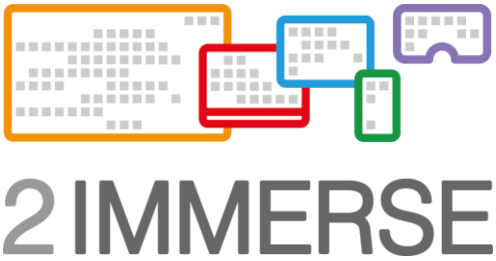
HORIZON 2020
The EU Framework Programme for Research and Innovation

Directorate General for Communications Networks, Content and Technology

Innovation Action

ICT-687655



D2.6 - Distributed Media Application Platform: Public Software Release

Due date of deliverable: 31 October 2018

Actual submission date: December 2018

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

Lead contractor for this deliverable: **BBC**

Version: 23 October 2018

Confidentiality status: **Public**

Abstract

This document first describes the public software release of the 2-IMMERSE Platform for the Development of Distributed Media Applications, Multi-Screen Experience Components and Production Tools. With these open-sourced software components, the community should be able to deploy their own 2Immerse platform and run similar distributed media applications.

This document further describes a reference architecture, called DMApp-RA, for distributed media application platforms. This abstract architecture is derived from our 2Immerse platform, and reflect the knowledge and insight obtained through our own platform-building experience in this domain. The reference architecture is shaped by our original platform architecture, the platform design and implementation choices, the chosen set of experiences, which represent a wide range of domains, and the technology and feature choices made implementing those experiences. The project parties believe this abstract architecture specification will be a useful guide for the community to generate architectural blueprints for their own Distributed Media Application platform, and defines a useful nomenclature for the domain at large.

Target audience

This is a public deliverable and could be read by anyone with an interest in the details of the platform, service prototypes and production tools being developed by the 2-IMMERSE project. As this is inherently technical in nature, we assume the audience is technically literate with a good grasp of television and Internet technologies in particular.

Impressum

Full project title: 2-IMMERSE

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.6 Distributed Media Application Platform: Public Software Release

Editors: Rajiv Ramdhany (BBC) and Mark Lomas (BBC)

Workpackage Leader: James Walker, Cisco

Technical Project Leader: Mark Lomas, BBC

Project Co-ordinator: Matthew Scarth, BBC

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

Executive Summary

2-Immerse is an open-source platform for the production, delivery and orchestration of distributed media applications. It is the first live end-to-end object-based broadcasting pipeline for synchronized multiscreen experiences, runs on any infrastructure, and follows industry best practices for scalability and resilience. It is a joint research effort between Cisco, BBC, BT, CWI, IRT, ChyronHego and Illuminations Media, and is partially funded by the EU "Horizon 2020" program.

The 2-Immerse platform provides a set of services, authoring tools and real-time production tools as a complete end-2-end suite that allows developers and users to build and control a multiscreen distributed experience temporally, spatially, and qualitatively. It also provides video synchronization services as well as discovery to determine what devices are participating in the experience.

A distributed media application is an experience that runs simultaneously on multiple devices such as TVs, termed "communal" devices, or tablets and phones, termed "personal" devices. As there are a multitude of devices with various capabilities, it's hard for developers to tailor their application to each type of device. 2-Immerse aims to makes this effort much easier by adopting the emerging concept of object-based broadcasting as defined by the HbbTV 2.0 standard. This approach allows developers to define their experiences as a set of objects with given requirements and constraints such as required device capabilities e.g. touch, sound, orientation etc., as well as screen layout definitions such as multiple/single instance(s), relative/absolute location, etc. This allows a layout engine to easily customize and adapt the experience. For example, we break down a football match broadcast into various objects such as the channel bug, score bar, the various video streams, tickers, etc., and can adapt the display to the current device orientation dynamically.

A subset of the final release software platform & components will be open-sourced with the Apache 2.0 so that external third parties can take advantage of the project's innovations to develop novel multi-screen experiences of their own.

This document describes a generalised platform architecture, called DMApp-RA, that may serve as a reference architecture for those wishing to build similar platforms.

The reference architecture was shaped by our original architecture, the platform design and implementation choices, the chosen set of experiences, which represent a wide range of domains, and the technology and feature choices made implementing those experiences.

List of Authors

Mark Lomas – BBC

Rajiv Ramdhany – BBC (Editor)

Contributors

Ian Kegel – BT

Jonathan Rennison - BT

Tal Maoz – Cisco

Aviva Vaknin - Cisco

Jack Jansen – CWI

Reviewers

Tal Maoz – Cisco

Aviva Vaknin - Cisco

Michael Probst - IRT

Table of contents

List of Tables and Figures

Glossary of terms

Term/acronym	*Definition/explanation*
Experience	2-IMMERSE developed, using its own platform, four innovative service prototypes of multi-screen entertainment 'experiences'. Unlike existing services, the content layout and compositions are orchestrated across a set of available screens using an object based broadcasting approach for dynamic, efficient, high quality, synchronized content distribution and rendering.
Distributed Media Application (DMApp)	2-IMMERSE multi-screen entertainment experiences are composed of many applications configured to work together to deliver the look and feel of a single application. 2-IMMERSE calls this collection a Distributed Media Application, or DMApp.
Distributed Media Application (DMApp) Component	In 2-IMMERSE, re-usable components are assembled within a Distributed Media Application (DMApp) to create coherent multi-screen experiences. DMApp components are typically media rendering objects created as part of the experience production and/or GUI components that support user interaction and/or implement application logic.
CI/CD	Continuous Integration and Continuous Delivery
Context	2-IMMERSE defines a 'context' as one or more connected devices collaborating together to present a media experience. Each context has a 'contextID' unique to its session. Multiple contexts may be active on a single LAN (e.g. a home network. Devices belonging to the same context must be able to discover each other using the DIAL protocol. Devices can join or leave a context at any time.
IPTV - Internet Protocol television	IPTV is a major concept within the 2-Immerse platform as it allows the delivery of next generation experience levels using object-based approaches. However, we made sure that our client stack is compatible with traditional linear broadcast using the HbbTV2.0 standard.

1 Introduction

This deliverable report provides a description of the software to be released publicly by the project to facilitate reuse of these code contributions; it also documents a reference architecture for distributed media applications obtained from a generalisation of the 2Immerse platform.

The 2-IMMERSE open source software represents three years' worth of continued incremental development and embodies the platform functions to run all five diverse multi-screen experiences created by the 2-IMMERSE consortium. It is offered to the community to accelerate further research and development and to enable SMEs, developers and other institutions in the media industry to build new multi-screen experiences quickly, especially for HbbTV2.0 devices, to run at scale. The open-source code provides experimenters with the capability to instantiate their own 2Immerse Distributed Media Application, hereafter DMApp, platforms and deploy similar DMApp-types. The platform was developed with capabilities that address particular use-cases in performing arts at home, live football, football in a social setting, motor track racing and theatre/drama teaching in schools. But because the development process was incremental and the set of use-cases diverse, it can be reused to build and run DMApps in other application domains. Some of the design choices and technology selection may not be an exact fit to other uses; however, the platform is in itself highly modular and it can be seen as a useful collection of reusable parts.

Describing which functions these reusable parts embody and how they fit into a system of systems is one of motivations behind eliciting a reference architecture. The DMApp Reference Architecture provides a more abstract template of an architecture for platforms that enable multi-device media-object orchestration. The abstract architecture is general enough to be free from implementation choices but provides a description in sufficient detail of platform functions to allow platform developers to *i)* understand the functionality required to orchestrate such experiences and *ii)* make their own technology/algorithm selections for these functions as per their own application-domain requirements. Further, the Distributed Media Application field is an emergent domain that over the last five years has seen a proliferation of frameworks such as MPEG MORE (4), specifications such as W3C Presentation API (5) and DVB-CSS (6), prototypes such as Vostok-K (7), Cisco's F resco (8) project, domain-specific languages including SMIL 3.0 (9) and BBC's Object-Based Toolkit (10) and more recently platforms such as the European project MediaScape (11) and 2Immerse (12) for orchestrated media multi-device experiences. Some of the prototypes can be viewed as specialised solutions for dedicated business opportunities without implementing generally applicable concepts whilst the range of applicability for the specifications may be limited.

To contend with this heterogeneity in DMApp solutions, we believe in the necessity of a common "lingua franca" i.e. a common understanding of the functions required, for the quick and pervasive development of innovative DMApp solutions in diverse application domains. Our reference architecture (called **DMApp-RA**) establishes this common ground by identifying a minimal set of unifying concepts, abstractions and their inter-relationships for DMApps. The reference architecture elicitation definition process described in this document follows a rigorous approach where the generalised architecture is shaped by successively analysing functions from representative platforms and generalising them. The methodology prescribes multiple iterations until a saturation point is reached. The reference architecture development process is exhaustive, and, in this document, it has been limited to the set of 2Immerse platform instances. The future maturation of the DMApp-RA is intended to be released separately as a useful specification and guide for the community. In essence, the DMApp-RA provides a good template for instantiating concrete DMApp system architectures in a particular application domain, leaving the implementation choices and arising compromises to the RA-adopters.

As a consequence of our generalisation process, the 2Immerse platform can be seen as an instantiation of the DMApp-RA and the public software code repositories are reference or sample

implementations of the DMApp-RA functions. The general description of these functions and the discussion of implementation alternatives included therein provides a useful guide to how these released software components fit into the overall platform.

The rest of this deliverable is structured as follows:

- **Section 2: Public software release** – this section described the software component released, the location of the code repositories and the terms under which they have been made public. It also identifies the software component community manager; the organisation responsible for the software component beyond the completion of the project.
- **Section 3: Reference Architecture for 2Immerse** - this section first explains the methodology used to derive a reference architecture. It then describes the application of the methodology to the DMApp domain by documenting the iterative analysis of 2Immerse platform instances. It presents the artifacts generated from the first and last iterations. Lastly, it presents an abstract system architecture view resulting from the analysis.
- **Section 4: Conclusion** – this section provides a summary of our contributions in this report and our recommendations for future work.

2 Public Software Release

2.1 Introduction

At IBC 2018, the members of the 2-IMMERSE consortium publicly announced that the object-based broadcasting platform it had developed, funded by the Horizon 2020 Framework Programme of the European Union, would be released as open source:

> *"2-IMMERSE is an EU co-funded innovation project which has developed and is launching a new open-source platform for Object Based Multiscreen Entertainment. The open-source platform is based on reusable components that will accelerate the development of new immersive multi-screen experiences, encourage the take-up of the HbbTV 2 specification and contribute towards its evolution."*
>
> (See section **Error! Reference source not found.** in Appendix for IBC Flyer)

One of the key reasons for publishing the 2-IMMERSE software under and open source licence is to ensure that the software we use today will be available and continue to be improved and supported in the future. We are providing the 2-IMMERSE software to the community as a starting point for further investigation into multi-device experiences. It is a sample implementation of the 2-IMMERSE multi-screen service reference architecture and may or may not be leveraged directly as-is by third parties. The individual component repositories can be seen as sample-implementations of the functions that a particular DMApp platform would need.

This section documents the open source software release by providing full details of the published software.

2.2 Open Source Software Description

2.2.1 What have we published?

The 2-Immerse core platform is comprised of 23 software repositories, including orchestration services, a client library and its dependencies. A further selection of supplementary repositories providing cloud-based media synchronisation and production tools are included, together with repositories containing documentation, tutorials, sample code and scripts for deploying instances of the platform.

All 2-Immerse software modules make use of publicly-available dependencies using the usual mechanism for the programming language in which they are written e.g. NPM for JavaScript, PyPI for Python, etc. The dependencies and their recursive dependencies are not included in the repositories themselves and are not themselves included in any open-sourcing process, however the license levels have been validated to have permissive open source licenses themselves.

The software repositories are listed in the table below together with the organisation nominated as community manager.

Repository	Description	Community Manager	Contributors
client-api	Client device framework	BT	BT, BBC, IRT, CWI
timeline-service	Service to orchestrate timing and events	CWI	CWI, BT, CISCO, BBC
layout-service	Service to orchestrate layout of content over a set of display devices in a multi-screen experience.	CISCO	CISCO
synckit	A JS library API to enable synchronisation of media as directed by a Synchronisation Service (a client-api dependency)	BBC	BT, BBC, IRT
hbbtv-lib	A library of modules for common HbbTV functionality	IRT	IRT, CH
websocket-service	Service to support service->service, service->client, and client-> client push communications	CISCO	CISCO, BT, CWI, BBC
dvbcsstv-lib	The JS library for the dvbcss browser proxy (client-api dependency)	BBC	BBC, BT, IRT
2-immerse	Documentation, tutorials, quick start guide, hosting instructions.	BBC	CISCO
auth-service	Authentication service	CISCO	CISCO, BT
auth-admin	Admin interface for the auth-service	CISCO	CISCO, BT
bandwidth-orchestration	Component Bandwidth Orchestration Service.	CISCO	CISCO
logging-service	A lightweight service which flattens a JSON structure sent via HTTP POST and pushes it to stdout, where it is collected and sent to Logstash and the rest of the ELK stack.	CISCO	CISCO, BT
wallclock-service	A lightweight time (WallClock) synchronisation service for frame-accurate synchronised experiences.	BBC	BBC, CISCO,IRT
shared-state-service	This is a fork of the MediaScape shared state service, licensed under the Apache License, Version 2.0.		CISCO, BT

shared-state-client	Client code from the Mediascape SharedState repository organised as a npm package.	BT	BT, BBC
oipf-object-polyfill	Polyfill to implement the oipfObjectFactory and allow implementations of objects to be created.	BBC	BBC
cloud-sync	Media Synchronisation Service and JS client library. Service comprises of a set of microservices. A demo-app hosting microservice is also provided.	BBC	IRT, BBC
2immerse-editor	Platform for creating 2IMMERSE presentations in the browser	CWI	CWI, BT
renderer	Realtime multi-device layout visualisation tool for the layout service	CISCO	CISCO
bandwidth-orchestration-client	Client for the bandwidth orchestration service. This client contains the SANDPlayer module that manages a Dash.JS player and send statistics to the BOS.	CISCO	CISCO
android-unified-launcher	Cordova applications to wrap unified-launcher web application as a native app for Android.	BBC	BBC, BT
system-images	HbbTV2.0 emulator firmware for the Intel NUC	BBC	BBC
launcher	Multi-experience launcher web application for TVs and companions scoping user enrolment, device discovery, device association, multi-device authentication, session discovery and session creation.	BBC	BBC, BT

Table 1 Software repositories to be published.

2.2.2 What have we not published?

- Media content or software repositories related to the 2-IMMERSE service trials (MotoGP at Home, Football at Home, Football FanZone, Theatre in School or Theatre at Home) due to content rights limitations.
- Rancher/AWS infrastructure components used to host 2-IMMERSE specific platform deployments, although alternatives templates are provided to enable adopters to host test instances of the 2-IMMERSE software platform on their choice of infrastructure.

2.2.3 Where is the software published?

All software has been published to GitHub under the 2-IMMERSE organisation[1] as publicly visible repositories.

2.2.4 Who will be maintaining and managing the software?

The nominated community managers are responsible for supporting their repositories, helping to set their future direction of development and gating any contributions. Community managers for a given repository have generally been nominated based on whomever first contributed to the repository and created it. That said, if another consortium partner has become the majority contributor or effective maintainer, then they have been nominated as the community manager.

2.2.5 Licence Type

All 2-IMMERSE software is published under the Apache Licence v2.0 that was used for the 2-IMMERSE dvbcss-clocks and dvbcss-protocols software repositories, published earlier in the project. Apache Licence v2.0 was chosen because it is permissive. It's summarised as:

> *"A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code."*

Apache Licence 2.0 is officially recognised by both the Open Source Initiative (OSI)[2] and the Free Software Foundation (FSF)[3] and is widely used. Apache Licence v2.0 was selected based on satisfying the desired terms below:

Is the licence recognised as a common Free and Open Source licences?	**Yes**
Who is permitted to examine the human-readable source code of the software?	**Anyone**
Who has permission to run the software?	**Anyone**
Who is permitted to adapt or modify the source code of the software?	**All licencees**
Who is permitted to redistribute the modified or unmodified source code of the software?	**All licencees**
Does the licence permit sub-licensing of rights?	**Yes, Unconditionally**

Table 2 Desired open source terms

2.2.6 When will it be available?

The software will be available publicly on GitHub end of January 2019.

[1] https://github.com/2-IMMERSE

[2] https://opensource.org/

[3] https://www.fsf.org/

3 A Reference Architecture for DMApp Platforms

The 2-Immerse platform was designed to support a wide variety of functionality and domains as demonstrated in the experiences, and their success validates the architecture. For this reason, the platform (through its incremental convolutions) is an excellent candidate for deriving a reference architecture for future distributed media application platforms. Reference architecture formulation requires the analysis of more than one platform. We followed an iterative methodology for deriving the reference architecture from 2Immerse platform instances but for further maturation of the reference architecture, we plan to extend our analysis to more DMApp systems and frameworks and disseminate the results to the community as a public document or publication (as part of the project results dissemination activity). A mature reference architecture is a useful guide for the community in understanding and building this type of systems. In this section, we provide the reference architecture derivation methodology and present its results.

3.1 The need for a Common Ground for the DMApp Concept

'Distributed Media Applications' is an umbrella term for interconnected technologies, devices, objects and services utilised in the provision of multi-device object-based media experiences for audiences. After many years of research and discussion, there is still no clear and common definition of the concept. The concept of DMApp emerged primarily from the convergence of different technological developments and fields. It builds on the emergence of innovative enabling functionalities that stem from object-based media, adaptive media streaming, web-application environments on hybrid televisions, standardised companion-screen discovery and media-synchronisation functions in HbbTV 2.0, as well as from the availability of mobile and IoT devices that can be co-opted into experiences. Although this is still an emergent domain, the last five years has seen an accelerated growth in the development of systems frameworks, specifications, prototypes, domain-specific languages and more recently platforms such as 2Immerse for orchestrated media multi-device experiences. This heterogeneity can be exemplified by considering a few DMApp experiences, including the ones in the 2Immerse project, developed for different application domains.

DMApp	Live	On-Demand	Pre-determined Timing	Non-deterministic Timing	Single-source	Multi-source	Single-context	Multi-context	Single-device	Multi-device	Master stream	No master stream	Interactive & user-driven
2IMMERSE *Theatre-at-Home*		✓	✓		✓			✓ (IDMS)		✓		✓	
2IMMERSE *MotoGP*		✓	✓			✓	✓			✓	✓		✓
2IMMERSE *Football Live FA Cup*	✓			✓		✓		✓		✓	✓		
2IMMERSE *Football Fanzone*		✓	✓		✓		✓			✓	✓		
2IMMERSE *Theatre-at-School*		✓		✓	✓		✓			✓		✓	✓

Table 3: Heterogeneity and variations across various DMApps

A quick analysis of the 2Immerse DMApps reveals several dimensions of variability, as shown in Table 3. The analysis of the use cases also reveal a good distribution in terms of variations in the DMApp concept.

Based on the diverse nature of DMApps and the resulting heterogeneity in DMApp platforms, we suggest that a common "lingua franca" for the DMApp domain, much like the thin-waist of the Internet protocol suite, is needed as a focal point for quick and pervasive development of innovative

solutions that can leverage different technologies developed for different application domains. Based on our experience of building diverse DMApps within the 2Immerse projects, specifically performing arts broadcasting, performing arts teaching, football, and motor track racing, we believe the consortium is well-positioned to lay the foundation for the much-needed common ground or a common DMApp platform architecture.

3.2 Reference Architecture for the Emergent DMApp Domain (DMApp-RA)

The identification of a *Reference Architecture for the DMApp domain* (hereafter, termed **DMApp-RA**) provides this common ground. By reference architecture, we refer to an abstract framework that comprises a minimal set of unifying concepts, abstractions and relationships for understanding significant relationships between the entities of an environment. Reference architectures are particularly useful in rich emergent domains as they provide a template solution for an architecture of a platform for a set of application domains. In particular, they identify core functions, structures and their respective elements, and the interaction between these elements. At this level of abstraction, they are independent of specific standards, technologies, implementations, or other concrete details. Thus, they provide a good template for instantiating concrete architectures in a particular application domain, leaving implementation choices and the arising compromises to the RA-adopters.

> *"According to RUP (Rational Unified Process), a Reference Architecture: …is, in essence, a predefined architectural pattern, or set of patterns, possible partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artefacts to enable their use. Often, these artefacts are harvested from previous projects."* (13)

The 2Immerse platform instances were developed and refined to solve the need of our use-cases in a general way so that the platform was reusable from one use-case to another; by extension, the platform may be applicable to other use-cases in the distributed multi-media application space. The caveat is that the platform was built and refined to fulfil specific sets of requirements and may reflect some design decisions, implementation compromises and technology choices that support those specific features for our experiences. Thus the 2Immerse platform is a good starting point for deriving a reference architecture and can be considered as one of its possible concrete instantiations. Businesses who want to create their own compliant DMApp systems would benefit from a Reference Architecture that describes the essential building blocks as well as design choices to deal with conflicting requirements regarding functionality, performance, deployment and security. Interfaces should be standardised, best practices in terms of functionality and information usage need to be provided.

Our goal, therefore, is to first derive a reference architecture from the 2Immerse platform used to develop and deploy 5 specific instantiations of the 2Immerse platform:

1) 2Immerse Theatre-at-Home
2) 2Immerse MotoGP
3) 2Immerse Live Football
4) 2Immerse Football Fanzone
5) 2Immerse Theatre-at-school

The DMApp Reference Architecture allows us to describe abstractly the 2-Immerse platform free of functional-duplication, technical debt and other design compromises that have naturally arisen as a result of resource constraints and time pressure.

In this document, we provide a taxonomy of concepts in the domain and build a lexicon of terminology that are crucial to establishing a common understanding of DMApp platforms across different application domains. The individual functions associated to the main concepts[4] are identified. A significant number of these functions are available as self-contained decoupled libraries, frameworks and packages in our implementation.

We have taken into account and integrated some concepts and standards from MPEG MORE and MediaScape in our platform design. An outcome of this is that generalisations of the architecture to accommodate variations in concepts and terminology will not result in drastic changes to the Reference Architecture. Concepts such as 'Source', 'Sink' and 'Node' are used in MPEG-MORE to broaden the scope of what can be orchestrated using DVB-CSS timing. In DMApp-RA, we make use of similar terms to refer to these particular concepts.

3.3 DMApp-RA Elicitation Methodology

We have chosen a reference architecture derivation approach that is concerned with identifying core domain functions and the variabilities of these functions across exemplar platforms. Due to the heterogeneity exhibited in our five DMApp implementations, we are confident that the 2Immerse DMApps provide a sound basis for the shaping of our platform and thus the reference architecture. This will reduce the number of iterations required in our methodology to identify all possible variations in DMApp-platform functionality.

The reference architecture elicitation approach is based on deriving **four models/views** of the platform based on an analysis of exemplar systems:

i) the **Domain Model**
ii) the **Functional Model**
iii) the **Functional** & **Non-Functional View**
iv) the **System Architecture View**

The purpose of the views is briefly summarised in the following sub-sections. The different views focus on different aspects of the platform thereby contributing individually to the quality of the reference architecture; they are complementary and enable the reference architecture extractor to fix key abstractions used in the architecture that cover all four views. This is a simplified derivation of the different sub-models in the Reference Architecture; other models could include: Information Model, Communication Model, Trust-Security-Privacy.

The elicitation process and relationships between the different views are illustrated in Figure 1.

[4] The reference architecture should be free from specific implementation choices, such as Redis, MongoDB or Polymer. These choices are already documented in our concrete "implementation architecture" and should only be referred to as **examples** when it helps to clarify aspects of the reference architecture.

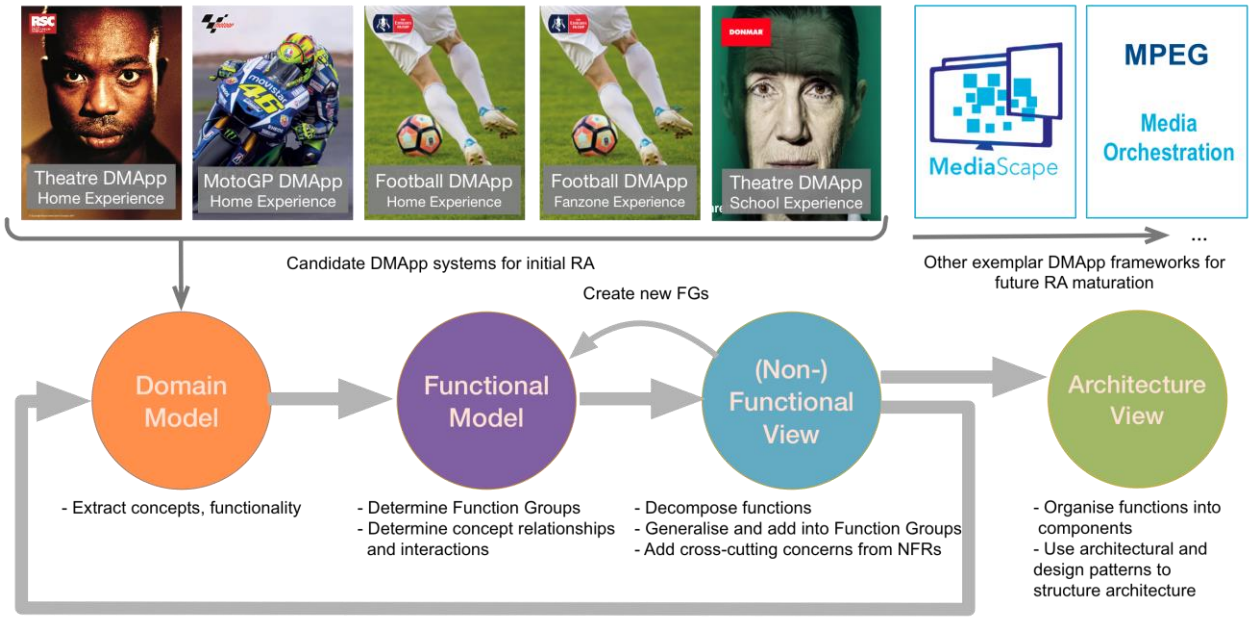


Figure 1: DMApp Reference Architecture elicitation methodology

3.3.1 The Domain Model

The Domain Model is a description of main abstract concepts belonging to a particular area of interest. The domain model also defines basic attributes of these concepts, such as name and identifier, as well as relationships between concepts. By including a definition for each identified concept, the domain model provides a common lexicon and taxonomy of the Distributed Media Application DMApp concepts.

The first step in the methodology involves examining the DMApp system to identify its main concepts, and the associations and relationships between these concepts (e.g. *is-a, has-a, has-many, uses*). Successive passes through this step in future iterations may require a generalisation or redefinition of some of the concepts.

3.3.2 The Functional Model – coarse functional decomposition

In deriving the functional model and view of DMApp-RA, functional decomposition is used as a method to break up the complexity of 2Immerse and other representative platforms in the DMApp domain, into smaller and more manageable parts, and to understand and illustrate their inter-relationships.

The output of the functional decomposition process produces functional descriptions at two levels of abstraction:

- The Functional Model
- The Functional View

The Functional Model is an abstract framework for understanding the main concepts in the platform for running DMApps, and the interactions between these groups. The model exists at a coarse-level of abstraction in that it identifies the main functionality groups, most of which are grounded in key

concepts of the DMApp Domain Model. The main concepts identified in the previous step are imported into this phase of the process and mapped onto existing Functionality Groups or converted into new ones if no overlap exists with current Functionality Groups (hereafter, FGs).

The purpose of the Functional View and the method to build it is described in the next section.

3.3.3 The Functional View – fine functional decomposition

The Functional View describes a finer level of abstraction, the system's functional components, including the component's responsibilities, their default functions, variations in their functions and primary interactions. The Functional View is developed by starting with the Functional Model and expanding the Functional Groups described therein. In succeeding iterations, the following sequence of steps is used to enrich and generalise the Functional View:

1. Identify functions associated with new concepts in the new platform, generalise them and group them into Functionality Groups
2. Determine if functionality bundled together with other functions belong to the same group or can be partitioned into another group
3. Identify commonalities and variabilities in the functions.
4. Capture the way the functionality is accessed by or itself invokes other functions.

3.3.4 The Non-Functional View

The Non-Functional View is concerned with aspects such as performance, resource usage, ease-of-use and extensibility.

This perspective of the reference architecture encourages platform-designers to explicitly propose abstractions and mechanisms for the provision of the non-functional behaviour.

In this way, the platform designers avoid the pitfall of fixing key non-functional aspects of the framework to default sub-standard solutions, which could constitute immutable decisions in the design of their platform. Wherever the non-functional behaviour depends on the platform deployment environment and different strategies can be applied to yield particular benefits to the platform user or hosted services, we can make the non-functional behaviour a variability in the architecture.

In the Non-Functional View, we consider aspects of the platform that influence non-functional properties. For each non-functional property, we decide on whether some additional platform functions are needed and how these functions are added to the Functional View.

3.3.4.1 Scalability

Upon examining platform throughput, we can add to our reference model the capability to handle 1000+ requests by using some pattern to do horizontal scaling. We can achieve concurrency through multiple processes by using a deployment pattern such as a service instance per container so that it is straightforward to scale up and down a service by changing the number of container instances.

The container encapsulates the details of the technology used to build the service. All services are, for example, started and stopped in exactly the same way.

3.3.4.2 Ease of use: server-side and client-side service discovery

Platform services have to able to call each other and invoke functionality via APIs. It is implausible to expect services to run at specific locations, e.g. hosts and ports, especially if a microservice pattern is enforced throughout the platform architecture. Consequently, platform builders must implement a mechanism that enables the clients of a service to make requests to a dynamically changing set of ephemeral service instances.

The pattern we can use in this case is server-side discovery (14). When making a request to a service, the client makes a request via a service router (e.g. a load balancer) that runs at a well-known location. The router queries a service registry and forwards the request to an available service instance.

3.3.4.3 Cross-Cutting Concerns

There are a number of functions that are related more to the ease of getting platform services up and running and monitoring the health of these services.

Examples of these cross-cutting concerns include:

- Externalised configuration - includes credentials, and network locations of external services such as databases and message brokers
- Logging - configuration of a logging framework such as log4j or logback
- Health checks - a URL that a monitoring service can "ping" to determine the health of the application
- Metrics - measurements that provide insight into what the application is doing and how it is performing
- Distributed tracing - instrument services with code that assigns each external request a unique identifier that is passed between services.
- Timeliness/responsiveness of system
- Elasticity (cf. Reactive Manifesto)
- Security considerations
- Degree of automation
- Life cycle concerns

Cross-cutting concerns may result in additional services/functions added to our functional view.

3.3.5 The Architecture View

The DMApp-RA Architecture View focuses on the structural aspects of the platform, as well as the interactions that take place within the system. It describes how the general functions defined in the functional view are organised into architectural abstractions and how these functions are accessed.

The procedure to generate the Architecture View is summarised as follows:

1. **Organize the functions from the Functional View into architectural abstractions.**

Design decisions that have been made in the exemplar platform services due to performance consideration, e.g. IPC vs. RPC, are not reflected in the Architecture View, while design patterns that provide clear benefits are adopted in the architecture.

High cohesion and loose coupling are desirable traits in Architecture View of the reference model, thus RA-builders strive to maintain a strong separation of concerns to ensure the platform services are independent. Decoupled functionality is more easily replaced by alternative solutions to fit the specific design constraints of the target platform.

2. **Determine how the functions will interact and how the functions will be accessed by other components.**

Possible function interaction implementation options include synchronous invocations, asynchronous request-reply, asynchronous events and unicast/multicast messages. Performance and flexibility are factors in this decision and achieved with patterns such as RMI, pub-sub, and shared-state. Some interaction patterns, such as group communication, can be fixed in our reference architecture if they provide very clear benefits, such as promoting loose coupling between services.

For example, asynchronous pub-sub communication patterns make it easier to integrate unanticipated functions and their variations.

The interaction mechanism chosen may cause side effects such as the inability to maintain strong consistency guarantees at all times in the system. The mechanism may be chosen to be utilized regardless if these side-effects are short-lived and/or do not impact overall functionality.

Another aspect of the Architecture View is the specification of interfaces and protocols required to invoke the functions. Languages such as RAML, IDL, and Google Protocol Buffers are useful for specifying interface/protocol.

3.4 DMApp-RA Domain Model

The main purpose of a domain model is to generate a common understanding of the target domain in question. With a common notion of the main concepts is it possible to argue about architectural solutions and to evaluate them. This domain model extraction therefore fixes the nomenclature of the domain by providing a taxonomy of concepts, and includes a definition of the main abstract concepts, their responsibilities, and their relationships. This section describes the abstract concepts relevant for DMApps and selects the key concepts to feature in the architecture.

3.4.1 DMApp Domain Model Concepts

3.4.1.1 DMApp Concept

A DMApp is the actual user experience. Variations of the DMApp concept include:

1) Live DMApps
2) On-demand or offline DMApps
3) Single-sender single-context single-device DMApp
4) Multi-sender single-context single-device DMApp
5) Single-sender single-context multi-device DMApp (master stream, no master stream)
6) Multi-sender single-context multi-device DMApp (master stream, no master stream)
7) Single sender multi-context multi-device DMApp (master stream, no master stream)
8) Multi-sender multi-context multi-device DMApp (master stream, no master stream)
9) Single-sender single-context multi-device dynamic-user-driven DMApp

A DMApp description specifies a set of DMAppComponents that will be flexibly presented on particular devices fulfilling prescribed roles based on state changes in the DMApp (e.g. DMApp chapters, progress of time, devices joining/leaving, etc.). It comprises of the following

- A description of **DeviceRoles** and DMAppComponents associated with them
- The list of devices in the **Context** (logical grouping of devices) fulfilling particular **DeviceRoles**
- A **LayoutDocument** specifying the presentation parameters of DMAppComponents such as device capabilites, minimum and maximum size, positioning anchors, priority level, target regions, volume-level, etc for participating Devices in the Context.
- A **TimelineDocument** that specifies the expected lifecycle and presentation state of the DMAppComponents on devices in response to DMApp state changes including DMApp chapter definition, DMApp current time, DMAppComponent loaded, ready, and end of presentation states.

- DMAppComponent state items to be reported and acted upon e.g. current time/speed changes and component status changes.

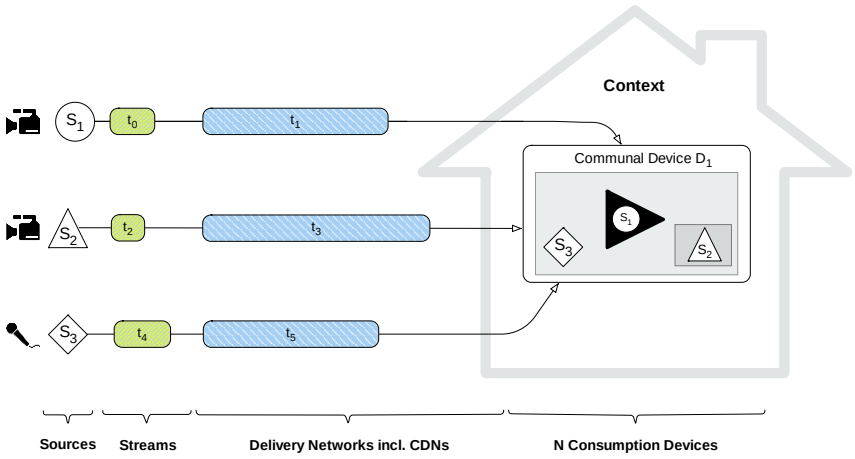


Figure 2: Single-device single-context live DMapp

3.4.1.2 DMAppComponent Concept

A **DMAppComponent** is a UI component or reusable widget that can be deployed, loaded and its playback controlled by a **DMAppRuntime** on a Device.

A DMAppComponent is a generalisation of the **MediaDataSink**, **TimedDataSink**, **DataSink** and **MediaDataSource** concepts**.**

A **DMAppComponent** has component ID, component class, component URL config parameters and state such as **LifeCycleStatus** which takes on the values { *initialized*, *started*, *finished, currentTime* }

3.4.1.3 Device-Role Concept

A **DeviceRole** defines the role the device takes on in an experience, the DMAppComponents displayed on each device specifies constraints per role. Roles include communal, personal, primary/secondary, and additionally may be defined as required or optional. [5]

A **DeviceRole** is fulfilled by a Device based on its **Capabilities** (see Section 3.4.1.4). Sample device roles include:

- Main screen role taken by a communal TV with largest screen-size, left-speaker role taken by secondary communal device, and right-speaker role taken by another secondary communal device
- Teacher companion or student companion device roles fulfilled by tablets
- Pub-landlord's control device role fulfilled by his smartphone

A **DMApp** needs all '*required*' **DeviceRoles** to be fulfilled for it to run.

[5] The sets of DMAppComponents for DeviceRoles in a DMApp are not mutually exclusive; a DMAppComponent can be deployable on two devices even if each device fulfills a different role. E.g. a stats DMAppComponent can be shown on a companion screen preferentially (higher priority), but allowed to be moved to a main screen if more screen space is needed later. Or, the stats DMappComponent can be shown on both main and companion screens (equal priority).

3.4.1.4 Device Concept

A **Device** is a generalisation of one of the following device type:

1. **communal,** for example, TVs, large screens, surround speakers,
2. **personal,** for instance tablets, phones, smartwatches, and wearables or
3. **shared-personal,** such as pass-around tablets and phones.

A **Device** has several capabilities associated with it that must be defined in the 2Immerse context; examples of capabilities include screen size and resolution, number of video and audio channels, touch interaction, etc. **Devices** are discoverable by other context devices through a chosen discovery mechanism such as DIAL, mDNS or Bonjour, on the local network.

Each **Device** has a status associated with it that indicates if it has been paired with a user account or on-boarded into a DMApp. Status can include one or more of the following:

- *unpaired*
- *paired* - a device that has been initialised with an access token and paired with a user account and may access the 2Immerse platform services
- *onboarded*
- *not-onboarded*
- …

A **Device** has a **SyncMode** to specify its synchronisation-mode in the Context. The **SyncMode** can be one of the following: {*Master*, *Slave*, *None*}. A master device provides timing to slave devices for synchronisation. A None SyncMode implies equal synchronisation status among the devices; the synchronisation timing is either achieved by reconciling the timing from all devices or by an external entity injecting timing into the DMApp.

3.4.1.5 User Concept

One or more **Devices** are paired to a **User** account.

A **User** launches a **DMApp** from a **Personal Device** and then on-boards Communal devices into the DMApp. Alternatively, the starting device can be a Communal Device.

A **user** interacts with **DMAppComponents** on devices with interaction capabilities, usually **personal** devices.

3.4.1.6 Context Concept

Starting the **DMApp** on a required device establishes a context, which can then be shared with other onboarded devices.

A **context** is a group of devices in a physical environment participating in a multiscreen experience.

3.4.1.7 Layout Concept

A **Layout** defines the presentation characteristics of DMAppComponents on the set of participating Devices in the Context.

A Layout is computed for a dynamically-varying set of DMAppComponents over a set of diverse devices; it specifies the physical placement of the active DMAppComponents on the context's devices.

Regions are logical rectangular display areas, which are mapped onto underlying physical devices by the calling application; each region may be mapped onto a single device.

Each DMApp defines a set of constraints per component type that must be honoured in the layout. Constraints may specify minimum & maximum size, priority level, video/audio capabilities, etc., and may target specific regions. Constraints are specified for both communal and personal device types

The LayoutDocument is a layout description (e.g. in JSON format) which contains the constraint set and is part of the DMApp description.

3.4.1.8 Layout Orchestrator Concept

A **Layout Orchestrator** computes the layout for a dynamically-varying set of DMAppComponents over a set of diverse devices and determines the best distribution, positioning and sizing of the DMAppComponents over the set of context devices, per the constraints, and returns a list of components with per device, with region, position and size. The orchestrator manages the context devices, stored in a backend database.

The **Layout Orchestrator** manages the set of devices in a given context along with the running components and provides the computation engine with full context and component constraint specifications for each layout computation [6].

The layout is generated by request or triggered by changes to the DMApp **context**, such as the arrival of a new device.

3.4.1.9 Timeline Concept

A **DMApp** has a **timeline** that models the expected lifecycle and presentation state of the DMAppComponents on devices in response to DMApp state changes such as DMApp current time, DMAppComponent-loaded, ready and end-of-presentation events. A **timeline** can be **fixed** or **editable**. The TimelineDocument is a specification of the timeline model and is part of the DMApp description.

3.4.1.10 Timeline Orchestrator Concept

A **Timeline Orchestrator** is responsible for executing the Timeline model of a DMApp as it is presented over a set of participating devices in a context. It uses authored timeline metadata, current DMApp time, DMApp state, such as the status of DMappComponents and optionally live triggers, to determine which **DMAppComponents** should be loaded for presentation as the timeline of the experience progresses. It also specifies a source of timing to the DMAppComponents for their playback and synchronisation.

TimelineEvents are triggers emitted (usually by the clients) that can cause different branches in the experience storyline to be taken.

TimelineEdits are modifications to the timeline document, provided by an external agent such as an Editor, that change the current experience.

A Timeline Editor is an external agent can dynamically issue live-edits to a DMApp's **Timeline**.

3.4.1.11 DMAppRuntime Concept

A **DMAppRuntime**, referred to in 2Immerse platforms as Client-API, is a runtime running on each device, that initialises/starts and coordinates the DMApp on that device based on its DeviceRole. It is responsible for executing the presentation state of the DMApp on that device, i.e., acting upon requests to (un)load/start/stop **DMAppComponents**, and for reporting state changes.

- It enables DMAppComponents to be loaded from internal and external sources, if necessary, by acting upon requests from the LayoutOrchestrator and the TimelineOrchestrator or by listening to specific DMApp state updates.

[6] In 2Immerse, the layout computation engine is stateless and computes a full layout for a given set of parameters. The layout service is stateful and maintains the context state in an underlying data store.

- It schedules the playback of DMAppComponents and provides a source of timing via the Timeline Clock, to drive their presentation.
- It monitors active DMAppComponents and reports their state or LifeCycle status, presentation status and current time position, to interested parties such as the Timeline Orchestrator and the Media Synchroniser.
- It enables communal devices to be discoverable and to advertise context/DMApp information to other devices for joining the DMApp. Personal devices via their DMAppRuntime can discover communal devices, launch the DMAppRuntime on these and instruct them to join a DMApp and its context.
- It performs local positioning of DMAppComponents and reports layout information and updates to the LayoutOrchestrator.
- It listens for DMAppState changes and disseminates them to other DMAppRuntimes, the Timeline Orchestrator, the Layout Orchestrator and other entities via available channels (such as HbbTV app-2-app, and shared-state).
- It allows DMAppComponents to be remotely accessed by another DMAppComponent using RPC.

3.4.1.12 DMAppState Concept

A DMApp has state, termed **DMAppState,** which may be shared and updated by distributed **DMAppRuntime**s, services or DMAppComponents.

DMAppState can have multiple **scopes** such as global (inter-context), context-local, device-local, and user-defined. **DMAppRuntimes**, services and DMAppComponents can subscribe to different DMAppState scopes.

3.4.1.13 Bandwidth and Bandwidth Orchestrator Concepts

A **DMAppComponent** that consumes and renders a media stream uses **media and/or audio bandwidth** resources on the network that is a finite resource

A DMAppComponent therefore reports **BandwidthUsage** (if supported by underlying media frameworks) and applies **BandwidthConstraints** determined by a **Bandwidth-Orchestrator** to stay within the prescribed bounds.

A **Bandwidth Orchestrator** collects **BandwidthUsage** from clients (DMAppComponents/ DMAppRuntimes) and allocates **Bandwidth** to running clients according to their defined QoS priorities (called **priorityLevels**).

3.4.1.14 Timeline Clock concept

A **DMAppComponent** controls its own internal playback media-timeline with respect to a supplied timing source or clock. This timing source, termed **Synchronisation Timeline**, is indicated by the Timeline Orchestrator and provided to the DMAppRuntime by the DMApp's **MediaSynchroniser**.

A **Timeline-Clock** is a linear scale against which the passage of time can be measured; it represents the temporal dimension of experiences and timed data playback. A Timeline-Clock is modelled using a **clock** abstraction, so that a timeline can be represented locally by a software **clock** object.

The relationship between the DMAppComponent's component timeline and a Synchronisation Timeline is expressed by a **CorrelationTimestamp**.

A **CorrelationTimestamp** models the temporal relationship between two timelines, and is a pair of two values each of which represents a time value on a timeline such that the two time values correspond to the same moment in time on the parent and child timelines. A CorrelationTimestamp

also provides **start** and **end** times; these define a bounded time period on the parent timeline during which the relationship expressed by the CorrelationTimestamp is valid.

A **Timeline Shadow** is a local estimate of a timeline from another device or destination against a reference time source, termed the **WallClock**. The estimate is refreshed when it receives new CorrelationTimestamps. A **Timeline Shadow** can be manifested locally through a software abstraction such as a clock object.

3.4.1.15 WallClock concept

A **WallClock** is a global reference time source against which media/data stream units can be timestamped at capture-time, in production or at presentation-time in a DMApp's master device. DMAppRuntimes obtain a local estimate of the WallClock and thus share a common sense of time. The WallClock is used as a base timeline for creating **CorrelationTimestamps**.

3.4.1.16 Synchronisaton Timeline and Media Synchroniser concepts

A **Synchronisation Timeline** is an abstract timeline used by all DMAppRuntimes to synchronise local media playback, if the relationship between the synchronisation timeline and the media timeline is known. The synchronisation timeline may be provided by a master DMAppComponent or generated from the timings of all devices or injected by the Timeline Orchestrator.

A **MediaSynchroniser** provides timing across the set of participating devices for the purpose of synchronising DMAppComponents playbacks. It allows one or more Synchronisation Timelines to be distributed to each device and a TimelineShadow clock object for use by the DMAppRuntime to synchronise DMAppComponents.

3.4.2 Main Abstractions

The concepts identified in the domain model provide the key abstractions (Functionality Groups) for the Functional Model:

• DMApp	• Context
• User	• Device
• DeviceRole	• Communication (
• Layout	• LayoutOrchestrator
• DMAppState	• Timeline
• DMAppRuntime	• DMAppComponent
• TimelineOrchestrator	• TimelineEventEmitter
• Bandwidth	• BandwidthOrchestrator
• MediaSynchroniser	• SynchronisationTimeline
• WallClock	• MediaDataSource
• MetaDataSource	• TimedDataSource
• DataSource	• ContentStore

Table 4: Main concepts in Domain Model

3.5 The DMApp-RA Functional Model

Next we evaluate the domain concepts in terms of the functions they encompass. We attempt to map them into existing Functionality Groups (FG) or create new ones, where the mapping is not possible. Where there is an overlap in functionality, we generalise the concepts and merge them into a new Functionality Group while ensure each FG is fairly self-contained and encompasses one

category of functions. After five iterations through the methodology (each evaluating a 2Immerse DMApp deployment), a generalised set of functionality groups was obtained. For the sake of brevity, only the results of the final iteration are presented in this section. Table 5 lists the final set of Functionality Groups, whereas the intermediate results of the Functional Model are presented in Appendix C (See section 8.1).

Functional Group	Description of Coarse Function
DMApp	Distributed Media Application – not technically a platform functionality group but may include user-defined functions, e.g., user score aggregation in a distributed quiz DMApp
DMApp Store	A repository of live or on-demand DMApps. Queried by DMApp launchers on devices to obtain current list of relevant DMApps
Media Source	A media capture device or source for a media stream
Media Sink	A DMAppComponent which consumes timed media data, e.g., audio/video streams, or untimed data, e.g., images, text, tweets.
Data Sink	A DMAppComponent which consumes non-audio-visual data
Timed Data Sink	A DMAppComponent which consumes non-audio-visual data but which has timing relationships between the data units
Device	A communal or personal display device
Device Role	A DMApp specific role for each device
Communication	Protocol stacks, messaging infrastructure, HTTP-based interaction schemes such as Web Services and REST, etc.
Layout Document	A set of **constraints** used to determines how/if DMAppComponents should be displayed on a device fulfilling a role
Layout Orchestration	Generates a component layout for a set of device, regions, layout constraints and active DMAppComponents, defining the size and positions of the components on the participating devices. The DMAppRuntime uses this layout to present DMAppComponents onto a set of rectangular areas representing graphic display regions in each screen
DMApp Timeline	A specification of the expected lifecycle and presentation state of the DMAppComponents on devices in response to DMApp state changes
Timeline Orchestration	Concerned with manipulating and controlling the execution model of a DMApp instance, i.e., unravelling the storyline of the experience with the passage of time or selecting a particular branch upon receiving events
Bandwidth Orchestration	Monitoring and managing the bandwidth consumed by streaming media (principally DASH video) components in a running DMApp to optimise the quality of experience
Timing and Media Synchronisation	Provides an accurate notion of timing to other components to enable starting DMApp components at the correct time and maintain/preserve intra-media timing to allow receivers to recreate the sender's/encoder's timing for correct playback. Also provide inter-device and inter-destination media synchronisation to respect inter-media temporal relationships set at DMApp creation time or decided during live production
State Management	Ensures that all devices and services have a consistent view of the distributed shared state as it changes dynamically. Declarative model for specified and dynamic binding of functionality to these variables. State variable change signalling
DMAppRuntime	Initialises, starts and coordinates the DMApp on a device based on its DeviceRole. Responsible for executing the presentation state of the DMApp on that device, acting upon requests to (un)load/start/stop **DMAppComponents**, and for reporting state changes.
Launching & Onboarding	Functions related to user enrolment to the DMApp platform, pairing of devices to user accounts, launching of DMApps on the host device and discovering other

	devices to onboard in the running DMApp on the local network
Security	Access control to platform functions, content stores. Authorisation of requests as per user permissions, encryption/decryption (DRM) of media streams
Management	Functions related to the configuration and management of the DMApp as well as the platform
Content Store	A secure data store to serve client applications, DMAppComponents and their assets, timeline and layout documents, and data/media streams.
Timeline Editor	Timeline editing to change the storyline by inserting at run time timeline snippets and their activation signals or event names. Live insertion of new timeline branches. Event triggering to activate pre-defined and templated sequence of timeline processes.
Data Playback	Capturing, packaging and efficiently distributing non-audio and non-video time sensitive data
Media Processor	Audio/video encoding and transcoding, multiplexing of streams into transport stream packaging units and injecting timestamps/sequence numbers to allow correct playback rate at the receiver and synchronised presentation of multiplexed streams.
Data Processor	Data stream creation and timestamping using global reference clock.

Table 5: Functionality Groups after 5 iterations

In addition to identifying and grouping common functions, it is necessary to specify how these functions are invoked. **Figure 3** depicts an abstract arrangement of the Functionality Groups that makes the interactions and dependencies between them explicit. In the reference architecture derivation phase, interface specifications will reify these interactions.

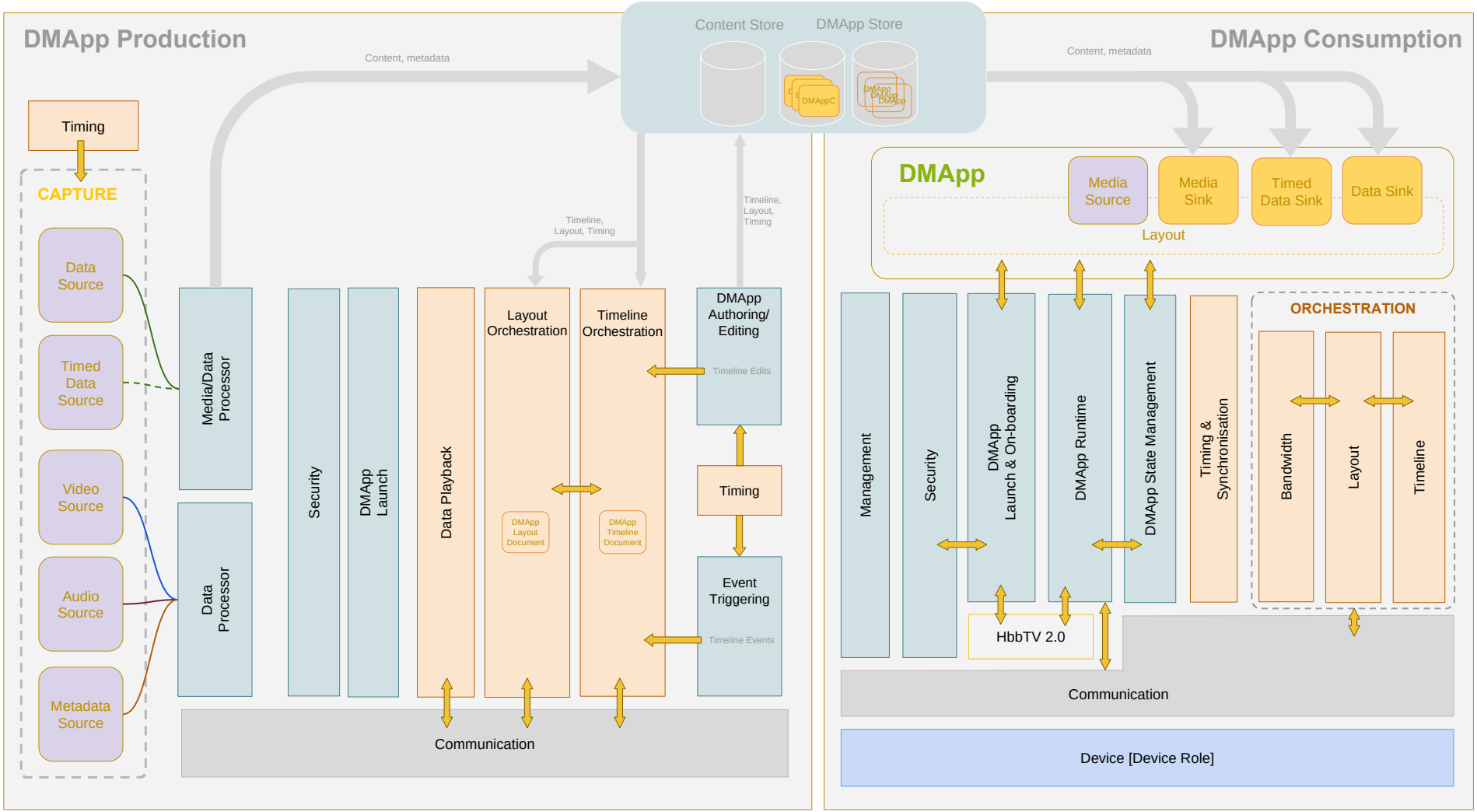


Figure 3: DMApp-RA Functional Model after 5th methodology iteration

3.6 The DMApp-RA Functional View

The Functional View is developed by first identifying self-contained units of finer-grain functionality obtained in our functional decomposition of candidate platforms. Each function is analysed to determine the FG to which it belongs; functions are moved to FGs to maintain a strong separation of concerns that may require the creation of new FGs. Each function is evaluated to determine whether it is a variation of an existing function extracted in the previous iteration or whether it is entirely new. In the former case, the existing function is generalised and its variation in functionality noted. It is possible to group together function-variants in sub-function groups.

Successive iterations produce a more generalised functional model and view. The iterations are terminated when the methodology reaches a saturation point, i.e., the functional decomposition of new DMApp platforms does not result in the identification of significantly new core functions.

3.6.1 Functional View for DMApp Production and Consumption

The first iteration of functional decomposition was based on the Functional Model derived from the Theatre-At-Home DMApp. It was, as expected, the most effort-intensive part of the process as it involves identifying the core functions. Since incremental addition of new functionality was made to the 2Immmerse platform to suit the use case requirements and application domain, our successive refinements of the Functional View also reflects the addition of new functions. The Functional View obtained after the functional decomposition of the Theatre-at-Home DMApp platform is provided in Appendix C.

In the fifth iteration, the Theatre-At-School DMApp (the final DMApp developed), whilst being in the same application domain as the first DMApp (performing arts), exhibits some markedly different variations in functionality:

- The timeline is completely user-driven: for example, the teacher decides on duration of each sub-lesson, sets activities for students on their tablets.
- It has looser and more complex coordination/control semantics: for example the teacher hands control to groups of students at given times in the lesson.
- Timing and sync-accuracy requirements are strict but shorter-lived: videos put up by the teacher on the main screen needs to be synchronised with student-tablets and the teacher's own tablet.
- User state, such as students' deliberations and observations, is created on-the-fly, and shared with and presented promptly on other devices with low-delay.

Additional functions identified from the functional decomposition of this DMApp are assessed to determine the FG they belong to and whether new FGs need to be created. The Functional View that results from this last iteration is presented in Figure 4 and Figure 5. Figure 4, shows the decomposition of functions related to DMApp orchestration and consumption. Figure 5, on the other hand, shows the functions required during DMApp live and non-live production.

After the five iterations, the Functional View is augmented with functions related to fulfilling Non-Functional Requirements. These are shown as red boxes in the FGs in Figure 4 and Figure 5.

A description of the Functionality Groups and their incumbent functions is deferred to the next section (See Section 3.6.2).

For the interested reader, the intermediate Functional View after the analysis of the Theatre-At-School DMApp is presented in Appendix C.

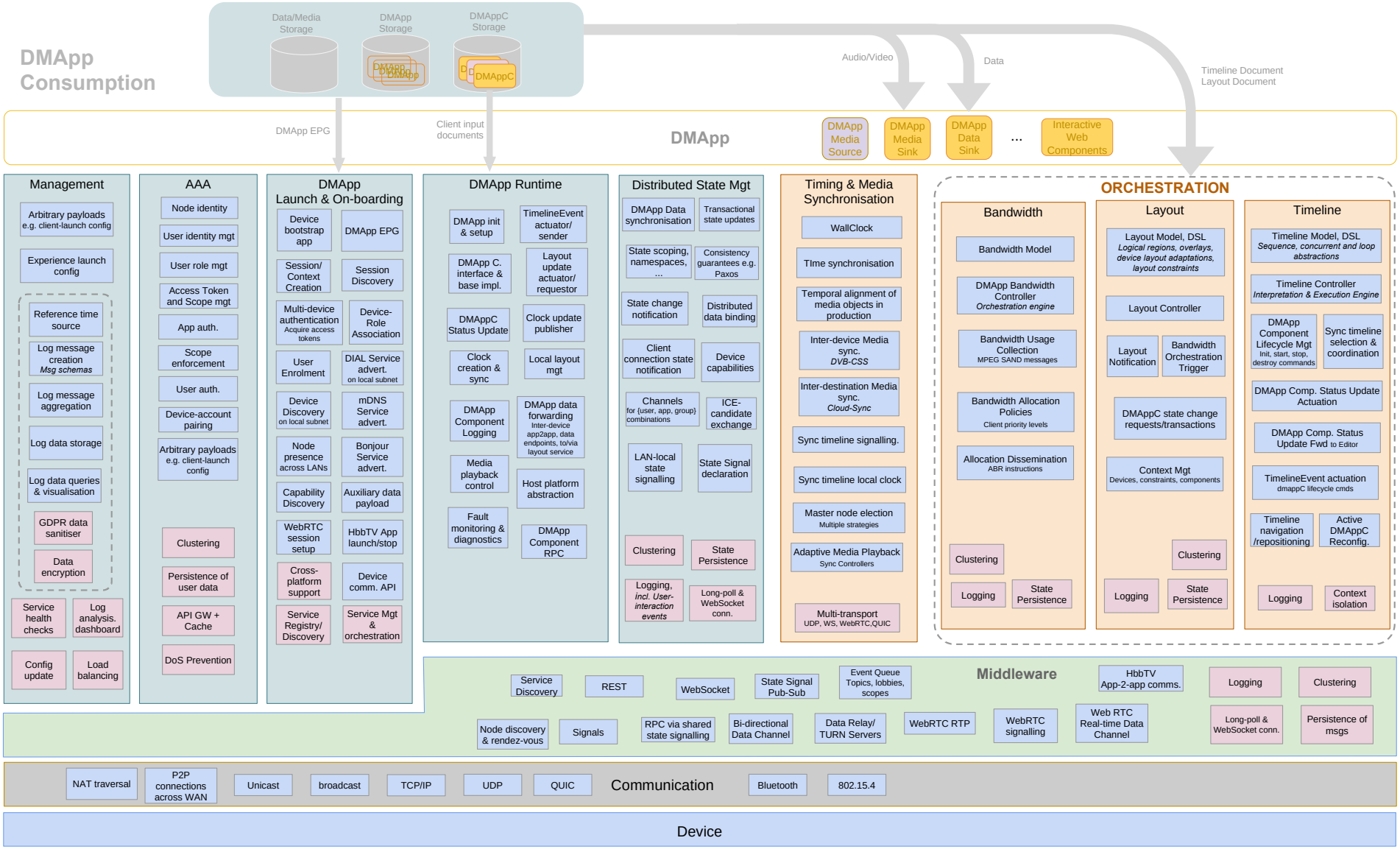


Figure 4: DMApp-RA Functional View (Part A) after 5th iteration (Theatre-At-School DMApp)

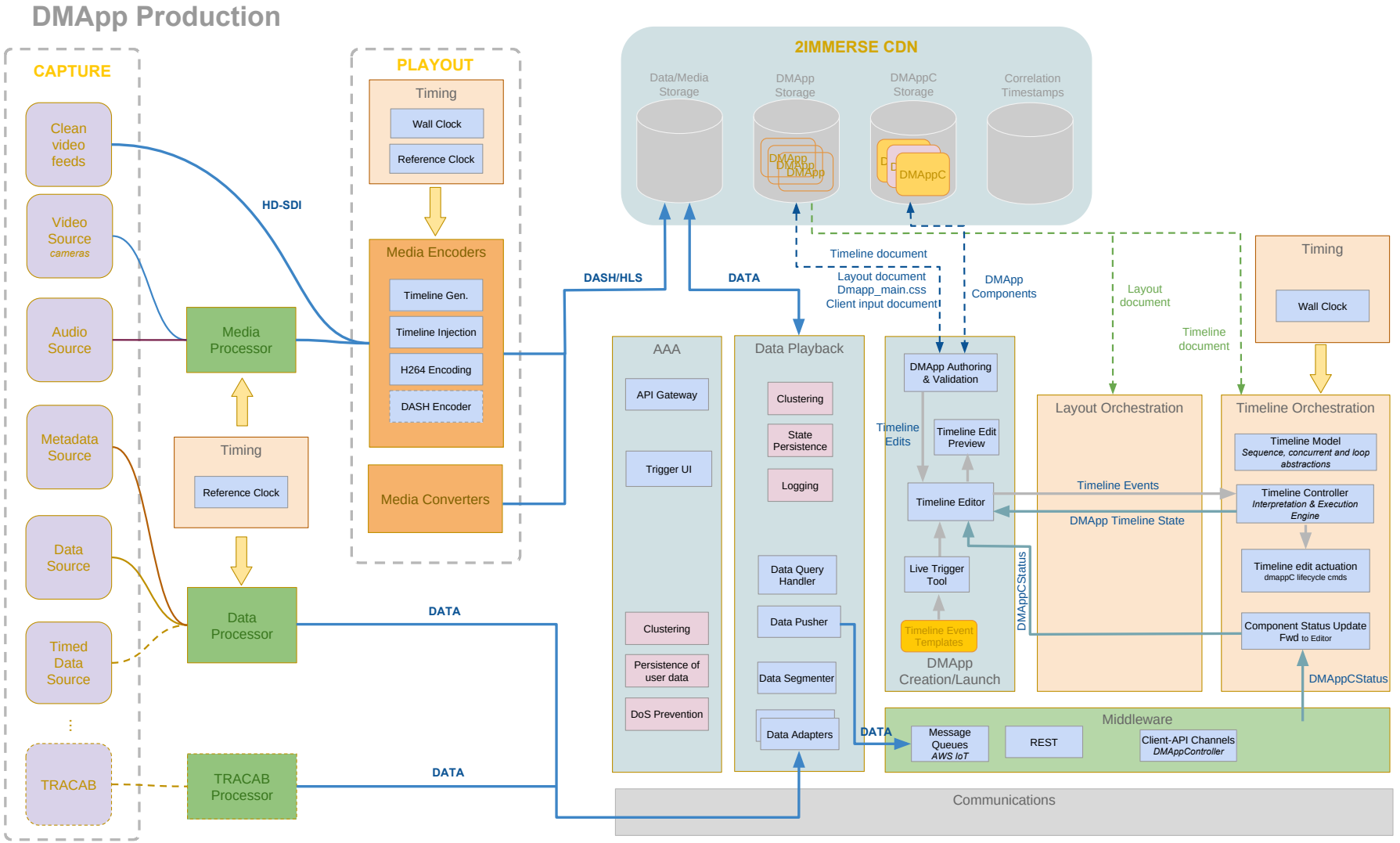


Figure 5: DMApp-RA Functional View (Part B) after 5th iteration (Theatre-At-School DMApp)

3.6.2 DMApp-RA Functional View Components

3.6.2.1 Device

This concept abstracts over the heterogeneous devices that are targeted for DMApp playback. It includes **communal** devices such as TVs, large screens, home speakers, **personal** devices such as tablets, phones, smartwatches, wearables) and **shared-personal, such as** pass-around tablets/phones.

These devices provide host-application environments for running the DMAppRuntime.

To ensure portability and consistency of DMApp component presentation across devices, it is desirable for the host application environment to support open web standards (HTML5, CSS and JavaScript) and wherever possible hardware-accelerated audio/video decoding. For communal devices such as hybrid televisions, a host application environment that implements the HbbTV 2.0 profile is preferred. The HbbTV 2.0 standard-compliance ensures that TV devices provide a mandatory set of functionalities to the DMApp via standardised JavaScript APIs. Personal devices are also able to interact with communal devices across interfaces defined for Companion-Screen communication in the HbbTV 2.0 standard.

On other communal device types, emulation of part of the HbbTV2.0 technology stack can be used to ensure consistent interaction interfaces with personal devices. For example, a standard device discovery mechanism that is guaranteed to work across different implementations.

In addition to particular application environments and APIs available therein, devices may have other capabilities that are relevant to the DMApp e.g. screen size, resolution, speaker, touch-screen, camera, microphone, other sensors such as accelerometer, gyroscope, ambient light sensor, etc.

3.6.2.2 HbbTV Stack (merged into Middleware, Media Sync, Launching & Onboarding)

HbbTV is an open-standard for hybrid digital televisions and an industry standard for the delivery of interactive entertainment services to consumers on connected TVs, set-top boxes and multiscreen devices, from broadcast and broadband sources. Interactive services take the form of HTML and Javascript running in a browser engine and can co-exist with the presentation of broadcast content. HbbTV defines a profile of HTML, CSS and Javascript capabilities for TVs to support, and also defines APIs to control functionality specific to TV devices (such as controlling the broadcast tuner).

HbbTV 2.0 adds a range of functionality to support interaction between the TV and a companion device:

- Companions discovering the TV on the home network.
- Companions launching an HbbTV app on the TV.
- TV launching an app on the companion.
- Bi-directional app-to-app communication between HbbTV app and companion app.
- Media synchronisation:
    - HbbTV apps knowing the current timeline position of TV content.
    - HbbTV apps supplementing broadcast with a replacement for the audio or video, synchronised to the broadcast but streamed via IP, with frame accuracy
    - Companions knowing what the TV is showing and synchronising to it with frame accurac

The HbbTV2.0 FG in the Functional View refers to the two implementations of the HbbTV2.0 software stack depending on the device role: the master screen or the companion screen.

3.6.2.3 DMApp Component

DMAppComponents are entities that encapsulate functionality and user interface elements in discrete entities which are individually specified and controllable by the Layout and Timeline Orchestration FGs. DMAppComponent can be downloaded, dynamically loaded, linked with the application, and its operation controlled by an application runtime. The need to be able to instantiate DMAppComponents on heterogeneous devices along with maintaining consistency in look and behaviour, makes WebComponents a natural choice for implementing these components.

WebComponents is a web standard for encapsulating HTML and the associated CSS style and JavaScript logic into a single HTML tag. This encapsulation and declarative usage style ensure that components can be easily used without requiring the host page to be aware of its internal implementation. JavaScript APIs enable these components and their associated behaviour to be defined as custom elements included in HTML pages. The JavaScript APIs allows a "shadow" DOM tree to be attached to the custom element, rendered and controlled separately from the main document DOM. From our experience of implementing DMAppComponents, we recommend that WebComponents be used as the basis for all DMAppComponents implementations. The implication is that some aspects of DMAppRuntime functionality need to be implemented in browser environments, such as Chromium and WebKit, to allow it to perform functions such as DOM-tree manipulation, attaching listeners to DOM element properties, etc. and functionality implemented natively be accessible via Javascript APIs using frameworks such as Cordova and Web Assembly.

A DMAppComponent is therefore a JavaScript object[7] that implements a defined and documented interface. The DMAppComponent interface includes control of the component's life-cycle, visibility, position, and related functionality, and provides references to utility interfaces for handling of clocks and time-based cueing, component parameters and shared state.

A common implementation pattern is for the host component to include a default implementation of all the required parts of the DMAppComponent interface, so that DMAppComponent authors only need to implement the parts which are relevant for their components, overriding the default behaviour where necessary.

For orchestration purposes, it is useful for DMAppComponents to have a set of pre-defined lifecycle states such as the ones shown in Figure 6.

[7] It is also possible to build DMAppComponents using native UI frameworks such as Qt, Google's Flutter SDK,iOS SDK, Android SDK. It is easier for DMAppComponent control and inpection, if the DMAppRuntime is implemented in the same environment or in a language that can be cross-compiled to different application target environments.

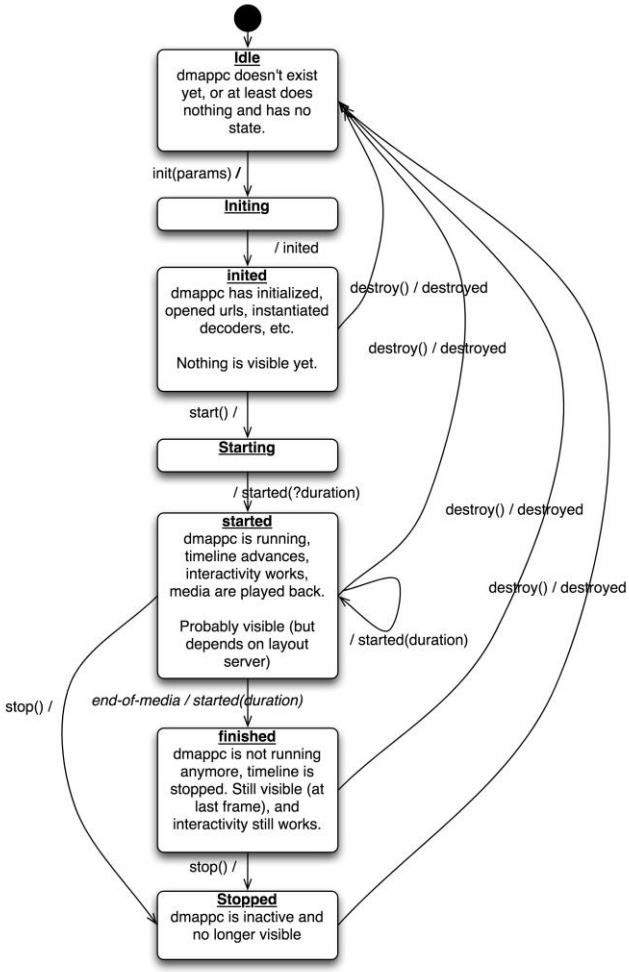


Figure 6: DMAppComponent lifecycle states

3.6.2.4 Communication

This FG encompasses protocol stacks including the transport, network and link-layer protocols (TCP, UDP, QUIC, and IEEE 805.15.4 low-power packet radio MAC), network abstractions, addressing issues, network management and device-specific communication stacks such as Bluetooth, IR or IEEE 805.15.4 for IoT Devices.

3.6.2.5 Middleware

The Middleware FG abstracts over the variety of interaction schemes built on top of the communication stacks running on DMApp devices and service platforms. It provides a number of interfaces/abstractions to the FG above for

- sending/receiving data including unicast, broadcast and full-duplex channels via WebSockets,
- invoking functions via RPC mechanisms (e.g. Web Services, REST gRPC, Apache Thrift, XML RPC),
- distributing information via communication paradigms such as pub-sub (message brokers), tuple-spaces, distributed shared memory, etc. Pub-sub message brokers can be used to build a distributed state management system that supports named scopes and binding of state signals to client code.

3.6.2.6 Timeline Orchestration

The Timeline Orchestration FG functions are concerned with manipulating and controlling the execution of a DMApp instance, i.e., unravelling the storyline of the experience with the passage of time or selecting execution paths upon receiving events, or moving along the experience timeline to another position (rewind/forward). The functions in this FG follow an MVC pattern, where a timeline document provides a starting execution model for the DMApp. The model is updated at the DMApp progresses with state such as current WallClock time, DMAppComponent clock time, DMAppComponent status, and user-interaction events. A controller acts on changes made to the model to update the DMApp View – the presentation of DMApp Components on devices.

Timeline Model

There are various strategies to modelling storylines for object-based multi-device experiences and orchestrating storylines. Notable approaches include the following:

- Modelling the storyline as a linear experience timeline that is driven by a real-world clock and defining the storyline branches and DMApp component presentation as timelines correlated to the main timeline by correlation timestamps. The main timeline could be provided by timing signalled in a broadcast video stream for example.
- Modelling the storyline as a Mealy-type Finite State Machine and defining inputs or DMApp state variables as a set of events, signals or parameters that are emitted or updated as the DMApp progresses, e.g., DMApp components start or complete their presentation, a device joins the context, or a user interacts with the DMApp. These would trigger DMApp state transitions. The state machine is itself time-invariant and DMApp progress can only be measured via state transitions.

The 2Immerse approach for timeline modelling is closer to the second strategy as an event-based storyline model with timing inference from other sources, was deemed to be more suitable for the heterogeneous requirements stemming from the various use cases. These requirements included the non-deterministic duration of some content as in football matches, or the user-interaction-driven experiences as in the Theatre-at-School use case.

The linear experience timeline model is insensitive to delays and makes the expression of non-deterministic presentation timing difficult. It requires that all timing relationships between media objects are pre-calculated. Whilst the main advantage of this timeline model is that it is an easy-to-understand representation of continuous media objects under deterministic timing conditions, it cannot handle situations where the duration of a media object changes over the lifetime of the presentation. A timeline model based on the explicit media timings alone may not be rich enough to model the various structured paths throughout a DMApp presentation.

Timeline Document Language

A Timeline Document specifies in string format the timeline of the DMApp. It contains the set of execution steps and rules for the DMApp; these are used to prime the Timeline Model and are then evaluated as events are produced during the DMApp. The timeline document language is a DSL (domain-specific language) that allows the specification of a timeline document and has the necessary constructs to enable the expression of diverse DMApp execution semantics, constraints and inter-media object temporal relationships.

Several hypermedia languages, such as Nested Context Language (NCL) (15) and Synchronized Multimedia Integration Language (SMIL) (16), allow for defining causal and constraint relationships between media objects.

The 2Immerse implementation of the Timeline Orchestration concept adopts a SMIL-like structured timing model (2) which provides timing facilities for when media objects are scheduled, and once activated, their lifespan. The model provides nested presentation structural elements for the

composition of media objects and a timing scoping mechanism where the timing of a media object can be inferred from its timing scope (by the context in which a media object is presented in relation to other objects) or its parent at presentation time. From SMIL, it also borrows structuring and timing mechanisms to model parallel and sequential composition (cf. time-graph), conditional composition, transitions and more complex behaviour such as user-interaction. For example, it provides elements and attributes to describe non-deterministic timing of media objects e.g. timing as a result of using interactive or event-based presentation. This feature, in particular, allows the Timeline Orchestration concept to listen to lifecycle events of DMApp component instances and to Timeline Events, and actuate particular storyline branches.

Timeline Controller

The Timeline Controller is responsible for

1) updating that model based on DMApp state changes e.g. lifecycle status changes of DMApp components, such as a video media object finished playing, or a clock driving a segment of the DMApp storyline reaching the end-time
2) comparing the changed model with the current DMApp presentation state
3) determining the changes required to the presentation state (e.g. DMApp components to be stopped/destroyed, new DMApp components to be initialised, initialised DMApp components to be started)
4) scheduling the presentation of DMApp components, allowing for delays such as component-loading, media download, media-processing pipeline-priming, component-unload, etc. whilst still maintaining the authored temporal relationships defined in the timeline document
5) requesting the Layout Orchestration concept to actuate DMApp component load-unload commands
6) providing a timing source (a clock source) to the individual DMApp components to slave their presentation to it

Media Synchronisation Timeline Coordination

This function provides master role designation and timing sources (clocks or timelines signaled in stream, correlation mappings) as synchronisation timelines to DMAppRuntimes. The DMAppRuntimes can then provide this timing information to the media synchronisation functions to achieve inter-device and inter-destination synchronization.

3.6.2.7 Layout Orchestration

The Layout Orchestration FG relates to functionality of a DMApp View Controller. It manages and coordinates access to the presentation surface of the DMApp and it validates/actuates on commands to present DMApp components on that surface based on each device presentation characteristics and constraints. The DMApp View or presentation surface is comprised of the output surfaces of a set of devices. In a multi-screen experience, the DMApp View is all the screens of the participating devices and represents the layout for DMApp components[8].

Given a set of media objects / DMApp Components, authored layout requirements, user preferences, and the set of participating devices and their capabilities, the Layout Orchestration FG will determine a best attempt for an optimal layout of components for that configuration. The functions include the following:

- *Context Management:* define the set of devices, constraints and components that comprise a context.

[8] The presentation surface also includes other output modes such as audio, olfactory media.

- *Layout Computation:* given a set of devices/regions, constraints and components, the layout FG will generate a layout defining the size and positions of the components on the layouts. The layout engine attempts to maximize the number of components laid out with minimum internal holes while adhering to the constraint set. The layout is generated by request or triggered by context updates.
- *Layout Notification:* each time a new layout is generated, notifications are sent to the affected devices
- *Bandwidth Orchestration Interaction:* if requested, the layout FG notifies the Bandwidth Orchestration FG when relevant events occur that affect bandwidth usage, e.g. video component start/finish.

3.6.2.8 Bandwidth Orchestration

In a DMApp running on the home network, if access to network bandwidth is unbridled and uncoordinated, a race condition is likely to develop as multiple video components end up competing for bandwidth (typically video chat (WebRTC) and DASH video streams). In adaptive video streaming, a variety of bitrate adaptation algorithms are designed to estimate available bandwidth, and request the best possible encoding. With concurrent adaptive video streams on multiple devices in the network, a race condition develops between these parallel algorithms. The on/off requests from multiple DMAppComponents (or sessions) that compete for bandwidth across a bottleneck link cause *i)* instability in the selected encoding, *ii)* bottleneck-link under-utilisation, and *iii)* disproportional shares of available bandwidth. These uncoordinated client-side ABR algorithms result in random oscillations between different bit rate representations selected in response to fluctuating bandwidth. This is detrimental to the QoE; instability, for example, appears to the user as images of varying quality over time. Further, under-utilisation may prevent clients from requesting the best possible encoding for the user.

This is a domain-wide issue and the MPEG's Server and Network Assisted DASH (SAND) specification (17) defines an architecture and interfaces to specifically address QoS and QoE support for DASH-based services.

The Bandwidth Orchestration FG includes functions aligned with MPEG SAND to monitor and manage the bandwidth being consumed by streaming media (principally video) components in a running DMApp and optimise QoE. Similar to the other orchestrators, an MVC pattern is applicable to this FG.

At the time of writing, the effectiveness of Bandwidth Orchestration approach taken via MPEG was limited. Achieving bandwidth allocation fairness at the application layer without transport-layer knowledge of flows is difficult. Web browsers do not expose transport-layer performance metrics to JavaScript to allow adaptive bitrate systems to make accurate available bandwidth estimation and fairer bit-rate selection decisions and improve the job that MPEG-SAND does.

Further, the nature of DASH segment downloads on diverse devices results in an intermittent pattern of requests which is not very nice to TCP. TCP has its own flow control mechanism – it will keep increasing its congestion window size until it detects a packet drop, upon which it will do a back-off and reduce the window size. The result of this behaviour is that every new HTTP request is launched with stale TCP state from the previous request. This on/off behaviour prevents TCP from reaching equilibrium (18). Rather than equal or weighted share of capacity, obtained bitrates appear to be determined by factors such as the time of arrival relative to other streams, the viewing platform and implementation, operating system supports, and the content provider. , As reported in this investigation: "Client-Driven Network-level QoE fairness for Encrypted 'DASH-S" (19), the actual reality is that the application-layer Bandwidth Orchestration feedback loop (in the 2Immerse implementation) is working independently with another feedback control loop lower down the protocol stack (the TCP flow-control mechanism). This application-layer feedback loop uses coarse-grain knowledge it can gain, e.g. estimating bandwidth from download times.

It has been suggested the transport layer is the best layer at which to achieve fairness, as the DASH application layer doesn't have enough of a global view to ensure fairness between DASH and other adaptive bitrate systems, such as the WebRTC video conferencing used in the Theatre-At-Home service trial. The conclusion is that cross-layer adaptation mechanisms are required i.e. metrics from both the application layer and the network layer are required to devise a system to improve fairness of bandwidth utilisation between networked DASH clients.

The functions of the Bandwidth Orchestration FG are independent of the choice of implementation. The algorithms for the collection of bandwidth usage may use alternative mechanisms than MPEG-SAND.

Bandwidth Model

The bandwidth model embodies the algorithm or computation model to partition bandwidth and decide on a bit rate selection for the different DMApp components in a single DMApp.

The model is initialised with the following information:

- **Available Bandwidth:** optional parameter specifying the available bandwidth for a given experience. If not supplied, the algorithm attempts to estimate the available bandwidth from the overall bandwidth usage of the experience.
- Active **DMApp components** in the Context (LAN) and **the available bit rates** for the media streams
- **Priority Levels:** user defined priority scale used to determine the required QoS of the video clients.

When talking about multiple playback client with several bitrate choices for each one and a set priority (QoS) for client, the problem of determining the optimal allocation of bandwidth among the clients maps directly on the Quadratic Knapsack computation problem (20). In essence, the problem is to find the correct choice of bit rate representations for active DMApp 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. This problem is NP-Complete and hard to solve and thus finding the optimal solution is impractical for real-time uses. It is possible to use tight approximative solutions (21), (22), (23), (24) which are complicated but time-consuming, though monumentally faster than calculating the optimal solution, as well as greedy algorithms that are not as tight but can run in real-time to reach a useful solution.

Bandwidth Usage Collection

This function collects bandwidth usage information as well as available bit rate representations for the video streams from running clients and reports this via MPEG DASH SAND messages.

Bandwidth Controller

The controller is responsible for updating the bandwidth-model based on updates about active DMApp components, their available bit rates, and on reception of SAND messages from DMAppComponents (that support SAND reporting). Thereafter, it performs the following functions:

- **Bandwidth Allocation:** based on the collected data, the controller periodically computes bandwidth allocation between running clients according to their defined QoS priorities.
- **Action Notification:** when appropriate, based on the bandwidth allocation computation, the service sends action notifications to the affected clients with ABR bitrate selection instructions. In the scenario where there are sustained reported stalls, or observed competition for bandwidth (i.e. bandwidth oscillations between multiple player components), the FG 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).
- **Layout Orchestration FG Interaction**

- It listens to requests from the Layout FG to initialise or stop computations for given DMApps
- In the scenario where none of the video players can move to a lower rate representation, the service should work with the Layout FG to determine which of the DMApp components can be terminated to improve the overall quality or experience for the remaining players.

- **Bandwidth Orchestration State Sharing**: Other platform services may benefit from being made aware when player components are experiencing issues (e.g. stalling), for example the Timeline FG.

3.6.2.9 Timing and Media Synchronisation

Distributed components and services responsible for the execution of a DMApp need to maintain an accurate notion of timing to instantiate DMApp components at the correct time and maintain/preserve the inter-media temporal relationships set at DMApp creation time or during live production.

Delay and delay variability introduced by delivery processes from source to receiver can result in unpredictable presentation behaviour.

- At the source-side (including the playout system), sources of delay and delay-variability include capturing, sampling, encoding, encryption, packetisation, protocol layer processing, and transmission buffering.
- When the media units are transported over packet-switched networks to the receivers, delay variability is added due to network jitter resulting in the original temporal relationships between the media units not being preserved; these need to be reconstructed at the receiving side via buffering and sync.
- At the receiver side, end-system jitter (CPU load, OS responsiveness), buffering and presentation delay also introduce variability. Even if the presentation of a media stream on two devices were to be perfectly time-aligned, imperfections in their clock will cause their clocks to drift and affect the accuracy of the media stream decoding and rendering processes.

The result of this delay variability is that even in single-device DMApps, end-to-end delay differences can occur when simultaneously delivering different media components via the same technology to the same receiver (e.g., an audio and a video stream sent in individual RTP/RTCP streams).

The same effect can be observed in multi-device DMApps when simultaneously delivering the same media content(s) via the same technologies to different devices, regardless of their location.

If the media objects are captured/sent by different sources and these are not time-aligned, the asynchrony will be even more pronounced.

These delay differences mean that the media objects need to be time-aligned or synchronised when they are presented on their host devices. For synchronisation or time-alignment to be achieved on the consumption devices, the following capabilities may be needed.

Time Synchronisation

All capture, production, orchestration and receiver devices must maintain a coherent notion of time by keeping their clocks synchronised to a global reference clock. Most types of media object presentation control assume that the clock ticks at the sources and destinations have the same advancement and the current local times are also the same, i.e. the use of *globally synchronised clocks* make simultaneous presentation-timing control simpler. Time synchronisation schemes such

as NTP, PTP or GPS can be used for adjusting clocks at different devices and maintain a synchronised WallClock.

Timing Signalling to Media Object Players

For correct media playback at the receiving device, the receiver needs to replicate the media source's time system; time and timing information are the tools to facilitate this task. Time indicates the exact moment when units of the media stream should be displayed and is obtained from the timing information signalled to the player.
Temporal relationships between media stream units are, for instance, signalled to the receiver devices by multiplexing timing metadata or timeline (timestamps, sequence numbers, source and group identifiers, markers, event information to Media Unit) in the streams. This timing information can be conveyed using *in-band timeline signalling,* i.e., timestamps, sequence numbers are included in the Media Unit packets. As an example, MPEG2-TS streams accomplish intra- and inter-media synchronisation via Transport Stream timelines inserted by the encoder and used by the decoder for play-out. Timelines and timestamps (time and timing) are inserted within the MPEG2-TS packets. Using the timeline injected, multiple media streams of a program inside the transport stream, e.g., video, audio, and subtitles can be time-aligned for presentation.

Temporal relationships between independent media-streams are signalled *out-of-band* to receiver devices using out-of-band schemes such as correlation timestamps or time-bases via SMIL.

Timing information signalling enables the Timeline Orchestrator to determine when to load/unload media objects (and decoder pipelines primed) and when they are to be started/stopped. For non-live DMApps, the DMAppRuntime on the master device signals the start time and the timeline-clock used to Timeline Orchestrator. A master timeline or clock may be preferred by the Timeline Orchestrator for pacing the playback of timed DMAppComponents. Sharing and maintenance of an accurately synchronised local estimate of the master timeline can be achieved through the services of media-synchronisation schemes such as Cloud-Sync, Shared Motion or DVB-CSS.

Media Synchronisation

Media synchronisation methods follow one of two approaches: *a distributed method*, or a *centralised meth*od to evaluate asynchrony between multiple devices and compute a presentation timing for all.

In the distributed method, each destination transmits information about the output presentation of media units, e.g. video frames, audio units, at the destination to all the other destinations. Each device is then responsible to calculate a *reference presentation timing* according to the received information from the other destinations by using the same method. All devices should then adjust their DMAppComponent presentation timing to the reference presentation timing.

In the centralised method, there is a single sync-controller component that gathers the output presentation timings from all the destinations. The controller determines the reference presentation timing and multicasts this information to all the destinations. When each destination receives the information about the reference output timing, it applies a local resynchronisation strategy to adjust its output timing to the reference output timing.

LAN-local Inter-Device Media Synchronisation

The reference presentation timing computation uses presentation timestamps from all devices and for the reference presentation timing to be relevant; the computation algorithm needs to be fed the freshest possible presentation timestamps. Because the accuracy of the presentation timing decays with time, it is recommended to keep, wherever possible, reference timing computation localised to the LAN (lower transmission delays for presentation timestamps). Thus, the need for a LAN-local Inter-Device Media Synchronisation function in DMApps where the context is a LAN.

A suitable choice here is DVB-CSS (25) – a master-slave synchronisation/timing control scheme 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. It takes advantage of the low-latency communications (including the use of low-latency network transport protocols) between the devices to achieve frame-accurate media synchronisation. One of the benefits of this scheme is its ubiquity in HbbTV 2.0 televisions. DVB-CSS is the mechanism adopted by HbbTV to achieve media synchronisation between a television and second-screens. With the widespread adoption of HbbTV 2.0, one can potentially rely on an inter-device media-sync solution being already present in the living room. DVB-CSS sync and app-control service endpoints are exposed via DIAL which is designed to work in a local network setup (LAN-broadcast for SSDP messages).

Inter-Destination Media Synchronisation

For DMApps that include more than one destination, an inter-destination media synchronisation solution (IDMS) can be used to achieve synchronised DMAppComponent presentation among geographically distributed users. Centralised or distributed variants of timing control schemes can be used to fulfil the IDMS functions.

In the centralised approach, the synchronisation is controlled by a synchronisation master, the Sync Controller, which is either the media source or a separate node, but not a peer. The synchronisation master collects timing information and sends timing instruction to which the peers must adhere. Examples of solutions that are based on Synchronisation Master Scheme include 2Immerse's Cloud-Sync service (20) and MediaScape's Shared Motion service (26).

Distributed IDMSes are, on the hand, are based on a distributed control scheme. They use distributed protocols to determine the reference playback timestamp to which the peers may synchronise. Timing information is exchanged in a P2P manner among the peers. This scheme has the highest robustness in terms of overall failure probability as the content provider is only required to host the multimedia content. Nevertheless, the peers have to trust each other as sources of timing information and must all be time-synchronised. The requirement that devices from different networks communicate with each other may be problematic if Network Address Translators (NATs) are used on the device's host network (especially if the peers are behind symmetric NATs). NAT punch-through solutions such as STUN servers may be needed in this case. Examples of solutions that implement a Distributed Control Scheme for IDMSes are (27) and (28).

Local Timing Abstractions

Representation and manipulation of timing through suitable abstractions, for example, software clock objects to represent media timelines (cf. dvbcss-clocks library).

Local Resynchronisation Strategies

Different strategies can be applied to reduce the asynchrony to imperceptible levels at the devices such as changing the playback speed or skipping media samples/frames. However, it is likely that each strategy will have some impact on the user's QoE (Quality of Experience) and the disruption may vary across media types and programme genres (29). Therefore, the strategy or combination of strategies for resynchronising the media player(s) should be selected so as to minimise the impact on the user's QoE.

Two possible strategies are:

- **Adaptive Media Playback and**
- **Media-Frame Skipping**.

The Adaptive Media Playback (AMP) function involves adjusting the media player's playback speed to reach the desired media head position after a finite period of time whilst incurring an acceptable degradation in QoE for the companion screen application user.

It is possible that the asynchrony value measured from the media player is sufficiently large that unless a radically-large value for playback speed is selected, the media player will take a long time to become synchronised with the Media Timeline Clock. It is preferable in such a case to use a playback adaptation strategy that skips forwards or backwards (a seek operation) to reduce the duration of the disruption. Although skipping is perceptually detrimental to the user experience, it is more effective to correct severe asynchronies quickly.

When a media player skips to an unbuffered point in the media stream, a further delay is introduced since the player has to fetch and buffer the required media frames. The MFS strategy therefore has to estimate the buffering delay $T_{bufferdelay}$ and preemptively account for it when requesting the media player to skip media frames. This buffering delay need only be factored in if the new seek position lies beyond the currently buffered media data.

Temporal alignment of media objects in production/ DMapp creation

The scheduling of starting/ending DMapp components can be made with respect to the DMApp-crafted timing or with respect to the timing of live events. In either case, time-aligning media streams simplifies creating temporal relationships between them for synchronised presentation at the consumer end. This function is made easier if all capture devices are time-synchronised in the first place.

Assuming all cameras, microphones and sensors are capturing in the same physical environment, time-alignment of the different sources can be achieved via the captured content, such as a physical event captured by all the different sensors. If physical objects are not visible to all the cameras, as when users point mobile phones in different directions, audiotracks cross-correlation can be used to time-align the video stream (cf. COGNITUS project).

The next step in creating these temporal relationships between the different media sources is to record cross-stream correlation timings during production and/or DMapp creation. These correlations can be recorded if the media streams carry timing metadata such as timestamps or sequence numbers in the form of a timeline. It is a common occurrence in play-out systems to have timelines signalled in raw media streams, even from the point of capture, as in SDI stream formats. The challenge is for these timelines to persist after stream transcoding (e.g. encoding to H264 and MPEG2-TS) and for the recording of cross-stream correlations to still be relevant when the streams are decoded at the receiver. Furthermore, the mode of delivery for the streams may be either via broadcast or broadband; candidate timelines for synchronisation need to be supported in encoding formats for both types of streaming. A suitable candidate is TEMI timeline injection into MPEG2-TS. TEMI support in media encoders ensure that the timing information injected in the input streams are not affected by the encoding process. The TEMI-carrying MPEG2 transport streams can be used to create DVB-T2 (broadcast) and MPEG-DASH (HTTP streaming) streams. It has been shown in (30) that this mechanism of signalling timing can be used to synchronise broadcast and IP-delivered streams.

3.6.2.10 DMApp Runtime

The DMApp Runtime is responsible for running the DMApp on each host device. Due to the heterogeneity of host device platforms (e.g. Android, iOS, HbbTV 2.0, laptop/PC browser environments), it needs to adapt to its execution environment and device capabilities. For example, on an HbbTV 2.0 device, it can leverage the Media Synchroniser API and App-2-App framework to

exchange timing information and application-messages with companion screen devices. It is the component that is ultimately responsible for overseeing the loading and running of DMApp Components in the DMApp. It does so through interaction with the Timeline Orchestration FG (pushing DMApp state such as start-time, DMApp status) and by listening to commands from the Layout Orchestration FG (for DMApp component (de-)activation and positioning in the UI.)

The functions of a DMApp Runtime can be enumerated as follows:

- **DMApp initialisation and service setup:** coordination of DMApp and context initialisation.
- **DMApp component interface and base functionality:** this forms the framework for all DMApp components. DMApp components can be loaded from external sources as necessary.
- **Receiving and actuating state updates from services:** state updates from the Layout FG and Timeline FG via multiple channels are processed, filtered and applied by the DMAppRuntime.
- **State reporting to other FGs:** updates to key state items, in particular clock changes and component status changes, are reported to the Layout and Timeline FGs.
- **Device discovery and advertisement of DMApp launching parameters:** On communal devices, the DMAppRuntime should enable a device discovery mechanism to allow discovery by personal devices. Possible protocol-choices for this function are DIAL, mDNS, Bonjour. DIAL, in particular, is the standard mechanism in HbbTV 2.0 devices. The DIAL solution also provides a mechanism to share an app-launch API. Companion devices can automatically discover TV devices on the local network and join existing sessions.
- **Clock synchronisation and management**: between devices by means of DVB-CSS, the app-to-app channel, the shared state service and/or the cloud sync service, and between components/modules on each device. Clock sync over app-to-app: when DVB-CSS sync is not available
- **Local layout and region management**: local positioning of components and reporting of layout region information and updates to the layout service.
- Layout region initialisation and management
- **State signals, inter- and intra-device data management and binding**: Intra-client communication and signalling propagation of data to/from/via: layout service, shared state service, DMApp components, inter-device app2app channel, internal client-API modules/functionalities, and other data endpoints, as necessary.
- **Media playback functionality**: the client-API includes a generic media player DMApp component.
- **Remote procedure call functionality**: DMApp components, sub-resources thereof, and other client-API resources can be directly addressed by other local or remote components or other client-API hosted entities.
- **Logging**: collected logs are annotated such that they can be attributed to the source DMApp component, module, or sub-section thereof.
- **Platform-specific functionality**: for Android, iOS, emulated TV and emulated companion platforms.
- **General software utilities**: the client-API contains various generic utility functionality intended for the use of DMApp component and DMApp authors.TODO: Jonathan to fill out section
- **General debugging and diagnostics framework**: various debugging, introspection, monitoring, fault-detection, fault-correction, reporting and other developer utilities.

A **DMAppRuntime** is configured with a set of parameters which include the following:

- DeviceID
- Device State namespace/scope
- DMappComponents
- Device Layouts (from Layout Orchestration)
- LayoutDocument URL (from DMApp EPG)

- TimelineDocument URL (from DMApp EPG)
- State signals (events) to bind listeners to (from DMApp description)
- DMApp variations: alternative TimelineDocuments (from DMApp EPG)
- Platform-deployment variations: edge, test (shared from launcher app on first device)
- Logging variations: (Analytics on or off, shared from launcher app on first device)

3.6.2.11 DMApp State Management

DMApps are inherently stateful. DMApp state comprises of

1. **context state** (identifiers, device membership),
2. **orchestration state** (device roles, layout, timeline, bandwidth, timing),
3. **DMappComponent state** (timing, status, execution state) and
4. **DMApp user state**.

This state changes dynamically throughout the execution of the DMApp.State is created as soon as a DMApp is launched on a device:

1) The DMApp Runtime on the device is configured with DMApp configuration parameters (e.g., DMApp name, DMApp image URL, DMApp join text, UI layout region definitions, media object origin server URL, WallClock source, etc.)
2) a context identifier is generated for that DMApp instance,
3) user-device pairing is recorded, and authentication tokens are issued to that device for platform access.
4) for non-live DMApps, the DMApp start time (on the WallClock) is recorded and notified to the DMApp orchestrator.

As additional devices are onboarded, they use discovery mechanisms (e.g. DIAL) to find the first device and exchange DMApp launch metadata, context data and timing resources. They also obtain from the device, the DMapp definition, the list of event names (or signals) the device (via its DMApp Runtime) can subscribe to and actuate on.

Throughout the DMApp execution, the following DMApp state changes dynamically:

1) the current deployment of DMApp components (they can change hosts e.g. user moves video to main screen),
2) the status of DMApp components (e.g. loaded, started, finished, unloaded),
3) DMapp component playback progress (e.g. current time on supplied Clock)
4) user-generated experience data (e.g. user score)
5) bandwidth quota for a DMApp component as indicated by the bandwidth orchestrator
6) Performance-, fault- monitoring data from various system components are logged

In addition to the different types of state, DMApp state has different scopes. The scope boundaries can be geographical: *device*, *LAN*, *distributed* or logical: *context*, *DMApp component*. Whilst device and network boundaries can be natural scope barriers, a system to manage and share state in multi-device DMApps must provide a scoping or namespace mechanism that prevents uncontrolled access to all DMApp state.

Distributed Shared State

In multi-device DMApps, state is not only distributed but also shared. Although, it is possible to centralise all state, state is still held locally on devices and modified during the DMApp execution;

this state, if shared, will be replicated at the parties who have an interest in observing changes and acting upon these.

To ensure that all devices and services have a consistent view of the distributed shared state, data synchronisation is required; this is challenging in multi-device applications especially when shared, server-side objects may be modified at any time and some level of consistency guarantees must be ascertained. Whilst strong consistency guarantees are beneficial to DMApp execution, enforcing these may require successive rounds of data exchange to achieve consensus. In DMApps (as in online games) where the presentation timing is crucial to the QoE, one must trade off consistency guarantees for low-latency shared state updates. For example, 2 phase-commit/distributed transactions to achieve strong data consistency is not an option for many applications. Low latency shared state updates, quick state-change signalling to observing clients and eventual consistency may be the implementation directive for a DMApp state management system.

LAN-local State Signalling

Centralising state management via a cloud-based service offers a simple approach to persist state for protection against system failures and for priming joining devices with the most up-to-date state. However, in DMApps that are purely state-driven (as opposed to driven by linear timelines) and whose execution only span a LAN, the communication with a cloud-hosted state synchronisation service may be an undesirable overhead. In the Theatre-at-School DMApp, for example, execution of the DMApp progresses via the teacher deciding when to activate particular sub-lessons (i.e. the DMApp is state-driven) and the DMApp context spans only the classroom. State update synchronisation mechanisms in these cases should favour LAN-local messaging to achieve low-latency updates and state-change notifications.

State-Signal-to-Component Binding

The shared state mechanism used can include support for multi-device data-binding schemes for application developers. This delivers a more declarative style of programming and reduces the amount of bespoke application logic required. This function insulates the DMApp Components from specifics of the shared-state method by providing an abstract programming interface for binding listeners to state-signals.

3.6.2.12 DMApp Launching & On-boarding

The DMApp Launching & Onboarding FG encompasses functions related to user enrolment to the DMApp platform, pairing of devices to user accounts, launching of DMApps on the host device and discovering other devices to onboard in the running DMApp on the local network. The FG is responsible for associating the actual devices with the device roles prescribed for the DMApp. The FG provides a UI that allows users to select the DMApp to join and the role the device will fulfil in the DMApp. It should also ensure that all mandatory device-roles in the DMApp are fulfilled (possibly with interaction with the Timeline Orchestration FG). As functions will be required to run on multiple device platforms (TV Emulator, Chromium and Cordova), cross-platform support is necessary.

The functions in the DMApp Launching & Onboarding FG include the following:

- **User Enrolment:** User signup/sign-in/sign-out
- **Device Discovery:** Find available devices that we can enrol (through pairing/association) in a session or that we can interrogate for existing sessions (multiple strategies, technologies and bridges between technologies exist)

- **Device communication:** communication protocol between discovered devices and devices associated with a session
- **Capability Discovery:** Find available devices by function
- **Association:** Dynamic association of devices with a specific session (context enrolment - join/launch, remote enrolment, notifications)
- **Session Discovery:** Discovery of running sessions (currently enumerated via DIAL, Editor service + hardcoded programmes, or via pre-agreed session code for inter-home)
- **Session Creation:** Launch of a new programme/session
- **Multi-device Authentication:** Secure platform access for devices by linking them with user identities. Provides a means of securely acquiring authorised tokens.

3.6.2.13 Security FG

Authentication, authorisation and access control functions are needed, as in most platforms, to control access to resources and functionality. For example, all platform clients need to be identified uniquely and associated with user accounts. Only legitimate clients should be given access to platform services or interact with and trust other peers. Dedicated authorisation functions are usually used to achieve this. Media streams should also be protected from illegitimate access. DRM solutions for media streams abound in the industry and a survey of them is beyond the scope of this document.

In summary, functions in the Security FG include the following:

- **Identity management:** Account creation, access token invalidation and expiration, admin user interface
- **Defined User Roles:** Admin, Demonstrator, User, Experience Admin, Debug
- **Authentication of users:** Credential verification, multiple forms of authentication could be used, including those that support Multi-Factor Authentication (MFA).
- **Authentication of applications.** If an application invokes other services, OAuth 2.0 refers to it as a "Client". This is true regardless of whether the application is running on an end user device or if it is a server-based application.
- **Generation of tokens that assert identities and granted scopes.**
- **Provisioning and tracking of granted scopes.** Specific applications and users can be granted access to specific features / resources.
- **Device account pairing**: Securely granting access credentials to remote devices
- **Assign unique ids to nodes**
- Assign unique ids to users
- An alternative **rendezvous mechanism** to exchange experience launch payload (when clients contact the auth function with the same DMApp and context identifiers). This can simplify DMApp launching on joining devices

3.6.2.14 Management FG

The Management FG comprises of an assortment functions related to the configuration and management of the DMApp as well as the platform.

For instance, during DMApp execution, the DMAppRuntime and the client application hosting it, need to be configured with the following (non-exhaustive) list of parameters for correct operation:

- a URL for the DMApp base web app (assuming that a generic app loads/runs bespoke web applications for each DMApp[9])

[9] This is purely an implementation strategy decision: it is entirely possible for DMApp developers to build custom native Android/iOS apps; these will need to include the DMAppRuntime functionality. For the purpose

- host device mode (TV, companion, TV emulator or companion device emulator),
- default URLs for services (e.g. the origin server or platform service endpoints),
- location of Layout and Timeline documents,
- UI region declarations for the particular device,
- the root layout region selector name,
- a base URL for loading styling media assets e.g. menu icons, images, CSS documents, etc.
- optional data to include in device advertisements during discovery (this is used to onboard new devices)

In terms of the actual DMApp platform, irrespective of its deployment setting (whether cloud-based, stand-alone), the dynamic nature of the deployment environment requires that

- configuration information about the platform services is maintained and synchronised across all deployment environments (e.g. each cloud-based host),
- platform capabilities are fulfilled by starting service instances on different hosts and their deployment coordinated[10]; it is likely that a single capability will have multiple service instances fulfilling it in the interest of load-balancing, performance via horizontal scaling and robustness,
- service-names are resolved to particular instances of a platform capability
- the health of platform services be monitored and services be restarted if necessary.
- A web UI is available for platform-admins to check the status of the services. The information displayed is obtained via the Rancher REST API.

In terms of timing, the platform requires that a global timing source (usually a monotonic clock) be provided for

- time synchronisation on presentation devices,
- time-alignment on capture devices use by media encoders for
- timeline signalling in media streams

Thus, the management FG also includes a global reference time source selection function.

Logging and monitoring provides functionality for the creation, aggregation, processing, storage and presentation of information relating to services and DMApps. This information may be used to monitor the health of these services and applications, to identify errors and bugs during operation or to provide a detailed understanding of how they are being used.

- **Message creation:** Services and DMApps create log messages in accordance with a pre-defined schema, containing sufficient information to identify the source of the message, a timestamp from a global clock and information about the context in which it was created.
- **Message aggregation:** Log messages from different sources are received by a single service which may include a processing pipeline to drop unwanted messages, enrich them with additional information (eg. geolocation data), correlate them with other messages (e.g. linking multiple devices within a session) or change their formatting.

---

of reuse, consistency and compliance to open standards, the 2Immerse project decided to follow the web component approach.

[10] In cloud-based platforms, containers provide lightweight OS virtualisation solution for packaging code and dependencies together (including system dependencies). Systems such as Kubernetes automate the deployment, scaling, and management of these containerised applications.

- **Indexed data storage:** A suitable database solution is used for efficient storage, indexing and lifecycle management of log data (including archiving).
- **Data querying:** Aggregated log data is retrieved and filtered using an appropriate query language, paired with a user interface.
- **Data visualisation:** Aggregated log data may be used to create graphs and other visualisations to help interpret performance or usage information. Graphs may be created retrospectively or updated in near real-time as new log messages are received.

3.6.2.15 Origin Server FG

Basic HTTP/S based service used to host client applications, DMApp Components and their assets, timeline and layout documents, and media.

A basic asset structure may be required for organizing these assets on the origin server.

In 2Immerse, the origin server was implemented using an AWS S3-backed web service serving as the origin, proxied by CloudFront for scalability and registered using Route53 for indirection.

3.6.2.16 Timeline Editor FG

For the purpose of the reference architecture we will only consider the role of the editor in allowing a director to modify the live experience of viewers while the experience is running.

Timeline editing is based on the following concepts:

- **Editable timeline:** a superset of the timeline document for the timeline service, including a number of timeline event templates.
- **Timeline event template:** an authored snippet of timeline that can be inserted into the timeline after supplying parameters such as start time and duration.
- **Timeline edit:** the operation of inserting such a snippet into the running timeline after supplying the parameters.
- **Client:** for the editor, a client is the complete set of components and services that play back an experience in a single home: context, DMApp, timeline and layout instances, client-api instances, etc.
- **Preview Client:** the client used by the director for previewing their edit operations.

The Timeline Editor functions include the following:

- **User Interface:** the editor should provide the director with a list of currently active event templates and allow them to instantiate these, as well as a list of already instantiated events and possibly allow them to be removed.
- **Timeline Document Management:** the editor will serve timeline documents to newly started clients, so these clients will have an up-to-date view of the experience as it is now, including all edits that have already been made.
- **Timeline Modification Forwarding:** the editor opens a websocket to broadcast timeline edits to all clients that are playing the current experience.
- **Timeline Modification Retrieval:** Timeline edits are sequenced, so client timeline service instances are aware when they have missed any edits. The editor will supply the missing edits on request.
- **Notification Management:** the editor listens to a websocket that the preview client timeline server instance sends state and clock updates to. The editor uses these to update the user interface, and to determine the correct clock value at which to insert timeline edits.

- **Preview Player Control:** if the editor is running in as-live mode it provides the producer with a player control panel to pause, resume and position playback. These commands are forwarded to the client-api in the preview player.

3.6.2.17 Data Playback FG

Data playback is the collection name for services capturing and distributing non-audio and non-video information related to the 2-immerse experience. The information can be used on client side as data driven graphics and or as content driver on companion devices. The type of this information is widely different depending on production or event. In for example MotoGP the information would mainly consist of timing data and in football mainly game scouting events.

- **Data source:** External information source for production related information. I.E. scouting data or timing data and similar.
- **Block of information:** State shift or event related to a specific time.
- **Subscribed category:** Information relating to a specific subject.

Functions include the following:

- **Manage different data sources**: Provide public API to the data storage, and or micro service adapters for existing data sources.
- **Persisting block of information:** Information available before, created during and available after a certain production event is time stamped in relation to the production time. Every data sample is stored for later playback retrieval.
- **Send live information**: For live productions, live captured data is relayed directly clients.
- **Respond to request of data**: For in live catch-ups and VOD situations client can request missing information.

3.7 The Architecture View

Once the concepts and functions have been saturated in the iterative analysis, the Functional View is transformed into the Architecture View. The architecture view presents the structural aspects of the DMApp-RA; It, describes how the general functions from the Functional View are organised into architectural services and how these functions are accessed. Although the Architecture View must be kept relatively abstract, some architectural design patterns are applied. For communication and interaction between the functional elements, particular interaction styles are recommended. This section builds upon the Functional View section to propose an abstract Architecture View for the DMApp Reference Architecture. Particular structural abstractions and interaction paradigms are suggested as general design patterns to include in the DMApp-RA; implementation patterns are precluded from the DMApp-RA, leaving adopters to fix these aspects when generating concrete architectures for their DMApp platforms. The DMApp-RA Architecture View is shown in Figure **7**.

3.7.1 Architectural Abstractions and Patterns for Functions/Function Groups

3.7.1.1 Microservices Architecture Pattern

Without loss of generality (as to where the functions are deployed and how they are invoked), it is possible to apply structural patterns to promote functionality cohesion and decoupling, as a means to manage the complexity of platforms. For functions that need to be accessed by other elements, we propose the use of microservice abstractions to encapsulate them.

Microservices are useful abstractions for building modular platforms and each microservice typically implements a set of distinct function. Each service has a well-defined boundary in the form of an RPC- or message-driven API. The Microservices Architecture pattern enforces a level of modularity that in practice is extremely difficult to achieve with a monolithic codebase. Consequently, individual services are much faster to develop, and much easier to understand and maintain. The caveat of this pattern is the potential for data duplication as each microservice maintains its own database[11]. More complex data management is the trade-off for this flexibility, especially if we want to achieve a level of data consistency approaching the ACID properties of more centralised relational schema-based solutions. It is possible to use events to implement transactions that span multiple services. A transaction consists of a series of steps with the opportunity to commit or rollback the changes implied by it. Each step consists of a microservice updating an entity and publishing an event that triggers the next step (31).

The DMApp-RA does not define implementation mechanisms for microservices and avoids making any assumptions in terms of a deployment topology. The microservices can be deployed as

- containerised services hosted on a cloud-platform, or
- services running on a dedicated device in the living room (e.g. compute sticks, set-top boxes), or
- server-less microservices (cf. AWS Lambda) that are submitted to a compute-provisioning platform
- a completely decentralised server-less topology with each device running a full set of microservices and each microservice interacting with its counterpart on another device in a P2P manner

The decision as to map a DMApp-RA FG into one microservice or number of them is best left to platform developers. They can use best implementation practice guidelines to decide on the granularity of the microservices and the functions they cover. A possible micro-servce based architecture is suggested in Figure 8. However, this should only be taken as one of many possible design alternatives.

3.7.1.2 Additional Infrastructural Services

API Gateway

Although it is possible for clients to directly invoke microservice APIs, there is often a mismatch between needs of the client and the fine-grained APIs exposed by each of the microservices. An API Gateway for distributed services provides single entry point into the system It may also have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling. The API Gateway handles some requests by simply routing them to the appropriate backend service. It handles other requests by invoking multiple backend services and aggregating the results.

Service Discovery

Because the microservices can be potentially distributed across different hosts, clients need to be aware of the microservice instances network location for service invocation. Server-side deployments require extra services such as service registrators to maintain a database of microservice locations. On-site deployments, on the other hand, can use service discovery protocols (e.g. SSDP, Bonjour, DIAL, mDNS) to enable services to advertise their locations to clients on the network.

[11] Data access becomes much more complex when we move to a microservices architecture. That is because the data owned by each microservice is private to that microservice and can only be accessed via its API. Encapsulating the data ensures that the microservices are loosely coupled and can evolve independently of one another.

3.7.2 Communication Paradigms and Interaction Styles

The communication model in the DMApp-RA supports communication paradigms for connecting elements in the DMApp domain. The communication between elements needs to support different paradigms: unicast is the mandatory solution for one-to-one connectivity and multicast and anycast are needed for fulfilling many other platform requirements, such as data collection and state-signalling, etc.

The DMApp-RA Architecture View (shown in Figure **7**) includes an **Event Bus** as a communication pattern for the interaction between distributed elements, services and clients. Although, the event bus implies an asynchronous, messaging-based mechanism and its support for the different communication paradigms, the DMApp-RA leaves the interaction style to be fixed by platform developers.

When selecting an IPC mechanism for a service, it is useful to think first about how services interact. There are two dimensions to the possible interaction styles. The first dimension is whether the interaction is **one-to-one** or **one-to-many**:

- One-to-one – Each client request is processed by exactly one service instance.
- One-to-many – Each request is processed by multiple service instances.

The second dimension is whether the interaction is **synchronous** or **asynchronous**:

- Synchronous – The client expects a timely response from the service and blocks while it waits.
- Asynchronous – The client doesn't block while waiting for a response, and the response, if any, isn't necessarily sent immediately.

The following table shows the various interaction styles.

	One-to-One	One-to-Many
Synchronous	Request/response	—
Asynchronous	Notification	Publish/subscribe
Request/async response	Publish/async responses	

Table 6: Interaction styles for inter-FG communication in DMApp-RA

Each service typically uses a combination of these interaction styles. For some services, a single RPC mechanism is sufficient. Other services might need to use a combination of IPC mechanisms.

Several IPC technologies are available to implement these interaction style and the selection of these is left to the platform developer. These technologies are usually based on the following mechanisms:

- Asynchronous, Message-Based Communication (e.g. AMQP, STOMP): Processes communicate by asynchronously exchanging messages. A client makes a request to a service by sending it a message. If the service is expected to reply, it does so by sending a separate message back to the client. The communication is asynchronous, the client does not block waiting for a reply. Instead, the client is written assuming that the reply will not be received immediately. Reactive programming models are useful abstractions to manage code complexity here.

- Synchronous, Request/Response IPC (REST, Apache Thrift, gRPC): a client sends a request to a service; the service processes the request and sends back a response. In many clients, the thread that makes the request blocks while waiting for a response. Other clients might use asynchronous, event-driven client code that is perhaps encapsulated by Futures or Rx Observables.
- Message formats (Protocol Buffers, Apache Avro): when using message-based IPC mechanisms (e.g. AMQP, Thrift, gRPC), it is desirable to use a cross-language message format; microservice implementations and clients may be written in different languages. Binary message formats are less human-readable as text formats but offer more efficient bandwidth utilisation.

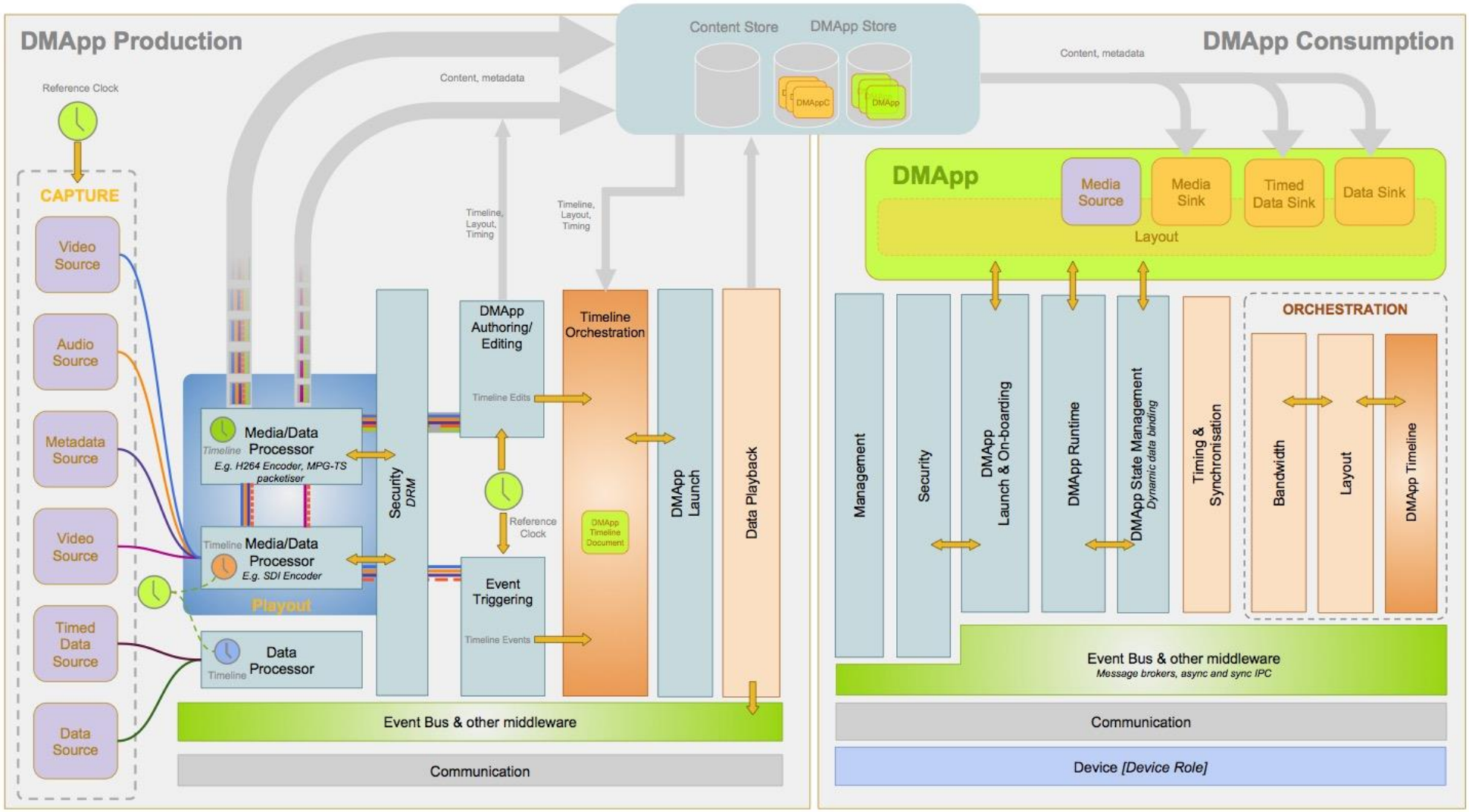


Figure 7: DMApp-RA Architecture View

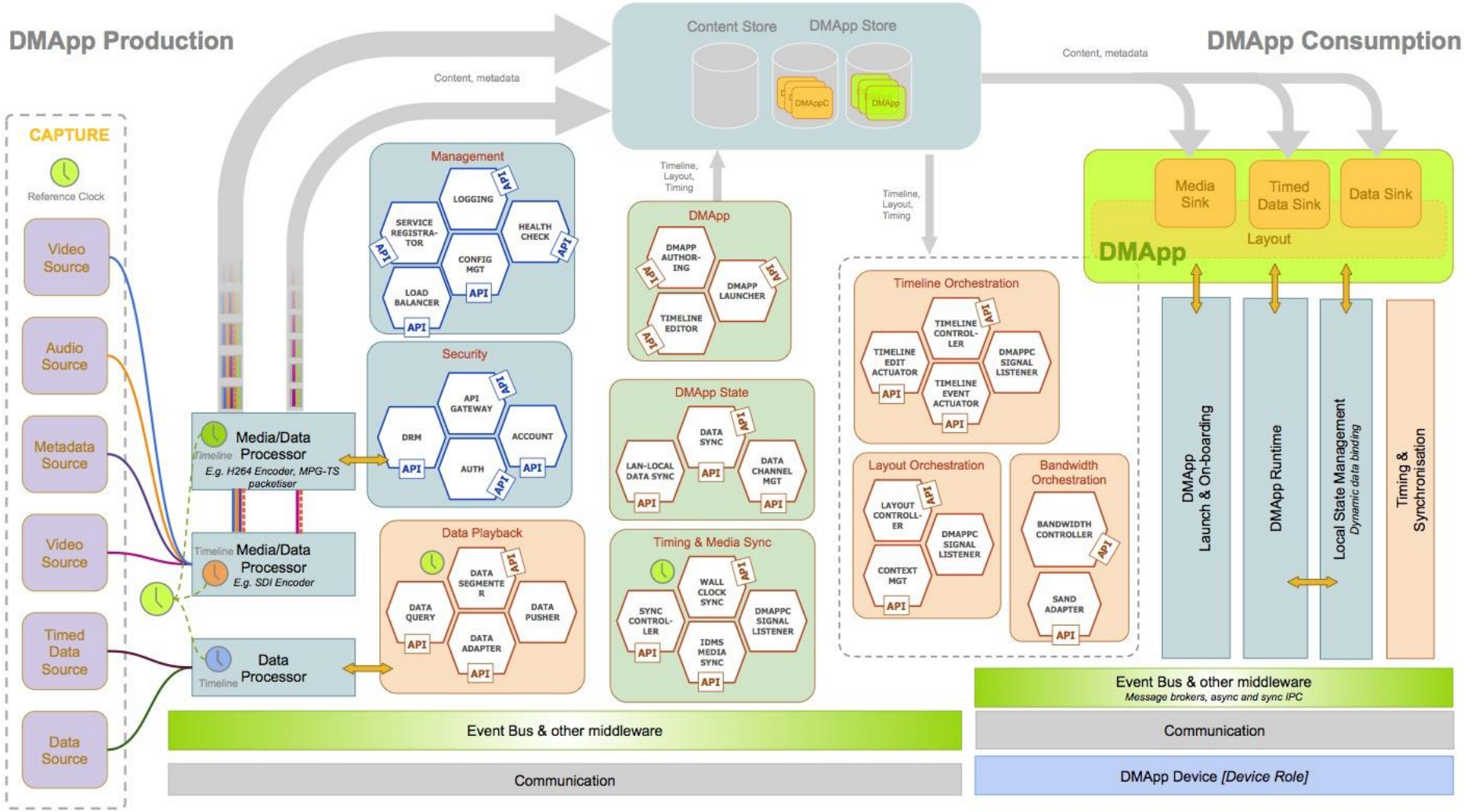


Figure 8: A possible Microservices Pattern application to the Architecture View

3.7.3 DMApp-RA Function Example Implementations

2Immerse software components provide example implementations of the DMApp-RA elements. Table **7** Provides a mapping between the 2Immerse open-sourced software components and the DMApp-RA Function Groups. The table also specifies the component's coverage of the functions.

Table 7: Example implementations of DMApp-RA elements

DMApp-RA Functional Element	Example implementation (Open-Source Repository)	Functions	Community Manager
DMApp Runtime	client-api	DMAppRuntime functions implemented in JS and packaged to run on Android, iOS, HbbTV emulators and HbbTVs **DMApp Init, DMAppC base impl, Timeline client, Layout client, Clock creation & sync, DMAppC status publisher**	BT
DMApp Launch & Onboarding	launcher	Multi-experience launcher web application for TVs and companions scoping user enrolment, device discovery, device association, multi-device authentication, session discovery and session creation. **DMApp EPG, Device-role association, Device Discovery, App Launch/Stop, Auxiliary data payload**	BBC
DMApp Launch & Onboarding	android-unified-launcher	Cordova applications to wrap unified-launcher web application as a native app for Android.	BBC
DMApp Launch & Onboarding, Timing & Media Synchronisation	hbbtv-lib	A library of modules for common HbbTV functionality **HbbTV App launch, Inter-device media sync for HbbTV devices**	IRT
Timing & Media Synchronisation	dvbcsstv-lib	The JS library for the dvbcss browser proxy (client-api dependency) DVB-CSS protocols for HbbTV Emulation **Inter-device media sync for HbbTV Emulators, HbbTV App-2-app comms.**	BBC
Timing & Media Synchronisation *Inter-device media sync for mobile devices*	synckit	A JS library API to enable synchronisation of media as directed by HbbTV Media Synchroniser (a client-api dependency). Android DVB-CSS protocol-clients **Inter-device media sync for Android**	BBC
Timing & Media	sync-protocols	JS library implementing client and server protocols for media synchronisation	BBC

Synchronisation		between TVs and companion screen applications via DVB CSS **Inter-device media sync for iOS, HbbTV Emulators**	
Timing & Media Synchronisation *Clock abstractions*	dvbcss-clocks	software "clock" objects in JS. Used to model media timelines, timing clocks and Timeline Shadows (for cloud-sync IDMS) **Sync timeline local clock**	BBC
Timeline Orchestration	timeline-service	Service to orchestrate timing and events **Timeline model, timeline controller, timeline event actuator, timeline edit actuator, sync clock selector**	CWI
Layout Orchestration	layout-service	Service to orchestrate layout of content for a multi-screen experience. **Layout model, layout controller, Context Management**	CISCO
Bandwidth Orchestration	bandwidth-orchestration	Component Bandwidth Orchestration Service. **Bandwidth model, Bandwidth controller, Bandwidth allocation policies**	CISCO
Bandwidth Orchestration	bandwidth-orchestration-client	Client for the bandwidth orchestration service. This client contains the SANDPlayer module that manages a Dash.JS player and send statistics to the BOS. **Bandwidth usage collection, Bandwidth allocation dissemination**	CISCO
DMApp State Management Middleware	shared-state-service	This is a fork of the MediaScape shared state service, licensed under the Apache License, Version 2.0. **Data synchronisation, scopes, change notifications**	
DMApp State Management	shared-state-client	Client code from the Mediascape SharedState repository organised as a npm package.	BT
DMApp State Management *LAN-Local State Signalling*	client-api	State change notification over App-2-App channels **Data synchronisation, scopes, change notifications**	BT
Middleware	websocket-service	Service to support service->client push communications **scopes, pub-sub message delivery via WebSockets**	CISCO
Timing & Media Synchronisation *Reference Clock*	wallclock-service	A lightweight time (WallClock) synchronisation service for frame-accurate synchronised experiences. **Time synchronisation, WallClock**	BBC

Timing & Media Synchronisation	cloud-sync	Media Synchronisation Service and JS client library. Service comprises of a set of microservices. A demo-app hosting microservice is also provided. **Inter-destination Media Synchronisation**	BBC
Security	auth-service	Authentication service **User identity management, user role management, access token & scope mgt, device account pairing**	CISCO
Security	auth-admin	Admin interface for the auth-service	CISCO
Management	renderer	Realtime multi-device layout visualisation tool for the layout service **DMApp monitoring**	CISCO
Management	logging-service	A lightweight service which flattens a JSON structure sent via HTTP POST and pushes it to stdout, where it is collected and sent to Logstash and the rest of the ELK stack. **Log message creation, log aggregation, log storage, log queries and visualisation**	CISCO
DMApp Authoring & Validation	2immerse-editor	Platform for creating 2IMMERSE presentations in the browser **DMApp Authoring & Validation**	CWI
HbbTV Emulator	system-images	HbbTV2.0 emulator firmware for the Intel NUC **HbbTV Emulation**	BBC

4 Conclusion

This report has presented a public proposition of the 2Immerse project to the community of users in the distributed orchestrated media application domain. This proposition is two-fold.

In order to enable the community to run and experiment with DMApps, a stand-alone version of the platform deployment has been made publicly available with usage instructions. The platform is a collection of useful functions and the function implementations (services, components, libraries) are themselves valuable resources for DMApp community developers. To help encourage similar DMApp building initiatives as 2Immerse, the consortium has publicly released all the software components required to run the platform services, create DMApp components and run authored DMApps. This report has provided a listing of all software components made open-source and the locations of the code repositories. This is the culmination of concerted efforts within the consortium to prepare code repositories (e.g. documentation) for public release.

Another aspect of the public proposition is a reference architecture for distributed orchestrated media application platforms. The report documents the derivation process of the reference architecture and presents the core functions, structures and their respective elements, and the interaction between these elements. The reference architecture called DMApp-RA, represents a good template for instantiating concrete DMApp-platforms and whilst it leaves implementation choices to the RA-adopters, it suggests possible solutions and mechanisms for building the functions. The reference architecture is also a useful guide to the 2Immerse project open-source contributions. Since it describes platform functions at an abstract level, it allows users to understand how the platform components fit within a DMApp system of systems.

5 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. MPEG Media Orchestration. *MPEG .* [Online] [Cited: 15 November 2018.] https://mpeg.chiariglione.org/standards/mpeg-b/media-orchestration/wd-isoiec-23001-13-media-orchestration-more.

5. Presentation API. *W3C.* [Online] 1 June 2017. [Cited: 12 November 2018.] https://www.w3.org/TR/presentation-api/.

6. DVB-CSS. *DVB.* [Online] 03 August 2017. [Cited: 10 November 2018.] https://www.dvb.org/standards/dvb_css.

7. BBC Taster. *The Vostok-K incident.* [Online] 2018. [Cited: 01 December 2018.] https://www.bbc.co.uk/taster/pilots/vostok.

8. Cisco. *The Future of TV: Coming Soon to a Wall Near You.* [Online] 12 February 2013. [Cited: 15 October 2018.] https://blogs.cisco.com/news/the-future-of-tv-coming-soon-to-a-wall-near-you.

9. W3C. *Synchronized Multimedia Integration Language (SMIL 3.0).* [Online] 01 December 2008. [Cited: 15 November 2018.] https://www.w3.org/TR/SMIL3/.

10. BBC R&D. *Object-Based Media Toolkit.* [Online] 2017 December 01. [Cited: 01 November 2018.] https://www.bbc.co.uk/rd/projects/object-based-media-toolkit.

11. MediaScape. *GitHub.* [Online] [Cited: 01 Novemver 2018.] https://github.com/mediascape.

12. 2Immerse. YouTube 2Immerse Channel. *YouTube.* [Online] [Cited: 10 January 2018.] https://www.youtube.com/channel/UCpGa5NU1Bbj8Nkz0vZi7IwA.

13. Reed, Paul. Reference Architecture: The best of best practices. *IBM Developer.* [Online] 15 September 2002. [Cited: 14 December 2018.] https://www.ibm.com/developerworks/rational/library/2774.html.

14. Richardson, Chris. Pattern: Server-side service discovery. *microservices.io.* [Online] 2015. [Cited: 15 October 2018.] https://microservices.io/patterns/server-side-discovery.html.

15. Moreno, Marcio Ferreira and Costa, Romualdo M. de R. Specifying Intermedia Synchronization with a Domain-Specific Language: The Nested Context Language (NCL). [book auth.] Mario Montagud, et al. *MediaSync Handbook on Multimedia Synchronization.* s.l. : Springer, 2018.

16. Bulterman, Dick C. A. SMIL: Synchronized Multimedia Integration Language. [book auth.] Mario Montagud, et al. *MediaSync Handbook on Multimedia Synchronization.* 2018.

17. *Enhancing MPEG DASH Performance via Server and Network Assistance.* Thomas, et al. 126, 2017, Vol. SMPTE Motion Imaging Journal.

18. *Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive.* Jiang, J., Sekar, V. and Zhang, H. s.l. : ACM CoNEXT, 2012.

19. *Client-Driven Network-level QoE fairness for Encrypted 'DASH-S.* Chen, Junyang, et al. s.l. : ACM Internet-QoE '16, 2016. DOI: https://doi.org/2940136.2940144.

20. 2Immerse. *D2.4 Distributed Media Application Platform and Multi-Screen Experience Components: Description of Second Release.* s.l. : 2Immerse, 2018.

21. *Approximation of the Quadratic Knapsack Problem.* Taylor, Richard. 1, January 2016, Operations Research Letters, Vol. 44. http://dblp.uni-trier.de/pers/hd/t/Taylor:Richard.

22. *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.

23. *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.

24. *Exact Solution of the Quadratic Knapsack Problem. Informs Journal on Computing.* Caprara, Alberto & Pisinger, David & Toth, Paolo. s.l. : INFORMS, October 1998, INFORMS Journal on Computing.

25. HbbTV. *HbbTV 2.0.1 Specification with Errata #3 Integrated.* [Online] [Cited: 05 October 2018.] https://www.hbbtv.org/wp-content/uploads/2018/02/HbbTV-SPEC20-00058-004-Errata-3-to-TS-102-796-V141.pdf.

26. *The media state vector: a unifying concept for multi-device media navigation.* Arntzen, Ingar M., Borch, Njål T. and P. Needham, Christopher. s.l. : 5th Workshop on Mobile Video (MoVid '13), 2013.

27. *Design and Simulation of a Distributed Control Scheme for Inter-destination Media Synchronization.* Montagud, M., Boronat, F. and Stokking, H. s.l. : IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013.

28. *Self-Organized Inter-Destination Multimedia Synchronization For Adaptive Media Streaming.* Rainer, Benjamin and Timmerer, Christian. s.l. : 22nd ACM international conference on Multimedia (MM '14), 2014.

29. *QoE-aware inter-stream synchronization in open N-Screens cloud.* Mu, Mu, et al. Las Vegas : 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), 2016.

30. *HbbTV-Compliant Platform for Hybrid Media Delivery and Synchronization on Single- and Multi-Device Scenarios,.* F. Boronat, D. Marfil, M. Montagud and J. Pastor. 3, s.l. : IEEE Transactions on Broadcasting,, 2018, Vol. 64. doi: 10.1109/TBC.2017.2781124.

31. Event-Driven Data Management for Microservices. *NGinx.* [Online] NGinx. [Cited: 01 December 2018.] https://www.nginx.com/blog/event-driven-data-management-microservices/.

32. Ramdhany, Rajiv. *Dynamic deployment and reconfiguration of ad-hoc routing protocols.* Lancaster : Lancaster University, 2011.

33. Bassi, Alessandro, et al. *Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model.* s.l. : Springer, 2016. ISBN:3662524945 9783662524947.

34. 2Immerse. *D4.4 Prototype Service Descriptions – Second Update.* 2018.

35. *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.

36. 2Immerse. *D3.3 User Interaction Design: the development of generic components & features to inform MotoGP Service Trials, Production Tools, and OnBoarding.* 2017.

37. mantl.io. *mantl.io.* [Online] [Cited: 10 January 2018.] http://mantl.io/.

38. rancher.com. *rancher.com.* [Online] [Cited: 10 January 2018.] http://rancher.com/.

39. consul.io. *consul.io.* [Online] [Cited: 10 January 2018.] https://www.consul.io/.

40. influxdata.com. *influxdata.com.* [Online] [Cited: 10 January 2018.] https://www.influxdata.com/.

41. mongodb.com. *mongodb.com.* [Online] [Cited: 10 January 2018.] https://www.mongodb.com/.

42. Registrator. *Registrator.* [Online] [Cited: 10 january 2018.] http://gliderlabs.github.io/registrator/latest/.

43. logspout. *github.com.* [Online] [Cited: 10 January 2018.] https://github.com/gliderlabs/logspout.

44. tyk.io. *tyk.io.* [Online] [Cited: 10 January 2018.] https://tyk.io/.

45. traefik.io. *traefik.io.* [Online] [Cited: 10 January 2018.] https://traefik.io/.

46. prometheus.io. *prometheus.io.* [Online] [Cited: 10 January 2018.] https://prometheus.io/.

47. GitLab Runner. *gitlab.com.* [Online] [Cited: 10 January 2018.] https://docs.gitlab.com/runner/.

48. kibana. *elastic.co.* [Online] [Cited: 10 January 2018.] https://www.elastic.co/products/kibana.

49. Logstash. *elastic.co.* [Online] [Cited: 10 January 2018.] https://www.elastic.co/products/logstash.

50. letsencrypt.org. *letsencrypt.org.* [Online] [Cited: 10 January 2018.] https://letsencrypt.org/.

51. mattermost.com. *mattermost.com.* [Online] [Cited: 10 January 2018.] https://about.mattermost.com/.

52. sensuapp.org. *Sensu.* [Online] [Cited: 10 January 2018.] https://sensuapp.org/.

53. uchiwa.io. *uchiwa.io.* [Online] [Cited: 10 January 2018.] https://uchiwa.io/.

54. typescriptlang.org. *typescriptlang.org.* [Online] [Cited: 10 January 2018.] http://www.typescriptlang.org/.

55. OAuth 2.0. *oauth.net.* [Online] [Cited: 10 January 2018.] https://oauth.net/2/.

56. 2-IMMERSE Auth Service API documentation. *2Immerse origin.* [Online] 10 January 2018. https://origin.platform.2immerse.eu/docs/auth-service/latest/.

57. Layout Service Documentation. *2Immerse Origin.* [Online] [Cited: 10 January 2018.] https://origin.platform.2immerse.eu/docs/layout-service/ .

58. React. *reactjs.org.* [Online] [Cited: 10 January 2018.] https://reactjs.org/.

59. bbc/pydvbcss. *github.com.* [Online] 10 January 2018. https://github.com/bbc/pydvbcss. .

60. 40 MHz Channels. *Metageek.* [Online] [Cited: 10 January 2018.] https://support.metageek.com/hc/en-us/articles/204490510-40-MHz-Channels.

61. Apple's secret "wispr" request. *blog.erratasec.com.* [Online] 10 January 2018. http://blog.erratasec.com/2010/09/apples-secret-wispr-request.html#.WUbAxvryuAw.

62. 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 .

63. 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/ .

64. Chromium Beta branch. *launchpad.net.* [Online] [Cited: 10 January 2018.] https://launchpad.net/~saiarcot895/+archive/ubuntu/chromium-beta.

65. List of Chromium Command Line Switches. *Peter Beverloo.* [Online] [Cited: 10 January 2018.] https://peter.sh/experiments/chromium-command-line-switches/.

66. 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.

67. 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#MElTQpycTAlU5Eut.99.

trustedreviews.com. [Online] [Cited: 10 January 2018.] http://www.trustedreviews.com/opinion/hdmi-2-0-vs-1-4-2913356#UeixwjfXgh3zvDbQ.99.

68. Intel NUC. *archlinux.org.* [Online] [Cited: 10 January 2018.] https://wiki.archlinux.org/index.php/Intel_NUC.

69. Quadratic knapsack problem. *wikipedia.org.* [Online] [Cited: 10 January 2018.] https://en.wikipedia.org/wiki/Quadratic_knapsack_problem.

70. David Pisinger's optimization codes . *diku.dk.* [Online] [Cited: 10 January 2018.] http://www.diku.dk/~pisinger/codes.html.

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

72. Universal Windows Platform documentation. *microsoft.com.* [Online] [Cited: 11 January 2018.] https://docs.microsoft.com/en-us/windows/uwp/.

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

74. W3C. *Presentation-API.* [Online] 2017. [Cited: 15 November 2018.] https://www.w3.org/TR/presentation-api/.

75. Ramdhany, Rajiv. *PhD Thesis: Dynamic deployment and reconfiguration of MANET routing protocsol.* s.l. : Lancaster University, 2009.

Open-source Platform for Multi-screen Entertainment

2IMMERSE

www.2immerse.eu

2-IMMERSE is an EU co-funded innovation project which has developed and is launching a new open-source platform for Object Based Multiscreen Entertainment.

The open-source platform is based on reusable components that will accelerate the development of new immersive multi-screen experiences, encourage the take-up of the HbbTV 2 specification and contribute towards its evolution.

Programme exemplars based on high-value content forms including; MotoGP, the Emirates FA Cup final and award winning Theatre productions have been created to define and demonstrate the core platform capabilities.

Visit the **2-IMMERSE Showcase in Hall 8 F46** to see:

- **Demonstrations** of how Object-based Broadcasting can transform watching multi-screen Sports and Theatre.
- **Production Tools** that enable the authoring of live object-based multi-screen TV programmes.
- **Platform** for distributing multi-screen presentations in a resilient, scalable and extensible manner.
- **Open-source Software** the project will release later this year.
- **Reference Architecture** to guide the development of live multi-screen TV services.

Main 2-IMMERSE Showcase - Hall 8 Stand F46 (Future Zone)

IBC Conference Presentation - Sunday 16th Sept at 16:45 in the Forum

Hall 10 F51 (IRT) - MotoGP demonstrator on an HbbTV 2 TV display

Hall 7 C21 (ChyronHego) - Broadcast tools for object-based production

BBC British Broadcasting Corporation

Illuminations Illuminations

IRT Institut für Rundfunktechnik

CHYRONHEGO ChyronHego

BT British Telecommunications plc

CWI Centrum voor Wiskunde en Informatica

cisco Cisco

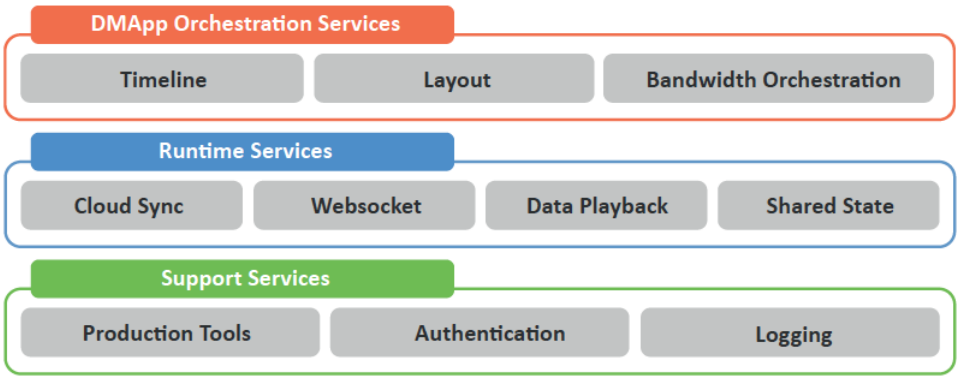
2-IMMERSE has been co-funded by the Horizon 2020 Framework Programme of the European Union under grant agreement n° 687655.

6 Appendix A – IBC 2018 Flyer

The **2-IMMERSE Object-based Broadcasting Platform** enables TV programming to be customised to suit the capabilities of available client devices and network bandwidth. Content can be adapted for a range of different environments allowing, for example, the presentation of football in a public fan-zone space to be quite different to the presentation in a home.

Object based broadcasting is enabled through a **Distributed Media Application** (DMApp), a set of software components orchestrated across the participating devices by a cloud service platform. The DMApp components typically render media, support user interaction and/or implement application logic.

The common client software environment we have adopted to support this is HTML/CSS/JavaScript, and in particular HbbTV 2. The service platform we use is a containerised platform managed using Rancher.



Orchestration, putting the right components in the right part of the right screen at the right time, is the key and distinctive task for the 2-IMMERSE DMApp. Orchestration involves three key services:

A **Timeline Service** that deals with temporal orchestration by interpreting an XML timeline document that controls the temporal composition of the media items and components.

A **Layout Service** that determines the optimum layout of components using authored layout requirements, user preferences, and the set of participating devices and their capabilities.

A **Bandwidth Orchestration Service** that will, where necessary, reallocate bandwidth to optimise the overall quality of experience by monitoring the bandwidth consumption of ABR audio and video components using the generalised architecture and protocols defined in MPEG SAND.

The 2-IMMERSE platform is broadly compatible with the MPEG MORE draft specification which provides a generalised architectural framework for object-based media orchestration. The temporal and synchronisation architecture adopted by MPEG MORE is DVB CSS, which is the basis of the 2-IMMERSE approach to synchronisation.

7 Appendix B – DMApp-RA Functional View Building Process

The following instructions describe how to generate a Functional View from the DMApp-RA Functional Model.

1. Identify functions associated with new concepts in the new platform, generalise them and group them into Functionality Groups
2. Determine if functionality bundled together with other functions belong to the same group or can be partitioned into another group
3. Identify commonalities and variabilities in the functions.
    - Some functions may be similar to functions seen from the previous iteration
    - Determine if each function is general or it is a variation (specific instance) of a general function.
    - For each general function or category, list the variabilities.
    - Variabilities denote the features that change between instantiations of the reference architecture.
4. Capture the way the functionality is accessed or interacts with other functions.
    - This is often as important as identifying the functionality itself.
    - Determine if the interaction can be generalised through a design pattern
5. Build the functional view by listing the function categories and then in each category, listing variabilities
    - In each category, specify
        i) the common responsibility **R** that generalises the different alternatives.
        ii) the different alternatives that realise **R**
        iii) the kind of variability required, e.g.
            - A common interface with different implementations
            - Changing platform behaviour by replacing entire services
            - Changing platform behaviour by the inclusion of new delta functions in services
            - Adding anomalous behaviour to a sequence of common operations
6. In successive iterations, repeat the generalisation process until no more general functions are created.

Figure 9 and Figure 10 show examples of Functional Views derived in other reference architectures or software frameworks.

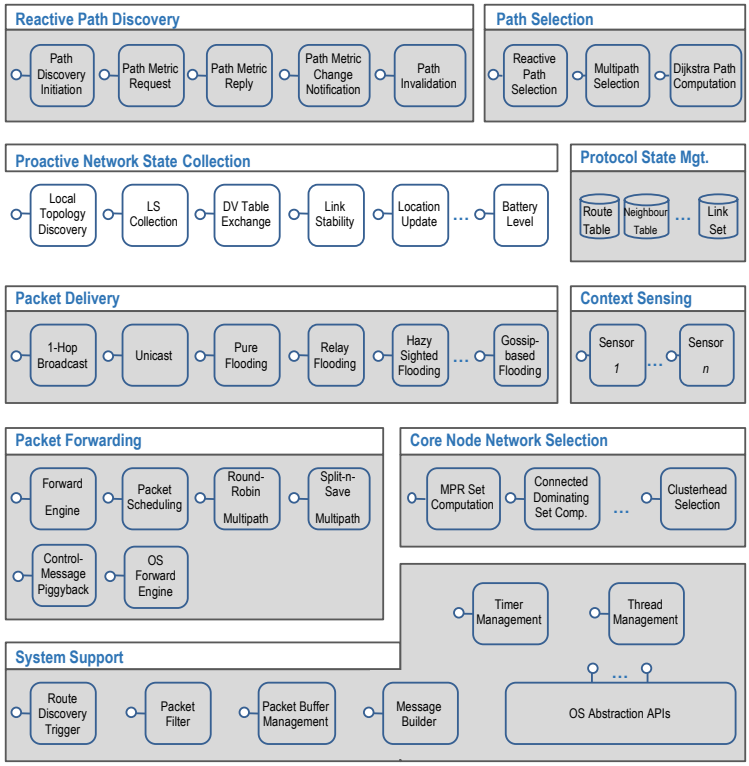


Figure 9: Functional View method applied to MANET routing protocols to identify Functionality Groups and functions (32)

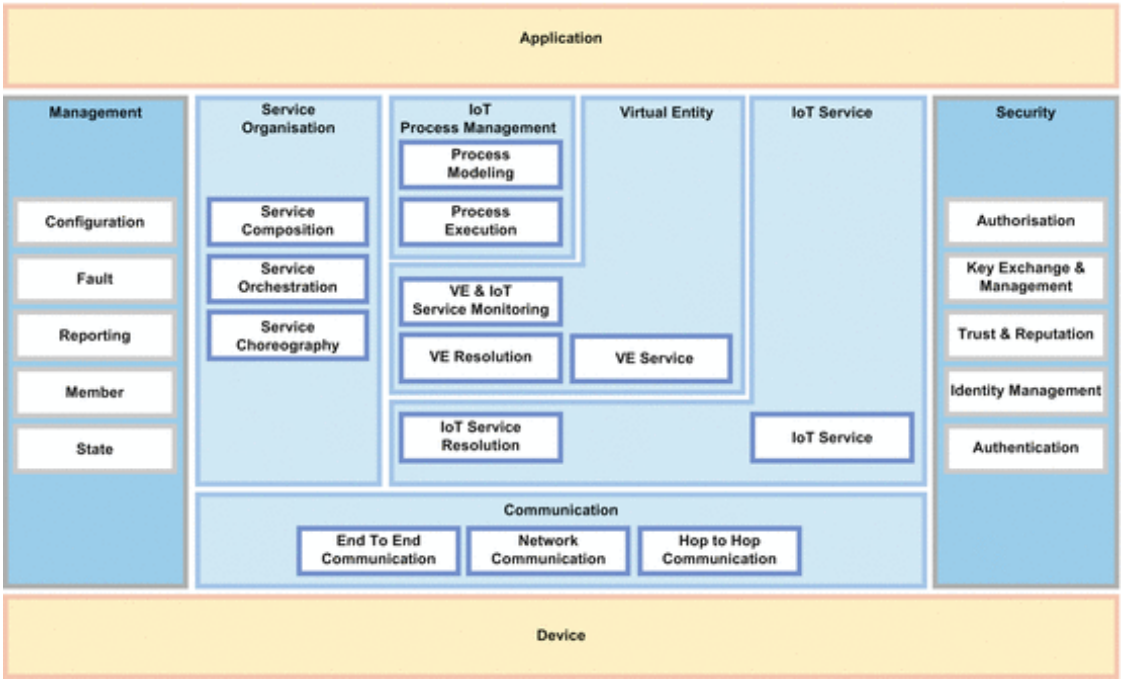


Figure 10: Functional View of the IoT Reference Architecture (33)

8 Appendix C – Intermediate Results of the DMApp-RA Elicitation Methodology

8.1 Functional Model

Our first iteration of the methodology (using the Theatre at Home DMApp) resulted in the FGs shown in **Error! Reference source not found.**

• DMApp	• DMAppComponent
• User	• Communication (Protocol stacks)
• Context	• LayoutOrchestrator
• Device (Personal/Shared/Communal)	• MediaDataSource
• State Management	• ContentStore
• Client-API	• HbbTV2.0 functions
• Timeline	
• Timeline Orchestration	
• Layout Orchestration	
• Media Synchronisation	
• WallClock	

Table 8: Functional Groups after first iteration (Theatre-at-Home DMApp)

8.2 First Iteration of the Functional View based on the initial Theatre-At-Home DMApp

The first iteration of functional decomposition was based on the Functional Model derived from the Theatre-At-Home DMApp. It was as expected, the most effort-intensive step part of the process as it involves identifying the core functions.

Since incremental addition of new functionality was made to the 2Immmerse platform to suit the use case requirements and application domain, our successive refinements of the functional view also reflects the addition of new functions.

The Functional View obtained after the functional decomposition of the Theatre-at-Home DMApp platform is illustrated in **Error! Reference source not found.Error! Reference source not found.**.

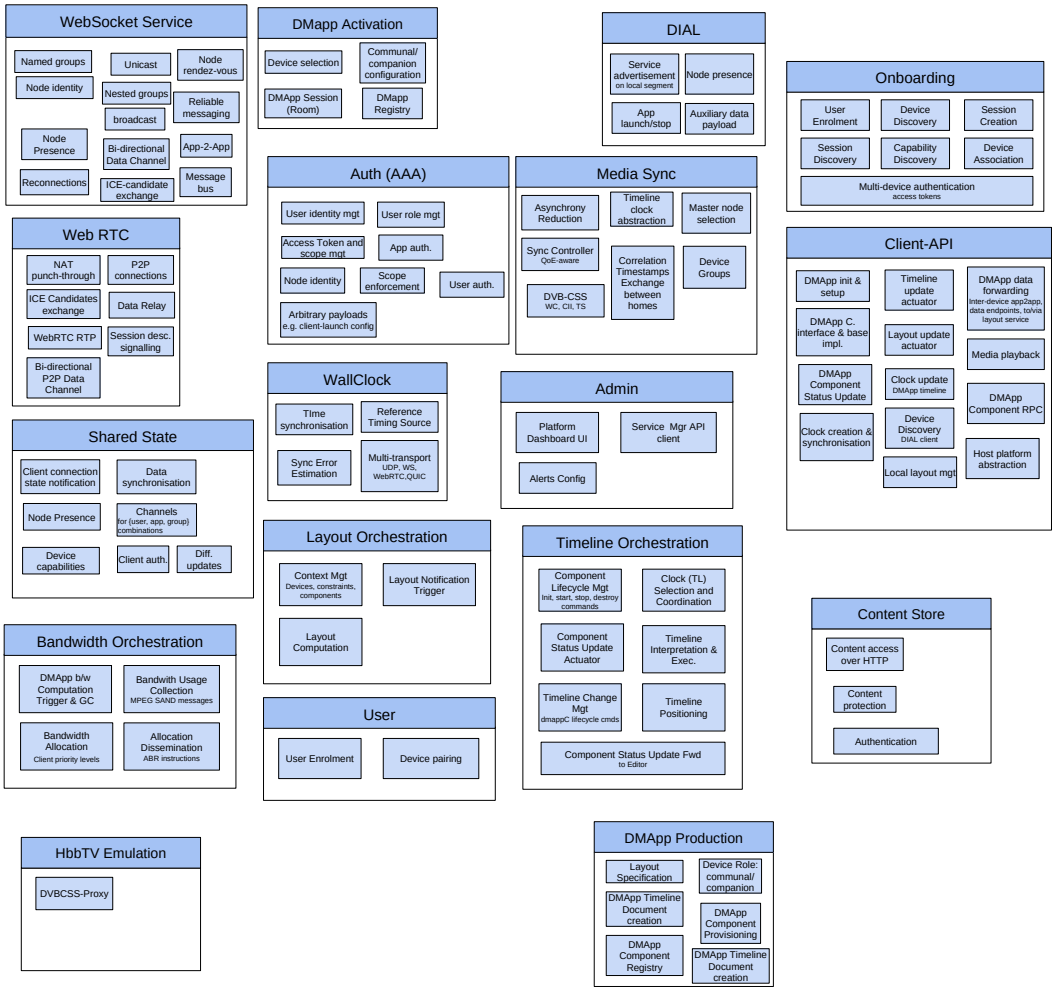


Figure 11: Functional View after first iteration

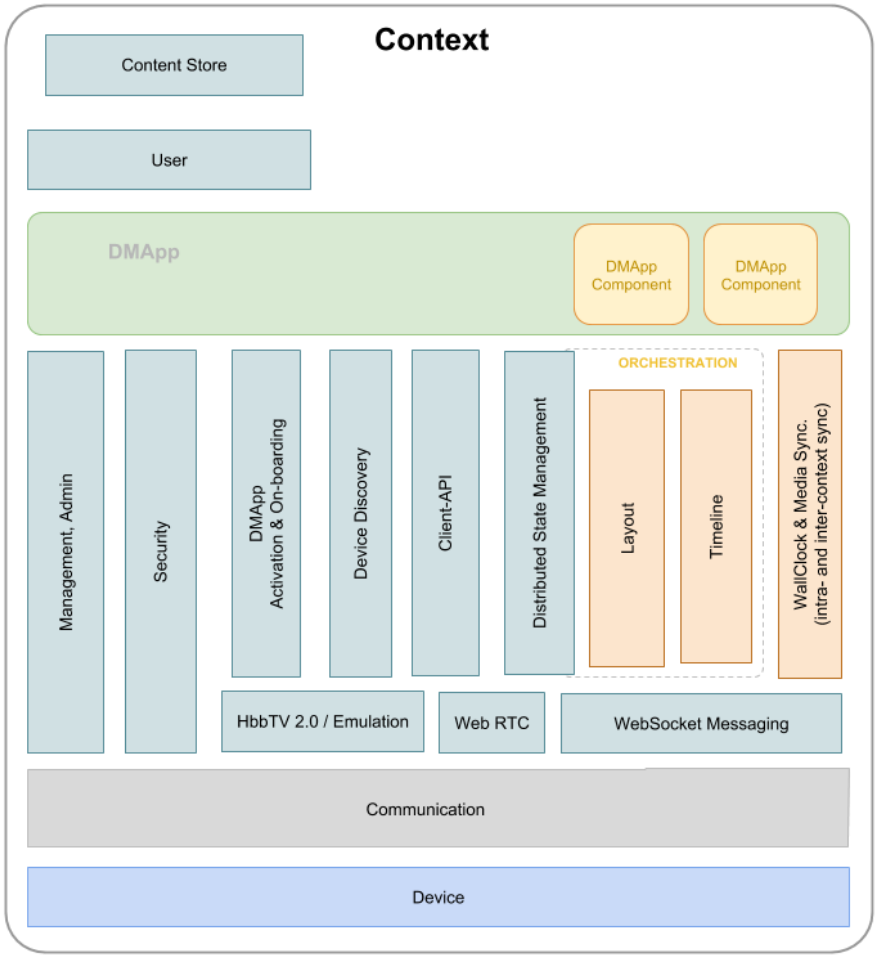


Figure 12: Functional Model after first iteration (Theatre-at-Home DMApp)

These were functionally decomposed in the next step of the process to form the Functional View in the first iteration.