Assignment 1: Report

Conceptual Architecture of Kodi

October 22nd, 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

Kodi is a free and open-source home media player and entertainment hub that is accessible across multiple platforms. Kodi's strength lies in the adaptability and user-friendliness while providing access to a wide range of features including videos, music and games. Kodi also supports add-ons and plugins that enable users to access online content or add new features.

This paper will outline and detail the conceptual architecture of Kodi. We will describe the overall structure of Kodi and its subsystems. We'll delve into how these subsystems interact and provide a thorough analysis of these interactions. Including the derivation process and evolution of the software over time. Through this we develop a solid understanding of how these systems interconnect, creating the user experience Kodi is known for. The paper then explores the flow of data and control through the illustration of use cases and sequence diagrams. This paper will serve as a comprehensive resource that provides an in-depth understanding of the conceptual architecture of Kodi.

Introduction

In today's world digital media has become an essential part of our everyday lives, the need for a flexible, user-friendly and open source media and entertainment hub has grown. With the many separate TV, Movie and entertainment services out there, Kodi has stood out to millions of satisfied users that utilise its ability to unify all the different multimedia services into a single platform. Starting in the early 2000's Kodi has evolved from a home theatre application to a media ecosystem, integrating video, music, gaming and more under a single banner.

Kodi's strength not only stems from this media ecosystem it has built throughout the years but also the adaptability it provides through easy integration of user-generated add-ons and plugins. This allows Kodi to further expand and enhance its features such as the addition of online content streaming or feature customization, ensuring each user can configure Kodi to match the needs and wants. With this, Kodi can become a personalised portal to a vast array of digital entertainment.

This report aims to explore the conceptual architecture of Kodi and the role it plays in creating the adaptability Kodi prides itself in. The report is broken down into multiple sections offering insight into different parts of the conceptual architecture; a general overview of the conceptual architecture, the Derivation Process, the Evolution Process, Control and Data flow of the software, along with a Data Dictionary and the naming conventions used throughout the report. We conclude with a "Lessons Learned" section where we iterate any challenges our team faced as well as noteworthy lessons we could take away from the report.

Table of Contents

System Components

Client Layer

The client layer is the front facing component of the software architecture. This handles the GUI which allows the user(s) to navigate movies, music and pictures through user friendly design. This should have a consistent design for each platform the project is designed for. As per the documentation provided in our references. Examples of devices this platform runs on are Android/IOS devices, Television and PC. In other words, the client layer is whatever operating system or device the user is currently on and using to access Kodi.

Presentation Layer

The Presentation Layer is the second layer in Kodi's software architecture and it interacts directly with the above client layer. This component enables the user to seamlessly interact with whichever operating system/device (client) the user is operating on at the time whether it be it a mobile phone using iOS, a computer using Linux, or a television using Android TV. This layer is the part of Kodi with which the user interacts, from its user interface to the processing of user inputs through a variety of peripheral devices. The subsystems/modules of the presentation layer that allows for these interactions are as follows:

User Input Module

The User Input Module consists of any peripheral input devices that allows for the user to interact with the Kodi software. This module interprets any inputs from devices such as keyboards, mouses, and television remotes, and relays it to the client. This module is the sole module that directly captures and utilises user input data; other modules utilise this feature to be able to analyse user input and apply it to other areas of Kodi's software such as their user interface.

Windows Manager Module

The Windows Manager module refers to a subsystem of the presentation layer where Kodi utilises libraries provided by an operating system such as Microsoft's dynamic link libraries (dll) files to allow for windows management tasks. These include actions like: minimising windows to the background of a device, dragging windows around, resizing windows, and more.

GUI Elements

Kodi's GUI elements refer to the digital interface the user interacts with on their device. These include visual components such as the windows, buttons, search boxes, volume sliders, and any other visual aspect of the software the user views

and interacts with. Kodi also features a unique modular GUI which allows for developers to be able to alter the GUI through their skinning engine.
Kodi's skinning engine allows for users to be able to customise their GUI provided by external developers. Each skin allows for a completely different way for you to interact with Kodi and allows for users to customise their experience to fit their individual needs.

Translation Module

The translation module allows for Kodi to be able to translate text on their user interface to allow users from different parts of the world to be able to interact with the software. The user is able to select which language they'd prefer their interface to be in and it will change the language accordingly.

Audio Manager Module

The audio manager is what Kodi uses to interact with the various audio settings of whichever client the user is utilising. These features include:

- Configuration of the available audio devices
- Audio output selection: Speakers, HDMI output, Bluetooth, etc
- Audio output mode selection: Stereo, Dolby Digital, DTS, etc
- Adjusting volume
- Audio delay

Business Layer

The business layer is the third layer of Kodi's software architecture that provides many services that the presentation layer is able to use and implement. It incorporates many different plugins, scripts, add-ons and libraries Kodi uses to better the user experience. This also incorporates the streaming client used for reading and playing media. The modules used in this layer are:

Python Module

The python module is one of Kodi's most important modules when it comes to the softwares customizability. This module 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. This module uses python-made plugins and scripts to allow developers to be able to create new features within Kodi and seamlessly apply them without the need to update Kodi's source code.

Web Server Module

The web server module is what allows a user to be able to manage their Kodi system remotely via Kodi's web interface. Kodi's web interface can allow for multiple

users to be able to access the same media library through their web browser or a completely separate device. This typically can only be done if the devices are on the same network for security reasons, but can also be done on a separate network if you have access to the IP the device running Kodi is using. Kodi is capable of doing this by using their web interface, as well as through the implementation of HTTP APIs which are protocols that allow for clients to access media between each other online via HTTP (Hypertext Transfer Protocol).

Player Core Module

The player core module is one of the most important components of Kodi as this is what processes, manages, controls, and plays all multimedia provided to Kodi. It is what allows the user to view many types of media files which is done through video and audio rendered as well as different codecs such as DIVX and AC3. Codecs allows for the decoding of data, which enables Kodi's software to interpret most media files received and play it to the user via their audio (PAPlayer) and video players (DVDPlayer).

Data Layer

The Data Layer contains all elements that are responsible for storing data. These data management functions relate to tasks like saving media, managing disks, and managing files. The components (and their respective subcomponents) that this layer contains are as follows:

Sources Manager

This component contains functions allowing users to add different sources. Some of these sources include storage drives such as HDD, CD and DVD. The source manager also allows users to access different media sources, via network protocols and functions, to meet Kodi's functional requirement of playing multimedia content. Another subcomponent of the Sources Manager is a function that allows Kodi to integrate with streaming devices. This allows Kodi to have a functionality that allows scheduled recordings, as well as one to support live streaming channels. Therefore, the sources manager component in the Data Layer should have three subcomponents: The Storage Drive, the Network Manager, and the Streaming Manager. one that handles storage drives (we will call this the Storage Drive Manager), one that handles media source access (we will call this the Network Manager), and one that handles streaming (we will call this the Streaming Manager).

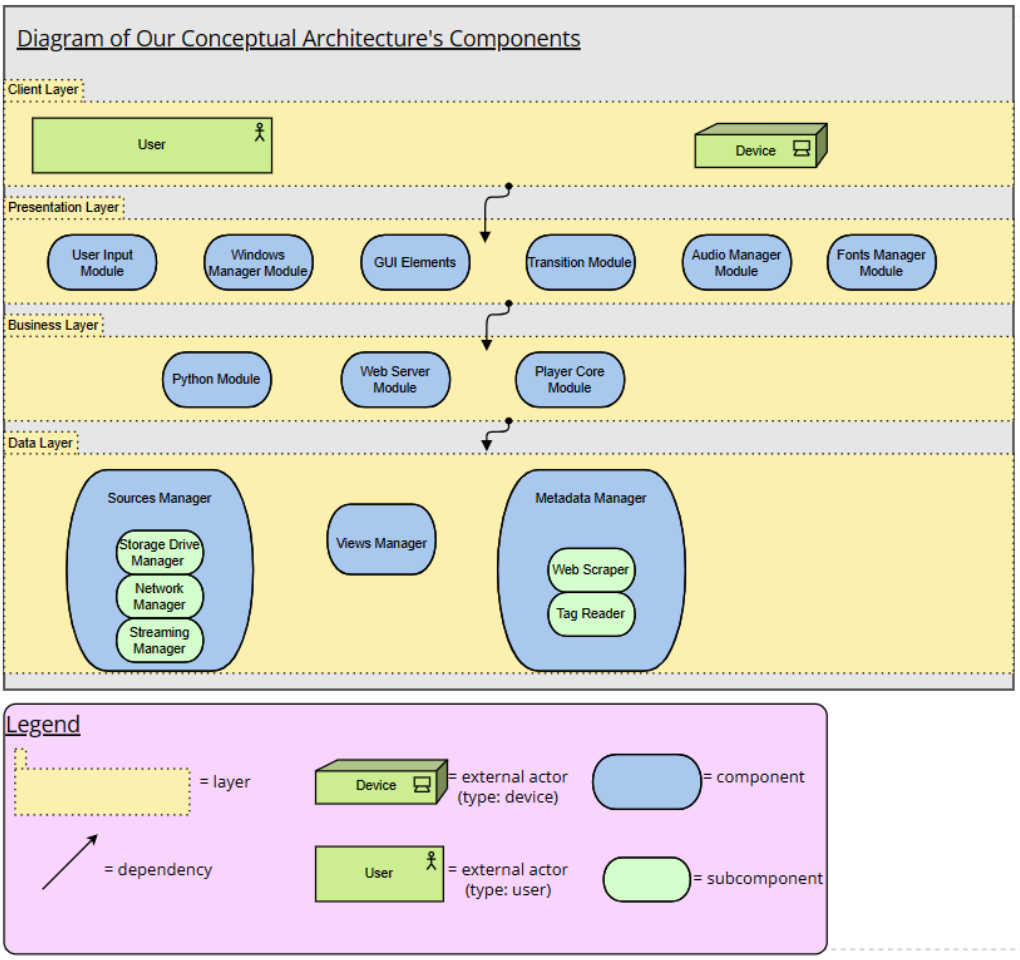
Views Manager

This component controls the different ways Kodi can be viewed. There are different visual organisations or formats that users can see when they are using Kodi. They are formatted here, and the views manager is the component responsible for presenting the different ways Kodi's content can be displayed for a user.

Metadata Manager

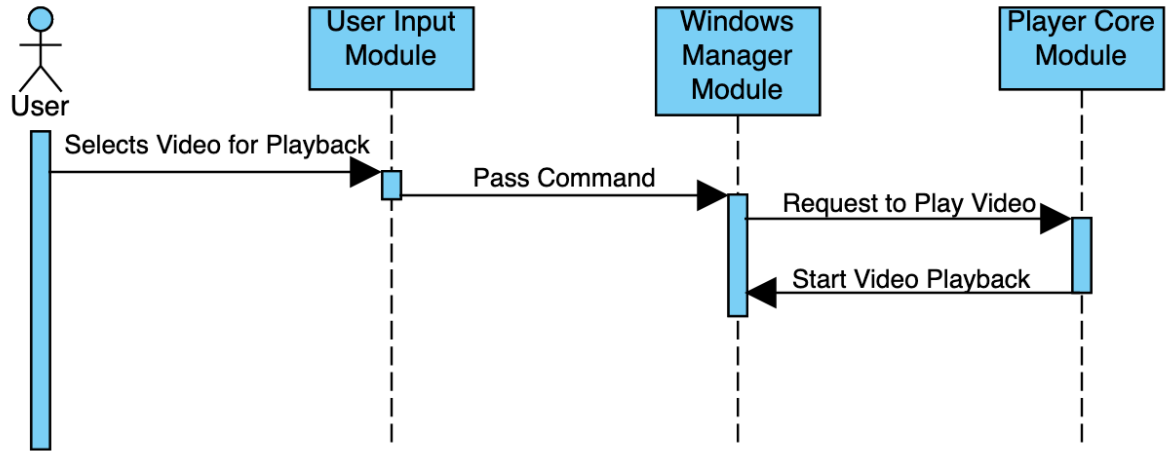
This component is responsible for finding, and presenting to the user, further information about the content that they are viewing. For example, in the use case that a user is selecting a movie to view, the Metadata Manager might find data about that movie to show the user, such as the year the movie was released, notable actors in the movie, a description of the movie, etc. The Metadata Manager's two main functions are to find data and organise tags. To find data, the Metadata Manager would have a subcomponent that scours the internet for databases with information about movies, TV, music, etc. This subcomponent would extract information from online sources and return it for formatting. To format tags, the Metadata Manager would include a subcomponent that reads data directly from files and uses that data to decide how content in the media library should be organised. For example, this component is responsible for organising music into genres. Therefore, the Metadata Manager component in the Data Layer should include two subcomponents: one to find information from the internet (we will call this the Web Scraper component) and one to organise tags of media (we will call this the Tag Reader component).

Diagram of Components

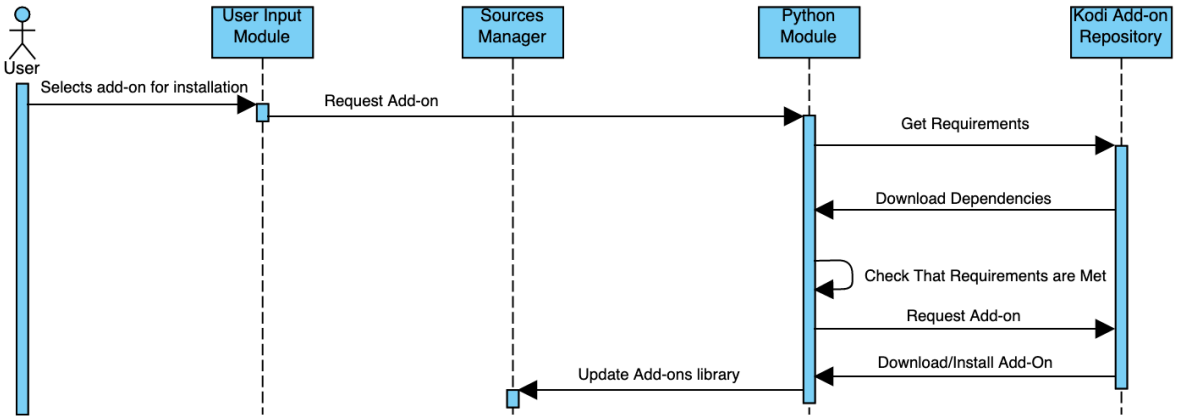


Use Cases:

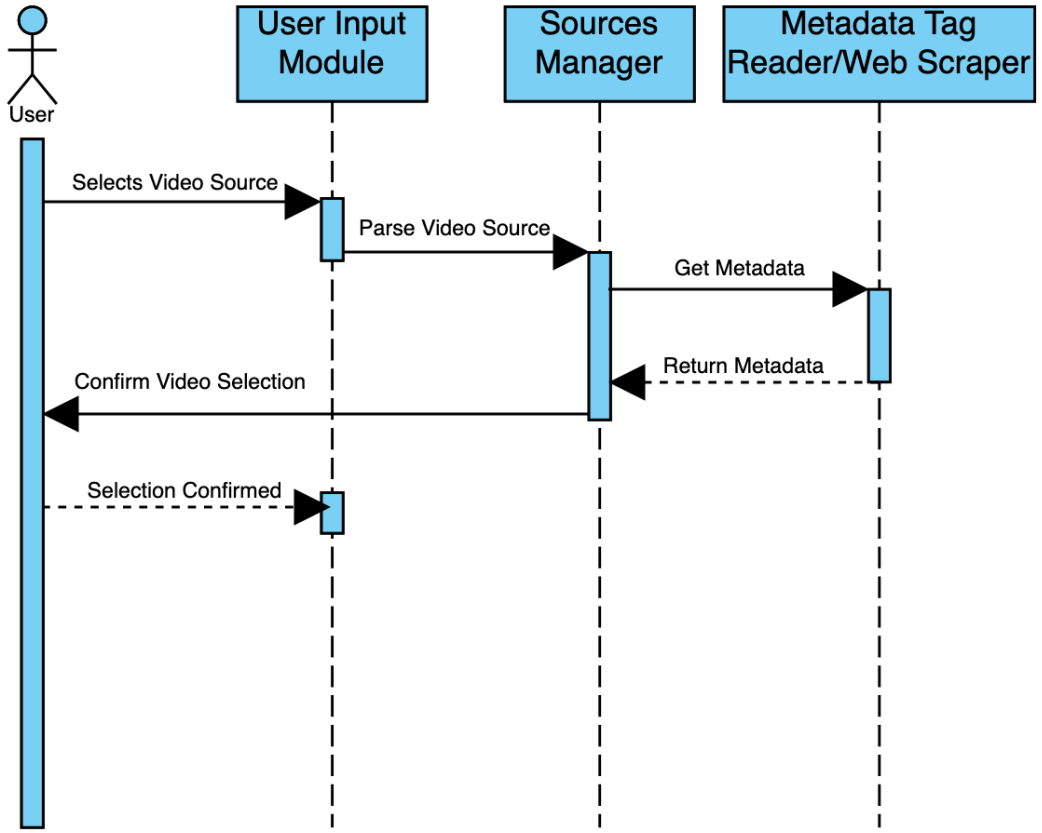
This sequence diagram illustrates the use case of playing a video on Kodi. Once the user selects a video for playback the User Input Module passes the command to the Windows Manager Module. Then, the Windows Manager Module sends a request to the Player Core Module to play the video and the Windows Manager Module displays video playback.



This sequence diagram illustrates the use case of adding an extension/add-on to Kodi. It shows what happens when the user selects an add-on for installation. First, the User Input Module passes the user's request to the Python Module to add the add-on. The Python Module then gets the requirements necessary for the add-on using the Kodi Add-on Repository and downloads the dependencies. The Python Module then checks that all requirements are met and downloads and installs the add-on from the Kodi Add-on Repository. Finally, the Sources Manager updates the Add-ons Library.



This sequence diagram illustrates the use case of adding a video source to Kodi. It shows how the User Input Module processes the user input and tells the Sources Manager to parse the video source. While parsing, the Sources Manager uses the metadata Tag Reader/Web Scraper to get metadata. Then, the user is shown the video files discovered along with the discovered metadata. The user is then asked to confirm the import.



Approach to Conceptual Architecture Development:

Our process to make the Conceptual Architecture for Kodi started out by looking at the wiki for Kodi. This helped us start to understand the general structure of its various systems. Simultaneously, we searched the web to try to find additional references that could help in our understanding.

Through this process we believe that Kodi is a layered architectural style for a few reasons. First it is organised into a presentation layer, business layer(aka logical layer), and data layer. Layered modules use these layers to help keep what is shown to the user separate from the logical operations that are happening behind the curtain. We think at this stage there should not be a concurrency view because it is mainly a layered architecture. However we have not looked in depth at the code enough to know for sure.

We also learned that Kodi is open source which allows for anyone to contribute to the project. However there is a limited group of developers that allow for these changes to go through. These members are called board members and they can be found on Kodi's public GitHub (the bottom link in our reference page).

After this review, we made a core set of components and, subsequently, divided up the work to further analyse these parts. Our main goal throughout this phase was to then think of use cases within Kodi's system, and the corresponding components associated with these use cases. This discernment played a pivotal role in prioritising the sections that required more in-depth scrutiny.

Naming Conventions

- GUI → Graphical User Interface
- DLL → Dynamic Link Library
- The Storage Drive → a sub-component of the Data Layer that handles the storage drives
- Network Manager → a sub-component of the Data Layer that handles media source access
- Streaming Manager → a sub-component of the Data Layer that handles streaming
- Web Scraper component → a sub-component of the Meta Manager component in the Data Layer that finds information from the internet
- Tag Reader → a sub-component of the Meta Manager component in the Data Layer that organises tags of media
- HTTP → HyperText Transfer Protocol
- API → Application Programming Interface
- HDD → Hard Disk Drive
- CD → Compact Disc
- Board members → group of members that allow any changes to get pushed to the main program

Data Dictionary

Architecture: The organisation of a system, its subsystems and their behaviour in software.

Codec: Software that encodes data to compress it into media, then decodes it to view/play the media. DIVX and AC3 are examples of codecs.

HTTP API: An api that utilises HTTP to send media between clients via the internet.

DLL: A DLL is a windows file library containing code and data that can be used by multiple programs at the same time.

Presentation layer: The visual aspect of Kodi's software This includes components like the GUI, and windows that Kodi uses.

Business layer: Is the layer containing the plugins, libraries, and scripts Kodi needs to function. This layer also contains the media player and codecs used by the media player.

Data layer: The layer containing components pertaining to managing, storing, and changing media data. This is the deepest layer of the Kodi architecture.

Lessons Learned

- Because the elements of this report are very closely related, it was important that we all understood the sections we didn't write. Therefore, a way we could have improved this report would have been to meet more often and explain our work to each other.
- Reading the given material was very helpful when coming up with our architecture. Therefore a lesson we learned was to always look at the given resources for an assignment before looking elsewhere on the internet for resources.

Conclusion

In conclusion, this report outlines a conceptual architecture of Kodi, beginning with a small description of its functionality. The report continues to detail the components of the architecture and includes a visual representation of its layered style. The report also includes three sequence diagrams of potential common use cases. And finally, the report explains how we decided on the above information, a dictionary of our elements, and things we learned while creating this report as a group. We will use this conceptual architecture going forward with our term project to have a deeper understanding of the code we are looking at.

References:

Architecture. Official Kodi Wiki. https://kodi.wiki/view/Architecture

Delft Students. *Architecting software to keep the lazy ones on the couch*. Delftswa. https://delftswa.github.io/chapters/kodi/

Torbet, G. (2019, April 29). *How to use the kodi web interface to control kodi*. MUO. https://www.makeuseof.com/tag/how-to-use-kodi-web-interface/

Gillis, A. S., & Lazar, I. (2022, February 15). *What is a codec?*. Unified Communications. https://www.techtarget.com/searchunifiedcommunications/definition/codec

Gill, S. (2023, August 25). *HTTP API VS REST API: 3 critical differentiators*. Hevo. https://hevodata.com/learn/http-api-vs-rest-api/#http

Deland, H. (2023, April 28). *Dynamic Link Library (DLL) - windows client*. Dynamic link library (DLL) - Windows Client | Microsoft Learn. https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

Team kodi. GitHub. (n.d.). https://github.com/orgs/xbmc/people