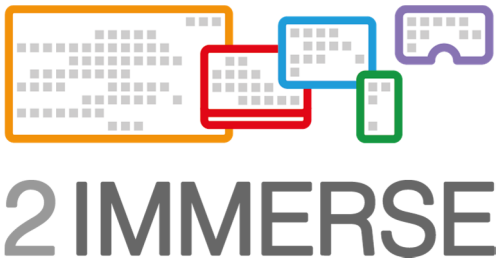
HORIZON 2020
The EU Framework Programme for Research and Innovation

2IMMERSE

ICT-687655

Delivering Single and Multi-Screen Content Services for Immersive, Customised and Shared Experiences in Homes and Social Spaces



D2.3/D5.1 Distributed Media Application Platform and Multi-Screen Experience Components: Description of First Release

Due date of deliverable: 30 Sept 2016

Date of resubmission (this version): 28 April 2017

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

Lead contractor for this deliverable: **Cisco**

Version: 28 April 2017

Confidentiality status: **Public**

Grant Agreement number: 687655 — 2-Immerse — H2020-ICT-2015/H2020-ICT-2015

Abstract

This document describes the first release of the 2-Immerse Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's first service prototype, "Watching Theatre 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 and D2.2.

Although originally defined as two separate deliverables, D2.3 (Distributed Media Application Platform: Description of First Release), and D5.1 (Multi-Screen Experience Components: Description of First Release), these have been combined into a single document which will be easier to read and navigate.

Target audience

This is a public deliverable and could be read by anyone with an interest in the details of the system architecture being developed by the 2-Immerse project. As this is inherently technical in nature, we assume the audience is technically literate with a good grasp of television and Internet technologies in particular. This document will be used by the Project Consortium as a reference as it develops the foundation platform which it describes.

Impressum

Full project title: 2-Immerse

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.3 Distributed Media Application Platform: Description of First Release

Editors: James Walker (Cisco) and Ian Kegel (BT)

Workpackage Leader: James Walker, Cisco

Project Co-ordinator: Helene Walters, BBC

Technical Project Leader: Phil Stenton, BBC

This project is co-funded by the European Union through the ICT programme under FP7.

Copyright notice

Executive Summary

The First 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, and the platform component interfaces defined in project deliverable D2.2.

The first trial to be undertaken by the 2-Immerse project is based on the "Watching Theatre at Home" service prototype. Technical development has therefore been focused on the prioritised requirements of this prototype, as expressed in deliverable D4.1, but it is important to note that the majority of the platform elements, multi-screen experience components and production tools created so far will be enhanced and used again for subsequent trials.

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

- At the core of the 2-Immerse architecture is Mantl, a modern platform for rapidly deploying globally-distributed services. It provides an integrated set of industry-standard open-source components and can be deployed with a variety of different cloud infrastructure providers. This is supported by a basic Origin Server, on which digital content assets, DMApp Components, timeline and layout documents are hosted. The project partners are using a set of private repositories hosted on a GitLab server to manage development of the platform services, client application and DMApp Components. A Continuous Integration/Continuous Development server is used to deploy services on Mantl from their GitLab source.
- The Timeline and Layout Services are unique to 2-Immerse and are based on their initial designs described in D2.2. The Timeline Service has been enhanced to cater for a varying number of companion devices and to schedule future media changes early for a more seamless user experience. The Layout Service has been iterated several times with many new features to support the user experience requirements of the Theatre at Home prototype, including a more efficient API, more flexible specification of layout parameters and an improved model for component lifecycle.
- Lobby and Call Server functions are important elements of the Theatre at Home experience and have been integrated into the Websocket Service since some of their functionality was already available there. The current implementation provides distinct lobbies for text chat and video chat and helps to establish WebRTC peer connections between lobby members.
- A significant new development has been the deployment of the Shared State Service from the FP7 MediaScape project. Its purpose is to provide a repository for key information which must be shared between different clients participating in a multi-screen experience. It includes a simple notification subscription model, managed through a client API, and reduces the amount of bespoke application logic required.
- A simplified inter-home synchronisation solution has been implemented using the WallClock service and the Shared State Service to enable the synchronised start of a Distributed Media Application.
- A flexible logging format has been implemented to allow logs from 2-Immerse services and client applications to be automatically processed by Elastic Stack component

Logstash. The Kibana web application has been configured to allow aggregated logs to be filtered, presented and analysed for efficient debugging and user experience monitoring.

- The Distributed Media Application (DMApp) is a collection of applications which are co-ordinated to deliver a multi-screen experience within a household. In order to facilitate the re-use of key functionality which is common to all experiences, the application stack implemented on each client device is split into multiple layers, including a bootstrapping application, an onboarding application and the application responsible for the individual experience (such as Theatre at Home).
- DMApp Components are a way to encapsulate functionality and user interface elements in discrete entities which are individually specified and controllable by the Layout Service. They are JavaScript objects which as a minimum meet a defined and documented JavaScript interface. 12 reusable DMApp Components have been developed for the Theatre at Home service prototype, and all have been based on the WebComponents web standard. They include components which play audio and video, present text and image content and provide real-time video communication and text chat. The Component Switcher is a key component which provides a UI to enable different parts of the experience (and hence DMApp Components) to be selected.
- The Bootstrap framework has been used to achieve a consistent look and feel across the Theatre at Home DMApp by providing a library of re-skinnable UI elements, a vocabulary of CSS class names that describe appearance and behaviour of web content, and styles and behaviours that facilitate the implementation of responsive layouts.
- As fully-implemented HbbTV 2.0 devices are not yet available and it can't be guaranteed that they will be for any of the trials, a TV Emulator device has been developed which implements the essential protocols required from the HbbTV and DVB specifications.
- The authoring process of the Theatre at Home service prototype has been analysed and will direct further development of more advanced production tools. However, a combination of off-the-shelf and bespoke tools have been used to support the authoring process so far, ranging from image and text editors to text timing editors and tools to visualise and validate layout and timeline documents.

As the first instance of a working platform for the delivery of an interactive, object-based multi-screen experience, this First Release forms a vital foundation for the subsequent prototypes which will be developed and taken to trial in the next 2 years of the project.

Note: Although originally defined as two separate deliverables, D2.3 (Distributed Media Application Platform: Description of First Release), and D5.1 (Multi-Screen Experience Components: Description of First Release), it seemed sensible to combine these into a single document which will be easier to read and navigate.

List of Authors

Mark Lomas – BBC

Tim Pearce – BBC

Rajiv Ramdhany - BBC

Ian Kegel – BT (co-editor)

Jonathan Rennison - BT

James Walker - Cisco (co-editor)

Jack Jansen – CWI

Fons Kuijk – CWI

Michael Probst - IRT

Table of contents

1 Introduction

This document describes the first 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 service prototype, "Watching Theatre at Home" (henceforth referred to as *Theatre at Home*).

Although originally defined as two separate deliverables, D2.3 (Distributed Media Application Platform: Description of First Release), and D5.1 (Multi-Screen Experience Components: Description of First Release), it seemed sensible to combine these into a single document which will be easier to read and navigate.

This first 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. With the initial development focus on the Theatre at Home service prototype, platform development has been focused on the infrastructure, services and client application to support this service prototype. Similarly, the development of Production Tools and Multi-Screen Experience Components has been prioritised according to Theatre at Home requirements.

The deliverable is structured as follows:

- Introduction - introduces the first release distributed media application platform, and explains how the rest of the deliverable is structured.
- Snapshot of the platform and components – provides a brief visual overview of the First 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 Theatre at Home prototype so far, as well as additional components whose development is anticipated in the short term.
- Production Tools – describes the first set of production tools developed to support the authoring of the Theatre at Home service prototype.

1.1 Terminology

The project has adopted some specific terms to describe aspects of the 2-Immerse platform and its operation. The following may be found within this document:

- **Experience** – 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.

- **Distributed Media Application (DMApp)** – 2-Immerse multi-screen entertainment experiences are composed of many applications configured to work together to deliver the look and feel of a single application. 2-Immerse calls this collection a Distributed Media Application, or DMApp.
- **Distributed Media Application (DMApp) Component** - In 2-Immerse, re-usable components are assembled within a Distributed Media Application (DMApp) to create coherent multi-screen experiences.
- **Context** – 2-Immerse defines a 'context' as one or more connected devices collaborating together to present a media experience. Each context has a 'contextID' unique to its session. There can be many contexts on a single LAN (eg. 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.

2 Snapshot of the platform and components

This section provides a visual overview of selected aspects of the First Release of the 2-Immerse platform and components. It is intended to give the reader an appreciation of the scope of what has been achieved so far by means of diagrams and screen captures which hopefully help to place in context the technical detail which follows.

2.1 Platform Architecture for the First Release

In deliverable D2.1 we introduced a 'layered' approach to documenting the 2-Immerse architecture, with layers defined as follows:

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

Figure 1 below shows the 2-Immerse Platform Architecture as implemented for the First Release. The grey blocks in Figure 1 show the relationships between the key architectural elements: the Platform Infrastructure, Platform Services, the Client Application (or DMApp), the DMApp Components and the Production Tools.

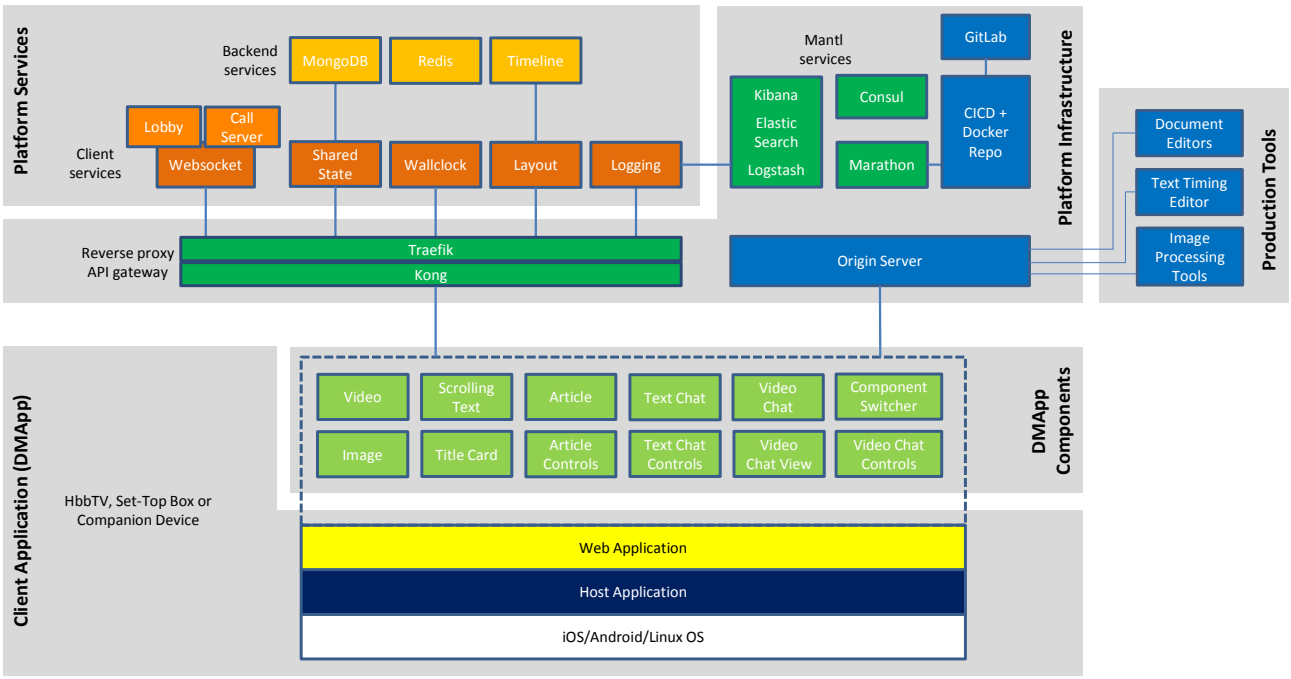


Figure 1: 2-Immerse Platform Architecture for the First Release

In deliverable D2.2, we outlined our decision to use Mantl to provide essential platform infrastructure to support the 2-Immerse platform services. The Mantl services deployed for the First Release (shown in dark green) are the reverse proxy Traefik, the API gateway Kong, the Elastic Stack for log data ingestion, indexing and analysis, service registry Consul and the container orchestration platform Marathon. Additional platform infrastructure (shown in blue)

has been deployed alongside Mantl to provide private code repositories (GitLab) and Continuous Integration/Continuous Deployment service to automate the build and deployment of service and application code. The Origin Server provides crucial hosting functionality.

The 2-Immerse services are divided into client-facing (shown in dark orange in Figure 1) and back-end (shown in light orange).

The Websocket service provides websocket communication functionality between client components and platform services, and encapsulates the Call Server and Lobby capabilities required to enable video and text chat for the Theatre at Home trial.

The Shared State service, based on a server developed by the FP7 MediaScape project, provides shared storage of state information supported by a notification subscription model. It makes use of back-end document-oriented database service MongoDB.

The Wallclock service provides time synchronisation capabilities for services and client nodes deployed within the 2-Immerse platform.

The Layout service, unique to 2-Immerse, is responsible for managing and optimising the presentation of a set of DMApp Components across a set of participating devices. The Layout service uses back-end service Redis as a performant in-memory data structure store. It communicates with the Timeline back-end service, also unique to 2-Immerse, which is responsible for the overall temporal orchestration of an experience (DMApp) within a single household.

The Logging service facilitates the transmission of log messages from Client applications to the Elastic Stack.

Figure 1 also shows how the Client Application (DMApp) architecture is centred around a Web Application (shown in yellow) hosted on a variety of different platforms, including iOS and Android for companion devices and Linux/Unix for HbbTVs and TV emulators. It shows the set of DMApp Components (shown in light green) developed for the Theatre at Home trial, whose presentation on companion and TV devices is controlled by the Layout service. More information on each of the components shown is provided in Section 6 of this document.

Finally, the Production Tools used to create the Theatre at Home experience are indicated separately. As explained later in Section 7, the tools used at this early stage in the project (shown in blue) were relatively simple and enabled manual creation and editing of image and text content, timeline and layout metadata and timing metadata for the Scrolling Script DMApp Component.

Readers should note that the architecture and components described here represent a snapshot of the First Release. The 2-Immerse platform will evolve and improve to support subsequent trials, particularly with the addition of new client-facing services, DMApp Components and Production Tools.

2.2 2-Immerse Wiki

The 2-Immerse Wiki, publicly-accessible via the 2-Immerse website, provides an interactive reference for many aspects of the 2-Immerse Platform (https://2immerse.eu/wiki/api/). It

includes detailed information about 2-Immerse services and high-level descriptions of their APIs, as illustrated in Figure 2. This reference was used to create deliverable D2.2 and will be updated as the platform develops.

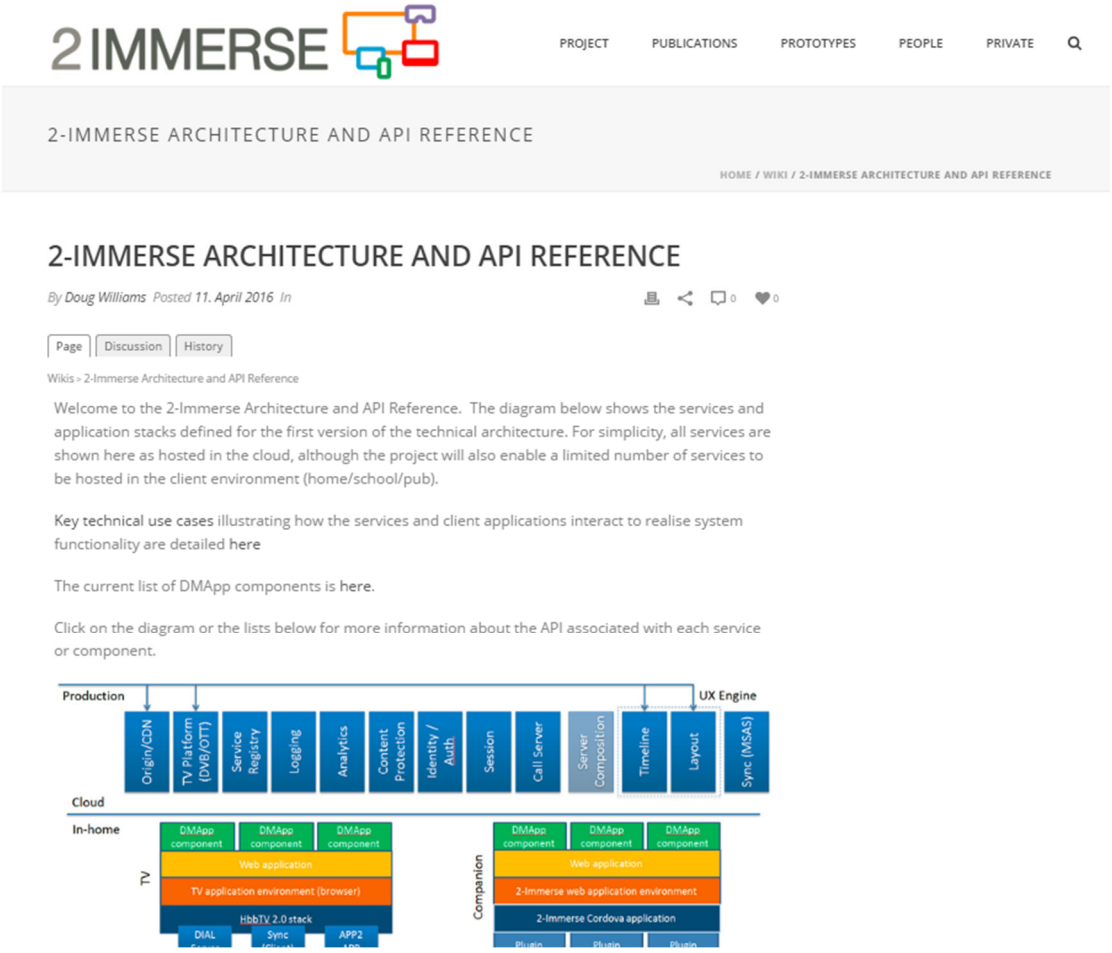


Figure 2: Architecture and API reference on the 2-Immerse Wiki

2.3 2-Immerse Code Repository (Gitlab)

The project partners are using a set of private repositories hosted on a GitLab server by IRT to manage development of the platform services, client application and DMApp Components. The Gitlab repositories (as illustrated by the screenshot in Figure 3) contain source code, resources and technical documentation, including low-level descriptions of APIs and the Timeline and Layout documents which are used to author multi-screen experiences.

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

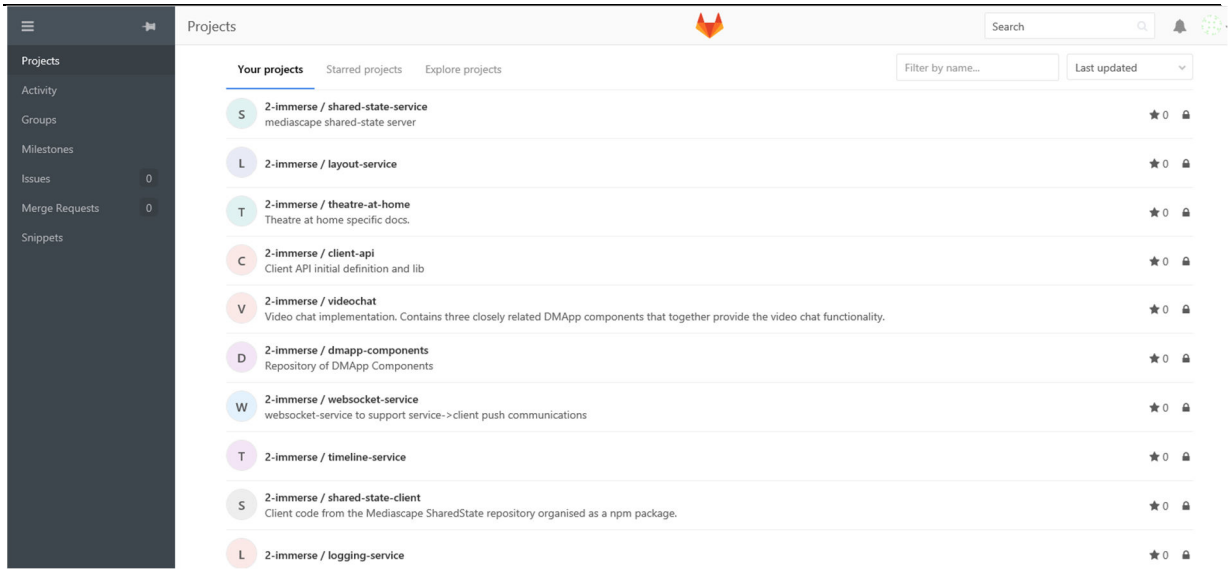


Figure 3: Repository list on the 2-Immerse Gitlab server

2.4 Mantl Services

Figure 4 shows the homepage of the Mantl web interface, which provides access to Mantl services which form the foundation of the 2-Immerse platform, including Marathon, Consul, Traefik, Elasticsearch and Kibana.

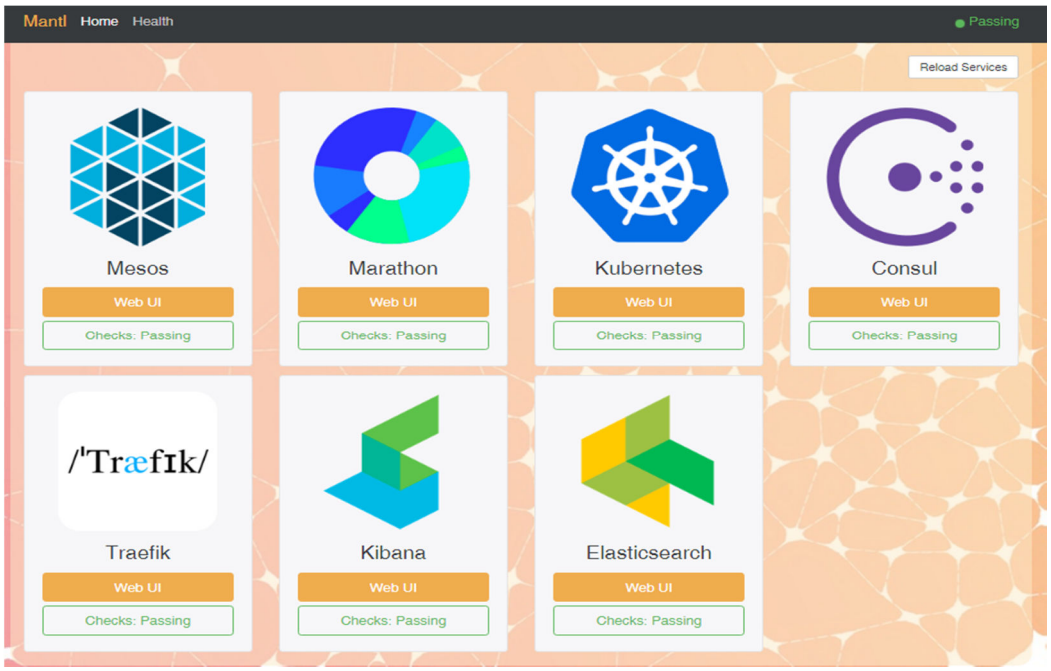


Figure 4: Mantl dashboard

Figure 5 shows the web interface for Marathon, which provides an overview of all micro-services running on the platform.

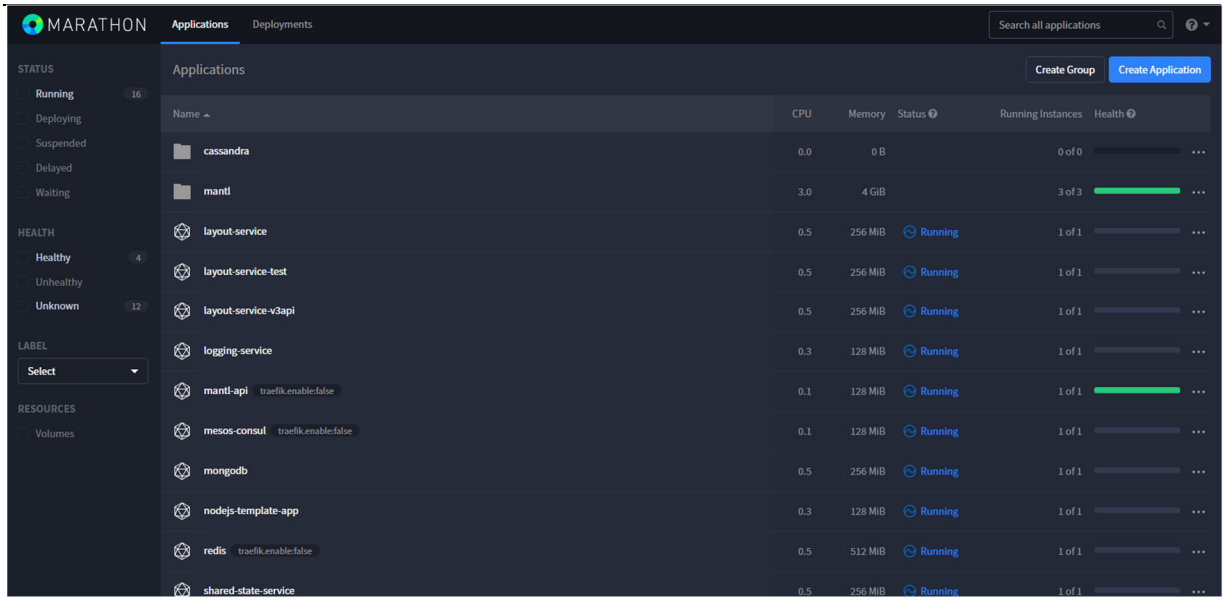


Figure 5: Marathon web interface showing status of micro-services

2.5 Backend Services

Figure 6 shows the web interface for the Kibana service, which provides near real-time access to combined log data from all clients and services which comprise the 2-Immerse platform.

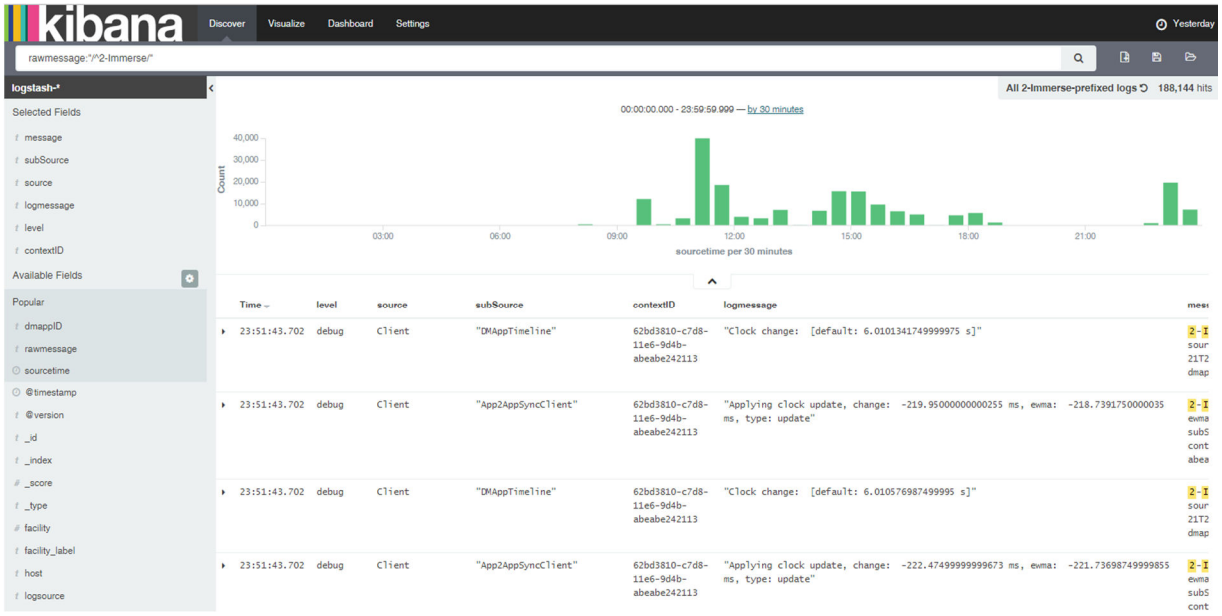


Figure 6: Kibana web interface showing how logs are displayed and searched

2.6 Timeline and Layout Documents

As explained above, new document formats have been designed to enable timeline and layout specifications to be authored for a multi-screen experience. Figures 7 and 8 below show incomplete extracts from the Timeline and Layout Documents created for the Theatre at Home trial.

The Timeline Document format is inspired by the W3C SMIL specification, extended with attributes specific to 2-Immerse requirements.

The Layout Document is written in JSON and specifies layout constraints for each DMApp Component within a DMApp.

```
<tl:document
  xmlns:tl="http://jackjansen.nl/timelines"
  xmlns:tim="http://jackjansen.nl/2immerse"
  xmlns:tic="http://jackjansen.nl/2immerse/component"
  xmlns:tlcheck="http://jackjansen.nl/timelines/check"
  title="Home Trial version 0.4">

  <tl:seq>

    <tl:par tl:end="master" title="Title">

      <tl:sleep tl:dur="5" tl:prio="high"/>

      <tl:ref tim:dmappcid="titlecard"
              tim:class="title-card"
              tim:url="../../dmapp-components/title-card/title-card.html"
              tic:title="Hamlet"
              tic:author="William Shakespeare"
              tic:brand="Royal Shakespeare Company"
              tic:brandImageUrl="../../media/rsc/rsc_logo.tmb-logo-200.png"
              tic:synopsis="The searing tragedy of young student Hamlet, tormented by his father's death
              tic:posterUrl="../../media/rsc/hamlet/hamlet-production-king-queen-1.jpg"/>

    </tl:par>

    <tl:par tl:end="master" title="Full show">
      <!-- Components that remain active throughout the whole experience -->

      <tl:ref tim:dmappcid="componentswitcher"
              tim:class="component-switcher"
              tim:url="../../dmapp-components/component-switcher/component-switcher.html"
              tic:articlegroupid="article-group-123"
              tic:imagegroupid="image-group-123"
```

Figure 7: Extract from a Timeline Document

```
{
    "version": 3,
    "dmapp": "200",
    "constraints": [
      {
          "componentId" : "titlecard",
          "personal": {
              "minSize": {
                  "width": 100,
                  "height": 100
              },
              "prefSize": {
                "width": 100,
                "height": 100,
                "mode": "percent"
              },
              "targetRegions": [ "stage" ],
              "priority": 10
          },
          "communal": {
              "minSize": {
                  "width": 100,
                  "height": 100
              },
              "prefSize": {
                "width": 100,
                "height": 100,
                "mode": "percent"
              },
              "targetRegions": [ "stage" ],
              "priority": 17
          }
      },
      {
          "componentId" : "RSClogo",
          "personal": {
              "minSize": {
                  "width": 100,
                  "height": 100
              },
```

Figure 8: Extract from a Layout Document

Figure 9 shows how a Layout document authored for the Layout Service can be visualised during testing and debugging. The boxes show how DMApp Components have been allocated to layout regions.

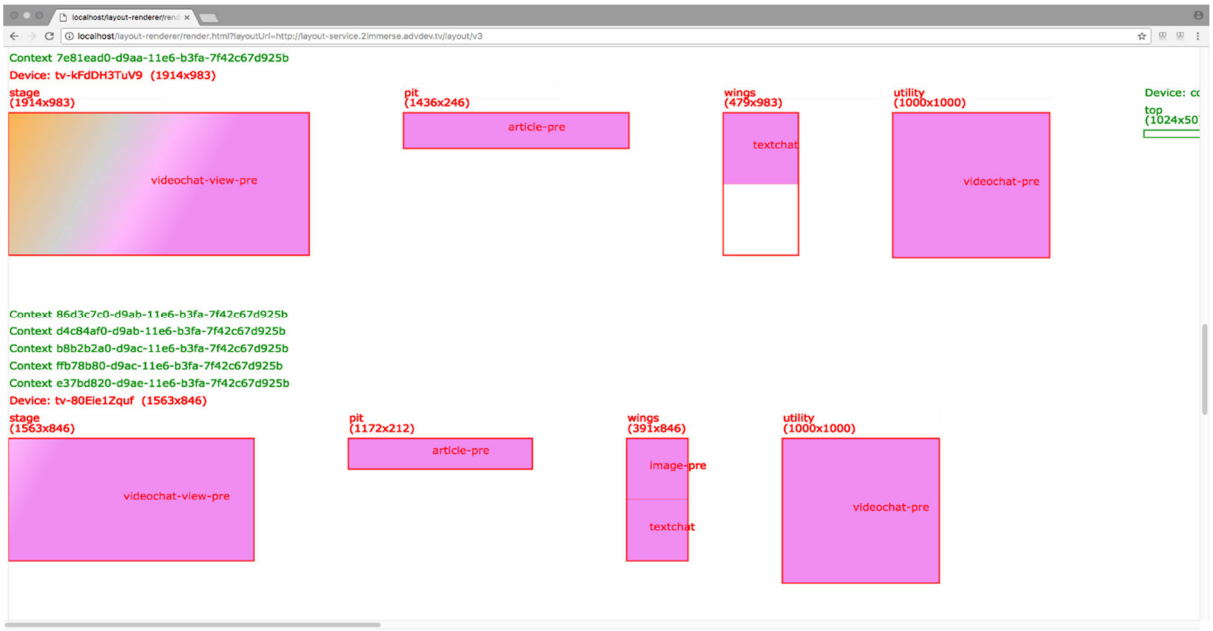


Figure 9: Example output from Layout Renderer tool

2.7 TV and Companion Client applications for the Theatre at Home trial

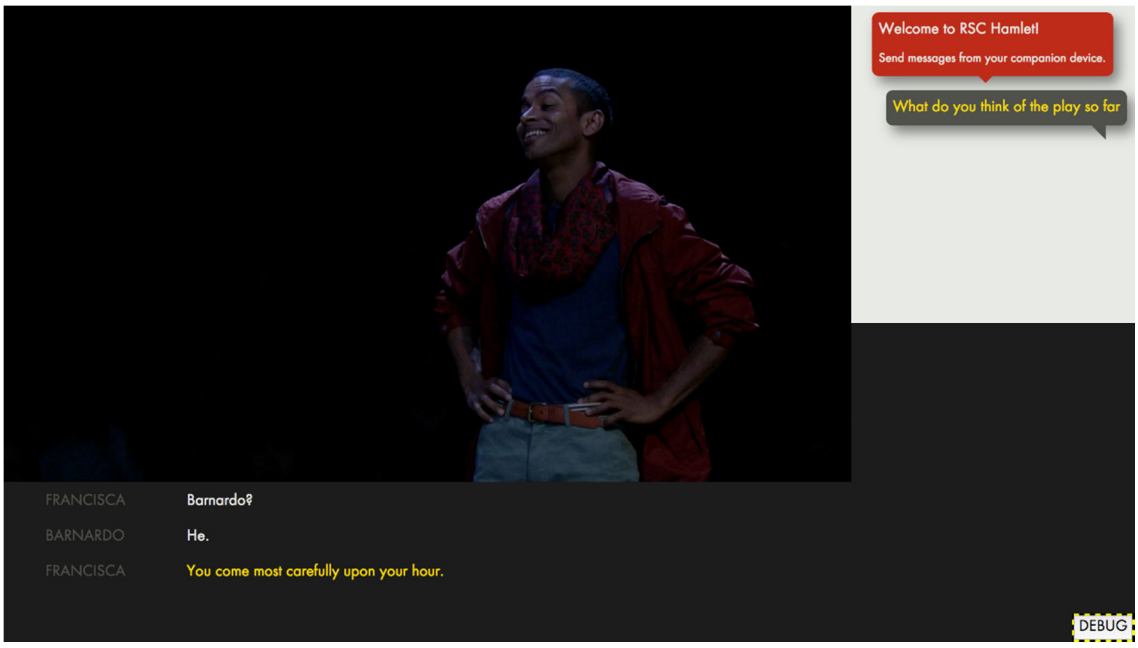


Figure 10: Example of TV Emulator display from the Theatre at Home DMApp

Figure 10 shows a screenshot of the TV Emulator client display while running the Theatre at Home DMApp. The figure shows the layout of DMApp Components during playback of the as-live video from the theatre. Beneath the Video component, the Scrolling Text component highlights the script line which is currently being spoken. To the right hand side, the Text Chat component displays the text chat shared between the households who are watching the play together.

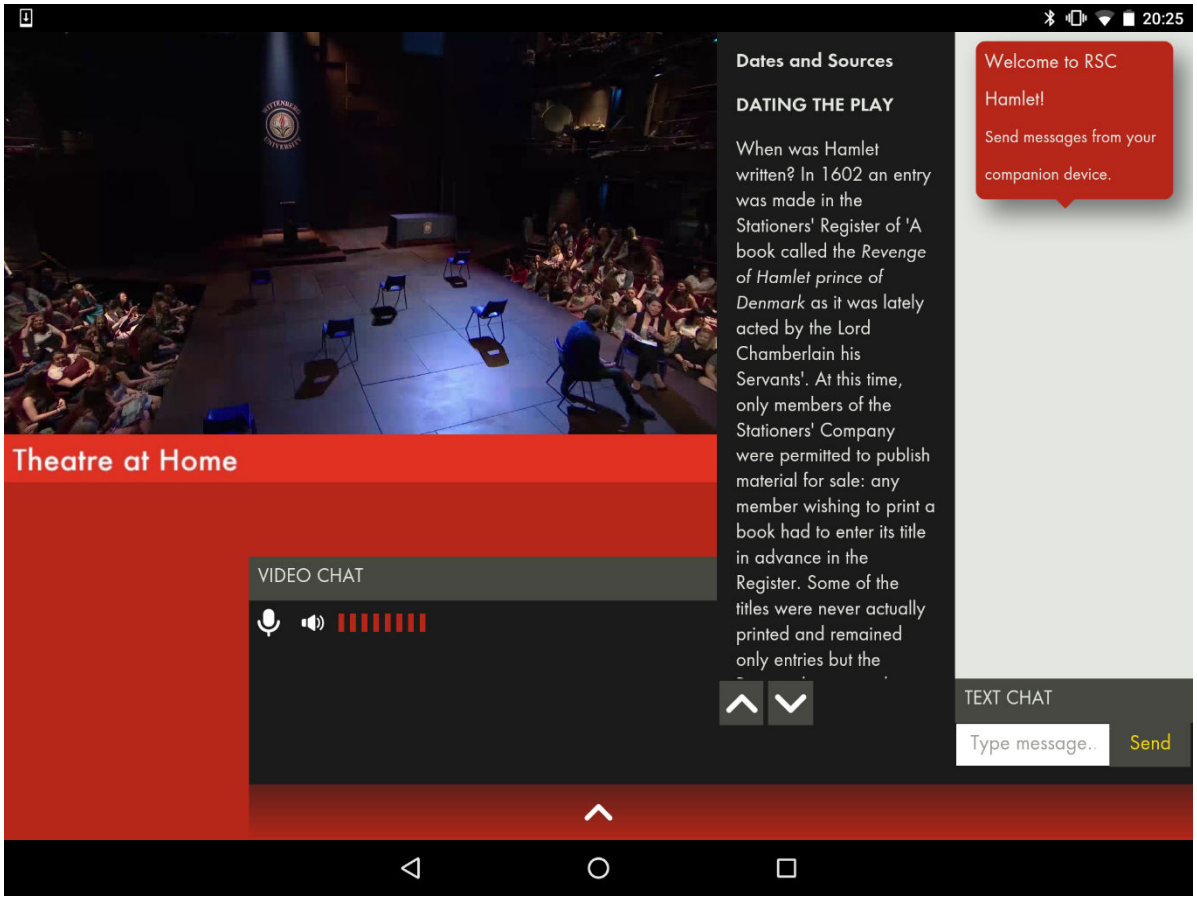


Figure 11: Example of Companion device display from the Theatre at Home DMApp

Figure 11 shows a screenshot of the Companion client display while running the Theatre at Home DMApp. The figure shows the layout of DMApp Components while households are waiting for the play to start. In the top left, the Video component presents a short interview video introducing the play. Beneath it, the Video Chat Control component enables users to mute their microphone or change the audio volume during video chat, which is displayed on main TV screen. To the right, the Article component displays supplementary information about the play, which can be scrolled using the arrow buttons of Article Control component beneath. On the far right, the Text Chat component is accompanied by the Text Chat Control component which allows messages to be entered and shared. At the bottom of the screen, the chevron allows the Component Switcher component to be shown and hidden as required. The Component Switcher allows additional content to be selected in Video, Article and Image Components, some of which are shown on the TV display as well.

3 Platform Infrastructure

This section of the deliverable describes the infrastructure deployed to support the 2-Immerse service platform. The core Mantl platform was outlined and described in D2.1 and D2.2, so here we summarise the platform as we have deployed it, together with additional service and components that we have adopted.

3.1 Mantl

As noted in D2.2, Mantl is a modern platform for rapidly deploying globally distributed services, typically as containers. It provides an integrated set of industry-standard open-source components. It is cloud infrastructure provider agnostic, and can be deployed on Amazon Web Service (AWS), Google Compute Engine (GCE), Microsoft Azure, OpenStack, Vagrant, Bare Metal etc.

Mantl is licensed by Cisco under the Apache Version 2 License.

Latest Mantl documentation: http://docs.mantl.io/en/latest/

3.1.1 Marathon / Mesos / Docker container platform

Containers provide a lightweight and secure mechanism for distributing and running applications and services, and are commonly used for micro-service deployment. Marathon is a widely-used container orchestration platform for Apache Mesos, which is a cluster manager providing data centre resource management to distributed applications and frameworks. Both Marathon and Mesos are an integral part of the Mantl platform. Mantl uses Docker as the Mesos container run-time engine.

Latest Marathon documentation: https://mesosphere.github.io/marathon/

Latest Mesos documentation: http://mesos.apache.org/documentation/latest/

Latest Docker documentation: https://www.docker.com/

3.1.2 Traefik – reverse proxy / load balancer

Traefik is an HTTP reverse proxy and load balancer that is part of the Mantl platform, exposing containerised services onto well-known service end-point URLs.

Latest Traefik documentation: https://traefik.io/

3.1.3 Kong – API Gateway

We are planning to use Kong as an API gateway for the service platform, which will facilitate identity management between multiple contexts joining a shared experience. At the time of writing this has not been implemented and so will be documented further in the next release.

Latest Kong documentation: https://getkong.org/

3.2 Origin Server

We have deployed a basic origin-server which we are using to host client applications, DMApp Components and their assets, timeline and layout documents, and media.

We have devised a basic asset structure for organizing these assets on the origin server.

3.3 CI/CD and Docker Repository

Within the environment which we are using to host the Mantl cluster we have also deployed a virtual machine to host a CI/CD (Continuous Integration/Continuous Development) server (running Jenkins http://jenkins-ci.org/), and a Docker registry.

We are currently using the CI/CD server to deploy services to Mantl. There is potential to use it as a build server for client applications in future.

3.3.1 Service Deployment Workflow

The typical workflow for deploying a service is as follows:

Develop service:

- Create a Docker file
- Create a JSON service template for Marathon deployment
- Push to source code repository on IRT GitLab

Run a Jenkins Job that will:

- Pull the project source code from IRT GitLab
- Build the Docker image and push to the Docker registry
- Run any automated build tests
- Deploy the Container using the Marathon REST API using the previously defined JSON service template

Currently Jenkins jobs are manually invoked by developers, but they could be triggered automatically by commits to the IRT GitLab repository in future.

4 Platform Services

This section of the deliverable describes the core platform services developed and deployed to date, and their current status.

4.1 Timeline

The Timeline Service is responsible for the overall temporal orchestration of the experience (DMApp) within a single household. The core design which we described in D2.2 has been implemented in this software release, with a number of changes in various areas.

The REST API has been updated to cater for a varying number of companion devices: any media item that is to be run on a companion device can have zero, one or many instances active at any time. Whilst the Layout Service is responsible for actually starting and stopping these instances as companion devices become available or go offline, the Timeline Service has to be aware of this to ensure seamless temporal orchestration. A second change from the initial design is that the 'dmappcStatus' call has been modified so that clients can inform the Timeline Service about the expected end time of media items (such as pre-recorded video or audio) before the media has actually run to completion. This allows the Timeline Service a "limited clairvoyance", which enables it to schedule future media changes early, leading to a more seamless user experience.

The Timeline Document format (an example of which was shown in Section 2) has been updated to support functionality required by the Theatre at Home trial, including providing relevant attributes for the various DMApp Component types which are used.

4.2 Layout

The Layout Service is responsible for managing and optimising the presentation of a set of DMApp Components across a set of participating devices. It is implemented as a node.js application that builds on a set of widely used npm packages.

In deliverable D2.2 we presented a v1 REST API (defined in RAML), and the format of the Layout Requirements Document was being developed. At that time we had simple node.js app using Osprey (https://github.com/mulesoft/osprey) to provide a stub implementation of the API, and were starting to build out the implementation behind the API.

Since that time, the Layout Service has developed as follows:

- Specification and implementation of the layout requirements JSON document format (an example of which was shown in Section 2)
- Specification and implementation of the websocket push message format, integrating with the websocket service
- Specification and implementation of a v2 API
    - Transaction API (a batch API for component state changes to avoid multiple spurious layout passes)
    - Override of layout priorities
    - Dynamic and template layout models

- Integration with Redis (http://redis.io), used as a performant in-memory data structure store as a backend to allow us to scale up service instances in future.

We have just completed specification of a v3 API. This includes a number of features required for the UX that has been defined for the Theatre at Home Service Prototype, including:

- Logical Regions (Extend the current 'physical' device model (i.e. a rectangle with dimensions as reported in device caps), to support multiple 'logical' regions that can be flexibly composed by the client.
- Support for layout co-ords returned by service as percentages rather than pixels
- Preferred Size as a constraint (in addition to minimum size), that can be over-ridden by clients
- Overlays (allowing a component to define logical regions to enable other components to be laid out as an overlay)
- Destroy Transaction

We are also reviewing at the current component lifecycle model to handle multiple instances of components, and component migration cleanly.

4.3 Websocket

The basic Websocket Service API has been implemented as defined in D2.2 with a few modifications.

The Lobby and Call Services were integrated into the Websocket Service implementation, since some of the required functionality is already provided by the Websocket Service.

The Websocket Service is implemented as a node.js application that builds on a set of widely used npm packages, notably socket.io.

4.3.1 Lobby

4.3.1.1 Lobby API Changes since D2.2

The Call Service proposed in D2.2 was based on peer-server – a WebRTC signalling service that accompanies the open source peer.js client library. Unfortunately, development and support of peer.js has effectively ceased and a more streamlined open source alternative called Simple-Peer has been adopted instead. For source code and API documentation, see the Simple-Peer GitHub page: https://github.com/feross/simple-peer

A Simple-Peer compatible call server does not need to be aware of the WebRTC signalling protocol unlike the peer-server/peer.js implementation referenced in D2.2. Only a 2-way channel of communication between remote peers is needed that permits the exchange of arbitrary payloads. This has simplified the 2-Immerse Call service implementation to the extent that we decided to roll it into the Lobby service implementation as a single message send/recv endpoint with payloads determined entirely by the WebRTC clients.

4.3.1.2 Lobby/Call Service Implementation

Clients connect to the Lobby service via a socket.io namespace and lobbies are implemented as socket.io rooms. This has simplified the Lobby service considerably whilst facilitating horizontal scalability. See notes on Lobby/Call service scalability.

The existing 2-Immerse Websocket Service is already implemented using socket.io so it made sense to implement the Lobby service as a plugin for this service to keep the number of websocket-like connections between the 2-Immerse service architecture and its clients to a minimum.

Lobby clients can connect to the socket.io '/lobby' namespace using the standard socket.io.client library and they will automatically benefit from websocket and long-polling support in addition to automatic reconnection. For convenience, we have implemented a small client library that provides high-level event notifications and lobby/call operations.

4.3.1.3 Scalability

The Lobby service implementation is socket-io adapter-safe; meaning that it only utilizes methods that are delegated to socket.io's adapter layer and it only uses socket.io's asynchronous interface. This is necessary because some adapter implementations are asynchronous such as the socket.io.redis adapter. This adapter in particular, will allow the Lobby/Call service to scale horizontally based on load and/or number of client connections. See: https://github.com/socketio/socket.io-redis

The Lobby service is also stateless which ensures the number of server instances can be scaled up or down robustly and permits the Lobby service to support a large number of concurrent lobbies and connections. As a result, lobbies are stateless entities that are distributed over a number of server instances.

The current implementation provides distinct lobbies for text-chat and video-chat and helps to establish WebRTC peer connections between lobby members.

4.3.1.4 Lobby Identifiers

Lobby identifiers are now an integral part of the on-boarding flow for 2-Immerse experiences and are derived from the inter-context sync identifiers used to group households together into a shared, synchronized experience.

4.3.1.5 Near-term changes

In the near-term, clients of the Lobby/Call service will need to specify a valid session token in order to instigate peer-to-peer calls and join lobbies. This will occur after the identity management aspects of the 2-Immerse service architecture have been configured. It will be possible for the Lobby service to exchange the user's session token for user credentials, such as a person's display name. User identity is essential for functionality related to user profiles, analytics, security, personal devices and social features. Currently, all callers and lobby members are anonymous, which is just sufficient for the first trial.

A longer-term task (required for the larger trials) involves replacing the default in-memory socket.io adapter with socket.io.redis to provide a scalable back-end implementation.

The Websocket/Lobby service and the shared-state service use socket.io independently, resulting in some duplication within the architecture. We are aiming to reduce this down to a single socket.io connection per client that goes via the Websocket-service. This will simplify maintenance and make it easier to reason about whether a client is on-line or off-line by avoiding 'partially' on-line states. The Websocket service is scalable and extends the reach of the Redis micro service event bus (proposed for inter-service communication) out to client devices via socket.io. The shared-state service will benefit from this scalable client-server communication and presence determination if it implements its communication layer using the Websocket service. The shared-state service may ultimately become a Websocket-service plugin too that uses Redis for persistent state storage.

4.4 Shared State

In the original system architecture described in D2.1, we had anticipated the need for DMApp Component state to be migrated as the components themselves are migrated between client devices, and had made provision for this in the original Layout Service API.

As the client application and component architecture has evolved, the requirements for inter-component communications and distributed functionality have become clearer. Although we have both the websocket service and the HbbTV app-to-app server websocket mechanisms available for inter-component communication, different patterns require different approaches (one-to-one RPC-like, one-to-many, many-many pub-sub etc.).

As we reviewed work done in previous projects, we thought that the shared-state server developed in the FP7 MediaScape project (http://mediascapeproject.eu/shared-state.html) would likely meet our requirements for a simple shared state model where clients have a simple notification subscription model, managed through a client API.

On this basis we have deployed the MediaScape shared state service within our platform. This was relatively straightforward, requiring Dockerisation of the node.js based server itself, and the MongoDB backend it uses. We have deployed MongoDB into Mantl using a Docker image publicly available on DockerHub.

https://github.com/mediascape/shared-state

4.5 Logging and Monitoring

In D2.2 we explained how internal platform logging would be provided by the Elastic Stack (formerly ELK – see https://www.elastic.co/products) instance provided within the Mantl platform.

The three components which comprise the Elastic Stack – Logstash, Elasticsearch and Kibana – all run within separate containers within the Mantl platform, as do each all of the 2-Immerse platform services. Docker is configured with a logging driver whose job it is to collect logs from containers and send them to an appropriate target. Within Mantl, this logging driver automatically takes all output sent to stdout and stderr by the 2-Immerse platform services and directs it to the Logstash syslog input plugin.

Although this makes it unnecessary for 2-Immerse services to use the syslog protocol themselves, it has been important for us to define a flexible logging format which specifies data fields which can be automatically extracted by the Logstash pipeline and indexed within

Elasticsearch. The following is an example log message, with data field prefixes shown in **bold**:

```
2-Immerse contextID:78ff9e20-9488-11e6-be3b-53bd9dd1ff12 dmappID:799a7e90-
9488-11e6-be3b-53bd9dd1ff12 deviceID:799a7e90-9488-11e6-be3b-53bd9dd1ff12
logmessage:'dmappID and deviceID are the same. This cannot be right!'
source:LayoutService sourcetime:2016-10-19T12:01:45.000Z
```

The complete range of fields extracted by Logstash at the time of writing is as follows:

- **contextID, deviceID, dmappID and dmappcID:** Identifiers for the current context, device, DMApp and DMApp Component respectively.
- **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 (ie. the log level)
- **sourcetime**: The timecode (local to the source) at which the message was created

The Client Application and its associated DMApp Components cannot send log messages to the Docker logging driver in the same way. Logging from these external sources is achieved by posting a JSON structure to a dedicated Logging Service which runs within Docker. The Logging Service reformats received JSON structures into the message format shown above and sends them to stdout, from where they are passed to Logstash in the same way as for other services. The Logging Service can receive a single log message at a time or an array of messages packaged within a single JSON structure.

Once log messages have been processed by the Logstash pipeline and indexed by Elasticsearch, they can be viewed and analysed within the Kibana web application. Kibana has been configured with a range of pre-defined searches which enable 2-Immerse platform logs to be filtered and presented in a meaningful way – for example using the 'sourcetime' field to ensure the correct sequencing of logs between multiple services. In addition, the debugging component which is integral to the Client Application automatically constructs a hyperlink to a Kibana search based on the context in which the DMApp is running.

Kibana also provides the ability to create dashboards which present multiple visualisations to summarise data collected during a particular time interval. An interactive '2-Immerse Sanity Check Dashboard' shows key data which supports debugging activities, for example allowing developers to identify which Components were active within a particular context, and which of these reported errors.

In addition, 2-Immerse plans to make use of Google Analytics as a complementary solution for logging of user interactions with DMApp Components. Google Analytics is much better suited to recording fine-grain interaction events which are linked to specific research

questions to be addressed by each of the project's trials. At the time of writing, Google Analytics event tracking had not yet been incorporated within the DMApp Components, but is expected to be completed in time to capture data from Theatre at Home triallists.

4.6 WallClock Service

The WallClock service provides time synchronisation capabilities for services and client nodes deployed within the 2-Immerse infrastructure. The notion of a common synchronized WallClock to facilitate the distributed synchronisation of media playback, was described in deliverable D2.2. Each terminal presenting or directing the presentation of media need to have a common notion of time – this is achieved by the maintenance of a local Wall Clock that is synchronised to a master WallClock.

4.6.1 WallClock sync protocol

The time synchronisation protocol implemented by the WallClock service is an adaptation of the DVB-CSS's WallClock synchronisation protocol (CSS-WC) to fit wider internet deployments. It provides a choice of transports (UDP, WebSockets) and message serialisation capabilities (binary message format, JSON format) to suit the context. For example, UDP and binary message formats are used in scenarios where inter-device synchronisation is needed. The WebSockets transport and JSON message formats are more suited to distributed synchronisation scenarios that require interactions across network boundaries.

The protocol is intended to be a request-response protocol that functions identically to that defined in clause 8 of the DVB CSS specification as the "Wall Clock protocol". As a quick reminder: the protocol is a request-response exchange that is initiated by the party wishing to sync its clock (client) to the other party (server).

When a node issues a WallClock protocol request or response, it needs to specify the type of the protocol message (in the "type' message field) as per the following values:

Value	***Meaning***
0	Request
1	Response with no follow-up planned
2	Response with follow-up planned
3	Follow-up

The messages can then be serialized to binary or JSON format. The binary format for a WallClock protocol message is identical to the DVB CSS format.

With regards to the JSON serialization format, the protocol message carries the same fields as the DVB CSS wall clock protocol, however instead of encoding them in a binary structure, they are instead carried in a JSON object. The properties of the object are as follows:

Example: (with explanatory comments that must be removed for it to be valid JSON)

```
{
    "v":    0,              /* version = 0 */
    "t":    2,              /* type = response with follow-up planned */
    "p":    0.0001,         /* server clock has 0.1 millisecond precision */
    "mfe":  50,             /* server clock max freq error = 50 ppm */
    "otvs": 19346582,       /* client request sent at 19346582.9826511 seconds */
    "otvn": 982651100,
    "rt":   29784724.1927,  /* server received request at 29784724.1927 seconds */
    "tt":   29784724.1938   /* server sent response at 29784724.1938 seconds */
}
```

4.6.2 Additonal Libraries and Deployment

The WallClock Service was implemented using node.js and the source code is available here*:* https://gitlab-ext.irt.de/2-immerse/wallclock-service/tree/master

The common functionality to all synchronisation protocols such as message serialisers, message handler registration was extracted into a separate JS library, called **sync-protocols** (https://gitlab-ext.irt.de/2-immerse/sync-protocols).

To enable developers to manipulate WallClock time and media timelines, a JS library (**dvbcss-clocks**) was developed and released (https://gitlab-ext.irt.de/2-immerse/dvbcss-clocks). This library is used by the Timeline Service and Client API to create clock objects for representing WallClock time and timelines.

The WallClock Service is deployed on the Mantl platform and exports the following endpoints:

- **WebSockets**: ws://wallclock-service.2immerse.advdev.tv:80
- **UDP**: wallclock-service.2immerse.advdev.tv:6677

The synchronisation accuracy achieved via the WallClock service algorithm is approximately ± 9ms. This lies within the range of typical video-frame durations (e.g. a 50 fps video) to achieve frame-accurate or near-frame-accurate synchronisation.

4.7 Inter-Home Sync

The provision of a solution to achieve inter-home media synchronisation was simplified for the initial service prototype. The reasons for this were two-fold:

- **Frame-accurate synchronisation is not mandatory for inter-home experiences** – When users are part of an experience spanning across homes and communicating via

voice or video conference, the perception of asynchrony becomes looser. Stricter synchronisation accuracy is, for instance, required if the experience spans only one room.

- **Cloud-based synchronisation solution required the proposal of a new algorithm** - the DVB-CSS media sync model relies on assumptions related to the local network context that do not lend well to a more distributed environment. It assumes the WallClock of local synchronisation slaves locked onto the WallClock of the master device via a UDP-based sync protocol. Also, the provision for slave media-sync functionality on HbbTV enabled devices is non-mandatory. This means that there is no guarantee that a DVB-CSS based cloud-sync solution would be compatible with future TVs. Hence, a decision was made to
    - push the cloud-sync client functionality on the TV to its application environment
    - free the cloud-sync service from the constraints of the DVB-CSS media sync solution by adopting a new sync algorithm that is closely inspired from DVB-CSS but works in more distributed manner.

Due to the looser sync accuracy requirement for inter-home sync and the longer-term nature of the cloud-sync development work, a simplified synchronisation solution for the 'Theatre at Home' service prototype was proposed. The longer-term cloud-sync solution (the Timeline Synchronisation Service described in D2.2) is currently under development and a first prototype is planned for release during the first quarter of 2017.

The simplified solution is based on using the WallClock service and the Shared-State Service to enable the synchronised start of the DMApp. Its operation is outlined in the following sub-sections.

4.7.1 Anchoring the DMApp Timeline

When a DMApp starts as a result of a DMApp component kicking off the experience at one of the client devices (e.g. the TV starts to play a video), the Timeline Service is made aware of the exact WallClock time when the first DMApp component started via a CLOCK_CHANGED event. This allows the Timeline Service to anchor the timeline for the current DMApp (experience) to a point in WallClock time: this is the time at which the DMApp timeline starts. After having received this time value, the Timeline Service instance responsible for the current DMApp can now schedule the loading/unloading of DMApp components as per the DMApp timeline description. Because of the shared WallClock time (accurate to approximately ±9ms), the various clients can now be instructed to start playing their DMApp components (scheduled to start at the same time in the experience).

4.7.2 Sharing Playback Timing Data via Shared-State Object

Whilst this timeline anchoring approach is sufficient to ensure the synchronised start of DMApp components, synchronising the playback of continuous media on various devices requires the exchange of additional timing information about media playback. This is because there are additional delays incurred at the client node when loading and playing a video stream such as fetching, buffering and decoding times. The start time of a DMApp video player component may not denote the actual time its video started playing.

To synchronise the playback of continuous media across different homes (context), the Shared-State service is used to share media playback progress timing information from one

context (the master) to another (the slave context). This timing information is in the form of a tuple (called the WallClock tuple) and contains the following data:

1. **sessionId** – the cross-context session for the current DMApp instance
2. **contextId** – the tuple originator's context identifier
3. **DMAppComponentId** – the identity of the DMApp component (optional)
4. **wallclockTime** – wallclock time when media progress was read
5. **mediaTime** – time on media timeline
6. **playbackStatus**: paused or play

The WallClock tuple is a per-session object created at the Shared State service and it is initialised by the first home joining the session. Each session is identified by a sessionId. This identifier is predetermined for each client as a result of the client joining a particular lobby (via the Lobby Service). Joining the session is achieved by writing to that object (in an atomic way) – e.g, adding contextId. This is done by the device that creates the context / loads DMApp in the first home.

Subsequent homes joining the session via the Lobby Service will receive a shared copy of the WallClock tuple via the Shared State service. This WallClock tuple object is unique for that session. Each device in the homes can then retrieve the WallClock tuple and use the **{wallclockTime, mediaTime}** pair of timestamps to synchronise their DMApp component(s).

4.7.3 Achieving Inter-Home Sync

The **{wallclockTime, mediaTime}** timing pair specifies a relationship between the common WallClock timeline and the master media timeline (the main video's timeline). For the Theatre-at-Home prototype, the main video in the subsequent homes need to be synchronised to the main video playing in the first home joining the session. The mediaTime timestamps coming from the first home can therefore be directly used as timing information to drive the playback of the same video in the other homes.

Based on elapsed WallClock time, the expected media time corresponding to the current position on the WallClock timeline can be then calculated.

The **dvbcss-clocks** library provides a **CorrelatedClock** object to represent the relationship between two timelines (the **{wallclockTime, mediaTime}** timing pair is called a correlation). At home receving the WallClock tuple object, the client device playing the main video instantiates a **CorrelatedClock** object to represent the master DMApp component's playback timing information and uses this clock object to drive the playback of its own video component.

5 Client Application

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

5.1 Overview

The definition of a DMApp has changed as the Theatre At Home trial implementation has evolved. Conceptually, a DMApp is still a single application which is distributed across multiple devices, but we now have a better understanding of what is reusable from episode to episode, genre to genre and how productions teams will author and deliver experiences.

Authoring a DMApp for a franchise involves the commissioning and development of:

1. A set of DMApp Components that can be reused by each scheduled episode
2. Custom layouts, glue code, styling and brand assets
3. Supplementary content
4. A reusable programme template describing the overall format of each episode

These ingredients constitute a DMApp, reused from one episode to the next. For example, each MotoGP race/episode needs a unique timeline instance document referencing object-based media for that particular race. For live races, there wouldn't be a pre-existing timeline document; but one might be recorded as the live production unfolds.

The key distinction is that the timeline document is not part of the DMApp – it is a media feed that the DMApp consumes, like a piece of video.

In order to clarify the definition of a DMApp further, it is useful to highlight the constituent parts from which an experience is constructed. The following table describes how these parts form the application stack which runs on the client device. In order to facilitate the re-use of key functionality which is common to all experiences, four different applications are provided, two of which are Single Page Web Applications (SPA).

The Cordova wrapper technology (https://cordova.apache.org/) has been used as the basis of the client application on companion devices. Further information is provided in Section 5.5.

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

5.2 Application Logic

Non-distributed web applications use a combination of JavaScript variables and events to implement client-side business logic. A typical example might be programming what happens when a user presses a button or the browser receives an XHR response.

In a distributed web application, changes to variables must be propagated between different browsers and events need to be system-wide. There are a great many frameworks that address these requirements, providing a home for the much-needed application ‘glue-code’. As explained in Section 4.4, 2-Immerse has chosen to adopt the MediaScape Shared State framework for this purpose. This section details how we are using it.

The Shared State service has been used to provide application developers with a multi-device data-binding scheme. This delivers a more declarative style of programming and has reduced the amount of bespoke application logic required. This scheme has insulated the DMApp Components from specifics of the shared-state service by providing a more abstract programming interface.

5.2.1 Scopes

State is logically divided into a number of different ‘scopes’. A unique scope name (resembling a file path) identifies each scope. Data can be shared between one or more components using a pre-agreed scope name, specified in the timeline document. This allows various groups of components to communicate state between one another.

5.2.2 Session Scopes

A session describes a group of layout contexts taking part in a shared, synchronized multiscreen experience. This is typically used for experiences that are synchronized between two or more homes. The session scope contains URLs for timeline and layout documents together with a wall clock tuple for synchronisation purposes. The session scope may also contain one or more lobbyIds and a list of layoutContextIds.

The scope name is a pre-determined identifier allocated during the on-boarding process, or hard-coded for trial purposes. The first layoutContext to join the session will initialise the shared state.

5.2.3 Global Scopes

A DMApp can store application data in the global scope allowing it to be shared by any number of different DMApp Components. A simple example might be a state governing the audio volume of the application. The shared state service maintains the authoritative volume level, which other DMApp Components read and write. Servers are a more reliable source of authority than client devices.

A single global scope is created per layoutContextId. Any DMApp Component can add, remove or change global state and these changes are automatically propagated to all subscribers. The name of the global scope is synthesized from the layoutContextId.

5.2.4 Group Scopes

A group scope provides a way for several related components to communicate with each other. A DMApp Component such as a control panel might mirror a subset of its attributes to a group scope. Another DMApp Component may then listen to changes to the values in this group scope. This provides some of the much-needed application ‘glue-code’ that would otherwise be implemented using a combination of JavaScript variables and event handlers in a non-distributed app.

Group scopes enable data-binding functionality to be constructed, permitting a more declarative style of programming.

For example, a control panel component could mirror attribute values such as paused, muted and volume to a group state. Then an HTML video element could listen to these value changes by subscribing to the group scope and adjust its volume accordingly.

The name of a group scope is synthesized from the layoutContextId and an arbitrary identifier string used to make the group unique. This is referred to as the *‘groupStateId’*.

5.2.5 User Scopes

The Shared State Service also has pre-defined scopes for storing user preferences on a per-DMApp or DMApp independent basis.

5.2.6 Keys, Values and State Mirroring

A scope stores a list of (key, value) pairs defined by the application and its components. Values can be any JSON value, but it is recommended that values are kept simple and atomic updates are performed using the client's transactional API.

For convenience, we have created an *UpgradeCustomElement()* function for mirroring the attributes of a custom HTML element to the shared state service. This functionality makes it possible to hook up DMApp control surface components to DMApp view components located on other devices.

The code deals with incoming parameter changes from the timeline service, which it sets as HTML attributes on the upgraded custom element. Changes to these attributes are detected using a DOM mutation observer and then automatically propagated to the shared-state service. Similarly, the code listens to changes in the shared state and updates the HTML attributes. The mutation observer is cleaned up when the DMApp Component is destroyed.

The timeline author must define a parameter called '*groupStateId*' that tells the components which group scope to share. This parameter can be omitted, in which case no state is uploaded to the shared-state service. The value of *groupStateId* is used to build a unique name for the scope into which the state is stored.

The *groupStateId* attribute is also attached to the HTML custom element as an attribute and can be changed at any time by the application or the timeline to bind a DMApp control surface to a different DMApp view component.

5.2.7 Data Bindings

The HTML attribute mirroring implementation also provides a mechanism for defining mappings between component attribute names and state data using evaluated expressions.

5.3 DMApp Component Interface

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

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

WebComponents is a web standard for encapsulating HTML and the associated CSS style and JavaScript logic into a single HTML tag. This can then be easily used without the internal implementation of the component, and the host page which uses it, needing to be aware of each other or unduly interfering with each other. WebComponents are used as the basis for all of the existing DMApp Components implemented so far.

The client-api framework includes a default implementation of all the required parts of the interface, such that DMApp Component authors only need to implement the parts which are relevant for their components, while still permitting overriding the default behaviour where necessary.

The DMApp Component interface includes control of the component's life-cycle, visibility, position, and related functionality, and provides utility interfaces for handling of clocks and time-based cueing, component parameters, shared state, and access to the owning client-api instance for non-component-specific functionality.

5.4 Styling

5.4.1 Requirements

We have identified the following styling requirements:

- Consistent visual appearance and behaviour of widgets across a DMApp's entire set of DMApp Components.
- Ability to re-skin DMApp Components based on the DMApp that instantiates them.
- Rapidly create and apply new themes/skins.
- Prevent certain DMApp Components from being re-skinned (e.g. in order to preserve brand guidelines)

5.4.2 Modular Style Sheets

The lazy-loading of DMApp Components introduces challenges for handling style efficiently (minimally). For example it's preferable to break monolithic style sheets down into individual fragments that DMApp Components load on demand, letting the browser de-duplicate rules.

The problem with a modular approach is that there is a greater chance of an inconsistent look and feel across the application, especially if some components have been imported from different authors. A component could for example define CSS rules that interfere with styling across the application. This may also lead to inconsistent styling on difference devices in the multiscreen experience.

5.4.3 CSS Vocabulary

What's needed is an agreed styling convention that the individual components follow; one that allows the appearance of every component to be overridden from a single location. Frameworks such as Bootstrap achieve this by defining a vocabulary of CSS class names that describe appearance and behaviour of web pages.

Bootstrap is a very well respected library of UI elements providing a set of re-skinnable widgets and was a pragmatic choice for the Theatre At Home trial. Note that the 2-Immerse platform doesn't tie us to using Bootstrap, we are just adopting its vocabulary of CSS class names to facilitate global re-skinning. Bootstrap does however provide styles and behaviours that we may use to implement responsive layouts.

We have elected not to use the shadow DOM due to its immaturity and the added complexity this introduces. Instead, the parent DMApp loads a global CSS theme and the styling rules are

automatically applied to child DMApp Components - assuming they have adopted the same CSS class names.

5.4.4 Component-scoped CSS Rules

In special cases where we don't want the global theme to affect a DMApp Component's appearance, the component can be styled separately - avoiding Bootstrap's class names. Alternatively, the component can override global CSS rules with local rules of higher specificity. Indeed a strategy we have adopted is to create a CSS namespace for each custom element that wraps component-specific style rules, preventing them from being overridden by global changes.

5.4.5 Theme Roller

Using a global style sheet in conjunction with per-component styles based around a well-defined CSS vocabulary has enabled us to use a Bootstrap theme roller to generate and apply a genre-specific theme for the Royal Shakespeare Company.

5.5 HbbTV Emulator and Adapters

The 2-Immerse architecture includes support for the protocols defined in HbbTV 2.0 for device discovery in the home network, called DIAL, and media synchronisation that is based on the DVB CSS specification.

As fully-implemented HbbTV 2.0 devices are not yet available and it can't be guaranteed that they will be for any of the trials, 2-Immerse decided to develop its own TV Emulator device which is based on the essential protocols from the HbbTV and DVB specifications. This also has the advantage that 2-Immerse is not constrained by the minimum requirements of HbbTV, e.g. support for more than one video decoder on a single device. In addition, 2-Immerse aims to demonstrate the 2-Immerse platform working with the first HbbTV 2.0 device prototypes provided by major TV manufacturers.

5.5.1 HbbTV TV Emulator

Being unable to use HbbTV devices off the shelf made it necessary to have implementations of both ends of the HbbTV protocols – i.e. for TV and companion. Fortunately, there are libraries available as open source at GitHub.

Fraunhofer FOKUS has created libraries for all the discovery and launch features of HbbTV as well as for the app to app communication feature.[1] For the Theatre at Home trial, 2-Immerse makes use of DIAL for the discovery of the TV Emulator, and the app to app communication feature to enable a companion to join the context of an DMApp instance running on the TV.

For DVB-CSS the BBC published libraries also on Github.[2] These are included in the DVB-CSS implementation used in the Theatre at Home trial on the device acting as the master.

[1] https://github.com/fraunhoferfokus/node-hbbtv

[2] https://github.com/bbc/pydvbcss

Both libraries provide so-called Polyfills, that are used if the underlying browser environment does not support an API natively. On the application side this means that the same API exists as if it would run on an HbbTV 2.0 device.

5.5.2 HbbTV Adapters

In principle, the actual operating system of a 2-Immerse client device does not matter as most of the client implementation runs in a browser environment. However, some of the chosen protocols from HbbTV 2.0, namely the discovery part of DIAL and the wall clock synchronisation of DVB CSS, are not directly available via standard HTML5 APIs.

Several frameworks exist that allow to deploy web based applications on all popular mobile platforms as Android and iOS. 2-Immerse has chosen the Cordova framework that builds native applications around a web application and also includes a plugin mechanism. The plugin mechanism is used to extend the browser environment for non-standard APIs.

Two plugins have been identified in the architecture, one for the HbbTV adoption of DIAL and the other for DVB CSS. For the Theatre at Home trial, the plugins for Android are integrated. The implementation for iOS is work in progress and will be used in addition in the next trials. On Android the DIAL implementation is based on the Fraunhofer FOKUS libraries on Github, the DVB CSS library was created within the project. For iOS both parts have been created within the project.

5.5.3 HbbTV Prototype

Parts of the HbbTV implementation for the Theatre in Home trial has been used as test and demo content with TV manufacturers for HbbTV 2.0 prototypes. The DIAL and DVB CSS plugins on Android have been tested and showcased with Samsung at IBC 2016. A short video clip of the demonstrator (early prototype), that shows the discovery of the TV, launch of an HbbTV app from the companion, and finally the media synchronisation between TV and companion, can be found at:

http://hbbtv-live.irt.de/hbbtv2/video/190902_Sync-Demo.mpg

6 Multi-Screen Experience (DMApp) Components

This section provides details of the specific capabilities of the DMApp Components designed and developed for the First Release, with notes on how they are being used within the Theatre at Home trial.

It should be noted that the scope of the first Theatre at Home trial is restricted to a single TV Emulator device, which presents content on the main household TV, and a single companion device, which presents additional content and allows user interaction with the experience.

6.1 DMApp Components available in the First Release

The table below provides a summary of all of the DMApp Components developed for the First Release, and hence for the Theatre at Home trial.

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 between devices and	In Theatre at Home, this component is used to present the theatre production on the TV emulator plus additional on-demand video content on the companion devices before and after the show.
Scrolling Text	This presents scrolling synchronised text, such as the script of a play. It will include synchronised buttons to show actor and other information.	The provision of a synchronized scrolling script on the TV emulator is a key feature of Theatre at Home.
Title Card	This presents an opening screen for the entire experience.	
Article	This can be used to present a range of additional content, including cast/creatives bios. Content is authored using a simple markdown format and the viewing position within the article can be synchronized between multiple instances of the component.	In Theatre at Home, Article components on both the TV Emulator and companion device are used to present a wide range of additional content. During the show, this is restricted to the companion only.
Article Controls	This enables the user to interact with content presented in the Article component.	For Theatre at Home, the control options provide the ability to scroll up and down within the Article, and is only available on the companion.
Image	This presents a static image on the screen.	In Theatre at Home, image content is shown on both the TV Emulator and companion device. An Image component is also used

		to notify users that the show will be starting imminently.
Text Chat	This enables text chat to be presented and displayed, including conversation history.	Text chat is a key feature of Theatre at Home and is available on both the TV Emulator and Companion at all times.
Text Chat Controls	This provides a UI for posting messages to the Text Chat component.	Text Chat Controls are only available on the companion device.
Video Chat	This enables multi-party audio/video chat between households (contexts). It co-ordinates the other Video Chat components and optionally presents video thumbnails which represent other locations in the call.	The initial Theatre at Home trial only involves pairs of households and so the video thumbnail presentation is not presented.
Video Chat View	This presents the remote video stream of currently active speaker in a Video Chat session and a picture-in-picture view of the local camera stream.	For Theatre at Home, Video Chat is only presented before and after the show and during the interval. The Video Chat View is always shown on the TV Emulator.
Video Chat Controls	This provides a control UI for a Video Chat session, including microphone and speaker level controls and an optional ‘push-to-talk’ button.	The Video Chat Controls are only available on the companion device.
Component Switcher	This key component provides a UI to enable different parts of the experience (and hence DMApp Components) to be selected. It is responsive to the device on which it is running and can be ‘collapsed’ so that it occupies minimal screen space when it is not needed.	A simplified Component Switcher has been implemented for Theatre at Home which allows content to be selected for presentation in the Image and Article components only. At this time, it is designed for use in a landscape aspect on a tablet device only.

6.2 DMApp Components in the near-term plan

In order to meet the timescales required for the Theatre at Home trial, a small number of lower-priority DMApp Components were not completed. The table below summarises these, some of which may be made available in the near term for evaluation by triallists.

Name	Description	Comments
Timed Text	This provides the ability to present text at specific timecodes, such as subtitle text (usually on one or two lines).	The Scrolling Text component has reduced the need for subtitles within the Theatre at Home trial.
Notification	This displays general purpose notifications, such as time until the performance starts, or when others have joined. The notification content could be text, graphics or audio, and could be animated.	As explained above, notifications have been implemented using an Image component in the first version of Theatre at Home, although this component would offer a more flexible and visually appealing solution.
Like Widget	This provides a user input to express a 'like' preference, and presents aggregated like data back to users.	This component would be presented on the companion device.

7 Production Tools

This section describes the production tools used for creating the Theatre at Home experience. The authoring process of this first production has been analysed and will direct further development of more advanced production tools.

Production of a multi-screen experience can be an iterative process. A production cycle involves design, preparation of resources, creation of the experience and evaluation of the result. In this section we describe this workflow and the tools used in each of these phases.

Figure 12: Examples of wireframes used to visualise the Theatre at Home multi-screen experience.

7.1 Design

The initial Theatre at Home use case has been described in the project plan. This initial concept was leading in creation of a document describing the experience in more detail, including priorities to certain aspects. This description served to prioritise the development of DMApp Components as presented in the previous section.

Using graphic design tools, wireframes such as those shown in Figure 12 were produced to visualise an impression of the multi-screen components. These wireframes and the list of essential and high priority components that evolved in turn was input to further streamline the concept of the experience.

This "final" concept was documented in a table specifying different phases of the experience, and for each phase which components should be active on which device, that is, showing up on the communal screen on a personal device or on both.

An essential aspect of the design of the multi-screen experience has been creating a list of all resources needed. These included videos of an introduction, of the play itself, and videos for promotion, pictures of the rehearsal and of individual actors, texts describing the synopsis of the play, the bios and other relevant information, and special files e.g., for timed text and subtitles.

7.2 Preparation

All resources have to be collected and uploaded to the server. For this we make use of either a shell script or FileZilla (https://filezilla-project.org/). However, first the resources may have to be converted to the appropriate formats.

To support MPEG-DASH (an adaptive bitrate streaming technique that enables high quality streaming of media) all video material has to be converted by breaking the content into a sequence of small HTTP-based file segments. In this way the content is made available at a variety of different bit rates.

All text that will be presented using the Article component (see previous section) has to be converted to Markdown (a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML). For this we used a tool called pandoc, see http://pandoc.org/.

For timed text to be presented by the Scroll-text component input has to be presented in a specific JSON format that specifies text, category and timing. This format can be created using a text timing editor developed for this purpose. The editor using this tool can insert timestamps into the document while watching the video (see Figure 13 below).

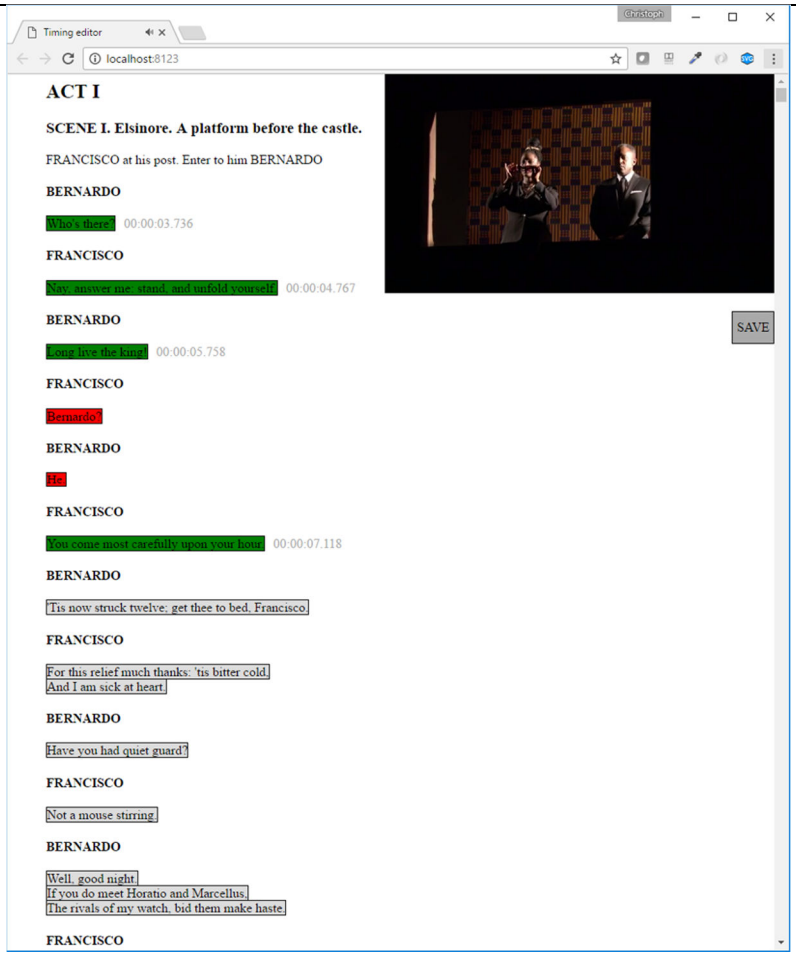


Figure 13: Editor for timed text annotations

Conventional image processing tools were needed to produce images of different resolutions, e.g. the Component Switcher needs thumbnails of the images of actors to be used as buttons to bring up information on that actor.

7.3 Creating the Experience

An experience is defined by its looks (the intricacies of styling has been described in section 5.4), by its temporal aspects (specified in a timeline document) and by its spatial aspects (specified by a layout document). Creating an experience involves creating a global style sheet (see 5.4.5) and timeline and layout documents and these documents need to be uploaded to the server.

7.3.1 Timeline Document

The temporal aspects of the experience (in essence: which DMApp Components are active at a given time) are specified by the Timeline Document. As explained in Section 2, the format of timeline documents is inspired by SMIL and is tailored to the 2-Immerse Timeline Service, which orchestrates media objects running on multiple devices.

Timeline documents have been created manually using an XML syntax editor (e.g. https://atom.io/).

7.3.2 Layout Document

As explained in Section 2, the Layout Document specifies for each DMApp Component within the DMApp the layout constraints that will be taken into account by the Layout Service whenever layout is evaluated.

The layout requirements document has a JSON format. This document has been created manually using a JSON-compliant syntax editor (https://atom.io/).

7.3.3 Validation

The syntax of the documents created can be easily checked by the editor tools, yet further validation (e.g. verifying the attributes of elements) is recommended. For this we developed several validation tools.

A timeline document can be validated with a tool running in a terminal window:

```
validate-timeline.sh --attributes --trace sample-dmapp/timeline.xml
```

The textual output of this validation tool includes errors, warnings, and the `--trace` option makes it report on activities of the Timeline Service that would be initiated at run time.

Similarly, the layout document can be validated by

```
validate-layout.sh sample-dmapp/layout.JSON
```

If you also want to verify that all DMApp Components that occur in the timeline document are mentioned in the layout specification you use:

```
validate-timeline.sh --attributes --trace --layout sample-dmapp/layout.JSON
sample-dmapp/timeline.xml
```

Optionally, documents can be run with the symbolic execution engine.

```
dryrun.sh --layoutRenderer –kibana --layoutDoc https://origin.2-
Immerse.advdev.tv/dmapps/sample-dmapp/layout.JSON
--timelineDoc
https://origin.2-Immerse.advdev.tv/dmapps/sample-dmapp/timeline.xml
```

This tool produces text output description of components becoming active. The optional parameter `` `--layoutRenderer` `` makes it open a browser window that will give you a live preview of where things will be rendered. The `` `--kibana` `` option opens a browser window that will show you all the output of the timeline and layout services for this run.

It is possible to use the `` `dry-run` `` script to test multiple devices at the same time.

7.4 Evaluation

For evaluation of the multi-screen experience we created different versions of the timeline documents, all of these sharing the same layout document.

1. We created timeline documents for each of the distinguishable phases of the Theatre at Home experience in order to be able to evaluate these individual phases on visual appearance, focusing on layout and styling. This visual inspection resulted in (re)placing some of the components.
2. We created a 1-minute and a 10-minute version of the full experience, running through each of the phases quickly. In this way we could evaluate transitions both technically

as well as on feel: the (set of) components becoming active and inactive. (This option also turned out to be very useful for debugging components and services).

3. A medium length experience (1-hour) was created to get a feel for the experience (this version was used in the so called “dress rehearsal”).
4. Finally, we have the full length experience to be evaluated on user satisfaction.

A future production tool should facilitate the functionality now obtained by these different versions of the timeline document.

8 Conclusion

This document has described the first release of the 2-Immerse Distributed Media Application Platform, Multi-Screen Experience Components and Production Tools that have been developed for the project's Theatre at Home service prototype. As the first instance of a working platform for the delivery of an interactive, object-based multi-screen experience, it forms a vital foundation for the subsequent prototypes which will be developed and taken to trial in the next 2 years of the project. 12 re-usable DMApp Components have been developed so far, and we expect the majority to be reused (and improved) in future iterations.

The project's second technology release will be delivered in just under a year's time and will document the evolution of this platform to support the MotoGP and Theatre in Schools trials. While the authoring process for the Theatre at Home DMApp has been largely manual, it has been carefully studied and will be assisted by the development of novel production tools during the coming year.