non-displaying component used in the MotoGP, Football at Home and Football Fanzone DMApps.
MotoGP Companion Control Panel	This component presents an interactive control panel on the companion device which enables user interaction with MotoGP content.	In the MotoGP DMApp, the companion control panel can be switched between three different modes during the race: *Leaderboard* – which indicates the position of each rider in the race and allows the user to view additional information about

Name	Description	Comments
		each rider through a swipe-able ‘card’ view which can be shown or hidden beneath each rider’s name. The position of each rider and other information is determined by live data provided by the Data Spooler component. The ‘card’ view also offers a video stream from the rider’s on-board camera, if available. *Events* – which provides a list of notable events which have taken place during the race so far and allows the user to replay them on demand. *View* – which provides a list of available video streams which can be displayed on the tablet companion device and optionally ‘cast’ to a Picture-in-Picture view on the TV emulator. When the race is finished, only the *Events* mode is available.
MotoGP Companion Notification	This presents a pop-up message on the companion devices to inform the user of important information.	In the MotoGP DMApp, this component is displayed at specific times before the race as determined by the pre-authored timeline, for example to remind users to select their favourite rider, or to signal that the race is about to start.
MotoGP Companion Panel Switcher	This presents an interactive menu on the companion device which enables the user to select between the different modes offered by the Companion Control Panel component, showing which mode is currently selected.	
MotoGP Companion Stats	This presents a table of lap time statistics for each rider on the tablet companion device.	In the MotoGP DMApp, the contents of the table are determined by live data provided by the Data Spooler component. Part of the table highlights statistics for a favourite rider if one has been selected by the user.
MotoGP Companion	This presents a title bar on the companion device which	In the MotoGP DMApp, the drop-down menu enables the

Name	Description	Comments
Top Bar	includes the current status of the DMApp and provides an interactive drop-down menu to customise and control the experience.	selection of TV Graphics Scale, Presentation Style, Audio Presentation and Favourite Rider.
MotoGP Inside MotoGP Panel	This component is a self-contained interactive video-on-demand player for the companion device which enables the user to browse and watch different video clips.	In the MotoGP DMApp, this component is presented during the build-up stage before the race starts and allows multiple users to independently watch different video clips taken from three categories: Tutorial, Technical and Catch-up.
MotoGP Laps Remaining	This graphic presents the number of laps remaining, changing as the race progresses.	In the MotoGP DMApp, the number of laps remaining is determined by live data provided by the Data Spooler component.
MotoGP Leaderboard	This presents the MotoGP leaderboard on the TV Emulator, indicating the current position of each rider in the race, and highlighting changes as the race progresses.	In the MotoGP DMApp, the Leaderboard component is overlaid on the main race video. The position of each rider is determined by live data provided by the Data Spooler component. The Leaderboard presentation is also determined by the Presentation Style and TV Graphics Scale selected by the user. In addition, it can be arbitrarily triggered to show gap times between any two riders.
MotoGP PIP	This component plays out 'Picture-in-Picture' video with surrounding overlay graphics on the TV emulator or companion device. This component uses an instance of either the Video or Video Panorama component depending on the media type.	In the MotoGP DMApp, Picture-in-Picture video streams can be overlaid on part of the main race video on the TV emulator. Multiple video streams can also be shown on the tablet companion device.
MotoGP Replay	This component presents a replay of a race event on the TV emulator, including a sequence of graphics and video clips.	In the MotoGP DMApp, this component may be triggered by the pre-authored timeline to display replays during the race, or interactively by selection of an event in the Companion Control Panel during or after the race.

Name	Description	Comments
MotoGP Spooler	This is a data spooler component which enables live data to be distributed to DMApp Components which require it.	This is a non-displaying component used in the MotoGP DMApp.
MotoGP TV Control	This component changes configuration options on the TV emulator in response to updates from the timeline service. These include whether user-controlled Picture-in-Picture components are enabled, and hiding some graphical elements during Race Review and Inside MotoGP modes.	This is a non-displaying component used in the MotoGP DMApp.
Prime	This is a generic graphics overlay component used for realising any kind of non-interactive on-screen graphics. It uses WebGL to ensure smooth animations. The scene representation parser is compatible with ChyronHego Prime broadcast graphics authoring software, which means 2-IMMERSE graphics can re-use existing broadcast graphics or vice versa.	In the Football at Home and Football Fanzone DMApps it was used for implementing graphics as score-clock, lower thirds, bill-boards etc.
Scrolling Text	This presents scrolling synchronised text, such as the script of a play. It will include synchronised buttons to show actor and other information.	The provision of a synchronized scrolling script on the TV emulator is a key feature of the Theatre at Home DMApp.
Text Chat	This enables text chat to be presented and displayed, including conversation history.	Text chat is a key feature of the Theatre at Home DMApp and is available on both the TV Emulator and Companion at all times.
Text Chat Controls	This provides a UI for posting messages to the Text Chat component.	This is used in the Theatre at Home DMApp. Text Chat Controls are only available on the companion device.
Title Card	This presents an opening screen for the entire experience.	This is used within the Theatre at Home DMApp.
Video	This is a video player which is capable of playing out video and audio on the TV emulator or companion device, with stereo audio. Video playback can be synchronized within and	This component is used in all 2-IMMERSE DMApps for video playback on TV and companion devices.

Name	Description	Comments
	between devices. It can play media in HLS, DASH, and other media formats. It can also be used to play live video acquired from a local video capture device.	
Video Chat	This enables multi-party audio/video chat between households (contexts). It co-ordinates the other Video Chat components and optionally presents video thumbnails which represent other locations in the call.	This is used in the Theatre at Home DMApp. The trial only involved pairs of households and so the video thumbnail presentation was not presented.
Video Chat Controls	This provides a control UI for a Video Chat session, including microphone and speaker level controls and an optional 'push-to-talk' button.	This is used in the Theatre at Home DMApp. The Video Chat Controls are only available on the companion device.
Video Chat View	This presents the remote video stream of currently active speaker in a Video Chat session and a picture-in-picture view of the local camera stream.	In the Theatre at Home DMApp, Video Chat is only presented before and after the show and during the interval. The Video Chat View is always shown on the TV Emulator.
Video Panorama	This component is an interactive 360 degree video player which plays out panoramic videos on the TV emulator or companion device.	In the MotoGP DMApp, 360 degree video is available for one of the on-board cameras during the live race and is presented in the same way as Picture-in-Picture video streams. The video can be interactively panned on the companion display, and when shown on both the TV emulator and the companion simultaneously, the view position will be synchronised between the two devices.

Table 5: DMApp Components developed within the 2-IMMERSE project

7 Production Tools

For the Final Release a new tool has been added to the production suite: the **pre-production tool**. This tool is intended to help create the initial storyline of an experience: assembling all media and DMApp components and story elements that are available before the live event happens.

In addition, based on user feedback (described in deliverable D3.4 (8)) the functionality in the *live triggering tool* from the previous release (deliverable D2.4 (3)) has been optionally split into two parts:

- The **Triggering tool** is used to prepare events before showing them, very similar to how the old live triggering tool worked: adding names, captions and other parameters to the events before showing them. The events are then either shown directly or enqueued for the trigger launcher.
- The **Trigger launcher** is a tool to allow exact control over when an event is shown (and removed again). It presents a set of buttons that show the events prepared and enqueued by the trigger tool and activates these events when that button is pushed.

The trigger tool can be run standalone, in which case the functionality is very similar to the old live triggering tool: after preparing an event the director can insert the finished event into the document. But for true live operation the trigger tool and trigger launcher will be operated by two people: the trigger launcher operator has a function similar to the GFX operator in that they prepare the event to be shown and push it to the trigger launcher (operated by the director). The director's sole responsibility is now to press the button at the right point in time to make the event show.

In Figure 7 you can see the whole triggering setup in operation during the FA Cup Final at Wembley Stadium: the trigger tool (called *Match GFX Config tool* in the caption), trigger launcher (*Stream Deck Trigger Tool*) and preview player (*Platform Preview player*).

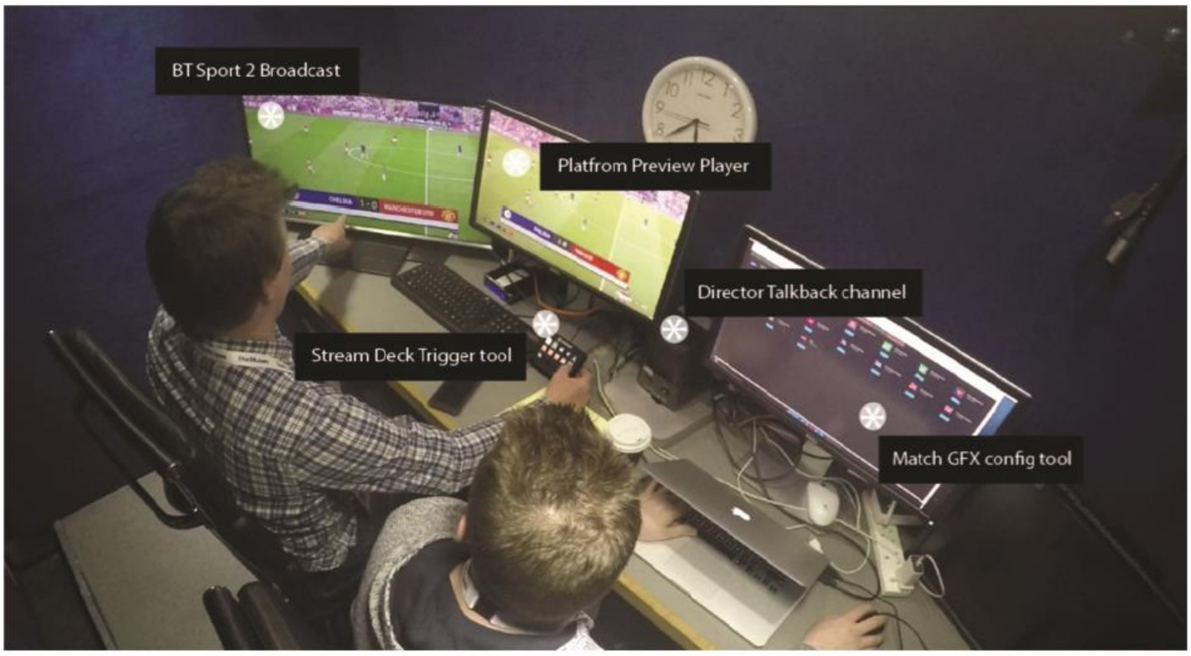
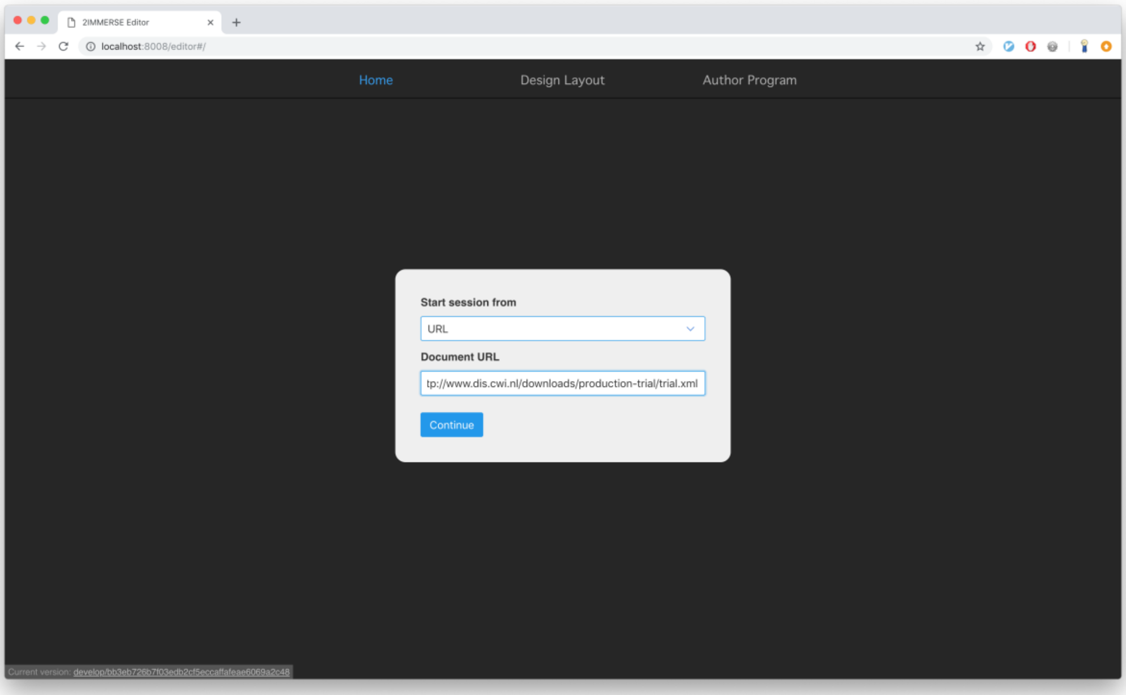


Figure 7: Triggering Tool and Trigger Launcher in operation during the FA Cup Final

7.1 Pre-production tool

The pre-production tool provides a solution for creating documents which can be played back on the infrastructure. It attempts to achieve this by combining a hierarchical editing approach, which enables the user to compose presentations by means of sequential parallel composition of media elements, with the traditional non-linear approach based on arranging elements on timeline tracks.

When starting the editor, the user has to provide a starting document. This document can either be completely empty or already contain screen layouts for preview devices, assets and media elements and also a basic program structure.



Start session from

URL

Document URL

tp://www.dis.cwi.nl/downloads/production-trial/trial.xml

Current version: develop/bb3eb726b7f03edb2cf5eccaffafeae6069a2c48

Figure 8: Starting the Pre-production Tool

Once the user has chosen a document either by uploading it from their machine or providing one via an URL, they are presented with the layout designer screen. If the starting document did not contain any predefined preview devices and layouts, this screen will be empty. The user can then manually add preview devices. These preview devices fall into two categories: communal devices and personal devices. Communal devices are usually TVs and are rendered in landscape orientation. Personal devices are phones and tablets and can be rendered in either portrait or landscape mode.

After adding the desired number of preview devices, the user can start designing their layout. The layout designer of the pre-production editor only supports the most basic type of layout, which is composed of a series of screen regions which divide the screen into a non-overlapping rectangles. The user can choose to split the currently selected region either horizontally or vertically to achieve their desired layout. More complex layouts (e.g. with overlapping elements or elements with a specific z-order) cannot be designed in such a way but rather have to be designed by a professional and added to a starter document. In case the layout specification is available as a file, it can also be directly loaded as a template by the user by selecting the corresponding button.

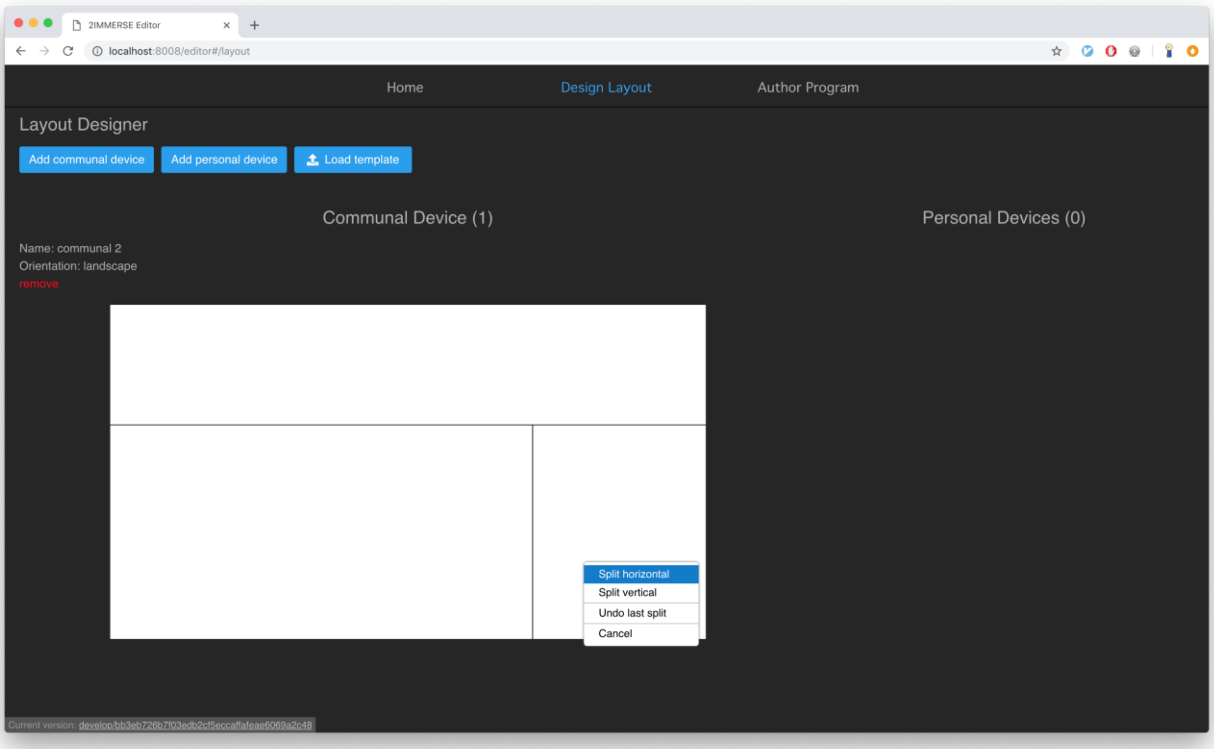


Figure 9: Pre-production Tool layout designer

A predesigned layout with two preview screens loaded from a starter document is rendered in the layout designer and can be inspected before proceeding. In this case, the user cannot modify these layouts, but they are able to add more preview devices or remove existing ones if need be. For easier orientation, each region in a preloaded layout has a different colour.

After this step, the user is ready to design the high-level program structure using the program author screen. This screen allows the user to segment the program into named chapters which can have sub-chapters. Chapters on the same level are played one after the other from left to right and in parallel with parent chapters. The main advantage of this segmentation is the fact that the user can for instance create a top-level chapter for a football match and add all the media items which shall be visible during the entirety of the match (e.g. the channel bug) to this top level chapter and then create two sub-chapters, one for each half of the game without having to worry about adding the channel bug separately to each of these sub-chapters.

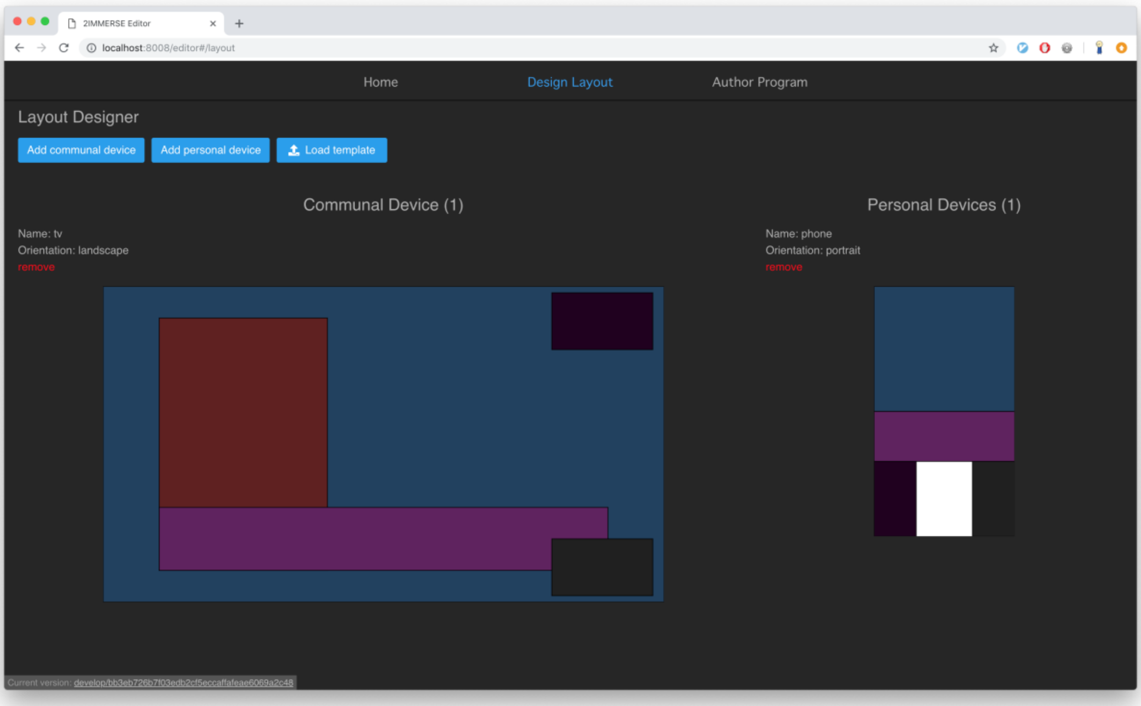


Figure 10: Pre-production Tool pre-loaded layouts

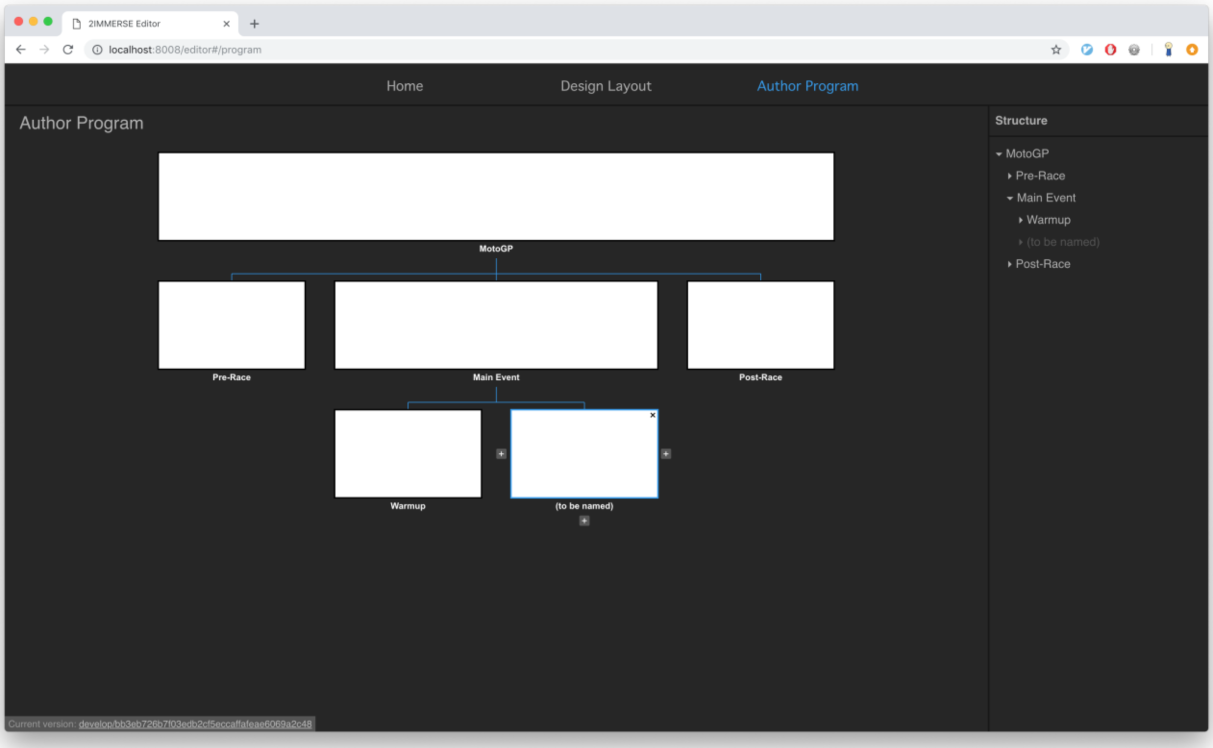


Figure 11 : Programme Authoring in the Pre-production Tool

New chapters can be added before, after or below a selected chapter by hovering over it and selecting the button with the plus sign on the appropriate side. New chapters are initially created without a name. A name can be assigned by selecting the label and typing the name into the dialogue. Chapters can also be deleted by selecting the X icon in the top right of each chapter. An exception applies to chapters which have sub-chapters and the root chapter, these cannot be deleted.

From here, if the user clicks on the white area of a chapter in the program, they are redirected to the timeline editor. This screen functions very much like a traditional non-linear video editor like Adobe Premiere. One can either drag elements directly from the asset list in the bottom right of the screen onto the desired timeline track or drop it into the desired region in the preview screen. The dragged elements are then added to the corresponding track. If an element is dropped over an existing element, the new element is inserted before it. To insert a new element at the end of the track, it must be dropped over the plus-symbols at the end of each track. Elements which are dropped onto the preview screen are always dropped at the end of a track. The timeline presents the sequence in which the elements will be played. That means, in order to get a better understanding of the actual sequence, also elements from child chapters must be rendered on the timeline. This also implies that when selecting the root chapter, the rendered timeline view represents the actual playout sequence of media item as if it had been designed in a non-linear editor. It is important to note in order to avoid inconsistencies and nondeterminism in the data structures, that elements can only be modified in the chapter to which they were assigned originally.

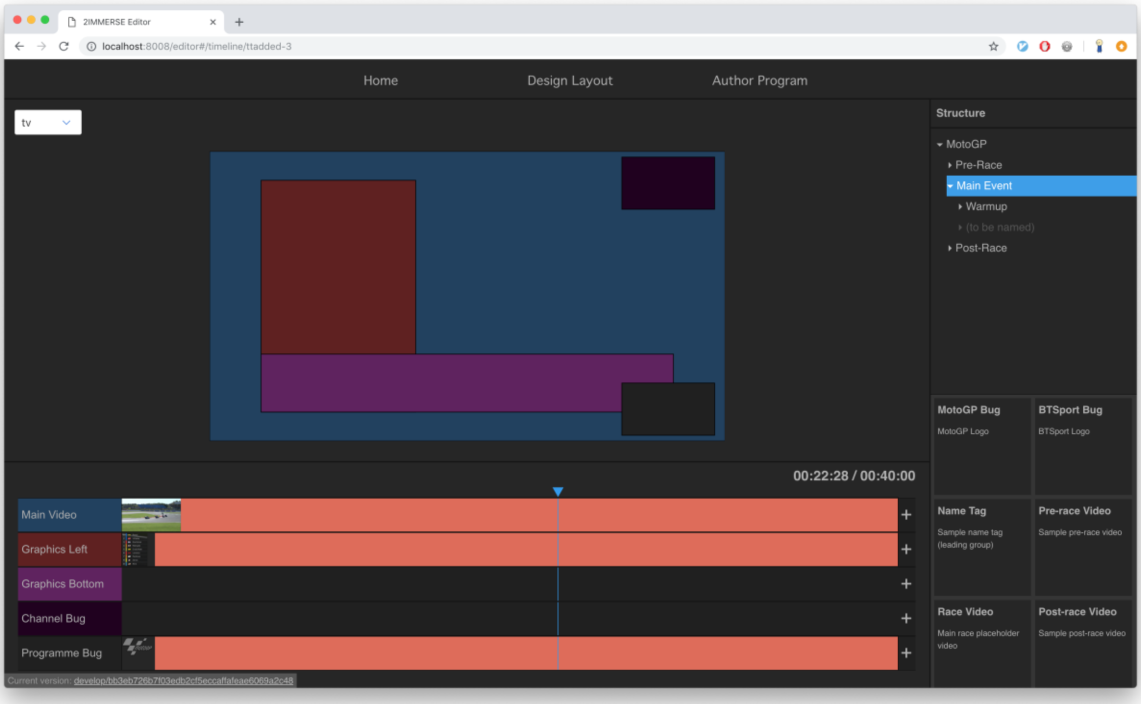


Figure 12: Pre-production Tool timeline editor

For instance, in Figure 13 the three video elements have been added to the “Main Video” track in the chapters “Pre-Race”, “Main Event” and “Post-Race” respectively. We are looking at the timeline from the perspective of the “MotoGP” chapter, which then presents these videos in a sequence, but since they have been added in different chapters, they cannot be modified here and are thus greyed out. To modify them, one needs to select the corresponding chapter. Elements can be deleted from a track by selecting them and dragging them away from the track. Clicking an element brings up a prompt which allows the user to modify the duration of a track. The concept of duration is an important one to expand on in this context. In our infrastructure, there are really two types of elements, those who have an intrinsic duration, such as videos, and those who don't, i.e. images. The user can drag an image onto a timeline and assign it a specific duration, but they can also drag it onto a timeline and not assign it a duration. What happens in that case is that the element stays visible until all of its sub-chapters have finished playing. In the example above, we have a MotoGP bug assigned to the region "Programme Bug". This bug has not been assigned a duration, therefore it stays visible until all videos have been played. This becomes immediately obvious when looking at the timeline view for the root chapter. From this also follows that a timeline track can only contain a single element without an intrinsic duration.

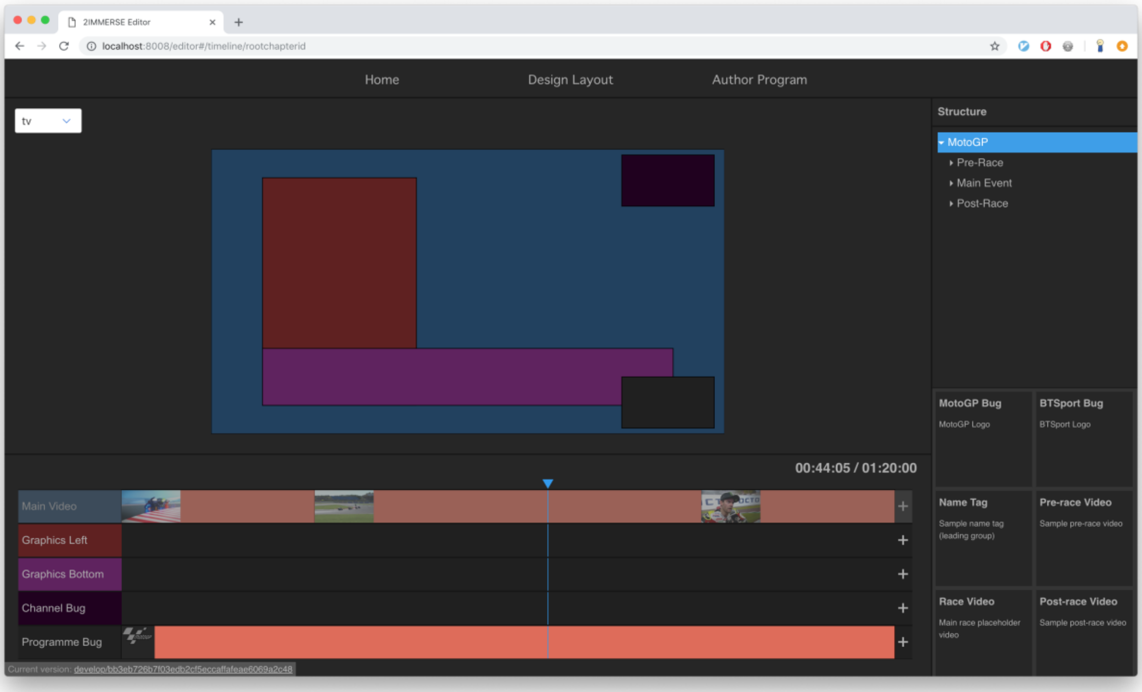
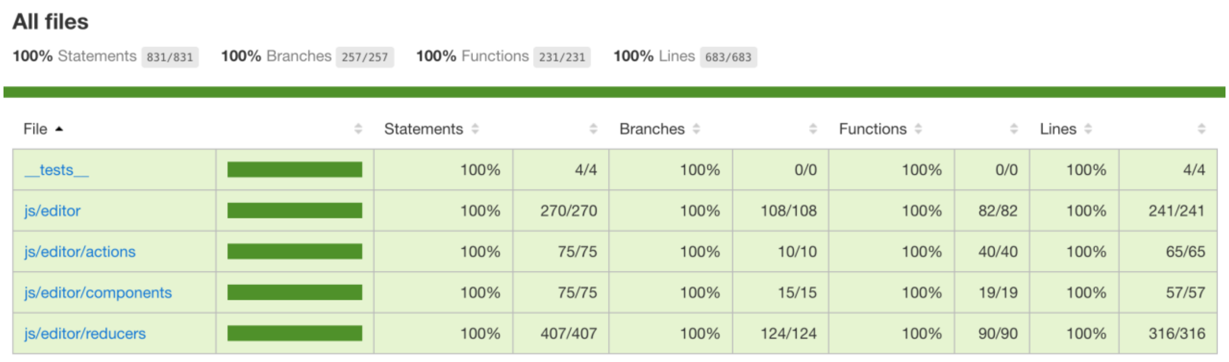


Figure 13: Pre-Production tool timeline hierarchy

In terms of architecture, the frontend maintains its own data structures which are managed by immutable and exclusive access to a global data structure. The user interface is transparently updated if the global data structure is updated. This has the advantage that there is a single source of truth and the impossibility of data races and inconsistencies. The global data structure is synchronised to the server on each update operation, which serialises the data immediately to the XML document format. All of this is managed by the internal business logic of the frontend, which is kept strictly separate from the presentation logic, which is solely responsible for rendering the user interface. The correctness of the business logic is ensured by a testing environment comprised of 18 test suites, containing 358 test cases with 789 assertions, achieving 100% statement as well as 100% branch coverage.



All files

100% Statements 831/831 100% Branches 257/257 100% Functions 231/231 100% Lines 683/683

File ▲		Statements		Branches		Functions		Lines	
__tests__	🟩	100%	4/4	100%	0/0	100%	0/0	100%	4/4
js/editor	🟩	100%	270/270	100%	108/108	100%	82/82	100%	241/241
js/editor/actions	🟩	100%	75/75	100%	10/10	100%	40/40	100%	65/65
js/editor/components	🟩	100%	75/75	100%	15/15	100%	19/19	100%	57/57
js/editor/reducers	🟩	100%	407/407	100%	124/124	100%	90/90	100%	316/316

Figure 14: Pre-production Tool testing environment

7.2 Frontend

The main changes in the Triggering Tool have been support for the two-tool live setting as described above. Figure 15 shows a screen shot of the tool as configured for live football. One of the new features is the status bar at the top showing which events have already been triggered by the director and which ones not yet. The trigger tool operator also has the option of removing those.

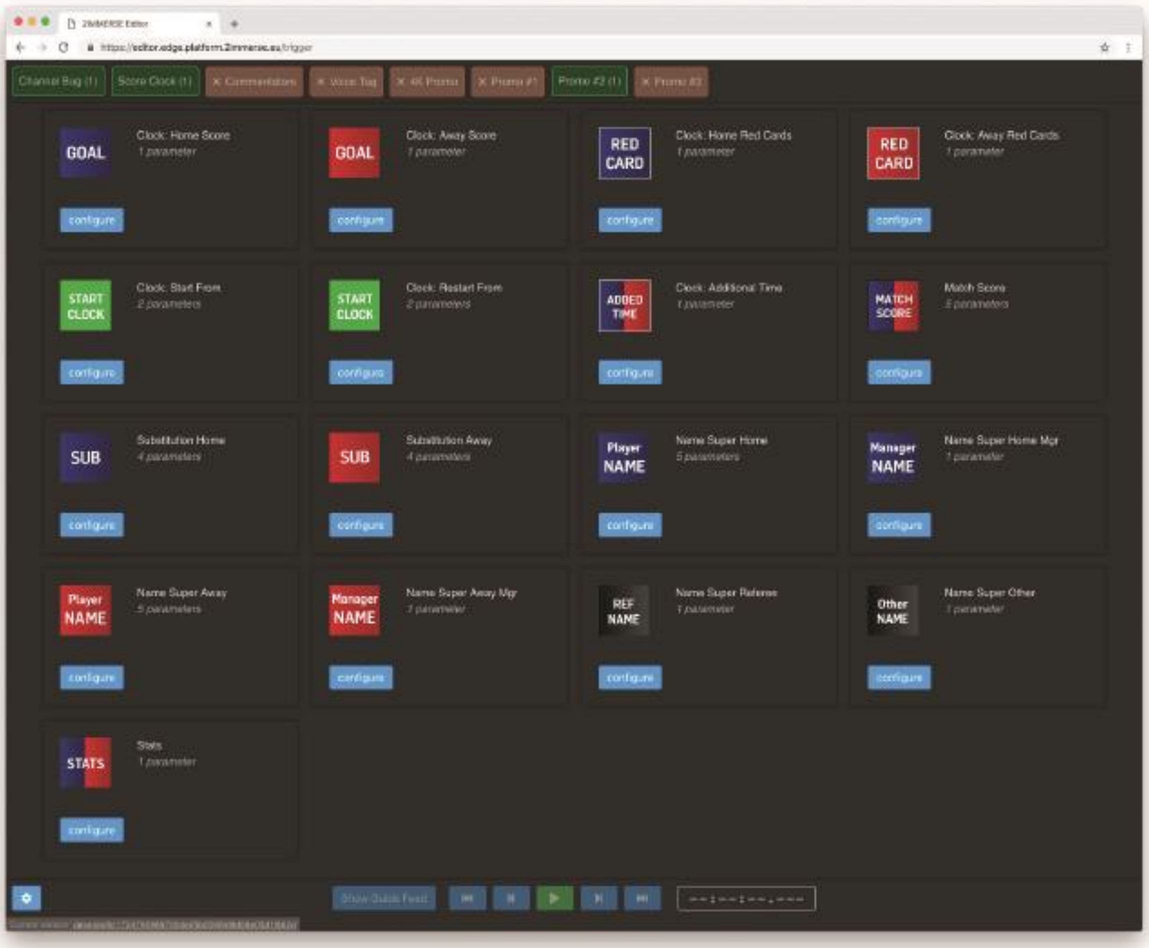


Figure 15: Triggering tool configured for live football

Beside these changes, there have also been some cosmetic changes since the previous release, based on user feedback: preview images have become more icon-based because this better fits the mindset of football TV directors. Things like sizes of images has also been changed to forestall having to scroll through the document as often.

The trigger launcher is a completely new tool. It can be operated in a web browser, for example on a touch screen such as a tablet, but in general is expected to be used with special hardware: the Elgato Stream Deck. This is a small hardware box with 15 physically clickable buttons with programmable images on them.

The trigger launcher assigns events to buttons as they get forwarded from the trigger tool.

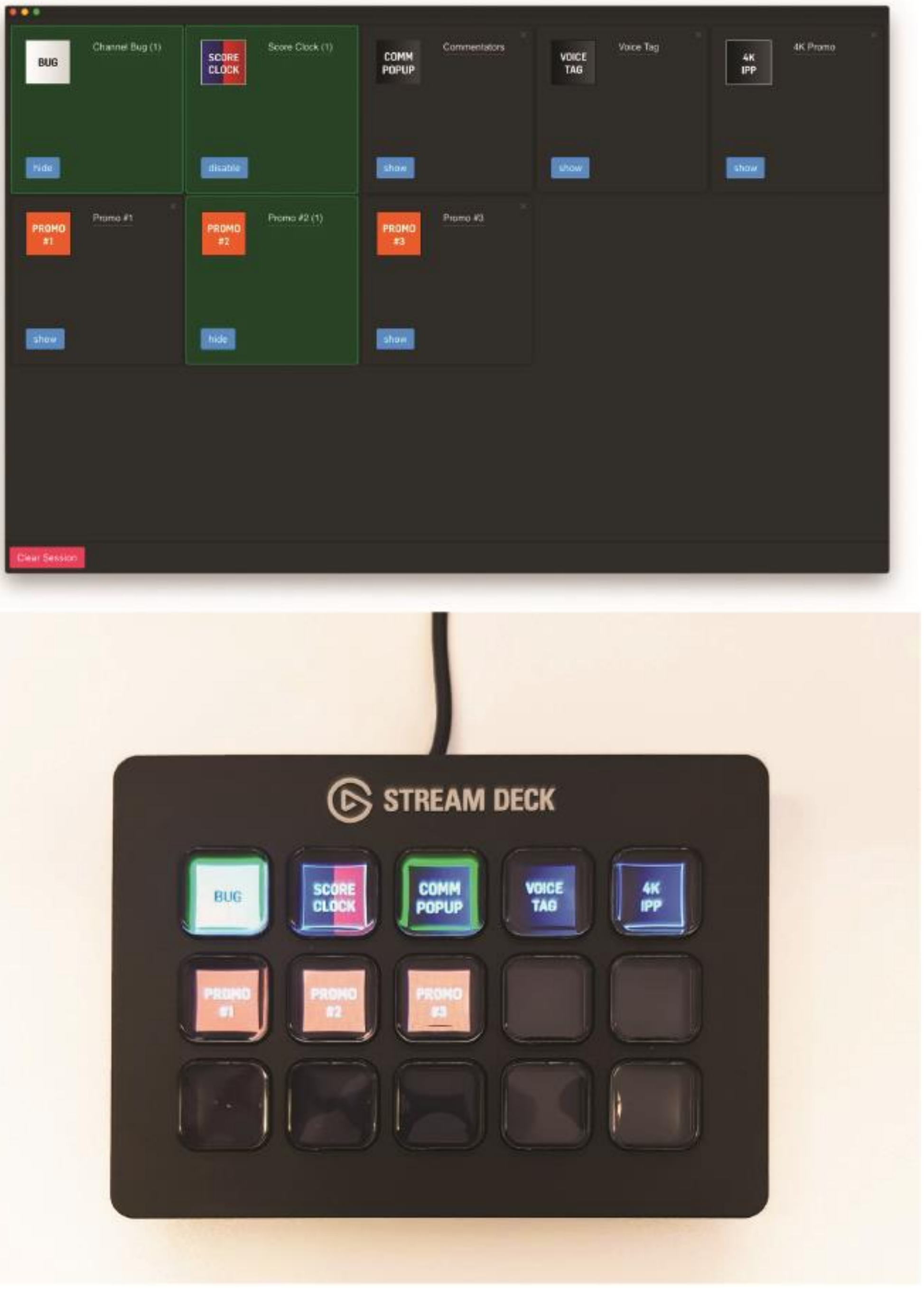


Figure 16: Stream Deck and the graphical UI of the trigger launcher

Figure 16 shows how the Stream Deck and the graphical UI of the trigger launcher work together: all the information is available on both the screen and the buttons. Where the green highlight on the UI shows that an event is currently active this is can also be seen from the green ring around the button on the Stream Deck. This setup with physical buttons is familiar to TV directors and hence leads to a reduced workload.

7.3 Backend

The editor backend is the engine implementing all of the functionality needed by trigger tool, trigger launcher and pre-production tool. The following major changes have been made since the previous release:

1. **Live Broadcast support**: For the previous release only near-live editing was implemented, and therefore only a single timeline service instance was kept synchronised with all the changes to the document: only the preview player for the director. For this release the timeline service instances of *all* live viewers need to be updated whenever an event is inserted into the experience. The update mechanism for timeline service instances has been implemented using websockets (in addition to REST calls) to enable broadcasting of updates to many viewers. In addition, viewers "tuning in" late are provided with the current temporal position of the document, so their timeline server instance can skip all the events that have happened in the past, and they see things like the match clock and video at the right time position.
2. **Preview video feed**: the preview player (watched by the director during the live triggering) shows a different video feed that is coming directly off the camera mixer, while the home viewers watch a DASH feed that is encoded, possibly encrypted and distributed through a CDN. Both preview player and viewer players play back the same timeline document, but because the direct feed seen by the director is a couple of seconds ahead of the feed seen by the viewers the inserted events are available at the viewer timeline service instances in time. The editor backend manages which video feed is shown in the different timeline documents.
3. **Trigger launcher support**: the previous release had the concept of *triggerable events*: xml snippets with parameters that could be inserted into the timeline by the live triggering tool. The new release adds the concept of *complete events*: those same snippets after the parameters have been filled in (using the trigger tool frontend) but before they have been activated by the trigger tool launcher. Support has also been added to events to ensure that their identity is maintained (from being *triggerable* to *complete* to *playing* to *finished*) so the user interfaces on the front ends can show the status of the event.
4. **Pre-production tool support**: the document structure has been extended to allow editing of chapters and media items (adding, deleting, renaming), and an API has been added to allow the fron end to inspect the current structure and to make those changes.
5. **Python 3**: the backend has been ported to Python 3, to ensure that it remains usable after Python 2.7 is no longer supported (7).

8 Platform Sustainability

This section describes updates being made to the platform and client applications to support their sustained use after the end of the project.

8.1 Reproducibility of 2-IMMERSE platform services

As mentioned in previous deliverables (D2.2 (1), D2.3 (10)), the main 2-IMMERSE platform instance, used for development, testing, experimentation and demonstrations, is hosted on AWS. To ensure success in post-project platform exploitation, it is crucial that the deployment of the platform is easily reproducible. There are a number of challenges to achieving this:

1. Current AWS-deployment to be decommissioned after project ends

This particular deployment has a finite lifetime; it is expected that all AWS-hosted platform instances will be decommissioned at the end of the project.

The consortium has worked on lean versions of the platform that can be used for standalone deployments. Whilst this type of deployment is sufficient for small demonstrators, it does not include the full set of platform capabilities and its scalability is limited. Therefore, for continuous post-project 2-IMMERSE platform exploitation, the consortium partners and third-parties at large need a way to continue to demonstrate and develop 2-IMMERSE technology and user experiences.

2. PaaS deployment environments agnosticism

It is entirely possible for project partners and third-parties wishing to run a separate instance of the platform use a PaaS other than AWS. However, the use of specific AWS infrastructure services prevents the platform from being PaaS-agnostic.

3. Manual bespoke configuration of some service components

Some service components have been configured manually with settings that are AWS or 2-IMMERSE-platform instance specific. For example, DNS entries for some services have been manually added. Further, some of the configuration steps lack documentation and config scripts. This is also compounded by missing knowledge about required configuration steps e.g. DNS records for existing instances

4. Legacy services in platform configuration

Legacy services still in the platform configuration following migration from Mantl, potentially would have led to confusion for third parties wishing to deploy their own instances. Mantl used the Registrator service to enable contaioner service registration

Work has been undertaken in the last quarter to solve these issues and include the following:

1. Generalise sys-ops and dev-ops scripts so they can be parameterised.
2. Prove that we can deploy an independent instance on alternative PaaS infrastructure successfully.
3. Work to migrate dev-ops configuration e.g. runners from GitLab/GitLab Docker registry to GitHub, TravisCI and DockerHub.
4. Trimmed the stack of 2-IMMERSE services to remove superfluous components.
5. Addition of infrastructure configurations to 2-IMMERSE open source repositories (deployment subdirectory documents everything infrastructure- and stand-alone-deployment related and includes scripts).
6. More detailed documentation (for third parties) of the infrastructure architecture (service registry, message buses, databases, DNS, load balancers, proxies etc.).

7. Provision of alternative solutions to replace AWS specific services/functions such as DNS, S3 bucket, load balancer etc.

Figure 17 shows a deployment topology of the 2-IMMERSE platform on an alternative cloud platform, Open Nebula. The deployment has necessitated trimming the stack of 2-IMMERSE services to remove superfluous components and replacing AWS infrastructure services with alternative solutions. For example, MINIO is used a file-service for implementing the functionality of the Origin Server. Bind9 is used as a mechanism by the Rancher container management system to update DNS server entries when services are started/stopped. The configuration includes 3 VM hosts for operational and management services (filebeat, healthcheck, elk, etc). For the 2-IMMERSE service stack, 3 more VMs (hosts A, B and C) are used to deploy Edge, Test and Production versions of the platform.

Load balancing is achieved by 1) the DNS server cycling through alternative DNS entries for a service and 2) by using NGinx as a proxy for service requests. A more detailed view of the service request routing and load balancing can be seen in Figure 18.

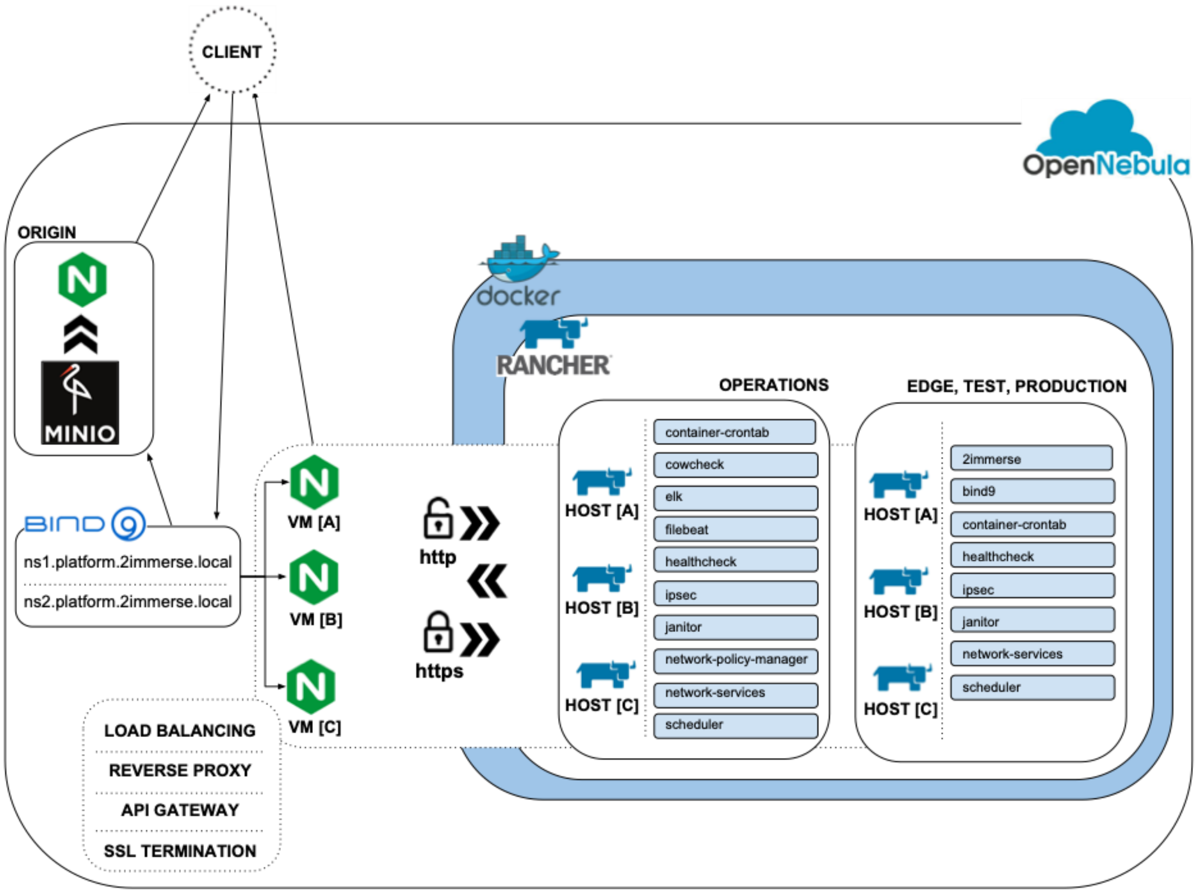


Figure 17: Deployment of 2-IMMERSE stack of services on OpenNebula at BBC

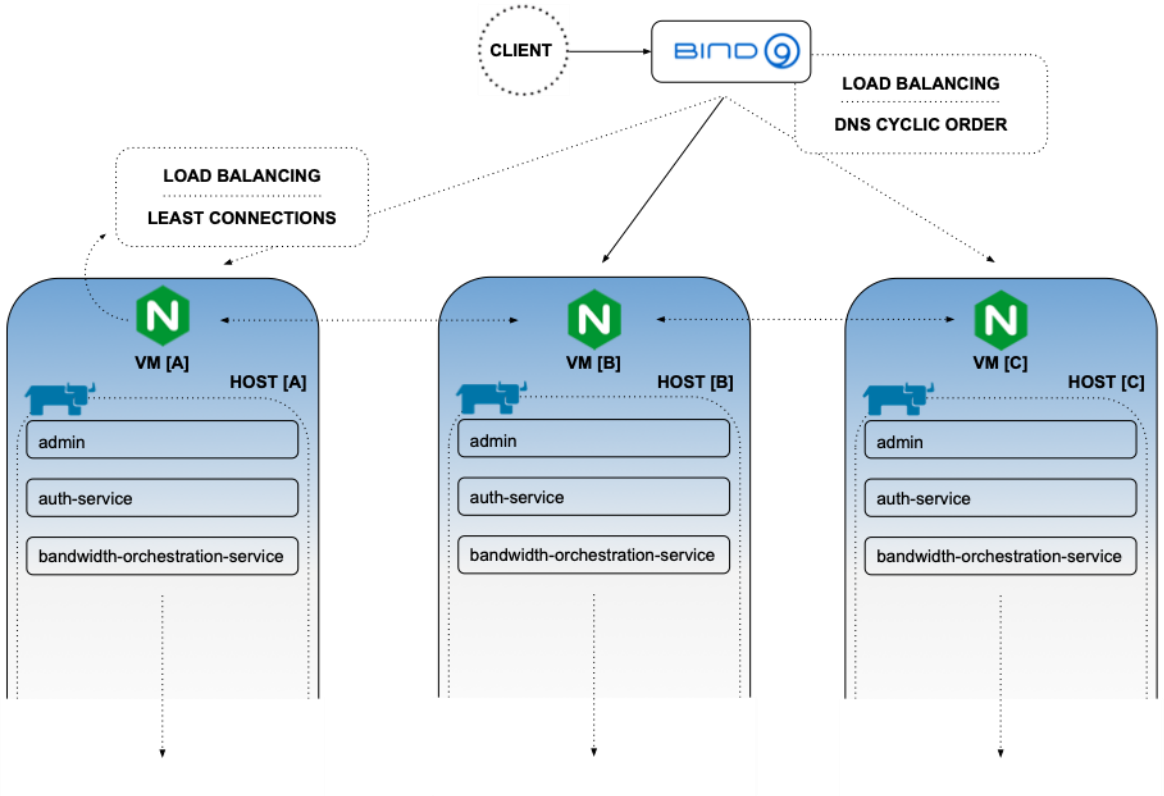


Figure 18: Load balancing via DNS and local proxy

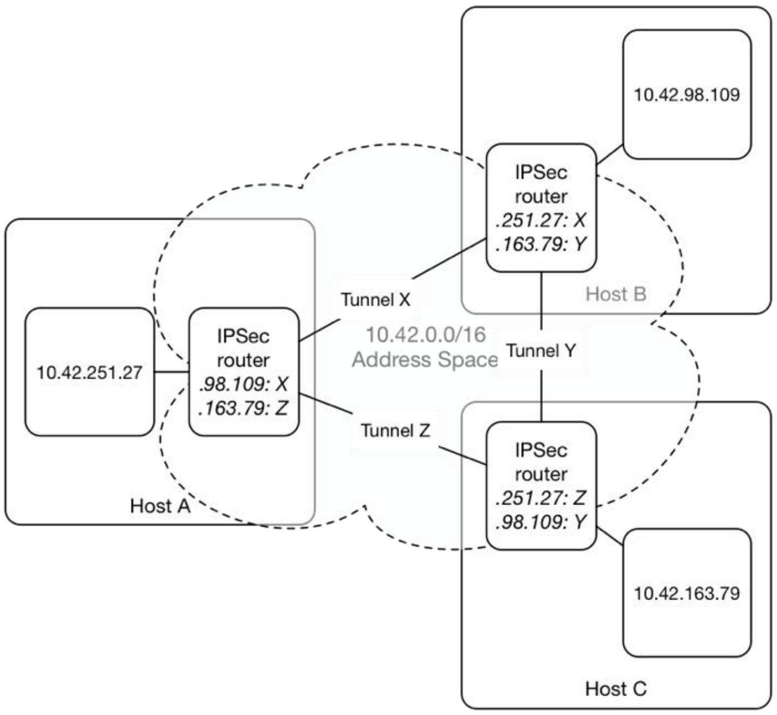

10.42.98.109
IPSec router .251.27: X .163.79: Y
Host B
Tunnel X
10.42.0.0/16 Address Space
Tunnel Y
10.42.251.27
IPSec router .98.109: X .163.79: Z
Tunnel Z
Host A
IPSec router .251.27: Z .98.109: Y
10.42.163.79
Host C

Figure 19: Network overlay over VM hosts created by Rancher

In summary, to make sure the platform can be reproducibly deployed on a host of different platforms, we have generalised the 2-IMMERSE infrastructure and improved its configurability. This is in the interests of platform sustainability and to make the software more accessible to third parties. In doing so, we have also improved the usability of the platform. The platform is easier to maintain as a result of improved documentation, remedial action and spring cleaning.

8.2 Reproducibility of client builds

All 2-IMMERSE services are built from source by a CI/CD pipeline into Docker images and uploaded to a Docker registry. Now the same engineering best practices have been adopted for client software builds. Whilst the results of the client builds are not Docker images themselves, we have adopted Docker images to create reproducible build environments for the client software. Once built, these “build-images” can be pushed to the same Docker registry as the 2-IMMERSE services or alternatively archived.

The Client API (the biggest client-side component) is now built entirely within a Docker container and the resulting artefacts are written to a filesystem volume shared with the host. Similarly, the two Android unified launcher applications (.apks) are built inside a Docker container containing the Android SDK and toolchain, allowing specific API levels to be targeted. The Unified Launcher web applications for TV and companion devices are also built inside a Docker container. Even the 2-IMMERSE TV emulator firmware (a small Debian Linux Distro) is built entirely within a Docker container, generating a hybrid .iso image.

A key benefit of this approach, is the ability to build the software on any host platform supporting Docker (Windows, MacOS, Linux etc.) without having to install anything on the host (other than Docker itself). This lends itself well to automated builds on CI/CD infrastructure. It also makes it easier to lock down the versions of build tools, SDKs and the OS itself, thereby preventing builds from breaking over time by ensuring the software we use today continues to be available in the future.

9 Platform Evaluation

Our goal is to make the 2-IMMERSE software good enough that people will want to adopt some or all of it. It has to be valuable and accessible to organisations outside the 2-IMMERSE consortium if we wish to foster a community of practice around immersive multi-device broadcasts.

The work done to prepare the software for open source release has addressed issues of technical debt, documentation and ease of use, however in order to know if the platform is any good at orchestrating multi-screen experiences, it is important to gather quantitative metrics of its performance. Given the time and resource available, we performed three targeted platform evaluations, each measuring a key characteristic:

1. Using real HbbTV2.0 devices
2. Cloud-based synchronisation accuracy
3. Network bandwidth orchestration effectiveness

These evaluations were chosen deliberately because they measured the effectiveness of the platform in ways that were not scoped by the service trials, yet were scoped core functionalities of the system. The evaluation results are presented in the following three sections.

9.1 HbbTV implementation

9.1.1 Overview

When 2-IMMERSE started, the HbbTV consortium including IRT and BBC had released a new specification, namely HbbTV 2, that includes support for multiscreen services that are built around a TV with one or more companion devices. 2-IMMERSE picked this specification up and integrated it in its platform architecture and implementation. See also D2.1 (2) chapters 5.3, 6.1.2 and 6.1.6.

Due to uncertain availability of devices during the lifetime of the project and limitations like, for example, the number of available video decoders, 2-IMMERSE decided to implement its own terminal devices that could be designed to match the requirements of the service trial scenarios, but making use of HbbTV 2 protocols which theoretically would allow for running a subset of the trial applications on TV sets that implement HbbTV 2 later on.

Throughout the project IRT worked with TV manufacturers on testing and showcasing HbbTV 2 features through interoperability events, bilateral testing and public presentations on trade fairs like IFA and IBC. Part of this implementation has been reused and integrated into the 2-IMMERSE platform.

During the last year of 2-IMMERSE, the platform and client implementation itself was tested with HbbTV 2 prototype and production devices. As an example, service the MotoGP trial application was chosen as it was the most advanced 2-IMMERSE application by the time. This report documents the outcome of this work including which modifications were necessary to run on HbbTV devices and which limitations of the service trial must be taken into account when running on these TVs.

A demo video of the application running on a TV can be watched on the 2-IMMERSE blog (11).

It was presented as an HbbTV 2 example at IRTs stand at IFA Berlin 2018, IBC Amsterdam 2018, Münchner Medientage (Munich) and the HbbTV Symposium.

9.1.2 Interoperability Tests

As a first step to test the basic performance and behaviour of the 2-IMMERSE client API, one of the 2-IMMERSE test cases was used in an official HbbTV interoperability event at IRT in March 2018 and tested on all available TVs. The test case with the internal ID 108 loads the client API and presents 3 DMApp components, an MP4 video, a scrolling script and a picture slide show. Most devices present in the workshop were able to run this test successfully. Table 6 shows the anonymised details. As the MotoGP content is protected to match the requirements from Dorna, also the support for clear key encryption was tested which is also a new feature in HbbTV 2.

Manufacturer	Model	2-IMMERSE 108	W3C EME Clear Key
Middleware vendor.	Nvidea Shield (Android)	Failed.	No support, not tested.
TV1	?	Success.	No support.
TV2	Production	Success.	Failed, there is support but still issues
TV3	Development	Success.	Failed.
TV4	Development	Failed, outdated browser but HbbTV 1.5 compliant.	No support.
TV5	?	Success.	No support.

Table 6: Anonymised results of TV interoperability Tests

The results show that 4 of 5 current TV implementations successfully run the selected test case. One TV implementation failed in this test. The manufacturer explained that the browser was not updated recently and that developments still only targets HbbTV 1.5. The use of clear key encryption as defined by HbbTV was not supported by the manufacturers present at the workshop. For the HbbTV trial with the MotoGP service this means that all content running on TV must be local and can't be stored on the origin server.

The MotoGP trial application was tested successfully on Samsung TVs with production firmware on 2018 hardware models.

9.1.3 Modifications for and limitations with HbbTV 2

This section first describes all modifications which were required on the client implementation and then describes the limitations that restrict a feature of the MotoGP service trial.

- Playback of adaptive bitrate video with DASH.js requires W3C Media Source Extensions which are available on some newer devices but not specified by HbbTV. The playback of video was changed to use the MPEG DASH player of the HbbTV terminal. For ease of implementation and testing a separate video DMAPP component was developed.
- Separation of audio and video into separate media presentations. To be able to flexibly chose and mix different audio sources, video and audio is delivered and presented through different components. This is not supported by HbbTV 2 (only for limited cases like broadcast video with IP audio), so video and audio had to be mixed.
- Application-to-media synchronisation is using a timeline that is driven by the "start" of a timeline document. Driving the timeline can be delegated to a media item, in which case the playtime or currentTime value is used to determine the current position on the timeline. In HbbTV 2.0 a concept of timelines for different media types is introduced which has been adapted from DVB CSS. For the HbbTV adaptation of the MotoGP trial this concept has been used. The benefit for on demand media is limited but it is necessary if the content would be streamed or delivered via broadcast.

- Media synchronisation with the companion device a.k.a. inter-device synchronisation in HbbTV is media centric, i.e. only the timeline of a chosen master media can be exposed via the protocols. As the 2-IMMERSE implementation is using a timeline associated with the timeline document where media items are associated to this had to be changed to always have an assigned master media. An alternative would be to use the app 2 app channel, but this would not allow for a precise media synchronisation.
- For synchronisation of multiple streams on TV, called multi-stream synchronisation, HbbTV uses a different approach as chosen by the 2-IMMERSE client implementation. In 2-IMMERSE a sync controller checks that slave media is in sync with the master media by changing play speed of the slave media. In HbbTV the device has to control that media playback is in sync, to achieve that it gets the correlation between master and slave media from the application. However, as there are no devices yet supporting more than one video stream, sync for PIP on TV has not been modified yet.
- The implementation is using the application to application channel for communication between TV and companion side. Message are rather short but too frequently for the TV implementation and tests have shown that messages get dropped. Therefor the app2app channel was modified to collect messages and send them in bunches at a lower and defined frequency.

The following is a list of features that don't work with HbbTV as they are not specified, not required or optional in the specification and not implemented by current TVs.

- Picture in picture: HbbTV only requires one decoder, supporting one video, audio and subtitle track to be displayed concurrently. More video decoders are optional, and availability can be signalled to applications.
- Mixing multiple audio tracks, not possible with HbbTV 2.0.1, but 2.0.2 includes Next Generation Audio, which allows for more flexible audio compositions.
- 360 presentations, 360 is not defined natively for HbbTV. However, there are experiments and even first services with server-side rendering of 360 scenes making use of W3C MSE if available on devices.

9.1.4 Onboarding with HbbTV

With HbbTV 2 there are two scenarios for bootstrapping a multiscreen service. The new companion screen APIs and protocols include discovery of TVs from within the home and the discovery of launcher applications from within an HbbTV application. Such launcher applications must be provided by the manufacturer of the TV or a third party that has access to the manufacturer specific protocol with the launcher.

9.1.4.1 Experience started from TV

A broadcaster can directly enhance his broadcast services with extra information via IP. When the user tunes to a broadcast channel an application information table is evaluated and as a result starts a broadcast related HbbTV application. This is treated as relatively safe and broadcast networks are still trusted. Though there are discussions and lab showcases for so-called man-in-the-middle attacks where the attacker simply over-powers the original broadcast signal with a malicious signal. DVB and HbbTV have reacted on these potential scenarios by introducing a specification that allows to re-introduce trust if such attacks would become a real threat.

If the application wants to spawn a second view on a companion device, it can try to discover launcher applications. These are maintained by the manufacturer of the user's TV and it is his responsibility to keep the communication with the launcher save and e.g. only accessible to broadcast related applications. To start a local communication between TV and the companion side the application must agree on so-call app specific endpoint which is a random string of up to 1000 bytes. The HbbTV application can randomly choose one each time it launches an application and transmit the secret endpoint to the companion side via the launch operation. The launch request can be interrupted by the TV implementation and/or the launcher and be granted only by approval or restricted to trusted applications.

9.1.4.2 Experience started from mobile applications

When the user starts his journey from a mobile application and wants to extend the application to the TV screen, HbbTV 2 allows to discover the TV via the DIAL protocol. DIAL stands for discovery and launch. The second part is used to start an HbbTV application on the TV and this is protected by user approval or whitelisting through the manufacturer of the TV. One of these methods is required by the specification. Approval can be stored and is bound to the URL of the application. The launch data send to the TV again can include some secret that is later used for app to app communication. The communication via DIAL does not use TLS and could be potentially spied out by a third party.

9.1.4.3 Manual joining an experience on TV

A user could watch a broadcast service on TV and is interested to use the broadcaster’s application on his mobile, but he is either not aware of or not willing to install the manufacturer’s mobile launcher app. Still it is possible to connect both sides, the broadcasters HbbTV app with the companion app. To still connect with the companion app, the HbbTV app could open the local end for the app to app communication service and the mobile app then connects to the remote end. The only technical requirement is that both apps use the same app-specific string for the connection (i.e. same sub path of the web socket URL).

The following sketches a solution that shall ensure security and that the user is asked for consent before pairing. The app to app channel is not encrypted and as such treated as an unsecure communication channel that is prone to man-in-the-middle attacks. HbbTV and mobile app need to share the knowledge about a specific app endpoint which they use to discover each other as a first step by opening a web socket to the local and remote endpoints respectively.

Using cryptographic means both the HbbTV and the companion app generate a (symmetric) key to encrypt any subsequent communication. Before starting secure communication both ends present the user a number or string that is based on the common secret (key), and let the user approve that it is the same. Now the secret key can be stored locally in the session or persistent (web) storage. Web storage for HbbTV applications is only visible to web pages of the same domain. From time to time a new key for encryption should be exchanged. If time between usages is too long to ensure safe communication, the initial exchange with user approval needs to be repeated.

9.1.5 Recommendations

- Use NGA (Next Generation Audio) for more flexible audio configuration, e.g. different mixes of commentary and “background noise”. This has been published recently by DVB for broadcast as well as for broadband in DVB DASH. The latter has been adopted by HbbTV 2.0.2
- Multi-video for picture-in-picture is already possible from the specification point of view. However, manufacturers are reluctant to include it if there is no strong business case.
- 360° video on TV has a lot of sceptics, though there are ways to achieve this today which are probably sufficient if there is enough demand to improve performance for it via the specification.

- The observed issues with the app2app channel in the MotoGP trial can be worked around by optimising the protocol and it is not a real limitation.
- Any other modification is not related to a limitation of the HbbTV specification.

It should be noted that TV manufacturers are reluctant to update the HbbTV specification with new features for upcoming revisions. The current focus of the members of HbbTV is to maintain existing features and keep up with developments of browser specifications and media formats as well as to stabilise implementations and services for new features introduced with HbbTV 2.

9.2 Cloud-based Media Synchronisation

The 'cloud-sync' service is an Inter-Destination Media Synchronisation (IDMS) scheme that involves distributing timing and control to a number of devices. It enables a particular device or component to distribute a timing source (e.g. a timeline embedded in the media stream, or a clock) to a number of interested peers (other devices or services). It does so by creating a local estimate of the timing source and keeping that estimate accurately synchronised with the original timing source by periodic updates.

At the source, the progress of the timing source is measured with respect to a global reference clock, the WallClock. The progress of the timing source is then disseminated via Correlation Timestamps. Time synchronisation protocols ensure that each cloud-sync client has an up-to-date copy of the WallClock. This local estimate (called a Timeline Shadow) is updated by periodic Correlation Timestamps in the form of a tuple: *{WallClock time, timeline time, speed}*. The timing source may have an arbitrary frequency and phase compared to the WallClock; these characteristics of the timing source are also manifested in its timeline shadows.

In addition to the **Master-Slave sync method** described above, the cloud-sync service also supports the **Synchronisation Master or Maestro** mode where the synchronisation master (here, the cloud-sync service) collects timing information from all clients, computes a target time position and sends the target timing instruction to all client peers for them to adhere to. It uses the concept of a *Synchronisation Timeline* to specify what the target timing should be.

9.2.1 Factors Affecting Synchronisation Accuracy

The following factors will affect synchronisation accuracy:

9.2.1.1 Accuracy of WallClock estimation

The design of the synchronisation solution in cloud-sync is grounded in the use of a time-synchronised WallClock; this is used as a reference time base in the first component of Correlation Timestamps i.e. the *WallClock time*.

A round-trip time-synchronisation method is used for synchronising the WallClock. The protocol involves taking regular measurements by timestamping requests and their corresponding responses and use the results to update an estimate of the WallClock.

The sequence diagram in Figure 20 illustrates how these times relate to the message exchange.

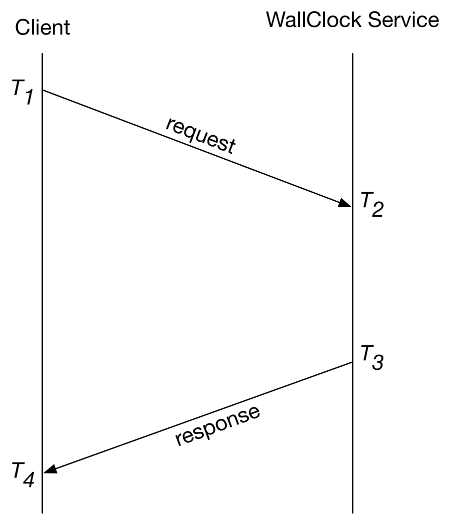


Figure 20: Round-trip time synchronisation

Based on the measurements, the round-trip delay and the offset between the WallClock Service's clock and the client device's clock can be calculated as follows:

$$round\ trip\ delay, \delta = (T_4 - T_1) - (T_3 - T_2)$$

$$phase\ offset, \theta = \frac{(T_3 + T_2) - (T_4 + T_1)}{2}$$

The method assumes that network latencies are symmetric. This is not necessarily true in real scenarios, but it is done because there is no easy way to detect asymmetry. The consequence of this assumption is that the measurement has error bounds.

The error bound due to network latency can therefore be quantified as:

$$potential\ error\ due\ to\ network\ latency = \pm\frac{rtt}{2} = \pm\frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

There are two other factors that affect the estimation and make it imperfect:

1. The *precision* with which the cloud-sync service and the client devices measure their clocks. For example: a clock that increments every nanosecond will only have 1 nanosecond precision. A clock that is in units of nanoseconds but is only updated every 1 millisecond will have 1 millisecond precision.
2. Clocks will have some *frequency error* (running slightly fast or slow) because they are not perfectly precise oscillators. This adds a small amount of inaccuracy to the calculation of the offset or correlation. But also, as time passes, two clocks that were previously synchronised will drift further apart so this increases the error over time. This growth over time is illustrated in Figure 21.

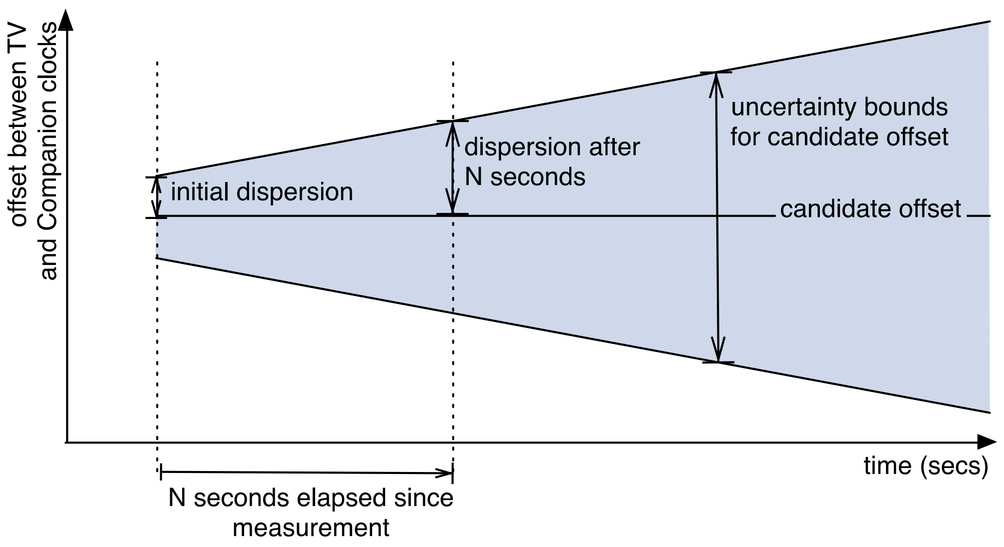

offset between TV and Companion clocks
initial dispersion
dispersion after N seconds
uncertainty bounds for candidate offset
candidate offset
time (secs)
N seconds elapsed since measurement

Figure 21: How dispersion (and therefore uncertainty) grows as time elapses since a measurement

All of these factors can be combined into an overall error-bound ε called the *dispersion*:

$$\mathcal{E}(\tau) = \pm(\mathcal{E}_0 + \tau\, \mathcal{E})$$

Where:

- $\mathcal{E}_0$ is the initial dispersion at the moment the measurement has been taken
- $\mathcal{E}$ is the amount by which dispersion increases per unit of time
- τ is the amount of time that has passed since this measurement was taken

A large portion of the WallClock estimation error (i.e. the dispersion) is due to network delay (RTT/2). Network characteristics such as throughput, delay, etc. during wall clock synchronisation therefore contribute to the accuracy of the client estimate of the global WallClock. This, in turn, determines the degree of accuracy for media synchronisation that can be achieved and sustained throughout the experience.

9.2.1.2 Presentation Timing

Once an estimate of a WallClock is obtained, a correlation relationship (e.g. Correlation Timestamp) between the WallClock and a video timeline is sufficient to build an estimate of that timeline at different client. This timeline estimate (i.e. the Timeline Shadow) can then be used to drive the playback of the same video on the device.

For instance, a local controller on the device can compare the actual time reported by the video player with the current time on the video timeline shadow and determine the asynchrony. Based on the magnitude of this asynchrony, it can select a strategy for playback adaptation that involves Adaptive Media Playback (rate adjustment) or Media-Frame Skipping (seeking).

There are two possible sources of error when reading the current time of the video object:

Presentation delay – this is the decoding latency of the media processing pipeline. i.e. this is the time between a media player reporting current playback of a media sample and the time this sample is actually output (pixel is lit on the screen or audio sample comes out of the speakers). This delay is fixed and device-platform specific.

Media Player Precision – this is the precision at which the video player reports changes in playback progress. Milli-second precision is desirable for playback adjustment. Even then, continuous playback adjustment may exacerbate the jitter value of the play head position (as illustrated by Figure 22).

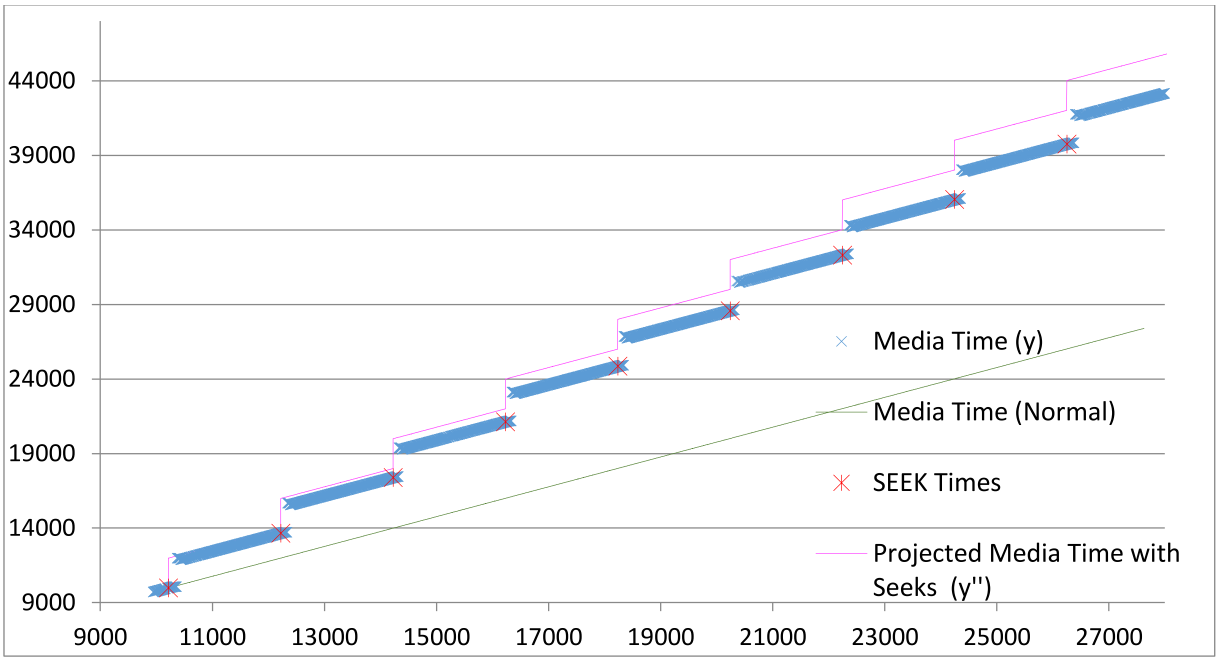


Figure 22: Jitter in Current Time value reported by AVPlayer on iOS (Seek +2s at 2s interval)

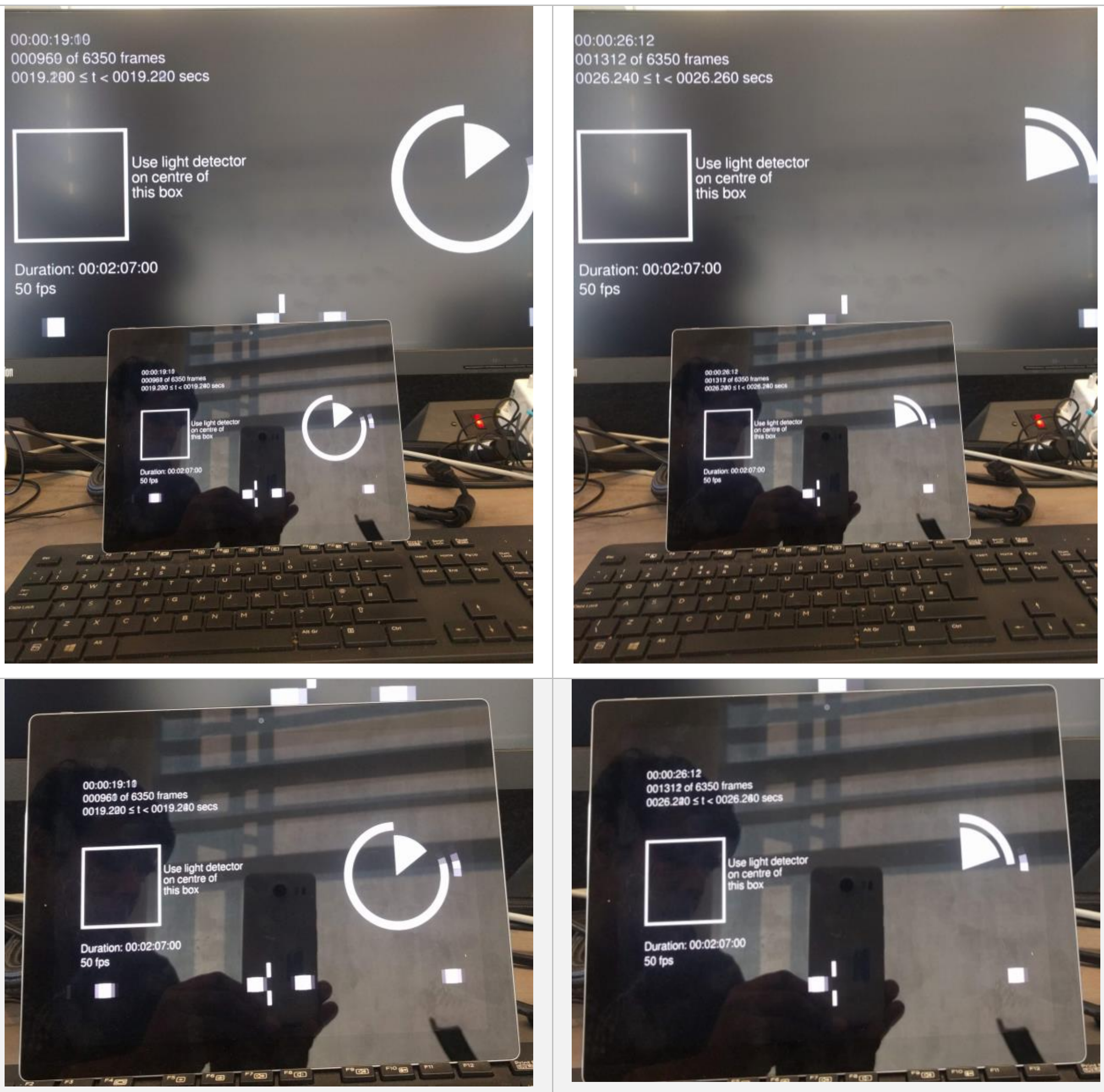
9.2.2 Coarse Sync Accuracy Measurement

We conducted an experiment to estimate coarsely the end-to-end accuracy of inter-client synchronisation using the cloud-sync service. A 50 fps non-interlaced video showing frame numbers and current time since the first frame was played on multiple clients synchronised to the same cloud-sync server. One of the clients was the master i.e. it provided its own video timeline to be used as a synchronisation timeline by other clients via the cloud-sync service.

The synchronisation error is quantified by taking a photo with a low-latency camera of the video playing on two distinct devices and then comparing the frame numbers and current time visualisation. We performed this experiment several times with the cloud-sync server hosted on the 2Immerse AWS platform and with two screens at BBC R&D in Manchester, UK. A laptop and an Android tablet (Google Pixel) were connected to the internet via WiFi. Each time, the WallClock dispersion value was also recorded to determine the error in our time estimates.

The frame number on the video changed at 50Hz i.e. every 20ms. With this method, therefore, we are able to only measure synchronisation error no smaller than 20ms. A synchronisation error of 20ms implies an average difference of 1 frame between the synchronised videos. The experiment was repeated 20 times. The two laptops differed by 2 frame for 5 runs, 1 frame for 10 runs and by 0 frame in the 5 remaining runs. Figure 23 and Figure 24 show the results from two sample runs in the experiment. The recorded dispersion value was 14 ms on average.

These results indicate that the average end-to-end inter-client synchronisation error in this instance is about 1 frame. This is close to lip-sync accuracy.


00:00:19:10
000960 of 6350 frames
0019.200 ≤ t < 0019.220 secs
Use light detector on centre of this box
Duration: 00:02:07:00
50 fps
00:00:26:12
001312 of 6350 frames
0026.240 ≤ t < 0026.260 secs

Run 1 of 20 | **Run 2 of 20**

Figure 23: Sync accuracy measurement (2 runs shown), video at 50 fps

Run 1			Run 2		
	Frame Number	**Current time**		**Frame Number**	**Current time**
Main screen	≈ 960	19.200 s ≤ t < 19.220 s	**Main screen**	≈ 1312	26.240 s ≤ t < 26.260s
Tablet	≈ 960	19.200 s ≤ t < 19.220 s	**Tablet**	≈ 1312	26.240 s ≤ t < 26.260 s

Figure 24: Sync accuracy measurement - reference and observed frame numbers, current time

9.2.3 Synchronisation Skew Measurement

To obtain more accurate measurements of the synchronisation error, we designed another experiment to attempt measure asynchrony at a finer-grain. To achieve this, we used a microcontroller (with an ADC) to sample the audio coming from two devices at high speed. An Arduino Due with 2 audio line-inputs was selected as the micro-controller platform. The hardware was reused from previous sync timing evaluation experiments; the design specifications of the sync-timing measurement kit can be found on GitHub (12).

We chose a sample rate of about 300,000 samples per second. Using fast DMA-assisted data transfer and a baud-rate of 90kHz, we were able to achieve a sampling rate of nearly 1 million samples per second and transfer the sampled values quickly via USB to a laptop. However, this sampling rate was an order of magnitude higher than sample rate of the audio (44.1 KHz) played back by the two devices. For this reason, the sampling rate of the audio signals were lowered to 300 KHz.

In the experiment, the two devices (a MacBook Pro laptop and an Android Tablet) were first made to play the same video and keep the two video instances synchronised using the cloud-sync service. The video was generated in such a way as to include flashes and beeps at non-repeating random intervals. The sampling of the two audio signals by the hardware kit produced two streams of values indicating the voltage sensed by the Arduino ADC on the particular signal channel. The streams of sample values were submitted to a signal analyser (implemented in Python) to determine the phase offset between the two signals. This phase offset provides an initial estimate the synchronisation error between the two devices. A more comprehensive calculation of the phase offset between the two streams of values involves using a sliding-window approach to repeatedly calculate correlation coefficients between two sets of sampled values from the streams. The window with the highest correlation coefficient then provides us with an accurate estimate of the phase offset between the two audio signals.

At the time of writing, we were only able to measure phase offset using the first approach (i.e. measuring phase offset using the signal-analyser). We repeated measurements of the phase offset using

1) 2 devices with different hardware (MacBook Pro laptop and Google Pixel Android tablet), and

2) 2 devices with exactly the same hardware and software configuration (2 Macbook Pro laptops).

The purpose of this was to determine whether the asynchrony noticed between dissimilar devices were due to different presentation delays. The results are summarised in Figure 25 and Figure 26.

These initial results confirm the observations from the first experiment (Coarse Sync Accuracy Measurement, Section 9.2.2); a synchronisation error of about 20ms is observed between the Macbook laptop and the Android tablet. When the experiment is repeated with two MacBook Pro laptops playing the synchronised video (Figure 26), the synchronisation error is reduced to around 5ms.

The results show that the synchronisation error can be mostly attributed to the difference in presentation delays in both devices.

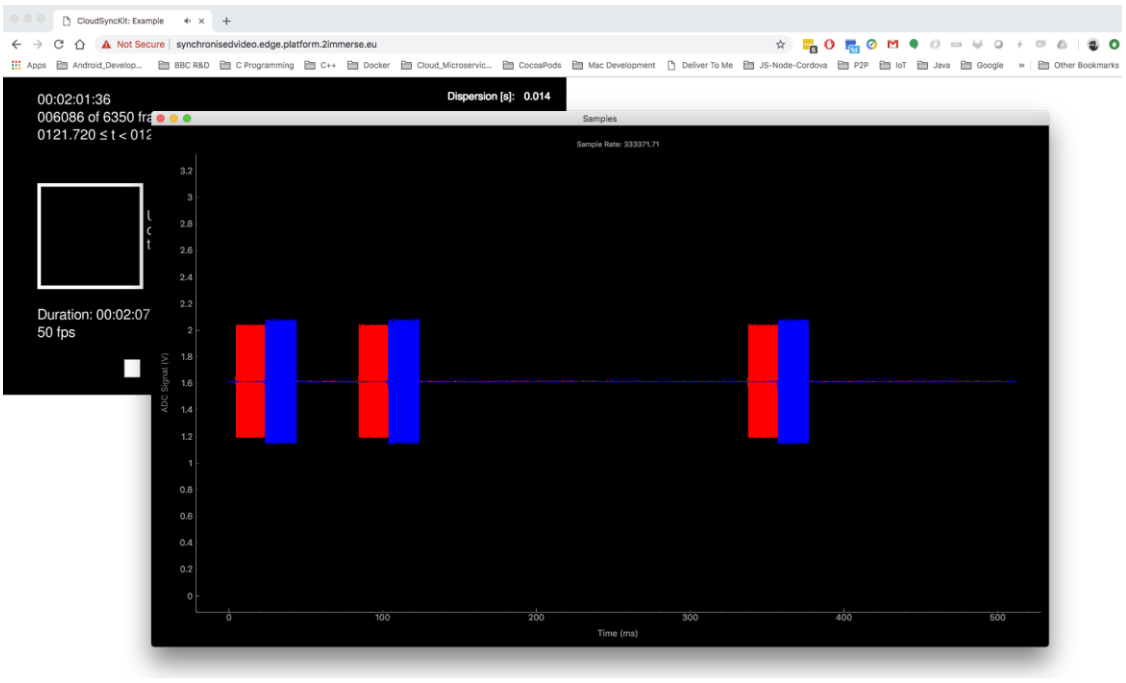


Figure 25: Asynchrony measurement between laptop and Android Tablet, dispersion = 15ms

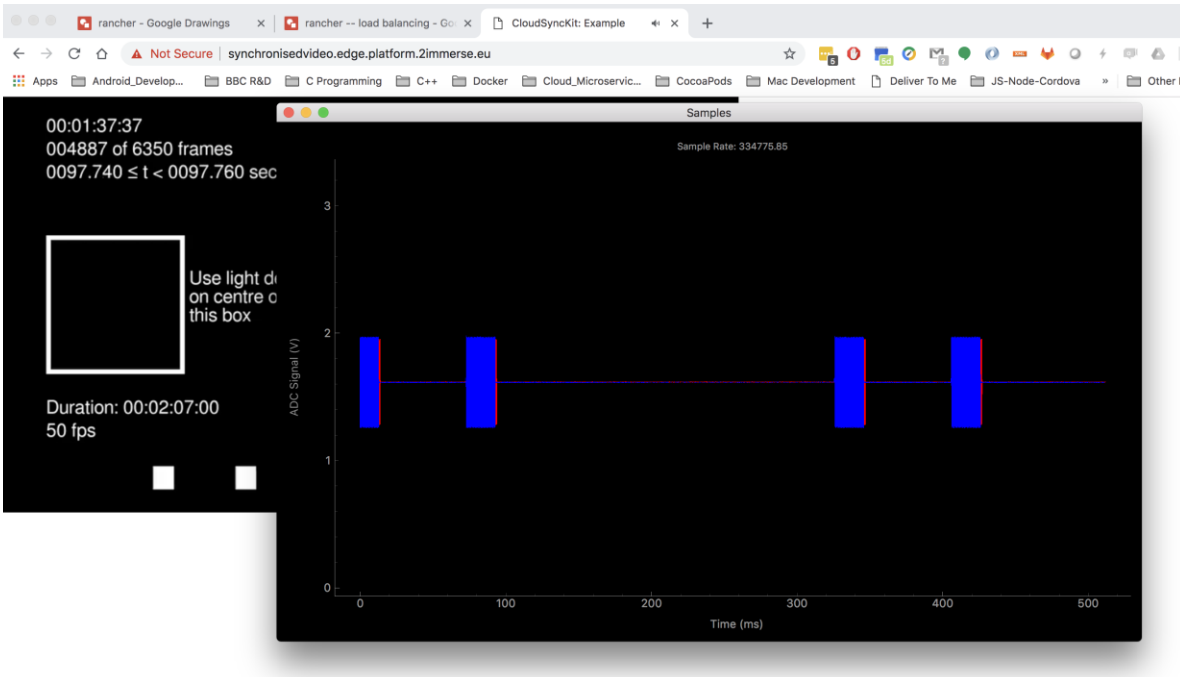


Figure 26: Asynchrony measurement between two Mac Book Pro 2017 laptops, dispersion = 17ms

9.2.4 Synchronised Timing Measurement

In a bid to determine the presentation delays on different devices, we devised another experiment to record the actual and expected times of flashes in a video segment played by each device and then use this delay offset to calibrate the video playback on the respective device.

As illustrated in Figure 27, the measurement system consists of:

1) Python code running on a PC or laptop that emulates the role of a Synchronisation master (a cloud-sync service client).
2) A microcontroller for measuring the timing of light and sound output from the device being measured.
3) A test video sequence that another device (the sync slave – another cloud-sync service client) plays.

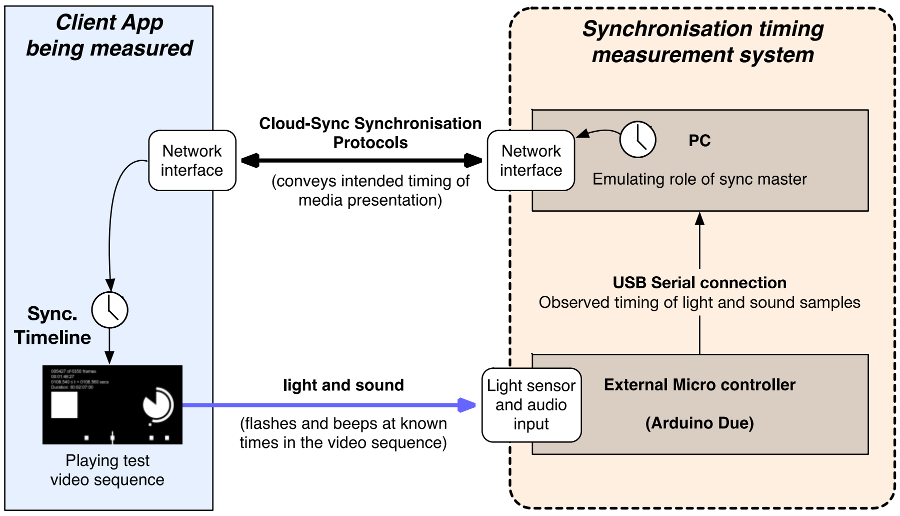

Client App being measured
Network interface
Sync. Timeline
Playing test video sequence
Cloud-Sync Synchronisation Protocols
(conveys intended timing of media presentation)
light and sound
(flashes and beeps at known times in the video sequence)
Synchronisation timing measurement system
Network interface
PC
Emulating role of sync master
USB Serial connection
Observed timing of light and sound samples
Light sensor and audio input
External Micro controller
(Arduino Due)

Figure 27: Cloud-sync timing measurement system

The PC code provides a timing source by starting a software clock and it also maintains a local synchronised WallClock. It then advertises this timeline to the client App (running on another device) for synchronisation. This synchronisation timeline is used to drive playback of a video sequence on the device being measured. An external microcontroller (an Arduino Due) samples the light and sound from the device being tested and notes the precise timing of those samples.

The Client Application must present a specific video test sequence containing flashes and beeps at defined times. The flashes are detected by a light sensor affixed to the display over the area of the image that flashes. The audio output (or a microphone) feeds a line-level audio input.

The microcontroller is told which light sensor and/or audio inputs should be read during data capture. Once the microcontroller has collected samples over a period of time, these are sent to the PC via a USB connection. Code on the PC detects the flashes and beeps from the sample data and translates the timings of the samples to that of the timeline advertised by the 'cloud-sync' protocols.

The PC can therefore match up the observed timings of flashes and beeps to those that it expected for the video clip and determine how much they differed from what was conveyed via cloud-sync.

In the experiment, for each device being tested, 20 runs of the measurement process were made. In each run, 5 time measurements were taken, each over a 15 second period. To reduce synchronisation error due to WallClock dispersion, the sync service and measurement tool were running on the same device.

The results for a Lenovo laptop display system are shown in Figure 28.

A mean error of 60.69s was observed over all the measurements with a standard deviation of 13.13ms. In the individual measurement sequences (e.g. -69.35ms, -61.53ms, -46.76ms, -53.44ms, -72.38ms), the errors differ by a mean value of +/- 15ms

From these observations, a calibration offset of 60ms for the device can be estimated i.e. the device's presentation delay is about 60ms. If playback of the device were to be adjusted by 60ms, an end-to-end error (client to client) of +/-15ms would be achieved.

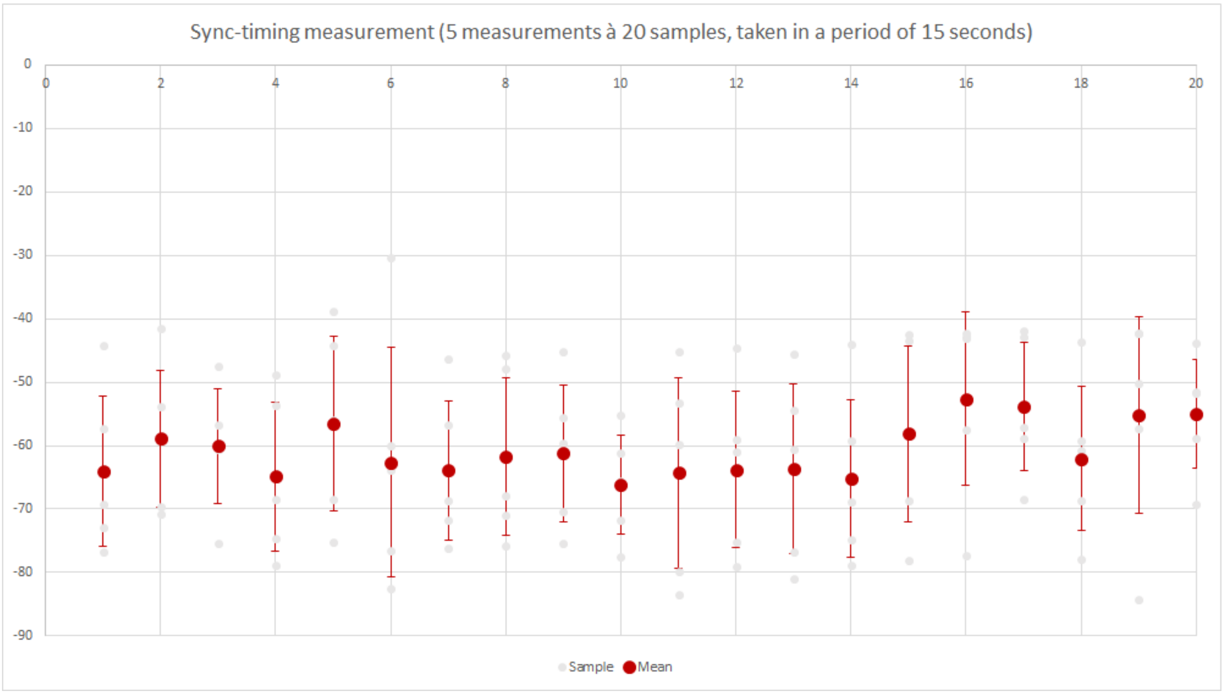


Figure 28: Difference between actual and expected timings on a Lenovo laptop

9.3 Bandwidth Orchestration

Deliverable D2.4 (3) described the design of the Bandwidth Orchestration Service, whose purpose is to provide architectural support for component bandwidth management within the 2-IMMERSE platform. It was unfortunately not possible to carry out meaningful evaluation of bandwidth orchestration performance during the trials and technical tests which took place during the final year of the project. The Bandwidth Orchestration Service was therefore evaluated using various synthetic tests that simulate several clients.

9.3.1 Bandwidth management algorithm

Here is a synthetic example of how its bandwidth management algorithm works. We have seven clients streaming DASH video with the following available bitrates:

819200, 2048000, 4096000, 8388608, 41943040

The network has an available bandwidth of 20000000 (we take off five percent for safety) and we choose to use three priority levels to group the components within the algorithm where the lower number the higher the priority. Table 7 (sorted by component priority) shows the state before and after a pass of the algorithm.

Component	Priority	Priority Group	Previous Bitrate	Action	New Bitrate
6	100	1	4096000	Upgrade	8388608
7	70	2	41943040	Downgrade	4096000
4	60	2	4096000	Downgrade	2048000
2	40	3	8388608	Downgrade	819200
5	20	3	2048000	Downgrade	819200
1	10	3	4096000	Downgrade	819200
3	10	3	4096000	Downgrade	819200

Table 7: Synthetic example of Bandwidth Orchestration Service in operation

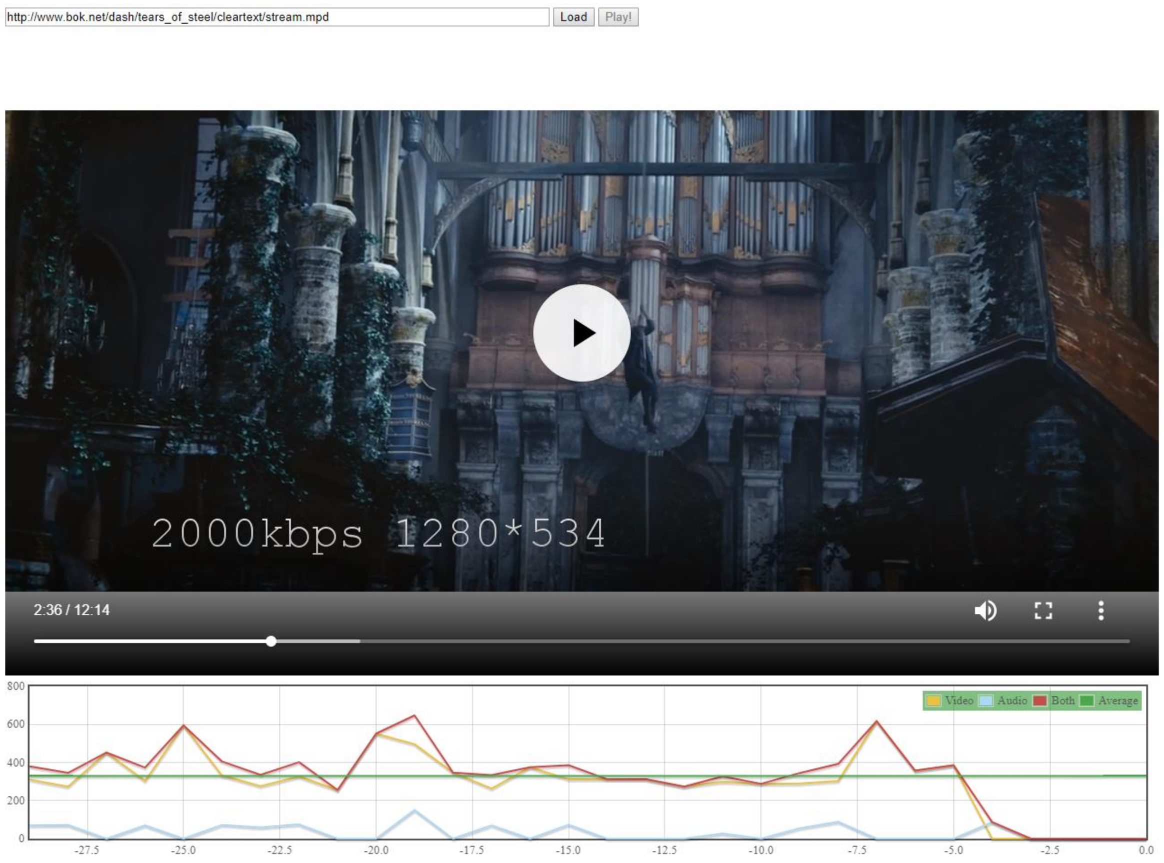
Total original bandwidth: 86573056 > 20000000

Total new bandwidth: 17809408 < 20000000

We can see that the algorithm managed to fit all the components into the available bandwidth while using the defined priorities. As this is an approximation algorithm, the result is not “optimal” but good enough for real-world uses and actually works in real-time (for the above 7 clients the entire computation take between 1 and 2 milliseconds) unlike the known approximation algorithms that could takes many minutes to hours to get a good approximation. We take into consideration that a single Bandwidth Orchestration server can manage many simultaneous 2-IMMERSE experiences and so should use minimal resources per experience.

9.3.2 Bandwidth estimation tests

While doing a small real-world test with two clients, we realized that the browser does not allow JavaScript code to get actual information about bandwidth usage and so we used some estimation mechanisms with limited accuracy (which is the best we can do with the current state of browsers). The results were reasonable but the noise level is noticeable.



http://www.bok.net/dash/tears_of_steel/cleartext/stream.mpd

Figure 29: Successful bandwidth estimation

In Figure 29 above we see our test client playing “Tears of Steal” as a DASH stream with several available bitrates. In the graph we see our attempt at estimating the bandwidth usage of the dash.js client. This is done by tracking the start/end times of segment downloads and their sizes and then trying to sum up the relative parts within predefined time periods. The X axis shows the measurement history where the right edge is “now” (we keep 30 measurements for averaging by default), while the Y axis shows the bandwidth usage in KB/s. For the most part this seems to be a relatively good estimation and our algorithm is able to rely on them and direct the clients to have a nice and smooth playback according to their set QoS.

However, when running multiple clients we sometimes observe the output shown in Figure 30.

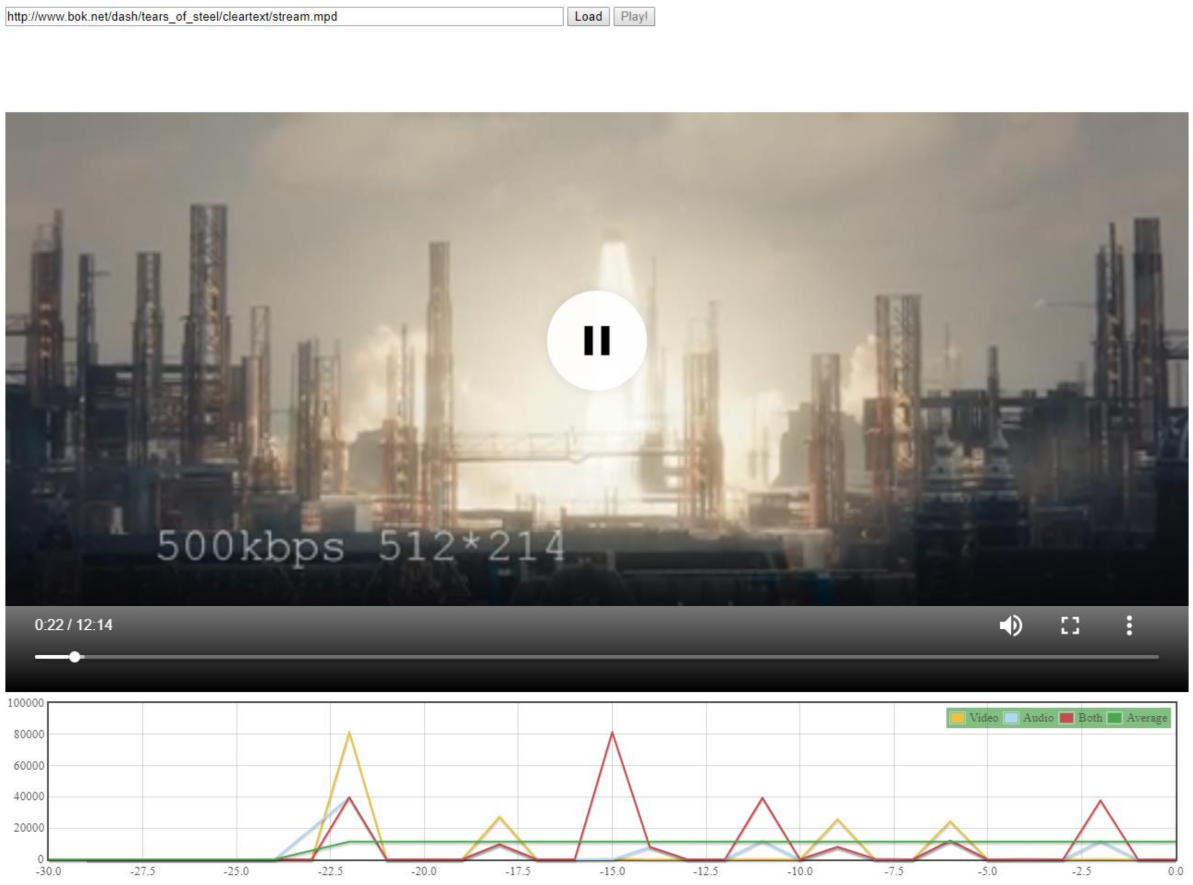


Figure 30: Problematic bandwidth estimation

Here we can see the client measurement is totally unrealistic because the browser starts preloading and caching segments but never reports that to the JavaScript components and thus it seems we have spikes of anywhere between 20MB/s and 80MB/s every few seconds instead of a smooth download curve like we saw in the previous example. This prevents us from doing proper bandwidth management for such clients and we simply ignore them.

There are some proposals to add APIs to get the required information from the browser so in the future it might be possible to use these to achieve better results.

10 Conclusion

This document has described the final release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's four service prototypes. At the end of the project, this release represents the most mature and robust implementation of the platform, components and tools with the functionality required to deliver all of the service prototypes, and in particular the three which have been completed during the final year of the project: Football Fanzone, Football at Home and Theatre in Schools.

The key features and highlights of this release have included:

- Development of the Football and Theatre in Schools DMApp implementations, with their corresponding requirements for new DMApp Components and updates to timeline, layout and Client API features.
- The Unified Launcher, a single all-purpose onboarding implementation that supports the diverse requirements of all the 2-IMMERSE service prototypes, including managing multiple device roles and the ability to launch an experience on multiple communal devices.
- A complete inventory of the 37 DMApp Components which have been developed during the course of the project, some providing core functionality which has been re-used across multiple DMApps, others providing specific functionality which meet the more precise requirements of a genre, but which could still be re-used within that genre.
- A standalone implementation of the 2-IMMERSE platform, which enables the 2-IMMERSE service prototypes to be demonstrated without requiring a connection to the cloud-hosted Rancher-managed instances of the platform.
- The evolution of the production tools for real-time triggering to provide new functionality required for testing in a live production environment, and the development of a new pre-production tool which is intended to help create the initial storyline of an experience.
- A description of work being carried out on platform sustainability to enable the 2-IMMERSE platform and the 2-IMMERSE open-source software release to be more easily set up and used by project partners and third parties after the end of the project.
- A description of three targeted platform evaluations carried out to measure the effectiveness of core functionality of the platform which it was not possible to test within the scope of the service prototypes.

At the end of the 2-IMMERSE project, the Final Release demonstrates that the project has delivered a fully-featured, extensible platform to support the end-to-end production, delivery and consumption of multi-screen experiences. The 2-IMMERSE Reference Architecture and open-source software release (described in Deliverable D2.6 (13)) has been derived from it in order to foster a professional community to support the successful adoption of object-based broadcasting among European TV production professionals.

11 References

1. **2-IMMERSE.** *D2.2 Platform-Component Interface Specifications.* 2016.

2. —. *D2.1 System Architecture.* 2016.

3. —. *D2.4 Distributed Media Application Platform and Multi-screen Experience Components: Description of Second Release.* 2018.

4. —. *D4.6 Football trial evaluation results.* 2018.

5. —. *D4.4 Prototype Service Descriptions – Second Update.* 2018.

6. —. *D3.5 User experience results: Interactions for Theatre in Schools.* 2018.

7. **Python Software Foundation.** Python 2.7 Release Schedule. [Online] [Cited: 18 12 2018.] https://legacy.python.org/dev/peps/pep-0373/.

8. **2-IMMERSE.** *D3.4 User Interaction Design: the development of generic components & features.* 2018.

9. **Netflix.** DIAL Protocol Specification. [Online] [Cited: 18 12 2018.] http://www.dial-multiscreen.org/dial-protocol-specification.

10. **2-IMMERSE.** *D2.3 Distributed Media Application Platform and Multi-Screen Experience Components: Description of First Release.* 2016.

11. —. 2-IMMERSE Blog - MotoGP roars out on a HbbTV 2 television. [Online] [Cited: 18 12 2018.] https://2immerse.eu/motogp-roars-out-on-a-hbbtv-2-television/.

12. **BBC.** Github - DVB companion synchronisation timing accuracy measurement. [Online] [Cited: 18 12 2018.] https://github.com/bbc/dvbcss-synctiming.

13. **2-IMMERSE.** *D2.6 Distributed Media Application Platform: Public software release.* 2018.