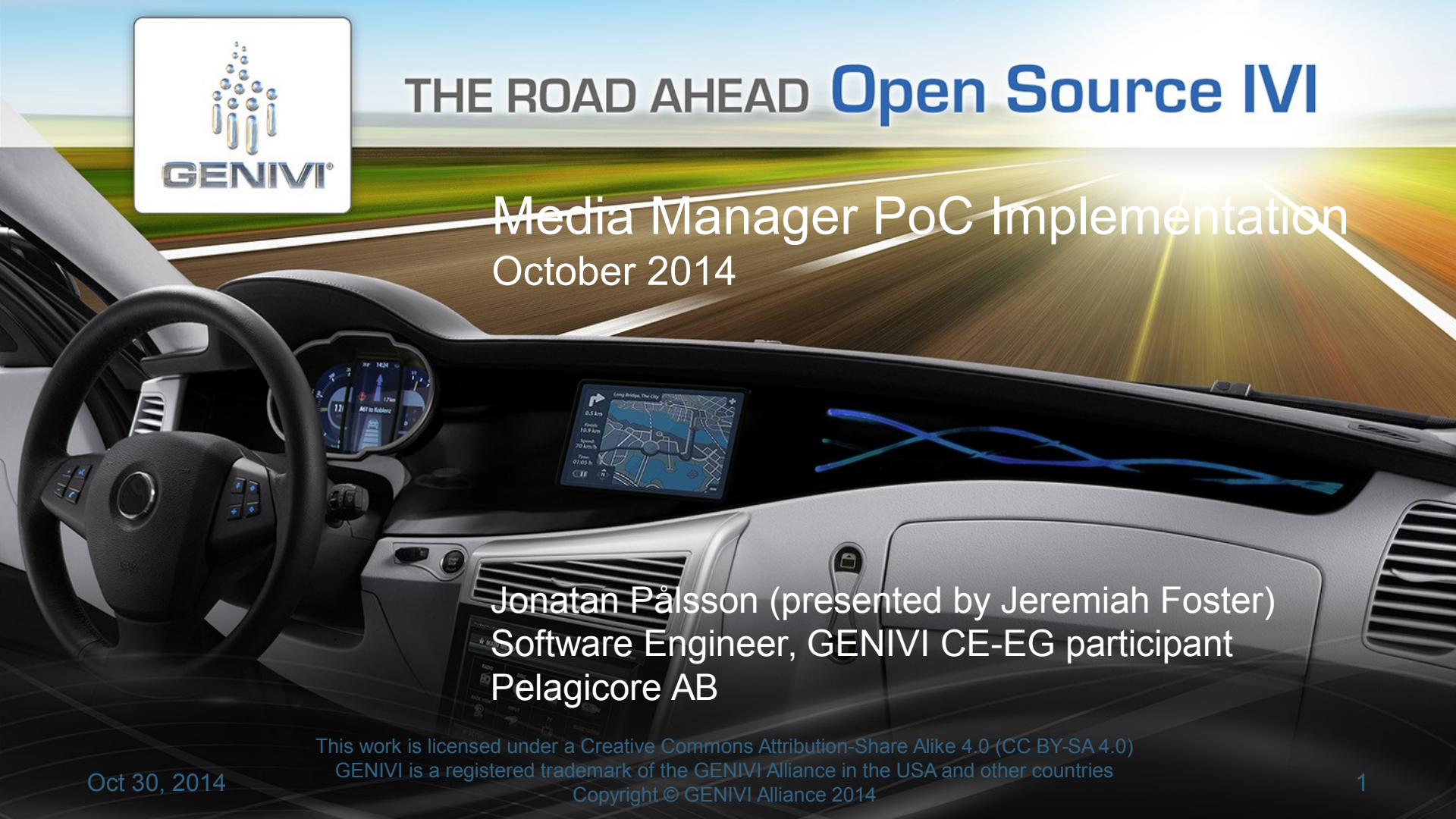


Media Manager PoC Implementation

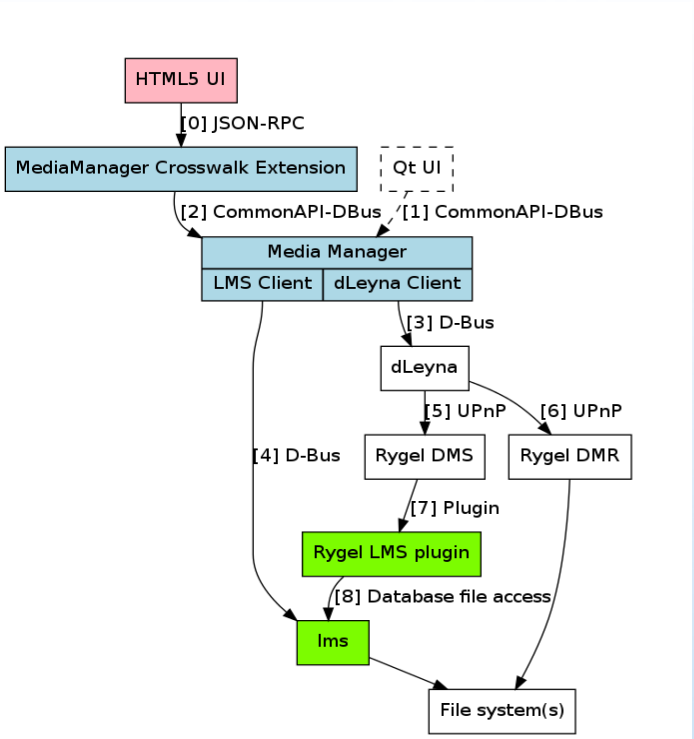
October 2014

Jonatan Pålsson (presented by Jeremiah Foster)
Software Engineer, GENIVI CE-EG participant
Pelagicore AB

Purpose of the PoC

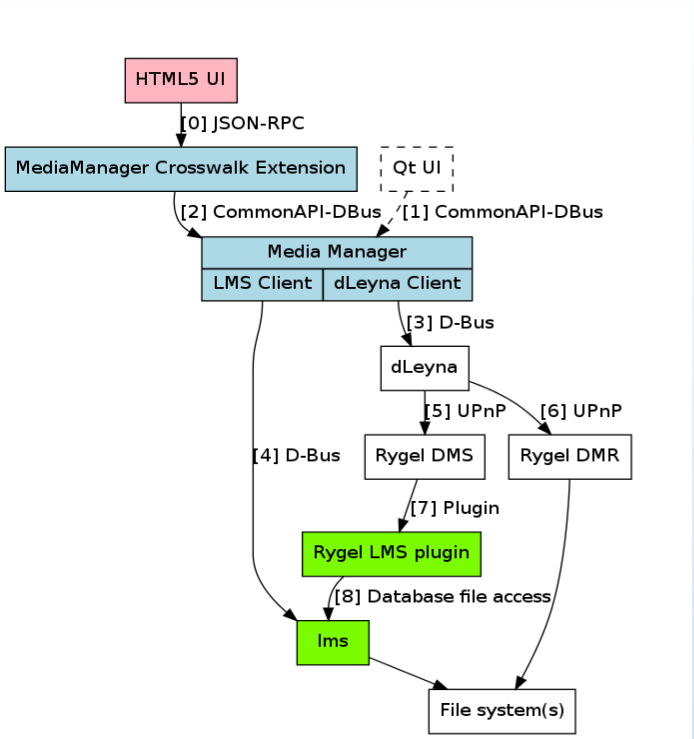
- Prove the APIs defined by the CE-EG
- Are certain HMI flows difficult to implement?
- Are we compatible with multiple different kinds of devices?
- Does the API allow an efficient implementation?
- Provide a hands-on experience of the project

Architecture



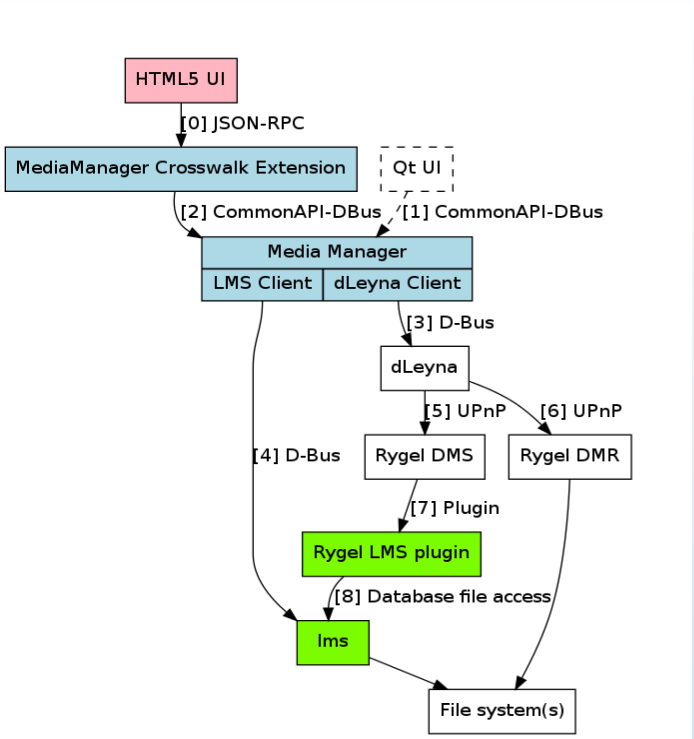
- HTML5 UI is a CrossWalk application
- CrossWalk extension is CommonAPI client, connecting UI to Media Manager
- Qt UI is a theoretical future extension, used instead of HTML5 UI
- Media Manager is the core component of the project, connecting all other components
- dLeyna is a DLNA client
- Rygel DMS is the service used for browsing
- Rygel DMR is the service used for rendering
- Rygel LMS plugin is extra logic for LMS interaction
- LMS is the media indexer
- File system is any mounted file system

Architecture



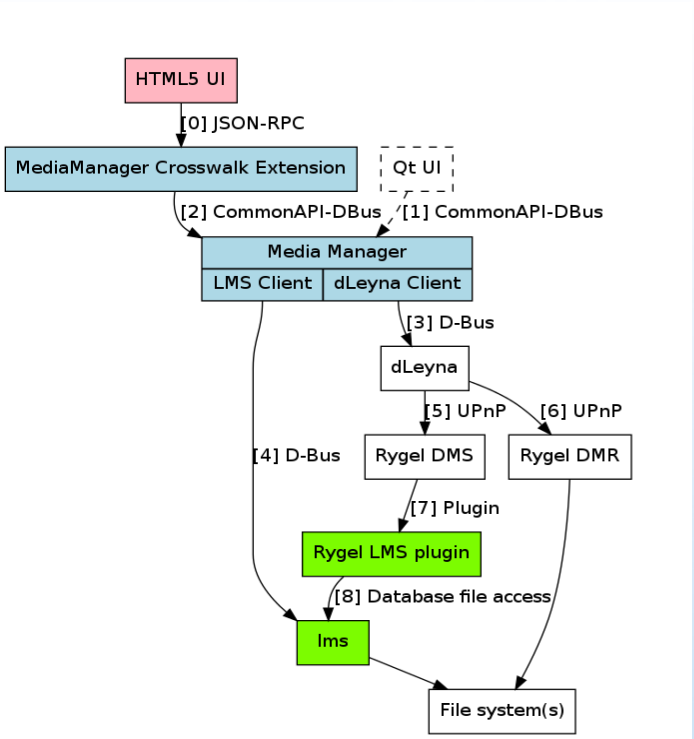
- HTML5 UI is sandboxed, system interaction via Extension
- IPC between HTML5 UI and Extension is JSON-RPC
  - No pre-defined IPC for CrossWalk extensions

Architecture



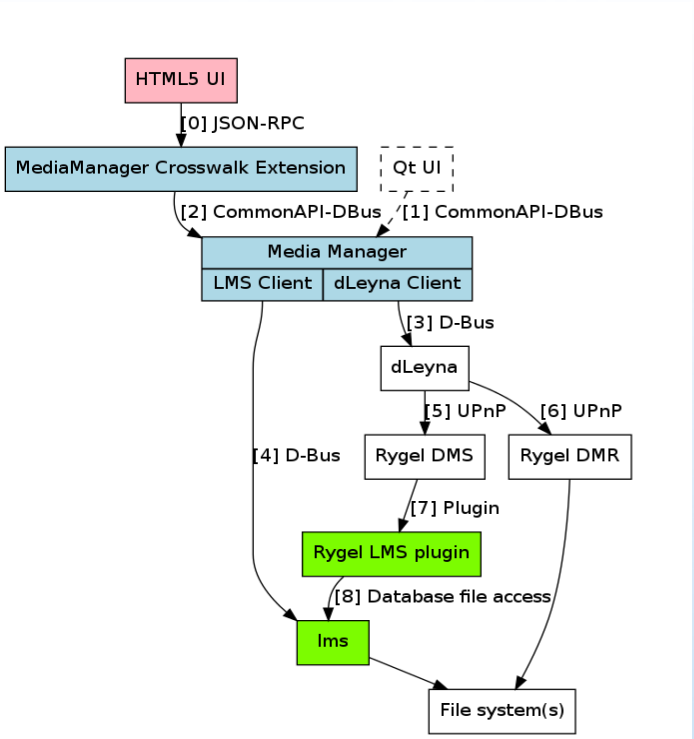
- MM CrossWalk Extension is a CommonAPI client
- Contains no application logic
- Could/should be replaced by auto-generated code from CommonAPI introspection
- Only used for CrossWalk, other UI:s would need new clients, Qt for instance

Architecture



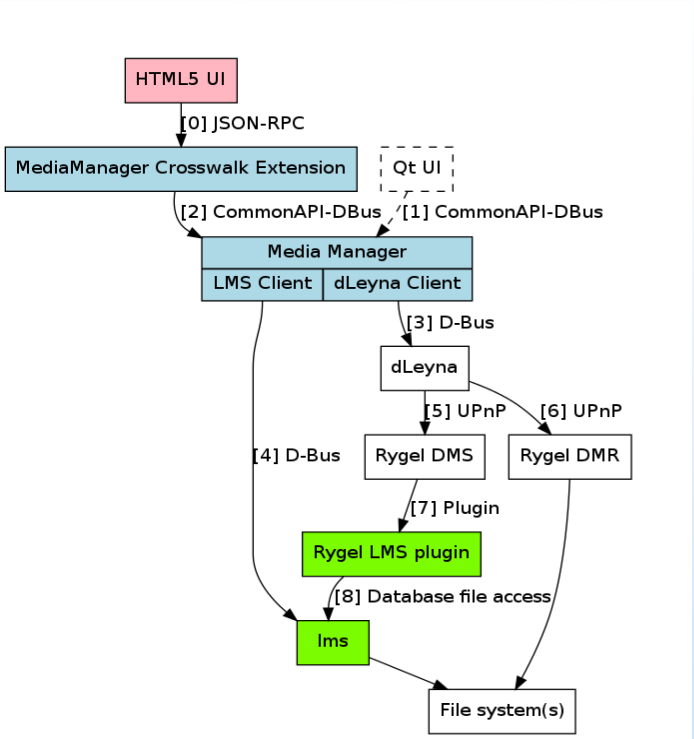
- MediaManager is a CommonAPI server
- Contains logic for media manager:
  - Current play queue
    - Ways to manipulate the play queue
  - Current play state
  - Connections to LMS, dLeyna and the HMI via the CrossWalk extension

Architecture



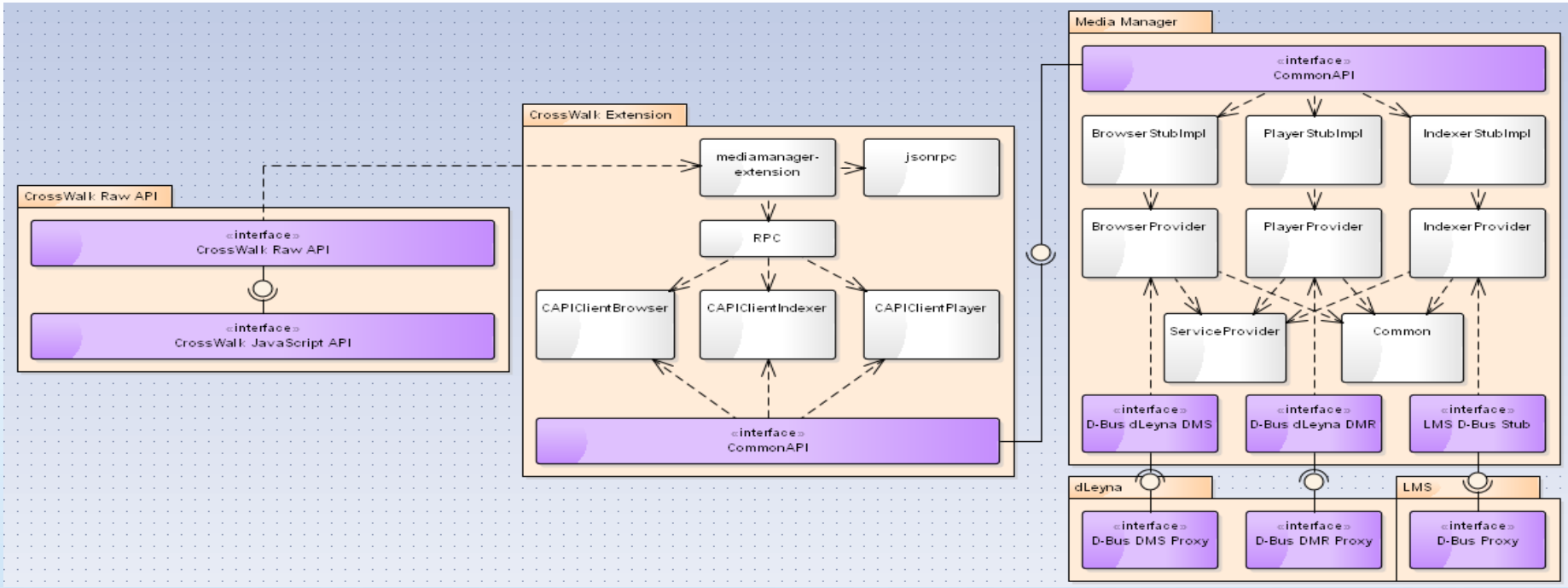
- dLeyna is a DLNA client
  - Interfaces with Rygel DMS and DMR
  - Extended with extra functionality for handling artists and album art in containers
- Rygel is a DLNA server
  - Provides browsing via DMS
    - Extended with artists and album art in containers
  - Provides rendering via DMR

Architecture

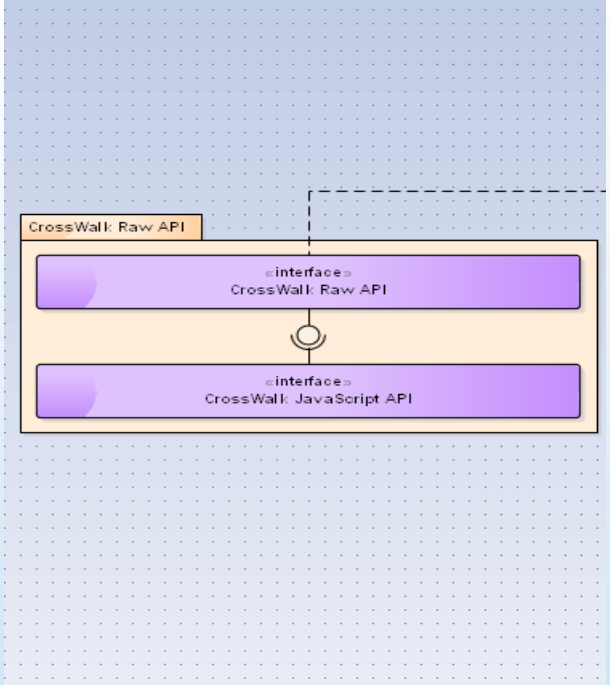


- Rygel LMS plugin allows Rygel to query the LMS database
- LMS is the Light Media Scanner which is used to extract metadata from media files
- The plugin runs queries against the LMS database
- The plugin reacts to signals (such as database being updated) sent by the LMS daemon

Internal architecture - Overview

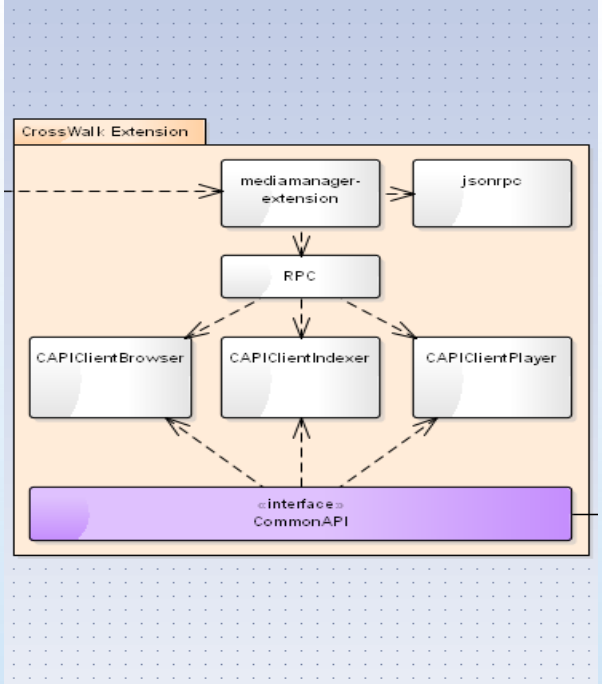


Internal architecture - CrossWalk



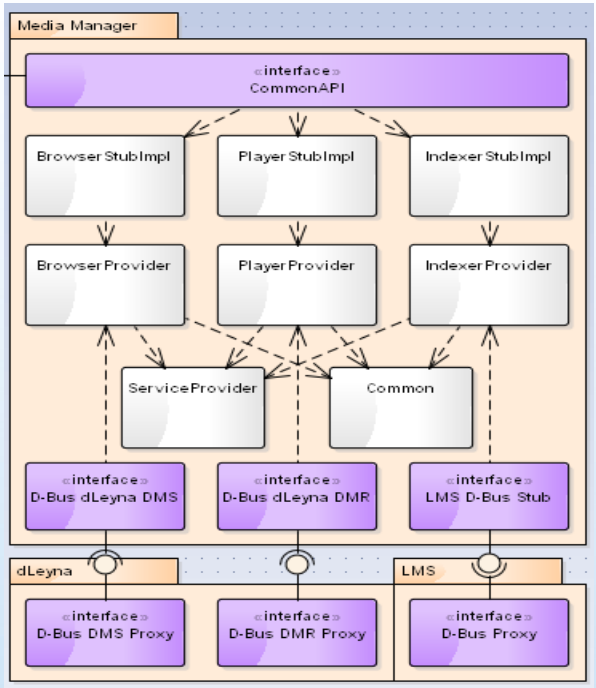
- CrossWalk “raw” async API carries JSON-RPC messages
- API presented as regular JavaScript functions in CrossWalk
  - Examples:
    Player.play()
    Player.pause()
    Player.openPlaylist()
    Browser.listContainers()

Internal architecture - Extension



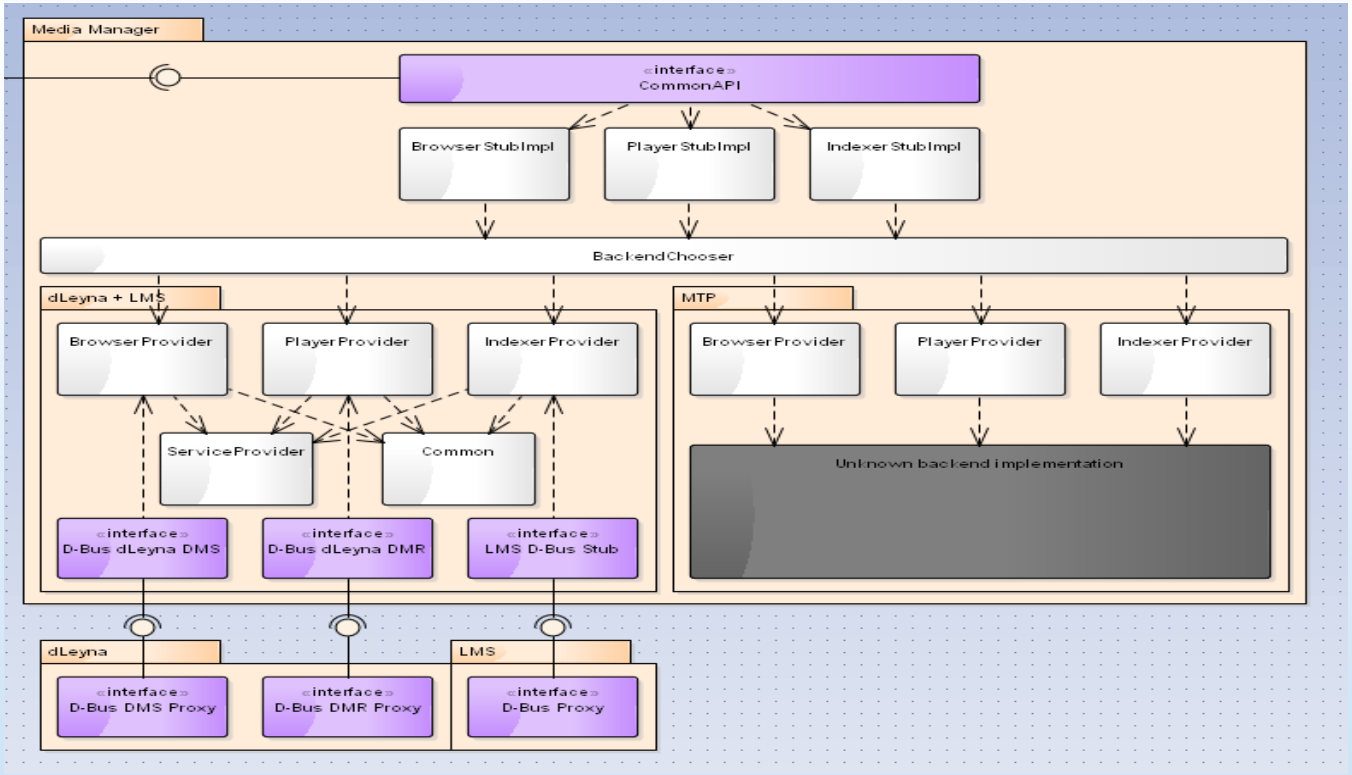
- mediamanager-extension uses the CrossWalk library to connect to the “raw” CrossWalk API
- Messages are converted to JSON-RPC using the jsonrpc module
- The RPC module connects JSON-RPC functions to the corresponding C-functions
- CAPIClientBrowser, Indexer and Player are CommonAPI stubs and contain logic for contacting the Media Manager using CommonAPI
- The CommonAPI block represents the CommonAPI libraries

Internal architecture – Media Manager



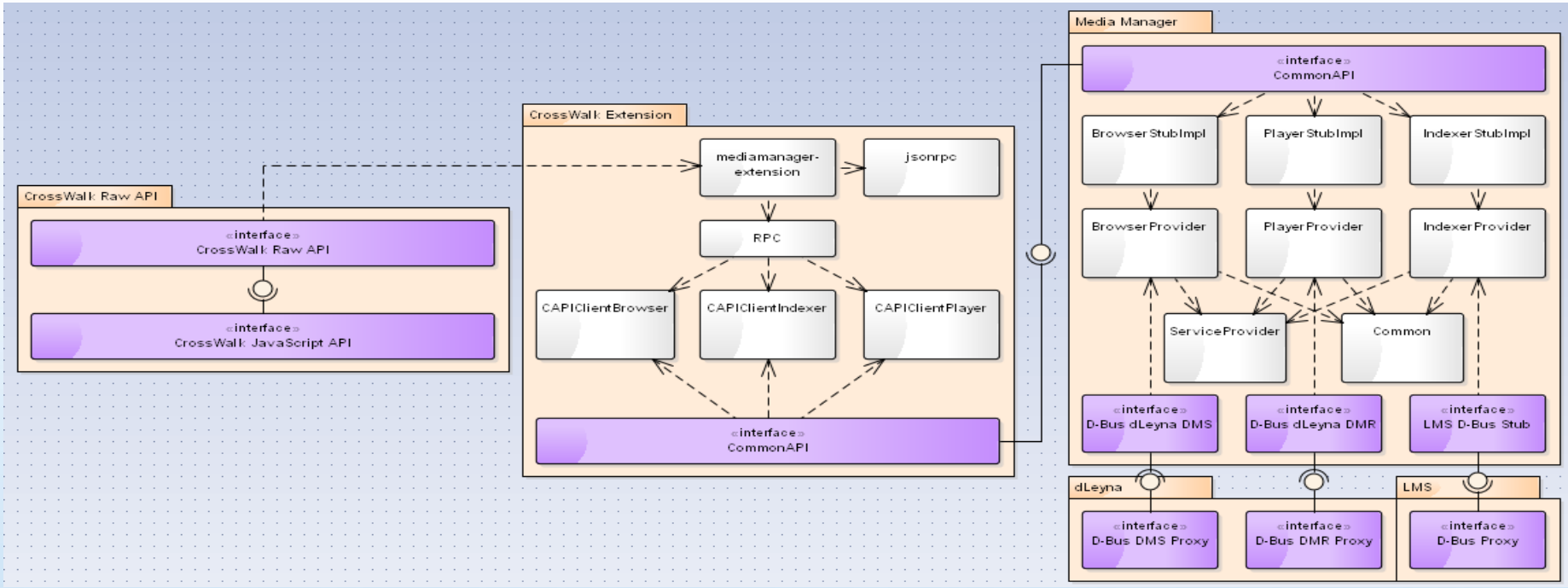
- CommonAPI block indicates CommonAPI library
- *StubImpl blocks indicate a conversion layer between the CommonAPI domain and the internal Media Manager domain
- The *Provider layer augments the functionality of the provided services
- The ServiceProvider provides a common base class for all providers
- Common contains some shared utility code
- The remaining interfaces represent D-Bus communication

Internal architecture – Future Media Manager



- Future example
- Backend Chooser added
- MTP added
- Further backends possible
- Not implemented in PoC

Internal architecture - Recap



Media Manager in Open Source

- Several open source projects are used in the Media Manager
- The most apparent ones are:
- dLeyna
- Rygel
- LMS
- Other components were used on lower levels as dependencies, for instance the GUPnP and libmediaart
- Several modifications have been made to the projects used
- Relevant modifications to dLeyna and Rygel have been sent for review and upstreaming to the respective projects
- Modifications to LMS are deemed to be out-of-scope for the upstream project, and will not be upstreamed

Building the Media Manager

- Possible to build for Debian and Tizen (probably more distributions as well)
- Both Debian and Tizen require CrossWalk in the system
- meta-ivi support is in the works
- Requires patched versions of Rygel, dLeyna and LMS
- Everything is built using Cmake
- Code for Media Manager: http://git.projects.genivi.org/ (All repos starting with media-manager)
- Code for patched Rygel: https://github.com/Pelagicore/Media-Manager-Rygel-patched/
- Code for patched dLeyna: https://github.com/Pelagicore/Media-Manager-dLeyna-patched
- Code for patched LMS: https://github.com/Pelagicore/Media-Manager-lightmediascanner-patched

Contributing to the project

- Discuss on the mailing list
- Send patches on the mailing list
- For use-cases and requirements
- Join the GENIVI CE-EG weekly telcos
- For HMI development
- Discuss on the AGL mailing lists

Jonatan Pålsson (presented by Jeremiah Foster)
Software Engineer, GENIVI CE-EG participant
Pelagicore AB

Purpose of the PoC

- Prove the APIs defined by the CE-EG
- Are certain HMI flows difficult to implement?
- Are we compatible with multiple different kinds of devices?
- Does the API allow an efficient implementation?
- Provide a hands-on experience of the project

POINT1:

The API for the Media Manager was defined in the Consumer Electronics and Connectivity work group within GENIVI. The work group chose to base the API on existing APIs used by open source projects.

There are three distinct parts to the Media Manager, which use three different APIs found in other open source projects. The three different parts of the Media Manager are; Browsing, Playback and Indexing.

For the browser API, the MediaServer2 API, specified by GNOME for the Rygel project, was used. This API is based around containers and items. It works much like a file system, where you can place containers, or items within containers, and an item can not have sub-items or sub-containers. This creates a tree structure. Furthermore, the MediaServer2 API allows searching and filtering.

For playback, we use the MPRIS2 API, which is used by, for example the VideoLan Client (VLC) media player, and the XMMS audio player. This API defines basic actions such as Play, Pause, Next and Stop. The API was extended with playlist functionality, such as replacing the current play list, removing an item from the list and replacing the list with the contents of a container.

For Indexing, we adopted the API used by LMS, which contains stopping, starting and some event mechanism to indicate that the database has been updated.

POINT1.1:

An HMI originally used for a different media player in Tizen was used to implement some HMI flows and to prove that the API can be used by a somewhat arbitrarily chosen HMI.

POINT1.2:

For the PoC, we have investigated that USB sticks and mass storage in general can be indexed and supported by the system. There are ongoing discussions in the expert group on verifying the API works with other kinds of devices, such as mobile phones and media players using special protocols for media transfer and control.

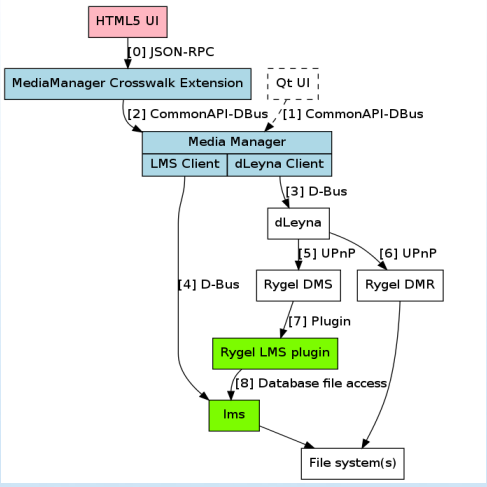
POINT1.3:

The purpose of the PoC was not to achieve high efficiency, but there have been efficiency considerations in terms of limiting the amount of data being sent and processed. For instance the offset and count mechanism in the browsing API, as well and the filtering mechanism of the browsing API allows the API consumer to tune the amount of data being sent.

POINT2:

Finally, a major goal of the PoC is to provide a hands-on experience of the use-cases and specifications crafted by the CE-EG. By constructing a PoC, some wording of the use-cases was updated, thereby clarifying the original intent of the use case, and in general, a concrete implementation of the use cases was achieved.

Architecture



- HTML5 UI is a CrossWalk application
- CrossWalk extension is CommonAPI client, connecting UI to Media Manager
- Qt UI is a theoretical future extension, used instead of HTML5 UI
- Media Manager is the core component of the project, connecting all other components
- dLeyna is a DLNA client
- Rygel DMS is the service used for browsing
- Rygel DMR is the service used for rendering
- Rygel LMS plugin is extra logic for LMS interaction
- LMS is the media indexer
- File system is any mounted file system

The figure on the left will be used in the next coming slides to give a big picture of the Media Manager component when interacting with the rest of the components used in the project.

In this slide, I'll just cover some terminology so that the following slides can be understood more easily.

1: HTML 5 UI is a CrossWalk application. This was developed for JLR and has been seen in previous demos of the AGL platform. Some functionality was added to the UI, such as enqueuing tracks

2: The CrossWalk extension is the CommonAPI client, which connects to the Media Manager CommonAPI server. This was developed in its entirety for this project

3: The Qt UI is a theoretical future extension. It was added to the diagram to show the level to integrate a new UI. There are no plans within GENIVI to develop this at this point.

4: The Media manager is the core component of the project, this component connects all the other components and facilitates communication between them. This component was also developed in its entirety for this project.

5: dLeyna is a DLNA client developed by Intel. It is used to communicate with the DLNA services exposed by the DLNA server.

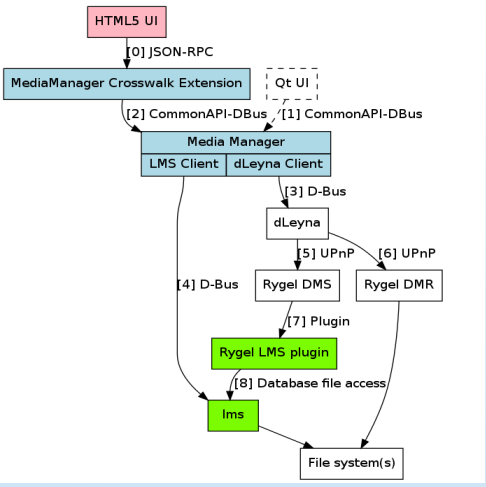
6/7: Rygel is the DLNA server used in the project. It provides services for browsing, callled Digital Media Server (DMS), and for rendering called Digital Media Renderer (DMS).

8: The Rygel LMS plugin provides extra logic for Rygel, allowing it to interact with the Light Media Scanner.

9: The Light Media Scanner is used to extract and store metadata contained in media files, for the PoC we support MP3 files.

10: File system means any mounted file system. Focus in the PoC has been USB sticks.

Architecture



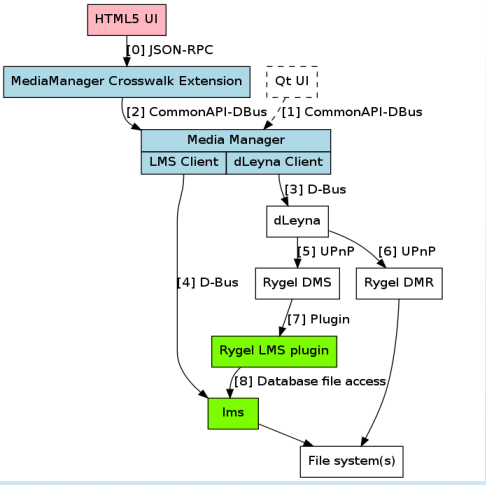
- HTML5 UI is sandboxed, system interaction via Extension
- IPC between HTML5 UI and Extension is JSON-RPC
  - No pre-defined IPC for CrossWalk extensions

The HTML5 UI, developed by JLR, is a CrossWalk application running within a sandboxed CrossWalk environment. CrossWalk is a web-runtime which allows HTML web pages to be packaged as apps, and via specially crafted extensions allow these web pages to access system features.

The HTML5 UI in this PoC is connected to the Media Manager via the Media Manager crosswalk extension. The connection between the UI and the extension is via JSON-RPC, which provides an asynchronous communication channel between the extension and UI.

There is no pre-defined IPC for CrossWalk, but rather one synchronous and one asynchronous “raw” communciation channel are available. Using these channels, any IPC can be implemented on top. JSON is easily constructed from JavaScript, and support for JSON parsing is available in CrossWalk and there are efficient C libraries for JSON, so JSON-RPC was chosen as IPC.

Architecture



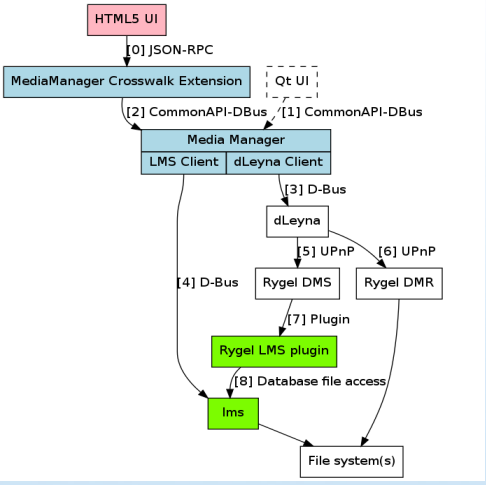
- MM CrossWalk Extension is a CommonAPI client
- Contains no application logic
- Could/should be replaced by auto-generated code from CommonAPI introspection
- Only used for CrossWalk, other UI:s would need new clients, Qt for instance

The Media Manager crosswalk extension is a CommonAPI client, which interfaces with the Media Manager core component. This component is only necessary for CrossWalk Extensions, and for other HMIs, such as Qt, the CommonAPI clients could be designed in different ways, such as integrated in to the HMI logic.

The extension itself contains little or no logic besides the logic necessary for the IPC. This essentially means that this component could be automatically generated from the CommonAPI code, but there is not yet a CommonAPI to JavaScript/CrossWalk extension generator available.

There have been some discussions within the Tizen team to further develop a D-Bus to JavaScript bridge, which could be used instead of this component, but some convenience wrapping from CommonAPI (mostly naming) would then be lost.

Architecture



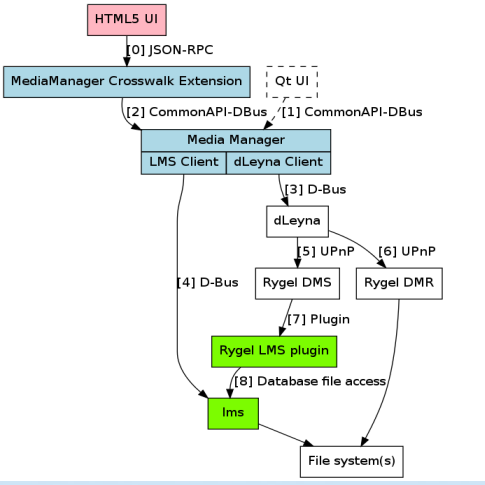
- MediaManager is a CommonAPI server
- Contains logic for media manager:
  - Current play queue
    - Ways to manipulate the play queue
  - Current play state
  - Connections to LMS, dLeyna and the HMI via the CrossWalk extension

The media manager component is the core component in the project. This process connects all the other subsystems, such as LMS and Rygel to the HMI. Most of the functionality in the Media Manager project has been delegated to Rygel or LMS, but some missing features, such as playlist management has been added to the Media Manager component in order to augment the functionality of LMS and Rygel.

The component contains logic for creating and manipulating the current play queue, such as adding, removing and replacing the play queue with the contents of a container. It also keeps track of the current play state, and keeps open connections to LMS, dLeyna and the HMI via the CrossWalk extension.

The functionality of this component is exposed via its CommonAPI interface, which in this PoC uses D-Bus for transport, and can be used by any CommonAPI-Dbus client. As mentioned previously, the API exposed is MPRIS2 with minor additioions for playback, MediaServer2 for browsing, and the LMS interface for indexing.

Architecture



- dLeyna is a DLNA client
  - Interfaces with Rygel DMS and DMR
  - Extended with extra functionality for handling artists and album art in containers
- Rygel is a DLNA server
  - Provides browsing via DMS
    - Extended with artists and album art in containers
  - Provides rendering via DMR

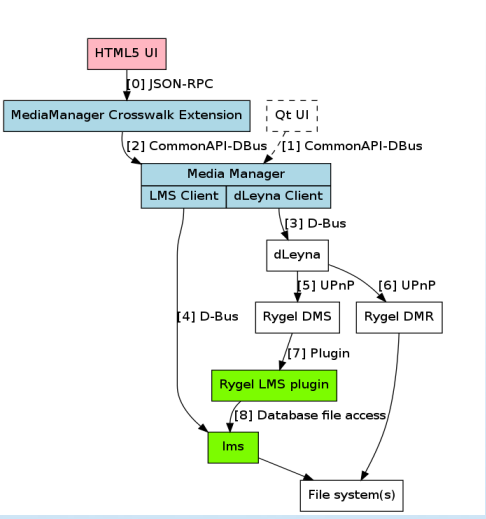
dLeyna is a DLNA client, which is used to interface with the Rygel UPnP server. dLeyna has a concept of connectors, where a connector is used to connect a DLNA consumer (in our case, the Media Manager) to dLeyna, which in turn will connect to Rygel. The connector used by the Media Manager is a D-Bus connector, which means the Media Manager connects to dLeyna over D-Bus, but this could be replaced by some in-process connector for increased efficiency.

dLeyna was mostly used as-is, but functionality for handing Artist and AlbumArtURL fields in containers was added.

Rygel was also mostly used as-is, with the exception of adding AlbumArtURL and Artist to containers. By adding this functionality, we can use containers to directly represent albums when browsing the DLNA server. Rygel already had support for these fields on an item level, but not on the more general object level, which is the common base class for both items and containers.

Rygel is used for both media browsing, via the Digital Media Server service, and rendering via the Digital Media Renderer service. The use-cases for the Media Manager project assumes a specific structure of the containers provided by the DMS, but if these use-cases are disregarded, a different DLNA server could be used instead of the local DMS associated with the Media Manager. Coupled with the Digital Media Renderer, which allows rendering any file (even remote files, since it can handle URIs), we can achieve remote playback and browsing. For the Media Manager, this has however been disbled, since the existence of multiple media sources and renderers creates additional requirements on the HMI in terms of selecting these services.

Architecture



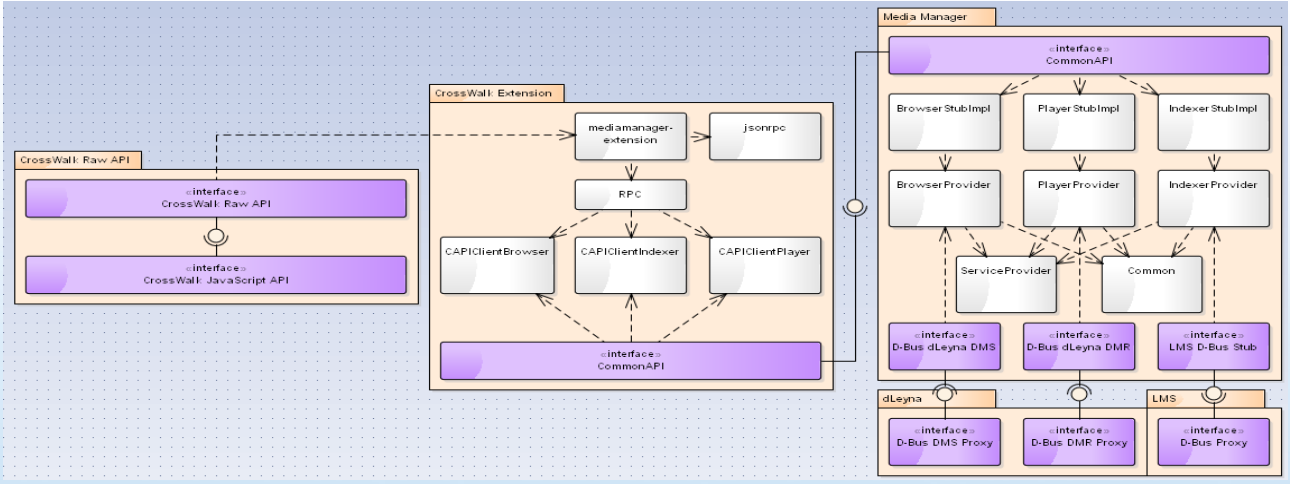
- Rygel LMS plugin allows Rygel to query the LMS database
- LMS is the Light Media Scanner which is used to extract metadata from media files
- The plugin runs queries against the LMS database
- The plugin reacts to signals (such as database being updated) sent by the LMS daemon

The Rygel LMS plugin was initially developed by Intel, and is already used by Rygel in Tizen. It is used to communicate between Rygel and the LMS database. Essentially, the LMS plugin transforms Rygel commands to SQL queries, and transforms the result back to Rygel containers and items.

The plugin was heavily modified to add functionality relating to search, and to change the structure of the data presented by the plugin. The output of the plugin directly corresponds to the structure of the media presented in the HTML5 UI.

The plugin also reacts to signals sent by LMS, most notably the signals sent when the database is updated.

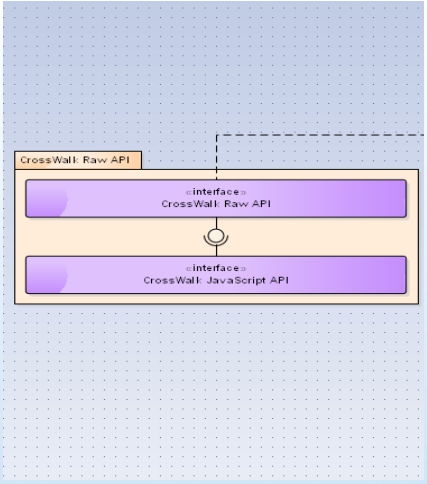
Internal architecture - Overview



This slide is meant to provide an overview of the internal architecture of the Media Manager. This diagram will be broken down in the following slides, from left to right. First I would like to explain some notation used in the diagram:

- The four big boxes represent different subsystems
- The purple boxes represent interfaces between components or Systems. This some form of IPC, such as D-BUS, or some communications protocol between two components, such as JSON-RPC.
- The Assembly icon (point to the ball and socket figure) indicates communication between interfaces.
- The dashed arrow indicates a dependency
- The gray boxes indicates distinct modules, such as classes

Internal architecture - CrossWalk



- CrossWalk “raw” async API carries JSON-RPC messages
- API presented as regular JavaScript functions in CrossWalk
  - Examples:
    Player.play()
    Player.pause()
    Player.openPlaylist()
    Browser.listContainers()

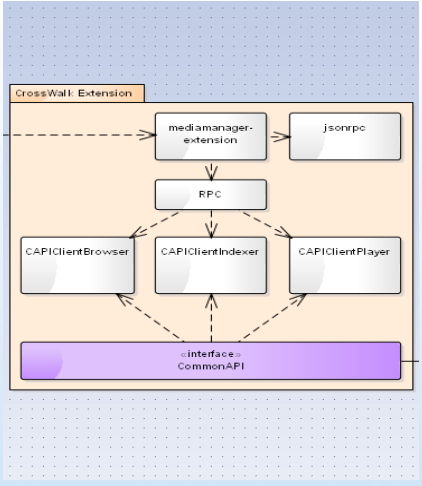
Let's first zoom in on the leftmost subsystem. This is represented in the diagram to show where the Media Manager connects to CrossWalk. The interface depicted in the graph here is a “raw” asynchronous API provided by crosswalk.

On top of the CrossWalk API, some RPC needs to be implemented by the application developer, and here we chose JSON-RPC. The JSON-RPC calls look like “regular” JavaScript calls to the users of the JavaScript API.

Some examples can be seen in the slide. The Media Manager JavaScript API exposes three top-level objects; the Browser, the Player and the Indexer. Using these objects the HMI developer can access all the functionality of the Media Manager. The first example shows how playback is started. The second shows pausing the third example shows the function used to replace the current play queue with the contents of a container, and the final example shows how to list containers.

All of these functions are asynchronous. The user of the API supplies a callback function with two parameters, one parameter for a success value, such as the containers from the listContainers call, and an error parameter used in case something goes wrong with the call.

Internal architecture - Extension



- mediamanager-extension uses the CrossWalk library to connect to the “raw” CrossWalk API
- Messages are converted to JSON-RPC using the jsonrpc module
- The RPC module connects JSON-RPC functions to the corresponding C-functions
- CAPIClientBrowser, Indexer and Player are CommonAPI stubs and contain logic for contacting the Media Manager using CommonAPI
- The CommonAPI block represents the CommonAPI libraries

The mediamanager-extension module is the connection point between CrossWalk and the extension subsystem.

The mediamanager-extension module uses a jsonrpc module to parse and create JSON-RPC messages.

The RPC module connects JSON-RPC functions to the corresponding C-functions in the Browser, Indexer and Player modules.

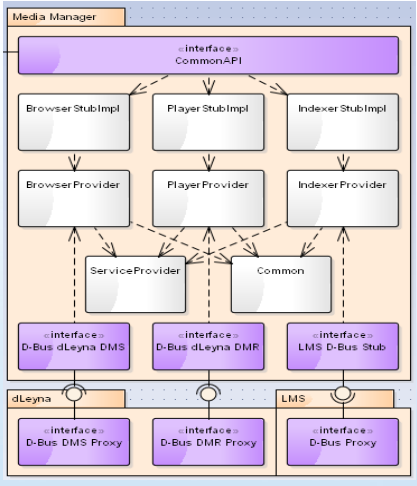
The CAPIClientBrowser, CAPIClientIndexer and CAPIClientPlayer modules are CommonAPI stubs and contain logic for contacting the Media Manager using CommonAPI.

The CommonAPI block represents the CommonAPI libraries.

Previously, it was mentioned that this subsystem could have been automatically generated. It now becomes apparent what must be generated if this is to be done. The RPC mechanism needs to be replaced or auto-generated, and then this new mechanism needs to be connected to the auto-generated CommonAPI stubs.

This auto-generated code could either have the form of an extension and would then look much like this extension, or it could all be done using a generic extension and move more of the logic in to the JavaScript client.

Internal architecture – Media Manager



- CommonAPI block indicates CommonAPI library
- *StubImpl blocks indicate a conversion layer between the CommonAPI domain and the internal Media Manager domain
- The *Provider layer augments the functionality of the provided services
- The ServiceProvider provides a common base class for all providers
- Common contains some shared utility code
- The remaining interfaces represent D-Bus communication

Again, the purple CommonAPI block indicates the CommonAPI library.

The BrowserStubImpl, PlayerStubImpl and IndexerStubImpl modules provide a conversion layer between the CommonAPI domain and the Media manager domain, converting CommonAPI data types to internal data types, etc.

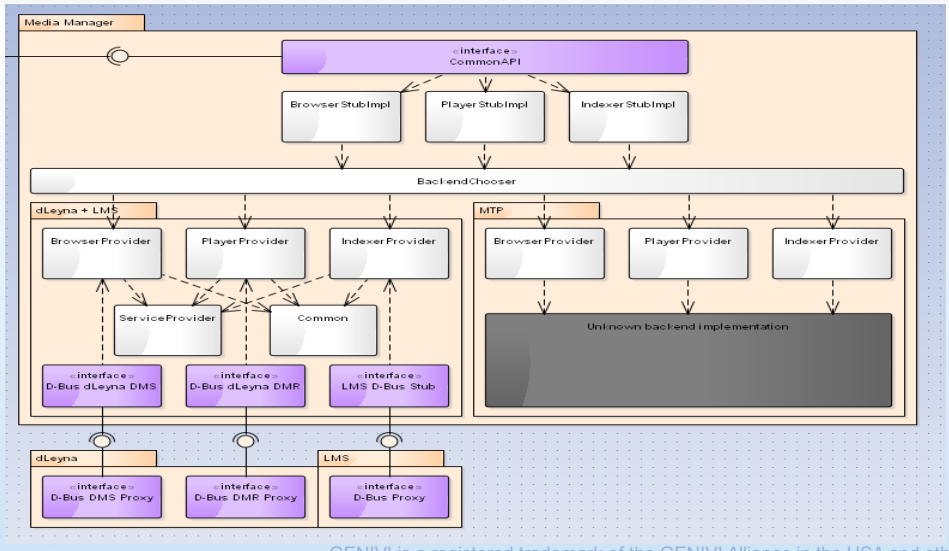
The Provider layer augments the functionality of dLeyna and LMS to fill in gaps in the requirements and to adapt functionality in dLeyna and LMS which doesn't follow the Media Manager use cases and specifications. An example is that Stop is treated as a Pause in the Media Manager. In dLeyna a Stop will clear any current buffers in GStreamer and reset any playback state, which is not desired in the Media Manager.

A different example is that the Media Manager should handle the current play queue, while dLeyna and Rygel are not designed for this, will offload the playlist functionality to the controlling component. The Media Manager thus adds playlist capabilities (plus some additional functionality to the MPRIS2 API) and hides this behind the same API as the rest of the functionality, so that this addition is seamless to the user.

The ServiceProvider contains some common D-Bus code for all Providers, and the Common module contains some common helper code.

The remaining six interfaces represent the D-Bus interfaces of dLeyna and LMS.

Internal architecture – Future Media Manager



- Future example
- Backend Chooser added
- MTP added
- Further backends possible
- Not implemented in PoC

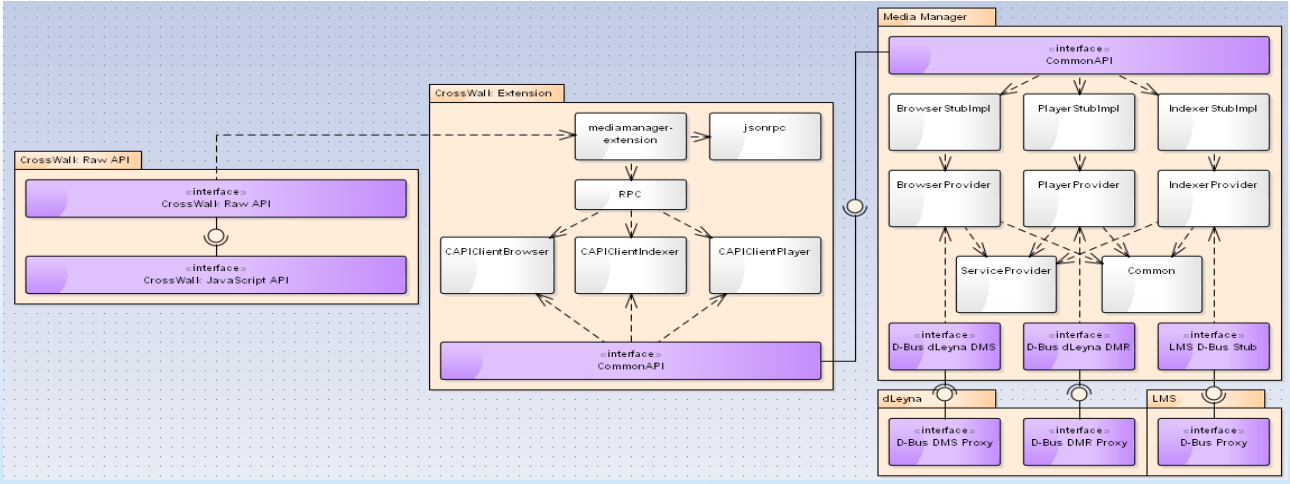
This slide shows an example of a possible future backend. MTP support could be added to the Media Manager by adding a BackendChooser module, which allows for example MTP to be selected rather than the current dLeyna backend.

Theoretically, backends could be selected through the URIs used to identify files, which means media from several backends could be mixed in the same play queue, and played seamlessly.

Further backends are possible using the same pattern, for instance support for proprietary devices and protocols.

MTP support and backend-choosing support has not been implemented in the current revision of the PoC, which is why the implementation specifics of the MTP support have been depicted as a grayed-out box.

Internal architecture - Recap



This slide is just to give a full view of the entire architecture again. It should now be easy to see where the different subsystems connect with each other.

Media Manager in Open Source

- Several open source projects are used in the Media Manager
- The most apparent ones are:
- dLeyna
- Rygel
- LMS
- Other components were used on lower levels as dependencies, for instance the GUPnP and libmediaart
- Several modifications have been made to the projects used
- Relevant modifications to dLeyna and Rygel have been sent for review and upstreaming to the respective projects
- Modifications to LMS are deemed to be out-of-scope for the upstream project, and will not be upstreamed

The Media Manager project aims to pick existing functionality from Open Source projects wherever possible. Any gaps found in the existing projects should be filled and relevant changes should be sent back to the original projects, or “upstreamed”.

Several projects were used, Rygel is heavily reliant on several libraries. A large part of the UPnP logic is for instance implemented as a separate library called GUPnP. Libmediaart is used internally in Rygel. This library is an offshoot from the Tracker project and is used to handle media art. This library is used indirectly by the Media Manager, and is mentioned to illustrate that the open source eco-system around the Media Manager is rather big.

For this project, dLeyna, Rygel and LMS were all found to have some functionality gaps which were filled under the project. dLeyna and Rygel were missing some metadata properties and needed some restructuring and additional code written to support all the metadata required by the Media Manager. LMS needed support for Album art indexing, which was also added.

Patches have been sent for review for the dLeyna and Rygel projects, but not for LMS. After some discussions with LMS developers, it was agreed that media art indexing should be handled outside of LMS.

Building the Media Manager

- Possible to build for Debian and Tizen (probably more distributions as well)
- Both Debian and Tizen require CrossWalk in the system
- meta-ivi support is in the works
- Requires patched versions of Rygel, dLeyna and LMS
- Everything is built using Cmake
- Code for Media Manager: http://git.projects.genivi.org/ (All repos starting with media-manager)
- Code for patched Rygel: https://github.com/Pelagicore/Media-Manager-Rygel-patched/
- Code for patched dLeyna: https://github.com/Pelagicore/Media-Manager-dLeyna-patched
- Code for patched LMS: https://github.com/Pelagicore/Media-Manager-lightmediascanner-patched

It is possible to build the Media Manager for Debian and Tizen, it is probably possible on more platforms as well, but these are the ones we've tried during the project.

Beware that CrossWalk must be available on the system in order to use the CrossWalk extension, and that this can be tricky to build in general.

Meta-ivi support is in the works, but feature completeness has been prioritized over this extra distribution support. In theory, there should be no problems building Media Manager in meta-ivi however.

Cmake is used throughout the project, and should provide a familiar build system for many developers.

In order to build the software, several pieces need to fit together, first all the dependencies need to be built, this means Rygel, dLeyna and LMS. These three will pull in several dependencies which also need to be built. CommonAPI also needs to be installed, and the GENIVI patches for D-Bus need to be applied.

Once all the dependencies are fulfilled, the software can be built natively on, for instance Debian or for Tizen using GBS.

Contributing to the project

- Discuss on the mailing list
- Send patches on the mailing list
- For use-cases and requirements
- Join the GENIVI CE-EG weekly telcos
- For HMI development
- Discuss on the AGL mailing lists

The back-end of the Media Manager project has been developed and specified by GENIVI, while the front-end has been developed by AGL. Therefore it is easiest to contact the corresponding group for questions regarding the different components.

For back-end discussions, patches to the code in the GENIVI repos, etc please reach out to the Media Manager mailing list on the GENIVI servers.

For use-case and requirement questions and discussions, please join the GENIVI DE-EG weekly telcos, or drop a mail on the mailing list.

For HMI related questions, please have a chat with the AGL team on their mailing lists.