

Directorate General for Communications Networks, Content and Technology
Innovation Action

ICT-687655



2IMMERSE

D2.4 - Distributed Media Application Platform and Multi-Screen Experience Components: Description of Second Release

Due date of deliverable: 30 November 2017

Start date of project: 1 December 2015

Duration: 36 months

Lead contractor for this deliverable: **Cisco**

Version: 11 January 2018

Confidentiality status: **Public**

Abstract

This document describes the second release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's second service prototype, "Watching MotoGP at Home". It provides an illustrated tour of the project's technical achievements to date, along with details of the current status of the platform and components and key features developed beyond those described in deliverables D2.1 (1), D2.2 (2) and D2.3 (3).

The description of the second release was originally defined as two separate written reports: D2.4 (Distributed Media Application Platform: Description of Second Release) and D5.2 (Multi-Screen Experience Components: Description of Second Release). To make the content easier to read and navigate, the complete description is now provided in this document, D2.4. Deliverable D5.2 now contains a series of videos which show the 2-IMMERSE Platform, Components and Production Tools in action.

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.

Disclaimer

This document contains material, which is the copyright of certain 2-IMMERSE consortium parties, and may not be reproduced or copied without permission. All 2-IMMERSE consortium parties have agreed to full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the 2-IMMERSE consortium as a whole, nor a certain party of the 2-IMMERSE consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

Impressum

Full project title: 2-IMMERSE

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.4 Distributed Media Application Platform and Multi-Screen Experience Components: Description of Second Release

Editors: James Walker (Cisco) and Pablo Cesar (CWI)

Workpackage Leader: James Walker, Cisco

Technical Project Leader: Mark Lomas, BBC

Project Co-ordinator: Helene Waters, BBC

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

Executive Summary

The second release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components is based on 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 (2).

The second trial to be undertaken by the 2-IMMERSE project is based on the “Watching MotoGP at Home” service prototype. Technical development has therefore been focused on extending the first release platform to address the prioritised requirements of this prototype, as expressed in deliverable D4.4 (4) (5). As with the first release, it is important to note that the majority of the platform elements, multi-screen experience components and production tools created for this second release will be enhanced and used again for subsequent trials.

The following summarises the key technical achievements of the second release:

- Extension of first release platform, to include new services and extend existing services to deliver functionality required by the MotoGP service prototype. New services include:
 - Auth and Auth-Admin Services – supporting user identity management and authentication.
 - Data Playback Service – supporting generic methods for the capture, transformation and distribution of production-related non-audio/video data streams.
 - Bandwidth Orchestration Service – supporting monitoring and management of bandwidth consumed by streaming media components, in accordance with MPEG’s Server And Network Assisted DASH (SAND).
 - Editor Service – supports editing operations to timeline documents via authoring front-end applications, and interaction with the rest of the 2-IMMERSE platform.
- Migration of the service platform from a private cloud environment to Amazon Web Services (AWS) and then subsequently migration from the Mantl container platform to Rancher.
- Development of Linux-based HbbTV2.0 Emulator firmware to run on Intel NUC devices to support service prototypes. Key features include:
 - Onboarding (supporting user network configuration, sign-in, device pairing, experience discovery and experience launch)
 - Integrated Wi-Fi router/access point
 - HbbTV2.0 services (App2App server, DVB-CSS server and DIAL server)
 - Web Kiosk
- Client API developments to support a DMAP launch configuration document, improve robustness, integration with production tools, bandwidth orchestration, Google analytics, and, improvements to the DMAP component interface.
- Authoring and development of the MotoGP service prototype DMAP and its constituent elements: timeline, layout, HTML and CSS documents, several DMAP components with a focus on data-driven animated graphics, and media asset preparation.
- Production tool development with a focus on the real-time triggering required for the MotoGP scenario, which now is a working prototype integrated with the 2-IMMERSE platform.

As the second instance of a working platform for the delivery of an interactive, object-based multi-screen experience, this Second Release forms the foundation for the remaining prototypes which will be developed and taken to trial in the final year of the project.

Note: The description of the second release was originally defined as two separate written reports: D2.4 (Distributed Media Application Platform: Description of Second Release), and D5.2 (Multi-Screen



Experience Components: Description of Second Release). To make the content easier to read and navigate, the complete description is now provided in this document, D2.4. Deliverable D5.2 now contains a series of videos which show the 2-IMMERSE Platform, Components and Production Tools in action.

List of Authors

Mark Lomas – BBC

Rajiv Ramdhany - BBC

Ian Kegel – BT

Jonathan Rennison - BT

James Walker - Cisco (co-editor)

Tal Maoz - Cisco

Pablo Cesar – IRT (co-editor)

Jack Jansen – CWI

Thomas Röggl - CWI

Fons Kuijk – CWI

Michael Probst – IRT

Reviewers

Ian Kegel – BT

Table of contents

Executive Summary	3
List of Authors	5
Reviewers	5
Table of contents	6
Glossary of terms	9
1 Introduction	10
2 Requirements	11
3 Snapshot of the platform and components	19
3.1 2-IMMERSE Service Platform	19
3.2 2-IMMERSE Code Repository (GitLab).....	20
3.3 Container Platform Services	21
3.4 Backend Services	23
3.5 HbbTV2.0 Emulator	23
3.6 TV and Companion Client applications for the MotoGP at Home trial	24
4 Platform Infrastructure	29
4.1 Mantl Platform Developments	29
4.2 Rancher Platform.....	30
4.3 Origin Server	33
4.4 CI/CD and Docker Registry	33
5 Platform Services	34
5.1 Timeline	34
5.2 Layout	35
5.3 Websocket	36
5.4 Shared State	36
5.5 Logging and Monitoring	36
5.6 WallClock Service.....	40
5.7 Synchronisation Service (Inter-Home Sync)	40
5.8 Authentication.....	40
5.9 API gateway	41
5.10 Data Playback	41
5.11 Bandwidth Orchestration Service.....	44
5.12 HLS Proxy	48
5.13 Docker-Hive	48

5.14	Server-Based Composition	49
6	Client Application Stack.....	50
6.1	Overview.....	50
6.2	MotoGP DMAP Implementation	51
6.3	Client API	54
6.4	HbbTV2.0 Emulator	56
6.5	Companion Devices	57
6.6	HbbTV showcases, tools and software libraries.....	58
7	Multi-Screen Experience (DMApp) Components.....	59
7.1	DMApp Components available in the Second Release	59
8	Production Tools.....	65
8.1	Architecture and Workflow	66
8.2	Document Format	67
8.3	Frontend.....	67
8.4	Backend.....	70
9	Conclusion	72
10	References	73
Annex A	Synchronisation Service (Inter-Home Sync).....	76
A.1	Limitations of current synchronisation approaches in 2-IMMERSE.....	76
A.2	Proposed Synchronisation Model	76
A.3	Architectural Overview.....	78
A.4	Implementation.....	80
Annex B	HbbTV2.0 Emulator Components.....	81
B.1	Operating System	81
B.2	HbbTV2.0 Emulator Services	81
B.3	On-boarding System.....	81
B.4	Admin Portal.....	82
B.5	On-boarding Steps.....	82
B.6	Network connectivity management layer.....	88
B.7	Integrated Wi-Fi router/gateway and access point	89
B.8	Captive Portal	90
B.9	Web Kiosk Service	92
B.10	Creating an SSH tunnel for remotely debugging Chromium.....	94
B.11	Web Server	94
B.12	4K/UHD TV Support.....	94

Annex C	HbbTV showcases, tools and software libraries.....	96
C.1	Showcase applications	96
C.2	HbbTV client libraries	96
C.3	Companion screen libraries for Android	97
C.4	Companion screen libraries for WinJS UWP.....	97
C.5	MPEG TEMI Timeline Inserter	97
C.6	Material resolution service.....	99

Glossary of terms

<i>Term/acronym</i>	<i>Definition/explanation</i>
<i>Experience</i>	2-IMMERSE is developing 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 content distribution.
<i>Distributed Media Application (DMApp)</i>	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.
<i>Distributed Media Application (DMApp) Component</i>	In 2-IMMERSE, re-usable components are assembled within a Distributed Media Application (DMApp) to create coherent multi-screen experiences.
<i>CI/CD</i>	Continuous Integration and Continuous Delivery
<i>Context</i>	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), but a device can only be a member of one context at a time. Devices belonging to the same context must be able to discover each other using the DIAL protocol. Devices can join or leave a context at any time.
<i>IPTV - Internet Protocol television</i>	Internet Protocol television (IPTV) is the delivery of television content using signals based on the logical Internet protocol (IP), rather than through traditional terrestrial, satellite signal, and cable television formats. IPTV is important to this project as it is IPTV delivery that enables the OBB (object based broadcasting) approach to content delivery.

1 Introduction

This document describes the second 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. The platform, components and tools will be continually developed but at this stage they have been built to be sufficient for the first and second service prototypes, "Watching Theatre at Home" (henceforth referred to as *Theatre at Home*), and "Watching MotoGP at Home" (henceforth referred to as *MotoGP*).

The description of the second release was originally defined as two separate written reports: D2.4 (Distributed Media Application Platform: Description of Second Release), and D5.2 (Multi-Screen Experience Components: Description of Second Release). To make the content easier to read and navigate, the complete description is now provided in this document, D2.4. Deliverable D5.2 now contains a series of videos which show the 2-IMMERSE Platform, Components and Production Tools in action.

This second release is a practical implementation of the system architecture defined in project deliverable D2.1, and the platform component interfaces defined in project deliverable D2.2 (2). With the development focus on extending the first release of the platform (which supported the Theatre at Home service prototype), to cover the requirements from the MotoGP service prototype, platform development has been focused on the infrastructure, services and client application to support this second service prototype. Similarly, the development of Production Tools and Multi-Screen Experience Components has been prioritised according to MotoGP requirements. The deliverable is structured as follows:

- Introduction - introduces the second release of the Distributed Media Application Platform, and explains how the rest of the deliverable is structured.
- Requirements – A summary of the high level technical requirements for the platform, DMAP and components.
- Snapshot of the platform and components – provides a brief visual overview of the second release and the platform architecture implemented to date.
- Platform Infrastructure – describes the infrastructure deployed to support the 2-IMMERSE service platform.
- Platform Services – describes the core platform services developed and deployed to date and their current status.
- Client Application – describes the client application stack developed to date and its current status.
- Multi-Screen Experience Components – describes the Multi-Screen Experience Components that have been developed for the MotoGP service prototype.
- Production Tools – describes the Production Tools developed to support the authoring of the MotoGP service prototype.

2 Requirements

As the user experience for the MotoGP service prototype was defined, a set of high-level technical requirements for the platform and client application were identified. In some cases, these were clearly new requirements for existing services in the platform architecture or for the re-use of existing DMAP Components at the client, and in other cases they suggested the definition of a service or set of services, new functionality in the Client API or new DMAP Components.

Table 1 below lists the set of high-level technical requirements for the platform and DMAP as user stories.

Scope	Reference	User Story
Layout	MGP001	As a producer of the experience, I want to be able to precisely define the size and position of graphical components presented overlaid over the main video component.
	MPG002	As a producer of the experience, I want to be able to control placement of a component within a device or device region.
	MPG003	As a producer of the experience, I want to be able to author the component layout to adapt to the physical dimensions of the display device, with either components growing to take advantage of extra space, or allowing the introduction of additional components in the extra space on a larger device (i.e. to be able to capitalise on available space using a form of responsive presentation)
	MPG004	As a consumer of the experience, I want to be able to migrate a component from a personal device (companion) to a communal device (TV)
Timeline	MGP005	As a consumer of the experience, I want to be able to scrub back through the experience timeline, and all of the presentation will be synchronised to the timeline (available components, layout, component content etc.). This scrubbing would likely be to bookmarks (producer or user created – see MGP006)
	MGP006	As a producer of the experience, I want to be able to author ‘bookmark events’ that are available to all consumers of the experience with minimal delay.
Authentication/user management	MGP007	As a consumer of the experience, I would like to be able to persist my experience preferences (e.g. experience level).
	MGP008	As a trial manager, I would like to be able to access and analyse consumer preferences persisted in the course of a trial.
	MGP009	As a trial manager, I want to be able to create a user profile and credentials for each participating triallist.
Data Playback	MGP010	As a consumer of the experience, all live data feeds that are part of the experience (for example driving data-driven components, or being used to drive presentation decisions) should be captured and made available on a time-shifted basis

Scope	Reference	User Story
		to make an 'as-live' experience as close as possible to the live experience.
Participation app backend	MPG011	As a producer of the experience, I would like to be able to use a back-end service to manage collation and aggregation of viewer rating and voting interaction during the experience.
DMap	MPG012	As a producer of the experience, I want to be able to author the experience to support multiple experience levels, e.g. expert / fan / novice
	MPG013	As a producer of the experience, I want the TV component experience level (Xp) to be presented according to the following rules: Single user – TV Xp set same as CS Multiple users same Xp - TV Xp set same as CS Multiple users different Xp - TV set to 'standard'
DMap Components	MPG014	As a producer of the experience, I want to be able to author a series of DMap components, each containing one or more graphics objects that support animation triggers (e.g. Race Info / Leader Board / Timing / Info Panel)
	MPG026	As a producer of the experience I want to use be able to export animation assets from Adobe Animate into animation components.
	MPG017	As a producer of the experience I would like to make 360 video available as part of the experience
	MGP018	As a consumer of the experience, I would like to be able to select the 360 PoV from my companion device
	MGP019	As a producer of the experience, I would like Interactive version of graphics components to be available on the Companion device i.e. Rider, team profiles, tyres, weather, track, rules, bike, tech
	MGP020	As a producer of the experience, I would like to prompt users to participate in quiz, play-a-long, betting etc.
	MGP021	As a consumer of the experience, I would like to be able to control: Options to show / hide TV graphics Selection of 3 x scaling / layout options (i.e. effective TV dimensions) Selection of 3 x experience level options (i.e. expert / fan / novice) Favourite rider / team What video is displayed on the TV picture-in-picture when available

Scope	Reference	User Story
	MGP022	As a consumer of the experience, I would like to be able to control companion presentation of: video feeds User selectable layouts Content within each layout slot User bookmark creation Participation prompts? (quiz, play-a-long, bet etc.)
Non Functional	MGP023	As consumers of the experience in the same household (context), we will use a single TV device, which is available throughout the experience.
	MGP024	As consumers of the experience in the same household (context), we will use multiple companion devices
	MGP025	As a trial manager, I expect to conduct trial session with 4-8 households participating concurrently in an as-live 'broadcast' with an overall participation of 40-80 households.

Table 1 – MotoGP High Level Technical Requirements

Early in the development phase, the project team worked closely together to refine these high-level requirements and create a set of prioritised User Experience Key Capabilities for the MotoGP service prototype, which were subsequently listed in D3.3 (6) Section 3.2.7, with Section 3.4 of the same document providing further details of these capabilities.

When the majority of the detailed design work for MotoGP User Experience had been finished and key enabling work within the platform was well under way, technical delivery was managed through a sequence of technical development milestones. Each milestone represented the completion of a new feature, and its description was used during regular acceptance testing of client and platform functionality. The description included the specifications of the TV Emulator, Companion Device, network environment and analytics requirements for each milestone. In addition, once a feature was implemented, a simple 'technical sample' DApp was often developed to show the feature working in isolation prior to integration within the MotoGP DApp. As would be expected, a degree of iterative development took place as details of the DApp design were refined, and so the milestone list was updated accordingly.

Table 2 below shows a simplified version of the milestone list at the beginning of the MotoGP trial, indicating the date on which each of the milestones were signed off and the link back to the applicable user story references from Table 1. It also indicates 5 milestones which were deprioritised to enable timely launch of the trial.

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
1	Live race video and leaderboard on TV On launching the MotoGP DApp on the TV Emulator, the main live race video should be shown full screen for duration of the timeline, with the MotoGP leaderboard overlaid on the video.	Mac Mini (Laertes build)	26 th July	MGP014 MGP015

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
	The leaderboard is populated from a file.			
2	<p>Add Picture-in-Picture videos</p> <p>PiP videos are shown overlaid on the main live race video, each framed with appropriate graphics and text.</p>	Mac Mini (Laertes build)	26 th July	MGP016
3	<p>Add Companion View Panel</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, a Viewport Panel is available on the companion. This panel:</p> <ul style="list-style-type: none"> • lists the alternative video feeds that are available as the experience progresses • allows the user to toggle presentation of each available alternative video feed on the companion and on the TV Emulator. In this first instance, feeds will be presented simultaneously on both displays. <p>When the maximum number of displayable videos has been reached, the user will need to deselect a video before a new one can be shown.</p>	Mac Mini (Laertes build)	4 th August	MGP022 MGP004 MGP003
4	<p>Add Companion Leaderboard Panel</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, a Leaderboard Panel is available on the companion. This panel should support:</p> <ul style="list-style-type: none"> • Tapping on a rider to show their 'card' • Swiping on the card to switch card views (Profile <-> Video <-> Data) <p>Additionally, it should be possible to select between the Viewport and Leaderboard Panels.</p>	Mac Mini (Laertes build)	1 st September	MGP019
5	<p>Add Companion Menu</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, the menu is available on the companion. The menu should be 'complete' as per the wireframe design (in terms of the tree of options), but not all of the menu options will be implemented. At this stage the menu should support:</p> <ul style="list-style-type: none"> • audio volume • user profile option selection (complete but not all functional) 	Mac Mini (Laertes build)	1 st September	MGP021
6	<p>Add Companion Events Panel</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, an Events Panel is available on the companion. At this stage, the panel displays a static list of events with no interaction options.</p>	Mac Mini (Laertes build)	1 st September	MGP005

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
7	<p>Add Instrumentation to Companion Panels</p> <p>While the MotoGP DApp on the TV Emulator and each Companion is running during the live race stage, one or more of the available Viewport Panels will be configured to send event data to Google Analytics to measure specific user interaction behaviours.</p>	Intel NUC (15/09 build)	21 st September	MGP008
8	<p>Drive Leaderboard from Data Playback Service</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, the leaderboard components on both TV and companion devices should update their presentation synchronised to the race videos, using the Data Playback Service.</p>	Intel NUC (15/09 build)	3 rd October	MGP010
9	<p>Add 360 Video Components</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, a Viewport Panel is available on the companion. On the tablet, this panel should additionally support presentation and selection of 360 video feeds - this will result in presentation as both a PiP video on TV and locally on the tablet. The tablet presentation should support panning around the 360 video feeds both on tablet and on TV.</p>	Intel NUC (15/09 build)	6 th October	MGP017 MGP018
10	<p>Add Companion Stats Component (with Data Playback Service)</p> <p>On launching the MotoGP DApp on the TV Emulator and each Companion, the Viewport Panel should have an option for a 'Lap and circuit times' component for presentation on the companion only. This should present lap and circuit times as defined in the wireframes, including the fastest laps for the user's favourite rider. This component should update presentation synchronised with the race video material, using the Data Playback Service.</p>	Intel NUC (15/09 build)	6 th October	MGP014 MGP010
11	<p>Add first GFX Component with entry and exit animations</p> <p>While the MotoGP DApp on the TV Emulator and each Companion is running during the live race stage, the "Battle For..." GFX component is displayed on the TV Emulator at a time authored within the timeline document.</p> <p>The component presents entry and exit animations.</p>	Intel NUC (15/09 build)	6 th October	MGP014
12	<p>Add Experience level personalisation</p> <p>On launching the MotoGP DApp on the TV</p>	Intel NUC (15/09 build)	6 th October	MGP012 MGP013

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
	Emulator and each Companion, component presentations should update to reflect the experience level (novice, standard, expert) as selected through the appropriate companion menu option(s).			
13	Add multi-channel audio selection On launching the MotoGP DApp on the TV Emulator and each Companion, presentation of the TV audio should respond appropriately when the menu options to change audio presentation are selected.	Intel NUC (23/10 build)	27 th October	
14	Add broadcaster control of layout during live race stage (replays) While the MotoGP DApp on the TV Emulator and each Companion is running during the live race stage, the layout of each can be changed at an arbitrary point in time determined during live production in order to show a replay of a specific event (which will subsequently appear in the Companion Events Panel). After the replay, each user's personalised presentation is restored.	Intel NUC (23/10 build)	27 th October	
15	Test with connection drops and restricted bandwidth All MotoGP DApp functionality available so far on TV Emulator and Companions will be tested while the network connection (wired or wireless) is dropped and reconnected, and bandwidth is restricted in order to identify performance and recovery issues.	Intel NUC (23/10 build)	1 st November (initial tests)	
16	Add On-boarding (launching TV DApp) An on-boarding launch flow is available, in which the Companion is used to launch the MotoGP DApp on the TV Emulator.	Intel NUC (10/11 build)	15 th November	
17	Add Inside MotoGP chapter On launching the MotoGP DApp on the TV Emulator and each Companion, a set of Inside MotoGP video on demand content will be available in the companion app.	Intel NUC (10/11 build)	15 th November	
18	Add Responsive TV Layout On launching the MotoGP DApp on the TV Emulator and each Companion, layout of the overlay components on the TV should respond appropriately when the menu options on the companion to change TV presentation scale are selected.	Intel NUC (10/11 build)	22 nd November	MGP003
19	Add Notifications While the MotoGP DApp on the TV Emulator	Intel NUC	24 th November	MGP020

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
	and each Companion is running during any stage, notifications are shown on both the Companion (as a pop-up component) and the TV Emulator (as an additional graphic) at a time authored within the timeline document.	(23/11 build)		
20	Add instrumentation to all components All user interactions and related events within the MotoGP DApp on the TV Emulator and each Companion are instrumented in accordance with trial requirements and centralised data capture and analytics are available.	Intel NUC (23/11 build)	1 st December	MGP008
21	Add Race Review Highlights chapter On launching the MotoGP DApp on the TV Emulator and each Companion, and reaching the end of the live race chapter of the timeline, there should be a race review chapter that follows. This will present a set of highlight race events in sequence (as reflected in the Companion Events Panel). For each event, a set of components (graphics, additional video etc.) will be presented on both TV and Companion. On selecting an event, the presentation should jump to that point in the race review highlights, and play through to the end of the chapter.	Intel NUC (23/11 build)	8 th December	
22	Add interactive playback to Companion Events Panel On launching the MotoGP DApp on the TV Emulator and each Companion, an Events Panel is available on the companion. This panel should support the following interaction: <ul style="list-style-type: none"> As the timeline progresses, events will populate in the event list, driven by the Data Playback Service. On selecting one of these events, the layout of the TV Emulator and each Companion can be changed in order to show a replay of that event. After the replay, each user's personalised presentation is restored. 	Intel NUC (23/11 build)	18 th December	MGP005
23	Complete all data-driven GFX Components While the MotoGP DApp on the TV Emulator and each Companion is running during the live race stage, all GFX components are displayed on the TV Emulator at a time authored within the timeline document, where appropriate presenting data provided by the Data Playback Service. The components also present entry and exit animations.	Intel NUC (23/11 build)	18 th December	MGP014 MGP010
A	Companion app running on iOS All companion functionality will be demonstrated	N/A	Not yet completed	In test at the time of writing.

	Milestone and Description	TV Emulator used	Date signed off	Reference and Comments
	running on an iPad and iPhone. This will be compared with the experience on an Android Tablet and Android Phone to identify performance issues.			
B	Add MotoGP Tutorial On launching the MotoGP DMAApp on the TV Emulator and each Companion, a tutorial is shown explaining the features of the MotoGP experience.		Superseded	Tutorial provided through VOD clips during Inside MotoGP.
C	Add sign-in and user profile option selection Following the on-boarding launch flow, the MotoGP DMAApp shows user sign-in and profile options selection. With sign-in enabled, user preferences (eg. experience level) will be persisted to and retrieved from the Auth Service.		Partially completed	Sign-in implemented. Nature of trial does not require persistent user preferences.
D	Inside MotoGP layout driven by experience level On launching the MotoGP DMAApp on the TV Emulator and each Companion, the Inside MotoGP components available and laid out on the companion will differ to reflect the user's experience level as defined in their user profile.		Superseded	No longer relevant as user preferences not available at this stage.
E	Bandwidth Management Service controls video QoE On launching the MotoGP DMAApp on the TV Emulator and each Companion, and then selecting multiple video streams to be presented on both the TV (live race and PiP) and Companions, the experience is adapted by the Bandwidth Management Service to accommodate changes in available bandwidth. This may involve reducing the quality of less important streams, or removing them from TV or companion entirely.		Not yet completed	In test at the time of writing.

Table 2 – Simplified List of MotoGP Development Milestones

These services have been migrated to run under a new container platform, Rancher, as described below in Section 3.3

3.2 2-IMMERSE Code Repository (GitLab)

The project partners continue to use a set of private repositories hosted on a GitLab server by IRT to manage development of the platform services, client application and DMAP Components.

We have extended our use of the GitLab feature set to support our CI/CD process, including automated service build and deployment to the edge platform instance, and use of container registries. Figure 2 below shows an example of a GitLab project Container Registry, and

Figure 3 below shows an example of a GitLab project CI/CD Pipeline.

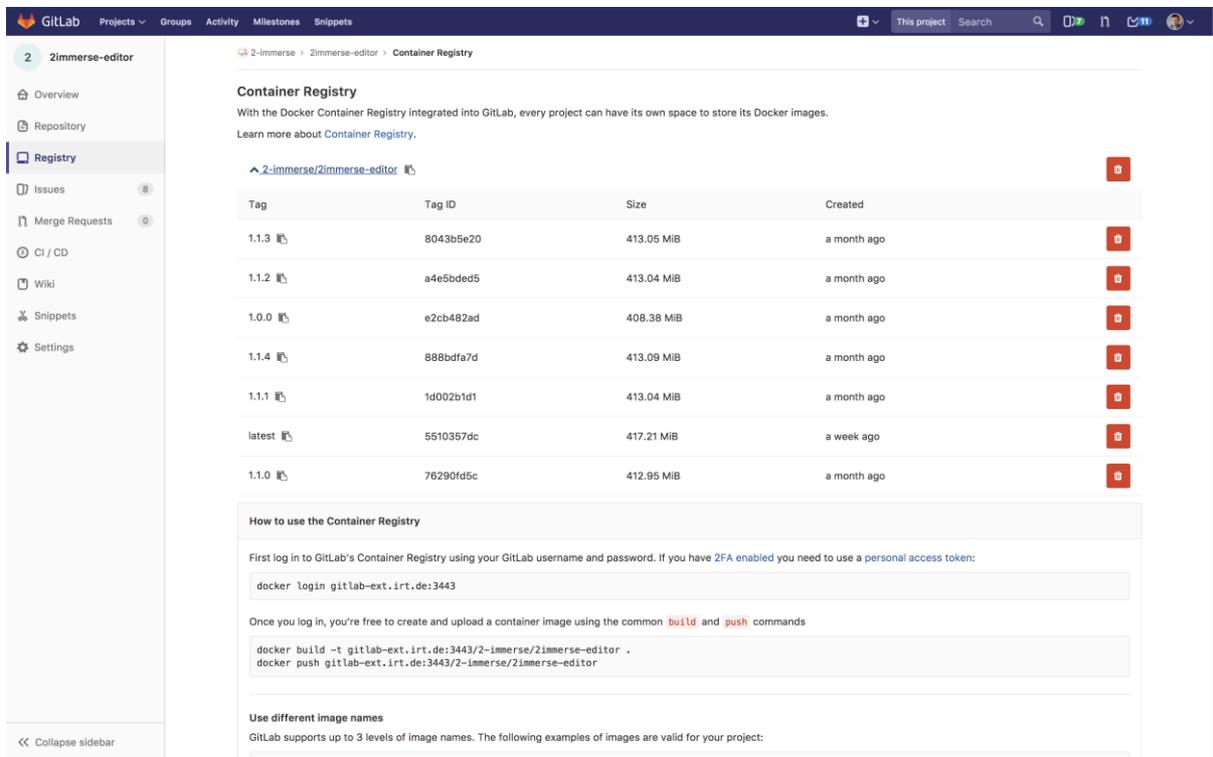


Figure 2 - GitLab Container Registry

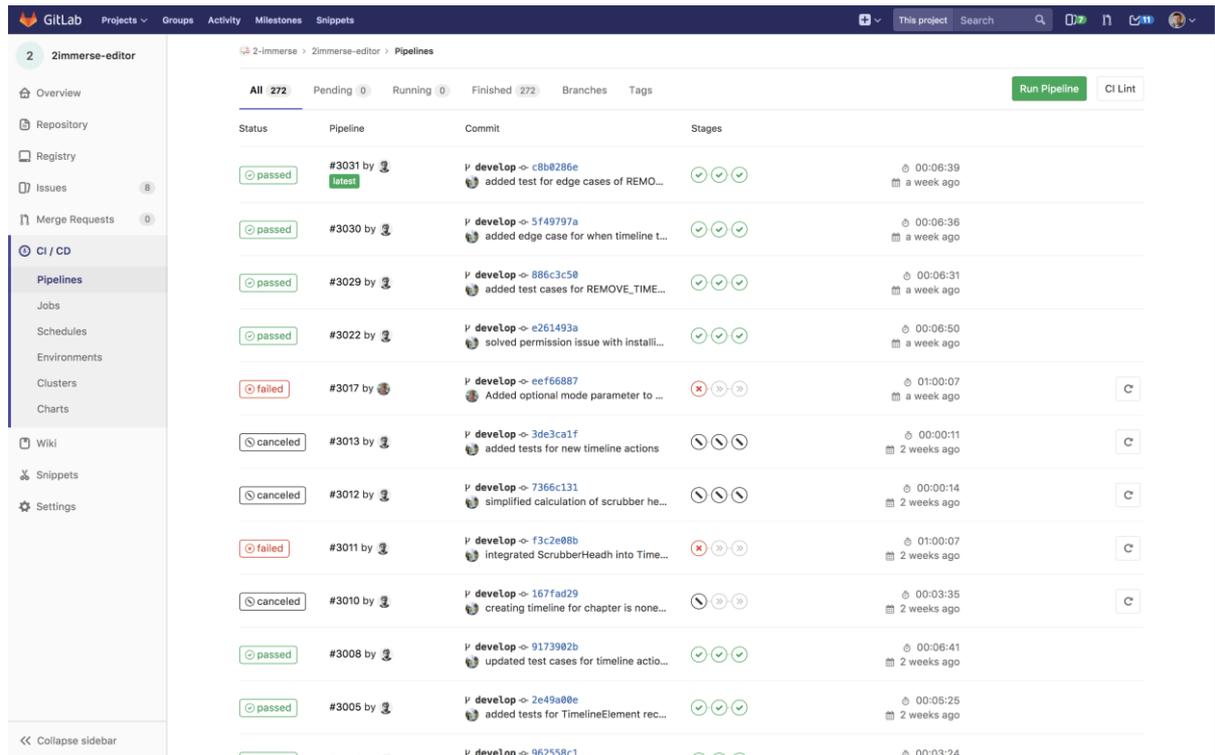


Figure 3 - GitLab CI/CD Pipelines

More details of the CI/CD process are provided in Section 4.4 of this document.

Access to the GitLab repositories can be made available on request.

3.3 Container Platform Services

During the last year, we have migrated the 2-IMMERSE Service Platform from Cisco’s OpenStack-based private cloud platform to AWS public cloud. We have then gone on to migrate from the previous container platform, Mantl, to a new platform, Rancher (still on AWS). The key driver in the decision to migrate from Mantl to Rancher was a lack of ongoing commercial sponsorship of the Mantl project. Rancher provides us with an open-source platform with similar high-level goals to that of Mantl, but with a critical mass of users and a strong roadmap. Further rationale behind these migrations is described in Section 4 below.

We have been able migrate our 2-IMMERSE services to this new infrastructure in a relatively straightforward process (and with minimal changes to our services), and are benefitting from the new container platform. These benefits include: more efficient resource utilisation, the ability to much more simply deploy new platform instances and scale resources accordingly, improved stability and a better integrated operational UX.

Figure 4 below shows the Rancher UX displaying an overview and status of the 2-IMMERSE platform services in our production environment. Figure 5 below shows the Rancher 2-IMMERSE service catalog, enabling us to ‘one-click’ deploy an instance of the service platform.

Stack:	2immerse	Add Service	Up to date	Active
Active	admin	Image: gitlab-ext.lrt.de:3443/2-immerse/auth-admin:1.0.2	Service	1 Container
Active	auth-service	Image: gitlab-ext.lrt.de:3443/2-immerse/auth-service:1.0.2	Service	1 Container
Active	bandwidth-orchestration-service	Image: gitlab-ext.lrt.de:3443/2-immerse/bandwidth-orchestration:1.0.6	Service	1 Container
Active	consul	Image: gliderlabs/consul-server:latest	Service	1 Container
Active	data-playback-dynamodb-to-iot-pusher + 1 Sidekick	Image: gitlab-ext.lrt.de:3443/2-immerse/data-playback-dynamodb-to-iot-pusher:1.2.1	Service	2 Containers
Active	data-playback-motogp-simulator + 1 Sidekick	Image: gitlab-ext.lrt.de:3443/2-immerse/data-playback-motogp-simulator:1.1.0	Service	2 Containers
Active	data-playback-request-reply-data-provider + 1 Sidekick	Image: gitlab-ext.lrt.de:3443/2-immerse/data-playback-request-reply-data-provider:1.1.0	Service	2 Containers
Active	editor	Image: gitlab-ext.lrt.de:3443/2-immerse/2immerse-editor:1.1.4	Service	1 Container
Active	influxdb	Image: influxdb:1.3.5	Service	1 Container
Active	layout-service	Image: gitlab-ext.lrt.de:3443/2-immerse/layout-service:2.0.8	Service	1 Container
Active	logging-service	Image: gitlab-ext.lrt.de:3443/2-immerse/logging-service:1.0.1	Service	1 Container
Active	mongo + 1 Sidekick	Image: mongo:3.4	Service	2 Containers
Active	redirect	Image: gitlab-ext.lrt.de:3443/2-immerse/docker-redirect:latest	Service	1 Container
Active	registrator	Image: gliderlabs/registrator:master	Service	2 Containers
Active	shared-state-service	Image: gitlab-ext.lrt.de:3443/2-immerse/shared-state-service:1.0.0	Service	1 Container
Active	timeline-service	Image: gitlab-ext.lrt.de:3443/2-immerse/timeline-service:1.5.1	Service	1 Container
Active	wallock-service	Image: gitlab-ext.lrt.de:3443/2-immerse/wallock-service:1.0.0	Service	1 Container
Active	websocket-service	Image: gitlab-ext.lrt.de:3443/2-immerse/websocket-service:1.0.1	Service	1 Container

Figure 4 - Rancher Production Platform UX

Catalog: 2immerse

- 2IMMERSE**: Launch a development 2immerse service stack. [View Details](#)
- traefik**: Deploy traefik active load balancer. [Already Deployed](#)
- MuleSoft**: API Designer. [View Details](#)
- 2IMMERSE**: MotoGP. Launch the MotoGP experience. [View Details](#)
- 2IMMERSE RENDERER**: Renderer. [View Details](#)
- Renovate**: Keep npm dependencies up-to-date. [View Details](#)
- sensu**: Sensu. Sensu monitoring platform. [View Details](#)
- sensu**: Sensu Agent. Sensu monitoring agent. [View Details](#)
- 2IMMERSE RSC**: Theatre At Home. Launch a Theatre At Home experience. [View Details](#)
- Tyk**: Tyk. [View Details](#)
- socket.io**: Websocket Test. [View Details](#)

Figure 5 - Rancher 2-IMMERSE Catalog

3.4 Backend Services

We continue to make use of Elastic Stack (Logstash, Elasticsearch, Kibana) for the aggregation, processing and analysis of service and client logs. In migrating to Rancher, we have been able to upgrade to the latest versions of these services and take advantage of the new features that they provide.

In addition, we have a number of other supporting services; either to help monitor and operate the platform, or services for the project team to use.

The operational support services include:

- Prometheus
- Tyk
- GitLab Multi-runner
- Sensu
- Docker-Hive
- Websocket-tester
- Layout Renderer
- Dashboard

The project services include

- API-designer
- Mattermost

These services are detailed in Section 4.2 of this document.

3.5 HbbTV2.0 Emulator

For the first release, we adopted a Mac Mini running MacOS X, with local services and Chrome running in Kiosk Mode. Whilst this proved sufficient for the Theatre at Home trials, it was recognised we needed to support the user journey that precedes and follows the actual experience itself, which we generally refer to as 'onboarding'. The requirements for onboarding are detailed in Section 2 of project deliverable D3.3.

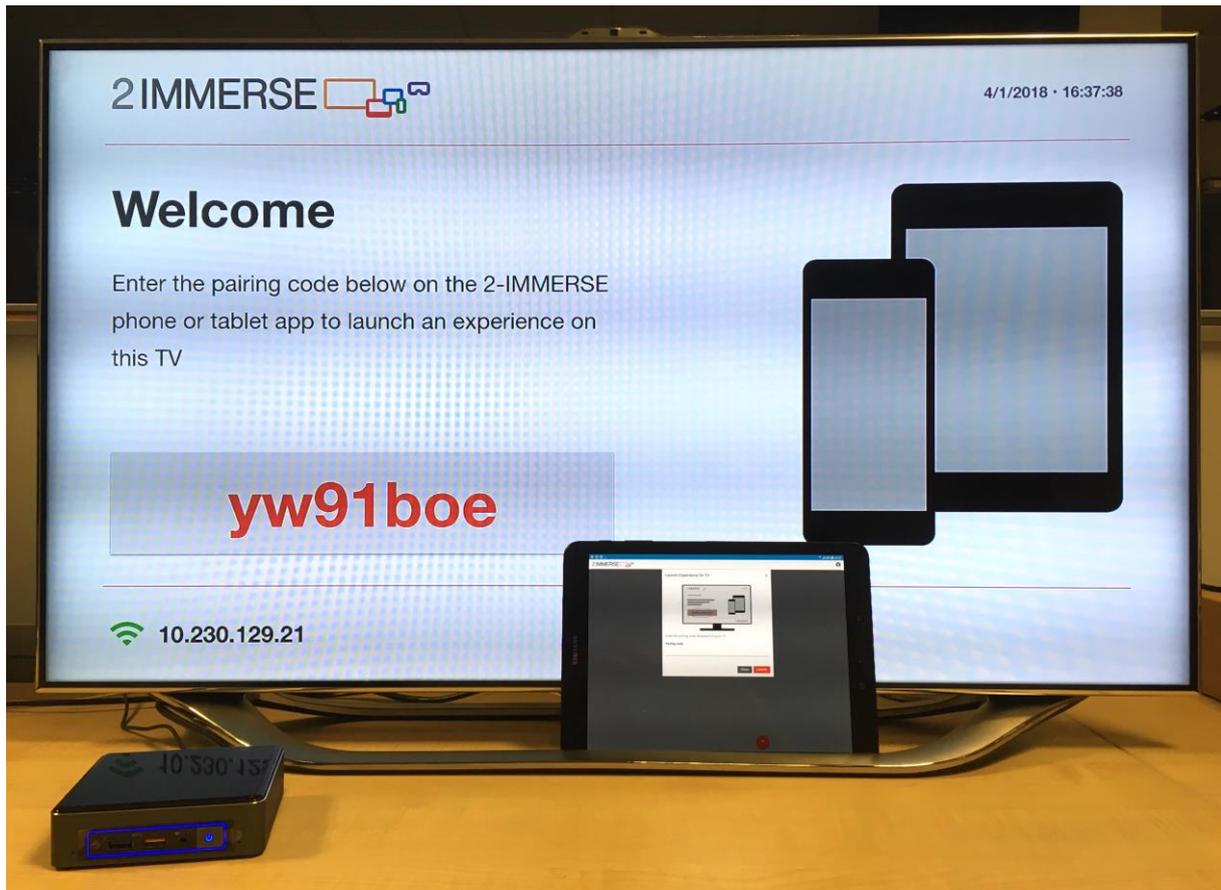


Figure 6 - HbbTV2.0 Emulator firmware showing onboarding screens

For the second release, we have developed a dedicated firmware, built on top of Ubuntu Linux, that supports a number of features including onboarding. In switching to a Linux-based firmware, we are able to more easily build and distribute firmware releases than we were able to for the Mac Mini. Figure 6 shows the HbbTV2.0 emulator firmware displaying the onboarding screens.

For the MotoGP trials the firmware has been deployed on Intel NUC small form factor PCs.

Comprehensive details of the HbbTV2.0 Emulator are provided in Section 5.4 of this document.

3.6 TV and Companion Client applications for MotoGP at Home trial

The MotoGP Service Prototype is described in detail in Section 5.3 of project deliverable D4.4 (4). The description that follows is an abridged version of this description.

The experience has been built to accompany BT Sport's presentation of a MotoGP race. That coverage has a timeline that covers three phases, the pre-race build up, the race itself and the post-race analysis. The features available to the user in our multi-screen experience vary in each of these phases. Within the experience these three phases are known as 'Inside MotoGP', 'Watch Live' and 'Race Review'.

In the following sections the features available to the user in each section are described; readers are again alerted to the video of the experience that may also help in their understanding (7)

3.6.1 Inside MotoGP

Alongside the commentary-led exposition prior to the race, access is provided to a variety of optional short-form video on demand (VoD) clips to enhance the user experience. These include GUIDE videos (that explain how to use the experience), CATCH-UP videos that bring viewers up to speed with recent MotoGP events and TECHNICAL videos that help viewers understand some of the more technical aspects of the sport.

By way of example, Figure 7 shows content from the Technical section, in this case providing some insight to the aerodynamics of bikes.

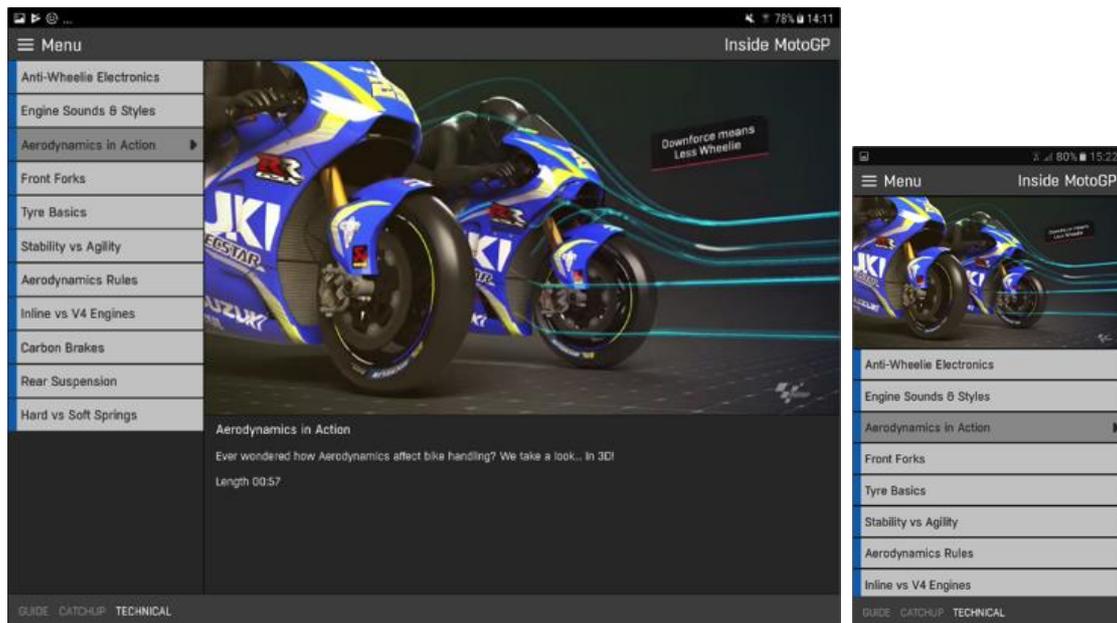


Figure 7 - Showing how, during the Inside MotoGP section that precedes the race, additional content is shown on the phone or the tablet.

3.6.2 Watch Live

The set of options available within the 'Watch Live' section comprises:

- Leader board (An interactive leader board enabling you to swipe to access different details about each rider)
- Events (A growing list of events that can be viewed as replays on the TV and tablet, often with an event consisting of multiple views presented synchronised)
- View (A list of video feeds and data components that can be selected for presentation on the companion device)
- Menu (A means of affecting the way the experience is presented across the screens)
 - TV graphics Size
 - TV presentation
 - TV audio balance
 - Your favourite rider

Once videos have been selected for display on the companion device from the View panel, there is a casting option that allows users to cast the video onto the main screen. This uses the familiar casting icons as shown in Figure 8



Figure 8 - Showing how videos on the tablet can be cast to the main TV screen as a picture-in-picture (PiP) top row and showing how the tablet can be used to remove picture-in-picture selections from the main screen (bottom row).

Figure 9 shows the interactive leader board component as presented on the tablet showing how selection expands the leader board entry and swiping reveals the bike cam (if available), split times and tyre set-up.

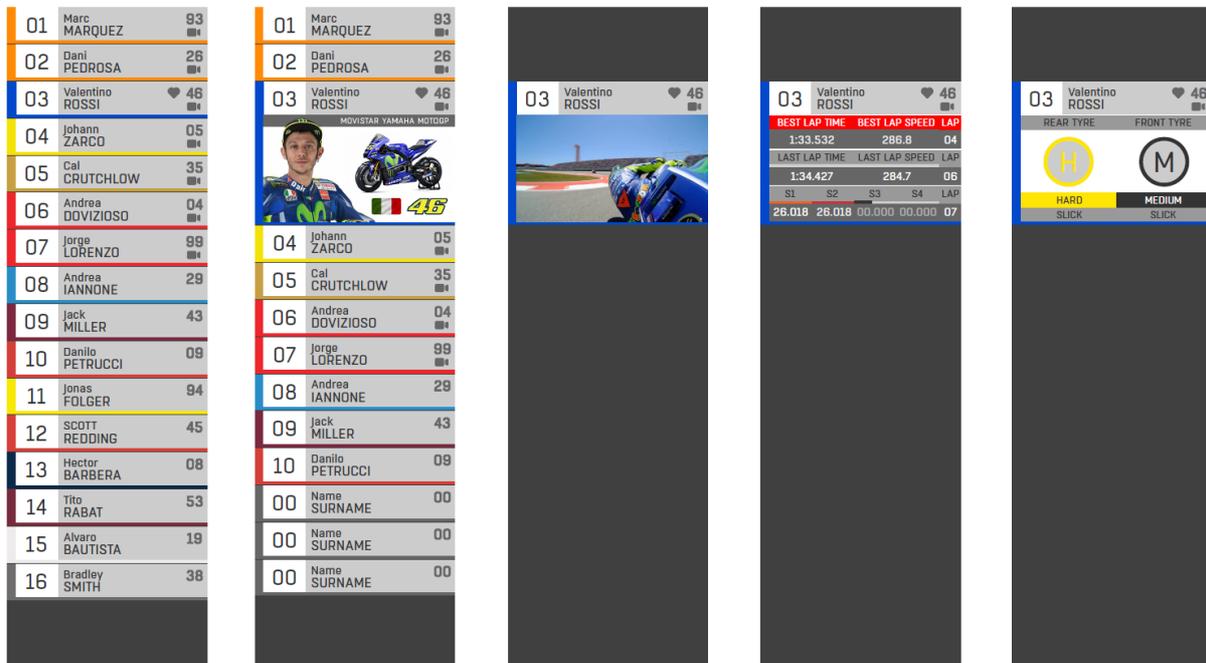
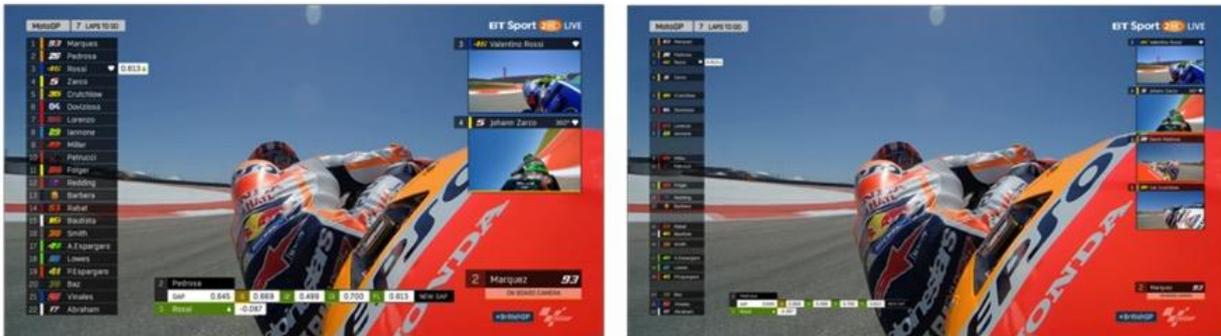


Figure 9 - The interactive leader board on the tablet, showing how selecting expands the leader board entry and swiping reveals the bike cam (if available), split times and tyre set-up

The presentation of media objects can be adapted to suit the context of the viewer through the Menu options, where context may include the size of the TV or whether the viewer has declared themselves an expert or a novice as shown in Figure 10.



Responsive TV Expert Mode layout optimised for 32", 50" and 65" TVs



Responsive TV Novice Mode layout optimised for 32" and 65" TVs

Figure 10 - This figure shows how, using the object based broadcasting approach, the graphics can be adapted to affect the size of the graphics to better suit different TV sizes (top row) or to provide more or less information on the leader-board to suit experts or novice viewers of the sport (bottom row)

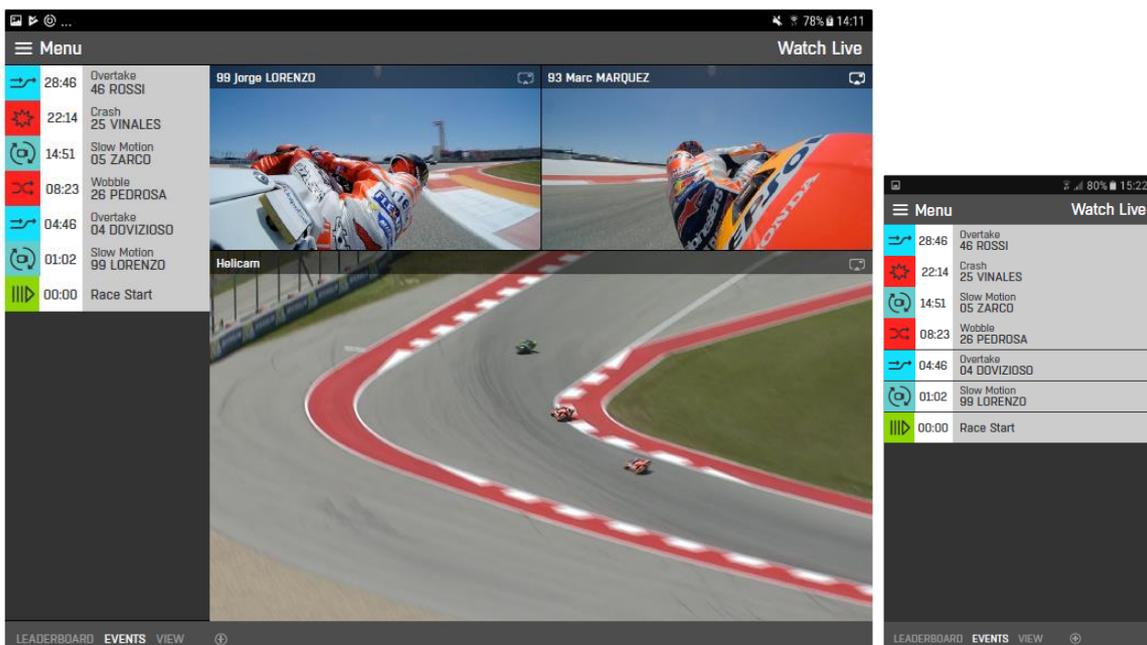


Figure 11 - Showing the events menu (on both the tablet and phone). Selecting an event during the Watch live stage initiates a replay of the event on both the TV (see Figure 12) and the companion device.

Selecting an event on the phone or tablet initiates a replay on the main TV screen (the Replay is also shown on the tablet). The events list presentation for tablet and phone is shown in Figure 11. The replay sequence on the TV and tablet is shown in Figure 12.

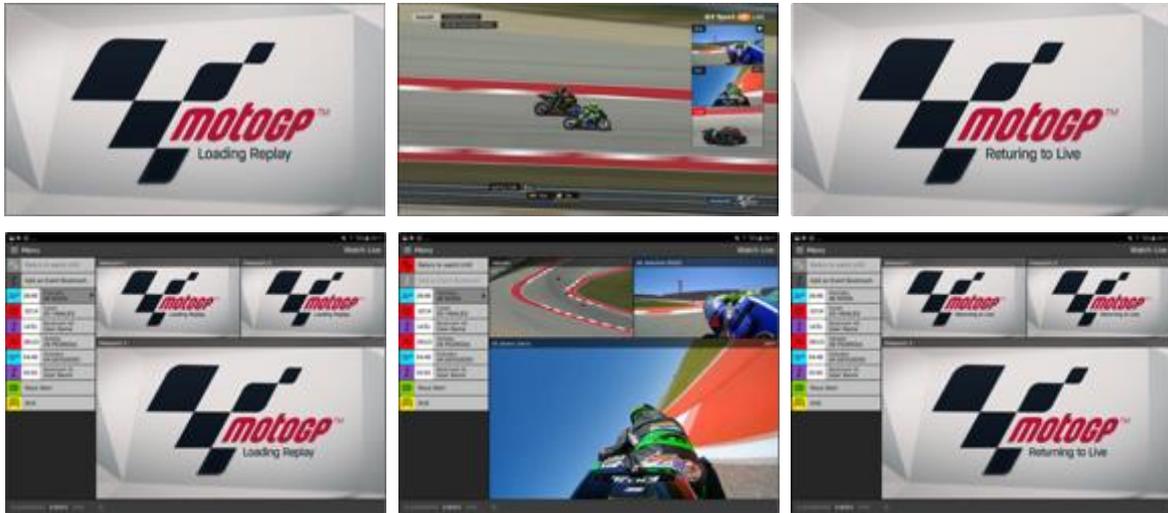


Figure 12 - The screen sequence for event replays on the Main TV (top row) and on the tablet (bottom row). The replay is preceded, ad followed by a short screen wipe animation of the MotoGP log to mirror the replays as used in the broadcast programme.

3.6.3 Race Review

Alongside the commentary-led post-match coverage presented on the TV, access is provided to a set of optional VoD replays that can be selected by the user. These are a super-set of the replays that were made available in the Watch Live section. These replays are only presented on the user's companion device but often consist of multiple views presented synchronised.

4 Platform Infrastructure

This section of the deliverable describes the infrastructure deployed to support the 2-IMMERSE service platform. (8)

4.1 Mantl Platform Developments

Toward the end of 2016, Cisco announced that the OpenStack-based private cloud platform in which we had deployed the initial platform instance would be closing down at the end of March 2017. Given this, we needed to redeploy the platform into an alternative cloud environment, and selected Amazon Web Services (AWS), on the basis that it was supported by Mantl, and that AWS pricing was competitive.

The basic deployment of a Mantl cluster using Terraform and Ansible (as done previously) worked more or less as expected and we were able to get a basic cluster up and running fairly quickly.

One area where we did make a change was for the Origin Server; previously we had used a dedicated virtual machine outside of the Mantl cluster to host and serve 2-IMMERSE DApp assets. Given the imposed transition, we felt it would be simpler and likely more robust to use AWS services directly to support this feature and we switched to using AWS S3 and CloudFront to host the 2-IMMERSE platform origin. This has proven to be a reliable solution.

In the period following initial deployment on Mantl on AWS, we made a series of changes to the Mantl cluster to improve stability of the cluster and service platform:

- Hosted under a subdomain of 2immerse.eu
- EFS volume mounts for data where appropriate (e.g. Elasticsearch indices, Mattermost data)
- Scheduled clean-up scripts to prevent disks filling up
- Upgrades to more recent backwards-compatible package releases where possible (e.g. Elasticsearch, Kibana etc.)
- Switching API Gateway from Kong to Tyk (rationale provided below in Section 5.9)
- Deployed Sensu on all nodes for improved monitoring and alerts
- Implemented a platform health dashboard

In March 2017 Cisco announced that they would no longer be sponsoring the Mantl (8) (9) project, and would turn over the Mantl-specific open source projects over to the open source community.

In practical terms this has meant that from that point on, there has been little active maintenance or development of any of the Mantl-specific open source projects that are part of the overall Mantl platform.

In the near term this hasn't been particularly problematic, it has simply meant that we couldn't rely on anyone else to fix bugs or issues, that there would be no new features, and that as new versions of software packages used in the Mantl platform were released, we could only adopt them if they were backward compatible with existing versions (as we quickly found examples where this wasn't the case, e.g. moving from Elasticsearch v 2.x to 5.x).

A small community has rallied around the open source project; however, they have had little impact on the project. It was also clear over time that within 2-IMMERSE:

- We simply couldn't dedicate the resources to trying to take on this Mantl support burden ourselves.

- We were spending increasing time managing the stability of the Mantl cluster for trials and demos.
- We also noted that the resource management of the Mesos framework foundation of Mantl was inflexible and inefficient, and we felt that more efficient options were likely to be available. By way of example we found that:
 - adding and removing Virtual Machines to an existing Mantl cluster was non-trivial.
 - Resource allocation for services could only be done on a worst-case usage basis, meaning often resources were reserved and effectively inaccessible to other services that might want to use them.

Whilst the stated goals of the Mantl project were a good fit for 2-IMMERSE, the lack of effective governance and resourcing of the project meant it was arguably not a viable solution in the long term, and it became increasingly clear that we needed to find an alternative container platform that would allow us to easily migrate the platform micro-services we had developed with a minimal change in the infrastructure and services that they relied on.

In fact, the key choices we had made in adopting Mantl, i.e. Docker as a container runtime, Consul for service discovery, Elastic Stack for logging and analytics are still valid with these still being popular choices in cloud and container platforms. Finding an alternative platform based on these technologies that allows us to deploy our existing services with minimal changes led us to investigate Rancher.

4.2 Rancher Platform

At a high level, the goals of the Rancher platform are very similar to those of Mantl; that is, a container management system with a portable layer of infrastructure services (that allow it to run in a range of public and private cloud environments). Rancher is available as open source, but also on a commercially-supported basis, and, significantly, it has a number of high-profile users in production. Rancher includes a community-contributed catalogue of popular applications and services that can be easily (in some cases ‘one-click’) deployed into a Rancher environment.

More information on Rancher can be found online (9)

Our Rancher cluster has been deployed as follows:

- Rancher management cluster (currently 3 x AWS t2.large instances)

We have set up four environments:

- Production (currently 3 x AWS t2.medium instances)
- Test (currently 1 x AWS t2.medium instances)
- Edge (currently 2 x AWS t2.medium instances)
- Ops (currently 2 x AWS m4.xlarge instances)

This compares favourably with the machines that were used to host the Mantl cluster: 8 x c4.xlarge (workers), 3 x m3.medium (control), 1 x m3.medium (edge) instances (which were struggling to run production, test and edge services).

Into the Production, Test and Edge environment we have deployed instances of the 2-IMMERSE ‘stack’, which can be 1-button deployed from the 2-IMMERSE Rancher catalogue. The catalogue entries are templates based on Docker-compose files (which was advantageous for us since we had been using Docker-compose for local platform testing). The service versions deployed on each of these instances will however vary; Edge will run the “latest” tagged Docker image from the registry, whereas Production and Test are configured to use specific release tags for each service. Our CI/CD process will auto-deploy latest builds of services into Edge as they become available.

The current set of services in the 2-IMMERSE stack is listed below in

Table 3:

Service Name	Service Description
admin	Web admin interface for auth service – see Section 5.8
auth	User Identity Management and Authentication, content decryption key distribution service – see Section 5.8
bandwidth-orchestration	Streaming media component bandwidth management – see Section 5.11
consul	Service Discovery (10)
data-playback-dynamodb-to-iot-pusher	Data Playback IoT Service push adapter – see Section 5.10
data-playback-motogp-simulator	Data Playback MotoGP live event simulator – see Section 5.10
data-playback-request-reply-data-provider	Data Playback HTTP/S interface to storage – see Section 5.10
editor	Timeline document editor backend – see Section 8.4
influxdb	Time series database used by bandwidth-orchestration (11)
layout	DMAApp component layout and adaptation to devices – see Section 5.2
logging	Client logging interface
mongodb	Object Store (12)
redirect	Service to redirect http to https
registrator	Service registry bridge for Docker (13)
shared-state	Shared component state propagation mechanism – see Section 5.4
timeline	Temporal DMAApp component orchestration – see Section 5.1
wallclock	Time synchronisation service for clients and services – see Section 5.6
websocket	Push message mechanism between service platform and clients, and between services. Includes lobby and call service functions – see Section 5.3

Table 3 – Second Release 2-IMMERSE service set

Alongside the 2-IMMERSE stack, the following additional stacks are typically deployed into these environments:

- Logspout (log routing for Docker containers) (14)
- Tyk (API Gateway) (15)
- Load Balancer (Traefik) (16)
- Prometheus (Metrics and alerting tool) (17)
- The standard Rancher infrastructure service set

The Ops environment hosts a series of stacks which are either concerned with platform operations and monitoring, or tools used by the project team. These include:

- Api-designer (Mulesoft RAML API Tool)
- GitLab-multi-runner (CI/CD runner) (18)
- Janitor (Docker image, container and volume clean up)
- Kibana (19)
- Letsencrypt (20)
- Logstash (21)
- Mattermost (messaging platform used by 2-IMMERSE technical team) (22)
- Renderer (layout rendering tool)
- Sensu (23) and Uchiwa (24) (Full stack monitoring + dashboards)
- Websocket-tester (websocket service test tool)
- Docker-Hive (scalable API testing)

We have some platform dependencies on AWS services (i.e. outside of the Rancher platform). These include:

- Data Playback service is using a DynamoDB instance hosted directly in AWS.
- An AWS ElasticSearch service instance, although we are using that with the Logstash and Kibana services in the Ops environment. This has proven more stable than the version initially deployed in Rancher.
- S3 and CloudFront for the Origin Server
- EC2 for compute (i.e. Virtual Machines)
- Route53 for DNS
- EFS for persistent data volumes

To deploy the Rancher-based platform into a different cloud platform, or on bare metal servers, these dependencies would need to be addressed.

The configuration of Rancher and its environments and stacks within AWS as described in this Section is shown below in Figure 13.

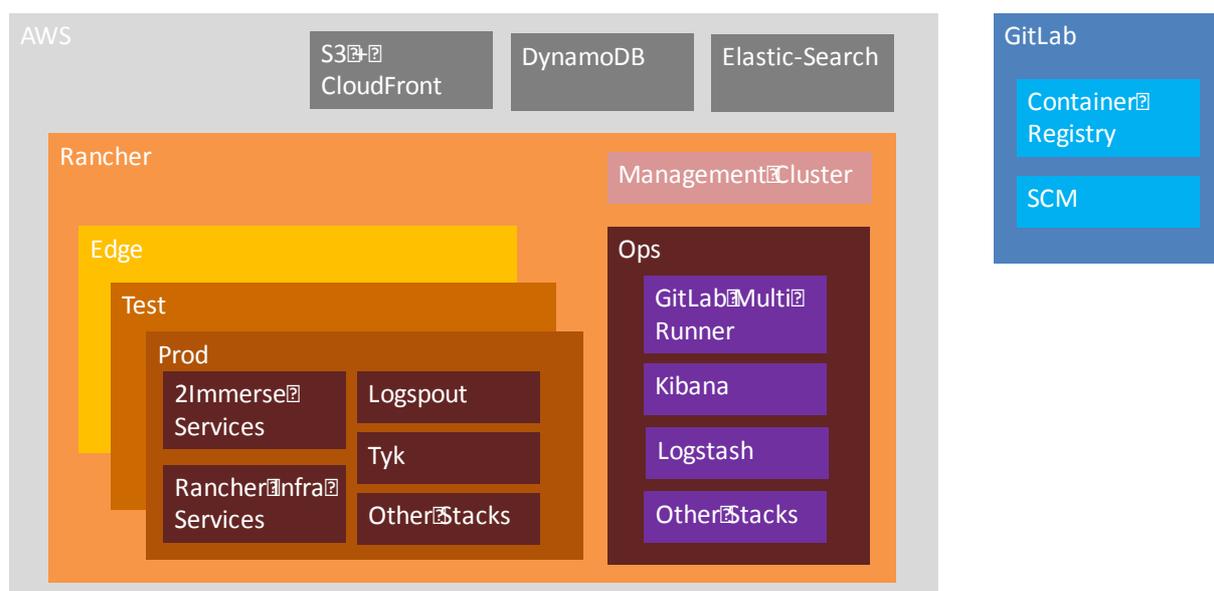


Figure 13 - Platform Infrastructure

4.3 Origin Server

As noted in Section 4.1, as part of the Mantl migration from Cisco's private cloud infrastructure to AWS, the origin server was implemented using AWS S3 and CloudFront to host the 2-IMMERSE platform origin.

4.4 CI/CD and Docker Registry

In the Mantl platform deployed for the first platform release, we were using a Virtual Machine with Jenkins installed to run our CI/CD builds (typically Docker builds) and to host a container registry for built container images.

In migrating to AWS, we made a change to the CI/CD process so that the GitLab repository would use a runner container itself deployed under Mantl to perform the builds triggered by repository commits, and the GitLab container registry feature to host built container images. This meant we no longer needed a machine dedicated to this task.

We moved to using multi-stage builds for some of the projects which simplified these build processes. Some of the projects build updated service documentation as part of the CI/CD process and publish this to the origin server.

In transitioning from Mantl to Rancher we followed this same pattern, with the GitLab multi-runner container deployed in the Rancher Ops environment. Both the GitLab multi-runner (and in fact GitLab itself) are available as community catalogue items in Rancher.

Rancher is configured to use webhook-triggered service upgrades on Edge so that the latest service builds are automatically deployed. Test and Production continue to use manual upgrades of tagged service versions.

5 Platform Services

5.1 Timeline

The Timeline service interprets the document that controls the temporal composition of all the media items and components in a 2-IMMERSE multi-screen experience across all devices: the TV and all the companion devices. Under control of the timeline document it sends commands to create, start, stop and remove the individual components and media items at the right time.

Since the first release, the Timeline service has been enhanced in a number of areas. To cater for companion devices going offline and coming online at will during the experience we have to ensure that the right components are shown, and in the right state (for example, temporal position for synchronised audio or video). To handle this, we have formalised the *component lifecycle* (init-start-stop-destroy) and implemented this in the Timeline service (as well as in the Layout service and clients).

To facilitate more reactive experiences, we have implemented functionality to allow the triggering of sections of the timeline document based on user interaction through a *timelineEvent* mechanism. This allows things such as pressing a button on a companion device to start a (possibly long) sequence of events on the television. A *repeat* container has been implemented (in addition to the *parallel* and *sequential* containers) to allow for such user interaction to happen multiple times. In addition, the repeat container could be used for other repetitive functionality in the experience.

Another area of enhancements has been the addition of an *update* facility to change parameters or layout aspects of active components under timeline control. This allows things like moving components to a different place on the screen (or even another device) or changing things like audio volume, or even media clip playback while keeping relative time position. Together with the *timelineEvent* this allows for seamless adaptation of the experience on user interaction.

A number of facilities have been added to support the new production tools which have been developed for the second release. A *temporal positioning* mechanism has been implemented that allows moving the “play head” back and forth, like the slider on a video player. Because of the structured nature of the timeline document this is non-trivial: the timeline service remembers the current state of all components, then uses symbolic execution to determine the expected state of all components after the moving of the play head and finally issues all the init/start/stop/destroy/update commands needed to move the experience to the new temporal position.

The symbolic execution mechanism is also used to allow insertion of new components into the timeline, which is the basic mechanism that underlies the live editing of a document in the Live Triggering Tool (see Section 7). For this release only insertion of components (or groups of components in parallel and sequential containers) has been fully implemented, and only insertion in the future or non-trivial: the timeline service Live Triggering Tool, insertion of components in the past and removal of components will be implemented later (to facilitate preview play in the preproduction tool).

Additionally, APIs have been added to allow the Authoring Tool backend to insert components and use temporal positioning, and interfaces to report back current document state to the tool (so it knows where it should insert new components during live triggering, and show visual feedback on component activity).

Finally, there have been enhancements in the area of platform integration: better logging, better support for containerisation and adaptations for the new platform infrastructure, modifications to cater for new versions of the layout and other services.

5.2 Layout

A number of significant feature changes have been made to the Layout service since the first release (as described in D2.3), which comprise:

- A migration from JavaScript to Typescript. Typescript is a typed superset of JavaScript that compiles to plain JavaScript. More information is available online: (25)
- A migration from Redis to MongoDB (12) as our data persistence engine (using Iridium Object Document Mapper). We had written our own object mapper for Redis but as it got more complex adopting an existing implementation was appropriate. We have seen no performance penalty in moving to MongoDB.
- Support for anchor constraints, which allow the ‘anchoring’ of components to an edge or corner of their target device / region.
- The specification of component constraint dimensions in physical units (i.e. inches), and of the display device pixel density in dots per inch (dpi). This allows the service to support responsive component presentation, i.e. that a component would be rendered at the same nominal real-world size on devices of the same pixel resolution but with different physical dimensions. This can be used with other components specified using pixel or percentage dimensions.
- The addition of a REST API for constraint management, allowing the addition, change and removal of constraint definitions for a running DMAP (for example by the Timeline service), to enable a more dynamic approach to layout.
- An upgraded Transaction API to allow components to reference constraints by ID. This allows a many-to-one component-to-constraint model. This means that multiple components can share the same constraint definition, giving more flexibility, and potentially much less repetition in layout constraint documents. It also enables rebinding of components to constraints i.e. ‘on-the-fly’ switching between different constraint definitions for a component, which gives us a much more dynamic layout capability, and which has been used in the MotoGP DMAP to facilitate presentation of replays.
- Addition of a websocket push message for ComponentProperties, so that when properties such component priority are changed, this information can propagate to other components that may need to be aware (which was the case for user control of PIPs in the MotoGP DMAP).
- Automated discovery of service dependencies through Consul (i.e. location of Websocket and MongoDB services).
- Health check API checking service dependencies i.e. if the services that the Layout service depends on cannot be discovered, then the health check will fail, typically resulting in service restart.

In particular the work in Migration to MongoDB enables the service to scale correctly.

The non-backwards compatible RAML API changes have been rolled into version 4 of the Layout service API, and the revised layout constraint syntax has been defined in a version 4 JSON schema.

Documentation for this service and its API is available online: (26)

5.3 Websocket

There has only been a single minor change to the Websocket service: the addition of a broadcast endpoint to support broadcast messages to all devices in a context.

We have spent some time looking at how to scale this service, and tried using the Socket.io driver for Redis to allow clustering. Basic tests showed that it worked, but that it requires sticky sessions from the load balancer to be useable, since it is stateful. This means that it wouldn't recover from instance failures, since the connection state would be lost with the failed instance. On that basis, we did not consider it a good solution for scaling.

5.3.1 Lobby

The Lobby service is used to co-ordinate communication services between homes (different contexts) within the Theatre at Home DMAP. The MotoGP DMAP is not designed to offer inter-context communication and so does not make use of the Lobby service. There have therefore been no changes made to this service for the second release.

5.4 Shared State

There have been no changes made to this service. The MotoGP DMAP does 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.

5.5 Logging and Monitoring

5.5.1 Logging

Internal platform logging for the second release is provided by the Elastic Stack in a very similar way to the description provided in deliverable D2.3 (3). As described in Section 3.2 above, two of the Elastic Stack components, Logstash and Kibana, are deployed within the Rancher Ops stack while Elasticsearch itself is hosted directly in AWS. Logs from 2-IMMERSE services are collected from containers by Logspout and logs from the Client Application and its associated DMAP Components are received by posting a JSON structure to the Logging Service in the same way as for the first release.

The range of fields extracted by Logstash has been extended to include:

- **contextID, deviceID, dmappID, dmappcID:** Identifiers for the current context, device, DMAP and DMAP Component respectively.
- **documentID:** An identifier for the current document being edited within the Production Tools.
- **instanceID:** A client-specific identifier used to support device discovery and synchronization.
- **api:** Denotes a REST API call, from which contextID, deviceID, dmappID and dmappcID can also optionally be extracted.
- **logmessage:** A descriptive message
- **body:** The body of a REST API call
- **xpath:** The specification of a location in a Timeline document, using the XPath format
- **source:** The service or application creating the message
- **subsource:** A specific module within the service or application

- **level:** The relative importance of the message (i.e. the log level)
- **sourcetime:** The timecode (local to the source) at which the message was created
- **fromlayer, tolayer:** Fields used to record changes in the video layer being presented from a specific DASH stream manifest.
- **env:** The platform environment currently being used, i.e. production, test or edge.

Additional saved searches, visualisations and dashboards have been created within Kibana to support debugging and also to support analysis of how the platform is used during MotoGP trials. For example, DASH layer change information can be plotted to evaluate the quality of different video streams being displayed simultaneously across multiple devices within a context. These real-time plots can be used to test and refine the operation of the Bandwidth Orchestration Service described in Section 4.11 below. A new dashboard has also been created to summarise the user identities accessing the MotoGP DMAApp and the start times of their different sessions.

One significant enhancement to logging and monitoring within the second release has been the integration of Google Analytics for monitoring user interactions with DMAApp Components. The purpose of this integration was enable the project team to answer specific research questions about the way triallists engaged with the various MotoGP DMAApp features as they progressed through the experience. Google Analytics (GA) event tracking is managed by a dedicated DMAApp Component running within the MotoGP DMAApp on every device. DMAApp Components are instrumented in accordance with a design document which has been developed in conjunction with the production team. GA events are fired by individual DMAApp Components and then collected and transmitted by the dedicated component. Custom Dimensions are used to automatically attach the contextID and deviceID to each event fired within a session scope. A Custom Metric is used to record the precise elapsed time the event is fired, which is necessary because GA does not provide flexible access to event timing information.

Within Google Analytics, several dashboards and custom reports have been created to allow triallists' behaviour to be examined, while built-in dashboards are also helpful to visualise the active triallist population. Figure 14 and Figure 15 below show examples of dashboards used to summarise a day of triallist activity and Figure 16 shows the 'Context Explorer' for one particular trial session.

It is important to note that no personally-identifiable data is recorded by Google Analytics. In order to associate a triallist household with a particular session, the contextID must be matched to the user identity which is only logged within the 2-IMMERSE platform.

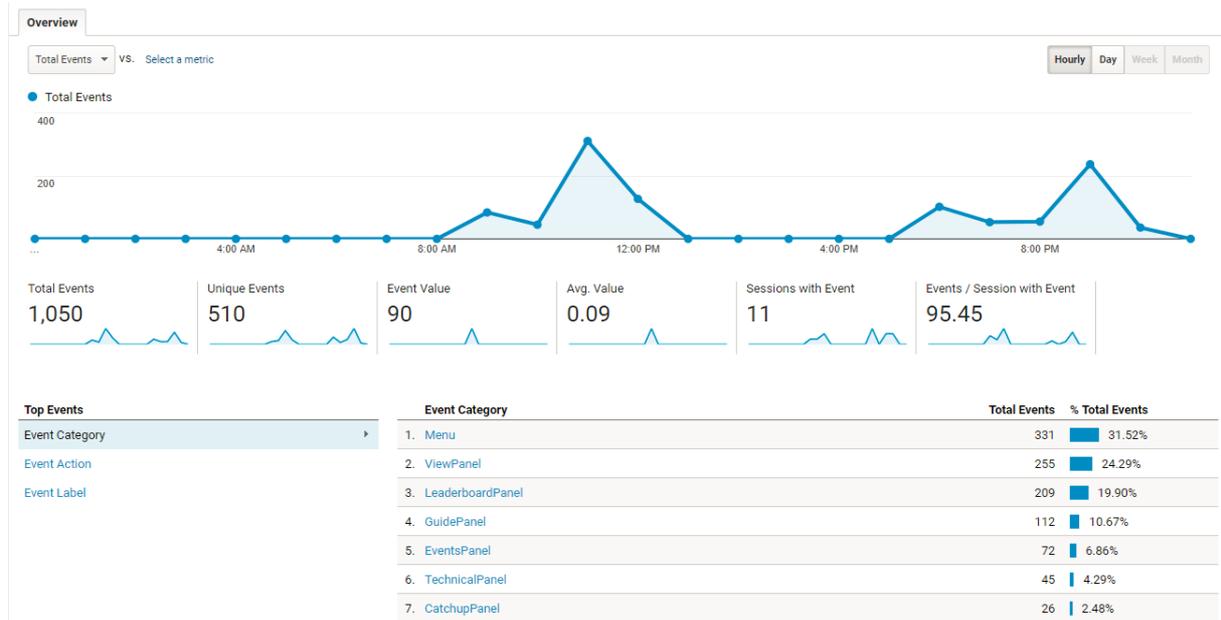


Figure 14 - Summary data for all activity on one day

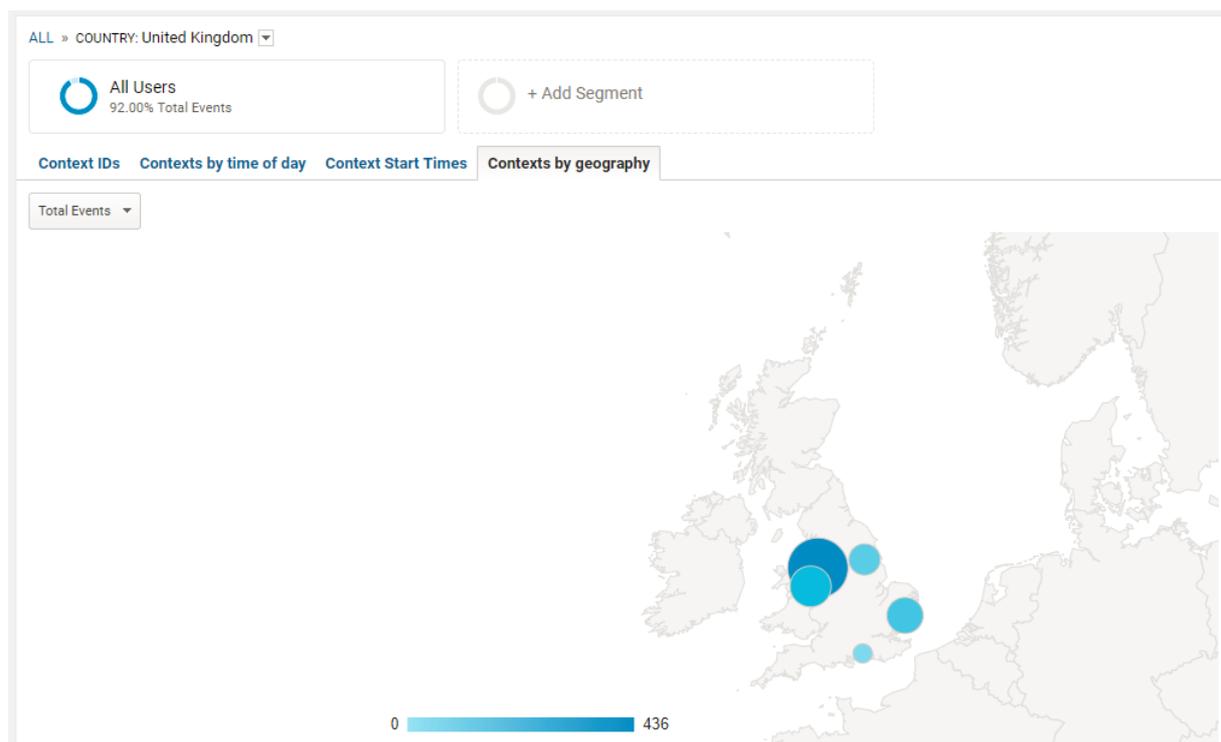


Figure 15 - Geographical distribution of sessions in one day

Context Explorer

Dec 10, 2017 - Dec 10, 2017

Context Filter
17.65% Sessions

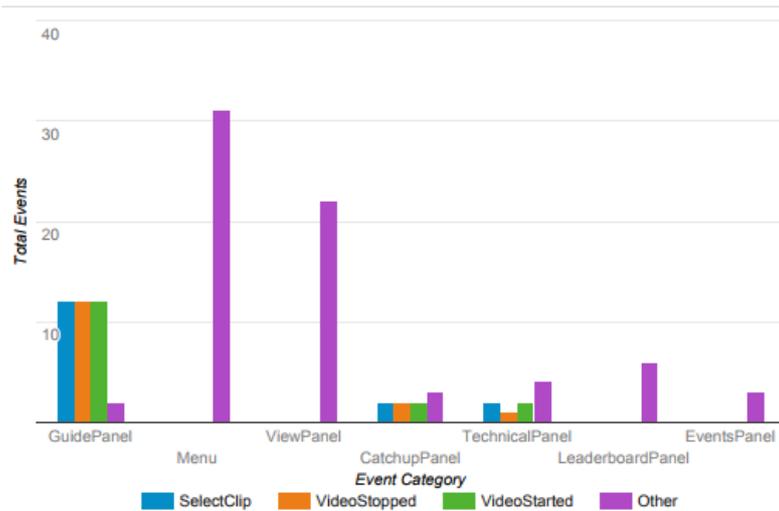
Total Events by contextID

contextID	Total Events
5a2d9e74ad9cbc0009ed4b6d	118

Total Events by deviceID

deviceID	Total Events
companion-K9VjZ7IXaQ	80
companion-pqEzVf074Y	38

UI Elements used



Total Events by Event Label

Event Label	Total Events
MotoGP™ Experience	9
/Watch Live	7
Favourite Rider	6
Menu	6

Figure 16 - Extract from 'Context Explorer' dashboard summarising user behaviour for one trial session. Note the split of activity between two different companion devices

5.5.2 Monitoring

In operating the Mantl infrastructure it became clear that an externally-visible dashboard that would expose metrics of platform health would be useful for the project team. Although much of this information is exposed in the various Mantl UIs we wanted to aggregate all of the relevant information into a readily accessible single dashboard.

To do this we built a simple dashboard using the Smashing Ruby framework that exposes the following information for production, edge and test instances:

- Most recent Docker-Hive test results (Red/Amber/Green)
- Marathon job status (Red/Amber/Green)
- Mantl Consul health check status (Red/Amber/Green)
- Machine health status (Red/Amber/Green)
- Bar graphs per cluster machine of CPU Load and Docker Volume percentage usage

- Graphs of unique context count, warning log message count, and error log message count per hour over the previous 12 hours from Elasticsearch

The per-machine stats were enabled by installing the Sensu (23) monitoring agent on each of the Mantl machines.

We also enabled an email alerting mechanism based on changes in the state of Mantl Consul health checks.

We are in the process of migrating this dashboard to the Rancher environment, although the Rancher UI generally exposes machine and service statistics much more comprehensively than the Mantl UIs.

We have deployed Sensu with the Uchiwa (24) dashboard into Rancher.

We have also deployed the Prometheus/Grafana (17) monitoring stack into the production environment as it is available in the Rancher community catalogue. This has to be deployed on a per-environment basis (rather than a single shared instance in the Ops environment, due to the way it interacts with Rancher).

5.6 WallClock Service

There have been no changes made to this service.

5.7 Synchronisation Service (Inter-Home Sync)

The first and second Releases of the 2-IMMERSE Platform rely on two main mechanisms to achieve synchronisation in the immersive media experiences realised so far:

- 1) DVB-CSS inter-device synchronisation protocols
- 2) The DMApps timeline correlation to a shared WallClock

However, both approaches suffer from particular limitations that constrain their applicability to the diverse set of possible synchronised distributed media experiences. In Annex A, we briefly describe the limitations of the current approaches, outline our proposal for a new synchronisation model and provide an architectural overview of the cloud-based Synchronisation Service. This service was not required for the MotoGP DMApp but it is expected to be relevant for the watching Theatre in Schools service prototype (to be confirmed), and an important part of our reference architecture. The new Synchronisation Service **has been implemented but not yet been integrated** with the 2-IMMERSE Platform; integration is planned in the next quarter.

5.8 Authentication

Within the original platform architecture proposed within deliverable D2.1 (1) we had identified the need for an Identity Management and Authentication Service, however, this had not been developed as part of the first release as per deliverable D2.3 (3).

The service has been implemented, along with a companion service 'auth-admin' which provides a web administration interface for the authentication service.

The principal role of the authentication service is to provide user identity and access to user profiles to 2-IMMERSE clients and services. It also functions as an identity provider for the API gateway, which enables controlled access to platform services.

The 2-IMMERSE platform uses OAuth2 (27) access tokens to secure API endpoints. All requests pass through the API gateway which evaluates authentication credentials for validity. The gateway is responsible for redirecting clients to the authentication endpoint and users can authenticate with the platform using the password flow.

Once a user is authenticated and they have unrestricted access to services, their token is forwarded with requests to backend services allowing the service to identify the user and lookup additional configuration specific to the currently logged in user (for example, user preferences persisted in their user profile).

Functionality provided by the authentication service includes:

- User Management:
 - Creation, update and deletion of users, their profiles, roles and groups.
- Access token management for authenticated users
- Access to media decryption keys for authenticated users
- Device management including supporting the on-boarding flow
- Health check

The Authentication service is implemented in Go, and uses MongoDB as its persistent user data store. The Auth Service API specification is available online (28)

5.9 API gateway

In the process of designing and developing the Auth service, and looking at the how it would integrate with the API gateway, it became clear that Kong would be a complex system to administer in our architecture and that a particular feature that we would need was only available as enterprise plugin (OAuth2.0 Introspection). This feature is available in Tyk, along with direct endpoint analytics. Tyk also provides a self-service developer portal, which is likely to be required in future as we engage with and support developers who would want to develop DMAApps, DMAApp Components and additional platform services.

On that basis, we have switched to using Tyk as our API Gateway in the Mantl infrastructure, and subsequently in the Rancher-based infrastructure.

More information is available on Tyk online (15)

5.10 Data Playback

5.10.1 Architectural Overview

The Data Playback service has been added to the platform for managing production-related non-audio/video information (data-streams) in a generic way. This could be information like timing data, scouting information, statistics etc. which come in different shape and forms depending on production type and available data sources. Today, this information is an essential component of the sports broadcasting experience.

The goal of the data playback service is to be able to capture, transform and distribute this information to clients in a scalable and re-usable way. The schematic of this process is shown below in Figure 17.:

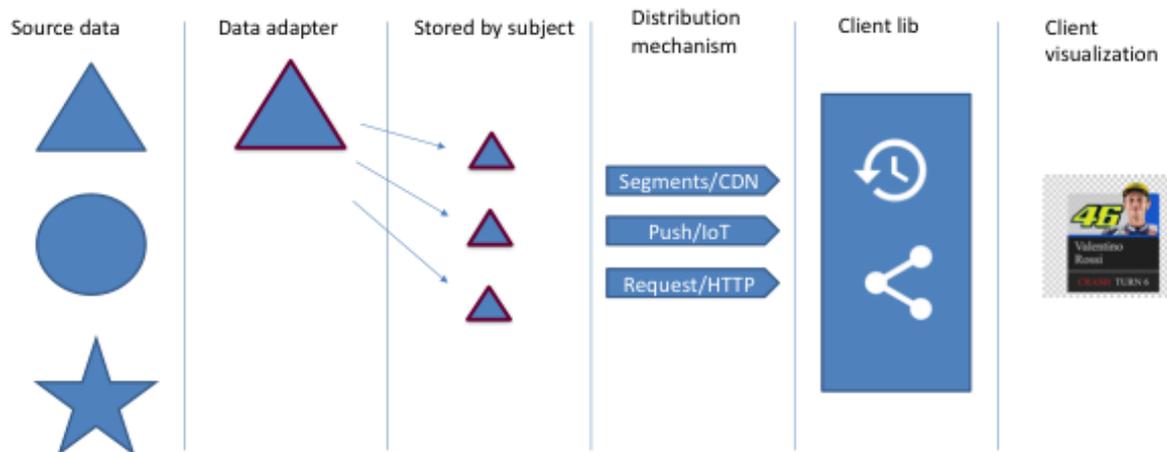


Figure 17 - Data Playback Service Overview

Different data sources are managed by specialised micro-services per source, called data adapters. The data adapters capture and transform the data for generic storage. From the storage point different mechanisms can be used to do the actual distribution. A client library helps the receiving end, and dispatches the data internally in the client.

Since it was known that the service should run under the AWS (Amazon Web Services), when designing this part of the system, applicable AWS services have been used for creating the service. While this to some extent contradicts the requirement of being portable, no services have been used where there is no equivalence available from other major cloud providers (like Google and Microsoft). There are also a lot of open source alternatives in these areas.

The above stages are described in detail in the following sections.

5.10.2 Adapter

To be able to meet different data capture requirements from different production types, a concept of an adapter is utilised. An adapter is a Docker-contained microservice hosted in the 2-IMMERSE platform specialised in capturing data and storing it in way that is accessible to the downstream system. The idea is that the Data Playback system should be able to adapt to existing production workflows instead of putting requirements on them to support a specific API. This means that these adapters need to be created, one for each new data source encountered.

An adapter needs to conform the data to a form where it can be stored in a time series to be synchronised with video. Another responsibility of the adapter is to separate the data into arbitrary subjects. The reason for the latter is to optimise bandwidth by avoiding unnecessary communication to the client.

5.10.3 Storage

As storage solution, AWS DynamoDB is used. This is provided in AWS as a SaaS (Software as a Service) and is described by Amazon as *“a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale”*. In the data playback service, it is used as a key/value store where each production is stored in a unique table. In a table, data is organised in timestamps under an event type.

5.10.4 Distribution

Three different distribution mechanisms are provided for supporting three different scenarios:

5.10.4.1 Segments/CDN

When there is a lot of data which is timewise related to video, for example when providing tracking data per player in football, the data can be served in chunks distributed through a CDN (Content Delivery Network). This mimics how DASH-encoded video is distributed and enables the data to travel through HTTP/S. The CDN is a hosted service at Amazon which means that the scaling problem is handled automatically.

5.10.4.2 Push/IoT

The Push/IoT (Internet of Things) mechanism is suitable for pushing small real-time messages, which could be exemplified by timing data, sensor data or state changes and similar. These will, compared to the segmented approach, have a much lower latency. This mechanism uses the AWS IoT service to enable scalability in a live situation. To use Amazon's own words: *"AWS IoT can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely"*.

As adapters are writing data to DynamoDB, a Docker-hosted micro-service, this acts as the glue between the storage and the IoT service. This service connects to a streaming port of DynamoDB and continuously transports all changes to the database to the IoT distribution.

5.10.4.3 Request/HTTP/S

This is a request/response mechanism over HTTP/S for retrieving backlogs of data (if a client joins a live broadcast or on-demand session which has already started) and for ad hoc queries against the storage. This is implemented as service running in Docker. This part of the service can be scaled more traditionally by running multiple instances of the container.

5.10.5 Client API

There is a client API to hide as much backend details as possible for the client. It is not necessary for the client to know which distribution mechanism is used, for example. After data is received it is synchronised and dispatched with the event timeline determined by the DMAP. For ad hoc queries, there is a specialised DMAP Component that can be instantiated as part of the DMAP.

5.10.6 MotoGP Implementation Overview

Figure 18 below shows the Data Playback implementation for the MotoGP trial. The flow of information is described by arrows. A blue background illustrates 2-IMMERSE-implemented micro-services and a yellow background shows AWS services.

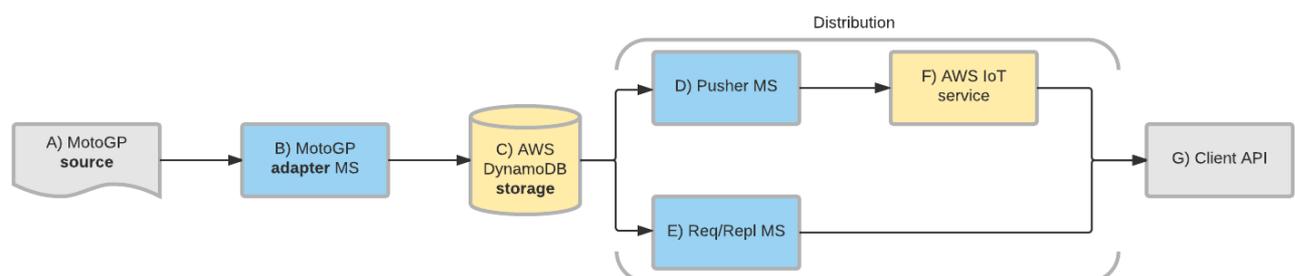


Figure 18 – MotoGP Implementation overview

The components are:

- A) MotoGP source: The source for the MotoGP trial is a text file containing timing data and other relevant race information
- B) MotoGP adapter MS (micro-service): The implemented MotoGP adapter simulates a live scenario by reading the source file and writing the data to DynamoDB in the pace determined by the race. This makes it possible to do the trial as-live, although it will be run on demand during the trials.
- C) AWS DynamoDB: Storage solution, provided by AWS as a service.
- D) Pusher MS: Link between storage and AWS IoT
- E) Req/Repl MS: HTTP/S interface to storage.
- F) AWS IoT service: Push interface to clients.
- G) Client API: Reusable library for interfacing with the backend.

5.11 Bandwidth Orchestration Service

One of the key learnings from the first release of the platform for the Theatre at Home trial was that within the 2-IMMERSE project we require architectural support for component bandwidth management. In the Theatre at Home trial we observed situations where:

1. Multiple video components would end up competing for bandwidth (typically video chat (WebRTC) and DASH video streams)
2. At the point where several components were being pre-loaded prior to a timeline transition, video bitrate would be impacted.

We note that MPEG's Server and Network Assisted DASH (SAND) specification (5) defines an architecture and interfaces to specifically address QoS and QoE support for DASH-based services.

5.11.1 Requirements

The high-level requirement for component bandwidth management is to be able to monitor and manage the bandwidth being consumed by streaming media (principally DASH video) components in a running DApp, typically orchestrated over multiple client devices by the layout service, to optimise the quality of experience. In a basic version:

- The DASH video player components on all of the client devices would report their status to a service that can monitor how much bandwidth each component is consuming, and whether it is managing to decode and present video without problems (stalls etc.).
- In the scenario where there are sustained reported stalls, or observed competition for bandwidth (i.e. bandwidth oscillations between multiple player components), the service should send control messages to the components to prevent such competition (i.e. by having one or more components switch to a lower rate representation).
- In the scenario where none of the players can move to a lower rate representation, the service should work with the Layout service to determine which of the player components might be terminated to improve the overall quality or experience for the remaining players.
- In the scenario where there is more than one instance of a video player presenting the same ABR stream, these may be managed to show a common representation where feasible (and hence avoid downloading multiple representations of the same content).
- Other platform services may benefit from being made aware when player components are experiencing issues (e.g. stalling), for example the Timeline service.

5.11.2 Design Considerations

The problem of finding the correct choice of bitrates for the laid-out components such that higher priority components will be minimally impacted at the expense of lower priority components, and that the total bandwidth required will not exceed a given limit, can be mapped to a known computational problem called the “Quadratic Knapsack Problem” (29). This problem is NP-Hard (and even NP-Complete), which means that an algorithm that finds an optimal solution will run at an exponential time. In practical terms, it means that the amount of time required to compute the optimal solution is too long to be usable, and therefore, we can only use an approximation algorithm. There are a few papers depicting such solutions (30), (31), (32), (33) however the existing implementation is very complicated and released for academic use only (34).

We'll start by assuming that all the components are using the same network (e.g. same Wi-Fi or LAN) as this is the simplest case and is the expected situation for our service prototypes.

There are two main types of scenario that will require different solutions:

1. We know what the bandwidth limit is (e.g. the capacity of the network under the assumption that it is fully available to us). This case can be solved by an approximation algorithm as mentioned above.
2. We don't know what the bandwidth limit is (e.g. we only know that components are struggling). In this case, the best we can do is to define a different problem where we try to find a minimal set of bitrate reductions (under the constraints of priorities) that will free up enough bandwidth to satisfy the maximum number of components. This problem can also be mapped into a QKP but will require a different mapping from the first solution above. However, in this case, we will never know when we have free bandwidth so we can possibly use higher bitrates if available (e.g. a shared network and some other user stops using the network, freeing up bandwidth).

In order to compute any of the above, we need to know which components should participate in the management algorithm, what their priorities and available bitrates are and what the bandwidth limit is, if one is known. The bitrates and priorities should be a part of the definition of the DMAApp while the bandwidth limit can be set when the DMAApp is initialised according to the specific instance. The Layout service can put all of this information into MongoDB so the management algorithm can find it and use it.

Some of the data matrices required by the algorithm are constant and can be computed and only once (or at least whenever the layout is changed to reflect components that are added, removed, or move to devices on other networks) and stored in the database or in memory while others change according to the collected statistics.

We can use a time series database such as Graphite or InfluxDB to store data collected from bandwidth usage reports sent from the various components. The background recommendation tasks will retrieve statistics from that DB to assess possible bandwidth starvation for components and compute required actions according to the above mention algorithms if needed.

More complex scenarios include DMAApps that are spread across multiple networks, DMAApps that share a network with other users, and DMAApps that do both. These cases are much harder to deal with and should probably be solved using the second solution mentioned above.

Since we know what the DMAApp is doing and which stream/content each component is playing, it is possible to use DASH SAND messaging to request segment caching in local gateways and thus reduce overall bandwidth. However, even a simple local cache would accomplish the task since the DMAApp is a single entity and thus caching every segment for a relatively short period of time (several seconds) is feasible and simple. When caching is involved, the above computation becomes much

more complicated as available bitrates are no longer constant and should change according to caching status (and thus even the above-mentioned constant matrices are no longer constant!), making the mapping dynamic. (33)

For an initial implementation, it will likely be simpler to use a greedy algorithm, which is farther away from the optimal solution. It has been mathematically proven that greedy algorithms can only achieve a 2-approximation to the optimal knapsack solution, meaning we'll be able to fit only up to half of the possible bitrates/prioritised-components into a known bandwidth limit than an optimal solution would fit.

Figure 19 below shows how the Bandwidth Orchestration Service (and its time series database) integrates with the existing services and clients.

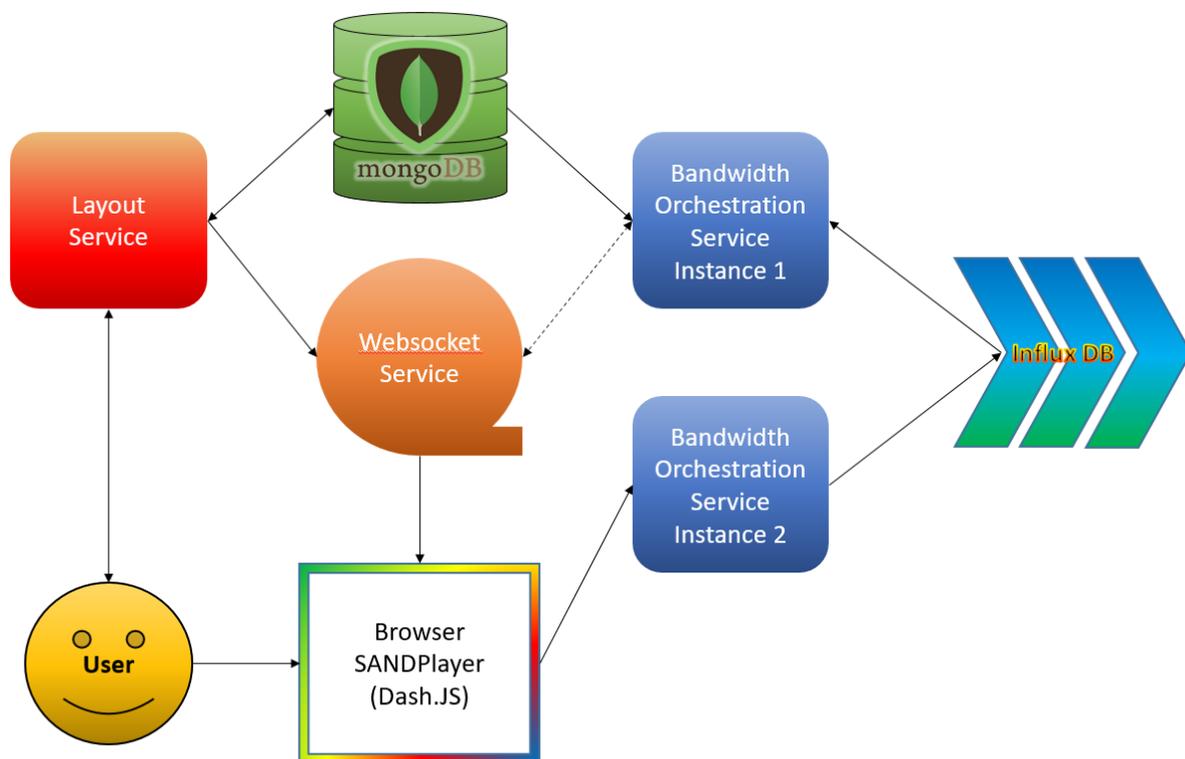


Figure 19 – Bandwidth Orchestration Service Client and Service Integration

5.11.3 Implementation

The Bandwidth Orchestration Service (BOS) has been implemented as a Node.js application. Background tasks are created and executed using the Agenda lightweight job scheduler. Agenda uses MongoDB to create queues and persist jobs. This gives us robustness against service failures and also allows us to scale up to multiple service instances.

MPEG SAND format metrics reported by DMAP components are persisted in an instance of InfluxDB.

Figure 20 below shows a sequence diagram for interactions between existing clients and services. The BOS is connected to the Websocket service. When a DMAP is launched in the Layout service, it sends an init message to the "bandwidth.orchestration" room, which the BOS is listening for.

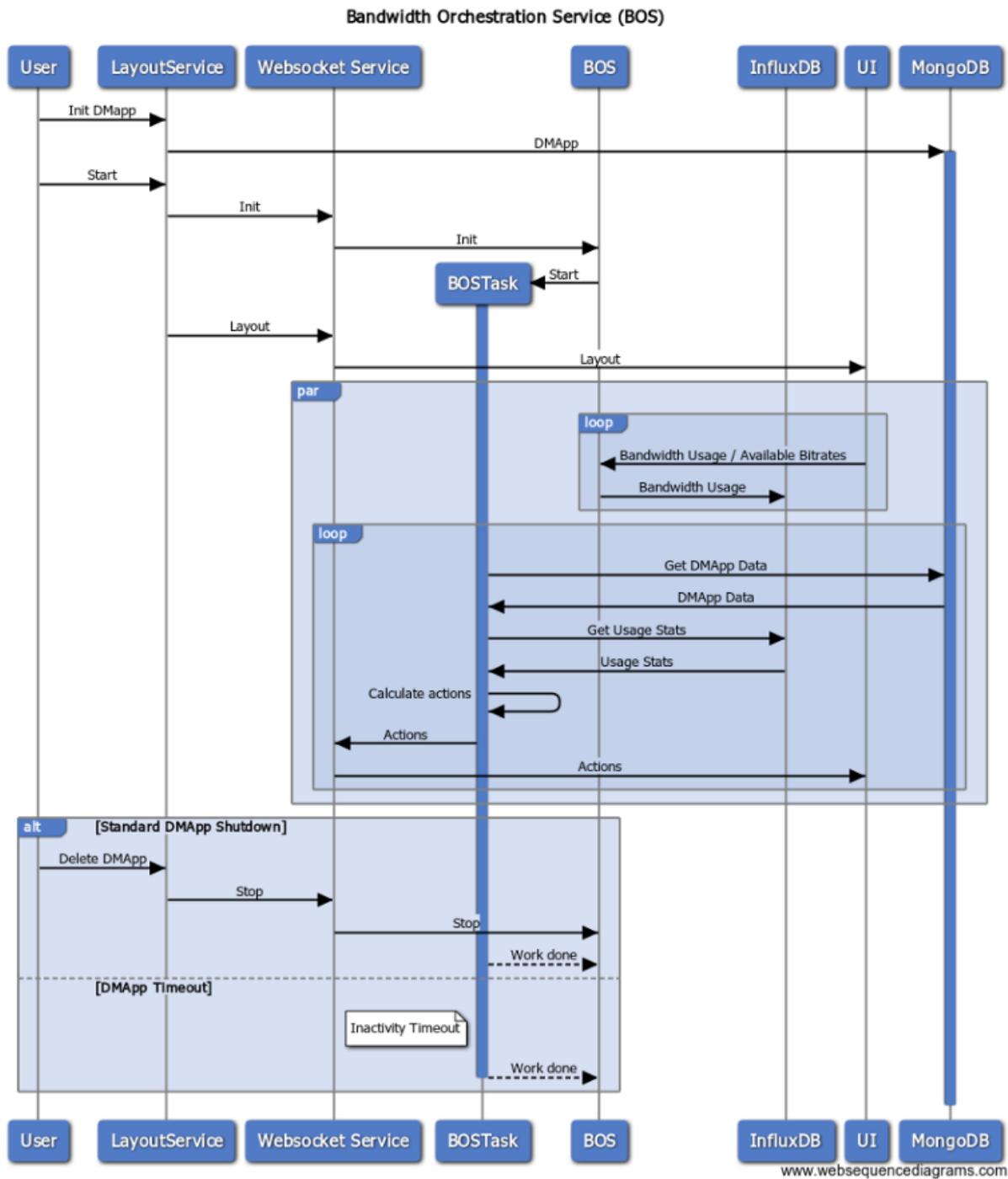


Figure 20 - Bandwidth Orchestration Sequence Diagram

The DASH player components send SAND specification metrics to BOS, which will filter out invalid data such as that which indicates using segments from the browser's cache (unrealistically high bandwidth usage). Valid data is pushed into InfluxDB.

As there is currently no server-side DASH manifest parser, we rely on the client-side manifest parsing to get the available audio/video bitrates for each player. These are also stored in InfluxDB.

The background tracking task runs every five seconds by default. BOS assumes that if no metric data has been received, the DMap hasn't started running yet. It will, therefore, not do anything until that happens.

As previously noted in the design consideration section, BOS uses a simple greedy solver to give a quick solution, and addresses the cases where the "availableBandwidth" is either known / specified, or not known. The greedy solver works by grouping components according their layout constraint-defined priority, and will then iterate these groups from lowest to highest priority, reducing the bitrates of the lower priority components.

Having determined some action recommendations for a DMAP component, it sends a message to the Websocket service in the "bandwidth.orchestration.{DMAppId}" room, to which the DASH client components are listening.

The currently-defined actions are "preserve", "downgrade", and "disable", where:

- "preserve" means to keep the current bitrate
- "downgrade" means the bitrate should be lowered to the suggested one
- "disable" means the component should be stopped to clear out bandwidth to higher priority components

5.11.4 Limitations

As previously noted, this algorithm is not optimal but is reasonable for a real-time algorithm.

Component bandwidth is averaged over the past five seconds to try and smooth out sudden spikes. However, if client bandwidth measurement is a best-effort approximation, then the metrics provided are often unstable and can lead to jumpy actions.

5.12 HLS Proxy

The HLS-Proxy is a simple reverse proxy to inject encryption key metadata (access token and key URLs) into HLS playlists. This would be required to support encrypted media playback on iOS platforms. As noted in Section 6.5.2 this is something we have been working towards but is not yet available.

5.13 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.

The currently implemented test suites are:

- 01 – layout context APIs
- 02 – layout DMAP APIs
- 03 – websocket service
- 04 – layout packer algorithm
- 05 – layout dynamic algorithm
- 06 – layout constraints API
- 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).

5.14 Server-Based Composition

Although some initial work was done looking at the architecture to support server-based composition, the project is yet to progress this to implementation.

6 Client Application Stack

This section of the deliverable describes the client application stack developed to date and its current status.

6.1 Overview

As noted in the description of the first release (3), there are a number of constituent client software parts that comprise the application stack to run a DMAP. These are summarised in Table 4 below.

	Constituent part	Content	Hosting
7	Timeline events	Timeline document or live-broadcast events	CDN, streamed or broadcast
6	DMAApp Components	video player, leaderboard, animation etc.	CDN
5	DMAApp 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 <i>Launches the DMAApp SPA and implements features common to all DMAApps</i>	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 <i>Launches the Onboarding SPA.</i>	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 <i>The DMAApp Operating System. Contains the bootstrapping application.</i>	DVB CSS, DIAL, Cordova extensions.	App store

Table 4 - DMAApp 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 MotoGP service prototype:

- MotoGP DMAApp Implementation – note that the DMAApp Components themselves are described in Section 7.
- Client API
- HbbTV2.0 Emulator
- Companion Devices
- HbbTV Showcases, tools and software libraries

6.2 MotoGP DApp Implementation

6.2.1 DApp Composition

The MotoGP DApp is composed of the elements listed below:

- **Client input documents**

There is one client input document for running as a TV, and one for running as a companion device.

These client input documents are used to launch the DApp on a TV device and on a companion device respectively.

The client input documents include:

- General configuration of the client-api
- A URL of the HTML document to include.
- A reference to the DApp control component. The TV client input document includes DApp control component configuration parameters including references to DApp configuration files.
- Declarations of layout regions and their properties.
- Options to enable additional debugging and development features when the DApp is being run for development or testing.
- The TV client input document includes URLs for the timeline and layout documents, and options to select alternative timeline documents.

- **Timeline documents**

Timeline documents are loaded into the Timeline service at Context and DApp initialisation. These include a list of all components which are instantiated during operation of the DApp, and the temporal relations between components.

The DApp currently includes the following timeline documents:

- A minimalistic timeline. This includes the minimum viable functionality to test operation of the DApp
- 3 test timelines starting from the beginning of the programme, the beginning of the MotoGP race, and the beginning of the race review section respectively. These include a small number of sample components, such that the DApp can be tested with more components running, and each programme section can be tested individually.
- A minimalistic timeline with added event definitions suitable for use with the Timeline Authoring Tool.
- 2 authored timelines generated using the Timeline Authoring Tool from the minimalistic timeline with added event definitions. The second of these timelines 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.
- Slight modifications of the above 2 authored timelines for testing and development purposes.

- **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 DMAP components within the DMAP, 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, MotoGP data, and other static files are uploaded to fixed URLs on the origin server where they can be referenced by MotoGP components as required.

- **Media**

Media assets (video and audio) are uploaded to fixed URLs on the origin server where they can be referenced by MotoGP components, configuration files and timeline documents as required.

- **DMApp control component**

The DMAP control component is an invisible DMAP Component which is referenced directly in the client input document and is loaded before context and DMAP are created or joined.

The DMAP 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 both TV and companion, the DMAP control component handles:

- Loading and parsing configuration files, and making the configuration available to other DMAP components. To avoid configuration mismatches, configuration files are loaded only on the TV control component, which distributes the parsed configuration to companion control components.
- Loading an authentication component which shows a username and password dialog, if authentication information is not already available.
- Initialising DMAP-wide shared variables: experience level and TV scaling, to an initial state, if they do not already have a valid state.
- Providing DMAP-level debugging interfaces for testing and troubleshooting.

On the TV, the DMAP control component handles:

- Adjusting the layout to fill the screen regardless of size/resolution (this is required for 4K TV support).
- Applying TV scaling variable layout changes.
- Aggregating the requested picture in pictures from all companion devices, and sending updates to the layout service to enable/disable picture in picture components on the TV and respective companion devices as necessary.
- Handling replays and event playback on the TV and where applicable across companion devices.

On companion devices, the DMAP control component handles:

- Detecting whether the current device should be considered a phone or a tablet, and adjusting the requested picture in pictures as necessary.
- Adjusting the user interface and which regions are enabled when switching between Inside MotoGP, Watch Live and Race Review modes, and when switching between phone and tablet modes.

- **Configuration files**

Some aspects of the MotoGP DMAP are controlled by separate configuration files. These files are loaded by the DMAP control component on the TV.

These files include:

- Events listing

This includes a list of all events which can be used for replays, and the details, including media URLs and descriptive text, for each event.

This configuration file is automatically generated from a spreadsheet which was created as part of the video editing and design processes.

- Inside MotoGP video listing
This includes a list of all videos shown in the Inside MotoGP section, and the details, including media URLs and descriptive text, for each video.
- Tyre configuration
This confirmation file includes MotoGP tyre information used to populate the tyre information section of the companion device leader-board panel.
- Picture in picture configuration
This configuration file includes the mapping of MotoGP on-board video streams to MotoGP riders, media URLs, and configuration options used when displaying video and 360°/virtual reality video as a picture in picture.

- **MotoGP-specific DMAPp components**

The MotoGP DMAPp includes a number of MotoGP specific components, these are described in detail in Section 7.1.

6.2.2 MotoGP data

MotoGP data as provided by Dorna is in a documented, timestamped XML format. Each line or data tag describes an event, an initial state, or an incremental change in state, at a particular timestamp. This data is required for multiple DMAPp Components in the DMAPp, for example the leaderboards and statistics/timing panels. The MotoGP data contains multiple event/data types.

For efficiency, it is preferable to minimise the number of copies of the MotoGP data which need to be downloaded and processed on each device.

As multiple components DMAPp Components require use of the MotoGP data, a common data sink module was developed which is used by all components which consume MotoGP data (except those described in Section 6.2.2.1). The data sink module builds and updates a model of the current MotoGP state which is synchronised to the DMAPp clock, by processing the MotoGP data and incrementally applying state updates and recording events at the time given by the timestamps in the data. A read-only view of this model is exposed to the owning DMAPp Component, which can query it and respond to emitted change and notification events as required. Using an updated model instead of forwarding incremental updates to DMAPp Components improves performance and allows both forward and backward seeks in the data due to clock step-changes to be handled correctly and efficiently.

The data sink module provides an abstraction where the interface for receiving data is independent of the type of the input. The raw input data (XML or otherwise) is not exposed to the DMAPp Component which uses the data sink module. Supported input types include:

- Single XML file (as provided by Dorna)
- Multiple XML files, with manifest
This is produced by processing the single XML file from Dorna with a script which divides it into smaller chunks which can be independently loaded, partially indexes it, and stores the details of the resulting chunk files and indexes in a separate XML manifest file. This improves efficiency and reduces client start-up time relative to using a single XML file.
- Data playback service
The data playback service provides a data streaming and subscription mechanism where individual event/data types can be acquired and streamed in real time. The data is returned in a JSON encoded form which differs from that used in the XML documents.

The data sink module acquires multiple event/data types and re-merges them to provide a consistent and updated model of the current MotoGP state.

- **Spooler DMAP Component**

This is a DMAP Component which itself uses a data sink module with one of the above 3 input types.

Data sink modules which use a Spooler DMAP Component as an input present the data model owned by the Spooler to their owning DMAP Components, instead of creating an independent model. As DMAP Components only have a read-only view of the data model, an individual component cannot inadvertently modify data used by another component, even though they are using the same data model.

In the MotoGP DMAP there is one Spooler component per device, and all other MotoGP data consuming components (except those in Section 6.2.2.1) use it as their data source. Consequently (excluding Section 6.2.2.1), each device only needs to download one copy of the MotoGP data and update one MotoGP data model, which has a significant performance benefit relative to each component individually acquiring and/or processing data.

6.2.2.1 Direct receive from data playback service

Some graphical animation DMAP Components connect directly to an instance of a data playback service receiver component, and consume the encoded form output by the data playback service directly. Only one instance of the data playback service receiver component as used by these components exists per device, this reduces the total quantity of data to be downloaded and processed. The reason that we have a mix of two different approaches in the implementation is because we are using the MotoGP trial to understand how to evolve data playback in the future.

6.2.3 Deployment

For testing purposes 3 copies of the MotoGP DMAP (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 MotoGP 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, DMAP 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 MotoGP git repository both locally and relative to the copy on the server before permitting a deployment to production and test.

6.3 Client API

6.3.1 DMAP Component Interface

DMAP 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 DMAP 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

DMAApp Component, but is instead designed to ensure flexibility of implementation, and support simple conversion or wrapping of existing 3rd party functionality into DMAApp Components.

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

Existing DMAApp Components designed for the first release of the interface continue to work with the second release and with intermediate unreleased versions. This includes DMAApp Components for the Theatre at Home trial.

Notable changes to the DMAApp Component interface are listed below:

- DMAApp Components can now directly instantiate DMAApp Components as children of themselves. This changes the organisation of DMAApp Components from a flat list to a tree. This significantly aids component re-use as a DMAApp Component can use the functionality and user interface of another DMAApp Component by composition. For example, a DMAApp Component can be built as an aggregate of one or more existing smaller DMAApp Components.
This change required various minor additions to the interface, however components which are instantiated as children of another component and/or which do not instantiate their own children are not required to be aware of this change in functionality or to the interface.
- In a change from the first release, the Layout Service may now indicate that a DMAApp Component is to be temporarily stopped and made invisible, but may be resumed in the future instead of being destroyed. Existing components do not need to be aware of this new state to function correctly, however the interface has been extended such that components may choose to handle this state in a custom way if necessary, or to enter this state of their own accord.
- The DMAApp Component interface has been changed such that components can now choose to defer their own destruction or their transition to a hidden state. This is useful for components which should show a visual transition when being removed, instead of disappearing from the screen immediately. Existing components and those which do not require an exit transition are unaffected by this change.
- Various additions have been made to the DMAApp Component interface to support additional and extended forms of component parameter filtering, validation, transformation and propagation to child components. Existing components are unaffected by this change.
- More DMAApp Component state change events have been added. These include events when the DMAApp Component is attached to or detached from the Document Object Model (DOM) document. Existing components are unaffected by new events which were not previously part of the interface.
- The DMAApp Component debugging and logging interfaces have been extended to support further enhanced component diagnostics and debug control.
- The interface by which DMAApp Components and other Client API module instances can send and receive local or remote messages from each other has been significantly altered. This is due to design choices in the original interface and addressing format which could not support sending messages to DMAApp Components which are descendants of other components, as now implemented. The updated interface and addressing format handle this new case and previous use-cases. This is not a significant backwards compatibility issue as this interface was very sparsely used in the first release.
- DMAApp Components may now be assigned to a separate local clock domain for media synchronisation purposes. This is to facilitate the use case where a group of media components are synchronised with each other instead of being synchronised with the wider DMAApp.

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: support for transparent propagation and aggregation of signal values between different devices, further value transformation/filtering mechanisms, and support for associating signal instances with type-specific names of either device or context scope.
These changes allow otherwise unrelated DMAPp Components or other client module instances to share, aggregate or observe state keyed by name either locally to a single device or across multiple devices in a context, in an explicit and straightforward way which is robust and minimises overhead.
- Addition of various utility methods and interfaces to ease development in general and encourage robust and reliable code. These include areas such as immutability, run-time error detection, and event management.

6.3.2 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:

- Addition of a documented input document format which can be used by the Client API to load and configure a DMAPp. This removes the need to develop custom loader pages for each DMAPp and reduces the overhead associated with developing a DMAPp. This includes a mechanism where the input document can specify optional features which can be enabled by developers and testers to run DMAPps in a debug or testing mode to more easily facilitate development and testing.
- Robustness and reliability improvements in various areas of the client including: media playback, media synchronisation, and error handling and reporting.
- Additional debugging commands and interfaces to facilitate easier and more rapid troubleshooting and development.
- Changes to facilitate use of the Client API with the Production Tools. These include remote control mechanisms to change the current clock time and send control messages to the DMAPp.
- Changes to the media player component to facilitate reporting to and control from the Bandwidth Orchestration service.
- Window resizing and device orientation changes are now handled and forwarded to the Layout Service as necessary.
- An additional interface to facilitate logging of Google Analytics events to a DMAPp Component which assigns itself as the handler and forwarder of analytics events.

6.4 HbbTV2.0 Emulator

6.4.1 Firmware Overview

The firmware and the TV emulator device have been developed to meet the needs of the public service trials. Together they have evolved into a fully functional set-top-box that will also be used for the Football at Home and Theatre In Schools service prototypes.

The firmware is a software bundle that emulates a subset of HbbTV2.0 features. It has been extended to satisfy the requirements of our four service trials and to help users configure multiple TV devices and launch experiences; a process called “on-boarding”. It also provides a dedicated wireless access point to improve the quality of network service in the venue in which it operates.

This section below gives an overview of the components of the firmware; full details of the individual components and how they work together to deliver a complete system and provided in Annex B.

6.4.2 High-level Components

The 2-IMMERSE TV emulator firmware consists of the following high-level components:

1. Operating System
2. HbbTV2.0 emulator services (App2App server, DVB-CSS server and DIAL server)
3. On-boarding system
4. Network connectivity management layer
5. Integrated Wi-Fi router/access point
6. Captive portal
7. Admin portal
8. Web kiosk service
9. SSH tunnel service for remote debugging of Chromium web pages
10. Web server

The diagram below gives an overview of the services running on the TV emulator. Each box highlights a system service started on boot, together with its main dependencies.

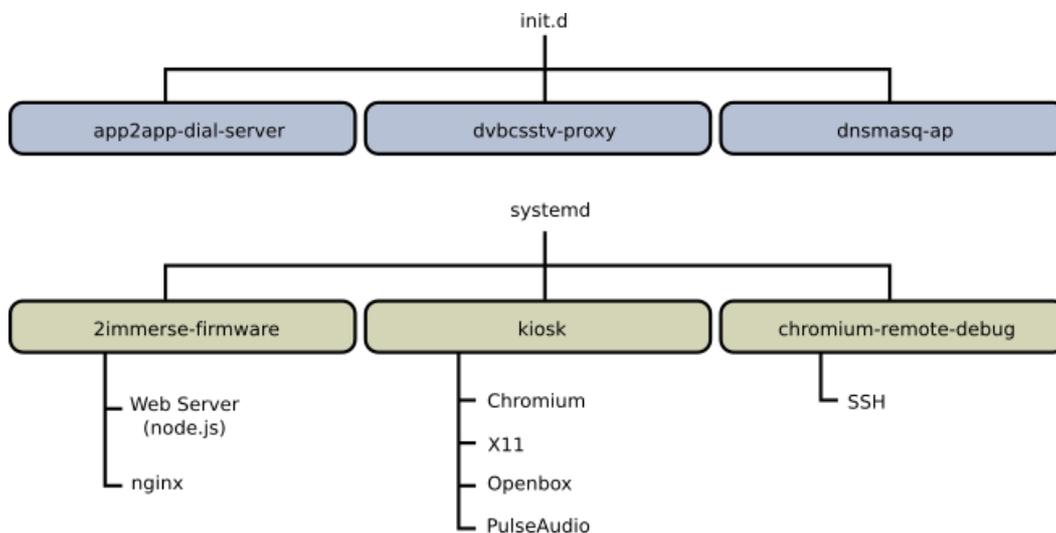


Figure 21 - Overview of firmware services running on the TV emulator device

6.5 Companion Devices

6.5.1 Android

Platform support for Android is mostly unchanged since the first release. The only notable change has been to the version numbers of dependencies to address minor issues which have been fixed in those dependencies since the first release was developed.

6.5.2 iOS

Platform support for iOS has been improved since the first release to address functionality which was incomplete or missing. Most notably this includes:

- Changes to device discovery to fix reliability and incompleteness issues.
- Changes and bug fixes to correct iOS-specific layout and styling issues.
- Support for encrypted HLS media playback is currently in development but as of the second release is not fully operational.

6.6 HbbTV showcases, tools and software libraries

2-IMMERSE partner IRT have developed showcases to demonstrate features of the HbbTV 2 specification to broad audiences at a number of occasions (including IFA and IBC 2017). These showcases have been helpful in convincing various stakeholders, including decision makers in broadcasting houses, of the HbbTV 2 feature set. Also, we have shared the showcase applications with HbbTV manufacturers for interoperability testing with their early product prototypes. Thus, IRT have been able to stimulate market adoption of HbbTV 2 from both ends of the value chain. By-products of the showcase development are a set of software tools and libraries. A subset of these libraries, the Android libraries for companion-screen synchronisation and automatic discovery of HbbTV devices, is already in use in the current 2-IMMERSE Client application stack. We plan to integrate the whole set of libraries and tools with the 2-IMMERSE platform in year three of the project (2018). On this basis, a validation of the 2-IMMERSE services is planned to be run on actual HbbTV 2 devices.

Annex C provides details of the showcase tools and software libraries for use in HbbTV applications: Android and Windows-platform-based companion applications that have been developed. It also introduces server-side components, including a stand-alone tool to generate a media timeline for Transport stream and a Material Resolution Service.

7 Multi-Screen Experience (DApp) Components

This section provides details of the specific capabilities of the DApp Components designed and developed for the Second Release, with notes on how they are being used within the MotoGP trial.

It should be noted that the scope of the MotoGP trial is restricted to a single TV Emulator device, which presents content on the main household TV, and multiple companion devices, which present additional content and allow user interaction with the experience.

7.1 DApp Components available in the Second Release

Table 5 below provides a summary of all of the DApp Components included in the second release, and hence for the MotoGP service prototype. The presentation of a track-based sport of course has very different requirements to Theatre at Home trial experience which was described in the first release (deliverable D2.3). While basic component functionality such as video playback and the presentation of images and formatted text has been re-used, a significant number of new components have been developed. Many of these components address the requirement to present 'live' and changing data about aspects of the MotoGP race through animated graphical elements on both the TV emulator and companion devices. While the table below focuses on their application to MotoGP, it is expected that a large amount of their functionality will be re-used when implementing the forthcoming Football at Home experience.

Name	Description	Comments
Video	This is an HLS/DASH player which is capable of playing out video and audio on the TV emulator or companion device, at video resolutions up to 1080p25 with stereo audio. Video playback can be synchronized within and between devices.	In the MotoGP DApp, separate instances of this component are used to present the main race video and two audio tracks (commentary and ambient audio, which can be independently controlled) on the TV emulator.
HTML Snippet	This presents formatted text-based content on the TV emulator or companion device.	In the MotoGP DApp, this 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.	In the MotoGP DApp, this is used for static graphical overlays including the channel and MotoGP logos.
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	In the MotoGP DApp, 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.

Name	Description	Comments
	depending on the media type.	
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 DApp, 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.
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 DApp, 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.
Laps Remaining	This graphic presents the number of laps remaining, changing as the race progresses.	In the MotoGP DApp, the number of laps remaining is determined by live data provided by the Data Spooler component.
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 DApp, 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.
Companion Stats	This presents a table of lap time statistics for each rider on the tablet companion device.	In the MotoGP DApp, 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.
Companion Control Panel	This component presents an interactive control panel on the companion device which enables	In the MotoGP DApp, the companion control panel can be switched between three different

Name	Description	Comments
	<p>user interaction with MotoGP content.</p>	<p>modes during the race:</p> <p><i>Leaderboard</i> – which indicates the position of each rider in the race and allows the user to view additional information about 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.</p> <p><i>Events</i> – 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.</p> <p><i>View</i> – 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.</p> <p>When the race is finished, only the <i>Events</i> mode is available.</p>
<p>Companion Panel Switcher</p>	<p>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.</p>	
<p>Companion Top Bar</p>	<p>This presents a title bar on the companion device which includes the current status of the DApp and provides an interactive drop-down menu to customise and control the experience.</p>	<p>In the MotoGP DApp, the drop-down menu enables the selection of TV Graphics Scale, Presentation Style, Audio Presentation and Favourite Rider.</p>
<p>Inside MotoGP Panel</p>	<p>This component is a self-contained interactive video-on-demand player for the companion device which enables</p>	<p>In the MotoGP DApp, this component is presented during the build-up stage before the race starts and allows multiple users to</p>

Name	Description	Comments
	the user to browse and watch different video clips.	independently watch different video clips taken from three categories: Tutorial, Technical and Catch-up.
Companion Notification	This presents a pop-up message on the companion devices to inform the user of important information.	In the MotoGP DMAP, 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.
Adobe Animate	This is a generic component which supports the playback and control of a JavaScript-based animation exported from Adobe Animate.	<p>Several MotoGP-specific animations were developed using Adobe Animate and derived from this component. These are listed below. The information presented in many of these components is determined by live data provided by the Data Spooler component.</p> <p>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.</p>
<i>Info-Rider</i>	This animation indicates that a specific rider is currently featured in the main race video.	
<i>Leading-Group</i>	This animation indicates that the current race video is showing the leading group of riders.	
<i>Info-Onboard</i>	This animation indicates that the current race video is showing the on-board camera from a specific rider.	
<i>Battle For</i>	This animation indicates that the current main race video is showing the battle for a specific position in the race.	
<i>Battle For Multi</i>	This animation provides more information on the battle for a specific position in the race, especially when more than two riders are involved.	

Name	Description	Comments
Lap-Comparison	This animation provides a comparison between the lap timings of two specific riders.	
Multi-Lap-Comparison	This animation provides a comparison between the lap timings of multiple riders over multiple laps.	
Radar	This animation provides a graphical representation of the race circuit and the positions of the riders on it.	
Split	This animation shows the distance (time gap) between two specific riders.	
Fastest Lap	This animation indicates that a new fastest lap has been recorded, providing rider and timing details.	
Info-Crash	This animation indicates that a crash has taken place and provides information about it.	
Info-Incident	This animation indicates that an incident (other than a crash) has occurred during the race and provides information about it.	
Info-Championship	This animation provides a 'live' view of the current top positions in the MotoGP championship table, based on the riders' positions in the race at that time.	
Info-Result	This animation confirms the winner of the race.	
Info-Standings	This animation shows the current top positions in the MotoGP championship table.	
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	This is a non-displaying component.

Name	Description	Comments
	modes.	
Spooler	This is a data spooler component which enables live data to be distributed to DMAApp Components which require it.	This is a non-displaying component. See section 6.2.2 for more details of this component and propagation of MotoGP data between components.
Google Analytics	This component aggregates user interaction events generated by other DMAApp Components and sends them to Google Analytics.	This is a non-displaying component.
IoT Data Fetcher	This component enables live data to be received from the Data Playback Service.	This is a non-displaying component.

Table 5 - DMAApp Components

8 Production Tools

Recent technical advances make authoring and broadcasting of object-based multi-screen experiences possible. Most of the efforts to date, however, have been dedicated to the delivery and transmission technology (such as HbbTV 2), and not to the production process. Media producers face the following problem: there is a lack of tools for crafting interactive productions that can span across several screens. Our intention is for 2-IMMERSE to contribute appropriate and adequate production tools for object-based multi-screen experiences that can reshape the existing workflow to accommodate the new watching reality. In 2-IMMERSE we have followed an iteratively user-centred process (interviews and focus groups, early prototypes), involving the potential users since the beginning, as reported in deliverable D3.3. Our process has led to the definition of three main tools:

1. A multi-screen scripting editor for pre-production and planning the experiences.
2. A near-live event editor for reducing the workload during live broadcasting by providing templates for certain events.
3. A live triggering tool with which the director can trigger in real-time the right events.

Figure 22 includes a visualisation of the tools and the workflow.

During year two we have given priority to the live triggering tool intended for the MotoGP scenario, which is now a working prototype integrated with the 2-IMMERSE platform. The pre-production tool is still under development, with most of the components of the frontend ready, and missing certain functionality in the backend.

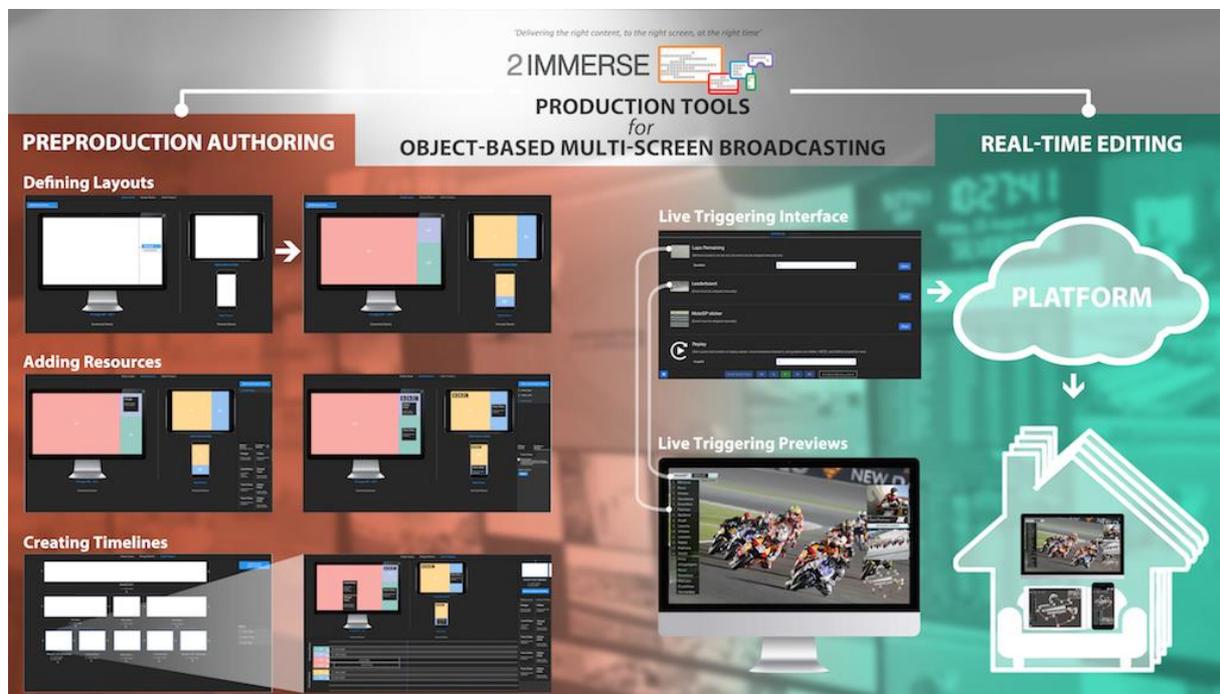


Figure 22 - The Production Workflow as envisioned by 2-IMMERSE

8.1 Architecture and Workflow

In order to implement the tools (pre-production and live triggering), we have devised the following architecture, divided into the following components (see Figure 23):

- **Authoring backend:** Acts as point of interaction between the frontend, the database and the rest of the 2-IMMERSE infrastructure. This layer is comprised of an HTTP server written in Python, enhanced with the ability to manage timeline documents and communicate with the Timeline service and receive/forward editing operations to it.
- **Pre-Production Authoring frontend:** The pre-production frontend enables the user to create interactive multi-screen experiences. It is implemented as a web application and is largely self-contained in that it only talks to the backend for storing experience data. All other business logic is implemented on the client side.
- **Live Triggering Authoring frontend:** The live triggering frontend tool is a particular part of the application which takes existing timeline documents, containing specially-annotated sections to facilitate the real-time insertion of events into a running timeline. In other words, this tool enables the user to load a document, play it and trigger predefined sequences of events at will in the running experience.

A production tool instance also includes a full Client Application stack, used for preview play during pre-production and to deliver the experience to the end users during live triggering. The following figure shows the full architecture of the production tool.



Figure 23 - Architecture of the Production Tools

8.2 Document Format

The production tool document format is the same as the timeline document XML format (see D2.2 (2) and D2.3 (3)) with some additional sections added to encode information on triggerable events, layout, and playback device parameters. For this release, with the live triggering tool fully implemented, the latter two simply store the layout.json and client.json documents almost verbatim inside the XML.

Triggerable events are standard timeline XML elements, using the standard timeline parallel and sequential composition and ref and update elements to control the media components. The main differences between a triggerable event and a normal timeline chapter are as follows:

- Triggerable events are stored in the document in such a way that the Timeline service ignores them during normal execution;
- Each event has a set of parameter elements that outlines which information has to be provided by the live triggering tool operator before the event can be triggered, and where this information needs to be stored in the event parameters (duration attributes, URLs for media items, text for labels, etc.).

It is important to note that the production tool document format is 100% compatible with the Timeline service. At any time during the production workflow, which could last for weeks or even months, the unfinished document can be previewed using the normal 2-IMMERSE platform. This allows judging the current state of production work in diverse settings.

8.3 Frontend

8.3.1 Overview

The frontend forms the main point of interaction for the user. It is implemented as a rich browser application, which invokes the feel of a native desktop application. The UI has been developed based on initial requirements by means of an iterative process with media production experts (see deliverable D3.3 (6)). A series of candidate prototypes has been created and evaluated to put together a final UI prototype, which shall be implemented to completion. For this, a series of detailed UI sketches was required, which outline every piece of functionality desired to appear in the final application.

This final application shall provide an event-based way of creating interactive media presentations. This is different from timeline-based editing in that it does not provide frame-accurate editing support. As already explained, frontend development for the second release has focused on the delivery of the live triggering tool UI to support the trial of the MotoGP service prototype.

To implement a rich web application, the frontend employs the frontend library React (35) developed by Facebook and in use in Facebook Chat and Instagram. It provides a component-based way of structuring the application. Each part of the user interface is defined in terms of a component, which can then be used and reused much like a standard HTML tag inside the application. Furthermore, these components can receive configuration parameters from parent components and manage their own internal state and potential child components. Data flow within the application is strictly unidirectional from parent to child components. This provides the assurance that each piece of data is kept consistent within the entire application and there is a single authoritative source of truth. In order to manage global application state within the UI during the editing process, the supplementary library Redux is used, which implements a simplified version of the Flux paradigm recommended by Facebook for the use together with React.

Further, in order to aid with the development of this rather complex UI application, the project employs the use of TypeScript (25). TypeScript is a superset of conventional ES6 JavaScript developed by Microsoft with the added benefit of a strong type system. Compile-time type checking erases a source of bugs introduced by JavaScript's weak dynamic typing.

React and TypeScript cannot be run by any browser natively and thus require a build toolchain to be put in place. This facility is provided by Webpack, a pluggable module bundler. It invokes the TypeScript compiler, which checks all types in the application, resolve modules and imports and then convert the React components to optimised JavaScript code and bundle them into a single file.

In order to take the most advantage of the existing 2-IMMERSE infrastructure, the frontend directly generates timeline documents, which can be interpreted and played by the Timeline service, thereby providing the most convenient way for previewing experiences during and after the creation process.

8.3.2 Pre-production Tool

The pre-production tool consists of four stages. The first one (see Figure 24) allows the user to add preview screens in various sizes and orientations. These screens can then be subdivided into regions through clicking. These subdivided regions then provide a space for DMAP Components be loaded into.

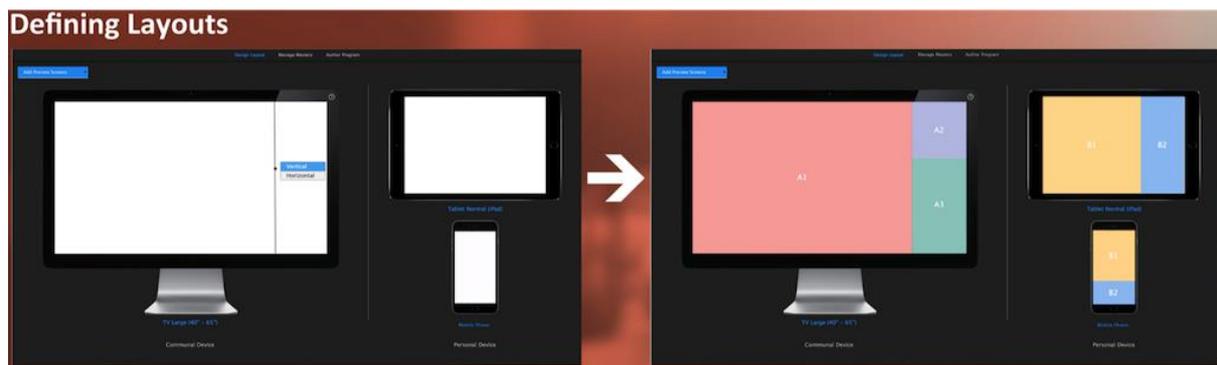


Figure 24 - Defining Layouts with the Pre-Production Tool (wireframe)

Once the user is content with their screen arrangement, the next step in the workflow (see Figure 25) is the creation and assignment of master layouts. A master layout in the context of the authoring platform is a set of DMAP Components assigned to screen regions, which are active all through the lifetime of said master layout. Here, the user can select preview screens created in the previous step and assign DMAP Components to screen regions through a drag and drop mechanism. The idea of these master layouts is very much akin to the concept of master slides in presentation software like Microsoft PowerPoint.

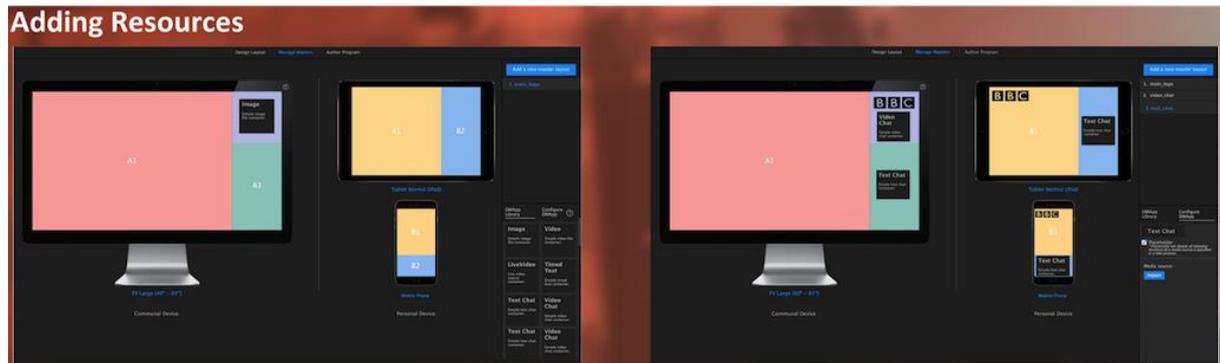


Figure 25 - Adding Resources with the Pre-Production Tool (wireframe)

The next step in the workflow (see Figure 26) is the actual authoring of the experience. In this case, this is presented as a tree of chapters. A chapter in this context is a self-contained piece of the final timeline, i.e. in a theatre performance the chapters might be pre-show, first act, intermission and second act. Moreover, chapters can have sub-chapters. This becomes important when assigning master layouts to chapters. For instance, one could create a master layout for displaying the broadcaster logo in the top right of the screen and create one root chapter with three sub-chapters constituting the actual experience. To display the broadcaster logo throughout the entire experience, one can assign the master layout with the logo to the root chapter and the application will apply the same layout to all sub-chapters.

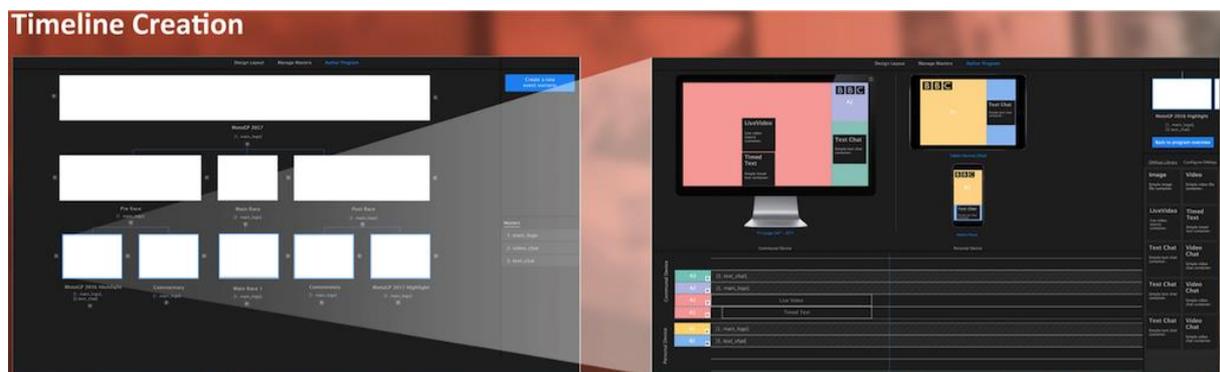


Figure 26 - Creating the Timeline with the Pre-Production Tool (wireframe).

After designing the chapter layout of the experience and assigning master layouts to chapters, the user can edit individual aspects of each chapter by simply clicking on a chapter. This puts the user into an interface which is very similar to conventional timeline-based video editors. The user can add multiple timeline tracks per screen region and add DMAPp Components to these regions by drag and drop, much like in the master layout creation step. The key difference here is that the user can fine-tune the arrangement, timing and length of components on multiple timeline tracks.

8.3.3 Live Triggering Tool

Another part of the workflow includes real-time injection of prepared pieces of timeline into a running experience (see Figure 27). For this purpose, the live triggering tool was implemented. With it, the user can start a new experience using an existing timeline document. This document should

contain a normal timeline, but in addition to that a series of events, which can be triggered while the presentation is running. These events are nothing more than timeline elements with DMAP components and timing arrangements, but they are only played back when the user chooses to. An example use case for this might be triggering the replay of a crash during a motor race. The general format of such a replay will be known beforehand, but we cannot know if and when a crash occurs and which cars or bikes will be involved. Therefore, the user can fill out the relevant information and trigger the event once a crash occurs.

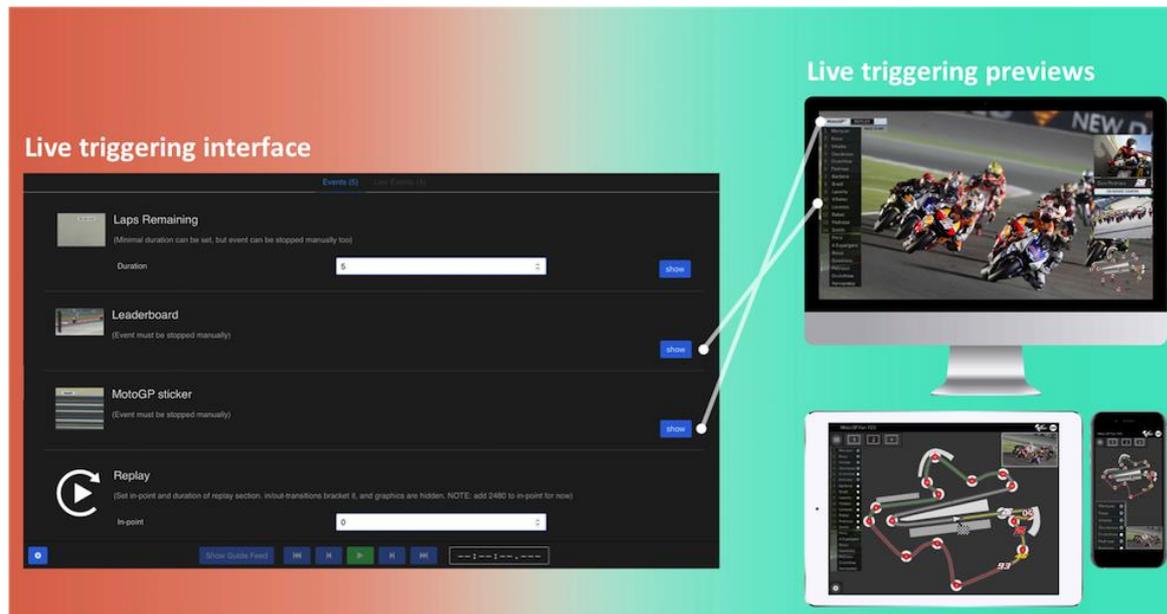


Figure 27 - Live Triggering Interface

Upon loading the timeline document, the live triggering tool scans the document for event definitions, parses them and displays them to the user as a list. Generally, an event can have any number of parameters. These parameters include text to display, numbers, list or external URLs to fetch data from. Parameters may be optional or required for the event to be triggerable. Once an event is triggered, the application relays the information to the backend server, which in turn will inform the Timeline service, which then actually inserts the event data structure at the current point in time in the running experience. After that, the triggered event will be displayed in a second list of events, those which are currently active. From this screen, the user can deactivate and/or edit active events at any time.

Once the experience has fully played out, the generated document, including all events that may have been triggered during the playout is available for download and can be replayed like a normal timeline document with the triggered events inserted into the timeline.

8.4 Backend

The authoring tool backend is a REST-like service implemented in Python, using the Flask application framework. The backend supports multiple independent documents being edited at the same time. For this release, the backend completely implements the functionality needed by the live trigger tool. The functionality needed for the preproduction tool will be implemented for the next release.

When a new session is started the backend will create the timeline, layout and client documents from the production document. A preview player is started (consisting of one or more clients and a timeline and layout service instance) and provided with the documents (thick blue arrows in the architecture diagram). The timeline service is informed it is part of a production toolchain, and it will send updates to the authoring backend informing it as the experience progresses. These updates consist of information such as “element X has started playing” and these allow the backend to maintain state information on individual components and elements (and where in the timeline the experience currently is).

The trigger tool frontend is given a list of the available events and their parameters, which it presents to the user (the producer, or trigger tool operator). When the user triggers an event to go live the front end sends this command to the backend. The backend determines where in the timeline the event needs to be inserted, finds the XML snippet that corresponds to the event and updates its internal document by inserting a copy of the event with the parameters filled in at the right location in the document. These XML document modifications are then forwarded to the timeline service, which inserts them into the running timeline, which then changes the experience as seen by the user (see Section 5.1). This implementation, together with the update mechanism outlined previously, essentially ensures that the authoring backend and the timeline server have identical copies of the document, and an identical view on the current execution state.

The backend can forward operations to multiple timeline service instances, thereby potentially changing the experience in all homes that are watching the experience concurrently. As the modifications are also made to the backend instance of the document, with all the right timing information, when that document is saved and later played back in a “normal” 2-IMMERSE playout setup, the triggered events happen at the right time in the experience.

There is a separate control flow channel from the frontend via the backend to the clients, to allow the trigger tool operator to pause and resume playback and scrub the main video. These are not intended for true live use of the trigger tool, but very useful in near-live editing, to do frame accurate insertion of events.

9 Conclusion

This document has described the second release of the 2-IMMERSE Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's MotoGP service prototype. To recap, the highlights of this second release have included:

- Extension of first release platform to include new services, and extend existing services to deliver functionality required by the MotoGP service prototype. New services include Auth and Auth-admin, Data Playback, Bandwidth Orchestration and the Editor services.
- Migration of the service platform from a private cloud environment to Amazon Web Services (AWS) and then subsequently migration from the Mantl container platform to Rancher.
- Development of Linux-based HbbTV2.0 Emulator firmware to run on Intel NUC devices to support service prototypes. Key features include Onboarding, Integrated Wi-Fi router/access point, HbbTV2.0 and the Web Kiosk.
- Client API developments and improvements.
- Authoring and development of the MotoGP service prototype DApp, and its constituent elements: timeline, layout, HTML and CSS documents, several DApp components with a focus on data-driven animated graphics, and media asset preparation.
- Production tool development with a focus on the real-time triggering required for the MotoGP scenario

As the second instance of a working platform for the delivery of an interactive, object-based multi-screen experience, it forms the foundation for the remaining service prototypes which will be developed and taken to trial in the final year of the project. The additional services and service capabilities, DApp components and the HbbTV emulator will all be reused (and improved) in the remaining Football at Home and Theatre in Schools service prototypes.

10 References

1. **2Immerse**. *D2.1 System Architecture*. 2016.
2. —. *D2.2 Platform-Component Interface Specifications*. 2016.
3. —. *D2.3 Distributed Media Application Platform and Multi-Screen Experience Components: Description of First Release*. 2016.
4. —. *D4.4 Prototype Service Descriptions – Second Update*. 2018.
5. **ISO/IEC 23009-5:2016**: "Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)". ISO/IEC. 2016. 23009-5:2016.
6. **2Immerse**. *D3.3 User Interaction Design: the development of generic components & features to inform MotoGP Service Trials, Production Tools, and OnBoarding*. 2017.
7. —. YouTube 2Immerse Channel. *YouTube*. [Online] [Cited: 10 January 2018.] <https://www.youtube.com/channel/UCpGa5NU1Bbj8Nkz0vZi7lwA>.
8. **mantl.io**. *mantl.io*. [Online] [Cited: 10 January 2018.] <http://mantl.io/>.
9. **rancher.com**. *rancher.com*. [Online] [Cited: 10 January 2018.] <http://rancher.com/>.
10. **consul.io**. *consul.io*. [Online] [Cited: 10 January 2018.] <https://www.consul.io/>.
11. **influxdata.com**. *influxdata.com*. [Online] [Cited: 10 January 2018.] <https://www.influxdata.com/>.
12. **mongodb.com**. *mongodb.com*. [Online] [Cited: 10 January 2018.] <https://www.mongodb.com/>.
13. **Registrator**. *Registrator*. [Online] [Cited: 10 January 2018.] <http://gliderlabs.github.io/registrator/latest/>.
14. **logspout**. *github.com*. [Online] [Cited: 10 January 2018.] <https://github.com/gliderlabs/logspout>.
15. **tyk.io**. *tyk.io*. [Online] [Cited: 10 January 2018.] <https://tyk.io/>.
16. **traefik.io**. *traefik.io*. [Online] [Cited: 10 January 2018.] <https://traefik.io/>.
17. **prometheus.io**. *prometheus.io*. [Online] [Cited: 10 January 2018.] <https://prometheus.io/>.
18. **GitLab Runner**. *gitlab.com*. [Online] [Cited: 10 January 2018.] <https://docs.gitlab.com/runner/>.
19. **kibana**. *elastic.co*. [Online] [Cited: 10 January 2018.] <https://www.elastic.co/products/kibana>.
20. **letsencrypt.org**. *letsencrypt.org*. [Online] [Cited: 10 January 2018.] <https://letsencrypt.org/>.
21. **Logstash**. *elastic.co*. [Online] [Cited: 10 January 2018.] <https://www.elastic.co/products/logstash>.
22. **mattermost.com**. *mattermost.com*. [Online] [Cited: 10 January 2018.] <https://about.mattermost.com/>.
23. **sensuapp.org**. *Sensu*. [Online] [Cited: 10 January 2018.] <https://sensuapp.org/>.
24. **uchiwa.io**. *uchiwa.io*. [Online] [Cited: 10 January 2018.] <https://uchiwa.io/>.
25. **typescriptlang.org**. *typescriptlang.org*. [Online] [Cited: 10 January 2018.] <http://www.typescriptlang.org/>.
26. **Layout Service Documentation**. *2Immerse Origin*. [Online] [Cited: 10 January 2018.] <https://origin.platform.2immerse.eu/docs/layout-service/>.

27. OAuth 2.0. *oauth.net*. [Online] [Cited: 10 January 2018.] <https://oauth.net/2/>.
28. 2-IMMERSE Auth Service API documentation. *2Immerse origin*. [Online] 10 January 2018. <https://origin.platform.2immerse.eu/docs/auth-service/latest/>.
29. Quadratic knapsack problem. *wikipedia.org*. [Online] [Cited: 10 January 2018.] https://en.wikipedia.org/wiki/Quadratic_knapsack_problem.
30. *A Dynamic Programming Heuristic for the Quadratic Knapsack Problem*. Franklin Djeumou Fomeni, Adam N. Letchford. s.l. : INFORMS, 22 July 2013, INFORMS Journal on Computing. <https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2013.0555>.
31. *Approximation of the Quadratic Knapsack Problem*. Ulrich Pferschy, Joachim Schauer. s.l. : INFORMS, 5 April 2016, INFORMS Journal on Computing. <https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2015.0678>.
32. —. Taylor, Richard. 1, January 2016, Operations Research Letters, Vol. 44. <http://dblp.uni-trier.de/pers/hd/t/Taylor:Richard>.
33. *Exact Solution of the Quadratic Knapsack Problem*. *Inform Journal on Computing*. Caprara, Alberto & Pisinger, David & Toth, Paolo. s.l. : INFORMS, October 1998, INFORMS Journal on Computing.
34. David Pisinger's optimization codes . *diku.dk*. [Online] [Cited: 10 January 2018.] <http://www.diku.dk/~pisinger/codes.html>.
35. React. *reactjs.org*. [Online] [Cited: 10 January 2018.] <https://reactjs.org/>.
36. *bbc/pydvbcss*. *github.com*. [Online] 10 January 2018. <https://github.com/bbc/pydvbcss>.
37. Apple's secret "wispr" request. *blog.erratasec.com*. [Online] 10 January 2018. <http://blog.erratasec.com/2010/09/apples-secret-wispr-request.html#.WUbaXvryuAw>.
38. Captive portal popups: the definitive guide [closed]. *serverfault.com*. [Online] [Cited: 10 January 2018.] <https://serverfault.com/questions/679393/captive-portal-popups-the-definitive-guide>.
39. Quick and dirty captive portal with dnsmasq. *reddit.com*. [Online] [Cited: 10 January 2018.] https://www.reddit.com/r/darknetplan/comments/ou7jj/quick_and_dirty_captive_portal_with_dnsmasq/.
40. Chromium Beta branch. *launchpad.net*. [Online] [Cited: 10 January 2018.] <https://launchpad.net/~saiarcot895/+archive/ubuntu/chromium-beta>.
41. List of Chromium Command Line Switches. *Peter Beverloo*. [Online] [Cited: 10 January 2018.] <https://peter.sh/experiments/chromium-command-line-switches/>.
42. Chrome remote debugging doesn't work with IP. *stackoverflow.com*. [Online] [Cited: 10 January 2018.] <https://stackoverflow.com/questions/6827310/chrome-remote-debugging-doesnt-work-with-ip>.
43. HDMI 2.0 vs 1.4: What's the difference? Read more at <http://www.trustedreviews.com/opinion/hdmi-2-0-vs-1-4-2913356#MEITQpycTAIU5Eut.99>. *trustedreviews.com*. [Online] [Cited: 10 January 2018.] <http://www.trustedreviews.com/opinion/hdmi-2-0-vs-1-4-2913356#UeixwjfXgh3zvDbQ.99>.
44. Intel NUC. *archlinux.org*. [Online] [Cited: 10 January 2018.] https://wiki.archlinux.org/index.php/Intel_NUC.
45. Universal Windows Platform documentation. *microsoft.com*. [Online] [Cited: 11 January 2018.] <https://docs.microsoft.com/en-us/windows/uwp/>.

46. Microsoft HoloLens. *microsoft.com*. [Online] [Cited: 11 January 2018.]
<https://www.microsoft.com/en-gb/hololens>.

47. ExoPlayer. *google.github.io*. [Online] [Cited: 11 January 2018.]
<http://google.github.io/ExoPlayer/>.

48. 40 MHz Channels. *Metageek*. [Online] [Cited: 10 January 2018.]
<https://support.metageek.com/hc/en-us/articles/204490510-40-MHz-Channels>.

Annex A Synchronisation Service (Inter-Home Sync)

A.1 Limitations of current synchronisation approaches in 2-IMMERSE

A.1.1 Intra-Home Synchronisation with DVB-CSS

DVB-CSS (HbbTV 2.0's media-sync mechanism) has been designed primarily for the purpose of media synchronisation between a master device such as a TV and companion devices, all residing on the same local-area network. The DVB-CSS media sync model relies on assumptions such as a strict hierarchical organisation of devices (master/slave) and the establishment of a common WallClock via UDP-based time-synchronisation protocols. These do not lend well to a more distributed environment where hierarchical master-slave device organisation and end-to-end UDP support cannot be assumed.

A.1.2 Inter-Home Synchronisation via DMAP's timeline correlation to a shared WallClock

An experience timeline (or DMAP timeline) is created as a result of the main DMAP component starting media playback on the master device. This timeline is used by the Timeline Service to schedule the loading/unloading of DMAP components at specified times during the experience. For accurate DMAP scheduling, a common timeframe is established in the form of a shared WallClock against which times on the DMAP timeline are correlated. The shared WallClock is provided by the WallClock service which provides a time-synchronisation to enable devices to maintain a local WallClock that is synchronised to a global WallClock. Inter-destination (or inter-home) synchronisation is achieved by the master device sharing a Correlation Timestamp with other devices via the Shared State Service

The Correlation Timestamp tuple consists of the following:

```
{wallclock-time, dmappc-time, speed}
```

and is sent by the master Client API instance upon changes in the correlation (e.g. DMAPComponent is paused).

There are a number of disadvantages with this approach:

- 1) The model for inter-context synchronisation is not easily reusable, requires developer participation and fits specific use cases only.
- 2) Synchronisation responsibility is put on the Client API; this Client API's functionality is getting increasingly complex and the current approach does not achieve a good separation of concerns.
- 3) The current synchronisation model assumes that WallClock Service and Timeline Service instances share an NTP-Synchronised common clock. NTP was however not designed to run in microservice environments; it cannot be assumed that containerised services have access to an NTP_synchronised system clock.
- 4) The current model relies on clients to initiate an experience timeline; this model suits distributed experiences that use on-demand content exclusively. It may not readily be applicable for synchronisation to live content.

A.2 Proposed Synchronisation Model

We propose a new synchronisation model where the synchronisation functionality is hoisted to the cloud and timelines become *first-class entities* i.e. timelines can be produced or consumed by different devices or services.

A.2.1 Multiple Timelines and Timeline Channels

The Synchronisation service allows devices/services to register timelines; these timelines can be *media-based* (e.g. a video playback on a particular device), or *artificial* (e.g. an experience timeline created by a service orchestrating the experience). In addition to timeline registration and discovery, the model introduces the concept of **timeline channels** where devices can stream updates about their timeline progress. Devices and services can thus discover timelines dynamically and subscribe to a timeline channel to obtain a local estimate of that timeline. The synchronisation mechanism is still underpinned by a shared WallClock (provided by the WallClock Service). Timeline channel stream updates are in the form of Presentation Timestamps (in DVB-CSS terminology):

```
{timeline-time, wallclock-time, speed}
```

A.2.2 Resilience to disconnections via Timeline Shadows

In distributed media experiences, network disconnections or application failure can result in a device's timeline to become unavailable. If this device provides the master timeline that driving the whole experience, the results can be potentially disastrous. In the first and second 2-IMMERSE platform releases, the experiences are vulnerable to poor network performance and device/application failures. An experience will be broken and become unresponsive to timeline changes if the DMAP timeline originated by a master DMAP Component becomes unavailable when the DMAP Component's host device dies.

The Synchronisation service borrows the notion of Device Shadows from the IoT domain to provide **Timeline Shadows**.

*A **timeline shadow** is the local estimate of a remote timeline's state usually in the form of a software clock object.*

A timeline shadow is kept up-to-date by periodically refreshing its state when the source timeline publishes an update via its channel.

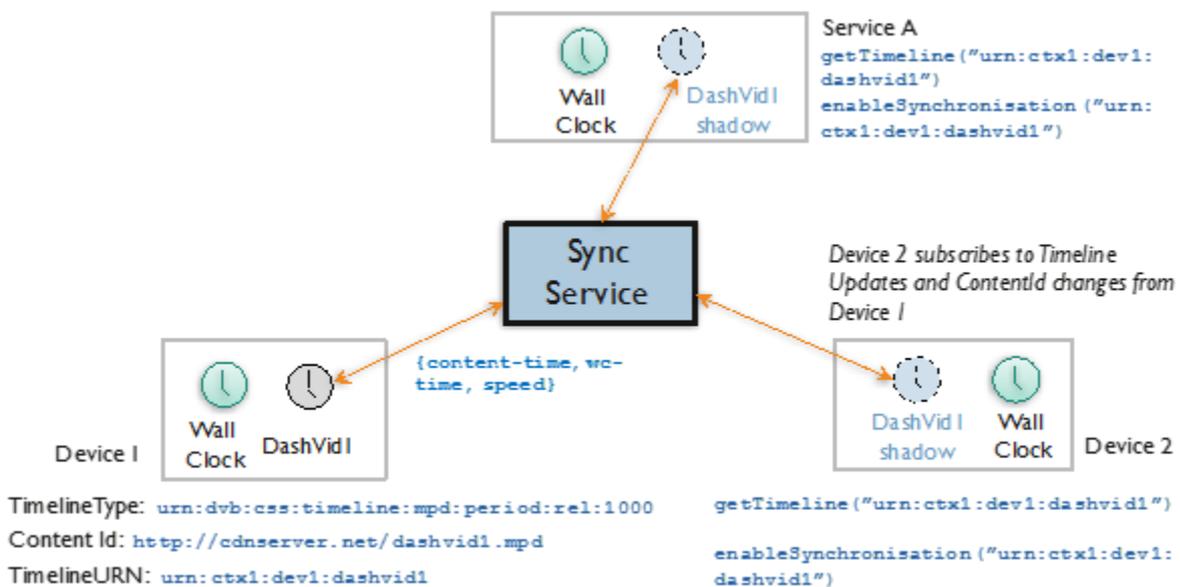


Figure 28 - Timeline Shadow - estimate of a remote Timeline as a clock object

The new Synchronisation service therefore allows devices to discover a relevant timeline and request for its timeline shadow to be made available. Figure 28 above illustrates a scenario where a device

(Device 1) publishes a timeline and that timeline is discovered by Service A and Device 2, and manifested as shadows locally (as software clock objects).

Timeline shadows are estimates of a timeline's state and each state update contains an error-value and an error growth-rate originating from the process the state update (Presentation Timestamp) was obtained. A timeline shadow's state contains the Presentation Timestamp, error-value and error growth rate of the last update.

If a timeline shadow ceases to be updated (as a result of the source timeline disappearing - e.g. host device dies), the timeline shadow is said to remain true until the error proportion of the timeline's reading exceeds a specified threshold.

One can specify different error growth rates to adjust the rate at which timeline shadows are rendered stale when the source timeline becomes unavailable.

A.3 Architectural Overview

The Synchronisation service uses the concept of **sessions** to differentiate between different synchronisation groups. This is analogous to the notion of sessions shared by experience establishment and control services in the 2-IMMERSE platform. This allows the service to dedicate resources to individual sessions such as sync-controllers, session- and timeline- channels. Sessions also act as a scoping mechanism for timelines and timeline channels. Devices can only discover timelines and access channels for timelines registered in their session. The end of a session (when the last device leaves) allows the resources to be recovered automatically by the service.

The Synchronisation service consists of the following components on the server-side:

- 1) Session Controller,
- 2) WallClock Service,
- 3) Sync Controller,
- 4) a distributed key-value store,
- 5) Sync Service Channel,
- 6) One or more Timeline Channels.

On the client-side, the Synchronisation service provides a library API called Cloud Synchroniser. The architectural organisation of the components is illustrated in the diagram below. The role of each component is further explained in the following sub-sections.

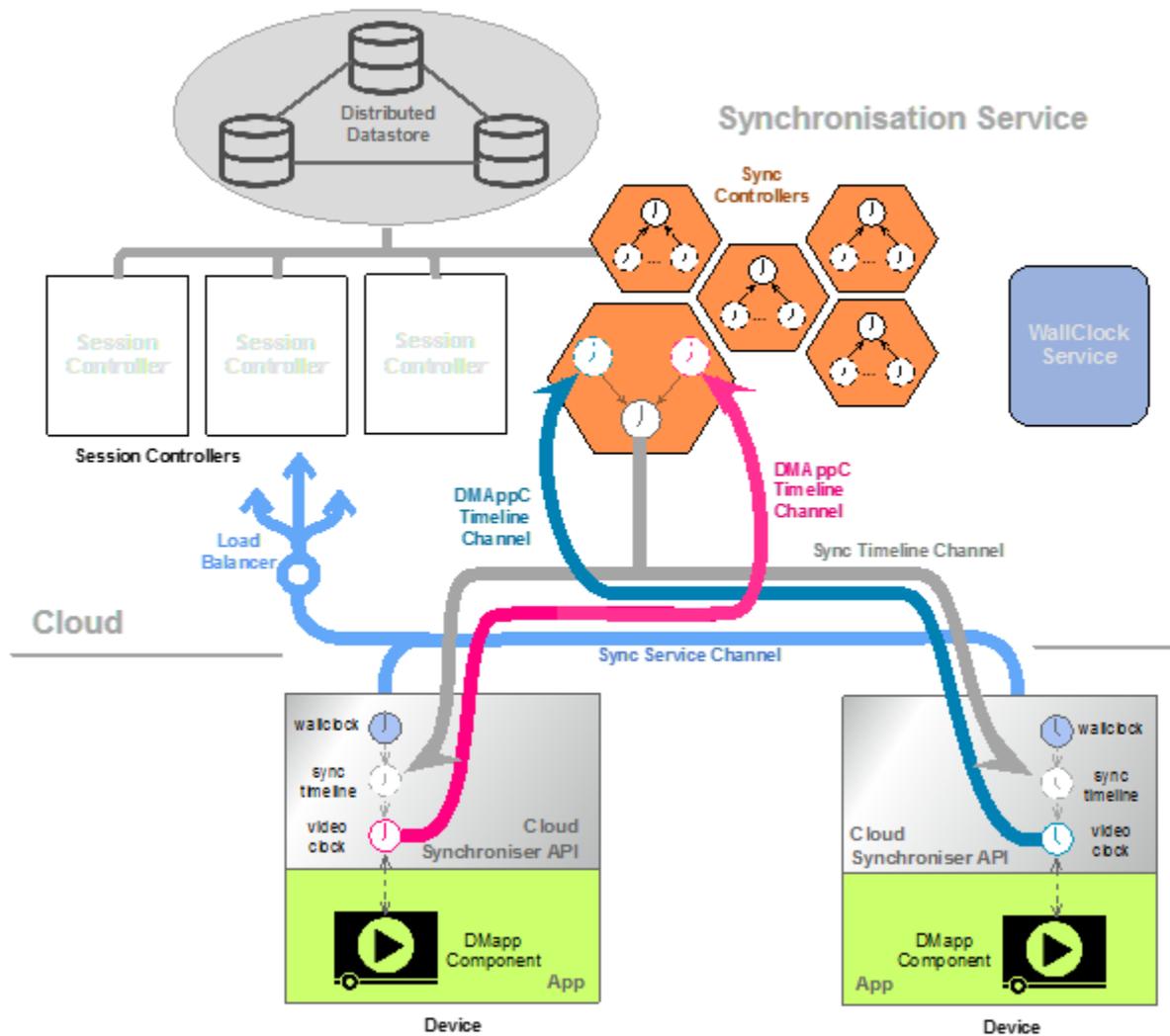


Figure 29 - Synchronisation Service Architecture

A.3.1 Cloud Synchroniser API

This is a client-side library that provides access to the Synchronisation service functionality to application instances in an immersive distributed media experience. The API provides an operation that allows devices to register with the Synchronisation service and specify their session membership. Other operations in the API enable devices to register timelines, discover other timelines, and request a synchronised copy of a timeline to be manifested locally as a clock object.

A.3.2 Sync Service Channel

This is a bi-directional communication channel that is used for on-boarding i.e. for devices wishing to participate in a synchronisation session to send the join-request.

A.3.3 Session Channel

This is a bi-directional communication channel for devices to send requests to (un)register timelines, query timelines, subscribe to a remote timeline and monitor content changes on devices.

A.3.4 Session Controller

A Session Controller handles requests coming from Cloud Synchroniser API instances for device registrations, timeline registrations and timeline queries. The component handles requests for different sessions concurrently. Each session's state is persisted in the distributed data store. Multiple session controllers can run concurrently on the same or different host machine.

A.3.5 Sync Controllers

A sync controller is requested by a session controller starting a new session to take on the synchronisation responsibilities for that session. It runs a synchronisation algorithm that takes one or more client timelines to produce a session-wide timeline for synchronisation. This timeline is called the Synchronisation Timeline and should be used by all Cloud Synchroniser API instances to synchronise the local playback of media/DMAApp components. The synchronisation algorithm adopts a specified strategy in terms of synchronisation bias and timeline control. For example, in an experience where all devices are slaved to the video timeline of a master device, the Sync Controller for this session will produce a synchronisation timeline that is the exact copy of the master device's video timeline. To improve robustness, a Sync Controller in master-slave mode may switch the provider of the synchronisation timeline to another device if it is notified about the departure of the master device. Furthermore, once synchronisation is achieved, a sync controller can enable the synchronisation timeline to be controlled (pause, seek) by one or more parties, as specified by the Session Controller.

A.3.6 Timeline Channels

Timeline channels are multicast channels where progress information for each timeline is published. Timeline progress is signalled in the form of Presentation Timestamps. When client devices query for a timeline, the Session Controller processing the request will respond by specifying the particular channel where the timeline's stream of updates are being published.

A.3.7 WallClock Service

As described earlier, the WallClock Service is used to establish a common time reference for all devices and services in the platform.

A.4 Implementation

A first version of the Synchronisation service (and its client Cloud-Synchroniser API) has been implemented by BBC and IRT. This version supports the core features such as device registration, timeline registration, timeline querying, timeline channels, timeline shadows, sync controllers and session controllers. The back-end services have been developed using a microservice architecture pattern. The team is currently evaluating the synchronisation performance of the service and preparing for integration and deployment in the 2-IMMERSE platform. Future iterations of the service to add advanced features such as new synchronisation strategies are planned after successful integration.

Annex B HbbTV2.0 Emulator Components

This Annex provides details of each of the high-level components of the 2-IMMERSE TV emulator firmware.

B.1 Operating System

The firmware is based on the latest Ubuntu Server 17.04 .ISO distribution. The software is built and installed as a Debian package. The firmware is targeted at amd64 devices but is general-purpose enough to run on i386 and ARM devices too. In fact, it was initially developed for an Odroid single board computer which uses Amlogic's ARM-based S905 SoC. The firmware also runs in a virtual machine using VirtualBox, emulating some Wi-Fi functionality. For service trials, we use a 7th generation Intel NUC because of the increased performance and stability it offers. For a detailed set of hardware and firmware requirements, see D3.3, Section 2.4. The firmware is built and distributed as a hybrid .ISO file. It is based on an Ubuntu Server .ISO but modified to include some additional Debian packages. The .ISO boots from a USB drive and presents the user with a menu offering them the choice of either installing the firmware on the target device or installing it into a VirtualBox virtual machine. The latter option also installs the VirtualBox guest additions and an alternative set of scripts to emulate a Wi-Fi network inside the virtual machine.

B.2 HbbTV2.0 Emulator Services

The 2-IMMERSE firmware emulates a subset of the HbbTV2.0 features. It runs a DIAL service (Discovery and Launch) and an 'app2app' server providing a local communication mechanism between devices over web sockets connections.

The firmware also runs a proxy server to allow the Chromium web browser to communicate with a python library that implements DVB protocols for companion synchronisation (pydvbcss). Web browsers don't expose the ability to communicate via UDP, so the DVB protocols can't be implemented directly in the browser. This is because the protocols are based on UDP datagrams which aren't available in the browser. For more information about the pydvbcss functionality, see (36)

B.3 On-boarding System

The firmware implements user journeys for network configuration, sign-in, device pairing, experience discovery and experience launch, but designing a simple user experience is hugely challenging when there are multiple devices. Complex configuration and setup procedures can be a barrier to the adoption of the technology, therefore the success of multi-screen experiences depends on how seamless we can make the setup and launch process.

The on-boarding flow runs in parallel across a number of devices, with some devices assisting others with the setup. To run a Distributed Media Application (DMap) across a group of devices, each member of the group must acquire:

1. An Internet connection
2. A valid access token
3. A unique device identifier
4. A common group identifier (layout context Id)

The firmware works with other devices to negotiate the value of these parameters, using the touchscreen of tablets and phones for remote keyboard input.

B.4 Admin Portal

On boot, the firmware starts a service that displays a locally-served webpage and checks for an Internet connection. If an Internet connection is detected, it proceeds to load a welcome page from the CDN via a URL that can be configured and parameterised using the firmware’s admin panel. This allows developers to repoint the firmware at production, test and edge server environments. It also allows developers to specify an arbitrary URL for the firmware to load.

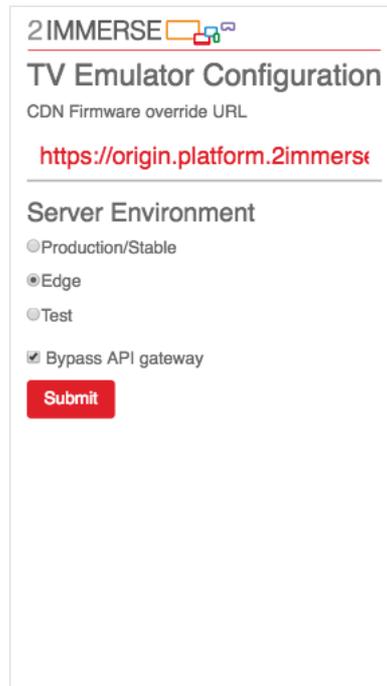


Figure 30 - NUC admin page (<http://<tv-emulator-ip-addr>:3000/admin>)

The entire user interface is hosted on the CDN, with the exception of the initial ‘checking for Internet connection’ and ‘let’s get connected!’ pages. This allows for rapid iteration of the user experience without rebuilding firmware. Each page displayed to the user provides instructions for configuring and launching experiences.

B.5 On-boarding Steps

There are three basic on-boarding steps to successfully launching an experience:

- Plugging in the equipment
- Connecting all devices to the Internet
- Launching/joining an experience

B.5.1 Plugging in the equipment

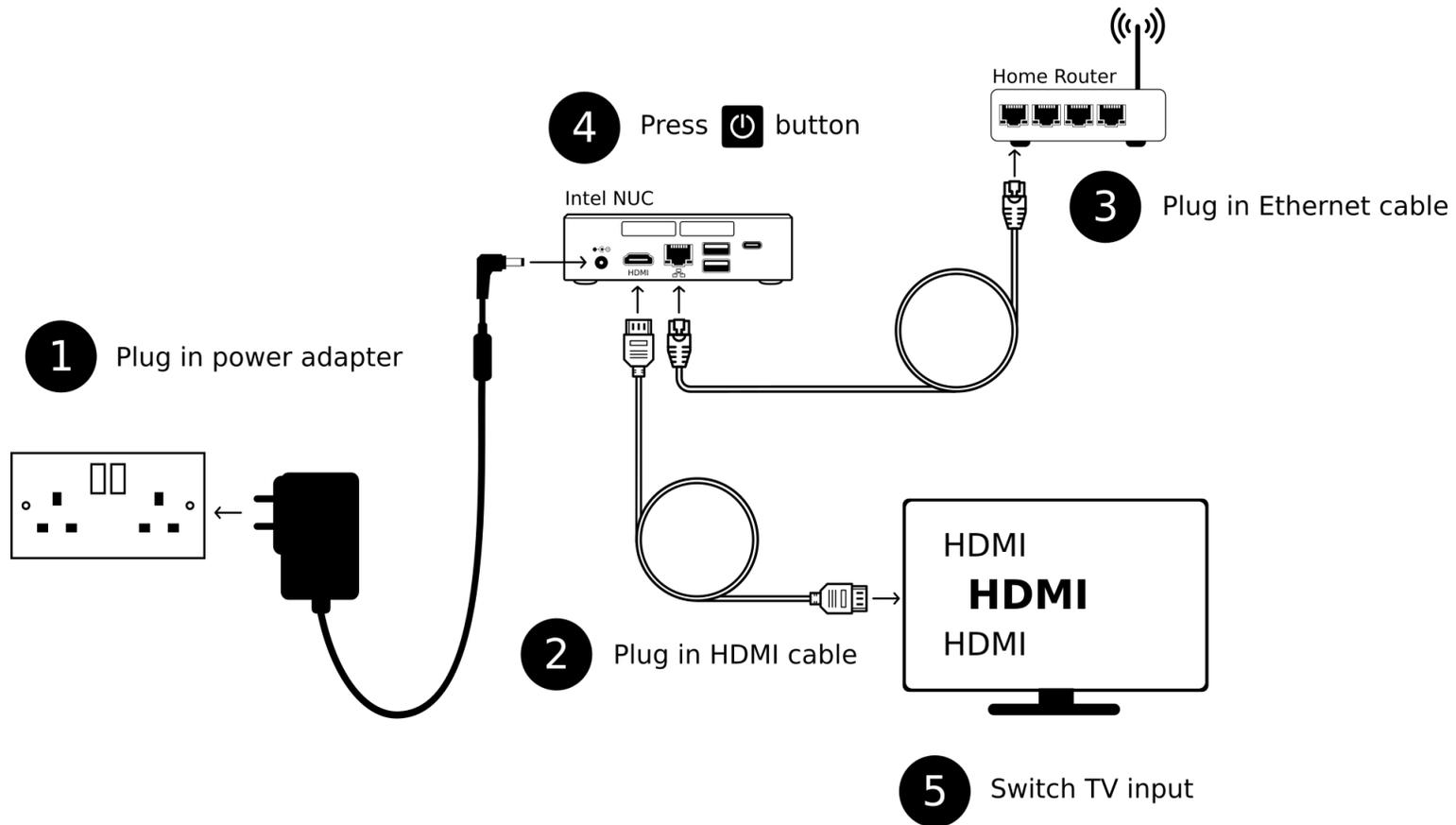


Figure 31 - Instructions for connecting the TV Emulator device

B.5.2 Connecting all devices to the Internet

The TV emulator device will display the following message on the TV until an Internet connection is detected.

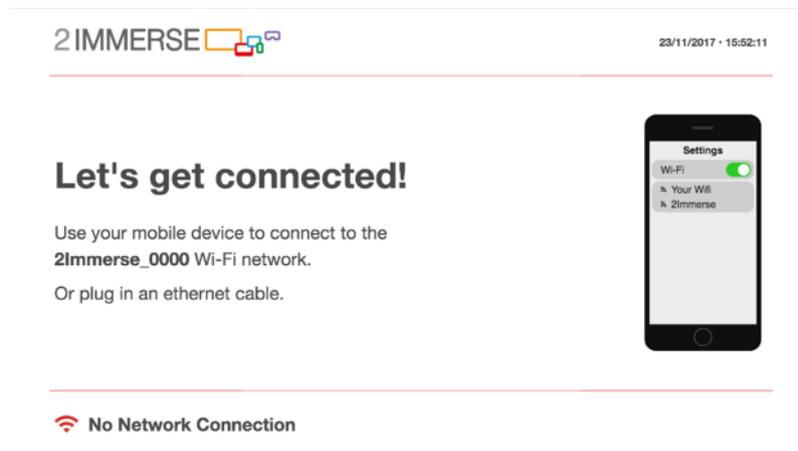


Figure 32 - The TV will display this screen when it doesn't have an internet connection

There are two ways to connect the TV emulator device to the Internet:

1. Via a wired Ethernet cable
2. By configuring a Wi-Fi connection

The default network/route of the wired interface has a lower routing metric than that of the Wi-Fi interface so it's chosen in preference by the kernel when routing requests to the Internet. This is because the wired connection is more likely to deliver a higher quality of service than a Wi-Fi connection. Using an Ethernet cable is also the easiest way to setup the device because there are fewer steps for the user to perform.

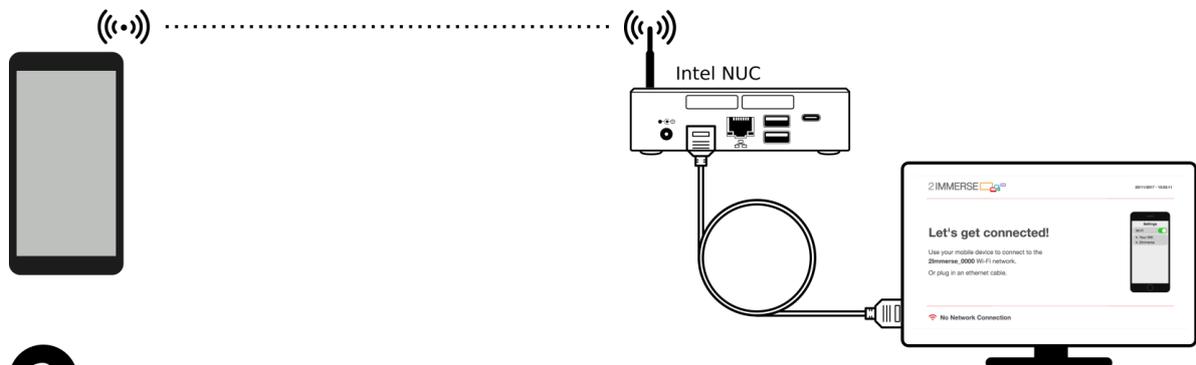
When using the Ethernet cable to connect to the venue's router, the TV emulator device will create its own Wi-Fi hotspot offering phones and tablets a wireless Internet connection that's potentially¹ better than the venue's existing hotspot.

If it's not possible to connect an Ethernet cable, the device can be configured to connect to the venue's Wi-Fi. In this mode, the device's own Wi-Fi hotspot doesn't offer companion devices a quality of service advantage. So instead, it is used to serve a captive portal page that presents users with a choice of locally-discovered venue Wi-Fi hotspots to use. A companion device can use this page to select which of the venue's Wi-Fi hotspots the TV emulator device should connect to.

This process involves temporarily connecting a companion device to the TV emulator device's Wi-Fi hotspot as outlined in the diagram below. The user can repeat the Wi-Fi configuration steps for the TV emulator device at any time as long as there isn't an Ethernet cable plugged in. This might be required if the user changes the venue's Wi-Fi password.

¹ The Intel Wireless-AC 8265 adapter used by the Intel NUC doesn't support 802.11ac in access point mode, however 802.11n is supported but only at 2.4GHz frequencies. 802.11n supports two channel widths; 20Mhz and 40MHz. Channel bonding is used to achieve 40Mhz widths but bonding is not recommended at 2.4GHz frequencies. This limits the maximum bandwidth between companion devices and the NUC to be about 130Mb/s. For more information about channel widths, see (31).

1 To connect the NUC to the internet, first connect the phone to '2Immerse_****' WiFi hotspot (password (if prompted): *twoimmerse*)

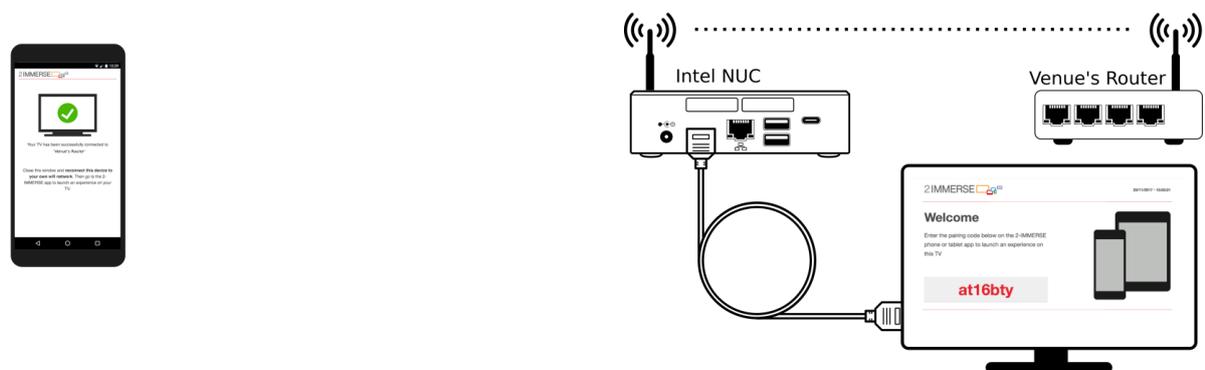


2 Select venue's WiFi from list



3 Enter venue's WiFi password

4 NUC will connect to venue's WiFi



5 Now the NUC is ready, connect the phone back to venue's Wifi

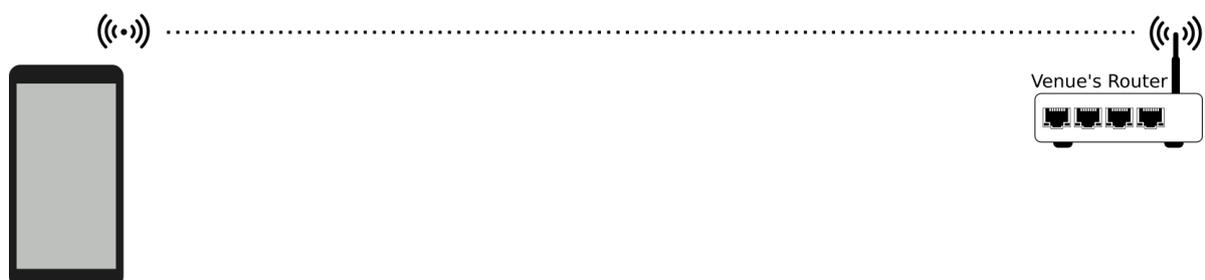


Figure 33 - Captive portal based Wi-Fi setup procedure

B.5.3 Launching/joining an experience

Once all devices are connected to the internet, an experience can be launched using the companion application on a phone or other device. For trial participants, the app is pre-installed on the phone's home screen. When run, it prompts the user to sign into a 2-IMMERSE user account by entering credentials given to them by the trial recruiter.

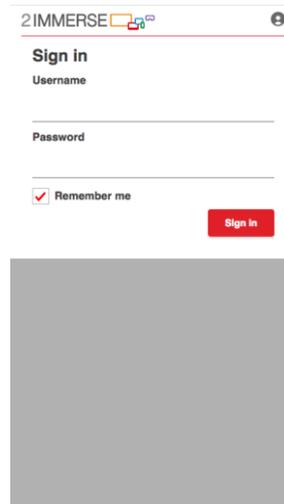


Figure 34 - Companion app sign-in screen

After successful sign in, the app will scan for running experiences. Initially it won't find any. The users are presented with in-app instructions of how to launch a new experience. This involves clicking the '+' button:

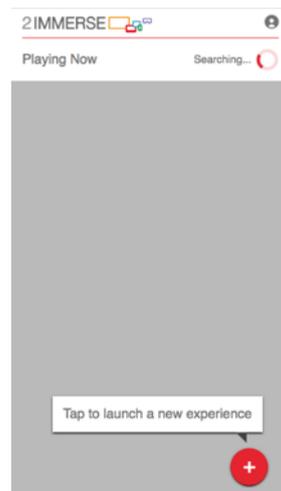


Figure 35 - Companion app showing no currently detected experiences

The user is then asked to enter the code displayed on the TV screen. This tells the app which TV to launch the experience on.

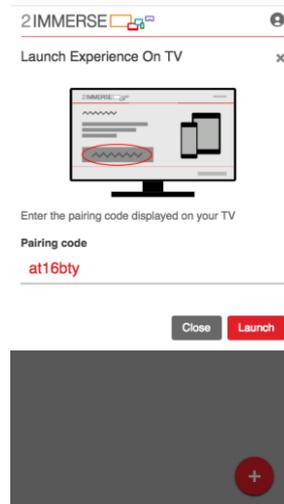


Figure 36 - Entering the TV pairing code on the companion app

Once the experience is running on the TV, the user can add the phone to the experience by choosing the 'Join' button:

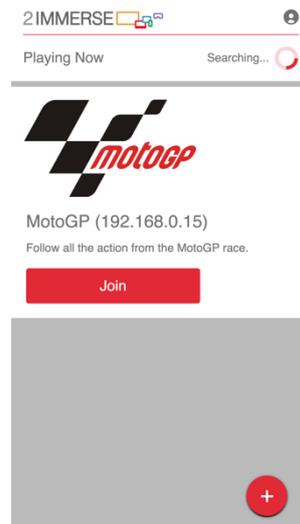


Figure 37 - Companion app showing a discovered experience that can be joined

Other devices running the companion app can also join the experience by signing into the 2-IMMERSE app using the same or different credentials and choosing the 'Join' button.

The low-level network interactions between the TV emulator, companion device and authentication service to achieve pairing are outlined by the following sequence diagram:

Pairing a keyboardless TV emulator device with a user account

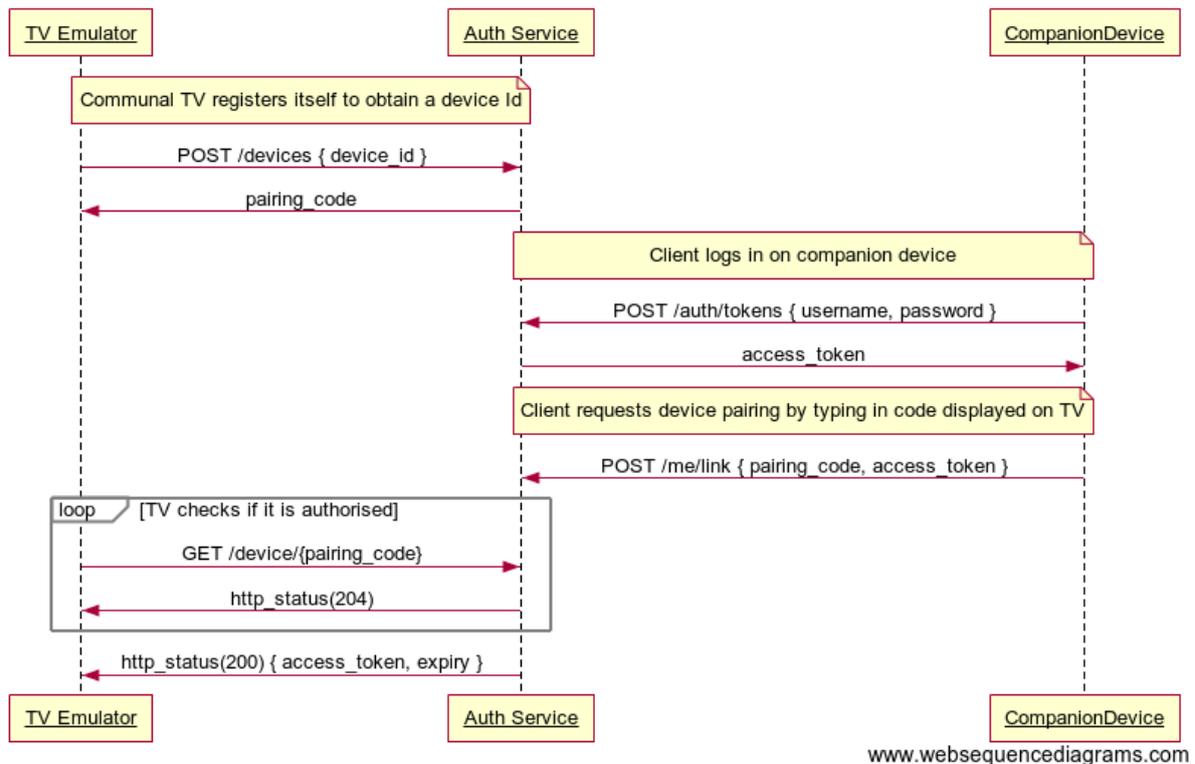


Figure 38 - Sequence diagram showing network interactions for TV emulator device pairing

B.6 Network connectivity management layer

The firmware is responsible for monitoring changes in network connectivity resulting from:

- The Ethernet cable being inserted or removed
- Users configuring/re-configuring Wi-Fi credentials
- The venue's router password being changed or becoming inaccessible

Ethernet cable pulls are detected using a system daemon called 'ifplugd'. It uses the presence (or absence) of a carrier signal to determine whether there is currently a wired connection to the venue's router. When changes in connectivity are detected, the daemon will bring the corresponding Ethernet interface up or down which in turn executes custom network management hooks. The network connection signalling scheme is summarised in the diagram below:

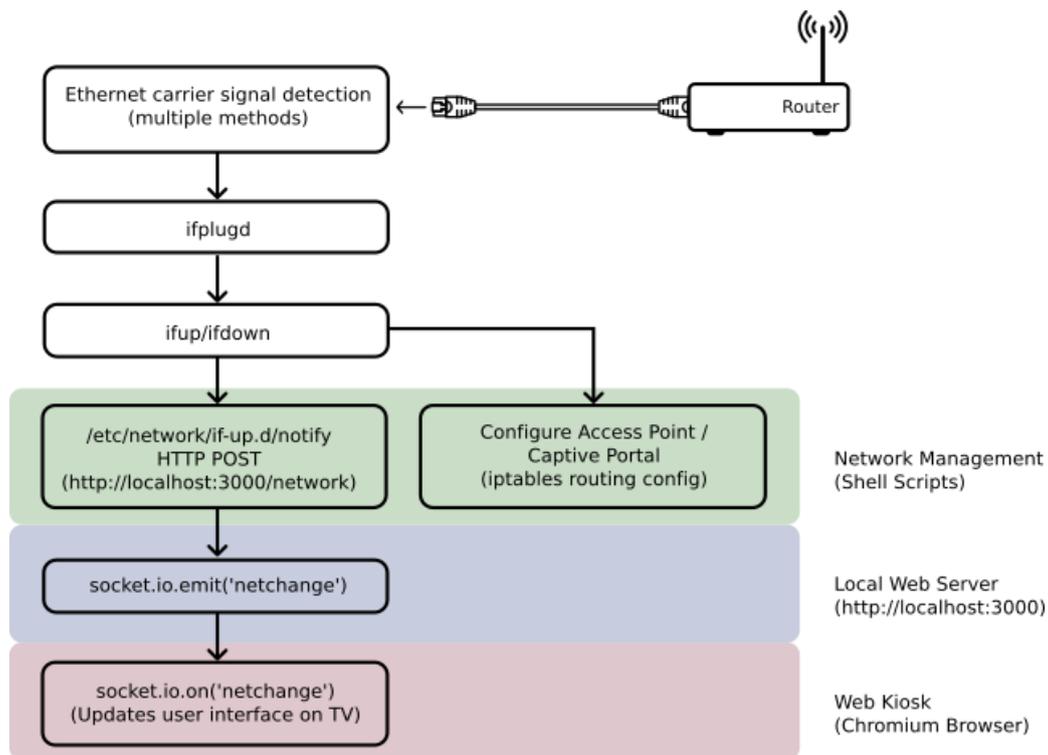


Figure 39 - Network connection signalling in the 2-IMMERSE firmware

The hooks are simple shell scripts that change the behaviour of the firmware based on whether there is a wired or wireless connection available. There are two scripts that are executed, represented in the diagram by the two green boxes.

The first script is used to notify the user of the network connection status in the user interface. It makes an HTTP POST request to the local web server running on the TV emulator device to inform it of the up/down phase of each network interface. The web server communicates this information to the web kiosk (Chromium browser) over a web socket connection, resulting in a status notification to the user.

The second script is used to switch the behaviour of the TV emulator's access point from a general-purpose router to a captive portal and vice versa.

B.7 Integrated Wi-Fi router/gateway and access point

Switching between router and captive portal modes of operation is achieved by changing the network routing rules (using 'iptables'). When operating as a router, all incoming packets from the network interface associated with the access point are redirected to the wired interface and DNS requests are serviced by the operating system's DNS resolver on port 53. When configured as a captive portal, all incoming TCP requests on port 80 are routed to the local web server on port 3000 which serves a captive portal page and all DNS requests on port 53 are redirected to a 'dnsmasq-ap' service running locally on port 5353. The 'dnsmasq-ap' service resolves all domain names to the IP address/port of the local web server.

The diagram below shows the wired interface's if-*.d scripts switching between router and captive portal configurations. It also shows the three network adapters configured on the TV emulator device; wlpap0 and wlp58s0 share the same physical Wi-Fi hardware.

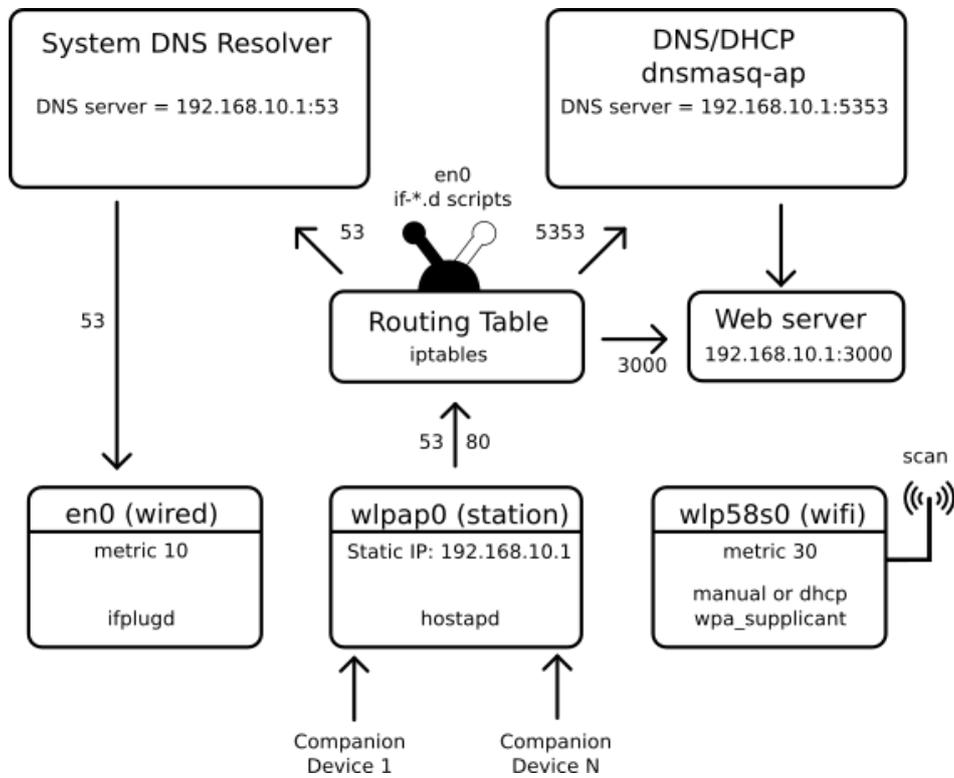


Figure 40 - Wired interface if-*.d scripts switch between router or captive portal configuration

The Wi-Fi interface wlp58s0 can be in one of two states. It can be configured with valid Wi-Fi credentials or un-configured. When un-configured, the interface must still be brought up by setting its configuration to 'manual' in order to be able to scan for networks. When connecting to a venue's Wi-Fi, wlp58s0 uses the 'dhcp' setting to automatically lease an IP address.

Wi-Fi credentials are generated using wpa_passphrase and managed by wpa_supplicant. The Wi-Fi interface is set back to 'manual' if the submitted credentials are incorrect. The firmware stores the previously configured WPA2 credentials in a configuration file which persists between reboots.

The firmware creates a new virtual interface for the access point (wlpap0) and the 'hostapd' daemon configures the Wi-Fi driver to make the interface behave as an access point. The firmware also assigns it a static IP address because the router needs to have an IP address on the same subnet as IP addresses allocated by the router's DHCP server. To route packets between wlpap0 and the wired Ethernet interface it's necessary to enable packet forwarding in the kernel.

The firmware derives a unique-ish ESSID name for its Wi-Fi hotspot based on the last four digits of the MAC address of the wired network interface (en0).

B.8 Captive Portal

The captive portal provides a way to configure the Wi-Fi connection of the TV emulator without the need to plug a keyboard in. Almost any modern phone, tablet or computer can be used to enter the Wi-Fi credentials of the preferred access point on behalf of the TV emulator via its captive portal

page. This doesn't require any special software to be installed on those devices because it leverages standard functionality built into their operating systems.

Wi-Fi in hotels and airports usually implement a captive portal web page that asks the user to sign in and/or accept terms and conditions in order to gain Wi-Fi access. In contrast, the captive portal web page hosted by the TV emulator presents the user with a choice of Wi-Fi access points with which to configure the TV emulator's Wi-Fi.

An alternative approach is to have the user connect to the TV emulator's Wi-Fi hotspot and type in the URL of the Wi-Fi configuration page into their web browser. This could be streamlined further by providing access to the configuration page in the 2-IMMERSE companion application.

B.8.1 Captive Portal Components

A captive portal can be built using components available in Ubuntu 17.04:

1. `hostapd` – a daemon that turns the Wi-Fi adapter into an access point
2. `iptables` – for redirecting packets to the captive portal page
3. DNS resolver – for resolving DNS requests to the local web server
4. DHCP server – for leasing an IP address to each companion devices
5. Web server – for serving the captive portal web page

'dnsmasq' is a combined DNS/DHCP server which is easy to configure and use. When a computer connects to an access point, it receives several pieces of information from dnsmasq:

1. An IP address and subnet mask (e.g. 192.168.10.5/24)
2. A broadcast IP address for the subnet (e.g. 192.168.10.255)
3. IP address of the gateway/router (on the same subnet e.g. 192.168.10.1)
4. The IP addresses of one or more DNS servers

After receiving this information, the OS checks to see if there is a captive portal associated with the access point by requesting a URL. For example, on apple devices the requested URL is <http://www.apple.com/library/test/success.html>.

The captive portal detection algorithm can be summarised as follows:

- GET/POST a URL (varies between OS's e.g. <http://www.apple.com/library/test/success.html>)
- If ([success.html] == expected content) => It's an open Internet connection
- If ([success.html] != expected content) => It's a captive portal
- If (http status code != 200 (i.e. success)) => There's no network

B.8.2 Captive Portal Detection Request

The URL request starts with a DNS request to one of the DNS servers advertised by the access point. Access points that implement a captive portal host their own DNS server which resolves all server names (e.g. apple.com) to the gateway's IP address where a webserver is running. A complementary approach is to resolve the DNS request as normal, but redirect all TCP traffic on port 80 to the webserver. This latter approach assumes the DNS server has an Internet connection and can forward on the DNS request. Captive portals effectively implement a "man-in-the-middle" attack.

The computer then makes a HTTP request to the resolved IP address to fetch the content. (For further information, see (37)).

Each operating system uses a different URL to test for the presence of a Captive Portal, see: (38)
All URLs are redirected regardless, so this is only relevant if the captive portal wants to infer the type of platform making the connection request.

The captive portal webserver always responds with a HTTP redirect (302). This informs the connecting computer that this is a captive portal and the OS will open a captive portal window showing the response, which is typically a web page with a login form. If there isn't a redirect, the user won't be shown a captive portal window. See this article for further information: (39)

Also, iOS expects the redirect URL to contain a domain name, not an IP address. If the redirect URL is <http://192.168.10.1> as opposed to <http://myhotspot.localnet> it assumes a home network, not a captive portal. This requires careful configuration of the webserver and DNS server hosted by the access point.

On Apple devices running iOS, the computer may also expect to be redirected to an XML file instead containing a `<WISPAccessGatewayParam/>` XML element. This is a standard referred to as 'CaptiveNetworkSupport'. The XML element contains the actual redirect URL, but also contains other information. iOS devices will only show the captive portal window if they receive a Captive Network Support XML file.

B.8.3 Captive Portal Detection on Samsung Devices

Samsung Android devices (eg. Galaxy S5-S8 smartphones) require all domain names used for captive portal detection to be resolved to public IP addresses by the DNS server (i.e. not 192.168.10.1). At the time of writing, this includes:

- connectivitycheck.android.com
- connectivitycheck.gstatic.com
- clients3.google.com

This is a well-documented difference between Samsung devices and other devices running Android. Samsung appear to perform an additional check on the type of IP address resolved by the DNS server. Since the TV emulator device might not yet have an Internet connection, we can't forward the DNS requests for these domain names to an Internet DNS server to be resolved. Consequently, they need white-listing by the local DNS server so that they are resolved to public-looking IP addresses. Then HTTP packets need to be routed to the local web server at 192.168.10.1 using 'iptables' rules.

B.9 Web Kiosk Service

The web kiosk service has four main components:

1. **Chromium** – The open-source version of Google Chrome
2. **Openbox** - A lightweight window manager supporting hints to enable Chromium to run full-screen
3. **X11** - The display server
4. **PulseAudio** – X11 audio server

The web kiosk runs automatically on boot as a systemd service. It is run as a non-root user with carefully configured group membership to permit audio and video access. The kiosk server starts after all other services have started and the system is idle. It depends on a 'getty' starting to ensure X11 can switch virtual terminals without error.

B.9.1 Chromium

The firmware uses a customised version of Chromium that provides hardware-accelerated video decoding under Linux. See deliverable D3.3, Section 2.4.5.3 for more details. It is available from the following personal package archive (PPA):

```
ppa:saiarcot895/chromium-beta
```

See: (40)

Chromium is automatically restarted by the kiosk service if it terminates unexpectedly and is deliberately restarted when the network configuration of the TV emulator changes. This is because changes to network state can happen at critical points in time such as during a page load and can leave the user interface in a partially loaded state from which there is no recovery without manual intervention.

Chromium is run in incognito mode to ensure it returns to a known good state and to prevent unexpected behaviour resulting from content/credential caching.

The firmware supports all service trials, including 'Theatre at Home' which implements live video chat. This requires Chromium to run with '`--disable-web-security`' to stop it prompting the user for permission to use the web camera and microphone.

Chromium is also configured to allow mixed http and https requests using the '`--allow-running-insecure-content`' option. This permits content on the CDN served via HTTPS to communicate with the local HbbTV2.0 emulation services which are accessed via unencrypted HTTP requests.

Here is the complete command-line invocation of chromium used by the kiosk service:

```
chromium-browser
--incognito -kiosk \
--window-position=0,0 --window-size=1920,1080 \
--disable-web-security \
--allow-running-insecure-content \
--no-first-run --fast --fast-start \
--disable-infobars --disable-translate \
--disable-session-crashed-bubble \
--enable-gpu-rasterization \
--remote-debugging-port=9222 'http://localhost:3000/device'
```

For an extensive list of all Chromium command-line options see: (41)

B.9.2 X11 Server

Usually, the X11 session is started by a display manager. However, when running a single-user system, display managers are an unnecessary waste of resources. Fortunately, the X11 session can be started without one. The display manager is responsible for showing a desktop login prompt, so running without a display manager conveniently eliminates a login step which is undesirable for a kiosk.

It is preferable to run X11 as a non-root user for added security, but also because Chromium under X11 won't run as the root user without specifying the `--no-sandbox` flag. Unfortunately, this flag

causes a notification bar to appear in the window that must be dismissed by the user, which is not good for a web kiosk experience.

To run X11 as a non-root user, it is necessary to install the `xserver-xorg-legacy` package and perform some additional configuration steps. Also, by default, `'startx'` can only be run from a console. To launch it as a `systemd` service, it has to be configured to run from outside a console.

Other niceties are configured using X11 such as disabling the screen saver, the mouse pointer and turning off power management signalling. This prevents the Kiosk from going to sleep during playback of an experience.

B.9.3 Pulse Audio

The kiosk service also runs the PulseAudio Sound Server X11 Startup Script and configures the audio to output over HDMI, setting the volume level to maximum.

B.10 Creating an SSH tunnel for remotely debugging Chromium

Chromium only accepts connections from localhost for debugging a page. If we want to remotely debug a page in Chromium from another computer on the network, we must trick Chromium into thinking the connection is coming from localhost. This is achieved by creating an SSH tunnel that tunnels requests to port 9223 (from any IP address on the network) to localhost on port 9222". Port 9222 is typically the port used to configure chromium to enable remote debugging.

The firmware starts the SSH tunnel at boot using a `systemd` service. See the following URL for more details: (42)

B.11 Web Server

The web server implements the firmware's control logic. It provides a number of REST endpoints that other parts of the firmware interact with. The web server also serves rendered content and static content to display on the TV, within the captive portal page and on the admin portal. The following list summarises the key responsibilities of the web server:

- Mocks Wi-Fi commands with versions that return static content
- Performs Wi-Fi scanning
- Restarts the kiosk service
- Manages websocket connections to the firmware's Chromium browser
- Renders HTML templates and serves static content
- Monitors and reports network connectivity
- Implements portions of the on-boarding flow

The web server works in conjunction with `nginx`. The `nginx` configuration file implements the majority of the captive portal HTTP request handling which includes redirects, WISPR header processing and proxy pass-through to the `node.js` web server. It also has permissions to use privileged port 80 out of the box, allowing the `node.js` web server to run with restricted permissions on port 3000 instead.

B.12 4K/UHD TV Support

The original firmware specification targeted HD TV resolutions, but we quickly discovered that many trial participants were using 4K TVs. A number of updates to the firmware and the Intel NUC were required to support 4K TVs.

B.12.1 CPU Overhead

We discovered a large CPU overhead as a result of stretching HD video content to UHD. This was resolved by enabling GPU rasterisation in Chromium. This also fixed a diagonal tearing problem that wasn't seen at HD resolutions. Investigation of CPU overhead also led to the discovery of redraw optimisations.

B.12.2 Signal Detection

Many 4K TVs weren't detecting the signal from the Intel NUC. This required a Linux kernel update to obtain newer Intel graphics drivers and a corresponding BIOS upgrade.

B.12.3 HDMI 2.0 Cable

A HDMI 2.0b cable is required to support 4K output at 60Hz, but some of the cables in use were HDMI 1.4 cables. HDMI 1.4 introduced the bandwidth required to deliver 4K video, but HDMI 2.0 is required for 4K video at 50 and 60 frames per second. In HDMI 1.4, the rate of 4K was limited to 24 frames per second. HDMI 2.0b cables are marked 'High speed' or 'Category 2'. See: (43)

B.12.4 IGD Memory for 4K

The default IGD Aperture Size on the Intel NUC is 256MB and the IGD Minimum Memory is 64MB. According to the ArchLinux wiki and other forums, the video memory and aperture size need increasing to support 4K@60Hz.

"If you want to use 4K graphic output, open the BIOS settings and set Devices and Peripherals -> Video -> IGD Minimum Memory to 512 MB and IGD Aperture Size to 1024 MB." - (44)

After the BIOS update, the available options for IGD minimum memory are fewer in number. The following settings were required for Samsung 4K TVs:

IGD Minimum Memory: 64MB

IGD Aperture Size: 1025MB

Annex C HbbTV showcases, tools and software libraries

This section provides an overview of the HbbTV 2 showcase applications that have been developed. After that, it introduces the different tools and software libraries that have been produced for the use in HbbTV applications (see Section C.2), Android-based (see Section C.3) and Windows-platform-based (see Section C.4) companion applications. Also, we introduce server-side components, including a stand-alone tool to generate a media timeline for Transport stream files (see Section C.5) and a Material Resolution Service (MRS, see Section C.6) for providing metadata on the correlation between companion streams and DVB services.

C.1 Showcase applications

Showcases included companion applications for the Android operating system and for the Universal Windows Platform.

C.1.1 Companion applications for the Android operating system

The companion applications for Android have been designed to run on mobile handheld devices, i.e. Tablet PCs and Smartphones. The apps play back alternative audio tracks (e.g. alternative language or audio description versions) for a TV programme in sync with the TV content. The Android applications have been implemented in Java. For synchronisation, the Android applications made use of the above-mentioned library, which was also subject to major architecture refactoring. Also, they integrated the MRS client for Android (see Section C.3.1) to retrieve information on the available companion content and its temporal relation to the TV content. The Clock Adapter for ExoPlayer (see Section C.3.2) was needed to adapt the video presentation to the time information from the MRS client and the synchronization library.

C.1.2 Companion application for the Universal Windows Platform

The companion app for Microsoft's Universal Windows Platform (UWP) (45) was designed for Microsoft's augmented reality platform HoloLens (46), though it could be potentially deployed to all other device classes running UWP compatible operating systems such as Windows 10, Windows 10 Mobile. The HoloLens displayed a sign language interpreter next to a TV set, translating spoken text of a news magazine on the TV. The UWP provides APIs in different programming languages. The companion application made use of the UWP's JavaScript API and the Device Discovery and the Media Synchroniser for UWP (see Section C.4.2). Also, it made use of the "dvbcss-clocks" and "dvbcss-protocols" libraries, which was published as open source and the MRS client library for JavaScript.

C.2 HbbTV client libraries

These client libraries are organised as single separate modules helping to simplify application development by wrapping recurring tasks involving the HbbTV terminal API.

C.2.1 Wrapper for the HbbTVCSManager manager

The companion screen manager object in HbbTV is used to retrieve service endpoints for app2app communication and inter-device synchronisation as well as to discover manufacturer-specific launcher applications on companion screens. The wrapper module provides a few convenience functions as well as useful constants e.g. for error handling of the launcher discovery.

C.2.2 Wrapper for the MediaSynchroniser object

Through the MediaSynchroniser object HbbTV apps can handle all new media sync features of HbbTV 2, i.e. inter device synchronisation, multi-stream synchronisation as well as App2AV synchronisation.

C.2.3 Wrapper for the Application manager

This component wraps the application object that is accessed through the oipfApplicationManager embedded object available on HbbTV implementations. It is used for multiple tasks, it controls the visibility of the browser, it is used to request and release key events from and to the underlying terminal, and finally also to start and stop applications. The wrapper module provides some useful constants and methods around these functions, and it also provides some guidance for implementers not yet familiar with HbbTV.

C.2.4 MRS client for JavaScript

The MRS client library is used to retrieve material information from an MRS service, tested with the MRS service described below (see Section C.6), for a given material. It also restructures the info to be easier consumed by applications.

C.3 Companion screen libraries for Android

C.3.1 MRS client for Android

The MRS client for Android is a Java version of the MRS client for JavaScript (see Section C.2.4).

C.3.2 Clock Adapter for ExoPlayer

The library for HbbTV 2-based companion synchronisation provides information on the TV's presentation time in form of a software clock. The companion app has the task of adapting the media presentation to this clock. Companion apps for our showcases made use of library ExoPlayer (47) for audio-video playback. ExoPlayer provides an API to pass software clock objects to which ExoPlayer adapts the playback position of audio or video. An adapter has been implemented which translates between the ExoPlayer clock format and the format of the clock returned by the synchronisation library.

C.4 Companion screen libraries for WinJS UWP

C.4.1 Device Discovery for WinJS UWP

A JavaScript library has been implemented for discovery of HbbTV devices via the DIAL protocol. The library makes use of UWP-specific JavaScript APIs for UDP messaging.

C.4.2 Media Synchroniser for UWP

This component adapts the play position of a video played back by the HTML video element according to the time information reported by the software clock from the dvbcss-protocols library.

C.5 MPEG TEMI Timeline Inserter

DVB CSS defines the concept of media timelines, the specific timelines for common media types and how timelines of different types can be correlated. For the HbbTV tests and showcases a tool was needed to insert a timeline into DVB transport streams, the media format used for DVB broadcast services, to be able to resynchronise the DVB service with broadband streams for rendering on the

client devices (see also the material resolution service that stores timeline correlations of related media).

The TEMI timeline inserter works offline and reads a DVB TS file from the local filesystem and writes the modified file with a TEMI timeline back to disc. The timeline is carried in packets containing tuples of a presentation timestamp (PTS) and a corresponding timeline value (TEMI) as a separate component that is ingested into the file/stream by replacing stuffing packets. The initial timeline value is mapped to the timestamp (PTS) of the first video frame of the selected video component in the TS found. A new timeline packet is generated by default every second using the most recent presentation timestamp from the video stream and a corresponding TEMI timeline value.

An experimental implementation for use with Dektec I/O devices with DVB ASI interfaces is included.

The implementation is based on Microsoft (MS) Visual Studio C++ and executables available for MS Windows.

C.5.1 Example usage

```
TsProcessor --source in.ts --destination out.ts --add_temi 1 256 255 50 254 9
```

The above Command Line Interface (CLI) command creates a timeline with a timeline id of 1, the timeline value for the first frame will be 0, the timeline will run at a timescale of 90000 ticks per second, timeline descriptors are inserted every second (using default values). The timeline is generated for the video component that has 50fps and has a PID of 255. The timeline descriptors are inserted with a PID of 256 and the PMT on PID 254 is updated to signal PID 256 with a component tag of 9. With a timeline id of 1 and a component tag of 9 the timeline selector for use in HbbTV becomes urn:dvb:css:timeline:temi:9:1

C.5.2 Command line parameters

Parameter	Description
--help	show usage information
--source <file path>	Path to source file (DVB transport stream)
--destination <file path>	Path to destination file. Requires --source
--add_temi <temi id> <temi pid> <pts pid> <fps> <pmt pid> <comp tag> [<interval>=90000 <timescale>=90000 <startvalue>=0]]	Add TEMI timeline descriptors. Generated packets will be generated as a new component and inserted with PID <temi pid>, using <temi id> as timeline identifier and <comp tag> for reference of the TEMI packets from the PMT <pmt pid>. <fps> is the frame rate of the video component on PID <pts pid>. The optional arguments timescale and interval define the tick rate of the TEMI timeline and the repetition rate (based on timescale) of TEMI timeline packets.
--live_stream	Use Dektec ASI devices as Input and Output. Experimental. Can't be used with

	--source or --destination
--parse_timeline <temi pid>	Parse TEMI descriptors from a recorded file and print information to stdout. Use with --source

C.6 Material resolution service

The Material resolution service (MRS) is based on the MRS protocol defined for DVB-CSS as a level of abstraction between application and content providers. In the HbbTV showcases it is used to store available companion streams and some metadata like title and type, and most importantly the correlation of the media timelines of the master media and the companion stream.

The service has two separate interfaces, one public interface that is used by clients to query available streams and related data, the other one is used to manage this data that is stored in a local database.

The service is implemented as a node.js application. The service APIs are implemented RESTful, API is implemented and documented using RAML. The internal database is currently file based using the NEDB npm module. The API of this database is a subset of that of MongoDB, i.e. the database could be exchanged quite easily if scalability gets a requirement.

C.6.1 Summary of MRS Query API

`/v1.1/MRS?contentId=<URL>`

Returns the material data currently stored for a content item identified by its content id. Details of content id and the response format can be found in the DVB CSS specification.

C.6.2 Summary of MRS Mgmt API

`/material/{index}` Create, update, delete and retrieve material information.

`/identifier/{index}` Create, update, delete and retrieve identifiers that can be referenced from materials to identify their types etc.

`/timelineInformation/{index}` Create, update, delete and retrieve timeline information that is linked to a particular material, which includes type and parameters of a timeline when it is used as synchronisation timeline.

`/mapping/{index}` Create, update, delete and retrieve mappings, which create the timeline correlation between two materials, typically one being a master media like a DVB service and the other a companion stream.