Assignment 2: Report

Concrete Architecture of Kodi

November 19th, 2023

Team (Group 9)
Boadi, Kayin - boadi.kayin@queensu.ca
Codrington, Grace - 21gcmc1@queensu.ca
Glassford, Ethan - 21ewg5@queensu.ca
Hung, Vincent - 20vhjh@queensu.ca
Kliger, Isaiah - 19ik6@queensu.ca
Neadow, Joshua - 20jjn2@queensu.ca

Abstract

The purpose of this report is to describe the concrete architecture of the Kodi platform. We plan to do this by comparing our first report that outlines a conceptual architecture of the platform, with its actual source code that we will go through using the *Understand* tool. We will investigate absences, convergences, and divergences between our conceptual architecture and the concrete. We aim to find new components and relationships that we may have missed in our conceptual architecture and use them to create a new architecture.

Our report begins by outlining the relationships between the top-level concrete subsystems, as well as outlining detailed descriptions of the subsystems themselves, as we have derived them from the *Understand* tool. Using this tool, we decided that our conceptual architecture's idea that the platform uses a layered architecture was correct. We still believe this to be true based on the top-down setup of the architecture, however, the layered structure in our concrete architecture exists within the subsystems themselves, rather than in the entire architecture.

After the explanation of our concrete architecture, we move on to explain the derivation process, or how we conceptualised a concrete architecture based on the code we went through with *Understand.* We move on to a reflexion analysis, detailing (using techniques and information from course content and lectures) how to derive architecture and compare the conceptual and concrete architectures.

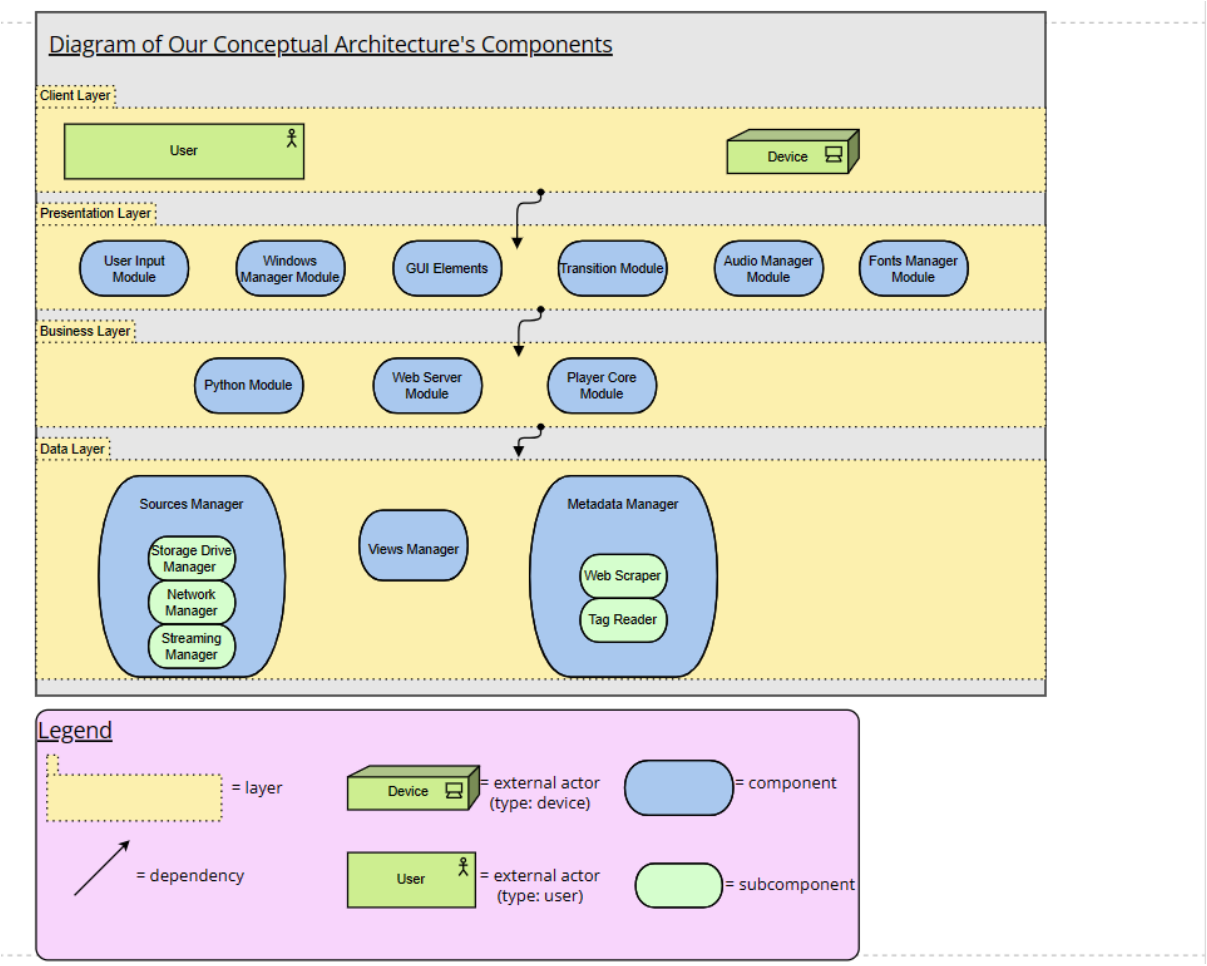
We also explain some discrepancies between the concrete and conceptual architectures we created for the Kodi platform. We describe new components and relationships that didn't exist in our conceptual architecture, but that we found in the concrete architecture, or *divergences.* We explain these new additions, as well as elements that did exist in our conceptual architecture, and are present in our concrete architecture, but are different, or *modified*.

After this, our report goes into a detailed description of one subsystem of Kodi. This subsystem is called Core, and it is responsible for the portrayal of media content on the platform. It appears to have a client server layout, and we detail its many dependencies and functions. After this, we explain a few use cases, and provide diagrams for them.

In conclusion, our report aims to explain how much more complicated the actual concrete architecture for Kodi is, especially in comparison to our conceptual architecture. We aim to demonstrate and outline Kodi's architecture in a way that promotes an understanding of the platform in a detailed, and sufficiently in-depth manner.

Table of Contents

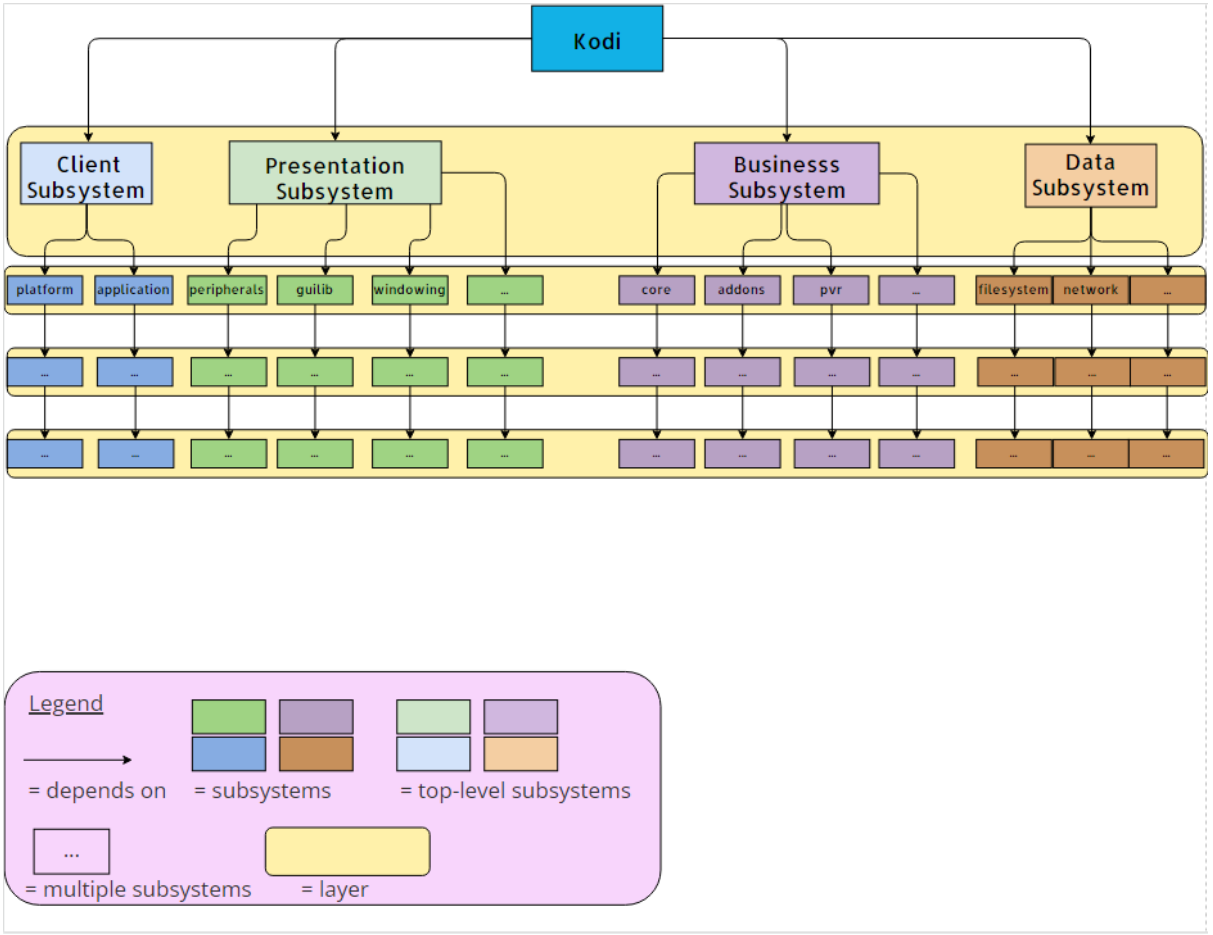
Conceptual Architecture



Derivation Process

The derivation process for our concrete architecture was done by looking at the dependency graph for Kodi's source code in Understand. After viewing which subsystems and components had the most dependencies, I added them to the architecture designer, which showed these subsystems and the components below them that they depend on. I then organised these subsystems into 4 different top-level subsystems based on our conceptual architecture: **Client**, **Presentation**, **Business**, and **Data**. Whichever component fit our description of what our conceptual architecture described it as, was put into that subsystem. For example, the ***platform*** subsystem was put under the **Client** subsystem as it was essential to the operating system compatibility described in our conceptual architecture. From there, all major components were put into one of the four layers previously mentioned, with each subsystem having components below them that they depend on in a layered structure, all dependencies eventually making a path to Kodi itself.

System Explanation of Concrete Architecture



System Components

Client Subsystem

The **Client** subsystem is the top-level subsystem that allows for Kodi to be accessible on multiple platforms or clients. This subsystem is the simplest of the top-level subsystems in terms of how many components it contains, but is incredibly important to Kodi's ability to be multiplatform with the ***platform*** component allowing for compatibility between multiple operating systems such as *Windows 10, Linux, Android, Darwin* (Apple's core operating system used on all of it's devices), and so many more platforms. This component uses libraries used by these platforms to make Kodi compatible on any device that uses these operating systems. This top-level subsystem is essential for Kodi to be a software usable anywhere on almost any device.

Presentation Subsystem

This is the top-level subsystem that makes up the user activity and interaction of Kodi. This subsystem consists of subsystems that manage the user input and the graphical user interface (GUI) of Kodi, as well as the different settings to modify Kodi based on user preference. One of the main components that this subsystem implements is ***guilib***, which is the main component that implements GUI libraries for the different platforms Kodi is available on. Another important component is the ***peripheral*** component which manages the different input devices that may be used by whatever client Kodi is on, such as keyboards, mouses, and joystick controllers. This subsystem also features the ***windowing***, which allows for basic window altering features depending on the platform used, such as minimising, resizing, and opening a program to match display resolution. All in all, this subsystem is what manages the visual components of the software.

Business Subsystem

The **Business** subsystem is composed of subsystems that work to manage features such as the add-ons, event managers, media players, and server accessibility. Its main components are the ***cores***, which handles all the media players of Kodi, ***add-ons***, which allow for the addition of user-made python add-ons to change the standard Kodi experience, ***pvr*** which allows for features similar to physical DVR devices like the recording of live-television, and much more subsystems. This is the subsystem that handles the main features of Kodi that allow users to have a smooth and easy-to-use media player experience.

Data Subsystem

The **Data** subsystem is made up of components used for managing multimedia files, the organisation of how different media are displayed and ways to access the data stored on external storage devices such as harddrives and CDs. Examples of the components of this top-level subsystem are: ***favourites*** which is a listing service that allows for the use of favourites lists to organise movies and menus into lists for easy access. Another component under the **Data** subsystem is ***cdrip*** which allows for the “ripping” of Compact Discs (CD)s, which is the process of extracting raw audio data from a CD. An important component under this subsystem is the ***network*** subsystem which allows for the **Data** subsystem to make http requests to a server to receive data usable by Kodi such as media hosted on another server.

Dependencies & Structure

All of the top-level subsystems are all bidirectionally dependent on each other, meaning that each subsystem depends on the other top-level subsystems and vice-versa. This is due to how the lower level subsystems from the top-level subsystems interact between each other. A lot of the lower level subsystems require services from other subsystems in different top-level subsystems such as ***settings*** from the **Presentation** subsystem being used by a large number of subsystems. Despite this, we still believe our concrete architecture adheres to a layered structure, where each top-level subsystem is

organised into a layered architecture, where the top-level subsystems are dependent on the subsystems below it. The entire program is not structured in a layered architecture as we originally thought, but more so four layered subsystems that interact with each other.

System Reflexion Analysis

New Subsystems

Platform Subsystem:

This subsystem is part of the client layer of Kodi. It has important functionalities that allow Kodi to interact across platforms. It was absent from our conceptual architecture because our conceptual architecture's client layer only included a user and their device, and we didn't account for the coding necessary to allow Kodi to be functional across different platforms.

Application Subsystem:

This subsystem is also part of the client layer, and has similar functionality to the platform subsystem, except that this subsystem is responsible for making Kodi accessible as an application on different devices. This was absent from our conceptual architecture, but we had a *devices* module.

PVR Subsystem:

This subsystem is part of the business layer. It allows for physical DVR plugins to Kodi, as well as containing many other systems to handle media playing on Kodi. These functionalities were absent in our conceptual architecture. We did not account for systems that were responsible for the recording of live television, for example, in our conceptual architecture.

CDRIP Subsystem:

This subsystem allows for the "ripping" of CDs, in other words it allows for raw data to be abstracted from CDs. It was absent from our conceptual architecture due to the absence of the platform subsystem (in the client layer) in our conceptual architecture. In other words, we did not account for different platforms and devices, such as CDs, in our original architecture.

Modified Subsystems

Peripheral Subsystem:

This subsystem is in the presentation layer of our concrete architecture, and it corresponds to the user input module in the presentation layer of our conceptual architecture. It is similar because it is responsible for interpreting any input from user devices, including keyboards, mouses, and controllers.

Windowing Subsystem:

This subsystem is in the presentation layer of our concrete architecture, and it corresponds to the windows manager module in the presentation layer of our conceptual architecture. It is modified from this element of our conceptual architecture because it handles window formatting, resizing, and displaying. It is different however, because it does not manage necessary libraries and dll files for linker elements of Kodi.

GUILIB Subsystem:

This subsystem is in the presentation layer of our concrete architecture, and it corresponds to the GUI elements module in the presentation layer of our conceptual architecture. It is responsible for implementing GUI libraries, in other words for displaying content that is interactive with the user, such as buttons, menus, and play/pause features.

Core Subsystem:

This subsystem is in the business layer of our concrete architecture, and it corresponds to the player core module in our conceptual architecture. We derived that this subsystem is a modification of our player core module because it allows the user to view many types of media files, and it interfaces with Kodi's many media players.

Addons Subsystem:

This subsystem is in the business layer of our concrete architecture, and it corresponds to the python module in the business layer of our conceptual architecture. We made this derivation because this subsystem manages python-made plugins and scripts. It also, like the python module, is responsible for user customization.

Favourites Subsystem:

This subsystem is part of the data layer of our concrete architecture and it corresponds to the metadata manager in the data layer of our conceptual architecture. It allows for personalised user information, namely a listing service that allows for Kodi's media to be organised into lists. It is similar to the metadata manager, because this listing service utilises tags and online information to make lists of organised content.

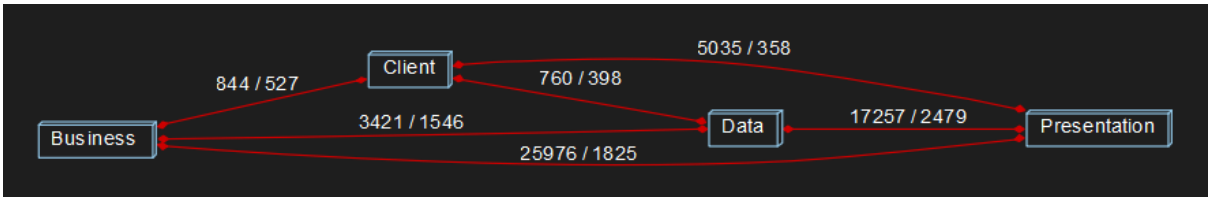
Filesystem Subsystem:

This subsystem is part of the data layer of our concrete architecture and it corresponds to the views manager in the data layer of our conceptual architecture. This subsystem is responsible for organising file types. For example, if a file is a .mp3 file, this subsystem will put it in the directory it's supposed to be found in, in this case that would be some sort of music/audio directory. The subsystem contains many functionalities that allow you to search through Kodi for file types, and organise them. This corresponds with the views manager because the views manager ensured that the right view was displayed, aligning with the correct media. In order to do this, the views manager (that we conceptualised) would have to organise different media into different directories, and then handle the formatting of each directory differently.

Network Subsystem:

This subsystem is part of the data layer of our concrete architecture, and it corresponds to the sources manager module in the data layer of our conceptual architecture. It is responsible for interaction with other sources, as was the sources manager in our conceptual architecture. The difference between this concrete subsystem and our conceptualised sources manager is the added functionality that allows Kodi to make requests to servers, and receive media hosted by other servers.

New Dependencies



The dependencies shown here illustrate what we have mentioned elsewhere in this report, both in our abstract and in our explanation of the components. We no longer believe that our layered architecture in our conceptual architecture was correct, as you can see here, the data "layer" depends on all three other layers, which was not in our original design. So, there are obviously many major new dependencies in our system that were not accounted for in our original architecture, including ones from the Data Layer→Presentation Layer, Data Layer→Client Layer, and Data Layer→Business Layer.

Reflexion Analysis (Business Subsystem):

As you can see in our concrete architecture it is still layered however new components have been added or changed in the Business subsystem inorder to improve efficiency, application, and quality of life.

Discrepancy: New games module where you can add and play games using the Kodi interface.

Rational: this module was added to adapt to the rising demand to use Kodi as a gaming interface. This development was only seen recently which is why it was not in the conceptual architecture. The Games module is dependent on several different modules such as addons, Agents, controllers, Dialogs, ports, tags, and windows. These allow Kodi to stream games on various different platforms.

Discrepancy: The Web Server module that originally was in the Business subsystem in our conceptual architecture was moved to the Data subsystem.

Rational: This is because it was the only module that used networking data in the Business subsystem. So it made more sense to move it to the Data subsystem since other modules in the Data subsystem use network modules as well.

Discrepancy: A new Weather subsystem added to the Business subsystem.

Rational: This new add-on was added due to a demand for the ability to see weather information natively in Kodi's interface. This was not originally seen as needed in our conceptual architecture.

Discrepancy: New Events module added to make and log events for Kodi

Rational: This was added to make logging events easier

Discrepancy: In our conceptual architecture we had one python module that allows for the use of externally developed add-ons which can drastically add to or change Kodi's user experience depending on what you need. In the concrete architecture this module has been split into two different modules. One module for interface and one for add-ons.

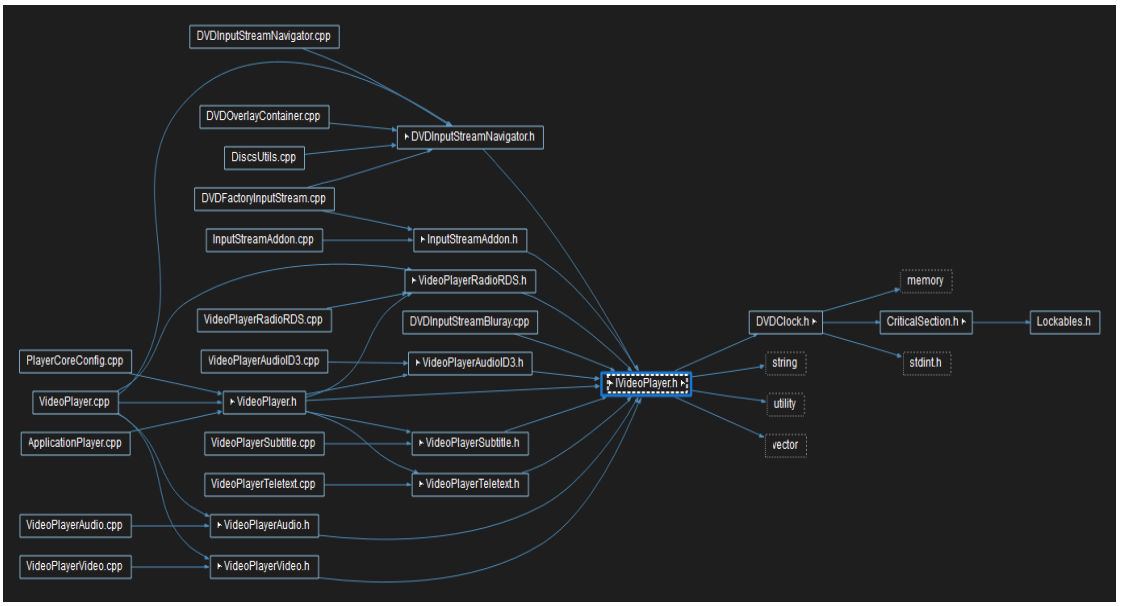
Rational: Having these two different modules makes it simpler to organise making it easier for the user to find what they need as well as easier to explain to others.

Core Subsystem

Explanation of Architecture

According to the description provided by the Kodi Documentation, Core is responsible for managing, processing, controlling and playing multimedia content. It acts as an interface between the files and the hardware or software required to playback. Its tasks can be further broken down into decoding, synchronising audio and visual content and managing playback flow (pausing, seeking, speedup).

The code for the video player and its subsequent components can be found in IVideoPlayer.h and the related files. Kodi has taken a modular approach to its code as seen in their stated coding guidelines, (https://kodi.wiki/view/Windows_development) this results in a very portable design where multiple elements of the video player capabilities are broken up as seen in the diagram below. Below is a description of the information and variables



IVideoPlayer.h provides and how they relate to a client-server architectural style.

IVideoPlayer is the interface to the server in an architectural sense, all other components that require video and audio playback and its related features depend on it to obtain information from the hardware or software that creates the video stream. This component comprises three main classes, two of which hold most of the key audio and visual information. The video information is provided by the rest of the “server”, regardless of whether a physical DVD is used it will go through this subsystem unless explicitly using a custom external player or community-made add-on. With those two exceptions, the information is provided through a physical DVD or video which will act as though it was read from a DVD.

IDVDStreamPlayerVideo & IDVDStreamPlayerAudio, both have their respective stream and information about the video or audio and fall under a public class named IDVDStreamPlayer. This provides accessible metadata that can be used.

```
  explicit IDVDStreamPlayerVideo(CProcessInfo& processInfo) :
IDVDStreamPlayer(processInfo) {}
  ~IDVDStreamPlayerVideo() override = default;
  bool OpenStream(CDVDStreamInfo hint) override = 0;
  void CloseStream(bool bWaitForBuffers) override = 0;
  virtual void Flush(bool sync) = 0;
```

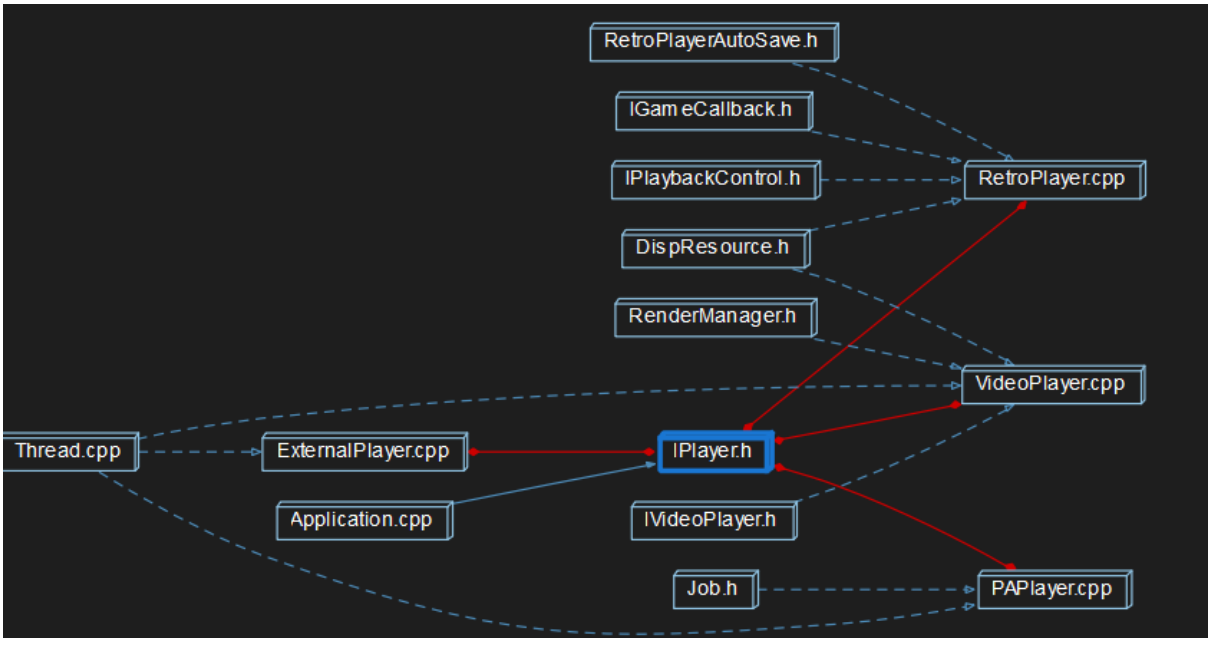
```
bool AcceptsData() const override = 0;
virtual bool HasData() const = 0;
bool IsInited() const override = 0;
void SendMessage(std::shared_ptr<CDVDMsg> pMsg, int priority = 0) override = 0;
virtual void EnableSubtitle(bool bEnable) = 0;
virtual bool IsSubtitleEnabled() = 0;
virtual double GetSubtitleDelay() = 0;
virtual void SetSubtitleDelay(double delay) = 0;
bool IsStalled() const override = 0;
virtual bool IsRewindStalled() const { return false; }
virtual double GetCurrentPts() = 0;
virtual double GetOutputDelay() = 0;
virtual std::string GetPlayerInfo() = 0;
virtual int GetVideoBitrate() = 0;
virtual void SetSpeed(int iSpeed) = 0;
virtual bool IsEOS() { return false; }
```

The third Class IVideoPlayer contains:

OnDiscNavResult() - What video or audio is obtained or on the disc
GetVideoResolution() - The resolution of the video or image
IVideoPlayer() - Constructor which holds the above two classes with streaming information

This is where functionality ends according to the conceptual diagram, however with the modularity and increasing demand for more functions on Kodi. The concrete architecture for the player core has three additional functionalities in the form of RetroPlayer.{cpp,h}, ExternalPlayer.{cpp.h} and PAPlayer.{cpp.h} These expand on the initial architecture to include the capability to launch and play video games as well as community-made add-ons to override IVideoPlayer.h with custom audio and visual rendering and audio-only sources.

The Player Core component that integrates these functionalities is IPlayer, by default this calls the main IVideoPlayer explored in the above section which renders both audio and visual information but depending on what the user wants can use the other three additions to Kodi's functionality.



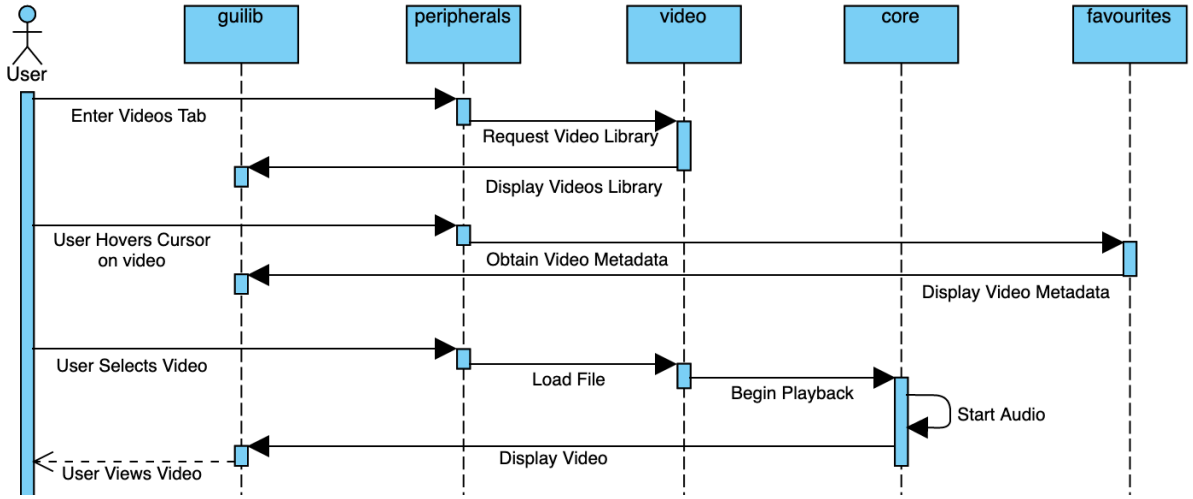
Reflexion Analysis of Core Module

Discrepancy: the modules RetroPlayer, ExternalPlayer, and PAPlayer were added to the conceptual architecture to allow for more functionality.

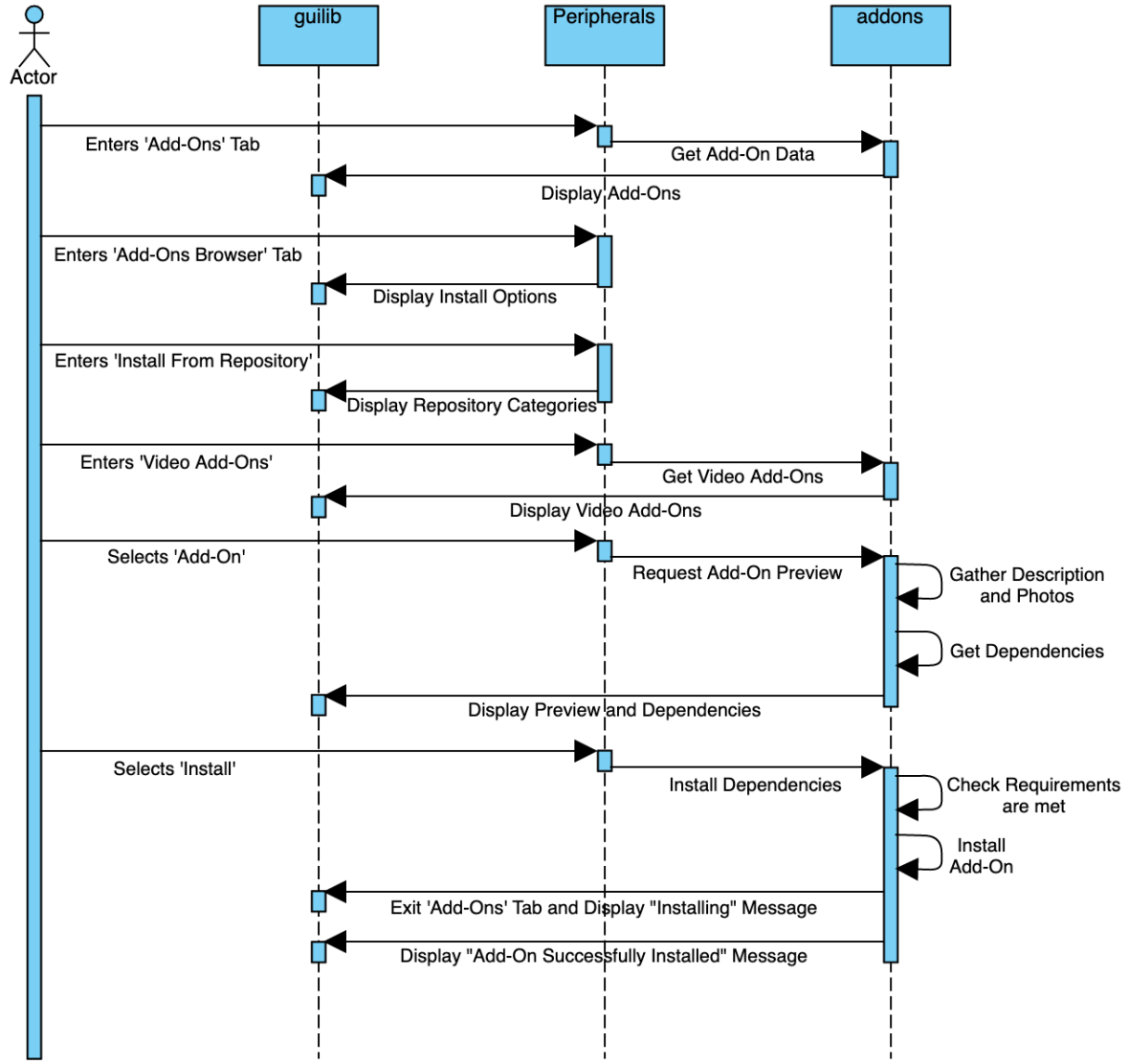
Reasoning: RetroPlayer is a component that allows for games to be played on kodi. ExternalPlayer allows the ability for the community to create addons that override the default IPlayer Core. This allows for more modularity and community input on kodi. PAPlayer allows for audio to be played in kodi.

Use Cases

This sequence diagram depicts the 'Play Video' use case in Kodi. The process initiates when the user accesses the Videos tab, triggering the video subsystem to retrieve videos from the current disk, displayed via the guilib subsystem. Hovering over a video prompts the favourites subsystem to fetch and display metadata (photos, descriptions, etc.) through the guilib subsystem. Upon video selection, the video subsystem loads the video, and the core subsystem initiates audio and video playback, with the video playback being displayed by the guilib subsystem.



This sequence diagram illustrates Kodi's 'Add an Add-On' use case. Initially, upon the user's entry into the 'Add-Ons' tab, the addons subsystem fetches and displays all installed add-ons via the guilib subsystem. As the user navigates to the 'Video Add-Ons' tab, the guilib subsystem updates the display accordingly. In the 'Video Add-Ons' section, the addons subsystem retrieves and presents all available add-ons through the guilib subsystem. When a user selects an add-on, its metadata and dependencies are displayed in a preview by the addons subsystem using the guilib subsystem. If the user opts to install, the addons subsystem verifies and installs the necessary dependencies. Following successful requirement fulfilment, installation begins, with progress and confirmation messages displayed via the guilib subsystem.



Lessons Learned

A major way that we could have improved this report was to ensure that every group member had *Understand* set up and ready further in advance. We found it very difficult to use, and had a lot of confusion about extracting some of the methods in Kodi's code.

A second lesson we learned was to communicate as we worked. It was very important for all of our sections to work well with each other, and we found a lot of major work to be done during team meetings and online discussions. We should have kept better track of what everyone was doing, and what had already been done, to make sure we were all prioritising the most important tasks so everyone could finish their work on time.

Lastly, it was very complicated to find exactly which methods or strategies that *Understand* used to find key modules. So, in our description of the player core module, we found another module that we think might align better to our work. We found this too late, as we had already formatted our player core module description, so this is another example of the lesson being learned that we should have gotten to know *Understand* better.

Conclusion

In conclusion, this report compares the conceptual architecture of kodi with its concrete architecture. First we summarise our findings from the software tool Understand as well as show our constructed concrete architecture in comparison to our conceptual architecture. Next we describe the different layers that Kodi uses as well as show how they interact with each other within the concrete architecture. Then we dive into an in depth reflexion analysis including any new subsystems that we found as well as any modified subsystems discovered. Additionally we describe the new systems that were added to the business layer of kodi and the reasoning why they were added. Then we give an explanation of the core subsystem showing their structure and dependencies. Finally we solidify our understanding of the concrete architecture with some use cases showing how this architecture is used.

References

Archive:paplayer. Archive:PAPlayer - Official Kodi Wiki. (2020, July 20). https://kodi.wiki/view/Archive:PAPlayer

AudioEngine. AudioEngine - Official Kodi Wiki. (2020, August 4). https://kodi.wiki/view/AudioEngine

Basic playlists. Basic playlists - Official Kodi Wiki. (2023, February 13). https://kodi.wiki/view/Basic_playlists

External players. External players - Official Kodi Wiki. (2021, July 25). https://kodi.wiki/view/External_players

Favourites. Favourites - Official Kodi Wiki. (2023, January 25). https://kodi.wiki/view/Favourites

Games. Games - Official Kodi Wiki. (2023, February 8). https://kodi.wiki/view/Games

Getting Started with Kodi Retroplayer. Kodi Community Forum. (2019, February 11). https://forum.kodi.tv/showthread.php?tid=340684

Karl-Bridge-Microsoft. (2020, August 19). *Windows Events*. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/events/windows-events

PVR. PVR - Official Kodi Wiki. (2023, January 25). https://kodi.wiki/view/PVR

Videoplayer. VideoPlayer - Official Kodi Wiki. (2020, July 12). https://kodi.wiki/view/VideoPlayer

Weather. Weather - Official Kodi Wiki. (2023, January 24). https://kodi.wiki/view/Weather

Wikimedia Foundation. (2022, November 9). *Dialog box*. Wikipedia. https://en.wikipedia.org/wiki/Dialog_box

Wikimedia Foundation. (2023a, April 25). *Peripheral bus*. Wikipedia. https://en.wikipedia.org/wiki/Peripheral_bus

Wikimedia Foundation. (2023b, July 5). *CD Ripper*. Wikipedia. https://en.wikipedia.org/wiki/CD_ripper

Wikimedia Foundation. (2023c, November 6). *Digital Video recorder*. Wikipedia. https://en.wikipedia.org/wiki/Digital_video_recorder

Wikimedia Foundation. (2023d, November 16). *Darwin (operating system)*. Wikipedia. https://en.wikipedia.org/wiki/Darwin_(operating_system)