European Commission
HORIZON 2020
The EU Framework Programme for Research and Innovation

Directorate General for Communications Networks, Content and Technology
Innovation Action

ICT-687655

D2.2 Platform-Component Interface Specifications

Due date of deliverable: 31 May 2016
Actual submission date: 22 July 2016
Resubmitted with minor changes: 28 April 2017

Start date of project: 1 December 2015 | Duration: 36 months
Lead contractor for this deliverable: **Cisco**
Version: **28 April 2017**
Confidentiality status: **Public**

Abstract

This document specifies the interfaces of platform components defined in the 2-IMMERSE project system architecture. As such, it is a refinement of the System Architecture defined in project deliverable D2.1, providing a more detailed definition of the platform components and services. This architecture, platform, and its components are designed to enable the four multi-screen service prototypes that will be delivered through the project.

Target audience

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

Disclaimer

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

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

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

Impressum

Full project title: 2-IMMERSE

Title of the workpackage: WP2 Distributed Media Application Platform

Document title: D2.2 Platform-Component Interface Specifications

Editor: James Walker, Cisco

Workpackage Leader: James Walker, Cisco

Project Co-ordinator: Helene Waters, BBC

Project Leader: Phil Stenton, BBC

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

Copyright notice

© 2017 Participants in project 2-IMMERSE

Executive Summary

This document specifies the interfaces of platform components defined in the 2-IMMERSE project system architecture. As such, it is a refinement of the System Architecture defined in project deliverable D2.1, providing a more detailed definition of the platform components and services. This architecture, platform, and its components are designed to enable the four multi-screen service prototypes that will be delivered through the project.

Our approach to specifying the platform components is, for each service, to provide a high-level functional description, including the component responsibilities, a set of verbs (actions), listeners and collaborators (i.e. other components that will interact with the component). We then provide details of the component interface.

For components offering a REST API, we have adopted the RESTful API Modeling Language (RAML) for API specification (http://raml.org/). This allows us to document the interfaces in a consistent approach, and to use tooling to support interface specification, validation and implementation.

These interface specifications have been developed collaboratively, through numerous conference calls, face-to-face meetings and tools such as the project WiKi. We have where possible developed stub versions of components implementing the interfaces as specified to allow basic validation of the specifications.

In the introduction of the document we provide a series of technical use cases, illustrating how the platform services and client applications interact to realise system functionality. This document does not specify the internal architecture and interfaces of the client device software stacks; this has been described at a high level in project deliverable D2.1 and is ‘work in progress’ at the time of writing.

Our focus on functionality for the platform and its components is driven by the initial service prototype (Watching Theatre at Home), and as such, the interface definitions are likely to evolve both as we implement the platform components, and address the expanding scope of the four multi-screen service prototypes through the project. We are currently identifying common functionality required by more than one service, and will consider partitioning such functions into their own micro-services that could be shared by these services. This specification document therefore offers a ‘snapshot’ of the interface specifications at a moment in time. The high-level component definitions have been made available on the project website, and links to these have been included in this document to allow readers to access up-to-date versions of this information.

© 2-IMMERSE Consortium 2017

List of Authors

Mark Lomas - BBC
Rajiv Ramdhany - BBC
Ian Kegel - BT
Jonathan Rennison - BT
James Walker - Cisco (also editor)
Jack Jansen - CWI
Michael Probst - IRT
Christoph Ziegler - IRT

Reviewers

Pablo Cesar - CWI

 © 2-IMMERSE Consortium 2017

Table of contents

Executive Summary 3
List of Authors 4
Table of contents 5
1 Introduction 8
1.1 Platform Overview 9
1.2 Platform Components 10
1.3 Key Technical Use Cases 13
1.3.1 Launch Experience 13
1.3.2 Add Companion Device 16
1.3.3 User Initiates Layout Change 18
1.3.4 Timeline Event 19
1.3.5 Tear Down Experience 20
2 Platform Infrastructure - Mantl 23
2.1 Features and capabilities 23
2.1.1 Core Components 23
2.1.2 Add-ons 23
2.1.3 Goals 24
2.1.4 Architecture 24
3 Platform Services 25
3.1 Service Registry 25
3.1.1 Functional Description 25
3.1.2 API Specification 28
3.2 Device Discovery 29
3.2.1 Functional Description 29
3.2.2 API Specification 29
3.3 Timeline 30
3.3.1 Functional Description 30
3.3.2 API Specification 32
3.3.3 Timeline Document Format 32
3.4 Layout 32
3.4.1 Functional Description 32
3.4.2 API Specification 36
3.4.3 Layout Requirements Document Format 36
3.5 Server-Based Composition 36
3.6 Timeline Synchronisation 37
3.6.1 Synchronisation Service 37
3.6.2 WallClock Synchronisation Service (WCSync) 41
3.6.3 Timeline Synchronisation Service (TimelineSync) 42
3.6.4 SyncKit's Synchroniser API 43
3.6.5 HbbTV's Media Synchroniser API 48
3.6.6 DMAppC and DMAppC Control API 50
3.7 Content Protection and Licensing 52
3.7.1 Functional Description 52
3.7.2 API Specification 52
3.8 Identity Management and Authentication 53
3.8.1 Functional Description 53

© 2-IMMERSE Consortium 2017

3.8.2 API Specification 56
3.9 Session Service 57
3.9.1 Functional Description 57
3.10 Lobby Service 58
3.10.1 Overview 58
3.10.2 Functional Description 59
3.10.3 API Specification 63
3.11 Call Service (SIP) 63
3.11.1 Functional Description 63
3.11.2 API 65
3.12 Logging 67
3.12.1 Functional Description 67
3.13 Analytics 70
3.14 Origin Server / CDN 70
3.14.1 Functional Description 70
3.14.2 API Specification 71
3.15 TV Platform 71
4 Conclusion 72
Annex A **DIAL Plug-In API Specifications 73**
A.1 getDialClient() 73
A.2 DialClient 73
A.2.1 Methods 73
A.3 Device 73
A.3.1 Properties 73
A.3.2 Methods 74
A.4 DeviceStatus 74
A.4.1 Properties 74
A.5 Usage 74
Annex B **2-IMMERSE Timeline Service API documentation version v1 ..... 76**
B.1 /context 76
B.1.1 /context 76
B.1.2 /context/{contextId}/dump 76
B.1.3 /context/{contextId}/loadDMAppTimeline 76
B.1.4 /context/{contextId}/unloadDMAppTimeline 77
B.1.5 /context/{contextId}/dmappcStatus 77
B.1.6 /context/{contextId}/timelineEvent 77
B.1.7 /context/{contextId}/clockChanged 77
Annex C **Timeline Document Format Design Considerations 79**
C.1.1 Requirements 79
C.1.2 Design 79
C.1.3 Format 80
C.1.4 Examples 81
C.1.5 API 81
Annex D **2-IMMERSE Layout Service API documentation version v1 ..... 83**
D.1 /context 83
D.1.1 /context 83
D.1.2 /context/{contextId} 84
D.1.3 /context/{contextId}/devices 84
D.1.4 /context/{contextId}/devices/{deviceId} 85

 © 2-IMMERSE Consortium 2017

D.1.5 /context/{contextId}/devices/{deviceId}/orientation 86
D.1.6 /context/{contextId}/dmapp 86
D.1.7 /context/{contextId}/dmapp/{dmappId} 87
D.1.8 /context/{contextId}/dmapp/{dmappId}/actions/clockChanged 88
D.1.9 /context/{contextId}/dmapp/{dmappId}/component/{componentId} 88
D.1.10 .../dmapp/{dmappId}/component/{componentId}/actions/init 89
D.1.11 .../dmapp/{dmappId}/component/{componentId}/actions/start 89
D.1.12 .../dmapp/{dmappId}/component/{componentId}/actions/stop 89
D.1.13 .../dmapp/{dmappId}/component/{componentId}/actions/hide 90
D.1.14 .../dmapp/{dmappId}/component/{componentId}/actions/show 90
D.1.15 .../dmapp/{dmappId}/component/{componentId}/actions/move 91
D.1.16 .../dmapp/{dmappId}/component/{componentId}/actions/clone 91
D.1.17 .../dmapp/{dmappId}/component/{componentId}/actions/status 92
D.1.18 .../dmapp/{dmappId}/component/{componentId}/actions/saveState 92
D.1.19 .../dmapp/{dmappId}/component/{componentId}/actions/restoreState 92
Annex E **2-IMMERSE Lobby Service REST API documentation 94**
E.1 Lobbies 94
E.1.1 /lobbies 94
E.1.2 /lobbies/{lobbyId} 94
E.2 JSON Schema for Lobby Service Web Socket Communications 95
Annex F **2-IMMERSE Timeline Synchronisation 99**
F.1 Intra- Home Synchronisation 99
F.1.1 Cloud Services 101
F.1.2 TV Components/APIS 102
F.1.3 Companion Device Components/APIs 102
F.2 Inter-Home Synchronisation 104
F.2.1 Cloud Services/APIs 107
F.2.2 TV Components/APIs 107
F.2.3 Companion Device Components/APIs 108

© 2-IMMERSE Consortium 2017

1 Introduction

This document specifies the interfaces of platform components defined in the 2-IMMERSE project system architecture. As such, it is a refinement of the System Architecture defined in project deliverable D2.1, providing a more detailed definition of the platform components and services.

Our approach to specifying the platform components is, for each service, to provide a high-level functional description, including the component responsibilities, a set of verbs (actions), listeners and collaborators (i.e. other components that will interact with the component). We then provide details of the component interface.

For components offering a REST API, we have adopted the RESTful API Modelling Language (RAML) for API specification (http://raml.org/). This allows us to document the interfaces in a consistent approach, and to use tooling to support interface specification, validation and implementation.

These interface specifications have been developed collaboratively, through numerous conference calls, face-to-face meetings and tools such as the project WiKi. We have where possible developed stub versions of components implementing the interfaces as specified to allow basic validation of the specifications.

This document does not specify the internal architecture and interfaces of the client device software stacks; this has been described at a high level in project deliverable D2.1 and is ‘work in progress’ at the time of writing.

Our focus on functionality for the platform and its components is driven by the initial service prototype (Watching Theatre at Home), and as such, the interface definitions are likely to evolve both as we implement the platform components, and address the expanding scope of the four multi-screen service prototypes through the project. We are currently identifying common functionality required by more than one service, and will consider partitioning such functions into their own micro-services that could be shared by these services. This specification document therefore offers a ‘snapshot’ of the interface specifications at a moment in time. The high-level component definitions have been made available on the project website, and links to these have been included in this document to allow readers to access up-to-date versions of this information.

In the remainder of this introductory section we give a brief recap of the 2-IMMERSE System Architecture and Platform services. We also give a summary of the required scope for each of these services for the initial “Watching Theatre at Home” service prototype, and an indication of the status and maturity of each of these components and interfaces. Finally, we provide a set of illustrative technical use-cases.

The chapters that follow cover the platform infrastructure that we intend to adopt as the basis of our cloud services, the interface specifications for each of the service components, and finally a conclusion reflecting on current status and next steps.

 © 2-IMMERSE Consortium 2017

1.1 Platform Overview

Within project deliverable D2.1 we describe the 2-IMMERSE system architecture. We have taken a ‘layered’ approach to documenting our system architecture in order to maximise clarity and maintain an appropriate ‘separation of concerns’:

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

This is shown below in Figure 1.

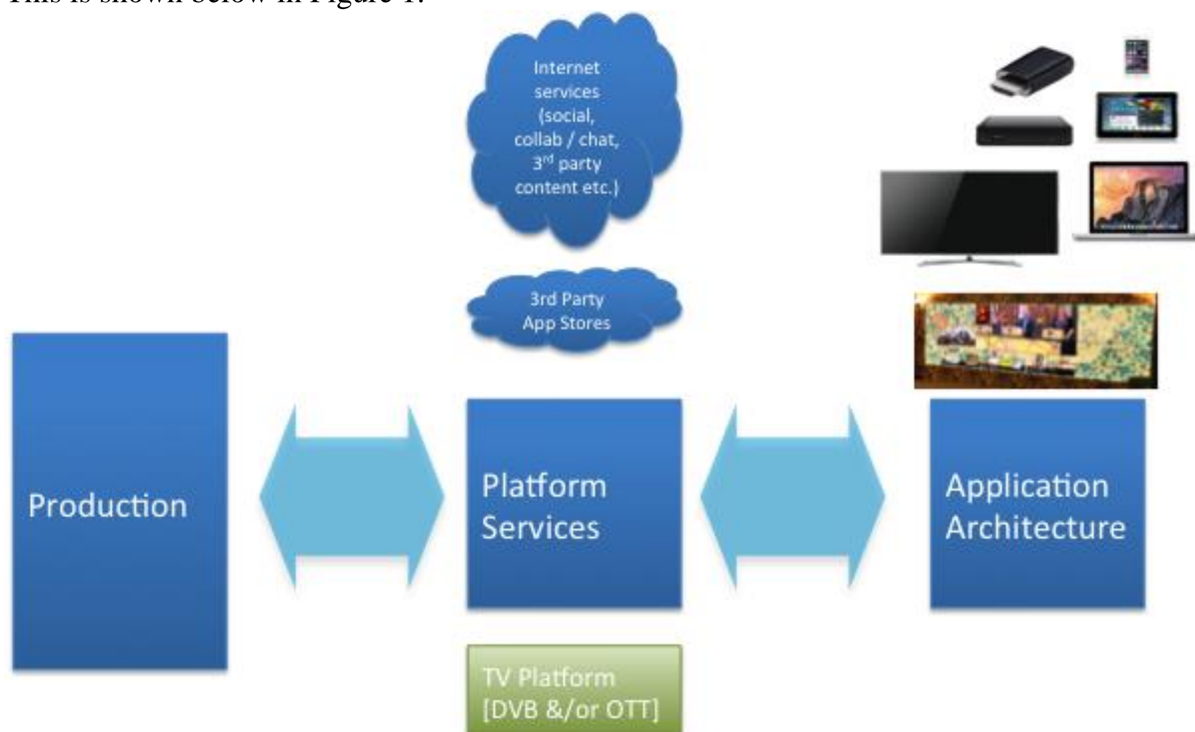

Figure 1 - High-Level Platform Architecture

Within D2.1 we define a set of core services that comprise the 2-IMMERSE platform. We note that for the UX Engine and Sync services, we can see two envisaged deployment options; one where these services are deployed in-home (for example running on the TV device), and alternatively with these services running in the cloud. For the first service prototype, we are focusing on the cloud deployment model. This is shown below in Figure 2. In this deliverable, we are documenting the interfaces of these platform service components.

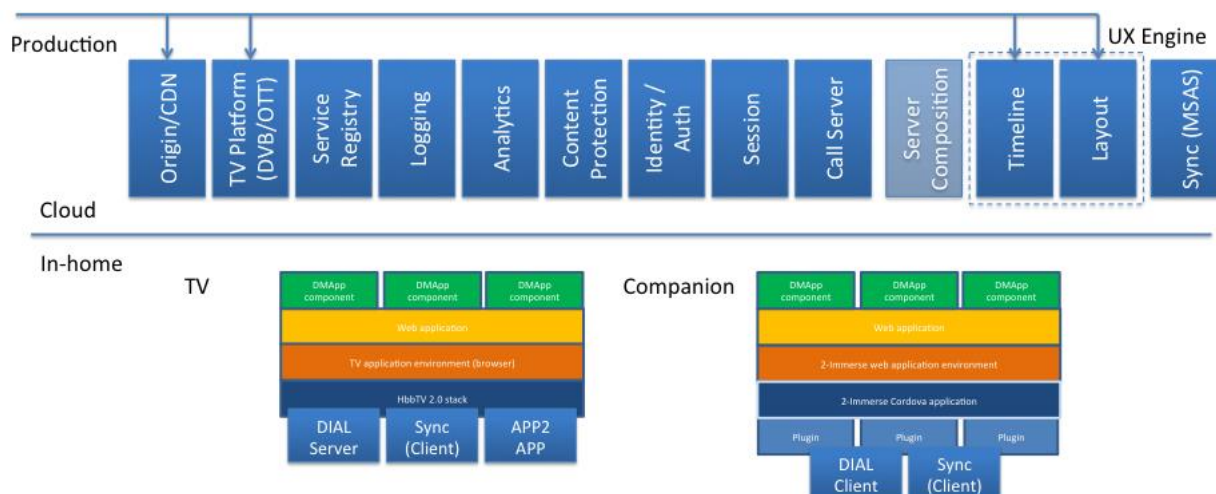

Figure 2 - Service / Client Deployment - cloud UX Engine Services

1.2 Platform Components

Table 1 below gives a summary of the service scope required for the initial service prototype, “Watching Theatre at Home”. It also summarises the current interface / implementation status and maturity; i.e. whether we are adopting an existing specification and / or implementation, or specifying and / or implementing something new, which by definition would be less mature.

Service	Scope for first service prototype	Status / Maturity
Service Registry	Required; needs to allow services to discover other services, and clients service discovery.	Adopting existing open source implementations (provided by Mantl platform)
Device Discovery	Required	Adopting existing partner implementations
Timeline	Required; needs to support a ‘pre-authored’ on-demand experience delivered ‘as-live’	New component. A basic implementation exists which interacts with the layout service, and will run a ‘hardcoded’ timeline
Layout	Required; needs to support UX being “wire-framed” in WP3	New component. A basic implementation exists which manages contexts and DMApps, and performs basic layout. Support for layout requirements and push updates to clients is in progress at the time of writing.

 © 2-IMMERSE Consortium 2017

Service	Scope for first service prototype	Status / Maturity
Server Based Composition	Out-of-scope	-
Timeline Synchronization	Required; needs to support intra-home sync, and inter-home sync well enough to enable inter-home communication.	Adopting existing partner implementations; modifications and possibly new components will be required.
Content Protection and Licensing	Minimal implementation (may not require running service)	-
Identity Management and Authentication	Required	Will review existing open source implementations for possible adoption / adaptation
Session	Required for authenticated service access	Will adopt existing open source implementation
Lobby	Required to support inter-home communication.	Adopting existing implementations, some modifications required
Call Server (SIP)	Required to support inter-home communication.	Adopting existing open source implementations, some modifications required
Logging	Required	Mature; adopting existing open source implementations (provided by the Mantl and Google Analytics platforms)
Analytics	Some offline post-trial use cases	-
Origin Server/CDN	Required	Adopting existing open source components for origin server.
TV Platform	Addressed by Origin Server/CDN	-

Table 1- Service Scope and Status / Maturity

In
Figure 3 below, we show the architecture of the platform for the first service prototype i.e. with the service components that are in scope for the first service prototype, indicating the interfaces between services. Note that:

- The session service manages authenticated access to the other platform services
- Having authenticated with the session service, we expect clients to access the service registry to discover the location of other platform services
- All deployed services would register with the service registry in order to be discoverable by clients and other services.
- All services will log using the logging service (which in turn will drive the platform analytics)
- Clients will not access the timeline service directly; the layout service manages interactions with the timeline service.

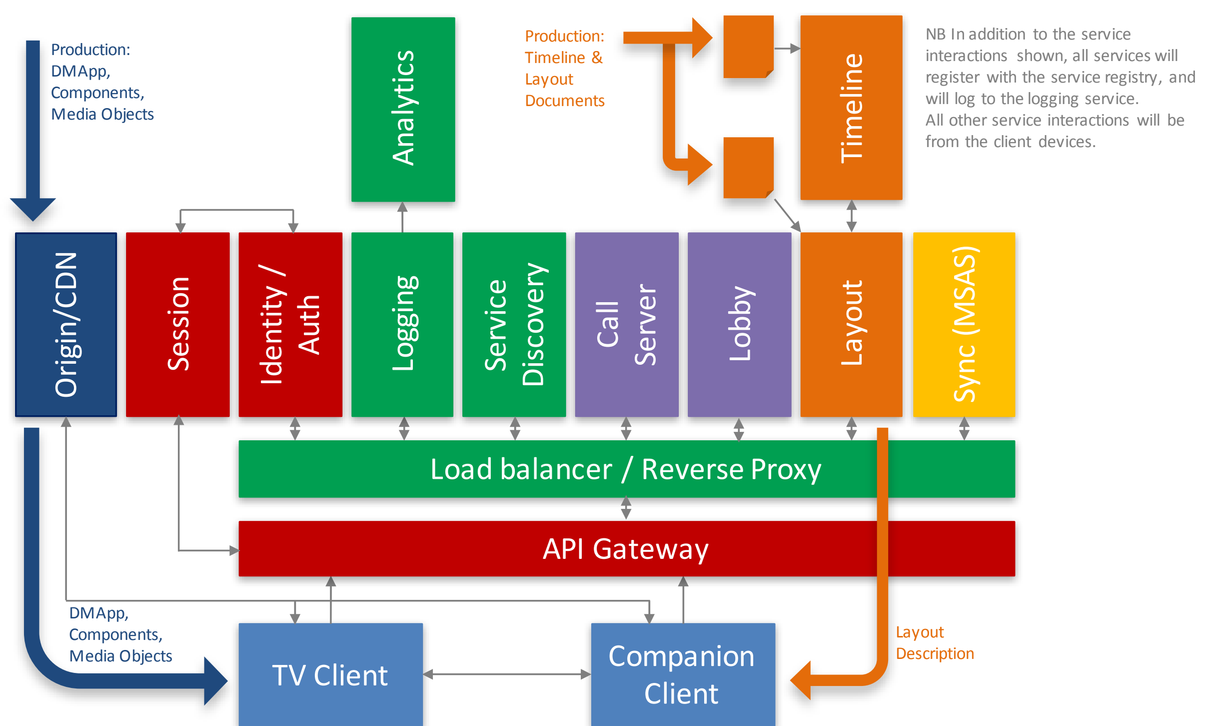

Figure 3 - Platform Service Architecture for the First Service Prototype

Note that we expect all of the services that are in scope for the first service prototype to be used in subsequent service prototypes, with the possible exception of:

- Timeline Synchronization - inter-home sync
- Lobby Service
- Call Server (SIP)

Which may not be required if these service prototypes do not require inter-home communication

© 2-IMMERSE Consortium 2017

In the sections that follow we will outline some key technical use cases, showing how client applications and platform service interact to realise system functionality.

1.3 Key Technical Use Cases

In this section of the document we provide a set of technical use cases, illustrating how the platform services and client applications interact to realise system functionality:

- Launch Experience
- Add Companion device
- User Initiates Layout Change
- Timeline Event
- Tear Down Experience

Latest documentation: https://2immerse.eu/wiki/technical-use-cases/

1.3.1 Launch Experience

This use case illustrates the process of launching an experience, with the TV client device creating a context, a companion client device joining that context. The TV client then launches a DMApp, generating an initial timeline position & layout, and once the DMApp components have been initialised on the clients, the timeline starts running.
Two variants are shown, Figure 4 below shows the process for a TV and companion device, whilst Figure 5 shows the process for a TV only.

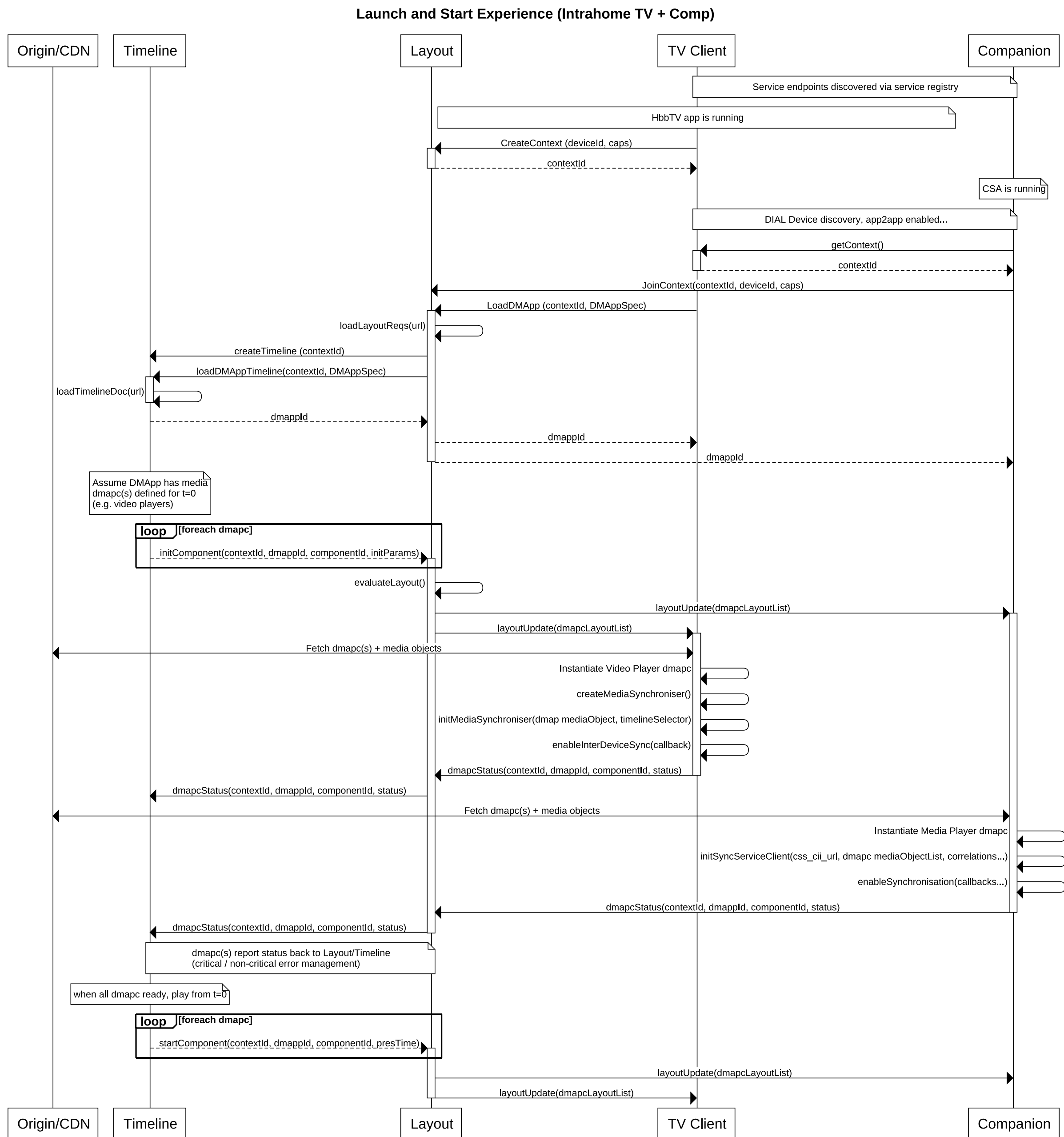

Figure 4 - Launch and Start Experience (Intrahome TV + Companion)

A detailed description of the Launch and Start Experience Intrahome TV and Companion use case as shown in Figure 4 follows; the Intrahome TV only version shown in Figure 5 is a subset of this.

Prerequisites:

- TV is running HbbTV app, and has discovered service endpoints via service registry
- Services have discovered other services through service registry

 © 2-IMMERSE Consortium 2017

Sequence:

- TV requests Layout Service to create a context
- User launches companion app, which discovers TV through DIAL and gets the contextId
- TV requests Layout Service to join this context
- TV requests Layout Service to load a DMApp into this context
- Layout Service loads the specified layout requirements document
- Layout Service requests the Timeline Service to create a timeline for this context
- Layout Service requests the Timeline Service to load a timeline document for this context
- Timeline Service returns a dmappid to the Layout Service
- Layout Service returns a dmappid to the clients
- For each DMApp component at the start of the timeline:
    - Timeline Service requests the Layout Service to initialize the component
    - Layout Service re-evaluates layout, and sends updated layout descriptions to affected client(s)
    - Client devices will fetch and instantiate the component implementation and required media objects from the Origin/CDN, making the appropriate media synchroniser configuration depending whether the component is a sync master or slave.
    - Client devices will return component status(es) to the Layout service, which forwards these to the Timeline Service
- When all of the DMApp components are ready, the timeline is started, and for each DMApp component:
    - Timeline Service requests the Layout Service to start the component
    - Layout Service sends updated layout descriptions to affected client(s)

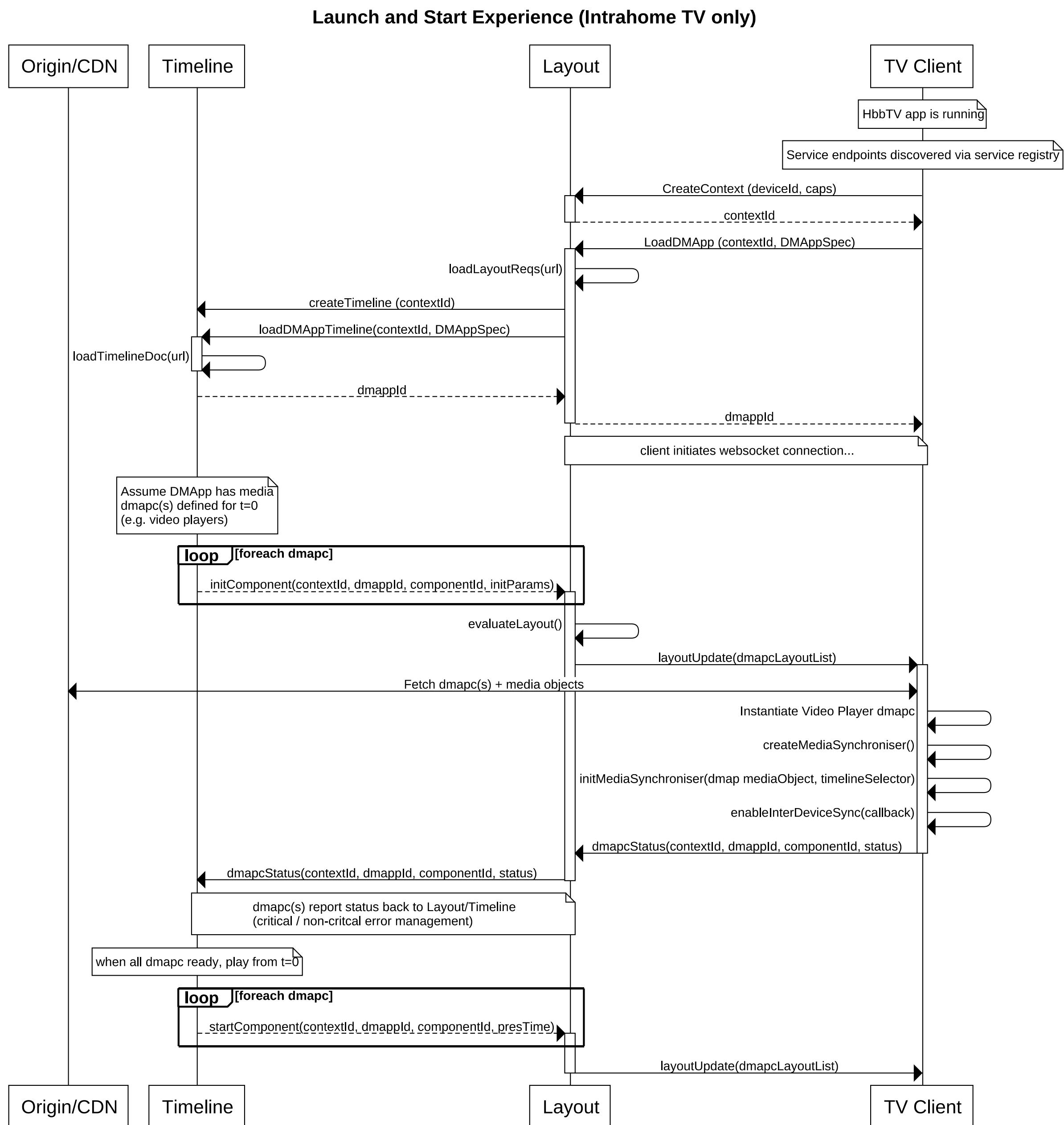

Figure 5 - Launch and Start Experience (Intrahome TV Only)

1.3.2 Add Companion Device

This use case (shown below in Figure 6) illustrates the process of adding a companion device to a context with a single TV device, where a DMApp is already running. Adding the device triggers an updated layout where DMApp components are migrated from the TV to the companion device.

 © 2-IMMERSE Consortium 2017

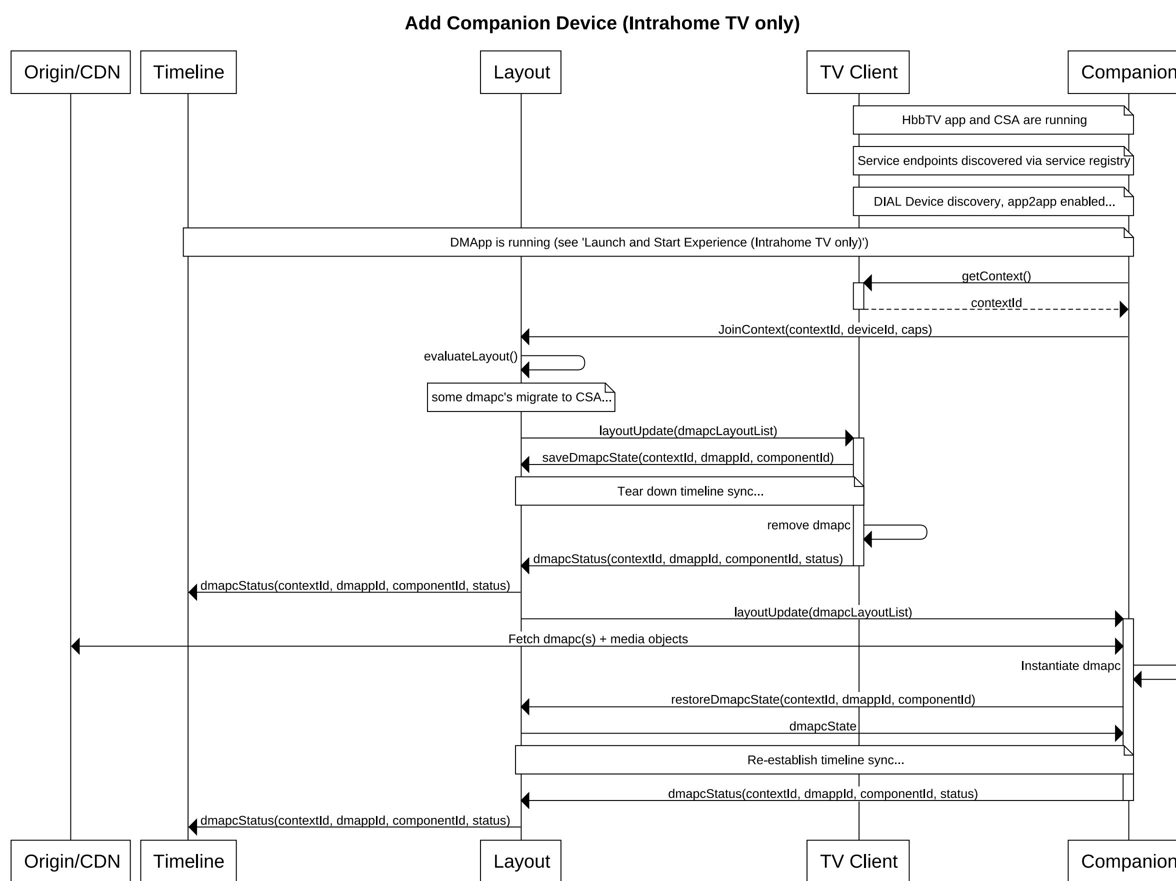

Figure 6 - Add Companion Device (Intrahome TV Only)

A detailed description of the Add Companion Device (Intrahome TV Only) use case as shown in Figure 6 follows.

Prerequisites:

- TV is running HbbTV app, and has discovered service endpoints via service registry
- Services have discovered other services through service registry
- The TV has launched a DMApp, the timeline is running and the TV is displaying components.

Sequence:

- User launches companion app, which discovers TV through DIAL and gets the contextId
- TV requests Layout Service to join this context
- Layout Service re-evaluates layout

NB – we assume in this example this results in a component migrating from the TV to the Companion

- The Layout service sends an updated layout description to the client device from which the component will move
- The Client persists the component state to the Layout Service, tears down the component media synchroniser, return component status(es) to the Layout service, which forwards these to the Timeline Service.
- The Layout service sends an updated layout description to the client device to which the component will move

- The Client device will fetch and instantiate the component implementation and required media objects from the Origin/CDN, retrieve the component state from the layout service, and make the appropriate media synchroniser configuration depending whether the component is a sync master or slave.
- The Client device returns component status to the Layout service, which forwards these to the Timeline Service

1.3.3 User Initiates Layout Change

This use case (shown below in Figure 7) illustrates the process of a user initiating a layout change, for example by using their companion device to 'drag' a DMApp component between devices triggering an updated layout where the DMApp component is migrated from the TV to the companion device.

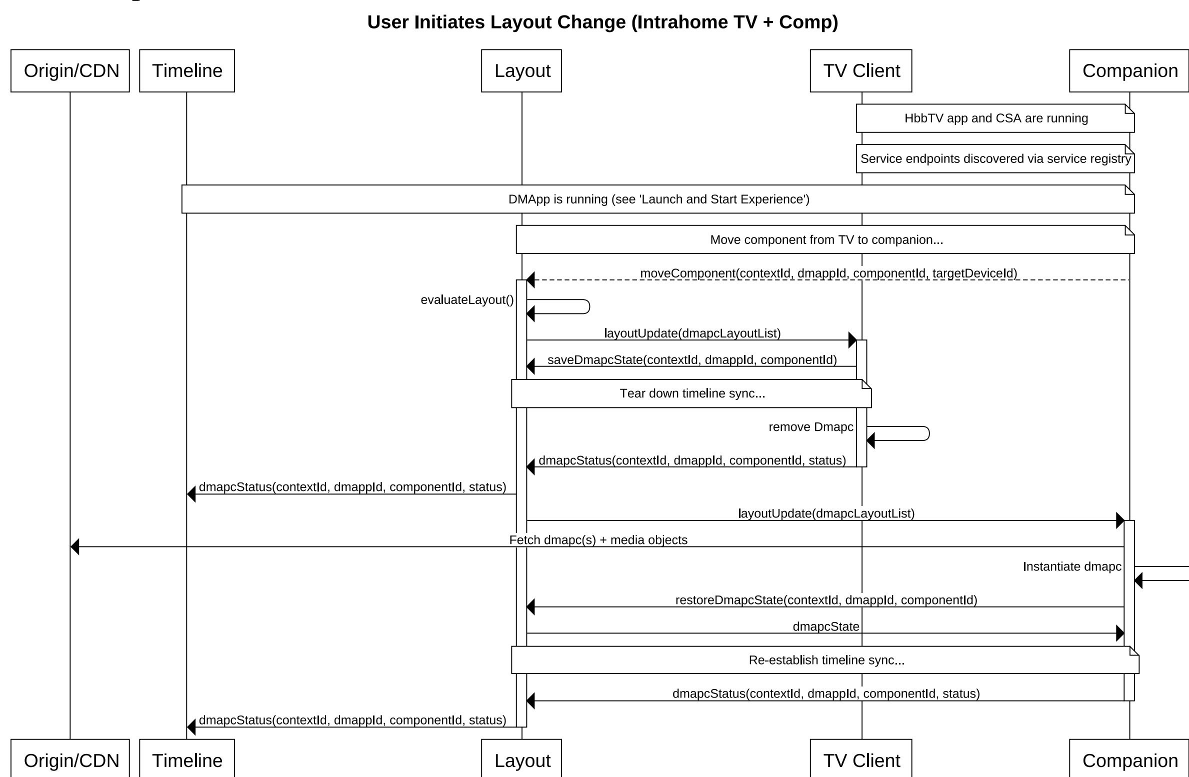

Figure 7 - User Initiates Layout Change (Intrahome TV and Companion)

A detailed description of the User Initiates Layout Change (Intrahome TV and Companion) use case as shown in Figure 7 follows.

Prerequisites:

- TV is running HbbTV app, Companion is running the companion app, both have discovered service endpoints via service registry
- Services have discovered other services through service registry
- TV has created a context that both devices have joined
- TV has launched a DMApp, the timeline is running and the TV is displaying components.

 © 2-IMMERSE Consortium 2017

Sequence:

- The user initiates a component move from their companion app.
- Companion requests Layout Service to move a component between devices
- Layout Service re-evaluates layout
- The Layout service sends an updated layout description to the client device from which the component will move
- The Client persists the component state to the Layout Service, tears down the component media synchroniser, return component status(es) to the Layout service, which forwards these to the Timeline Service.
- The Layout service sends an updated layout description to the client device to which the component will move
- The Client device will fetch and instantiate the component implementation and required media objects from the Origin/CDN, retrieve the component state from the layout service, and make the appropriate media synchroniser configuration depending whether the component is a sync master or slave.
- The Client device returns component status to the Layout service, which forwards these to the Timeline Service

1.3.4 Timeline Event

This use case (shown below in Figure 8 - Timeline Events (Intrahome TV and Companion)) illustrates the process of a timeline event triggering a layout change. In the example, we assume a timeline event where a new DMApp component is instantiated.

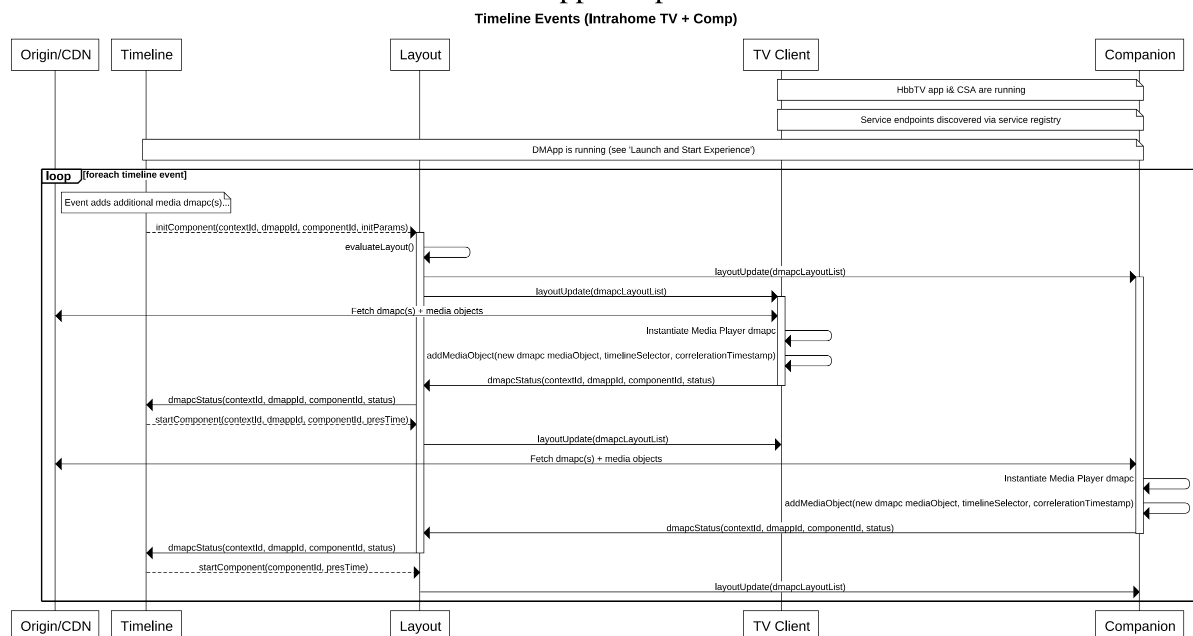

Figure 8 - Timeline Events (Intrahome TV and Companion)

A detailed description of the Timeline Events (Intrahome TV and Companion) use case as shown in Figure 8 follows.

Prerequisites:

- TV is running HbbTV app, Companion is running the companion app, both have discovered service endpoints via service registry
- Services have discovered other services through service registry
- TV has created a context that both devices have joined
- TV has launched a DMApp, the timeline is running and the TV is displaying components.

Sequence:

- The running timeline in the Timeline Service reaches a point where several new DMApp components are started. For each new DMApp component:
  - Timeline Service requests the Layout Service to initialize the component
  - Layout Service re-evaluates layout, and sends updated layout descriptions to affected client(s)
  - Client devices will fetch and instantiate the component implementation and required media objects from the Origin/CDN, making the appropriate media synchroniser configuration depending whether the component is a sync master or slave.
  - Client devices will return component status(es) to the Layout service, which forwards these to the Timeline Service
- When all of the new DMApp components are ready, the timeline is started, and for each DMApp component:
  - Timeline Service requests the Layout Service to start the component
  - Layout Service sends updated layout descriptions to affected client(s)

1.3.5 Tear Down Experience

This use case (shown in Figure 9) illustrates the process of tearing down a running DMApp in a context with a TV and companion device.

 © 2-IMMERSE Consortium 2017

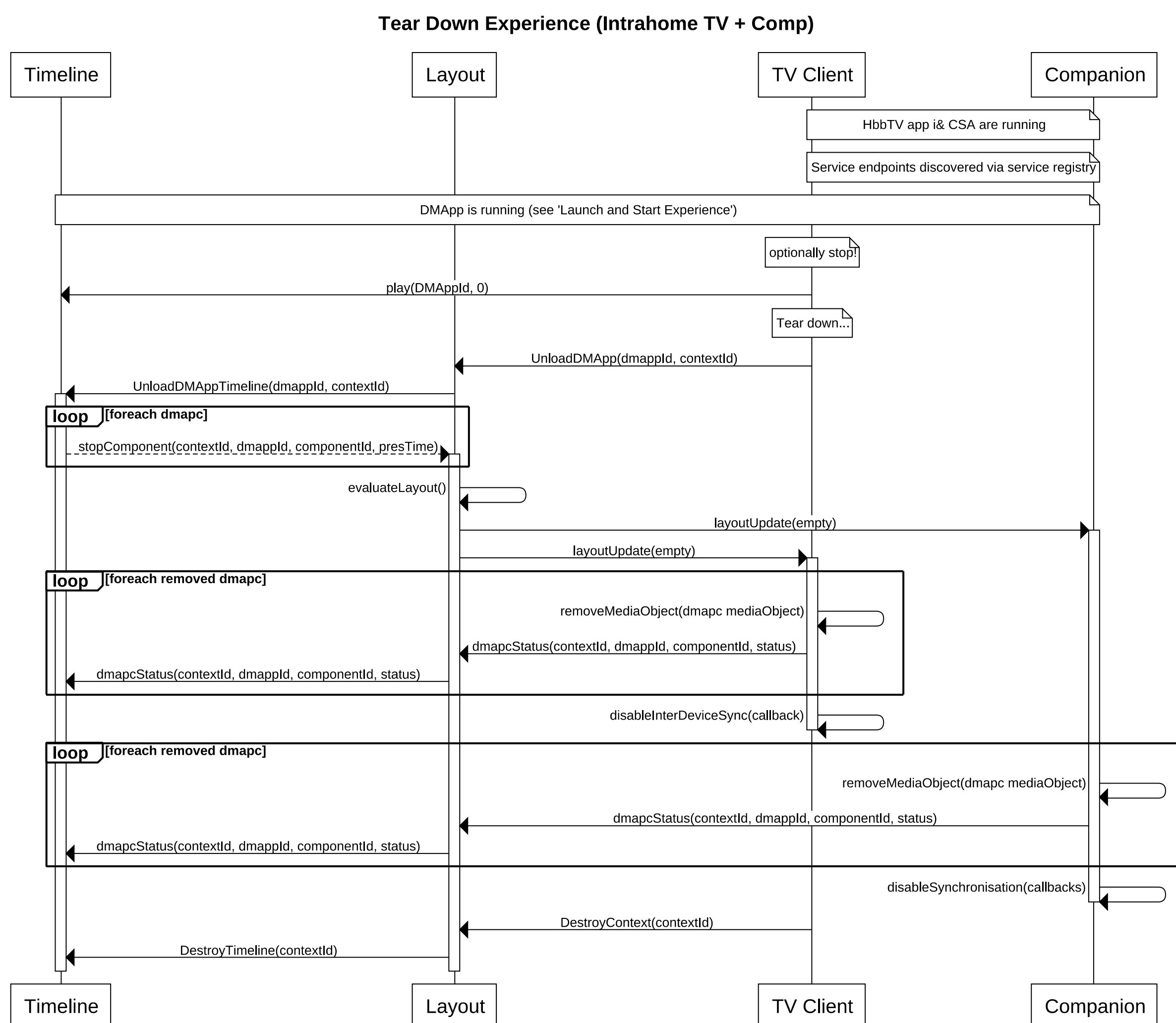

Figure 9 - Tear Down Experience (Intrahome TV and Companion)

A detailed description of the Tear Down Experience (Intrahome TV and Companion) use case as shown in Figure 9 follows.

Prerequisites:

- TV is running HbbTV app, Companion is running the companion app, both have discovered service endpoints via service registry
- Services have discovered other services through service registry
- TV has created a context that both devices have joined
- TV has launched a DMApp, the timeline is running and the TV is displaying components.

Sequence:

- TV requests the Timeline Service to stop the running timeline
- TV requests the Layout Service to unload the DMApp
- Layout Service requests the Timeline Service to unload the DMApp timeline.
- For each active DMApp component:
    - Timeline Service requests the Layout Service to stop the component

- Layout Service re-evaluates layout, and sends updated (i.e. empty) layout descriptions to affected client(s)
- Clients remove the components, tear down the component media synchronisers, and return component status(es) to the Layout service, which forwards these to the Timeline Service.

- TV requests the Layout Service to destroy the context
- Layout Service requests the Timeline Service to destroy the context timeline

 © 2-IMMERSE Consortium 2017

2 Platform Infrastructure - Mantl

Within the project, we have chosen to adopt Mantl as the foundation of our service platform. Mantl is a modern platform for rapidly deploying globally distributed services. It provides an integrated set of industry-standard open-source components. It is cloud infrastructure provider agnostic, and can be deployed on AWS, OpenStack, Vagrant, Bare Metal etc. Mantl is licensed by Cisco under the Apache Version 2 License.

Mantl provides basic infrastructure to allow us to deploy and manage our platform services as 'microservices', which can be scaled as required. The Mantl platform includes features which will directly support some of our platform service requirements; e.g. Consul for Service Discovery (registry), and ELK Stack for logging.

Client access to all services will be via SSH. Authentication will be required, using the session service to obtain a token which is passed with each REST API call.

Latest Mantl documentation can be accessed from: http://docs.mantl.io/en/latest/

A summary of the Mantl platform is given in the sections that follow.

2.1 Features and capabilities

The features and capabilities of the Mantl infrastructure are described below.

2.1.1 Core Components

- Consul for service discovery
- Vault for managing secrets
- Mesos cluster manager for efficient resource isolation and sharing across distributed services
- Marathon for cluster management of long running containerized services
- Kubernetes for managing, organizing, and scheduling containers
- Terraform deployment to multiple cloud providers
- Docker container runtime
- Traefik for proxying external traffic
- mesos-consul populating Consul service discovery with Mesos tasks
- Mantl API easily install supported Mesos frameworks on Mantl
- Mantl UI a beautiful administrative interface to Mantl

2.1.2 Add-ons

- ELK Stack for log collection and analysis
- Logstash for log forwarding
- GlusterFS for container volume storage
- etcd distributed key-value store for Calico
- Calico a new kind of virtual network
- collectd for metrics collection
- Chronos a distributed task scheduler
- Kong for managing APIs

© 2-IMMERSE Consortium 2017

2.1.3 Goals

- Security
- High availability
- Rapid immutable deployment (with Terraform + Packer)

2.1.4 Architecture

The base platform contains control nodes that manage the cluster and any number of agent nodes. Containers automatically register themselves into DNS so that other services can locate them. The high level Mantl architecture is shown below in Figure 10

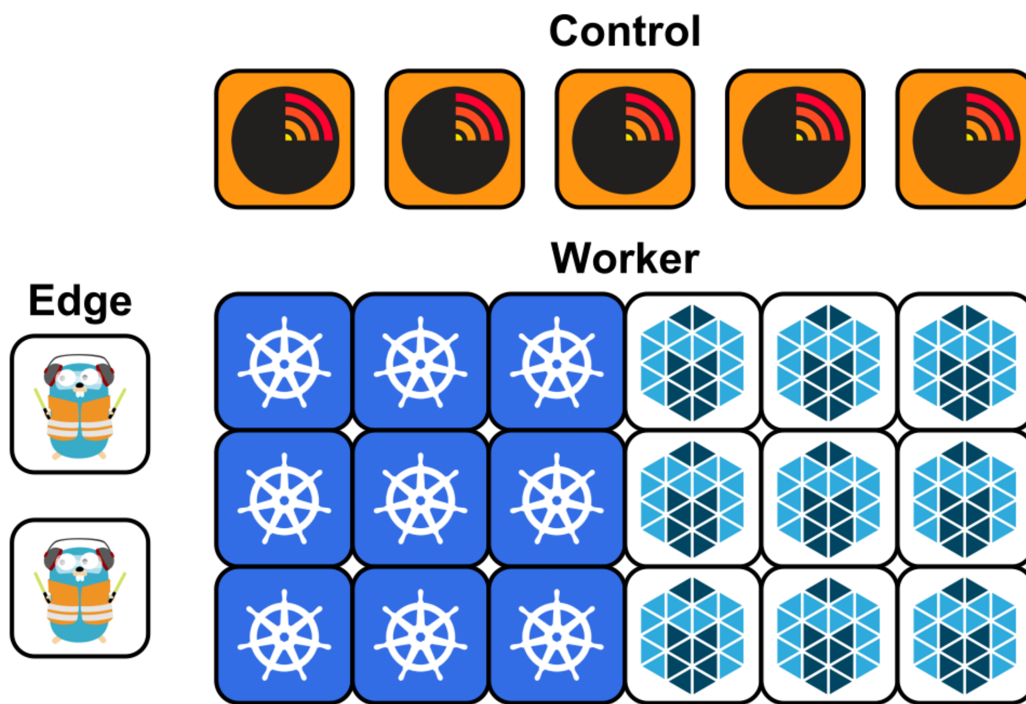

Figure 10 - Mantl High Level Architecture

2.1.4.1 Control Nodes

The control nodes manage a single datacenter. Each control node runs Consul for service discovery, Mesos and Kubernetes leaders for resource scheduling and Mesos frameworks like Marathon.

The Consul Ansible role will automatically bootstrap and join multiple Consul nodes. The Mesos role will provision highly-available Mesos and ZooKeeper environments when more than one node is provisioned.

2.1.4.2 Agent Nodes

Agent nodes launch containers and other Mesos- or Kubernetes-based workloads.

2.1.4.3 Edge Nodes

Edge nodes are responsible for proxying external traffic into services running in the cluster.

 © 2-IMMERSE Consortium 2017

3 Platform Services

This section provides information on the platform services. It provides a functional description and details of the API specification, and where applicable a document format. It is derived directly from the project's online documentation and is therefore necessarily quite hierarchical.

3.1 Service Registry

Latest service documentation: https://2immerse.eu/wiki/service-registry/

3.1.1 Functional Description

This section provides the functional description of the Service Registry.

3.1.1.1 Responsibilities

The 2-IMMERSE system architecture is built from a number of defined services, each scoped with specific roles and responsibilities. These services will be designed to scale elastically, running the required number of instances to meet dynamic load requirements. The array of services types and instances need to collaborate and interoperate with each other to deliver the 2-IMMERSE experience.

The Service Discovery component is responsible for maintaining a dynamic service registry of deployed service instances, together with querying interfaces to share addressable attributes of healthy service instances across the service community hosted in the system. The service registry catalogue contains metadata to describe service types as functional entities, together with a list of operational service instances for a given service type within the system. The Service Registry is an essential component required to provide effective Service Discovery within an elastically scalable system infrastructure.

There are two main approaches to Service Discovery, either client-side or server side discovery, each with their own benefits and drawbacks. These approaches are well documented and can be researched across many website resources.

Server side discovery is preferred for 2-IMMERSE, this will support:

- Interoperability of cloud hosted services, providing the required discovery for the array of service types and instances that need to interoperate within the server/cloud ecosystem. Server side Service Discovery can be scaled into a quorum of discovery instances to accommodate increasing levels of discovery requests as the 2-IMMERSE system scales.
- Tracking the operational health of registered service instances.
- Brokering access control policies between services.
- Load balancing service transaction requests across operational instances.
- Providing an administration console to visualise the operational status of service instances within the system.
- Client API Gateways. An API gateway provides a single addressable endpoint (URL domain name) for external clients to communicate with server side services. The API gateway provides a set of functional interfaces that require transactions with a collection of server side services to fulfil them. An API gateway can be either agnostic or specific to a client device type, the latter option provides tailored APIs to suit the specific requirements of the correlating device type. Client devices are

© 2-IMMERSE Consortium 2017

configured with the domain name of the correlating API gateway to connect with. These API gateways employ their own load balancing mechanism to build a quorum of gateway instances capable of servicing the volume of connected client devices. An API gateway quorum interfaces with the server side Service Discovery component to capture the addressable attributes of the available server service instances; facilitating subsequent service transactions within the context of executing client interface functions.

There are a number of established Service Discovery components, examples include:

- Netflix Eureka, a component originally used within Netflix OSS as part of a client side Service Discovery approach. Eureka has subsequently been adopted within the Spring Cloud infrastructure employed by server side discovery of Spring Cloud micro-services.
- Consul is a set of open source components, providing tools for service registration and discovery. Cisco has released Mantl as an open-source stack of tools used to build micro-service infrastructures. Mantl utilises a variety of tools for system integration, including Consul.

Cisco proposes adopting Mantl as the tool set employed to build and integrate the server side infrastructure for 2-IMMERSE. The open source stack provides opportunities to explore and understand the technical details of tools as required, and offers flexibility to provision the infrastructure through commercial cloud vendors (i.e. Cisco Intercloud Service or AWS) or private on premise lab environments.

Considering the availability and adoption of existing tools for Service Discovery, the following sections seek to capture a high-level abstraction of key interface verbs to clarify the respective roles required from service discovery.

3.1.1.2 Service Discovery Verbs

This section lists verbs used by the Service Discovery service.

3.1.1.2.1 Register

A provisioned service instance will call this to register availability as a hosted service within the system. The calling service will pass a collection of attributes to identify and address itself, including but not limited to:

- Service Type
- IP address (resolvable to Service Discovery)
- Port
- Health Check URL – Pre-defined and integrated end-point to retrieve health attributes.

The service registry will group this service instance with other instances of the same service type, then assigns and returns a unique service instance ID.

3.1.1.2.2 Renew

A registered service instance will call this to renew a registered status, operating as a service registration heartbeat. The calling service will pass the registered service instance ID. Failure to post a heartbeat within a configurable time frame results in automatic removal of the service instance from the registry.

 © 2-IMMERSE Consortium 2017

3.1.1.2.3 De-Register

A registered service instance or system administration console calls this to remove a service instance from the registry. The calling component will pass the registered service instance ID.

3.1.1.2.4 List All Services

A Service Discovery client will call this to query back all registered service type IDs and correlating instance IDs.

3.1.1.2.5 List All Service Instances

A Service Discovery client will call this to query back all registered service instances for a specified service type. The calling client uses this action to refresh a local cache of available service type instances.

The calling component will pass the service type ID.

3.1.1.2.6 Query Service Instance

A Service Discovery client will call this to query back service instance attributes sufficient to notify of service state and to facilitate an interface transaction with service instance end-points. The calling component will pass the registered service instance ID.

3.1.1.2.7 Put Service Instance Off-Line

A registered service instance or system administration console calls this to set an off-line status for a specified service instance. The calling component will pass the registered service instance ID. This status is included within the dataset returned as part of service instance query. An off-line status indicates a temporary out of service state, typically to perform maintenance.

3.1.1.2.8 Put Service Instance On-Line

A registered service instance or system administration console calls this to transition a specified service instance from an off-line to an on-line status. The calling component will pass the registered service instance ID.

3.1.1.2.9 Perform Health Checks

A Service Discovery client will call this to trigger a round-robin query of all registered service instances to pull back status attributes. The calling component can optionally pass a service type ID to filter queries to correlating service type instances. Note: The Service Discovery component will be configured to periodically retrieve status attributes of registered services. This action provides a manual override to trigger retrieval on demand.

3.1.1.2.10 Get Health Status

A Service Discovery client will call this to retrieve attributes describing the operational health of registered service instances. The calling component can optionally pass parameters to shortlist health status retrieval to a specific service type or instance ID.

3.1.1.2.11 Configure

Used by a System Administration entity to override default values of operational attributes of Service Discovery instances. This will be specific to the tool adopted by 2-IMMERSE, but should include:

- Hosting attributes to facilitate flexible deployment within a cloud environment
- Service Registration attributes i.e. renew heart beat intervals, service health check intervals etc.

3.1.1.2.12 Start

Called by a System Administration console to launch a Service Discovery instance. The mechanism is parameterised to start a new Service Discovery quorum with a master instance, or start an additional instance as part of an existing Service Discovery quorum.

3.1.1.2.13 Stop

Called by a System Administration console to terminate a Service Discovery instance. Addressing a master Service Discovery instance will terminate the correlating quorum of instances, otherwise the single Service Discovery instance will be terminated within the quorum.

3.1.1.2.14 Get Status

Called by a System Administration console to retrieve status attributes of a Service Discovery quorum. The available attributes will depend on the Service Discovery tool adopted by 2-IMMERSE. The attributes shall provide insight to the instances and their operational status within the quorum, including aggregated information regarding transaction times achieved across service discovery requests made from calling components.

3.1.1.3 Service Discovery Collaborators

3.1.1.3.1 Server Service Instances

Provisioned service instances will interface with Service Discovery to manage inclusion and status in the System Registry. Server service instances use Service Discovery to acquire the addressable attributes of available service instances they need to collaborate with to fulfil service functions.

3.1.1.3.2 External Client API Gateway

Use all the Service Discovery verbs to query for service instance end-points available within the server system. An API Gateway uses service discovery to acquire the addressable attributes of available service instances and implement algorithms to load balance requests across them.

3.1.1.3.3 System Administration Console

Use all the Service Discovery verbs to monitor and control the state of the Service Discovery quorum and registered system services.

3.1.2 API Specification

We have adopted Consul as the Service Registry implementation, as it is part of the Mantl platform. Consul provides a REST API which is documented at https://www.consul.io/docs/agent/http.html

The Mantl platform also includes Træfık; a modern HTTP reverse proxy and load balancer made to deploy microservices with ease. It supports several backends to manage its configuration automatically and dynamically. Using Træfık enables allows straightforward client access to microservices. Documentation for Træfık is available here: https://docs.traefik.io/

 © 2-IMMERSE Consortium 2017

3.2 Device Discovery

The Device Discovery service searches for devices that can be part of a DMAPP experience. Device Discovery isn't part of the service platform per se, but takes place between client devices in a home environment.

Latest component documentation: https://2immerse.eu/wiki/api/discovery-plugin/

3.2.1 Functional Description

3.2.1.1 Verbs

Key verbs used by the Device Discovery Service.

3.2.1.1.1 startDiscovery

Starts the device discovery process (see also "deviceListChange").

3.2.1.1.2 stopDiscovery

Stops the device discovery process.

3.2.1.1.3 getDevice

Returns information on a particular device that has been found during the discovery process (see also "deviceListChange"). The information includes endpoint points for app-to-app communication and inter-device media-synchronisation.

3.2.1.2 Listeners

This section include a list of listeners.

3.2.1.2.1 deviceListChange

References to discovered devices are maintained in a list. The deviceListChange listener is called in case this list changes. Changes can have different causes including:

- A new device has been discovered
- A known device disappeared for example because:
    - It has been switched of
    - It left the local network

3.2.1.2.2 deviceStatusChange

The deviceStatusChange listener is called in case the status of a particular device changes (see also "getDevice").

3.2.2 API Specification

A first implementation of the device discovery service is the DIAL Plug-In. The DIAL Plug-in is an abstraction of the DIAL protocol for discovery of HbbTV 2.0 devices. For discovered devices, it provides all communication endpoints for application launch, app-to-app communication and inter-device communication.
The API is described at https://gitlab-ext.irt.de/2-immerse/cordova-plugin-discovery, and is also documented in Annex A- DIAL Plug-In API Specifications.

3.3 Timeline

Latest service documentation: https://2immerse.eu/wiki/timeline/

3.3.1 Functional Description

The timeline service is responsible for the temporal orchestration of a DMApp (within a single household). Think: the director of an orchestra that reads the score and cues the individual musicians to play their parts. At service start-up it needs to be given the parameters needed to contact the layout service for this DMApp, the parameters it needs to get the score document (or possibly the document itself) and some way to subscribe to the timeline synchronisation service for this DMApp.

3.3.1.1 Verbs

The verbs used for the Timeline service are described below

3.3.1.1.1 createcontext

Sent by the layout service to indicate a new context has been created. The layout service needs to at least pass two parameters to indicate how the timeline service can access the layout service and the synchronisation service for this context, for example layoutServiceUrl and syncServiceUrl.

3.3.1.1.2 loaddmapp

Sent by the layout service to indicate a new Distributed Media Application is starting.
A parameter (for example timelineDocumentUrl) needs to be passed to indicate the document describing the orchestration for this DMApp.

Issue: either the timeline starts playing straight away, or a second call startTimeline is needed. It will now interpret the score (according to the current time updates provided to it by the sync service) and emit calls to the layout service to control starting and stopping of DMApp components (where media items such as specific images and such are considered a special case of a DMApp component).

3.3.1.1.3 stoptimeline

Informs the timeline service that this timeline has stopped playing. Note: unclear whether this call is needed.

3.3.1.1.4 timelineevent

Informs the timeline service that eventIdentifier has occurred, so it can adjust its timeline accordingly. The intention of this call is to allow alternate paths through the timeline (think: interactive quizzes, or secondary content on different levels such as beginner / intermediate / advanced that the user can switch between, or that the DMApp can switch between based on a user’s answers to previous quiz questions or something), and there must be a way for the DMApp to signal to the timeline service that such an alternative path has to be selected. Maybe this call needs a globalTimeSpecification if it can be scheduled for the future.

3.3.1.1.5 clockchanged

The exact form of this call (or calls) is unclear, at the moment, but the timeline service will need to be informed somehow when global (per-household) clock changes. The sync service will need to tell it.

 © 2-IMMERSE Consortium 2017

Discussion: It is not yet clear whether we also need pause and resume (but if we need the functionality maybe clockChanged can be used for that, by telling the timeline service that the clock speed is now zero). It is also not yet clear whether we need seek (and, again, these may be implementable using clockChanged).

3.3.1.2 Collaborators

The following verbs are needed by the timeline service, and implemented by the Layout Service.

3.3.1.2.1 initcomponent

Initialize a component and assign parameters (media URL, etc.) needed by a component. Note: the need for the parameters is based on a model where the DMApp generally contains abstract components, such as "a video renderer", and the actual video to use is encoded in the timeline document. An alternative model is that the DMApp itself already contains all the needed information, but this limits reuse of the DMApp (think: translating it into a different language).

Even if the component already knows all its parameters beforehand an initialize call is still needed: the intention is that it allows a component to do preparation work before it becomes visible and/or active. Think of things like prefetching media items and such.

3.3.1.2.2 startcomponent

Starts the component at a given time globalTimeSpecification. Ideally, this time will be in the future when the call is issued, so the timeline service can issue a couple of these calls to different components and know they will all start at exactly the same time. If the time given is already in the past the component should start as soon as possible.

3.3.1.2.3 stopcomponent

Stops the component at a given time globalTimeSpecification. See startComponent for timing issues.

3.3.1.2.4 modifycomponent

Modifies the internal (and possibly the external) state of a component at a given time globalTimeSpecification. Which bits of state can be modified remains to be seen (and probably depends on the component type), but one can think of media position, playback speed, etc. This call is needed so orchestrated changes (like a seek on the timeline, or jumping to alternative content) can be done in an orchestrated manner, under control of the timeline service.

Discussion: The order of the first three commands is always initComponent-startComponent-stopComponent. This set of verbs presumes that componentIdentifiers are like xml-ids and long lived. The initComponent parameters may not be needed for static DMApps, but will be needed when dynamically inserted content is used (for later use cases). The intention of including the globalTimeSpecification in the calls is that these calls can be issues prematurely, so the receivers have a chance to prepare (for instance by loading the media items). If a globalTimeSpecification specifies a time in the past the action should be executed as soon as possible.

3.3.2 API Specification
The API specification for the Timeline Service is documented in Annex B - 2-IMMERSE Timeline Service API documentation version v1

3.3.3 Timeline Document Format
The Timeline Service needs a Timeline Document Format; the definition of this is currently work in progress. Our design considerations for this format are presented in Annex C - Timeline Document Format Design Considerations.

3.4 Layout
Latest service documentation: https://2immerse.eu/wiki/layout/

3.4.1 Functional Description
This section provides a functional description of the layout service.

3.4.1.1 Responsibilities
The layout service is responsible for managing and optimising the presentation of a set of DMApp Components across a set of participating devices (i.e. a context).

The resources that the layout service exposes through its API are:

- context – one or more connected devices collaborating together to present a media experience
- DMApp (Distributed Media Application) – a set of software components that can be flexibly distributed across a number of participating multi-screen devices. A DMApp runs within a context.
- component – a DMApp software component

For a running DMApp (comprising a set of media objects / DMApp Components that varies over time); it's authored layout requirements, user preferences, and the set of participating devices in the context (and their capabilities), the layout service will determine an optimum layout of components for that configuration. It may be that the layout cannot accommodate presentation of all available components concurrently.

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

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

The layout model that the layout service will adopt is to be determined, but a range of options exist from very simple (a single component being shown full screen with a simple chooser), through to non-overlapping grid based arrangements, overlapping models such as Picture-in-picture, through to a full 3D composition of arbitrary shaped components.

Note that:

 © 2-IMMERSE Consortium 2017

- In the initial implementation, it will only be possible to run a single DMApp within a context. This constraint may be removed in the future (and the API shouldn't preclude this).
- From the user experience wire frames that WP3 are developing for the watching theatre at home service prototype, it looks likely that some kind of template based layout model will be adopted, with a template defining the size and position of target regions for particular devices / device types.
- All of the client-facing API's require a reqDeviceId parameter; this is for logging purposes so that the originating client device for each call is captured.
- The timeline service instructs the layout service which DMapp components are running, through init, start and stop API calls (these are not intended to be used by client devices).
- Context management will need to handle devices that stop participating within a context without explicitly leaving it, as might happen if someone took their device away without shutting down the app, or if the battery died (for example, by running timeouts for each device that reset whenever the device interacts).

3.4.1.2 Verbs

This section lists verbs used by the layout service.

3.4.1.2.1 CreateContext

A Client Web Application (typically the TV) will call this to create and automatically join a new context. The calling Client Web Application will pass a unique device ID and its capability metadata. The passed device ID will be joined to the context automatically (i.e. there is no need for this device to call JoinContext).

Issue: Does a new context need to be linked with a household/account ID? Or will this be derived from the device ID?

3.4.1.2.2 JoinContext

A Client Web Application (typically a companion device) will call this to join an existing context. The calling Client Web Application will pass the context ID, its unique device ID and its capability metadata.

NB: for a companion device, we assume that an existing context ID will have been passed to it by the TV over App2App communication following the DIAL device discovery process.

3.4.1.2.3 GetContext

A Client Web Application can call this to determine if it is a member of a context, and if so, which other devices are also members.

3.4.1.2.4 LeaveContext

A Client Web Application will call this to leave an existing context. The calling Client Web Application will pass the context ID and its unique device ID.

NB: if all of the devices leave a context then it will be implicitly destroyed.

3.4.1.2.5 DestroyContext

A Client Web Application will call this to destroy an existing context. The calling Client Web Application will pass the context ID and its unique device ID.

3.4.1.2.6 DeviceOrientationChange

A Client Web Application will call this to notify the layout service of its host device orientation changing.

3.4.1.2.7 GetDMApps

A Client Web Application will call this to get a list of running DMApp(s) for this context. The calling Client Web Application will pass the context ID.

3.4.1.2.8 LoadDMApp

A Client Web Application will call this to request that a DMApp (Distributed Media Application) is loaded in the specified context. The calling Client Web Application will pass the device ID, context ID and URLs for DMApp Timeline Document & Layout Requirements.

In response, the layout service will load the required layout requirements. The Timeline service will be called to load and run the corresponding timeline document.
In the initial implementation, it will only be possible to run a single DMApp within a context. If a DMApp is loaded, then subsequent LoadDMApp calls will fail until that DMApp is removed.

NB – The layout requirements will be validated when loaded by the service and the loadDMApp call will return an error if the requirements are invalid / malformed.

3.4.1.2.9 UnloadDMApp

A Client Web Application will call this to request that a DMApp is removed from the context it is running in.

3.4.1.2.10 GetDMAppInfo

A Client Web Application will call this to get information about the DMApp, this will include the current DMApp Component list.

3.4.1.2.11 clockChanged

Proxy API call for a timeline service instance; informs the timeline server of the current mapping of wallclock to presentation clock

NB: The component management verbs below (hide/show/move/clone) may require end-user role / permissions for some use cases, but this is not required for the Watching Theatre at Home trial.

3.4.1.2.12 InitComponent

A Timeline Service instance will call this to initialize a DMApp component for the specified DMApp (not a client device API). This will include configuration info that a Client Web Application will use to instantiate the DMApp Component.

3.4.1.2.13 StartComponent

A Timeline Service instance will call this to start a DMApp component for the specified DMApp at a specified time (not a client device API)

 © 2-IMMERSE Consortium 2017

3.4.1.2.14 StopComponent

A Timeline Service instance will call this to stop a DMApp component for the specified DMApp at a specified time (not a client device API)

3.4.1.2.15 GetComponentInfo

A Client Web Application will call this to get information about the specified DMApp Component, including layout information.

3.4.1.2.16 HideComponent

A Client Web Application will call this to request that the specified DMApp component is hidden (i.e. removed from layout). The calling Client Web Application will pass the context ID, component ID and its device ID.

3.4.1.2.17 ShowComponent

A Client Web Application will call this to request that the specified DMApp component is shown (i.e. restored to layout). The calling Client Web Application will pass the context ID, component ID and its device ID.

3.4.1.2.18 MoveComponent

A Client Web Application will call this to request that the specified DMApp component is moved to a specific device (and optionally, a position on that device). The calling Client Web Application will pass the context ID, component ID, it's device ID, the target device ID and optionally a position.

NB: The state of the DMApp Component will need to be migrated to the target device, this will be done using the SaveState / RestoreState API calls.

3.4.1.2.19 CloneComponent

A Client Web Application will call this to request that the specified DMApp component (i.e. a new instance is created) to a specific device (and optionally, a position on that device). The calling Client Web Application will pass the context ID, component ID, it's device ID, the target device ID and optionally a position.
Issue – this will require involvement of the timeline service. The state of the DMApp Component will need to be migrated to the new instance, this will be done using the SaveState / RestoreState API calls.

3.4.1.2.20 Status

A Client Web Application will call this to send DMApp Component status updates to the service.

3.4.1.2.21 SaveState

A Client Web Application will call this to save DMApp Component state (typically when migrating a DMApp component between devices). The DMApp Component state is supplied as part of the call.

3.4.1.2.22 RestoreState

A Client Web Application will call this to retrieve previously saved DMApp Component state (typically when migrating a DMApp component between devices). The DMApp Component state is returned in response to the call.

3.4.1.2.23 ChangeImmersion
If the layout service supports the concept of immersion per Fresco, the layout service will have a verb to support changing immersion level (TBD)

3.4.1.2.24 AudioPresentation
For object based audio presentation, the layout service might have an API for managing this (TBD)

3.4.1.2.25 LayoutUpdate
Whenever the layout changes, an update is pushed to affected listening Client Web Applications via a suitable mechanism (e.g. web sockets). These updates will be pushed as a set of 'deltas', rather than by sending the complete layout, which would require the Client Web Application to compare current state with the updated state and derive the deltas. This will include notification of the need to save state DMApp state in the event of component migration to another device.

Should the Client Web Application require a complete set of layout information, it can use the GetDMAppInfo API call.

3.4.1.3 Collaborators
This section lists the services with which the Layout service will collaborate.

3.4.1.3.1 Timeline
- CreateContext and LoadDMApp calls will be forwarded to the Timeline service.
- DMApp component init, start and stop calls are listened for

3.4.1.3.2 Client Web Application
- All of the verbs listed above are called by Client Web Application instances with the exception of DMApp component init, start and stop calls

3.4.2 API Specification
The API specification for the Layout Service is documented in Annex D- 2-IMMERSE Layout Service API documentation version v1

3.4.3 Layout Requirements Document Format
The layout Service needs a layout Requirements Document Format; the definition of this is currently work in progress.

3.5 Server-Based Composition
Out of scope for initial Watching Theatre at Home trial.

 © 2-IMMERSE Consortium 2017

3.6 Timeline Synchronisation

Latest service documentation: https://2immerse.eu/wiki/sync/

The Timeline Synchronisation solution in 2IMMERSE is a collection of services, protocols and components that enable DMApp components on devices participating in an experience to synchronise to a source of timing information representing the progress of the experience. The temporal progress of the experience is represented by a timeline called the **Synchronisation Timeline**. Synchronisation of media objects is achieved by individual devices receiving progress information from a Synchronisation Timeline source. Each device then aligns the playback of its DMAppc components with the Synchronisation Timeline. Timeline Synchronisation in 2IMMERSE seeks to support both **intra-home synchronisation** (also called interdevice synchronisation) and **inter-home synchronisation**. Our solution combines standardised mechanisms for interdevice synchronisation in the home (such as DVB-CSS) with cloud-based services and proposes new protocols to achieve its distributed synchronisation offering.

The APIs of salient components / services are specified in this section. A more detailed description of the use of these components and services to achieve intra-home and inter-home synchronisation is provided in Annex F - 2-IMMERSE Timeline Synchronisation.

3.6.1 Synchronisation Service

This section describes the functional role of the synchronisation service.

3.6.1.1 Responsibilities

The Synchronisation Service provides operations for the creation and management of microservice instances that direct the synchronisation of distributed DMApp components for a Distributed Multimedia Application (DMApp). Its primary role is to provide support for distributed media synchronisation in the form of a cloud-based service to disseminate timeline progress updates to devices e.g. for inter-home media synchronisation. However, it can also be used in use cases involving *intra-home media synchronisation* to provide a common time reference between devices in the home and an external entity that is directing the experience (for example, a cloud-based Timeline Service).

The usage pattern for the Synchronisation Service is as follows. If the Timeline Service needs to provide synchronisation capabilities for a Distributed Multimedia Application (an experience session), it requests the creation of a SyncService instance via the Synchronisation Service API. A SyncService object is a per-session entity that consists of a WallClock synchronisation (WCSync) microservice coupled with a Timeline Synchronisation (TimelineSync) microservice.

The **WallClock synchronisation (WCSync) microservice** offers time synchronisation capabilities to distributed applications via the WCSync protocol.

The **Timeline Synchronisation (TimelineSync) microservice** receives timeline updates from a timeline source (the Synchronisation Timeline source) and disseminates these updates to the applications that are connected to it using the CSS-TS protocol.

3.6.1.2 Verbs

This section lists the verbs used by the synchronisation service.

3.6.1.2.1 CreateSyncService

A Timeline Service instance will call this to create an instance of the SyncService object to offer synchronisation capabilities for all devices in a session. A SyncService object will obtain a WCSync (WallClock synchronisation) microservice instance and a TimelineSync (Timeline Synchronisation) microservice instance and will ensure that both a running on the same system host.

3.6.1.2.2 InitSyncService

A Timeline Service instance will call this to initialise a SyncService instance with a timeline selector string describing the type of the timeline to be used for synchronisation. The SyncService's ability to emit events/callbacks is also enabled. A SyncService instance is identified by a session identifier (sessionId).

3.6.1.2.3 DestroySyncService

A Timeline Service instance will call this to close the service endpoints associated with this SyncService and destroy this SyncService instance.

3.6.1.2.4 EnableSynchronisation

A Timeline Service instance will call this with a sessionId to enable the synchronisation operation of a SyncService instance. The SyncService will

1) allow clients to connect to its WallClock synchronisation microservice to perform clock synchronisation and
2) forward Synchronisation Timeline progress updates received from a master entity to its TimelineSync microservice clients.

3.6.1.2.5 DisableSynchronisation

A Timeline Service instance will call this with a sessionId to disable the synchronisation operation of a SyncService instance. A callback/event is emitted by the SyncService instance, once WallClock Sync and TimelineSync microservices associated with this SyncService are stopped.

3.6.1.2.6 GetWallClockSyncURL

A Timeline Service instance will call this with a sessionId to obtain the WCSync microservice endpoint for that SyncService instance.

3.6.1.2.7 GetTimelineSyncURL

A Timeline Service instance will call this with a sessionId to obtain the TimelineSync service endpoint for that SyncService instance. This is the endpoint from which Synchronisation Timeline updates are received from clients via the TimelineSync protocol.

3.6.1.2.8 GetTimelineSyncMasterURL

A Timeline Service instance will call this with a sessionId to obtain the TimelineSync service master endpoint for that SyncService instance. This is the endpoint a sync master uses to push Synchronisation Timeline updates to sync slaves.

3.6.1.2.9 GetNrOfSlaves

A Timeline Service instance will call this with a sessionId to poll the number of synchronisation slaves connected to this SyncService's TimelineSync service instance.

 © 2-IMMERSE Consortium 2017

3.6.1.2.10 GetLastUpdate
A Timeline Service instance will call this to return the last timeline update that the SyncService's TimelineSync microservice received.

3.6.1.3 Required Events
This section lists the event definitions required for the Timeline service.

3.6.1.3.1 TimelineServiceUnavailable
If the Timeline Service that created this SyncService instance becomes unavailable, this SyncService is notified via a TimelineServiceUnavailable event. This event may be generated by a service (e.g. HashiCorp's Consul) who is responsible for monitoring the health of microservices. he SyncService will start the procedure to close down service endpoints and allow itself to be destroyed. This avoids the case of orphaned SyncService instances, should their Timeline Service instance close down due to failure.

3.6.1.4 Provided Events
This section lists the event definitions provide by the Timeline service

3.6.1.4.1 SyncEndpointsAvailable
This event/callback is emitted when WCSync and TimelineSync service endpoints become available after synchronisation is enabled on a SyncService instance (i.e. EnableSynchronisation operation invoked). The Timeline Service should listen to this event and share the WCSync and TimelineSync service endpoints URL to devices in this session via the Layout Service.

3.6.1.4.2 SyncEndpointsClosed
This event/callback is emitted when WCSync and TimelineSync service endpoints are closed. This will happen, for example, when synchronisation is disabled on a SyncService instance.

3.6.1.4.3 SyncTimelineUnavailable
This event/callback is emitted to all synchronisation slaves when a synchronisation master becomes unavailable due to failure or network partitions.

3.6.1.5 Collaborators
- Timeline Service
- Client applications using SyncKit library

3.6.1.6 API Specification

void createSyncService (String sessionId)			
Creates an instance of the synchronisation service to offer cloud-based synchronisation capabilities to all devices in this session.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	An identifier for this experience session.

void initSyncService() (String timelineSelector, CorrelationList correlations Callback timelinesUnavailableCallback)			
Initialises a SyncService for cloud-based synchronisation with a timeline from a master device/entity.			
Parameters	*Name*	*Type*	*Description*
	timelineSelector	String	Type and location of the timeline to be used by the SynchronisationService API.
	correlations	CorrelationList	For each DMAppComponent, a correlation timestamp mapping the component's media object timeline to the synchronisation timeline
	timelinesUnavailableCallback	Callback	callback triggered when one or more synchronisation timelines become unavailable.

void enableSynchronisation (String sessionId ,Callback onSyncURLsAvailable)			
Enables the SyncService synchronisation capabilities. The SyncService creates server endpoints for wallclock synchronisation and for timeline update propagation. The nrOfSlaves property can be used to poll for the number of connected slave TV/companion applications.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify the SyncService instance.
	onSyncURLsAvailable	callback	callback to report TS/WC server URL endpoints when they become available

void disableSynchronisation (String sessionId, Callback onEndpointsClosed)

 © 2-IMMERSE Consortium 2017

Disables the synchronisation service for this session.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify the SyncService instance.
	onEndpointsClosed	Callback	Optional callback function.

String getWCSyncURL (String sessionId)			
Returns the WallClock synchronisation endpoint URL as a string. This is the WC-Sync protocol server endpoint devices connect to, to perform application-level WallClock synchronisation.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify SyncService instance.

String getTimelineSyncURL (String sessionId)			
Returns the Timeline Synchronisation server endpoint URL as a string. This is the protocol endpoint devices connect to, to receive synchronisation-timeline updates (also known as Control Timestamps). Devices can also submit their own current time position – this should be its media object's time converted to the corresponding time on the Synchronisation Timeline.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify SyncService instance.

String getTimelineSyncMasterURL (String sessionId)			
Returns the Timeline Synchronisation master server endpoint URL as a string. This is the protocol endpoint devices/entities connect to, to send synchronisation-timeline updates (also known as Control Timestamps). Synchronisation-timeline updates received from the TSMaster protocol client are forwarded to all clients connected to the Timeline Synchronisation server endpoint.			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify SyncService instance.

void setContentId (String sessionId , String contentId)			
Optional operation to set a content identifier to specify the content currently played by a synchronisation master (if the master entity is a DMApp component).			
Parameters	*Name*	*Type*	*Description*
	sessionId	string	A session identifier to identify SyncService instance.
	contentId	String	Content identifier

3.6.2 WallClock Synchronisation Service (WCSync)

Latest service documentation: https://gitlab-ext.irt.de/2-immerse/sync-protocols

3.6.2.1 Responsibilities

To facilitate the distributed synchronisation of the media, each terminal presenting or directing the presentation of media need to have a common notion of time. This is achieved through the maintenance of a Wall Clock that is synchronised to a master WallClock. This is the real-time clock against which the progress of media playback or the progress of the whole experience can be measured.

The WallClock Synchronisation service provides a lightweight time synchronisation mechanism by implementing the server functionality of the WCSync protocol algorithm. It responds to Wall Clock Synchronisation protocol requests from slave HbbTV terminals or Companion Screen Applications to synchronise their own internal Wall Clock with that of the service.

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

A description of the WallClock synchronisation protocol is available in Section 13.7 of the HbbTV 2.0 specification.

3.6.2.2 Collaborators

This section lists the Objects with which the Wall Clock Synchronisation Service collaborates.

3.6.2.2.1 SyncKit's Synchroniser Object

A Synchroniser object is a particular implementation of the SyncKit framework's Synchroniser API. It allows applications to synchronise with a source of timeline updates (e.g. a cloud-based SyncService instance). The Synchroniser object uses the Wall Clock Synchronisation protocol to synchronise its own internal Wall Clock with that of the service.

3.6.3 Timeline Synchronisation Service (TimelineSync)

Latest service documentation: https://gitlab-ext.irt.de/2-immerse/sync-protocols

3.6.3.1 Responsibilities

To facilitate distributed synchronisation of the presentation of media, the Timeline Synchronisation Service, implements a CSS-TS service endpoint. A slave terminal or Companion Screen Application connects to the CSS-TS service endpoint to establish a session of the Timeline Synchronisation Protocol.

This protocol conveys messages containing setup-data and Control Timestamps and Actual, Earliest and Latest Presentation Timestamps that relate Wall Clock time to the Synchronisation Timeline.
A description of the CSS-TS Timeline synchronisation protocol is available in Section 13.8 of the HbbTV 2.0 specification.

 © 2-IMMERSE Consortium 2017

3.6.3.2 Collaborators

This section lists the objects with which the Timeline Synchronisation Service collaborates.

3.6.3.2.1 SyncKit's Synchroniser Object

A Synchroniser object is a particular implementation of the SyncKit framework's Synchroniser API for synchronising with a source of timeline updates (e.g. a cloud-based SyncService instance). The Synchroniser Object uses the Timeline Synchronisation protocol (TimelineSync service) to request for and receive synchronisation timeline updates.

3.6.4 SyncKit's Synchroniser API

Latest service documentation: https://gitlab-ext.irt.de/2-immerse/synckit

3.6.4.1 Responsibilities

The SyncKit framework is a collection of native and browser-based components that provides synchronisation facilities to both native and browser-based applications on mobile devices. Its browser-based components can also be used in HbbTV application environments to synchronise the TV's content to an external source of timing information.

A **Synchroniser Object** is a particular implementation of the Synchroniser API that supports one of these synchronisation modes:

1. *Distributed synchronisation* - the synchronisation timeline source is outside the local network e.g. a cloud-based service. In this mode, it uses the WallClock Synchronisation (WCSync) microservice available in the SyncService operating for that session, to synchronise its internal WallClock. To request for and receive updates about the progress of the synchronisation timeline, it submits a request to the TimelineSync microservice using the aforementioned Timeline Synchronisation protocol (Section 3.6.3). This protocol then delivers timeline updates as Control Timestamps to the Synchroniser Object.
2. *Interdevice synchronisation* - the synchronisation timeline source is on the same network e.g. an HbbTV terminal. A Companion Screen Application will use the Synchroniser Object to synchronise its DMAppComponents against the Synchronisation Timeline advertised by an HbbTV 2.0 television. This object uses the DVB-CSS protocols to achieve inter-device synchronisation. It uses CSS-CII protocol to be notified about the TV's content identifier and to discover the TV's wallclock/timeline synchronisation service endpoints. Internally, it connects to the CSS-WC Wall Clock synchronisation protocol endpoint to synchronise it WallClock to that of the TV. It requests for Synchronisation Timeline updates for that particular content (the contentId is included in the request) via the CSS-TS timeline synchronisation protocol (described in Section 13.8 of the HbbTV 2.0 specification).

Applications create and initialise a Synchroniser Object with the location of the Synchronisation Timeline source. From this location, the Synchroniser object can retrieve the location of the WCSync and TimelineSync service endpoints. Applications can use the Synchroniser Object in two ways:

1. register for periodic time updates about the current time on the Synchronisation Timeline
2. create a SyncController object and plug it in a DMApp component to synchronise its playback

© 2-IMMERSE Consortium 2017

3.6.4.2 Verbs

This section lists the verbs used by the Synchroniser Object.

3.6.4.2.1 getSynchroniser

A web application (on the companion device or TV) will call this to obtain a (CSS-)Synchroniser singleton object for synchronising its DMApp components to an external source of timing information.

3.6.4.2.2 destroySynchroniser

A web application will call this to destroy the (CSS-)Synchroniser object. This will result in existing connections to protocol endpoints to be closed.

3.6.4.2.3 initSynchroniser

A web application will call this to initialise a Synchroniser object for *interdevice* synchronisation or for *distributed* synchronisation based on the parameters supplied. It detects the sync mode: either **interdevice** or **distributed**, from the timeline source supplied. The Synchroniser object uses a Synchronisation Service URL to retrieve the Wall Clock Synchronisation service and the Timeline Synchronisation service endpoints. These can be either service endpoints on the TV (CSS-WC and CSS-TS protocol servers respectively) or microservice interface locations on the cloud (WC-Sync and TimelineSync microservices respectively). The Synchroniser object is also initialised with a timeline selector – a string that describes the type of the timeline to be used for synchronisation. This is necessary for intra-home synchronisation use cases as HbbTV televisions can access a number of timelines signalled in broadcast or IP-delivered streams.

3.6.4.2.4 enableSynchronisation

A web application will call this to start the Synchroniser object. This will trigger connection requests to be sent to synchronisation services. The availability of the Synchronisation Timeline is signalled to the application via an event/callback.

3.6.4.2.5 disableSynchronisation

A web application will call this to stop the Synchroniser object. The completion of the process to close all outbound connections to synchronisation protocol endpoints is signalled via an event/callback.

3.6.4.2.6 registerForTimelineUpdates

A web application will call this to receive periodic updates about the current time on the Synchronisation Timeline. This is the time on the Synchronisation Timeline at the moment the update is sent.

3.6.4.2.7 getCurrentTime

A web application will call this to get the current time on the Synchronisation Timeline.

3.6.4.2.8 getCurrentWallClockTime

A web application will call this to get the current time on the local WallClock that is synchronised with an external WallClock.

 © 2-IMMERSE Consortium 2017

3.6.4.2.9 createSyncController

A web application will call this to create a SyncController object to handle the synchronisation of a media object (DMAppComponent). A CorrelationTimestamp[1] is specified to enable timestamps to be converted from Synchronisation Timeline and media object timeline and vice-versa.

3.6.4.2.10 getContentId

A web application will call this to obtain the synchronisation master's current content identifier. For use cases where inter-home synchronisation is involved, an empty string will be returned as the synchronisation master is the Timeline Service.

3.6.4.2.11 getApp2AppURL

A web application will call this to obtain the URL for an App2App service that enables bi-directional communication between applications.

3.6.4.2.12 getMasterIPAddr

A web application will call this to obtain the synchronisation master's URL. For inter-device sync, the TV's IP address on the local network is returned. For synchronisation using cloud-based services, the IP address of the SyncService instance is returned.

3.6.4.2.13 getMasterFriendlyName

A web application will call this to obtain the synchronisation service host name.

3.6.4.2.14 getSyncURL

A web application will call this to obtain the synchronisation service endpoint location. For inter-device sync, the TV's CSS-CII protocol endpoint is returned. For synchronisation using cloud-based services, the Synchronisation Service's URL is returned.

3.6.4.2.15 getSyncTimeline

A web application will call this to get an object describing the Synchronisation Timeline (timeline selector, unitsPerTick, unitsPerSecond and accuracy)

3.6.4.2.16 setSyncAccuracy

A web application will call this to specify a synchronisation accuracy threshold. It will be notified if the synchronisation accuracy degrades beyond this threshold.

3.6.4.3 Events

This sections lists the Events relevant for the Synchroniser object

3.6.4.3.1 WallClockSynced

Event to notify the application that the internal WallClock is in a synchronised state

3.6.4.3.2 SynchronisationTimelineAvailable

Event to notify the application that the Synchronisation Timeline is available at the device i.e. the device has started receiving Sychronisation Timeline updates from the synchronisation master.

[1] A pair of timestamps (master time, media time) mapping synchronisation timeline time to a media object time.

3.6.4.3.3 CurrentMasterTimeUpdate
Event to notify the application about the current time on the Synchronisation Timeline.

3.6.4.3.4 ContentIdChanged
Event to notify the application about a change in the contentId e.g. when a channel was changed on the TV or the master device started playing another media object.

3.6.4.3.5 LowSyncAccuracy
Event to notify the application about the degradation of synchronisation accuracy beyond the threshold specified.

3.6.4.3.6 SynchronisationTimelineUnAvailable
Event to notify the application about the unavailability of the Synchronisation Timeline. This is can be due to disconnections with the WallClock synchronisation and/or Timeline Synchronisation protocol endpoints.

3.6.4.4 Collaborators
This section lists onbjects with which the Synchroniser object collaborates.

3.6.4.4.1 Cloud-based SyncService
A Synchroniser object will request a SyncService for its WCSync and TimelineSync microservice interface URLs.

3.6.4.4.2 Cloud-based WCSync microservice
A Synchroniser object will submit a WallClock synchronisation request to the WCSync microservice.

3.6.4.4.3 Cloud-based TimelineSync microservice
A Synchroniser object will request for Synchronisation Timeline updates.

3.6.4.4.4 HbbTV CSS-CII Service Endpoint
A Synchroniser object will request an HbbTV for its CSS-WC and CSS-TS service endpoints.

3.6.4.4.5 HbbTV CSS-WC Service Endpoint
A Synchroniser object will submit a WallClock synchronisation request to the CSS-WC server on the TV.

3.6.4.4.6 HbbTV CSS-TS Service Endpoint
A Synchroniser object will submit a WallClock synchronisation request to the CSS-TS server on the TV.

3.6.4.5 Properties

Name	Description
contentId	A content identifier for the content currently shown on the master device
app2AppURL	CSA endpoint for the App-to-App-Communication channel
masterIPAddr	IP address of master device
masterFriendlyName	Friendly name for master device

 © 2-IMMERSE Consortium 2017

Name	Description
SyncURL	The selected master device's URL for interdevice synchronisation (CSS-CII endpoint)
syncTimeline	A timeline object with properties such as timeline selector, unitsPerTick, unitsPerSecond and accuracy.
syncAccuracy	The synchronisation accuracy threshold for the Synchroniser object. Callback function onSyncStateUpdate is invoked if the current accuracy exceeds syncAccuracy.

3.6.4.6 API Specification

void initSynchroniser()(String sync_url,String timelineSelector)			
Initialises a Synchroniser object for *interdevice* synchronisation or for *distributed* synchronisation based on the parameters supplied. It detects the browser's platform and configures itself by loading platform specific modules. From the timeline source supplied, it detects the sync mode: either **interdevice** or **distributed**.			
Parameters	*Name*	*Type*	*Description*
	sync_url	String	Inter-device sync: CSS-CII server Cloud-sync: SyncService URL
	timelineSelector	String	Type and location of the timeline to be used by the Synchroniser for synchronisation.

void enableSynchronisation (float syncAccuracy, Callback onSyncStateUpdate, Callback onContentIdChange)			
Starts the Synchroniser object's sync operation. This will trigger connection requests to be sent to synchronisation services. The availability of the Synchronisation Timeline is signalled to the application via an event/callback			
Parameters	*Name*	*Type*	*Description*
	syncAccuracyMin	float	Desired sync accuracy level
	onSyncStateUpdate	Callback	Optional callback to receive updates e.g. WallClock syn'ced
	onContentIdChange	Callback	Optional callback to receive contentId change notifications

void registerForTimelineUpdates (Callback timelineUpdateCallback)			
Registers the application to receive periodic updates about the synchronisation (master) timeline progress.			
Parameters	*Name*	*Type*	*Description*
	timelineUpdateCallback	Callback	Callback function to notify the application of the current time on the Synchronisation Timeline. This is the time in seconds on the master timeline.

void disableSynchronisation (function callback)			
Disables the interdevice/inter-location synchronisation of an application. Closes connections to synchronisation protocol endpoints. The Synchroniser leaves the media objects in their current state. The Synchroniser object goes back to the initialized state. The enableSynchronisation() method can be called again to enable sync.			
Parameters	*Name*	*Type*	*Description*
	callback	function	Optional callback function.

SyncController createSyncController (CorrelationTimestamp correlation, float resync_interval, function resyncStatusCallback)			
Creates a JS synchronisation controller object that can be used to control the playback of a video player, audio player or HTML slide show.			
Parameters	*Name*	*Type*	*Description*
	correlation	Correlation Timestamp	Correlation between media object timeline and synchronisation timeline
	resync_interval	float	Interval for evaluating DMAppC's playback progress
	callback	function	Optional callback function to report SyncController status

float currentTime ()
Returns the current time in seconds on the master timeline (i.e. the synchronisation timeline). This timestamp has at least millisecond precision.

3.6.5 HbbTV's Media Synchroniser API

3.6.5.1 Responsibilities

HbbTV 2.0 specifies a MediaSynchroniser object to allow TV applications to synchronise their streams or to allow companion device applications to synchronise their content against a TV stream.

If an HbbTV App will participate in multi-stream or inter-device synchronisation (as a slave or master), it creates a MediaSynchroniser object. A JS-based API is available for creating and manipulating a MediaSynchroniser object. The API's factory object is available on the global scope in the HbbTV App environment via a plug-in.

3.6.5.2 Collaborators

- SyncKit's Synchroniser Object (in interdevice sync mode)

3.6.5.3 API Specification

The MediaSynchroniser API is specified in clause 8.2.3 of the HbbTV 2.0 specifications. Relevant parts of the API are reproduced here for the sake of completion.

void initMediaSynchroniser() Object mediaObject, String timelineSelector)

 © 2-IMMERSE Consortium 2017

Initialises a MediaSynchroniser for multi-stream synchronisation and for inter-device synchronisation as a master terminal.After this method has been called, it is only possible to use this MediaSynchroniser object as a master for multi-stream synchronisation and/or inter-device synchronisation.			
Parameters	*Name*	*Type*	*Description*
	mediaObject	A/V Element	The media object (video/broadcast object, AV Control object, or an HTML5 media object) that carries the timeline that will be used by the MediaSynchroniser API.
	timelineSelector	String	Type and location of the timeline to be used by the MediaSynchroniser API.

void initSlaveMediaSynchroniser (String css_ci_service_url)			
Initialises a slave MediaSynchroniser for inter-device synchronisation of the presentation of media objects on this terminal (referred to as the slave terminal) and the media presentation on another terminal (referred to as the master terminal). After this method has been called, it is only possible to use this MediaSynchroniser object as a slave for multi-stream synchronisation and/or inter-device synchronisation.			
Parameters	*Name*	*Type*	*Description*
	css_ci_service_url	String	The URL of a DVB CSS CII endpoint at the master terminal.

void addMediaObject (Object mediaObject, String timelineSelector, CorrelationTimestamp correlationTimestamp, Number tolerance, Boolean multiDecoderMode)			
Adds a media object, i.e. video/broadcast object, AV Control object or HTML5 media object, to the MediaSynchroniser. If the MediaSynchroniser was initialised with the initMediaSynchroniser() method, or if inter-device synchronisation has been enabled, then the terminal shall start to synchronise the media object to other media objects associated to this MediaSynchroniser as a result of this method call.			
Parameters	*Name*	*Type*	*Description*
	mediaObject	A/V Element	Video/broadcast object, AV Control object, or an HTML5 media object
	timelineSelector	String	Type and location of the timeline to be used by the MediaSynchroniser API.
	correlationTimestamp	Correlation Timestamp	An optional initial correlation timestamp that relates the media objects timeline to the synchronisation timeline.
	tolerance	Number	An optional synchronisation tolerance in milliseconds.
	multiDecoderMode	Boolean	An optional parameter that defines whether component selection for this media object is performed separately (as defined in clause 10.2.7.3) or collectively with other media objects on this MediaSynchroniser (as defined in clause 10.2.7.4).

void removeMediaObject (Object mediaObject)

Removes a media object from this MediaSynchroniser.			
Parameters	*Name*	*Type*	*Description*
	mediaObject	Object	The media object to be removed.

void enableInterDeviceSync (function callback)			
Enables inter device synchronisation of a master terminal or slave terminal. If it is already enabled then this call shall be ignored. If the MediaSynchroniser was initialised using the initMediaSynchroniser() method then the terminal become a master terminal as defined in clause 13.3.3. If the MediaSynchroniser was initialised using the initSlaveMediaSynchroniser() method then the terminal become a slave terminal as defined in clause 13.3.5. The callback method shall be called when the endpoints are operable. The nrOfSlaves property can be used to poll for the number of connected slave terminals or companion applications.			
Parameters	*Name*	*Type*	*Description*
	callback	function	Optional callback function.

void disableInterDeviceSync (function callback)			
Disables the inter device synchronisation of a master or slave terminal. If the terminal is a master terminal it shall cease to be a master terminal as defined in clause 13.3.4. Once the terminal is no longer a master terminal then the callback function shall be called. If the terminal is a slave terminal it shall cease to be a slave terminal as defined in clause 13.3.6. Once the terminal is no longer a slave terminal then the callback function shall be called.			
Parameters	*Name*	*Type*	*Description*
	callback	function	Optional callback function.

3.6.6 DMAppC and DMAppC Control API

3.6.6.1 Responsibilities

A DMApp component is a browser-based element (e.g. a widget) that loads a media object (e.g. video, audio, web slide-show) and starts/stops playback as directed by an external entity. A DMAppControl API provides media playback control operations. These are used by SyncController objects to adapt the playback to be in sync.

3.6.6.2 Collaborators

- SyncKit's SyncController

3.6.6.3 DMAppC Properties

Name	**Description**
contentId	A content identifier for the content currently loaded in the DMApp component
mediaType	Media type e.g. "video/mp4, audio/aac"
playerType	Media player e.g. "HTML5 video element", "native AVPlayer", "Text Scroller", "HTML slide-show"

 © 2-IMMERSE Consortium 2017

Name	Description
Time	Current time of media
Speed	Speed of playback
duration	Length of media
loadedTimeRanges	Buffered time periods of media playback
offsets	Audio offset, video offset
isPlaying	Boolean set to true if playback has started

3.6.6.4 DMAppC Control API

void play ()
Starts the playback of the media object loaded in the DMApp component.

void stop()
Stops playback

float[] getAvailableTimeRanges ()
Returns the time range of the content that is currently buffered by the DMApp component.

void setSpeed(float speed)			
Sets the speed of playback.			
Parameters	*Name*	*Type*	*Description*
	speed	float	A value ranging from 0.0 to 2.0

void seekTo(float position, Optional function completionCallBack)			
Seek to time position in content.			
Parameters	*Name*	*Type*	*Description*
	speed	float	A value ranging from 0.0 to 2.0
	completionCallBack	callback	Optional function callback to report when playback adaptation has finished.

void setCorrelation(float hostTime, float contentTime, Optional function completionCallBack)			
Seek to time position in content.			
Parameters	*Name*	*Type*	*Description*
	hostTime	float	System time on the host device
	contentTime	float	Time position on media timeline
	completionCallBack	callback	Optional function callback to report when playback adaptation has finished.

3.7 Content Protection and Licensing

For the home theatre trial, this is only access control to content. Propose we use CENC and a key pre-loaded on the TV device.

Latest service documentation: https://2immerse.eu/wiki/content-protection/

3.7.1 Functional Description

Content protection of media assets involves mechanisms to prevent users of third parties consuming or acquiring content outside of authorised usage scenarios.

This generally includes a mechanism by which media data is encrypted using one or more encryption keys, and these key(s) are communicated from a licensing/key authority to the playback device out of band either in advance, at the start of playback, or during playback, subject to satisfying any licensing/authorisation requirements.

3.7.2 API Specification

A remote API would be required in the case where the required key(s) to play the content are not present on the device, and must be acquired to begin/continue playback.

In the case where an existing DRM system was being used, the API and endpoints used would be as specified for that system.

A possible API for a custom, simple DRM scheme suitable for use with MPEG-DASH CENC protected media on a TV-emulator device may include the following:

3.7.2.1 Get playback key(s):

Request encryption key(s) for an item of content.

```
{
  "message" : "get_keys",
  "cid" : "content ID",
  "kid" : [ "key ID", ... ],
  "auth" : { relevant authentication details/token(s) as may be defined in identity/auth module }
}
```

3.7.2.2 Get playback key(s) response:

```
{
  "message" : "get_keys_response",
  "cid" : "content ID",
  "kid" : [ "key ID", ... ],
  "keys" : [ "key value", ... ]
}
```

Where ‘cid’ (content ID) and ‘kid’ (key ID) are as defined and embedded within the MPEG-DASH CENC media.

In the case of MPEG-DASH CENC, content IDs, key IDs and key values can be assumed to be opaque 128-bit values.

In order to protect authentication details/token(s) and encryption key(s) in transit, communication between the device and the licensing/key authority should be suitably encrypted (i.e. using TLS).

 © 2-IMMERSE Consortium 2017

Content could be explicitly marked as protected using the 2-IMMERSE scheme either by adding a tag to the MPD manifest within a 2-IMMERSE specific namespace, or by generating a unique 2-IMMERSE owned system ID and embedding that within the PSSH of the media.

3.8 Identity Management and Authentication

Latest service documentation: https://2immerse.eu/wiki/id-auth/

3.8.1 Functional Description

This section provides a functional description of the Identity Management and Authentication Service

3.8.1.1 Responsibilities

The user identity service is responsible for managing user identity, privileges and profiles. It allows information about users to be created, shared and aggregated for use in distributed media applications and allows the state of an experience to be captured and stored in a persistent way.

The User Service is comprised of three major functional areas:

1. User Context sharing
2. Profile management
3. Authentication services

Adaptors (or plugins) are used to provide the user service with authentication and profile repository capabilities.

3.8.1.2 Collaborators

- Session Service
- Authentication Service
- Profile Storage Service (e.g. LDAP or custom solution)
- Logging Service

3.8.1.3 Definitions

- User Profile – is a collection of global and application-specific information that applies to the user such as personal credentials, personal preferences and application configuration.
- User Context – is a storage structure that maintains information about a user such as active repository connections, identities and profiles.
- Personal Device – is any device that a user has logged into with their personal credentials and selected a personal profile.
- Communal Device – is any device that multiple users have privileges to configure due to a communal profile having been selected

3.8.1.4 Multiple Users

A profile describes how an experience has been personalised for a given layout context by one or more users. This includes tweaking layouts and choosing active DMApp components or media sources. A profile may also contain other types of state data such as history and usage activity.

Personalising a multi-device, multi-user experience can be a time-consuming process, so it's important to store settings that a user is happy with so they can be applied quickly in future. This is useful for quickly swapping between primary users of the system and is especially useful for demonstration purposes.

Each user in the household may want to configure experiences differently to meet their own needs and preferences. The system should select the appropriate stored configuration depending on who is watching.

3.8.1.5 User Roles

Sometimes multiple people will be watching together, however only one of the users will be responsible for launching the experience. They can be considered the owner of the experience. The 'Experience Owner' is the user whose profile is selected to initially configure an experience and store context state data. A profile selection can be achieved using something similar to the Netflix's "Who's Watching?" screen (as shown below in Figure 11). Changes to the communal configuration made by any of the users in the layout context should be stored to the experience owner's chosen profile.

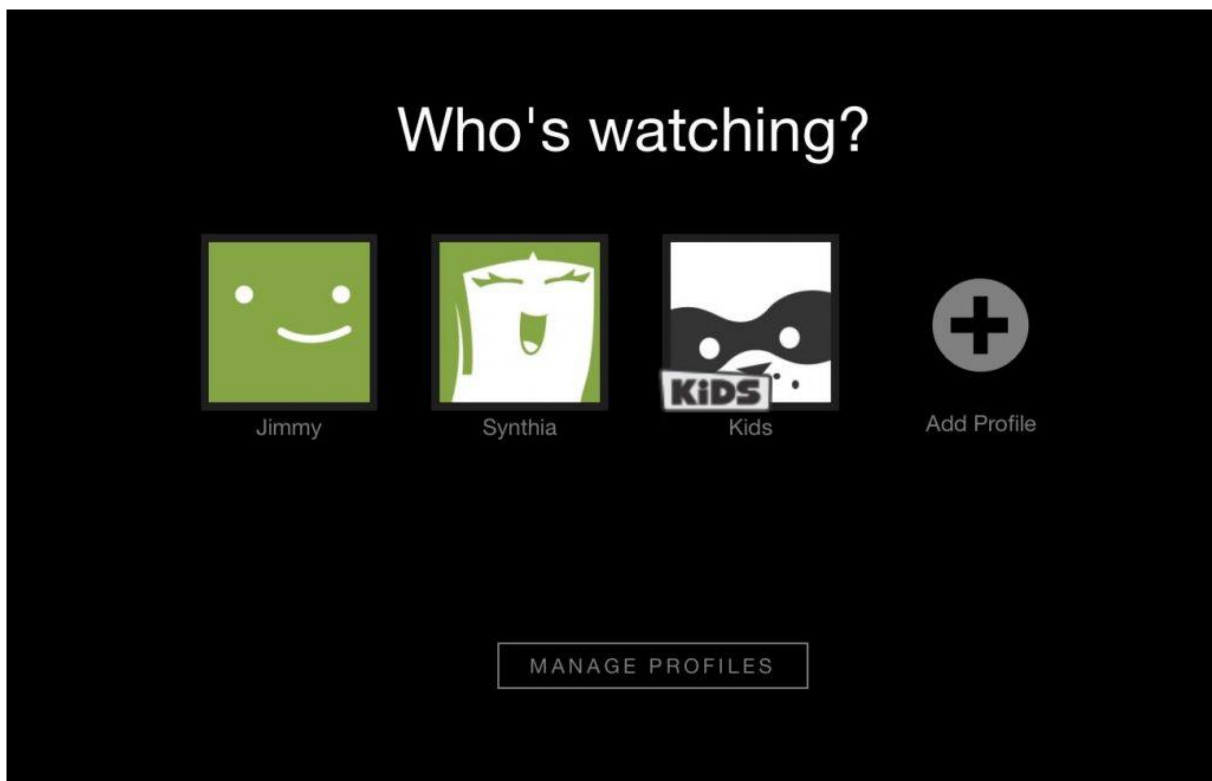

Figure 11 - Netflix "Who's Watching?" Screen

A user may wish to reuse their profile in a different layout context, such as an upstairs bedroom or friend's house. It is therefore important to associate profiles with the users that created them as opposed to any specific layout context or device.

3.8.1.6 Privileges (Communal and Personal Devices)

Any user can make changes to the presentation of an experience on a communal device such as a television and those changes are always stored to the experience owner's profile. The exception being the "Football in the pub" scenario where configuration privileges are heavily

 © 2-IMMERSE Consortium 2017

restricted on communal devices with only the landlord being able to substantially reconfigure the experience.

Privileges are important for personal devices too. For example, a mobile phone is generally considered to be a personal device and other users must seek permission to override the layout and presentation choices of the logged-in user. Configuration changes made to a personal device are stored in the logged-in user's profile as opposed to the experience owner's profile. Permissions are much more important in public contexts, such as the "Theatre in Schools" and "Football in the pub" use cases because trust in these environments is a bigger issue.

3.8.1.7 Profiles

A Profile stores key/value pairs that specify preferences and configuration data for a user. A default profile (authored and distributed by the broadcaster) is used to populate each user profile with sensible values. The user can override the default values at any time with their own personal preferences. Profiles persist from one experience to the next.

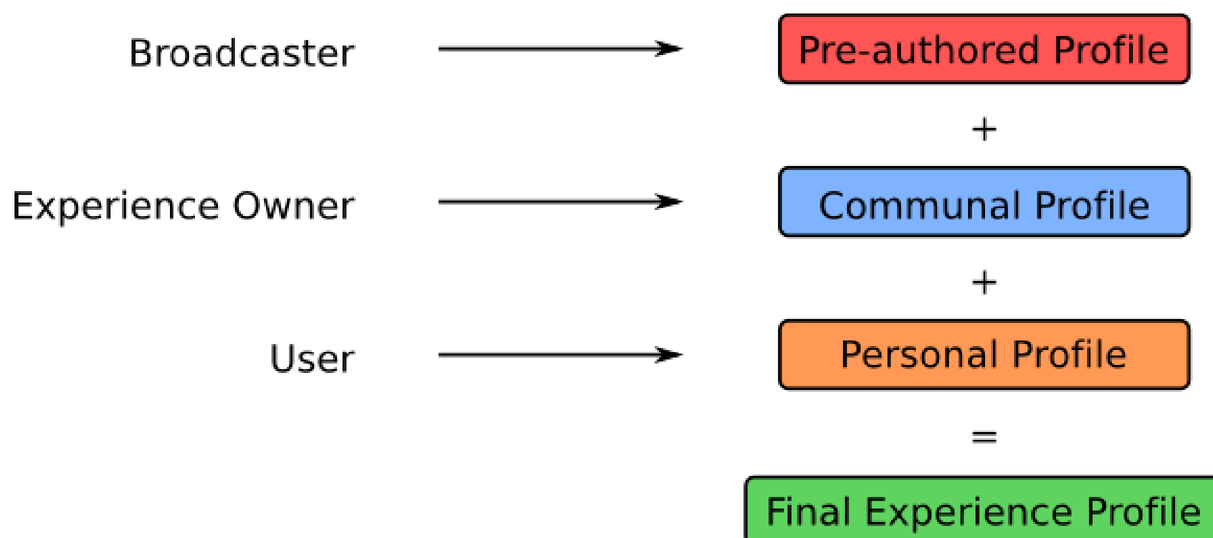

Figure 12 - Data from different profiles are combined to configure an experience

Profiles of each user are combined with the default profile to deliver the final configuration (as shown above in Figure 12). User profiles can store both application-specific and application-independent data.

Application-independent data transcends any one experience. Examples include first/last name, credential sets, accessibility settings, usage metrics or settings common to a group of experiences such as a genre.

3.8.1.8 Identity

User identity is important for personalisation and configuration, but it is also needed by the lobby system for initiating real-time communications and for grouping the right viewers into a shared theatre box. Unique identifiers and human readable names are required per user together with authentication or certification.

3.8.1.9 User Service Verbs

This section lists the verbs used by the Identity Management and Authentication Service

3.8.1.9.1 GetUserContext

Authenticates user and returns the user context
Returns: UserContext
Params: userService, username, password/certificate, domain

3.8.1.9.2 Get/SetCredentialChallengeCallback

Callback for handling credential challenges (OAuth)
User Context Verbs

3.8.1.9.3 LoadProfile

Fetches named profile from server
Returns: Profile
Params: userContext, profileName

3.8.1.9.4 AddProfile

Adds a named profile to the user context
Returns: Profile
Params: userContext, profileName

3.8.1.9.5 SaveProfile

Store profile changes
Returns: status
Params: userContext, profileName

3.8.1.10 Profile Verbs

This section lists the profile verbs used by the Identity Management and Authentocation Service.

3.8.1.10.1 GetValue

Retrieve a named value from a profile
Returns: value or Profile
Params: profile, attributeName/Path

3.8.1.10.2 SetValue

Store a named value to a profile
Params: profile, attributeName/Path, attributeValue

3.8.1.10.3 HasValue

Check for existence of a named value in a profile
Returns: boolean
Params: profile, attributeName/Path

3.8.1.11 Events

This section lists the events relevant for the Identity Management and Authentication Service

3.8.1.11.1 OnProfileChanged

Notification that an external change to a profile has been made.

3.8.2 API Specification

To be specified

 © 2-IMMERSE Consortium 2017

3.9 Session Service

Latest service documentation: https://2immerse.eu/wiki/session/

3.9.1 Functional Description

Definition: A user session is a Single Sign-On (SSO) session as defined by Sun's OpenSSO project and as described by the corresponding OpenSSO Session Service Framework.
(See https://java.net/downloads/opensso/docs/architecture/session_arch.pdf). Note that OpenSSO is a framework that's independent of 3rd Party implementations.

The Session Service provides the functionality to maintain information about an authenticated user's session across all services participating in a Single Sign-On (SSO) environment. The Session Service satisfies a number of critical functions, which enable users to authenticate once, yet access multiple resources such that successive attempts by a user to access protected resources will not require the user to provide authentication credentials for each attempt.
It provides the fundamental administrative and monitoring capabilities for managing account holder sessions. In particular, it generates session identifiers and implements session life cycle events (e.g. session creation, session destruction, etc.), sending state change notifications so that all participants in the same SSO environment are notified.

Each additional device that the user signs in to will be issued with a unique session token and a new session will be created on the server. The session is destroyed when the user has signed out, the session expires or an administrator destroys it.

An SSO session is defined as the interval between the moment the user of an account first signs in to create a session and the moment they log out of the session.

3.9.1.1 Responsibilities

- Create and destroy sessions
- Validate session access tokens
- Retrieve session properties, such as user identity.
- Store and retrieve user or application-defined session properties
- Notify listeners of changes to session state and properties.
- Administration functionality to enumerate and destroy sessions
- Manage session life cycle

3.9.1.2 Collaborators

- User Identity service
- Logging service
- Profile storage service
- Account service

3.9.1.3 Verbs & Events

There are lots of off-the-shelf session management, authentication and user identity frameworks available. The intention is to adopt an existing framework but abstract the API to permit alternatives to be selected at a later date and for functionality to be mocked to simplify development and testing. The following OpenSSO API documentation illustrates the set of verbs required:

https://blogs.oracle.com/docteger/entry/opensso_and_rest

We may choose to implement our own REST end-points that transform and forward the requests onto a chosen 3rd party service providers.

3.10 Lobby Service

Latest service documentation: https://2immerse.eu/wiki/lobby/

3.10.1 Overview

The lobby service provides a means for users to join a virtual group using a pre-agreed lobby name. It is the on-boarding mechanism used to group multiple layout contexts together for synchronised playback and is used as a filter for real-time communication between different groups of users.

3.10.1.1 Definitions

- Client – is an instance of the 2-IMMERSE application. In general, there is one client running per device, but a web browser could be running multiple instance of the application in different tabs.
- Layout context – is a temporary collection of clients on the same LAN that are collaborating to present one multi-device broadcast.
- User – is a person who has signed into a client. A user can be signed into multiple clients simultaneously.
- Host – is a user who launches a programme and therefore owns the layout context.
- Lobby – is a set of layout contexts that are watching a synchronised programme together.

3.10.1.2 Lobby Members

The lobby service displays a table of users grouped by layout context, as shown below in Figure 13.

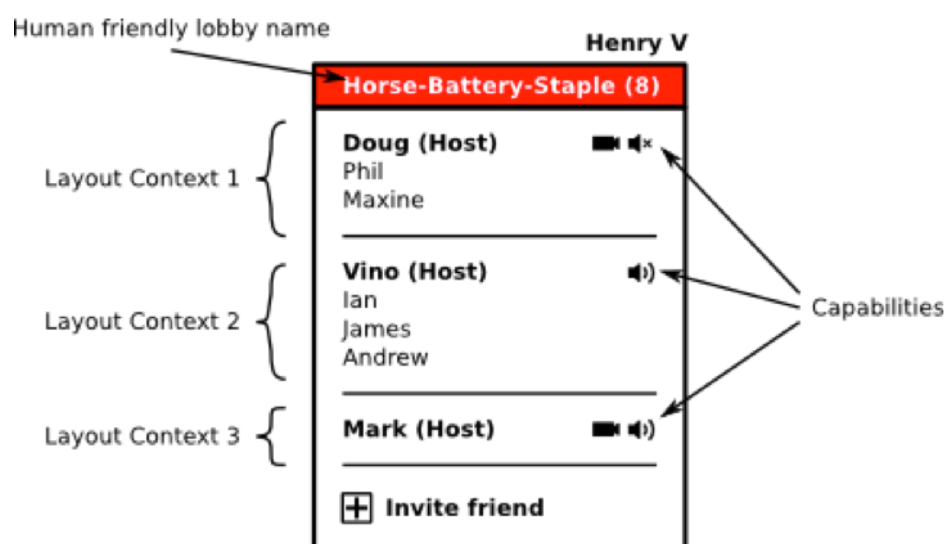

Figure 13 - Example Lobby Service User Table

 © 2-IMMERSE Consortium 2017

The challenge is to ensure the table accurately reflects user presence at all times. A user is considered present if a device they are signed into is in regular contact* with the lobby service.

** Regular contact can be achieved using a heartbeat mechanism over a stateless connection (e.g. long poll) or inferred by monitoring a permanent socket connection. Socket.io is a node.js library that abstracts away the mechanism used.*

Each device in the layout context must therefore be instructed to declare and update its presence with the lobby server and must be capable of displaying the lobby table (DMApp component) if instructed to by the layout context. These requirements demand that a lobby client component be instantiated on each device in the layout context.

A (layoutContextId, userId) pair uniquely identifies a lobby member. This is because a user could be signed into several clients with their credentials and those clients could be used in different layout contexts.

3.10.1.3 Joining A Lobby

Synchronisation between layout contexts is brokered by the lobby service but it requires one user from each layout context to instigate a ‘Join Lobby’ operation. This must pull other members of the layout context into the lobby automatically. There are two possible schemes:

1. The 'Join Lobby' operation broadcasts a message to all other devices in the layout context asking them to connect to a specified lobby.
2. The layout context joins the lobby, submitting and updating user presence information on behalf of members of the layout context.

In the first approach, the lobby service is duplicating some of the presence tracking done by the layout context. In the second approach, where the layout service is acting as a proxy, the layout service is taking on additional responsibilities that prevent a clear separation of concerns. An alternative to both approaches is to separate out user presence management into a separate service, which the lobby and layout services can depend upon. This could also act as a proxy for the call service, keeping it informed of contactable video-chat enabled peers.

3.10.2 Functional Description

The lobby service is multi-tenant, allowing many lobbies to be hosted by a single service instance. Lobby membership information is distributed between lobby server instances as opposed to being persisted to a database. Horizontal scaling of lobby service instances is triggered when memory or network connection limits exceed a pre-defined threshold or latencies increase to a level that delivers a poor user experience.

A lobby doesn’t exist prior to the first user joining and ceases to exist after the last user has left. Users can join and leave the lobby at any time and all members are notified of these events.

Each client establishes a single secure web socket with one of the lobby service instances, either directly or through a presence service. The connection is used to track the client and to notify them of lobby events. It allows the client to subscribe to user leave/join/message notifications from the server and allows the server to detect disconnections.

Clients making a lobby service request directly or indirectly via a presence service must provide a valid session access token in order to use the lobby service API. The token can be sent to the server via a cookie for both REST and web socket communications. The lobby service validates the access token and requests the session’s userId from the session service. Users are internally identified to the lobby service by their unique userId (which is the same as the call service's caller Id).

On joining a lobby, a client is issued with a list of existing lobby members. Clients can track changes to lobby membership by processing leave and join notifications from the server. An administrator can connect to the lobby service for the purpose of moderation without joining a lobby.

The lobby service subscribes to the session events, such as session expiration, session invalidation and user sign-out. It uses these events to automatically evict users from the lobby and drop their connections.

A client wishing to create a new lobby can request a unique lobbyId from the lobby service. The lobbyId is a memorable, human readable string that can be shared by individuals via conventional channels of communication such as over the phone or by embedding it in a URL. It is the key piece of information used to group participants for synchronised media playback between layout contexts. A commonly used scheme is a hyphen-separated list of 3-4 of the most commonly used English language nouns.

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

3.10.2.1 Responsibilities

- Provide a framework for group activities involving a number of users.
- Maintain lobby connections and manage user membership
- Share membership information with clients
- Generate lobby events on the micro service message bus for subscribers to listen to.
- Notify clients of join and leave events
- Generate unique lobbyIds
- Provide administration and chair functions.

3.10.2.2 Voice/Video/Text Chat

Lobbies provide a means of establishing video and voice chat between groups and users by making their callerIds (userIds) available to other members of the lobby. The call service uses these callerIds to broker connections between peers so that they can execute the session initiation protocols required for real-time video chat. It is then the call service's responsibility to resolve which client device to forward the offer to.

Text chat is different to video and voice chat because conversation history must be preserved and unlike video chat, it is unlikely to require a dedicate device. There are two use cases:

1. Communal text chat - everyone in the lobby can see the conversation

 © 2-IMMERSE Consortium 2017

2. One-on-one chat - a user wishes to have a private chat with one of the other lobby members.

It is likely that a separate service will be used to manage the lobby's communal chat history and to broadcast text messages to lobby members. Any client can render a view of chat history and add new messages. This doesn't require an offer/answer signalling mechanism; all clients participate. One-on-one chat may still be handled by a separate chat service, but will use offer/answer semantics via the call service to establish communication.

3.10.2.3 Architecture

This section describes the architecture of the lobby service.

3.10.2.3.1 Overview

A lobby server is responsible for running the core business logic and for maintaining persistent web socket connections with clients or a separate presence server. The number of server instances can be scaled up and down to meet demand. Clients within the same layout context could therefore be connected to different lobby servers and their connections may be lost if servers are taken offline, requiring the client to reconnect.

A load balancer will distribute web socket connections to different server instances to spread the load and then use the publish/subscribe machinery of RabbitMQ, Redis or ZeroMQ to route messages between the server instances. Each server will maintain a list of connected clients grouped by lobbyId and will use routing keys to keep the number of messages between servers to a minimum.

3.10.2.3.2 Consistency & State

Lobby membership and connectivity is shared between lobby server instances without the need for a database. Lobby server instances communicate with each other to keep themselves up-to-date using message queues and routing exchanges. A publisher/subscriber model is used in conjunction with routing keys and RPCs to send lobby events to the right server instances. Changes to lobby membership are published as messages and routed to other server instances. Messages destined for the members of a lobby can be filtered using routing keys. This causes them to be routed to only those server instances responsible for managing connections to other members of the lobby. Servers that manage client connections for a given lobby can subscribe to the message queue using the lobby's routing key. When combined with a limit on lobby occupancy, the number of server-to-server messages can be kept to a minimum, permitting good scalability.

3.10.2.3.3 Alternatives

An alternative scheme is to use a database to manage lobby membership and allow new servers to be provisioned in the event of scaling or failover, but this also requires a garbage collection service to ensure old lobbies and users are removed from the database on client disconnect or server failure. The benefit of using websocket / persistent connections is that garbage collection is implicit and so the record of lobby membership cannot become out of date with respect to the list of server connections.

3.10.2.4 Collaborators

- Logging service
- Session service

- User Identity service
- Call service
- Layout service

3.10.2.5 Verbs

This section describes the verbs used by the Lobby service.

3.10.2.5.1 AllocLobbyId

Generates a unique, human readable lobbyId.
Returns: Unique lobbyId
Params: ssoToken: session access token

3.10.2.5.2 Join

Adds the user specified by the session access token to the specified lobby and broadcasts a 'joined' notification event to each connected client. This method has no effect if the user is already a member of the specified lobby. A list of lobby members is returned in the response. If the lobby is full, the user will not be added to the lobby and an error response is returned.
Returns: members: a list of lobby members
Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby to join

3.10.2.5.3 Leave

Removes the user identified by the session access token from the specified lobby and broadcasts a 'left' notification event to each connected client. This method has no effect if user has already left the lobby.
Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby to leave

3.10.2.5.4 BroadcastMessage

Broadcast an application-defined message to all connected clients. This is intended to allow application specific functionality to be layered on top of the lobby whilst keeping the lobby service as simple as possible.
Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby to broadcast to
message: application defined message payload

3.10.2.5.5 Close

Close a lobby and notify all connected clients by sending a 'disconnect' message. This is an administrative function intended for use by a chair or system administrator.
Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby to leave

3.10.2.5.6 Kick

Evicts a user from a lobby and sends a 'disconnect' message to all clients of the user. Also notifies remaining clients by sending a 'leave' notification. This is an administrative function used for moderation purposes and is intended for system administrators.

 © 2-IMMERSE Consortium 2017

Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby to leave
userId: the user to evict

3.10.2.5.7 GetMembers

Retrieve a list of lobby members.
Params:
ssoToken: session access token
lobbyId: identifier or name of the lobby

3.10.2.6 Events

This section describes Events relavnat for the lobby service

3.10.2.6.1 Disconnect

An administrator or chair has disconnected the client's connection with the lobby.

3.10.2.6.2 Joined

A user has joined the lobby.
Payload: userId indicating the user that has joined the lobby.

3.10.2.6.3 Left

A user has left the lobby.
Payload: userId indicating the user that has left the lobby, either explicitly, as a result of a loss of connection with the lobby service or as a result of being evicted from the lobby by a chair or administrator.

3.10.2.6.4 Message

A user has broadcast a message
Payload: (userId, message) indicating the user that issues the message and the message itself.

3.10.2.6.5 Error

This event is received in response to a failed connection attempt such as an authentication failure or the lobby being full.
Payload: (error code) indicating type of error

3.10.3 API Specification

The API specification for the Lobby Service is documented in Annex E: 2-IMMERSE Lobby Service REST API documentation.

3.11 Call Service (SIP)

Latest service documentation: https://2immerse.eu/wiki/call-server/

3.11.1 Functional Description

The call service allows users to discover and communicate with each other. It acts as an introductory service based on identities as opposed to IP addresses and can broker direct peer-to-peer connections between devices.

3.11.1.1 Caller Id

The advantage of IP addresses / ports is that they uniquely identify a device, however with identities, it isn't clear which device should receive the call. This is because a user may be

signed into multiple devices at the same time. In addition, not every device is capable of real-time video chat. This leads to three options:

1. Signal every device the user is signed into (Google Hangouts approach)
2. The layout context restricts the video chat DMApp component to a single capable device
3. The user nominates a device within the layout context to be used for real-time comms

A final possibility is that the user nominates one device for video/audio comms and allows the video to be rendered on other devices. The option chosen will depend on the quality of the resulting user experience and the capabilities of the devices in the home and is subject to user testing. The call-server must therefore maintain a flexible approach by mapping between a single call identifier and many devices.

3.11.1.1.1 Example

Taking option 3 as an example, the user has signed into a number of client devices and must nominate one of them to be advertised via the call service. This could be achieved when the client device creates a secure web socket connection with the call service, passing a session access token via a cookie. The token would be used to identify the caller via a request to the session service. Connection attempts from other devices or applications will result in an error response indicating that only one device per user may be registered with the call service.

3.11.1.2 Signalling

The call service implements signalling for WebRTC applications via web sockets and XHR (as shown below in Figure 14). Signalling allows initiation of peer-to-peer communication sessions by exchanging control messages that initialise or close communication and report errors. Clients can use the call service's signalling mechanism to exchange ICE candidates obtained separately from STUN/TURN servers. ICE candidates are the public IP addresses and ports that clients should use to communicate with each other and are the result of establishing NAT punch-through or relay. Signalling is also used to negotiate codecs, video resolutions and communication protocols via Session Description Protocol (SDP). Transport type can also be negotiated as reliable (TCP-like) or unreliable (UDP-like).

 © 2-IMMERSE Consortium 2017

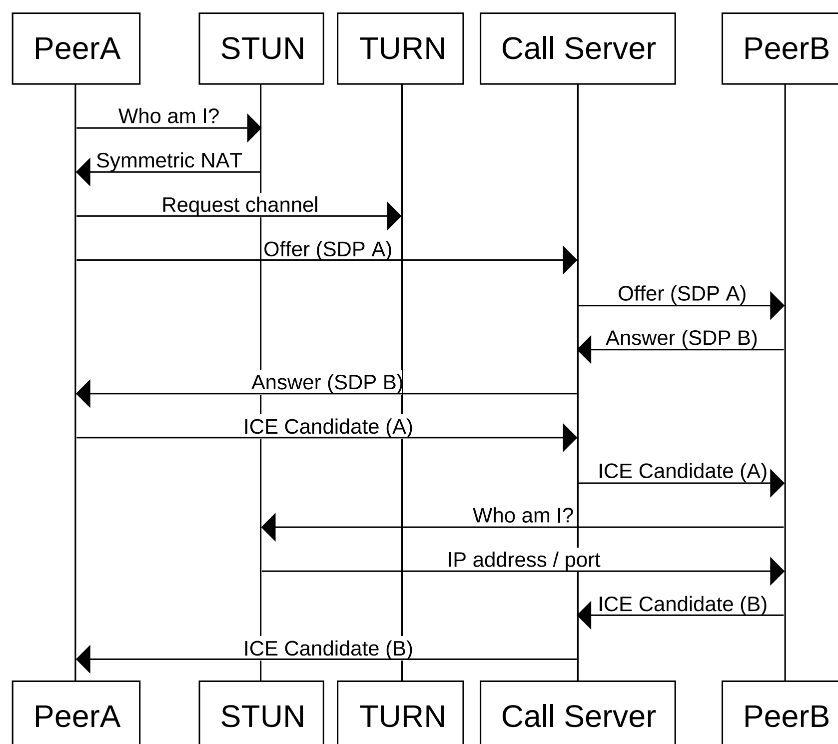

Figure 14 - Call service: WebRTC Signalling Plane

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

3.11.1.3 Responsibilities

- Broker connections between clients
- Permit clients to create, answer and hang-up media calls (audio & video).
- Permit clients to send and receive arbitrary datagrams peer-to-peer
- Notify clients of disconnections
- Utilise 2-IMMERSE user identities and authorize or disallow calls based on successful session access tokens

3.11.1.4 Collaborators

- STUN/TURN service
- Session service
- Logging service

3.11.2 API

PeerJS and peer-server are the nominated client and server technologies for implementing a WebRTC-based call service. The source code is available here:
https://github.com/peers/peerjs
PeerJS is the client library and is documented here: http://peerjs.com/docs/ - api
PeerServer is a call server that brokers connections between PeerJS clients. No peer-to-peer data goes through the server; it only acts as a connection broker. The source code is available here: https://github.com/peers/peerjs-server
Both PeerServer and PeerJS are distributed under the MIT license.

3.11.2.1 Changes Required For 2-Immerse

PeerJS uses randomly assigned user identifiers and tokens generated by the client when connecting a user to PeerServer. 2-IMMERSE has its own userIds and session access tokens that must be used instead.

PeerServer must be modified to perform a request to the session service to validate the authenticity of the token and to lookup the corresponding userId. This userId must replace the randomly generated userId assigned by the PeerJS client.

PeerJS needs to be modified so that it doesn't randomly generate access tokens or userIds, but instead sends the user's session access token in a secure cookie.

3.11.2.2 Configuration

PeerJS/PeerServer must be configured to use secure web sockets and to enable its use behind a reverse proxy. This requires 2-IMMERSE certificates to be generated for TLS/SSL.

3.11.2.3 Verbs

In addition to connection/disconnection, the PeerJS server routes the messages, and their payloads between peers. It supports the following message types:

- **Expire** - message request to peer expired due to inactivity
- **Leave** - used to notify 'hang-up' of the connection
- **Candidate** - used to exchange ICE candidates (i.e. public facing IP addresses obtained from the STUN/TURN server)
- **Offer** - used to create a connection with an SDP (Session Description Protocol) payload
- **Answer** - response to an offer containing an SDP payload

The server will issue a Leave message on behalf of a disconnected peer and will queue outstanding messages destined for peers that it's waiting for to reconnect.

3.11.2.3.1 Offer

When a peer starts a call to another peer, it creates an offer. This description includes all the information about the caller's proposed configuration for the call (the SDP payload). The recipient then responds with an answer, which is a description of their end of the call. In this way, both devices share with one another the information needed in order to exchange media data.
Params:
ssoToken: session access token
peerId: peer to contact
sdp: SDP (Session Description Protocol)

3.11.2.3.2 Answer

Message sent in response to an offer. This includes all the information about the responder's configuration for the call (the SDP payload).
Params:
ssoToken: session access token
peerId: peer to contact
sdp: SDP (Session Description Protocol)

 © 2-IMMERSE Consortium 2017

3.11.2.3.3 Leave

Message sent to tell the recipient that the peer is leaving the call. This message can also be issued by the call server if a peer has disconnected unexpectedly.
Params:
ssoToken: session access token
peerId: peer to contact

3.11.2.3.4 Expire

Message sent by the call server to tell peers that their message has expired due to the recipient being un-contactable or unresponsive.
Params:
ssoToken: session access token
peerId: peer to contact

3.11.2.3.5 Candidate

Message sent between peers to announce their ICE candidates for the purpose of establishing direct peer-to-peer or relayed connections.
Params:
ssoToken: session access token
peerId: peer to contact
iceCandidates: List of ICE candidates

3.12 Logging

Latest service documentation: https://2immerse.eu/wiki/logging/

3.12.1 Functional Description

This section describes afunctional description of the logging service

3.12.1.1 Responsibilities

The Logging Service provides a consistent mechanism for monitoring all aspects of system activity which developers and producers consider to be important.

Activities to be logged should include:

- User interactions with devices in the client environment.
- Interactions between components in the production environment (such as video servers, metadata and graphics feeds).
- Interactions between devices in the client environment to discover and launch DMApps, and to synchronise media objects between devices.
- The request and delivery of media objects and streams.
- The presentation of DMApp components as determined by the UX Engine (Timeline and Layout services).
- The authentication of users, client devices and services via the Session service.
- Communication sessions set up between DMApp components in different locations, mediated by the Lobby.

The goal of the Logging Service is to produce a consistent set of logs for each production session, which we define here as the up-time of the prototype 2-IMMERSE platform during an individual trial event, such as a theatre play, MotoGP race or football match. The service acts as a log aggregator to ingest, store and index log data. It will provide ingested log data to

the Analytics Service, which can be used to present and analyse its data. The service must be started before all other services and should be the last service to be shut down. It may also be started independently of a production session to enable developers and producers to read and analyse log data.

2-IMMERSE intends to evaluate two logging solutions in parallel for the its first trial, Theatre at Home:

1. Logstash and Elasticsearch, both components from the Elastic Stack (formerly ELK – see https://www.elastic.co/products) are proposed as the internal 'platform' logging solution for 2-IMMERSE. Logstash provides a flexible, open source data collection pipeline, while Elasticsearch provides storage, plus indexing and analytics functions. We will preferably use the ELK instance provided within the Mantl platform (http://docs.mantl.io/en/latest/components/elk.html). Log events will arrive from a number of different sources. Logstash offers a wide variety of input plugins, which can also be combined with filters and output plugins, to handle different data sources and aggregate them into a common format within the Elasticsearch database.
2. Google Analytics is a very popular web analytics solution which provides (among others) data collection, consolidation and reporting capabilities for web applications (see https://www.google.co.uk/analytics/standard/features). It is available free of charge and sophisticated event tracking can integrated using Google's analytics.js library. Google Analytics is proposed as a complementary solution for logging of user interactions with 2-IMMERSE DMApp components. These events (button clicks, page scroll/swipe) are potentially more frequent than interactions between the DMApp components and 2-IMMERSE services and their relationship with the user experience makes Google Analytics a more appropriate tool for capturing and processing them.

We anticipate that the following logging scenarios will be implemented:

1. Proprietary 2-IMMERSE services (such as Layout, Timeline, Synchronisation, Lobby) will send individual events to Logstash using the Syslog protocol. Documentation for the syslog input plugin is provided here: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-syslog.html.
2. 2-IMMERSE services based on standard components (such as Service Registry, Session, Call Server) will ideally also send individual events to Logstash using the Syslog protocol. However, if the services already implement a different logging mechanism, an appropriate plugin will be selected to import their output.
3. The Client Web Application will record logs in two different ways:
    1. Platform-related events will be sent directly to Logstash as pre-defined JSON structures over HTTP or HTTPS, to prevent issues with firewall traversal. Documentation for the http input plugin is provided here: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-http.html
    2. Higher resolution events (such as user interactions – button clicks etc.) will be sent to Google Analytics using one or more event trackers and the analytics.js library, in accordance with Google's documentation at https://developers.google.com/analytics/devguides/collection/analyticsjs.

The Logstash http input plugin supports basic HTTP authentication or SSL, with client certificates validated via the Java Keystore format. I would suggest that we use basic

 © 2-IMMERSE Consortium 2017

authentication to avoid complexity, unless there is a need for personal data to be recorded within log messages.

3.12.1.2 Verbs

As the Logstash input plugins present their own protocol-specific interfaces (Syslog or HTTP POST, for example), the use of a verb here is purely illustrative.

3.12.1.2.1 logevent

A platform event from a 2-IMMERSE service which is passed to the Logstash syslog input plugin must conform to the Syslog protocol (RFC 5424). An example message might use the following RFC5424 structure:

```
PRI VERSION space TIMESTAMP space HOSTNAME space APP-NAME space PROCID space MSGID space STRUCTURED-DATA space MSG
```

…and look like the following:

```
<34>1 2016-06-16T18:00:00.000Z layoutservice.2immerse.eu layout_service - 0 - [layout_service] New context created, id=5730
```

Notes:

- APP-NAME defines the source of the log, from a controlled vocabulary.
- PROC_ID is not used, hence '-'
- MSGID is used to define the layout context for this message. If the log message doesn't apply to a layout context (eg. it is from the service registry), special values of context_id can be used (e.g. 0).
- STRUCTURED-DATA is not used, hence '-'
- MSG contains the full log message, which should be consistent to facilitate parsing and analysis. I suggest using square brackets to provide hierarchical information about the origin of the message

A platform event from the Client Application event may be passed to the Logstash http input plugin as follows:

```
{
 "source_name" : "tv_client"
 "source_timestamp" : "2016-06-16 18:00:30 +0000"
 "context_id" : "5730"
 "message" : "[tv_client id=222][dmapc id=123] media_player: playback started"
}
```

Notes:

- source_name defines the source of the log, from a controlled vocabulary.
- source_timestamp is the time at which the event was logged by the reporting component (as opposed to the time the log was received, which will be appended by Logstash).
- context_id is the layout context for this message. If the log message doesn't apply to a layout context (e.g. it is from the service registry), special values of context_id can be used (eg. 0).
- the message format should be consistent to facilitate parsing and analysis. I suggest using square brackets to provide hierarchical information about the origin of the message.

3.12.1.3 Collaborators

As a minimum, the following services should deliver logs to the Logging service:

- Service Registry
- Timeline
- Layout
- Synchronisation
- Identity Management and Authentication
- Session
- Call Server
- Lobby

In addition, the web application on the TV and Companion clients should also deliver logs. In order to correctly preserve the order of logs, all services and web applications should ensure that the clock they use to generate source timestamp is synchronised regularly to an external NTP source. Alternatively, there may be scope to use a shared Wall Clock managed by the Synchronisation service – further investigation is required here.

3.13 Analytics

Some real-time analytics may be provided in later use cases.

Latest service documentation: https://2immerse.eu/wiki/analytics/

3.14 Origin Server / CDN

Latest service documentation: https://2immerse.eu/wiki/origin-server/

3.14.1 Functional Description

The origin server/CDN is responsible for efficiently serving media objects, DMApp Components and any other static resources to client devices. This will be accomplished with at least an origin server operating as the source of truth for all content. The origin server will host a client interface API for requesting content and a publishing interface API for managing content (uploading, delete etc.). The publishing interface will be accessible to authorised users only. The client interface will have a suitable client access level (TBD). The origin server will have suitable resilience and ability to recover after a failure from a backup system or replication procedure.

The CDN can evolve to also include edge servers that cache content requested from the origin server at various geographic locations. The choice of delivery server (origin server or edge servers) will then be based on best performance e.g. the closest to the client requesting the content. The edge servers will host a caching interface API for managing caching policies for authorised users.

3.14.1.1 Client Interface Verbs

This section describes verbs relevant for the Origin Server/ CDN

3.14.1.1.1 getresource

A Client Web Application will call this to request content from the CDN. The content may be delivered from either the origin or edge server depending on DNS geographic lookup.

 © 2-IMMERSE Consortium 2017

3.14.1.2 Publishing Interface Verbs (Origin Server Only)

This section describes verbs relevant for the Publishing interface (the Origin Server) only.

3.14.1.2.1 createresource

An authorised user will call this API on the origin server for adding new content.

3.14.1.2.2 deleteresource

An authorised user will call this API on the origin server for deleting content.

3.14.1.2.3 listresource

An authorised user will call this API on the origin server for listing all content.

3.14.1.2.4 getlogs

An authorised user will call this API on the origin server to request logs for published resources.

3.14.1.3 Caching Interface Verbs (Edge Server Only)

This section describes verbs relevant for the caching interface/ edge server only

3.14.1.3.1 setcachelimit

An authorised user will call this API on the edge server only for setting caching limit for content.

3.14.2 API Specification

The publishing interface to the Origin Server will be SFTP. This allows project partners to use a range of available clients (command line, or GUI such as FileZilla).

The client interface will be a subset of HTTP to support read-only access to content.

3.15 TV Platform

For the watching ‘Theatre at Home’ Service Prototype, all video content will be pre-encoded and delivered on-demand (although it will be presented ‘as-live’). As such, the role of TV Platform is fulfilled by the Origin Server/CDN.

© 2-IMMERSE Consortium 2017

4 Conclusion

In this document we have presented the 2-IMMERSE platform interface specifications, providing a more detailed definition of the platform components and services defined in deliverable D2.1

As noted in the executive summary, our focus on functionality for the platform and its components is driven by the initial service prototype (Watching Theatre at Home), and as such, the interface definitions are likely to evolve both as we implement the platform components, and address the expanding scope of the four multi-screen service prototypes through the project. We are currently identifying common functionality required by more than one service, and will likely partition such functions into their own micro-services that could be shared by these services. This specification document therefore offers a ‘snapshot’ of the interface specifications at a moment in time. The high-level component definitions have been made available on the project website, and links to these have been included in this document to allow readers to access up-to-date versions of this information.

At the time of writing we have basic implementations of the following services:

- Timeline service
- Layout service
- Lobby Service
- Origin Server

With Service Discovery and Logging capabilities being provided by the underlying Mantl infrastructure that we have chosen to adopt.

Work is progressing on integrating these services through an iterative development approach, as we progress towards the initial service prototype. Work is also progressing on the client application, and the internal architecture and interfaces of the client device software stack is under development.

The next Work Package 2 deliverable is D2.3: Distributed Media Application Platform: Description of First Release, and we anticipate that where our architecture and platform component interfaces evolve significantly from those described in D2.1System Architecture, and this deliverable, this will be reflected in this D2.3.

 © 2-IMMERSE Consortium 2017

Annex A DIAL Plug-In API Specifications

A.1 getDialClient()

If the DIAL Plug-in has been added to the CSA (companion screen application), the method getDialClient() is available on the global scope of the CSA. Use getDialClient() to retrieve a DialClient object.

A.2 DialClient

Objects of type DialClient allow discovering DIAL-ready HbbTV devices on the local network.

A.2.1 Methods

A.2.1.1 startDiscovery()

Starts the search for DIAL-ready HbbTV devices on the local network.

A.2.1.1.1 Parameters:

Name	Type	Description
onDeviceListChanged	function	Callback function that is called if the set of discovered devices changes. The function is passed an array of objects of type Device.

A.2.1.2 stopDiscovery()

Stops the search for devices on the local network.

A.2.1.3 showPicker()

Launches a GUI displaying a list of discovered devices. The user is asked to select a device from that list.

A.2.1.3.1 Parameters:

Name	Type	Description
onPicked	function	Callback function that is passed the selected device.

A.3 Device

Object representing a DIAL-ready HbbTV terminal. The object provides all necessary information to establish an App-to-App-Communication channel and to synchronise its media presentation to the presentation on the HbbTV terminal by means of the DVB CSS protocol.

A.3.1 Properties

Name	Type	Description
friendlyName	string	Human readable name of the HbbTV terminal
app2AppURL	string	CSA endpoint for the App-to-App-Communication channel
interDevSyncURL	string	Inter-Device Synchronization endpoint
userAgent	string	User agent string of the HbbTV terminal browser
dialUrl	string	URL of the HbbTV DIAL app. Send a POST request with an XML AIT to this URL to launch an application on the TV.

A.3.2 Methods

A.3.2.1 on()

Sets an event listener.

A.3.2.1.1 Parameters

Name	Type	Description
eventName	string	Name of the event (see Events).
callback	function	Callback function that is called when the event occurs.

A.3.2.1.2 Events

Name	Description
statechange	Is called if the state of the device changes. The callback function is passed the DeviceStatus.

A.4 DeviceStatus

A.4.1 Properties

Name	Type	Description
TODO	TODO	TODO

A.5 Usage

Below Figure 15 illustrates the sequence of interaction between the CSA and the DIAL plug-in. Following steps are depicted:

1. The CSA calls the method getDialClient() to retrieve a DialClient object.
2. The DIAL plug-in passes the DialClient object to the CSA.
3. The CSA calls startDiscovery() on the DialClient object to start the discovery of HbbTV terminals. As a parameter of this call, it passes a callback function, which is invoked by the DialClient object as soon as a device has been discovery.
4. The DialClient calls the callback function and passes an array of Device objects to the CSA.
5. The CSA requests the DialClient object to present a dialog to the user, which presents a list of discovered device to the users and requests them to select one. As a parameter of this call, it passes a callback function, which is invoked by the DialClient object as soon as a device has been selected by the users.
6. The DialClient invokes the callback function and passes the Device object that corresponds to device chosen by the user.

Alternatively, to steps “5.” and “6.” the CSA can present a dialog itself. This way it can ensure a coherent experience. Once the CSA has retrieved a Device object, it can use the given information to launch an application on the corresponding HbbTV terminal and to establish an App-to-App-Communication channel for exchange of textual information (e.g. JSON formatted control commands) between the CSA and the application running on the HbbTV terminal. The Device object comprises also all relevant information to establish a session for media-presentation synchronisation between CSA and HbbTV terminal.

 © 2-IMMERSE Consortium 2017

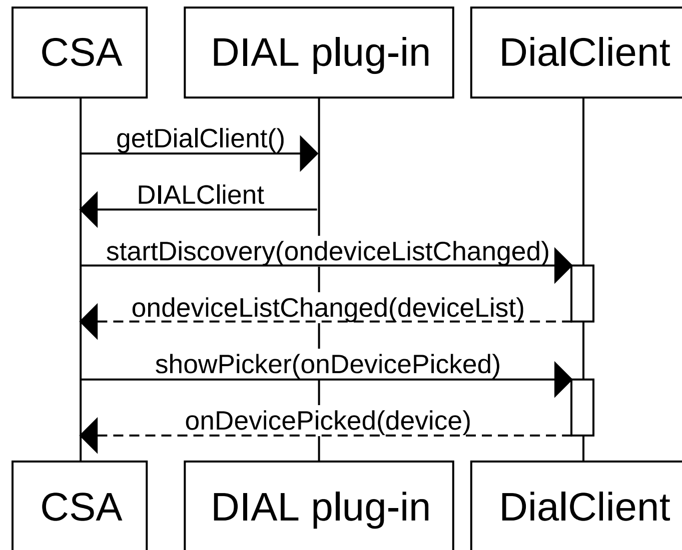

Figure 15 - Sequence diagram illustrating how to use the DIAL plug-in to discover HbbTV terminals

Annex B 2-IMMERSE Timeline Service API documentation version v1

http://2immerse.eu/timeline/v1

NB This documentation has been auto-generated from the RAML API Specification document

---

B.1 /context

B.1.1 /context

B.1.1.1 post

Create a timeline for the given context

B.1.1.1.1 Query Parameters

- contextId: string
- layoutServiceUrl: string

B.1.1.1.2 Response: 204

B.1.1.2 get

Get list of initialised contextIDs

B.1.1.2.1 Response: 200

- Body Type: application/json
- Body Example:
  ["1234"]

B.1.2 /context/{contextId}/dump

B.1.2.1 get

Dump some debugging info about this context

B.1.2.1.1 URI Parameters

- contextId: string

B.1.2.1.2 Response: 200

- Body Type: text/plain

B.1.3 /context/{contextId}/loadDMAppTimeline

B.1.3.1 put

Load a document into the timeline for a given DMApp

B.1.3.1.1 URI Parameters

- contextId: string

B.1.3.1.2 Query Parameters

- timelineDocUrl: string *(required)*

 © 2-IMMERSE Consortium 2017

- dmappId: string *(required)*

B.1.3.1.3 Response: 204

B.1.4 /context/{contextId}/unloadDMAppTimeline

B.1.4.1 put

B.1.4.1.1 URI Parameters

- contextId: string

B.1.4.1.2 Query Parameters

- dmappId: string *(required)*

B.1.4.1.3 Response: 204

B.1.5 /context/{contextId}/dmappcStatus

B.1.5.1 put

B.1.5.1.1 URI Parameters

- contextId: string

B.1.5.1.2 Query Parameters

- dmappId: string *(required)*
- componentId: string *(required)*
- status: string

B.1.5.1.3 Response: 204

B.1.6 /context/{contextId}/timelineEvent

B.1.6.1.1 put

B.1.6.1.2 URI Parameters

- contextId: string

B.1.6.1.3 Query Parameters

- eventId: string *(required)*

B.1.6.1.4 Response: 204

B.1.7 /context/{contextId}/clockChanged

B.1.7.1.1 put

Informs the timeline server of the current mapping of wallclock to presentation clock

B.1.7.1.2 URI Parameters

- contextId: string

B.1.7.1.3 Query Body

- Type: application/json
- Example:

```
{   "wallClock" : 1467290807.506383,   "contextClock" : 3.456000 }
```

© 2-IMMERSE Consortium 2017

B.1.7.1.4 Response: 204

 © 2-IMMERSE Consortium 2017

Annex C Timeline Document Format Design Considerations

The document format is inspired by SMIL, but cut down to an absolute minimum, and fixing various issues we think exist with SMIL.

C.1.1 Requirements

What we want to keep:

- hierarchical containment
- automatic inference of timing
- parallel and sequential composition
- conditional composition

What we want to get rid of, and why:

- layout. This should be left to CSS or the host language or whatever.
- human-friendly format. As most documents will be generated by software there is no real reason for convenience features for humans that muddle the semantics.
- XML dependency. Probably XML is going to be the standard external representation, but we want to be able to also represent documents in JSON or in a palatable in-core object format. This should help palatability to the JavaScript community.

What we want to gain:

- Serializability of current state. After a document has been playing back for some time it we able to save it so that when this new document starts playing it starts exactly where it was when it was saved.
- Timegraph equivalence. The document, when running, must be its own timegraph. This may be the same requirement as the previous one, seen from a different angle.
- Editability. It must be possible to modify the document (and hence the timegraph) while the document is running. We think it is enough to specify the semantics the following operations:
    - delete element (and its complete subtree)
    - insert child element
    - insert parent element
- Timing and synchronization semantics that can be explained in about one page of text.

C.1.2 Design

We use XML-centric language here, but wherever we say "element" you could also read "object" (and where we say "attribute" you could read "instance variable"). We will also use the "tl:" XML namespace, just to make clear that the element may have lots of other attributes but these are irrelevant to the timing semantics of the document.

Every element should do one thing, and one thing only. The only attributes the element has are those that are vital to the semantics of the element. This should lead to an enormous

simplification of semantics (when compared to SMIL) because there is no more need to explain interplay between attributes (such as for SMIL end/dur/repeatDur/repeatCount/fill). It should also help serializability.

Every element has a virtual clock. This clock may be independent (either from a media clock derived from whether audio or video it is playing, or from a wall clock) or it can be slaved to the clock of its parent or one of its children.

If the virtual clock of a media element is slaved to the clock of its parent then the media clock should follow the virtual clock, so media playback may need to speed up or slow down (or skip or pause) to resynchronise. We will probably need a couple of attributes eventually to state how this should be done, but the coupling of the virtual clock to the media clock is a purely local effect and does not affect the timegraph.

Timing and synchronisation relationships only exist between parents and children. There is no synchronisation between sibling elements, formally, this all goes via the parent. But, practically speaking, normally a parent will pick up its clock from one of its children, and slave the clocks of its other children. When compared to SMIL we do lose the ability to specify out-of-tree synchronisation requirements, such as for SMIL syncBase.

Virtual clocks have a priority, and the idea is that a <tl:par> element picks up the clock from its highest priority child or from its parent. It then uses this clock to drive the other children (or its parent). This relationship is dynamic, so as children start and stop different clocks can become the master.

These simplified clock and synchronisation designs should enable editability, and again help with serializability and simplified semantics.

C.1.3 Format

<tl:ref tl:prio="100" tl:fill="freeze|remove">
Media element. Whether this is actually called tl:ref or something else (such as a tl:video, tl:audio, etc) remains to be seen. It could even be that there is no element as such but an attribute on an element in the host language (for example tl:timeAction="none|visibility|display|..." as per HTML+SMIL). The tl:prio is used by a tl:par parent (or actually closest tl:par ancestor) to determine clock priorities and select the master clock. tl:fill is used by the tl:par parent or ancestor to determine what to do when a non-master element ends while the master element clock is still running: either pause it or remove it.

<tl:par tl:end="first|all|master" tl:sync="true|false" tl:prio=... tl:fill=...>
Parallel composition. All children run in parallel. The end of the tl:par depends on the tl:end attribute: either when the first of its children has ended, the last of its children has ended or its timeline master child has ended. We may also want to specify a specific child (by xmlid) to determine when the tl:par ends.
tl:sync determines whether the children (except the master) are synchronised to the tl:par, or whether they are free-running. The latter essentially creates a completely independent timeline.

 © 2-IMMERSE Consortium 2017

It may be better to move the functionality of the tl:sync attribute to the child node, so it is possible to easily specify that most children of a par are synchronised but some are running on an independent timeline. Then we would get something like a tl:independent="true|false" attribute. But it would only be allowed inside a tl:par parent...

<tl:seq>
Sequential composition. The children run one after the other. We think tl:seq should not have the tl:prio and tl:fill attributes (to be confirmed).

<tl:sleep tl:dur="10s" />
Do nothing. Stop running after the given tl:dur.

<tl:wait tl:event="...." />
Do nothing. Stop running when the event happens. How this event is specified (and whether we need the tl:event parameter in the first place) remains to be seen, and probably depends on the host language.

<tl:conditional tl:expr="...">
Conditionally run the single child, based on whether tl:expr evaluates to true or not. This may be better specified as an attribute (so we don't have this *single child* requirement). Language for the expression is to be determined.

<tl:excl> and <tl:switch>
We may want some form of exclusives but this remains to be determined.

<tl:repeat ...>
We probably want some way to repeat things.

C.1.4 Examples

Various constructs become quite a bit more convoluted in this language than they are in SMIL. First and foremost, not having begin, dur and end attributes means that <video begin="5s" dur="10s" .../> will have to be encoded as

```
<tl:par tl:end="first">
  <tl:seq>
    <tl:sleep tl:dur="5s"/>
    <tl:video .../>
  </tl:seq>
  <tl:sleep tl:dur="10s"/>
</tl:par>
```

This is a nuisance from a coding point of view, but it has a great advantage for the semantics. For example, it is absolutely clear that the "5s" is not with respect to the video clock, and the 10s may be (depending on clock priorities).

C.1.5 API

There is going to be some sort of an API between the objects (at runtime), specifically between parents and children. From parent to child, there will be some sequence of init()/start()/stop()/pause()/resume(), and some calls to modify clock relationships slaveClockToMe()/becomeMasterClock(). There will be some callbacks from child to parent

inited()/started()/stopped()/paused()/resumed() and some callbacks between clock master and clock slave clockChanged().

 © 2-IMMERSE Consortium 2017

Annex D 2-IMMERSE Layout Service API documentation version v1

http://2immerse.eu/layout/v1

NB This documentation has been auto-generated from the RAML API Specification document

---

D.1 /context

D.1.1 /context

D.1.1.1 get

get context (returns a context if deviceId is a member of a context)

D.1.1.1.1 Query Parameters

- deviceId: string *(required)*
- reqDeviceId: string *(required)*

D.1.1.1.2 Response: 200

- Body Type: application/json
- Body Schema:
{ "title": "Context", "type": "object", "properties":{ "contextId": { "type": "string" }, "deviceIds": { "type": "array", "items": { "type": "string" } } }, "required": ["contextId", "deviceIds"] }
- Body Example:
{ "contextId": "1234", "deviceIds": ["5678"] }

D.1.1.1.3 Response: 404

D.1.1.2 post

create context and join deviceId using supplied Caps (Capabilities)

D.1.1.2.1 Query Parameters

- orientation: string *(required)*
- deviceId: string *(required)*
- reqDeviceId: string *(required)*

D.1.1.2.2 Query Body

- Type: application/json
- Schema:
{ "title": "CapabilityParams", "type": "object", "properties":{ "displayWidth": { "type": "integer" }, "displayHeight": { "type": "integer" }, "audioChannels": { "type": "integer" }, "concurrentVideo": { "type": "integer" }, "touchInteraction": { "type": "boolean" }, "sharedDevice": { "type": "boolean" }, "orientations": { "type": "array", "items": { "type": "string" } } }, "required": ["displayWidth", "displayHeight","audioChannels","concurrentVideo","sharedDevice", "orientations"] }

D.1.1.2.3 Response: 201

- Body Type: application/json

- Body Schema:
  { "title": "Context", "type": "object", "properties":{ "contextId": { "type": "string" }, "deviceIds": { "type": "array", "items": { "type": "string" } } }, "required": ["contextId", "deviceIds"] }
- Body Example:
  { "contextId": "1234", "deviceIds": ["5678"] }

D.1.1.2.4 Response: 204

D.1.2 /context/{contextId}

D.1.2.1 get

get context information (array of deviceIds)

D.1.2.1.1 URI Parameters

- contextId: string

D.1.2.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.2.1.3 Response: 200

- Body Type: application/json
- Body Schema:
  { "title": "Context", "type": "object", "properties":{ "contextId": { "type": "string" }, "deviceIds": { "type": "array", "items": { "type": "string" } } }, "required": ["contextId", "deviceIds"] }
- Body Example:
  { "contextId": "1234", "deviceIds": ["5678"] }

D.1.2.1.4 Response: 404

D.1.2.2 delete

destroy context

D.1.2.2.1 URI Parameters

- contextId: string

D.1.2.2.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.2.2.3 Response: 204

D.1.2.2.4 Response: 404

D.1.3 /context/{contextId}/devices

D.1.3.1 post

join deviceId to contextId using supplied Caps (Capabilities)

D.1.3.1.1 URI Parameters

- contextId: string

 © 2-IMMERSE Consortium 2017

D.1.3.1.2 Query Parameters

- orientation: string *(required)*
- deviceId: string *(required)*
- reqDeviceId: string *(required)*

D.1.3.1.3 Query Body

- Type: application/json
- Schema:
{ "title": "CapabilityParams", "type": "object", "properties":{ "displayWidth": { "type": "integer" }, "displayHeight": { "type": "integer" }, "audioChannels": { "type": "integer" }, "concurrentVideo": { "type": "integer" }, "touchInteraction": { "type": "boolean" }, "sharedDevice": { "type": "boolean" }, "orientations": { "type": "array", "items": { "type": "string" } } }, "required": ["displayWidth", "displayHeight","audioChannels","concurrentVideo","sharedDevice", "orientations"] }

D.1.3.1.4 Response: 201

- Body Type: application/json
- Body Schema:
{ "title": "Context", "type": "object", "properties":{ "contextId": { "type": "string" }, "deviceIds": { "type": "array", "items": { "type": "string" } } }, "required": ["contextId", "deviceIds"] }
- Body Example:
{ "contextId": "1234", "deviceIds": ["5678", "6789"] }

D.1.3.1.5 Response: 404

D.1.4 /context/{contextId}/devices/{deviceId}

D.1.4.1 delete

leave context (removes deviceId from contextId)

D.1.4.1.1 URI Parameters

- contextId: string
- deviceId: string

D.1.4.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.4.1.3 Response: 200

- Body Type: application/json
- Body Schema:
{ "title": "Context", "type": "object", "properties":{ "contextId": { "type": "string" }, "deviceIds": { "type": "array", "items": { "type": "string" } } }, "required": ["contextId", "deviceIds"] }
- Body Example:
{ "contextId": "1234", "deviceIds": ["5678"] }

D.1.4.1.4 Response: 204

D.1.4.1.5 Response: 404

D.1.5 /context/{contextId}/devices/{deviceId}/orientation

D.1.5.1 put

change device orientation

D.1.5.1.1 URI Parameters

- contextId: string
- deviceId: string

D.1.5.1.2 Query Parameters

- orientation: string *(required)*
- reqDeviceId: string *(required)*

D.1.5.1.3 Response: 204

D.1.5.1.4 Response: 400

D.1.5.1.5 Response: 404

D.1.6 /context/{contextId}/dmapp

D.1.6.1 get

get running dmappId's

D.1.6.1.1 URI Parameters

- contextId: string

D.1.6.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.6.1.3 Response: 200

- Body Type: application/json
- Body Schema:
  { "title": "DMAppIdList", "type": "array", "items": { "type": "string" } }
- Body Example:
  ["testDMApp1", "testDMApp2"]

D.1.6.2 post

load DMApp

D.1.6.2.1 URI Parameters

- contextId: string

D.1.6.2.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.6.2.3 Query Body

- Type: application/json
- Schema:

© 2-IMMERSE Consortium 2017

{ "title": "DMAppSpec", "type": "object", "properties":{ "timelineDocUrl": { "type": "string" }, "layoutReqsUrl": { "type": "string" }, "timelineServiceUrl": { "type": "string" } }, "required": ["timelineDocUrl", "layoutReqsUrl"] }

- Example:

{ "timelineUrl": "http://2immerse.eu/apps/testDMApp1/timeline.json", "layoutReqsUrl": "http://2immerse.eu/apps/testDMApp1/layout.json" }

D.1.6.2.4 Response: 201

- Body Type: application/json
- Body Schema:

{ "title": "DMApp", "type": "object", "properties":{ "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "spec": { "type": "DMAppSpec" }, "components": { "type": "array", "items": { "type": "DMAppComponent" } } }, "required": ["DMAppId", "contextId", "spec", "components"] }

- Body Example:

{ "DMAppId": "testDMApp1", "contextId": "1234", "spec": { "timelineUrl": "http://2immerse.eu/apps/testDMApp1/timeline.json", "layoutReqsUrl": "http://2immerse.eu/apps/testDMApp1/layout.json" }, "components": [] }

D.1.6.2.5 Response: 404

D.1.7 /context/{contextId}/dmapp/{dmappId}

D.1.7.1 get

get DMApp info (includes a list of current components for the requesting device)

D.1.7.1.1 URI Parameters

- contextId: string
- dmappId: string

D.1.7.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.7.1.3 Response: 200

- Body Type: application/json
- Body Schema:

{ "title": "DMApp", "type": "object", "properties":{ "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "spec": { "type": "DMAppSpec" }, "components": { "type": "array", "items": { "type": "DMAppComponent" } } }, "required": ["DMAppId", "contextId", "spec", "components"] }

- Body Example:

{ "DMAppId": "testDMApp1", "contextId": "1234", "spec": { "timelineUrl": "http://2immerse.eu/apps/testDMApp1/timeline.json", "layoutReqsUrl": "http://2immerse.eu/apps/testDMApp1/layout.json" }, "components": [] }

D.1.7.1.4 Response: 404

D.1.7.2 delete

unload DMApp

D.1.7.2.1 URI Parameters

- contextId: string

- dmappId: string

D.1.7.2.2 Query Parameters
- reqDeviceId: string *(required)*

D.1.7.2.3 Response: 204

D.1.7.2.4 Response: 404

D.1.8 /context/{contextId}/dmapp/{dmappId}/actions/clockChanged

D.1.8.1 post
Informs the timeline server of the current mapping of wallclock to presentation clock

D.1.8.1.1 URI Parameters
- contextId: string
- dmappId: string

D.1.8.1.2 Query Body
- Type: application/json
- Schema:
{ "title": "Clock", "type": "object", "properties":{ "wallClock" : { "type": "number" }, "contextClock" : { "type": "number" } }, "required": ["wallClock", "contextClock"] }
- Example:
{ "wallClock" : 1467290807.506383, "contextClock" : 3.456000 }

D.1.8.1.3 Response: 204

D.1.9 /context/{contextId}/dmapp/{dmappId}/component/{componentId}

D.1.9.1 get
get DMApp component info

D.1.9.1.1 URI Parameters
- contextId: string
- dmappId: string
- componentId: string

D.1.9.1.2 Query Parameters
- reqDeviceId: string *(required)*

D.1.9.1.3 Response: 200
- Body Type: application/json
- Body Schema:
{ "title": "DMAppComponent", "type": "object", "properties":{ "componentId": { "type": "string" }, "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "config": { "type": "DMAppComponentConfig" }, "startTime": { "type": "string" }, "stopTime": { "type": "string" }, "layout": { "type": "Layout" } }, "required": ["componentId", "DMAppId", "contextId"] }
- Body Example:
{ "componentId": "9876", "DMAppId": "testDMApp1", "contextId": "1234" }

 © 2-IMMERSE Consortium 2017

D.1.9.1.4 Response: 404

D.1.10 .../dmapp/{dmappId}/component/{componentId}/actions/init

D.1.10.1 post

initialise DMApp component - this is a Timeline service API (not a client device API)

D.1.10.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.10.1.2 Query Body

- Type: application/json
- Schema:
  { "title": "DMAppComponentConfig", "type": "object", "properties":{ "url":{ "type": "string" }, "class":{ "type": "string" } }, "required": ["url","class"] }

D.1.10.1.3 Response: 201

D.1.10.1.4 Response: 204

D.1.11 .../dmapp/{dmappId}/component/{componentId}/actions/start

D.1.11.1 post

start DMApp component - this is a Timeline service API (not a client device API)

D.1.11.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.11.1.2 Query Parameters

- startTime: string *(required)*

D.1.11.1.3 Response: 201

D.1.11.1.4 Response: 204

D.1.12 .../dmapp/{dmappId}/component/{componentId}/actions/stop

D.1.12.1 post

stop DMApp component - this is a Timeline service API (not a client device API)

D.1.12.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.12.1.2 Query Parameters

- stopTime: string *(required)*

D.1.12.1.3 Response: 201

D.1.12.1.4 Response: 204

D.1.13 .../dmapp/{dmappId}/component/{componentId}/actions/hide

D.1.13.1 post

hide DMApp component

D.1.13.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.13.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.13.1.3 Response: 201

- Body Type: application/json
- Body Schema:
{ "title": "DMAppComponent", "type": "object", "properties":{ "componentId": { "type": "string" }, "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "config": { "type": "DMAppComponentConfig" }, "startTime": { "type": "string" }, "stopTime": { "type": "string" }, "layout": { "type": "Layout" } }, "required": ["componentId", "DMAppId", "contextId"] }
- Body Example:
{ "componentId": "9876", "DMAppId": "testDMApp1", "contextId": "1234" }

D.1.13.1.4 Response: 204

D.1.14 .../dmapp/{dmappId}/component/{componentId}/actions/show

D.1.14.1 post

show DMApp component

D.1.14.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.14.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.14.1.3 Response: 201

- Body Type: application/json
- Body Schema:
{ "title": "DMAppComponent", "type": "object", "properties":{ "componentId": { "type": "string" }, "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "config": { "type": "DMAppComponentConfig" }, "startTime": { "type": "string" }, "stopTime": { "type": "string" }, "layout": { "type": "Layout" } }, "required": ["componentId", "DMAppId", "contextId"] }
- Body Example:

© 2-IMMERSE Consortium 2017

{ "componentId": "9876", "DMAppId": "testDMApp1", "contextId": "1234" }

D.1.14.1.4 Response: 204

D.1.15 .../dmapp/{dmappId}/component/{componentId}/actions/move

D.1.15.1 post

move DMApp component

D.1.15.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.15.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.15.1.3 Response: 201

- Body Type: application/json
- Body Schema:
{ "title": "DMAppComponent", "type": "object", "properties":{ "componentId": { "type": "string" }, "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "config": { "type": "DMAppComponentConfig" }, "startTime": { "type": "string" }, "stopTime": { "type": "string" }, "layout": { "type": "Layout" } }, "required": ["componentId", "DMAppId", "contextId"] }
- Body Example:
{ "componentId": "9876", "DMAppId": "testDMApp1", "contextId": "1234" }

D.1.15.1.4 Response: 204

D.1.16 .../dmapp/{dmappId}/component/{componentId}/actions/clone

D.1.16.1 post

clone DMApp component

D.1.16.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.16.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.16.1.3 Response: 201

- Body Type: application/json
- Body Schema:
{ "title": "DMAppComponent", "type": "object", "properties":{ "componentId": { "type": "string" }, "DMAppId": { "type": "string" }, "contextId": { "type": "string" }, "config": { "type": "DMAppComponentConfig" }, "startTime": { "type": "string" }, "stopTime": { "type": "string" }, "layout": { "type": "Layout" } }, "required": ["componentId", "DMAppId", "contextId"] }
- Body Example:

{ "componentId": "9876", "DMAppId": "testDMApp1", "contextId": "1234" }

D.1.16.1.4 Response: 204

D.1.17 .../dmapp/{dmappId}/component/{componentId}/actions/status

D.1.17.1 post

update DMApp component status

D.1.17.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.17.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.17.1.3 Query Body

- Type: application/json
- Schema:
{ "title": "DMAppComponentStatus", "type": "object", "properties":{ "status":{ "type": "string" } }, "required": ["status"] }

D.1.17.1.4 Response: 204

D.1.17.1.5 Response: 404

D.1.18 .../dmapp/{dmappId}/component/{componentId}/actions/saveState

D.1.18.1 post

save DMApp component state

D.1.18.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.18.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.18.1.3 Query Body

- Type: application/json

D.1.18.1.4 Response: 201

D.1.18.1.5 Response: 404

D.1.19 .../dmapp/{dmappId}/component/{componentId}/actions/restoreState

D.1.19.1 get

restore DMApp component state

 © 2-IMMERSE Consortium 2017

D.1.19.1.1 URI Parameters

- contextId: string
- dmappId: string
- componentId: string

D.1.19.1.2 Query Parameters

- reqDeviceId: string *(required)*

D.1.19.1.3 Response: 200

- Body Type: application/json

D.1.19.1.4 Response: 404

Annex E 2-IMMERSE Lobby Service REST API documentation

https://lobby.2immerse.eu/

NB This documentation has been auto-generated from the RAML API Specification document

---

E.1 Lobbies

E.1.1 /lobbies

E.1.1.1 post

Generates a unique, human readable lobbyId.

E.1.1.1.1 Response: 201

- Body Type: application/json
- Body Schema:
  { "title": "LobbyId", "type": "string" }
- Body Example:
  { "lobbyId" : "alpha-beta-gamma" }

E.1.1.1.2 Response: 401

E.1.1.2 get

Get list of active lobbies

E.1.1.2.1 Response: 200

- Body Type: application/json
- Body Schema:
  { "title": "Lobbies", "description": "Array of lobbyIds", "type": "array", "items": { "type": "string", "pattern": "^[a-z]+-[a-z]+-[a-z]+$" } }

E.1.1.2.2 Response: 401

E.1.2 /lobbies/{lobbyId}

E.1.2.1 get

Retrieve a list of lobby members.

E.1.2.1.1 URI Parameters

- LobbyId: string

E.1.2.1.2 Response: 200

- Body Type: application/json
- Body Schema:
  { "title": "Members", "type": "array", "description": "All lobby members as a list of userIds", "items": { "type": "string" } }

 © 2-IMMERSE Consortium 2017

E.1.2.1.3 Response: 401

E.1.2.2 delete

Close a lobby and notify all connected clients by sending a disconnect message. This is an administrative function intended for use by a chair or system administrator.

E.1.2.2.1 URI Parameters

- LobbyId: string

E.1.2.2.2 Response: 204

E.1.2.2.3 Response: 401

E.1.2.3 /lobbies/{lobbyId}/{userId}

E.1.2.4 delete

Evicts a user from a lobby and sends a disconnect message to the user's client devices. Also notifies remaining clients by sending a leave notification. This is an administrative function used for moderation purposes and is intended for system administrators.

E.1.2.4.1 URI Parameters

- LobbyId: string
- UserId: string

E.1.2.4.2 Response: 204

E.1.2.4.3 Response: 401

E.2 JSON Schema for Lobby Service Web Socket Communications


```
{
	"$schema": "http://json-schema.org/draft-04/schema#",
	"title": "Lobby websocket protocol",
	"description": "JSON payloads for requests, responses and events transfered to the lobby service via
websocket",
	"type": "object",
	"oneOf": [{
		"$ref": "#/definitions/join"
	}, {
		"$ref": "#/definitions/members"
	}, {
		"$ref": "#/definitions/error"
	}, {
		"$ref": "#/definitions/leave"
	}, {
		"$ref": "#/definitions/broadcast"
	}, {
		"$ref": "#/definitions/joined"
	}, {
		"$ref": "#/definitions/left"
	}],

	"definitions": {

		"lobbyId": {
			"type": "string",
```

```
			"pattern": "^[a-z]+-[a-z]+-[a-z]+$"
		},

		"appdata": {
			"type": "object",
			"description": "Application-defined data object"
		},

		"join": {
			"description": "Join lobby request",
			"properties": {
				"type": {
					"type": {
						"enum": ["join"]
					}
				},
				"lobbyId": {
					"$ref": "#/definitions/lobbyId"
				},
				"appdata": {
					"$ref": "#/definitions/appdata"
				}
			},
			"required": ["type", "lobbyId"],
			"additionalProperties": false
		},

		"members": {
			"description": "Members list returned in response to a successful join request",
			"properties": {
				"type": {
					"type": {
						"enum": ["members"]
					}
				},
				"members": {
					"type": "array",
					"description": "All lobby members as a list of userIds",
					"items": {
						"type": "string"
					}
				}
			},
			"required": ["type", "members"],
			"additionalProperties": false
		},

		"error": {
			"description": "Error response",
			"properties": {
				"type": {
					"type": {
						"enum": ["error"]
					}
				},
				"code": {
					"type": "integer"
				},
				"message": {
```


 © 2-IMMERSE Consortium 2017

```
                    "type": "string"
                }
            },
            "required": ["type", "code", "message"],
            "additionalProperties": false
        },

        "leave": {
            "description": "Leave lobby request",
            "properties": {
                "type": {
                    "type": {
                        "enum": ["leave"]
                    }
                }
            },
            "required": ["type"],
            "additionalProperties": false
        },

        "broadcast": {
            "description": "Broadcast to lobby clients request",
            "properties": {
                "type": {
                    "type": {
                        "enum": ["message"]
                    }
                },
                "message": {
                    "type": "object",
                    "description": "Application-defined message object"
                }
            },
            "required": ["type", "message"],
            "additionalProperties": false
        },

        "joined": {
            "description": "Member joined lobby event",
            "properties": {
                "type": {
                    "type": {
                        "enum": ["joined"]
                    }
                },
                "userId": {
                    "type": "string"
                },
                "appdata": {
                    "$ref": "#/definitions/appdata"
                }
            },
            "required": ["type", "userId"],
            "additionalProperties": false
        },

        "left": {
            "description": "Member left lobby event",
            "properties": {
                "type": {
```


© 2-IMMERSE Consortium 2017

```
                        "type": {
                            "enum": ["joined"]
                        }
                    },
                    "userId": {
                        "type": "string"
                    }
                },
                "required": ["type", "userId"],
                "additionalProperties": false
            },

            "disconnect": {
                "description": "User was disconnected from a lobby event",
                "properties": {
                    "type": {
                        "type": {
                            "enum": ["disconnect"]
                        }
                    }
                },
                "required": ["type"],
                "additionalProperties": false
            }
        }
}
```


 © 2-IMMERSE Consortium 2017

Annex F 2-IMMERSE Timeline Synchronisation

The Timeline Synchronisation solution in 2IMMERSE is a collection of services, protocols and components that enable DMApp components on devices participating in an experience to synchronise to a source of timing information representing the progress of the experience.

> *The temporal progress of the experience is represented by a timeline called the* ***Synchronisation Timeline****.*

DMApp Synchronisation in 2IMMERSE seeks to support both **intra-home synchronisation** (also called interdevice synchronisation) and **inter-home synchronisation**. Our solution combines standardised mechanisms for synchronisation-in-the-home (such as DVB-CSS) with cloud-based services and proposes new protocols to achieve its distributed synchronisation offering.

F.1 Intra- Home Synchronisation

Companion devices synchronise their content playback to a master device e.g. a TV playing a broadcast or IP-delivered stream. All devices reside on the same network and each device instructed by the Timeline Service to load one or more DMApp Component to play media objects as part of the experience. The main screen (usually the TV, here we assume an HbbTV2.0 device) is instructed to play the master media stream i.e. a video stream representing the TV programme.

Interdevice synchronisation in the home is achieved using the HbbTV2.0 Media Synchronisation mechanism (HbbTV2.0 specifications can be found here), itself a reification of the DVB-CSS specifications.

In this particular context, the main device is assumed to be the TV and in terms of synchronisation responsibilities, the master. In the same vein, the companion devices are synchronisation slaves. The timeline of the master device's content e.g. the timeline of a TV programme is used as the synchronisation timeline. Provided mappings between the Synchronisation timeline and the DMApp media objects' timelines are available, the companion devices can synchronise their DMApp components to the master TV using the DVB-CSS suite of protocols.

Note: Time synchronisation between devices, and between devices and cloud-based services is used to establish a common time reference. WallClock synchronisation between the TV and the companion device on the home network is distinct from WallClock synchronisation between the TV and the cloud-based Synchronisation Service. Inter-device time synchronisation is performed using DVB-CSS (CSS-WC protocol). Time synchronisation between a home device and the cloud-based Synchronisation Service is done via a generalisation of CSS-WC protocol, called WCSync.

Figure 16 below shows the Services, components and APIs used for intra-home synchronization.

Figure 16 - Services, components and APIs for intra-home synchronisation

 © 2-IMMERSE Consortium 2017

F.1.1 Cloud Services

The Timeline Services causes DMappComponents to be loaded onto the TV and the companion device (via a layout update sent by the Layout service).

The Synchronisation Service is used in the intra-synchronisation setup for providing a time synchronisation facility for the purpose of keeping the Timeline Service informed about the progress of the experience (via a CLOCK_CHANGE call on the Timeline Service API).

It exposes a WallClock-Sync protocol endpoint for the Timeline Service to advertise to the main TV DMApp component (the one the Timeline Service will consider to be running synchronisation timeline). WallClock synchronisation with the cloud-based Synchronisation Service at the TV is required so that the TV can report the progress of the synchronisation timeline (main media object timeline on TV) to the Timeline Service. Pairing a media time with a wallclock time renders a timeline update less susceptible to the extreme variations in propagation times.

Cloud Services	Description	Available	Repo	Owner
WC-Sync protocol server, JS client	Web-sockets based time sync protocol	dvbcss-clocks: timeline clocks in JS	https://gitlab-ext.irt.de/2-immerse/dvbcss-clocks	BBC – Rajiv/Matt
Synchronisation Service	Service to enable sync for a session	pydvbcss: TS-Server (python), timeline clocks in python dvbcss-clocks: timeline clocks in JS	https://gitlab-ext.irt.de/2-immerse/cloud-sync	BBC - Rajiv
Timeline Service (TS-Client/TS-Server)			https://gitlab-ext.irt.de/2-immerse/timeline-service	CWI -Jack

F.1.1.1 WallClock Sync Protocol

This protocol is similar in principle to DVB-CSS's WallClock synchronisation protocol but provides a choice of transports (UDP, WebSockets) and message serialisation capabilities (to UDP message, to JSON, etc).

F.1.1.2 Synchronisation Service API

See section 3.6.1.

TV Components	Description	Available	Repo	Owner
HbbTV2.0 MediaSynchroniser API	A synchroniser object for apps in HbbTVs; JS API	DVB CSS "TV in a browser": dvbcsstv-lib JS library + proxy server	https://gitlab-ext.irt.de/2-immerse/dvbcss-browser-proxy	BBC –Matt
Device Discovery API	Fraunhofer Fokus node-hbbtv		https://gitlab-ext.irt.de/2-immerse/cordova-plugin-	IRT-Michael/Christoph

			discovery	
2IMMClient API (WC-Client, DMApp timeline status)	Wallclock sync client and reporting DMAppC timeline progress	Christoph's JS scheduler	https://gitlab-ext.irt.de/2-immerse/client-api	BT - Jonathan
DMAppComponent	Read timeline progress		https://gitlab-ext.irt.de/2-immerse/client-api	BT - Jonathan

F.1.2 TV Components/APIS

These are APIs for components and services made available to an HbbTV web app as JS libraries.

F.1.2.1 MediaSynchroniser API (from HbbTV2.0 specifications)

See section 3.6.5 HbbTV's Media Synchroniser API

F.1.3 Companion Device Components/APIs

These are native and web-based components that export APIs to synchronise the playback of DMApp components based on timeline updates received from the TV via the DVB-CSS suite of protocols.

Companion Device Components	Description	Available	Repo	Owner
CSS-SyncKit Native Library (iOS, Android)	A native library for DVB-CSS-enabled synchronisation on companion devices	CSS-SyncKit-iOS (BBC), CSS-SyncKit-Android (IRT)	https://github.com/bbc/dvbcss-synckit-ios	BBC – Rajiv IRT – Michael/ Fabian
Simple CSS-SyncKit JS API	A basic API to allow an object to register for TV timeline updates	ios_sync.js	https://github.com/bbc/dvbcss-synckit-ios	BBC – Rajiv IRT – Michael/ Fabian
CSS-SyncKit JS API (synckit.js) (Synchroniser, SyncController JS objects)	An API that allows a sync controller object to be created and plugged into DMAppComponents for synchronisation with a DVB-CSS TV			BBC – Rajiv IRT – Michael/ Fabian
Native Media Players JS API	API to load and control native media players from JS	iOS, Android AVPlayer, dvbcss-synckit-ios audio player		
Device Discovery JS API	API to discover HbbTV devices	dvbcss-synckit-ios (BBC) cordova-plugin-discovery (IRT)	https://gitlab-ext.irt.de/2-immerse/cordova-plugin-discovery	IRT – Michael/ Christoph

 © 2-IMMERSE Consortium 2017

Companion Device Components	Description	Available	Repo	Owner
DMAppComponent MediaControl API	API to control the playback of media in a DMAppComponent			BT-Jonathan

F.1.3.1 SyncKit Synchroniser API

If a Companion Screen Application wants to participate in inter-device synchronisation, it creates a CSS-Synchroniser object. This is a singleton of type CSS-Synchroniser that allows a Companion Screen App to synchronise its media players to a timeline produced by the DVB-CSS protocols.

The factory object is available on the global scope in the webview environment via a plug-in.

If a TV/mobile web application wants use the 2IMMERSE cloud-based Synchronisation Service, it creates a Synchroniser object.

See section 3.6.4 for more details.

F.1.3.1.1 SyncController Object

A SyncController is an object that can be plugged into a DMAppComponent to synchronise that component's playback to the Synchronisation Timeline. In the intra-synchronisation configuration, a SyncController will receive Synchronisation Timeline updates from the DVB-CSS machinery (CSS-SyncKit library on companion devices). It is in fact created using the CSS-SyncKit Synchroniser object.

The following code snippet illustrates how a SyncController is created and attached to a DMAppComponent.

```
// assume a DMAppComponent object dmappc
var synchroniser = objectFactory.createSynchroniser();
synchroniser.initSynchroniser(sync_url,SET);
synchroniser.enableSynchronisation();
var syncControllerObject = synchroniser.createSyncController(correlation);
dmappc.syncController = syncControllerObject;
```

A SyncController will tell a DMAppComponent the media time it has to reach at a particular system time in the future by passing it a correlation timestamp (media time, host time). It is up to the DMAppComponent to then apply the necessary playback adaptation tactic to fulfil this new (media time, host time) relationship. For example, to maintain a level of QoE, a DMAppComponent may decide to increase the playback speed slightly (x1.2) till it has caught up with the new timing relationship (between media time and host time).

F.1.3.2 JS Native Media Player Control API

This API defines a set of common properties and operations for native media players including, positioning properties such as size, coordinates

Using this API, a native player can be instantiated and manifested to a web-app as a browser-based DMAppComponent and then controlled via a browser-based SyncController object.

The specification of this API and its implementation is planned for the second-year of this project.

F.2 Inter-Home Synchronisation

In the inter-home synchronisation scenario, devices at different locations synchronise their content playback as part of a distributed synchronised experience. The synchronisation timeline can be one of the following:

1. the **timeline of an elected master device**,
   The timeline used for synchronisation comes from a particular DMAppC component on a device e.g. a DMAppC component on a TV in Home1 playing a broadcast stream. This broadcast stream could provide a TEMI or a PTS timeline to be used for synchronisation

2. a **timeline set by a central coordinator** e.g. an experience timeline as set by the Timeline Service

For our use cases, we assume the latter scenario - the Timeline Service is responsible for starting the experience and for informing participating devices of the progress of the experience 's timeline. The Experience Timeline is the Synchronisation Timeline. The Timeline Service is also responsible to start a Synchronisation Service instance to enable synchronisation for that particular experience instance. For recall, an experience is a DMApp (Distributed Media Application) described by a DMAppSpec document and can span multiple homes/locations. A DMAppSpec is an experience specification that describes the following:

1. DMAppComponents that must be instantiated
2. DMAppComponent repository location
3. Schedule for each DMAppComponent with respect to the experience timeline,
4. Time correlations for each DMAppComponent's media object with the experience timeline e.g. (mediatime=15.34s, experiencetime=20.45s)

A SyncService instance is created by the Timeline Service to propagate Synchronisation Timeline updates (i.e. timestamps on the experience timeline) to all 2IMMERSE applications within the same session (experience instance). The devices running the applications may span across multiple contexts (and physical locations) but must belong to the same session.

The SyncService exposes 3 endpoints to services and applications participating in the synchronised experience:

1. A WCSync service interface to enable WallClock synchronisation via the WCSync protocol (an variant of the CSS-WC protocol)
2. A TimelineSync service interface to enable timeline synchronisation updates to be sent to all slave terminals. The slave applications
3. A TimelineSync Master service interface to allow a master entity e.g. the Timelime Service to push timestamps (WallClock time, sync timeline time) to the TimelineSync Service and eventually to all the slave terminals.

 © 2-IMMERSE Consortium 2017

Figure 17 below shows how Inter-home media synchronisation is achieved via the cloud-based synchronisation service.

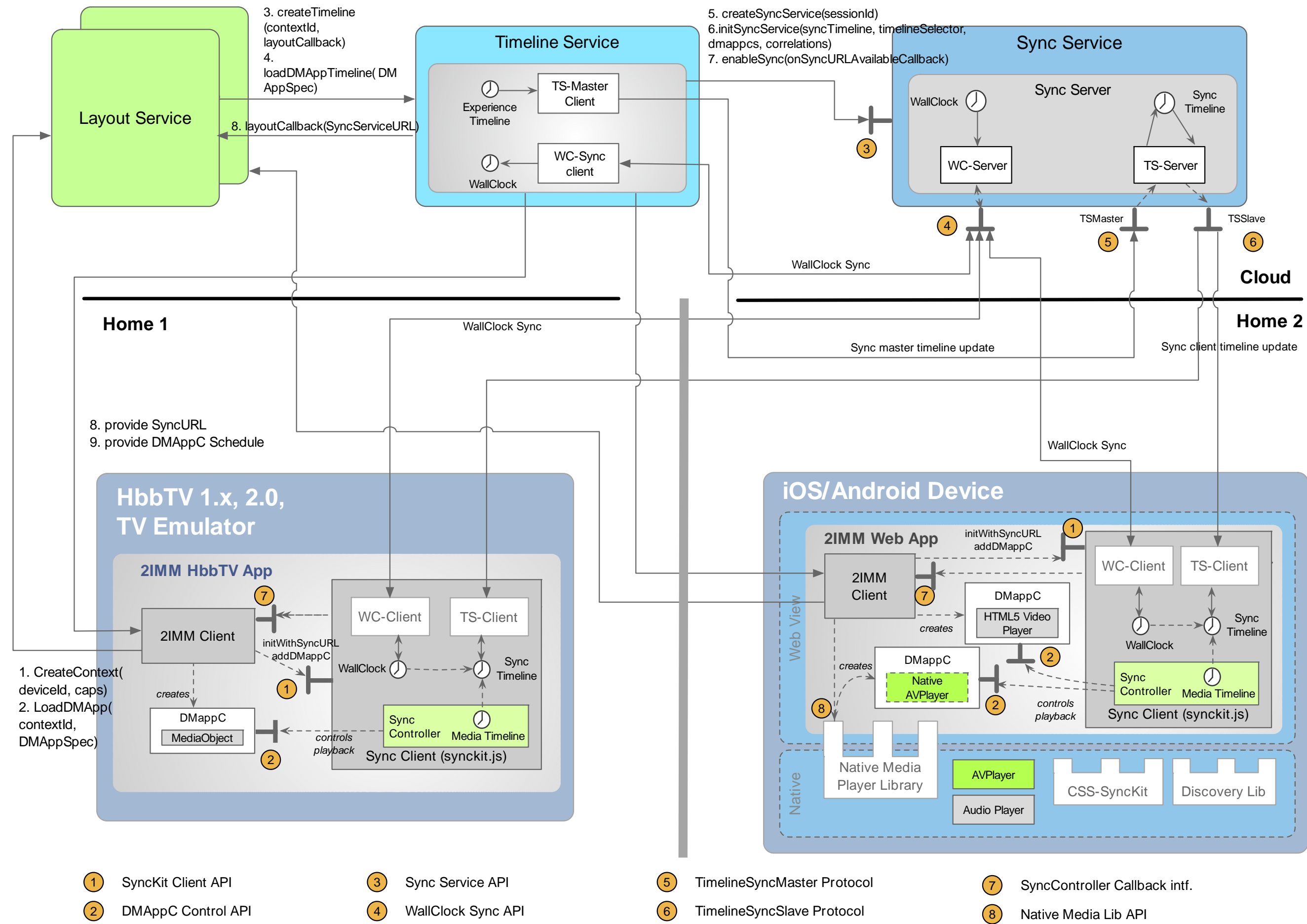

Figure 17 - Inter-home media synchronisation via the cloud-based synchronisation service

 © 2-IMMERSE Consortium 2017

F.2.1 Cloud Services/APIs

Cloud Services	Description	Available	Repo	Owner
WC-Sync protocol server, JS client	Web-sockets based time sync protocol	dvbcss-clocks: timeline clocks in JS	https://gitlab-ext.irt.de/2-immerse/dvbcss-clocks	BBC – Rajiv/Matt
Synchronisation Service	Service to enable sync for a session	pydvbcss: TS-Server (python), timeline clocks in python dvbcss-clocks: timeline clocks in JS	https://gitlab-ext.irt.de/2-immerse/cloud-sync	BBC - Rajiv
Timeline Service (TS-MasterClient, WC-Client)			https://gitlab-ext.irt.de/2-immerse/timeline-service	CWI -Jack
TSMaster and TS-Slave Protocol	Protocol servers and clients			BBC-Rajiv/CWI-Jack

F.2.1.1 WallClock Sync (WC-Sync) Protocol

The WC-Sync protocol performs time synchronisation at the application-layer to establish a common time reference between distributed services and applications. Although, NTP is widely used for synchronising system time, the availability of a synchronised system clock within application environments such as browsers cannot be assumed for the range of devices envisaged for our experiences. Thus, the need for a bespoke, lightweight time-synchronisation protocol.

The WC-Sync protocol synchronises software clocks at the device applications with the WallClock at the Synchronisation service. It is similar in principle to DVB-CSS's WallClock synchronisation protocol but provides a choice of transports (UDP, WebSockets) and message serialisation capabilities (to UDP message, to JSON, etc).

An estimate of the Synchronisation Timeline at each device is predicated from its synchronised local WallClock. The largest contributor to the Synchronisation Timeline error is therefore the WallClock synchronisation error. Hence, a large error in the WallClock synchronisation is likely to limit our ability to synchronise media objects on distributed devices accurately. For frame-accurate synchronisation of video (at 50 fps), a synchronisation accuracy of ~10ms is desirable.

F.2.1.2 Synchronisation Service API

See section 3.6.1

F.2.2 TV Components/APIs

TV Components	Description	Available	Repo	Owner

TV Components	Description	Available	Repo	Owner
SyncKit: JS timeline clocks Wallclock Sync protocol client, TS-sync Client, SyncController		Early version of timeline clocks in JS, CSS-TS client in JS	SyncKit, SyncController: https://gitlab-ext.irt.de/2-immerse/synckit https://gitlab-ext.irt.de/2-immerse/dvbcss-clocks TS-Client: https://gitlab-ext.irt.de/2-immerse/sync-protocols	BBC – Matt/Rajiv
Native Media Players		iOS, Android AVPlayer, synckit-iOS audio player		
2IMMClient		IRT's JS scheduler		
DMAppComponent + control interface				

F.2.3 Companion Device Components/APIs

Companion Device Components	Available Implementations/ reusable parts
SyncKitClient (JS timeline clocks, wallclock sync, TS-sync, sync controller)	Early version of timeline clocks in JS, CSS-TS client in JS
Native Media Players	
	IRT's JS scheduler
DMAppComponent + control interface	

 © 2-IMMERSE Consortium 2017