I. Intro

The motivation for the 2Immerse layout service is to provide a cloud based, highly available, layout engine that could provide a layout for a dynamically varying set of digital application components over a set of varying viewing devices. The service manages the set of devices and running components and provides the layout engine with a full layout specification for each layout computation. It then returns a list of components per device, with region, position and size.

The engine computes layouts for a ***context***. A context is comprised of devices, ***DMApps*** (Distributed Media Applications). In particular, it contains groups of display devices which may be communal (e.g. TV) or personal devices (e.g. mobile phone) and DMApps, each containing a set of rectangular display objects, termed DMAppComponents, and henceforth referred to as ***components***. Components are rectangular and possess attributes determining its size and position ***constraints*** as well as device compatibility. The constraints for the components in a DMApp are defined in the layout rules document. Constraints are defined per component per device type (communal vs. personal). Components may be visible (or active) or invisible.

The Layout service maintains the context, DMApp & component definitions and state in a backend (MongoDB). Whenever the state changes, a new layout is automatically computed.

The layout engine computes a layout for each group and lays out the active devices over the set of devices in each group. In addition, logical regions may be defined over the physical devices which do not have to be the same size as the physical device, for example, a region that is larger than the device display can be used for scrolling.

II. Defining the Constraint Document

The constraint document is written in JSON format according to the specifications defined in ***api/v4-document-schema.json***. Definitions contain the API version level (=4), and an array of constraints. Each constraint has an ID, a constraint definition for personal devices and a constraint definition for communal devices. Each definition must specify a positive numeric priority, where higher values represent higher priorities and 0 represents invisible.

All size definitions are specified using a JSON that defines the ***width*** and ***height*** using numeric values, and the **mode** with which to interpret the values as a string, with value ***percent***, ***px*** or ***inches***.

Values that may be defined include

- ***priority*** **-** the numeric priority level of the component, where higher values represent higher priorities and 0 represents invisible.
- ***aspect*** – maintain the specified aspect ratio when sizing the rectangle. Aspect is specified as a string on the format ***<w>:<h>***, e.g. “16:9”
- ***minSize*** – the minimum size that a component can be resized to.
- ***prefSize*** – the maximum size for a component. The algorithm attempts to use this size.
- ***targetRegions*** – the set of regions upon which the component may be placed. If not specified, the component will be placed on any region that meets the other requirements.

- ***audio*** – if **true**, the component will be placed on a region where the underlying device has audio capabilities defined.
- ***video*** - if **true**, the component will be placed on a region where the underlying device has video capabilities defined.
- ***anchor*** – a component may be anchored to one or more of the region borders, or be centered vertically. Anchors are specified as an array of strings which may be one or more of the values ***right, left, top, bottom***, and ***vcenter***.

III. Generating a Layout

The sequence of events needed to generate a layout are as follows:

- Create a context and store its id.
- Add device(s) to the context.
- Add a DMApp (specifying the layout rules document URL to the context.
- Initialize components
- Start components.

Initializing components triggers a simulated layout generation, so that the calling app can know on which device a component will most likely be displayed and can preload prerequisites.

Starting the components, and all subsequent state changes will trigger a complete layout computation. The actual layout is sent via the **Websocket Service** to all registered listeners. It is also possible to retrieve a full layout directly using the REST API.

IV. Sample Layout Service API Calls

The complete API definition can be found in **api/layout-service.raml**

Context and DMApp ID names are generated by the service, all other IDs are supplied by the caller.

Note: the reqDeviceID (requesting device ID) field in the REST query parameters is used for logging.

A. Context

Create

POST /layout/v4/context?reqDeviceId=<requesting device id> Returns Context ID.

Get

GET /layout/v4/context/{contextID}?reqDeviceId=<requesting device id>

Delete

DELETE /layout/v4/context/{contextID}?reqDeviceId=<requesting device id>

Retrieve last computed layout

GET /layout/v4/context/{contextID}?reqDeviceId=layoutRenderer

B. Devices

Add a new device

Device properties include display width and height in pixels, the number of audio channels, the maximum number of concurrent audio and video components supported, touch support, valid orientations which may be one or more of ***landscape*** and ***portrait***, communal or personal device, device type specified as a string, resolution specified as dpi, and a list of regions specifications.

Each **region** has an id and display width and height in pixels.

```
POST /layout/v4/context/{contextID}/devices?deviceId=<device ID>&orientation=<orientation>&reqDeviceId=<id>
BODY:
        {
         "displayWidth":1920,
         "displayHeight":1080,
         "audioChannels": <number supported audio channels>,
         "concurrentVideo": <number supported simultaneous video components>,
         "concurrentAudio": <number supported simultaneous audio components>,
         "touchInteraction": true | false,
         "communalDevice": true | false,
        "deviceType": "tv" | "communal"
        "orientations": [ <valid orientations>]
        "regionList": [
                    {
                       "regionId": <id>
                       "displayWidth": 1280,
                       "displayHeight": 50
                     }
        ]
        }
```

Retrieve device information and status

```
GET /layout/v4/context/{contextID}/devices/{device id}?reqDeviceId=<id>
```

Change device orientation

```
PUT /layout/v4/context/{contextID}/devices/{device id}/orientation?orientation=<orientation>&reqDeviceId=<id>
```

Update device display resolution

```
PUT /layout/v4/context/{contextID}/devices/{device id}/displayResolution? reqDeviceId=<id>
BODY
{
      displayWidth: <width>
      displayHeight: <height>
}
```

Destroy the device

```
DELETE /layout/v4/context/{contextID}/devices/{device id}?reqDeviceId=<id>
```

C. DMApp

Create a new DMApp

POST /layout/v4/context/{contextID}/dmapp?reqDeviceId=<id>

BODY

```
{
   "timelineDocUrl": "http://2immerse.eu/apps/testDMApp1/timeline.json",
   "timelineServiceUrl":  "http://localhost:8001/timeline/v1",
   "layoutReqsUrl": "<origin server>/layout.json",
   "extLayoutServiceUrl":"http://localhost:8000/layout/v1"
}
```

Returns DMApp ID.

The timeline elements are needed only if running with a Timeline Service.

Get DMApp

GET /layout/v4/context/{contextID}/dmapp/{dmappID}?reqDeviceId=<id>

Retrieve the set of active components with their state and layout.

GET /layout/v4/context/{contextID}/dmapp/{dmappID}/components?reqDeviceId=<id>

Delete a DMApp

DELETE /layout/v4/context/{contextID}/dmapp/{dmappID}?reqDeviceId=<id>

D. DMAppComponent

- Retrieve a components state and layout

GET /layout/v4/context/{contextID}/dmapp/{dmappID}/components/{componentID}?reqDeviceId=<id>

- Specify DMappComponents state change, where valid states are: init |start | stop | destroy

More than one state change may be set for one or more components in one transaction.

POST /layout/v4/context/{contextID}/dmapp/{dmappID}/components/transaction

BODY

```
{
  "time": 1,
  "actions":[{
    "action": <state>
     "components": [
                  {
                              "componentId": "dmappcid_masterVideo",
                               "constraintId": "master_video"
                  },
                              "componentId": "component2",
                              "constraintId": "constraint2"
                  }
    ]
```

```
    }]
}
```

- Update the priority of a component

1. Personal / communal priorities are set individually
2. Scope specification for the priority, there are 3 valid values:
    a. “device’ -> per device only
    b. “group” -> all devices in the specified group
    c. “context” -> all devices the current context
3. Multiple updates in one call:
    a. Multiple priority overrides per scope/devtype are specified in an array
    b. Multiple overrides are specified in an array

POST /layout/v4/context/{contextID}/dmapp/{dmappID}/components/{componentID}/setPriority

BODY

```
[{
    “scope”: “group” | “context” | “device”,
   “devtype”: “personal” | “communal”,
   “overrides”: [
       {
        “id”: <group id> | “context” | <device id>,
       “priority”: <priority>
       }
    ]
}]
```

- Update the preferred size of a component

POST /layout/v4/context/{contextID}/dmapp/{dmappID}/components/{componentID}/setPrefSize?reqDeviceID=<device upon which to change the preferred size>

BODY

```
{
   “height”: <height in mode units>,
   “width”: <width in mode units>,
   “mode”: px | inches | percent
}
```

V. Layout Service Message Format

Overview

Various API calls / interactions will result in the layout service evaluating a new layout for a context (and the DMApp(s) running in it). When this happens, the layout service needs to push a notification to each of the affected clients, so that they can update their presentation accordingly.

Registering for Messages via WebSocket Service

Join the websocket with the JSON:

```
{
        room: <context ID>.<device ID>
}
```

The API calls that will result in a new layout evaluation are:

- from a Timeline Service instance:
- start DMApp component
- stop DMApp component

N.B. calls to init DMApp component do not trigger a full layout evaluation - see create message below

- from a client device
- hide DMApp component
- show DMApp component
- move DMApp component
- change device orientation
- join context
- leave context

Basic requirements

To push 'instructions' to the client(s) about what they need to do to manage DMApp components, that are

- simple for the client to process
- minimise the need for the layout service to track client state
- Reuse JSON schema objects for DMApp components from REST API RAML spec.

Messages

In essence we can distill this down to 3 types of action for one or more DMApp components:

- create (what, and optionally, where & when)
- destroy (what, and optionally whether to save state for a move)
- update (what, and partial where / when)

where:

- 'what' will include a unique DMApp component reference (i.e. a contextId/DMAppId/componentId triple), and may include config information including if state needs to be saved or restored
- 'where' will include device / layout information
- 'when' will include start or stop time i.e. when the change should be enacted.

N.B.

- Each message has a sequential messageId to make it easy to track / debug, and also to allow clients to acknowledge receipt / action of messages using status calls...
- Each message can address a list of DMApp components.

- A component move is handled as a destroy on the old device (optionally saving component state), and a create on the new device (optionally restoring component state)
- These messages will be sent from layout service to clients using the websocket service: https://2immerse.eu/wiki/websocket-service/

Create

```
  "create": {
    "messageId": 1,
    "deviceId": "dev0",
    "components": [ {
      "componentId": "1",
      "DMAppId": "200",
      "contextId": "0",
      "config": {
        "class": "dashvideoplayer",
        "url": "2-immerse.eu/mydmapp/videoplayer",
        "restoreState": true
      },
      "startTime": null,
      "stopTime": null,
      "layout": null
    },
    {
      "componentId": "2",
      ...
    } ]
  }
}
```

This message is sent whenever a timeline service instance calls 'init DMApp component', or as part of a component move sequence.

In the case where a timeline service instance calls 'init DMApp component': A 'pre-layout' is performed, this simply identifies which of the devices the component could run on, and a create message is sent to each of the devices to allow them to speculatively pre-load the component, using the information in the config child object. startTime, stopTime & layout attributes will be null.

In the case where this happens as part of a component move sequence: startTime & layout attributes will be fully populated. If a config.restoreUrl attribute is provided, the client should restore component state from this API endpoint.

Destroy

```
  "destroy": {
    "messageId": 2,
    "deviceId": "dev0",
    "components": [{
      "messageId": 2,
      "DMAppId": "200",
      "contextId": "0",
```

```
        "config": {
          "saveState"
          : true
        },
        "stopTime": "0:12:20.0"
      },
      {
        "messageId": 2,
        ...
      } ]
    }
}
```

This message is sent whenever a timeline service instance calls 'stop DMApp component', or as part of a component move sequence.

At the specified stopTime, the component should be removed. If a config.saveUrl attribute is provided , the client should save component state to this API endpoint.

Update

```
"update":{
      "messageId":3,
      "deviceId":"dev0",
      "components":[{
        "componentId":"1",
        "DMAppId":"200",
        "contextId":"0",
       "startTime":"0:34:03.1",
       "layout":{
        "position":{"x":5,"y":5},
        "size":{"width":800,"height":500},
        "deviceId":"1238"
        }
      },{
      "componentId":"2",
      ...
   }]
  }
}
```

This message is sent whenever layout for an existing component changes (typically position, size or zdepth), or a component is hidden / shown under user control (via a client device). The presentation should be updated at the specified startTime. When a component is hidden, the update message layout.position the child object will be set to null, and when it is re-shown it will contain a fully specified position. deviceId will not change on an update message, since this is handled through a component move sequence.

Implementation Considerations

In an optimal implementation, push messages will only be sent to devices that are hosting the component which is the subject of the message. However, clients should not assume this to always be the case (i.e. they may receive message about components they do not host).

The timeline service will preferably send component start & stop messages just ahead of time i.e. giving enough time to evaluate layout and push the resulting messages to affected clients. Calculating layouts and pushing messages to clients a long time in advance will require a significantly more complex state management implementation in the layout service, and possibly require that the layout service tracks timeline; neither is desirable as they bring complexity to the design and implementation.

In a situation where multiple components are started / stopped at the same time, there may be value in a 'batch' or 'list' style call where start or stop times can be set for multiple components in a single call. Without this we will end up sequentially iterating a layout, with each iteration resulting in a set of transient push messages. It would be cleaner and simpler to evaluate the layout start / stop times are know for all the affected components.

IV. Using the Layout Service Standalone

You can use the layout renderer app to render a layout wireframe of the active layouts by running the layout server. The layout services requires a running Mongodb it can connect to for its backend, however the remainder of the servers are optional, i.e., it does not require timelime service, websocket service or consul. If the websocket service is specified it will push messages that can be retrieved by other applications.

Command line options include:

```
.option("-p, --port <port>", "Port to listen on", 3000)
.option("-w, --websocket <service>", "WebSocket service name", "websocket-service-dev")
.option("-t, --timeline <service>", "Timeline service name", "")
.option("-m, --mongodb <service>", "MongoDB service name", "mongodb-dev")
.option("-d, --database <database>", "Database name", "layout_service")
.option("-c, --consul <host>", "Consul host", "https://consul.service.consul:8500")
.option("-v, --verbose", "Verbose logging (deprecated use '--log-level DEBUG' instead)", false)
.option("-l, --log-level <level>", "Log level [DEBUG, INFO, WARN, ERROR, FATAL]", "INFO")
```

The basis API sequence is:

1. Create a context

Create context

POST {{url}}/layout/v3/context?deviceId=immersivelab&reqDeviceId=immersivelab

which will return a contextId to use in subsequent API calls, e.g.

```
{
        "contextId": "2f6c3300-409e-11e7-b690-f3c0d1052922"
}
```

2. Create a device in the context

Add device to context

POST
{{url}}/layout/v3/context/{{contextId}}/devices?deviceId=immersivelab&reqDeviceId=immersivelab&orientation=landscape

body:

```
{
  "capabilities":{
        "displayWidth":1862,
       "displayHeight":1092,
       "audioChannels":2,
        "concurrentVideo":0,
       "touchInteraction":false,
       "communalDevice":true,
       "orientations" :["landscape"],
       "deviceType":"tv"
    },
    "regionList":[]
}
```

Load DMApp

POST {{url}}/layout/v3/context/{{contextId}}/dmapp?reqDeviceId=immersivelab

body:

```
{
        "timelineDocUrl":"",
        "layoutReqsUrl":"<origin-server>/layout-v3-noregions.json",
        "extLayoutServiceUrl":" /layout/v4",
        "timelineServiceUrl":""
 }
```

which will return a dmappId in the response to use in subsequent API calls:

```
{
        "DMAppId": "adcc93a0-40a0-11e7-b690-f3c0d1052922",
        "contextId": "2f6c3300-409e-11e7-b690-f3c0d1052922",
        "spec": {
                "timelineDocUrl": "",
                "layoutReqsUrl": "<origin-server>/layout-v3-noregions.json",
                "extLayoutServiceUrl": "/layout/v4",
                "timelineServiceUrl": ""
                },
        "components": [],
        "timestamp": 3812813531677513
}
```

- The components array is empty, as no components have been added yet.
- The supplied timelineDocUrl & timelineServiceUrl are empty since we are not using the timeline service.
- You can access the current dmapp state as returned by the Load DMApp call above, including component layout, as follows:

Get DMApp Info

GET {{url}}/layout/v3/context/{{contextId}}/dmapp/{{dmappId}}?reqDeviceId=immersivelab

Once the DMApp is loaded (which for the layout service, means loading the layout constraints document), you can then make a sequence of transaction api POSTs with different JSON payloads to move components through their lifecycle state:

Component Transaction

POST {{url}}/layout/v3/context/{{contextId}}/dmapp/{{dmappId}}/transaction

Init body:

```
{
        "time": 1, "actions": [{
          "action": "init", "config": {
            "url": "",
            "class": "video"
          },
          "componentIds": ["dmappcid_masterVideo"], "parameters": {
            "debug-2immerse-dur": "3600",
            "syncMode": "master",
            "mediaUrl": "client_manifest.mpd"
          }
        }]
      }
```

Start body:

```
{
        "time": 11, "actions": [{
          "action": "start",
          "componentIds": ["dmappcid_masterVideo"]
        }]
      }
```

NB - at this point if you do a 'get DMApp Info' you should see the component layout info in the response...

Stop body:

```
    {
        "time": 21, "actions": [{
        "action": "stop",
        "componentIds": ["dmappcid_masterVideo"]
        }]
    }
```

Destroy body:

```
{
       "time": 31, "actions": [{
       "action": "destroy",
       "componentIds": ["dmappcid_masterVideo"]
       }]
}
```

This sequence can be repeated for components in the layout constraints document as required by the application.

Note that the layout service will be pushing layout messages via the websocket service as layout is re-evauated following transaction API calls, but you can also get the updated layout as previously shown

Finally:

Unload DMApp

DELETE {{url}}/layout/v3/context/{{contextId}}/dmapp/{{dmappId}}?reqDeviceId=immersivelab

Destroy Context

DELETE {{url}}/layout/v3/context/{{contextId}}?reqDeviceId=immersivelab