A Proposal for AR Browser Interoperability

This document published by AR Standards Community. Feedback via e-mail to browserinterop@arstandards.org (and you must join the mailing list http://arstandards.org/mailman/listinfo/browserinterop to post) is welcome.

Document history

Version	Date	Remarks	Author
0.9	28 October 2013	Bundled document from separate Google Docs	Christine Perey
1.0	31 October 2013	Added diagrams and restructured document	Dirk Groten
1.1	1 November 2013	Edits to architecture description, figure titles in architecture section, added footer and detailed instructions for how to provide feedback	Christine Perey
1.2	21 January 2014	Edits to introduction Edits to CARIF	Christine Perey Martin Lechner

Authors

Editor	Affiliation	Sections
Stefan Missingler	Metaio	AR Launch URL Scheme, CDP Request RESTful API
Dirk Groten	Layar	Scenarios, Architecture and Definitions
Martin Lechner	Wikitude	Basic Feature Set, CARIF

Table of Contents

1. Introduction 2
2. Architecture and Definitions 2
3. Basic Feature Set for AR Browser Interoperability 5
4. Scenarios 6
5. AR Launch URL Scheme 9
6. CDP request RESTful API 11
7. Common AR Interchange Format (CARIF) for Basic AR Browser Interoperability 12
8. Conclusion 16

1. Introduction

This document contains the approach developed collaboratively by the participants of the AR Browser Interoperability Engineering group (organized by the AR Standards Community) to achieve basic interoperability with geospatial AR experiences published in proprietary platforms and that can be opened and presented in the user's AR browser of choice (provided that the AR Browser publishing environment supports the architecture proposed).

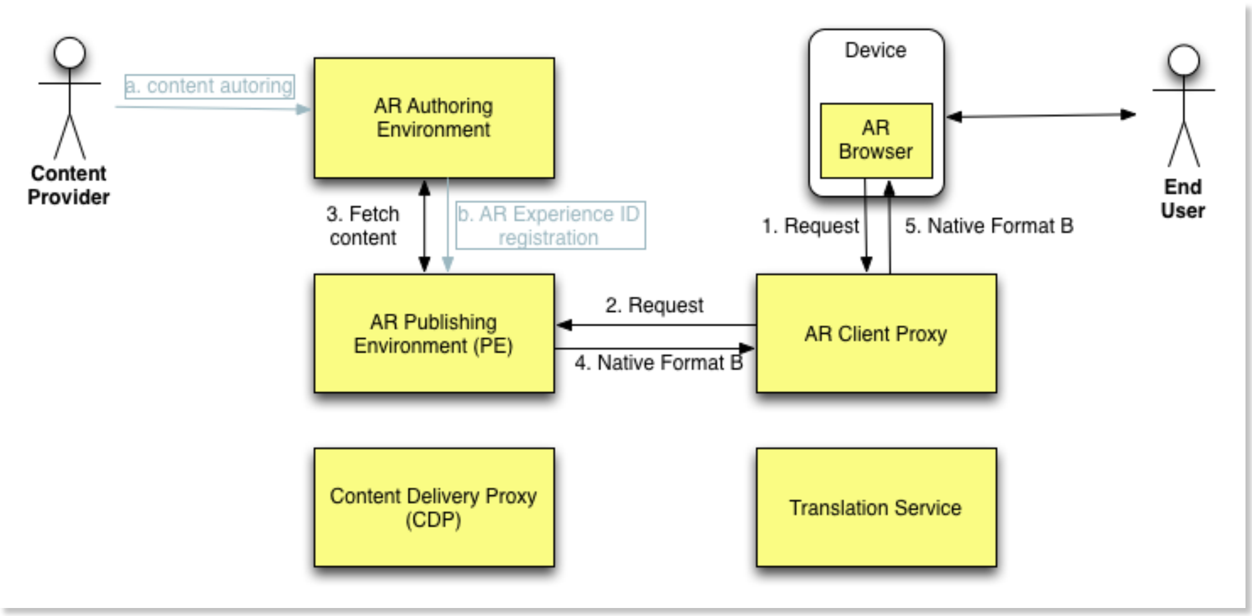
The purpose of the document is to describe the approach that has been implemented by providers of three AR browsers and which will be the basis of the first AR Browser interoperability experiment.

If you wish to provide feedback in an e-mail to the AR Browser Interoperability mailing list, register for this mailing list: http://arstandards.org/mailman/listinfo/browserinterop

2. Architecture and Definitions

The figure 1 below depicts the main components of the architecture defined in this document. Each component is defined in more detail after figure 2. The interoperability architecture maps directly to the internal architecture of modern AR Browser publishers involved in this task force. Figure 1 assumes that only the AR Browser defined by the AR Browser publisher is involved. The AR Browser always sends user requests via the AR Client Proxy. The AR Client Proxy redirects the request to and receives the experience content from the AR Publishing Environment that has requested and received the AR Experience from the service designated in the Content Provider's AR Authoring Environment.

Figure 1. Common AR Browser, Publishing and Authoring Environment Architecture



In Figure 2, the end user is also using an AR Browser published by company B. The AR Browser only supports the native API of company B's Publishing Environment and only parses content it receives in native format B. It continues to communicate user requests only to the AR Client Proxy defined by company B. The AR Client Proxy of company B will recognize that it has received a non-native request and send the request to its Translation Service. Assuming the content requested was developed and is in the format of the AR Browser Publisher company A, the Translation Service requests the content from a Content Delivery Proxy (CDP) and receives a response in a Common AR Interchange Format (CARIF). The CDP is hosted by company A and "knows" how and where to request/retrieve the content.

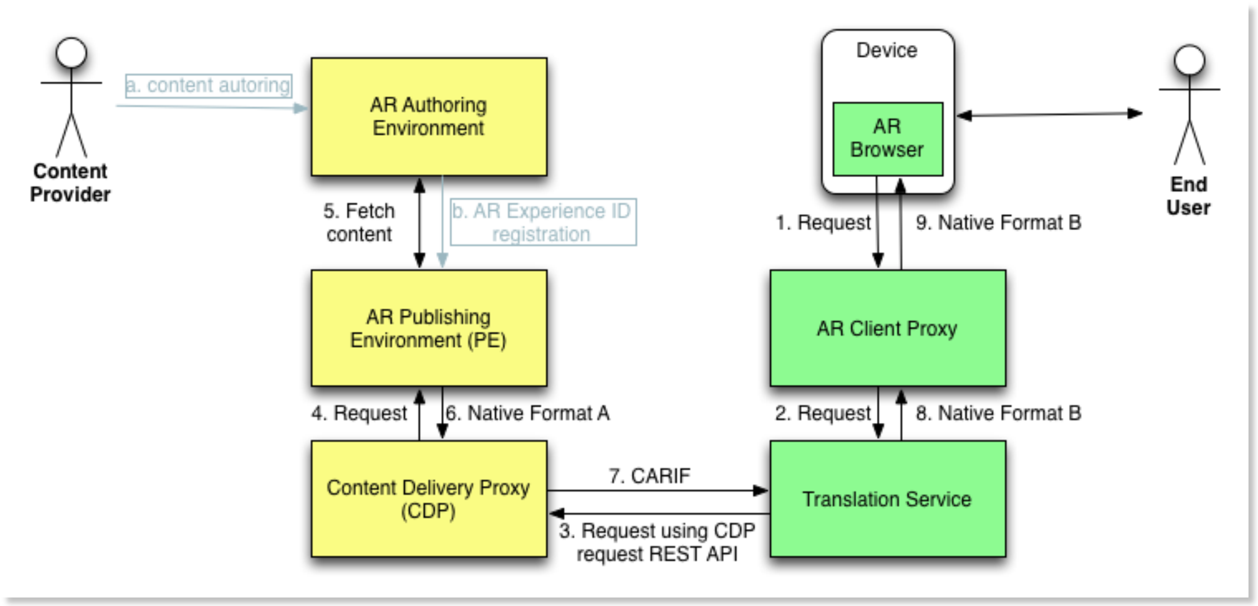


Figure 2. Proposed AR Browser Interoperability Architecture

AR Browser

AR Browsers are native applications developed and maintained by the AR Publishing Environment providers to view the AR Experiences that the content providers publish using their systems. The AR Browser application (a type of AR "Player") is typically provided at no charge on app stores as part of a closed service platform. An AR Browser uses the browser publisher's specification to

- Interpret
  - User input
  - Real world (input from local and remote sensors, either or both CV and Geospatial)
- Manage delivery/presentation of experience
- Manage communication with additional services

The publisher of the digital assets to be included in an AR Experience (aka Content Provider) provides a URI to the content handled in the mobile AR Browser, following the syntax defined in the AR Browser Publisher Environment's specifications. The AR Browser provider's AR Client Proxy redirects the request from the AR Browser.

Content Provider

An entity (person, company) that creates content for and makes the content available in AR experiences.

AR Publishing Environment (PE)

A service where the unique AR Experience ID (unique to the AR publishing environment, see further below) is created and registered; it can respond to queries to fetch the content for the AR experience and return the content in its own native format.

[Note: Is the environment where to best implement access control?]

[Note: What happens if there is paid content? The publishing environment will return an appropriate response to the requester, depending on whether the requester of the content is allowed to get the content or not. An error along the lines of "Error, this content is not available for other platforms." will be returned. As long as we have a common protocol.]

AR Authoring Environment

Quite a general term to describe any server/platform/service where content creators can create and/or host content for AR Experiences. This could be one webserver, or a collection of services split between various parties (e.g. authoring and hosting on different locations).

In the context of this document, it's an endpoint from which the PE can fetch content, using a native format that is agreed between the owner of the PE and the owner of the AR Authoring Environment. Where the content comes from, who hosts the content, in what format it is stored and whether it's dynamic or static is not relevant for the interoperability of AR Browsers described in this document.

Common AR Interchange Format (CARIF)

The format to exchange content between the Content Delivery Proxy and Translator Service.

AR Client Proxy

The client requests content from the AR Client Proxy, that always returns native format to the client. Based on the requested Experience ID, the AR Client Proxy will decide whether to invoke the Translator Service or go directly to the Publishing Environment.

Translator Service

It translates from CARIF to a native format to the client for use in AR Experiences delivery. Exact implementation is not important. This could even be on the client itself or be a simple pass-through if the client understands CARIF. The Translator Service gets the content from the Content Delivery Proxy.

Content Delivery Proxy (CDP)

Delivers the content in the CARIF to the Translator Service. It doesn't matter how it fetches the content (but in one example it can fetch it from the Publishing Environment). It converts the content from the native format to the CARIF that is defined elsewhere.

Content Delivery Proxy URL

URL of the CDP

AR Experience ID

A string that uniquely identifies the experience for the Content Delivery Proxy, who knows where and in which format the experience is hosted.

The combination of AR Experience ID and CDP URL globally identifies any AR Experience. (The CDP URL may include the AR Experience ID).

3. Basic Feature Set for AR Browser Interoperability

Each AR browser supports different features but they also have features in common across the platforms. For the initial interoperability experiment, browsers will provide interoperability for geo-based AR experiences. The following feature set is supported for the experiment:

Property	**Format**	**Description**
id	UTF8	unique identifier for a POI within the AR Experience
Location (Latitude, Longitude, Altitude)	WGS84, Altitude optional	the location
Title	UTF8	Title of the POI
Image	JPG/PNG	A thumbnail depicting the content
Description	UTF8	Description about POI

Each POI also has a set of optional actions associated with it. The AR Browser provides its own GUI to permit the user to trigger these actions.

Action Type*	**Input**	**Description**
Open website	URL to the website	Opens up a fullscreen web browser. Can also be used to open still images.
Play (fullscreen) video	URL to the video	Opens up a fullscreen video player
Play audio	URL to the audio file	Plays the audio in a (full screen?) player
Send email	Mail link in *mailto:* scheme	Opens an email composer
Call a phone number	Phone link in *tel:* scheme	Calls the phone number
Send a text message	SMS link in sms: scheme	Opens a text message composer
Route to destination	Route link in Google Maps / Apple maps scheme**	Opens up a view showing the route to the destination, whether on a map or in AR or in a separate maps application

* Each action should also contain a custom label for the user to define. So a developer could label ‘open website’ “Visit our webpage” or ‘Send email’ “Contact us”

** junaio supports route: to abstract between Google and Apple: e.g. route:daddr=123,345

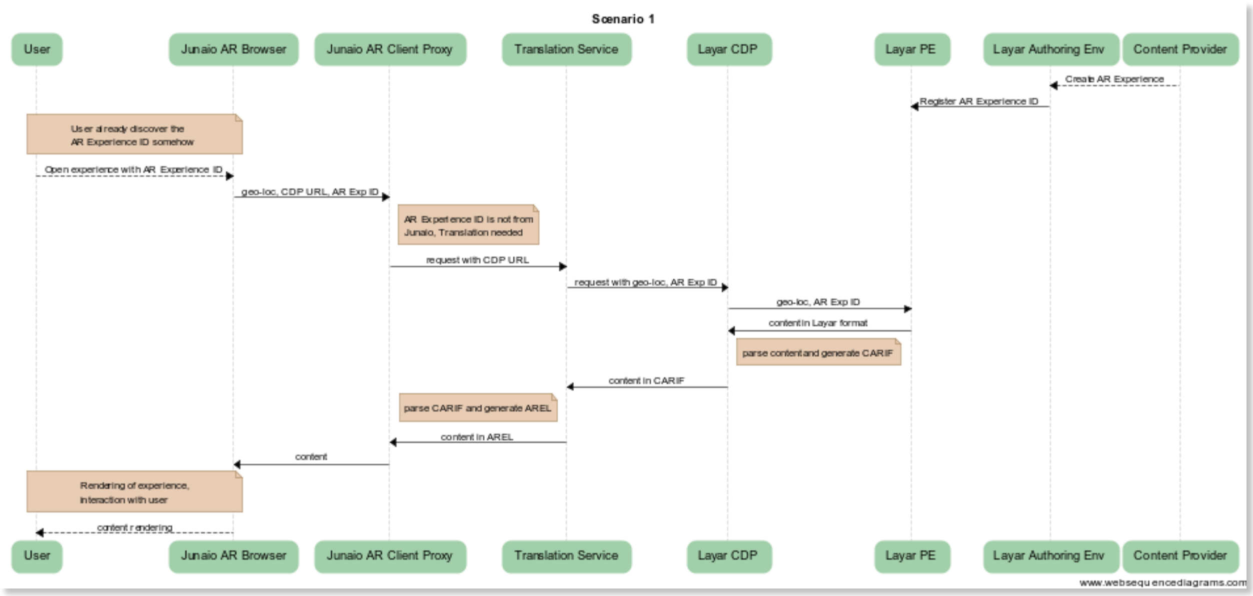
All these actions and properties can be statically configured and thus easily converted from one browser’s syntax into that of another. This ensures that initially there are little/no changes in the actual AR-Browser applications involved and the proof of interoperability can quickly be shown.

4. Scenarios

In order to illustrate how the architecture sketched above would work in various scenarios, we’ve made step-by-step

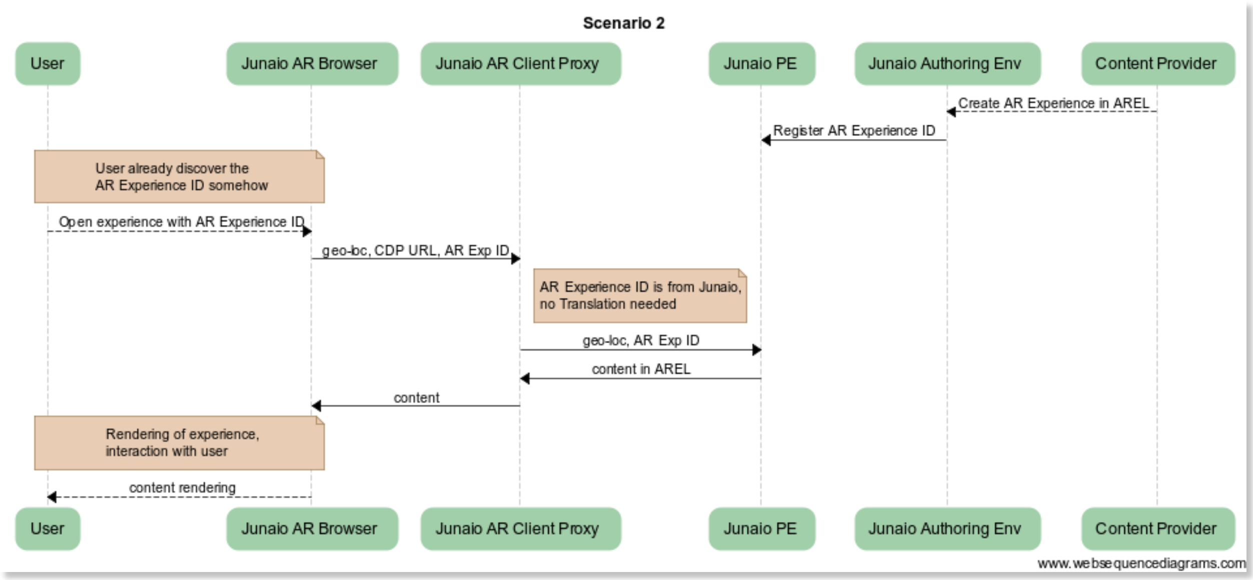
A. Scenario 1: junaio client with Layar layer

1. Content provider creates an AR Experience using the Layar Authoring Environment in the Layar format and registers an AR Experience ID with the Layar AR Publishing Environment.
2. User using the Junaio client wants to open that AR Experience ID hosted on Layar. Note: He already has discovered the AR Experience ID.
3. Junaio client sends request to junaio AR client proxy containing geo-location, CDP URL and AR experience ID
4. AR Client proxy determines it’s not a junaio link and invokes the translator service.
5. Translator service connects to the CDP from Layar (defined by the CDP URL)
6. Layar CDP queries the Layar PE for the content identified by the AR experience ID
7. Layar CDP parses Layar content and returns CARIF
8. junaio translator service parses CARIF and returns junaio native format to client through the AR client proxy.


Scenario 1
User
Junaio AR Browser
Junaio AR Client Proxy
Translation Service
Layar CDP
Layar PE
Layar Authoring Env
Content Provider
Create AR Experience
Register AR Experience ID
User already discover the AR Experience ID somehow
Open experience with AR Experience ID
geo-loc, CDP URL, AR Exp ID
AR Experience ID is not from Junaio, Translation needed
request with CDP URL
request with geo-loc, AR Exp ID
geo-loc, AR Exp ID
content in Layar format
parse content and generate CARIF
content in CARIF
parse CARIF and generate AREL
content in AREL
content
Rendering of experience, interaction with user
content rendering
www.websequencediagrams.com

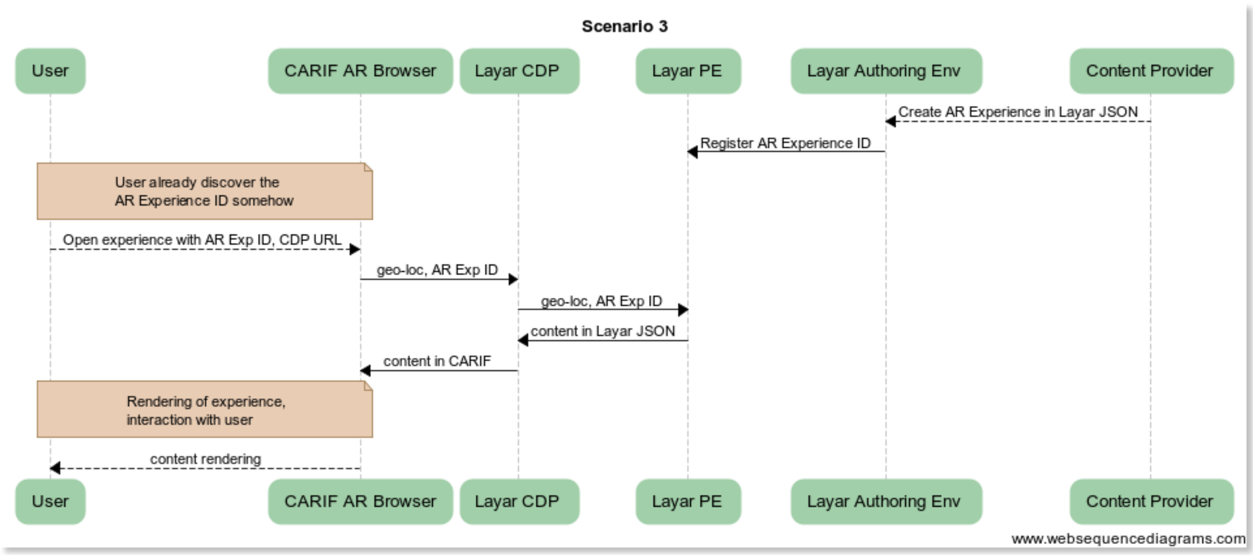
B. Scenario 2: junaio client with junaio channel

1. Content creator creates an junaio experience in the AREL format and registers an AR Experience ID with the junaio AR Publishing environment.
2. User using the Junaio client wants to open that AR Experience ID hosted on junaio. Note: He already has discovered the AR Experience ID.
3. Junaio client sends request to junaio AR client proxy containing geo-location, CDP URL and AR experience ID
4. AR Client proxy determines it is a junaio link and directly queries the junaio PE.
5. junaio client proxy returns native format to client.


Scenario 2
User
Junaio AR Browser
Junaio AR Client Proxy
Junaio PE
Junaio Authoring Env
Content Provider
Create AR Experience in AREL
Register AR Experience ID
User already discover the AR Experience ID somehow
Open experience with AR Experience ID
geo-loc, CDP URL, AR Exp ID
AR Experience ID is from Junaio, no Translation needed
geo-loc, AR Exp ID
content in AREL
content
Rendering of experience, interaction with user
content rendering
www.websequencediagrams.com

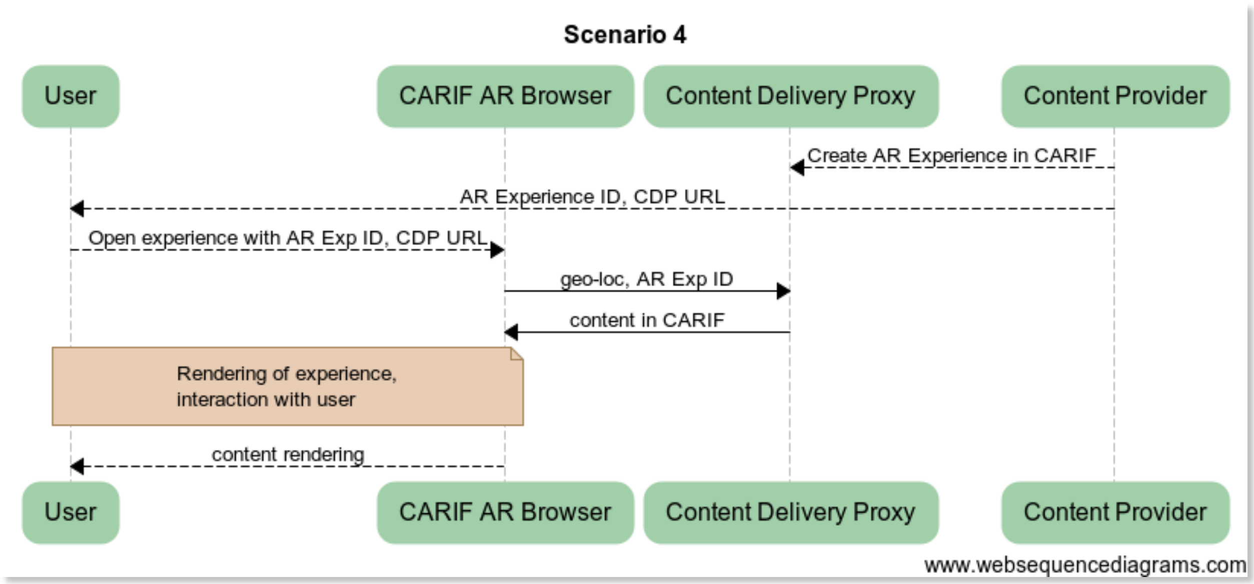
C. Scenario 3: Browser that understands CARIF, experience in Layar

1. Content creator creates an AR Experience in the Layar format and registers an AR Experience ID with the Layar AR Publishing environment.
2. User using the CARIF browser wants to open that AR Experience ID hosted on Layar. Note: He already has discovered the AR Experience ID.
3. CARIF browser sends request to Layar CDP URL and AR experience ID
4. Layar CDP queries the Layar PE for the content identified by the AR experience ID
5. Layar CDP parses Layar content and returns CARIF
6. CARIF browser can parse response directly.


Scenario 3
User
CARIF AR Browser
Layar CDP
Layar PE
Layar Authoring Env
Content Provider
Create AR Experience in Layar JSON
Register AR Experience ID
User already discover the AR Experience ID somehow
Open experience with AR Exp ID, CDP URL
geo-loc, AR Exp ID
geo-loc, AR Exp ID
content in Layar JSON
content in CARIF
Rendering of experience, interaction with user
content rendering
www.websequencediagrams.com

D. Scenario 4: Browser understands CARIF and experience also hosted in CARIF

1. Content creator creates an AR Experience in CARIF and “tells” the user to open it in his CARIF browser.
2. User using the CARIF browser wants to open that AR Experience ID hosted at the CDP URL.
3. CARIF browser sends request to CDP URL with AR experience ID
4. CDP returns content in CARIF directly
5. CARIF browser can parse response directly.


Scenario 4
User
CARIF AR Browser
Content Delivery Proxy
Content Provider
Create AR Experience in CARIF
AR Experience ID, CDP URL
Open experience with AR Exp ID, CDP URL
geo-loc, AR Exp ID
content in CARIF
Rendering of experience, interaction with user
content rendering
www.websequencediagrams.com

5. AR Launch URL Scheme

In order to be able to launch an AR experience that is not native to the user’s preferred AR Browser, there must be newly defined interoperability components implemented on all the AR (Publishing Environments) platforms, including proxies, and a common “AR launch URL scheme” is defined.

Protocol: ar:// or sar://

Parameters: AR Experience ID , CDP URL

```
Scheme =
protocol “://” CDP_URL “/” ExperienceID “/?requestedParams=” parameters “&”
userParameters

protocol            =     “ar” | “sar”
CDP_URL             =     URL of CDP
ExperienceID        =     ID of the experience
parameters          =     keyword | keyword “+” parameters
keyword             =     “pos” | “alt” | “hacc” | “vacc”
userParameters      =     userParameter | userParameter “&” userParameter
userParameter       =     paramName “=” paramValue
paramName           =     URL encoded string
paramValue          =     URL encoded string
```

Keyword description

The keywords tell the AR browser which information to send.

pos	Browser sends lat,lon in WGS84
alt	Browser sends altitude in meters
hacc	Browser sends horizontal accuracy in meters
vacc	Browser sends vertical accuracy in meters

Protocol description

protocol AR	use HTTP (default port 80)
protocol SAR	use HTTPS (default port 443)

Examples

```
ar://api.junaio.com/242454/?requestedParams=pos+alt&filter=food
sar://api.lay.ar/1234/?requestedParams=pos
```

With requestedParams:

```
sar://api.lay.ar/1234/?requestedParams=pos+hacc&filter=food
```

“open experience from api.lay.ar using HTTPs on port 443 with ID 1234 and add the device’s lat + lon + horizontal accuracy [m] to it”
This triggers a request to https://api.lay.ar/1234/?lat=42.5&lon=11.5&hacc=15&filter=food

Without requestedParams:

```
ar://api.car.if/3445/?myParam1=test
```

“open experience from api.car.if with ID 12345”
This triggers a request to http://api.car.if/12345/?myParam1=test

6. CDP request RESTful API

This is the request from the Translator Service to the Content Delivery Proxy (CDP).

API endpoint:

```
http[s]://[server]/experiences/[experienceID]/?
```

e.g.

```
http://api.car.if/experiences/12345/
```

GET parameters

Parameter	RequestedParam	Description	Example
lat	pos	Latitude [WGS84]	48.122101
lon	pos	Longitude [WGS84]	11.601563
hacc	hacc	Horizontal accuracy of lat,lon as estimated by the sensors [m] If lat, lon cannot be determined by the client, then the value of hacc should be set to -1 and lat, lon should still be set to valid values, whatever those might be (e.g. 0,0).	15
vacc	vacc	Accuracy of altitude [m]	30
alt	alt	Altitude [m]	519
userid	-	unique hash identifying the user, client-specific	

HTTP request headers (RFC2616):

Parameter	Description	Example
Accept-Language	ISO	en_US
User-Agent	user agent of the client, preceded by the proxy user agent	User-Agent: junaio-server/1.0 com.metaio.junaio/5.0 (iPhone5S/iOS7.0)
Via (?)	to be investigated	
X-Forwarded-For	http://en.wikipedia.org/wiki/X-Forwarded-For	X-Forwarded-For: 13.242.183.43, 82.135.125.82

7. Common AR Interchange Format (CARIF) for Basic AR Browser Interoperability

The participants of the AR Browser Interoperability Task Force need a Common AR Interchange Format that supports the minimum interoperability feature set and is extensible in the future. This work is not performed in a standards working group and the editors are not all members of a common Standards Development Organization or seeking to prove a particular draft or ratified standard.

At the proposal of the CARIF editor, the CARIF currently under examination is based on the draft ARML 2.0 specification (http://www.opengeospatial.org/projects/groups/arml2.0swg). An example of an encoding of a POI in CARIF in this experiment is outlined below.

```
<carif>
<arml>
  <ARElements>
    <Feature id=”a239sadflkjwt90qu”>
      <anchors>
        <Geometry>
          <gml:Point>
            <gml:pos>
              {longitude} {latitude} {altitude}
              #e.g. 12.3456 45.678 -4
            </gml:pos>
          </gml:Point>
        </Geometry>
      </anchors>
      <name>
        This is the title
      </name>
      <description>
        <![CDATA[ A very long description of this thing]]>
      </description>
      <metadata>
        <actions>
          <action mime-type=“text/html” label=“Open webpage”>
            <uri>
              http://www.wikipedia.org
            </uri>
          </action>
        <action mime-type=“audio/mp3” label=”Play sound”>
          <uri>
            http://www.soundcloud.com/djsam/dkeij.mp3
          </uri>
        </action>
        <action mime-type="video/mp4" label="Play video">
          <uri>
            http://www.vimeo.com/download/?id=di4i4353463
          </uri>
        </action>
        <action mime-type="application/send-sms|send-email|call">
          <uri>
            tel:+4422394460 |
            sms:+31492014945 |
            mailto:foo@example.com?
              cc=bar@example.com&subject=Greetings%20
              from%20Cupertino!&body=Wish%20you%20were%20here!
          </uri>
        </action>
        <action mime-type="application/route-to">
          <uri>
            route:daddr=5.342435,45.23421143
          </uri>
        </action>
      </actions>

        <image href="http://mybucket.s3.amazonaws.com/anIcon.png" />
      </metadata>
    </Feature>
  </ARElements>
</arml>

  <ARExperience>
    <name>Title of the Experience</name>
    <description><![CDATA[ A description here]]></description>
    <metaData>
      <icon href="http://some.doma.in/myIcon.png" />
      <author>James Doe</author>
      <webpage href="http://something" />
    </metaData>
  </ARExperience>
</carif>
```

The following tags are used:

tag name	**description**	**example and valid values**	**mandatory**
`arml`	The root tag for the XML document		Y
`ARElements`	The container for POIs		Y
`Feature`	The container for a single POI. Optionally contains an id attribute with an ID uniquely identifying the POI in the document		Y
`anchors / Geometry / gml:Point`	An extensible parent-child relationship of tags, supporting other anchor-types (like CV-based anchors) in the future		Y
`gml:pos`	The position of the POI, in a blank-separated list of decimal numbers defining latitude, longitude (WGS84) and optionally altitude (in meters)	12.345 34.567 340 12.345 34.567	Y
`name`	The name (title) of the POI		N
`description`	A description on the POI		N
`metadata`	The container for any metadata for the POI		N
`actions`	The container for any action (call, route to etc.) defined for a POI.		N
`action`	The definition of a single action. An action has two attributes: • mime-type (required) Defines the type of the action. Must be set to one of the following: o `text/html` for opening a webpage o `audio/mp3` (or any other audio-format) to play an audio file o `video/mp4` (or any other video-format) to play a video file o `application/send-sms` to send an SMS o `application/send-email` to		N

	send an email o `application/call` to initiate a phone call o `application/route-to` to route to the POI from the current location. • label (optional): Defines a text which is displayed along with the button to trigger the action in the browser.		
`uri`	The URI defining the action. Depending on the mime-type of the action, the URI must comply with the following schemes: • A web URL for `text/html, audio and video` actions • `tel:+4422394460` for `application/call` actions • `sms:+31492014945` for `application/sms` actions • `mailto:<email>?cc=<email>&subject=<subject>&body=<body>` for `application/send-email` actions • `route:daddr=5.342435,45.234211` for `application/route-to` actions		Y
`image`	Contains a mandatory `href` parameter which points to a thumbnail that represents the POI.		N
`ARExperience`	The container for metadata concerning the entire experience		N
`name`	The name (title) of the experience		N
`description`	A description on the experience		N
`metadata`	The container for any additional metadata for the experience		N
`icon`	Contains a mandatory `href` parameter that points to an icon for the experience.		N
`author`	The author's name of the experience		N
`webpage`	Contains a mandatory `href` parameter that points to a webpage of the experience.		N

8. Conclusion

In this document we define an AR Browser Interoperability architecture allowing AR Browsers from various vendors to load content that was originally authored to work with a different AR Browser. We've defined a basic feature set that can be supported using this framework, specified an AR Launch URL that would be supported by all interoperable AR Browsers, allowing content authors to "advertise" their experiences on the web and specified an API and format for fetching content for an AR Experience from one AR Browser vendor platform to another.