**The Conceptual Architecture Behind Kodi**

Junsu Cho, 21jc30@queensu.ca

Aidan Macdonald, 20aam43@queensu.ca

Ava Salas, 20awys@queensu.ca

Amanda Shum, 19awss@queensu.ca

Lucy Wu, wu.lucy@queensu.ca

CISC326/ CISC/CMPE 322

Prof. Karim Jahed

October 21, 2023

**Abstract**

KODI is an open source media centre software that has become a versatile platform for digital media consumption. KODI's architecture is founded on a client-server model, which allows users to access the platform through a multitude of operating systems such as Windows, MacOS, Linux, and various embedded systems. With that, KODI offers developers an opportunity to create customized add-ons, extend functionality, and plug-ins.

KODI’s architecture places strong emphasis on its open-source nature and their community of developers. To ensure that KODI continues to be an ever-evolving platform, developers have access to KODI’s well-documented design principles and architecture. At the heart of its architecture, KODI’s modular design allows developers to create unique extensions, which enables KODI to support a wide range of media types and UI.

This report contains an extensive overview of KODI’s architecture, conveying KODI’s role as a media player application with the potential to transform devices into a digital media centre. At the highest level, KODI has a layered architecture with four key layers, divided into a Client layer, Presentation layer, Business layer, and a Data layer. Each layer primarily interacts with the layers immediately above and below it, which is critical for maintaining a clear separation of needs and concerns, which, in turn, simplifies the development process. It is important to note that since each layer has specific responsibilities, which means developers can focus independently on a specific area without the complexity of other layers.

KODI’s architecture emphasizes modularity, modifiability, a layered design, minimal dependencies, and collaboration to ensure that it remains a powerful and adaptive platform for media consumption. Developers can create and enhance features with ease while maintaining stability with no scalability concerns. KODI’s architecture balances the needs of developers whilst keeping up with being on top of multimedia software.

**Introduction & Overview**

With consumption of multimedia on the rise, it is important to understand the unique innovation that allows the multimedia platform, KODI, to stand out amongst applications for digital entertainment. The purpose of this report is to show that KODI’s appeal as a versatile free media player application extends beyond just a media player and is a powerful platform that is a great alternative home theatre software system.

At its core, KODI is an open-source software that aims to transform your home computer or smart devices into a comprehensive digital media centre. Through its user-friendly interface that serves as a dashboard to easily access and manage content, as well as customizable skins and easy navigation, users can organize and play videos, music, podcasts, photos, and other various digital media content through the platform.

One of KODI’s distinctive features is its ability to organize and manage large amounts of multimedia collections. It uses a unique method to retrieve various information about all kinds of media. In addition, KODI allows users to enhance their experience using add-ons and plugins, which is a testament to its open-source nature. One of KODI’s core functionalities is their powerful playback engine, which can handle a wide spectrum of media formats. Seamless playback experience can be done via local and cloud media.

With KODI’s efforts to give a flawless media consumption experience, it cycles back to its focus on community and development. A collaborative effort between developers and users ensures bug fixes, regular updates, and new add-on features that make KODI both relevant and at the top of media centre software. With the ability to manage a centralized database, distributed media sources, and remote access, KODI is able to open up possibilities for managing extensive multimedia libraries and delivering content to a wide audience. The conceptual structure of KODI balances a variety of elements and functionalities to become a popular application for users to consume digital media.

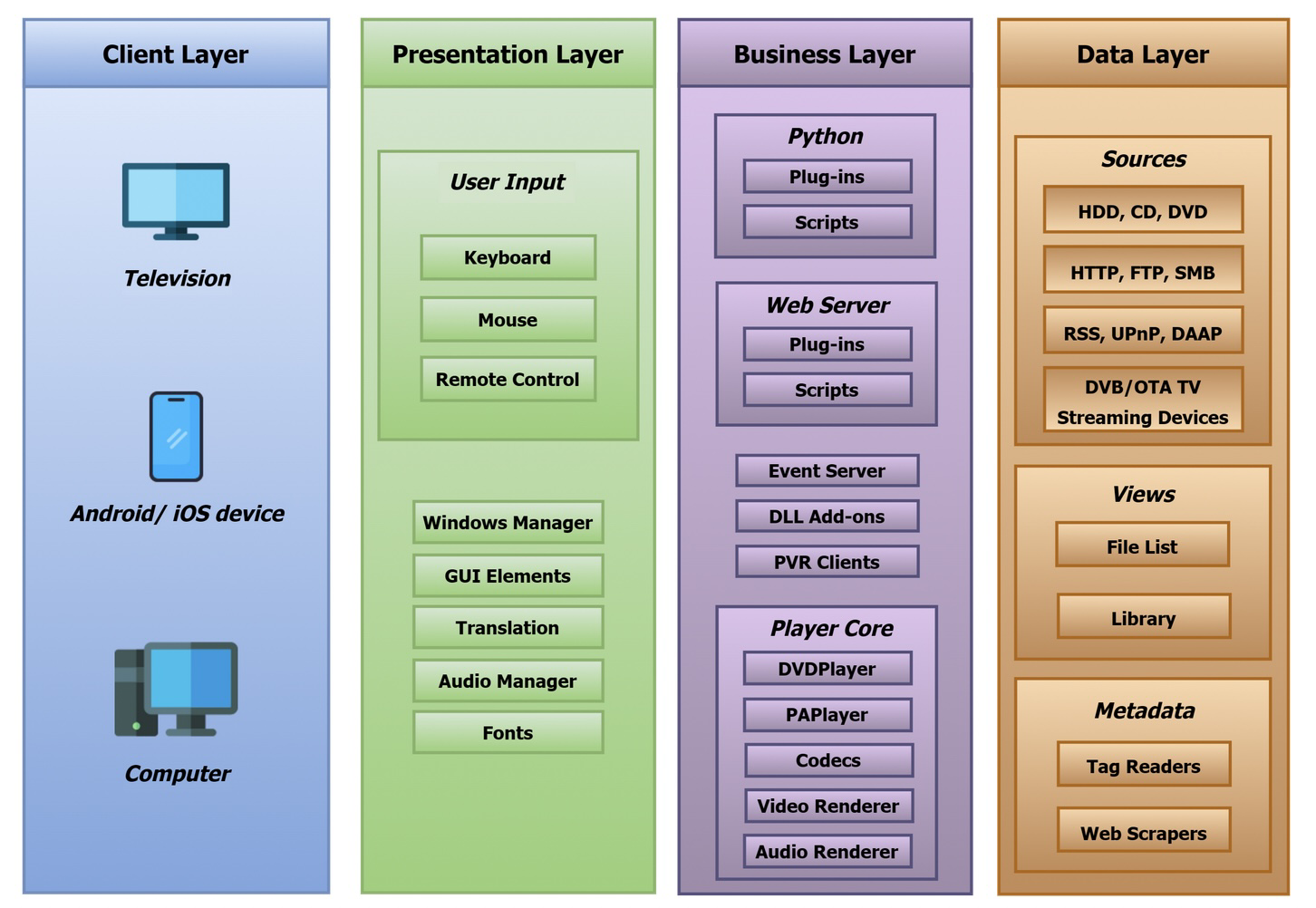
The subsequent sections of this report will look at the fundamental elements of KODI’s conceptual framework, as well as explore its architecture, use cases, and the results of our findings.

**Architecture**

As an open-source software that caters towards a wide variety of operating systems and hardware platforms, KODI’s architecture is greatly impacted by external dependencies beyond its scope. KODI has hundreds of different developers evolving its software to meet the needs of an ever-changing environment. Consequently, the importance of Modifiability and Modularity becomes strikingly evident.

Following the differentiation of architectural styles outlined by Garlan and Shaw, KODI employs a Layered Architecture: a method of organization in which each layer services the layer above it and is a client to the layer below (Module 03, n.d.). They support design based on increasing levels of abstraction.

At the highest level, KODI has a layered architecture with four key layers, with each layer interacting closely with the layers above and below it (Garlan, 1993). To further bolster modifiability of the software, KODI introduces the use of modules. These modules are independent of each other and carefully divide the layers to minimize ripple effects across the architecture. In the case that one of the modules has failures, the software can still be used (Architecting, n.d.).



(Architecture Diagram)

*The Client Layer* is the uppermost layer that directly communicates with the user. It utilizes lower layers to provide the user with interactive features. The Client Layer also acts as a bridge that allows proper intercommunication between layers.

*The Presentation Layer* is composed of all files and packages, from all of the layers, that are related to the skins, fonts, translations, and GUI of the media player. This is one of the most popular modules to modify, and so it benefits greatly from added modularity (Architecting, n.d.). It also contains modules that manage user inputs and audio output.

*The Business Layer* is responsible for most of the core functions of the program. It contains all of the most important modules that are essential to the home theatre software. It provides core functionality and logic that manages features and data. The Business Layer interacts with the user interface in that it is in charge of presenting the application to the user. This layer will take the requests translated from user input from the user interface and handle them. This layer also manages media playback by controlling the playback of various types of multimedia content. KODI’s business layer is also responsible for managing the user’s media content, which may include organization of media libraries, metadata retrieval, and the creation of a structured catalog in a way that it is easily accessible to the user.

The Add-on Module (Python / Add-on manager) consists of files and packages responsible for the integration of add-ons. KODI also contains its own Python interpreter since its custom add-ons are written in Python. These add-ons are developed by third-parties, giving ample room for errors (Architecting, n.d.). This highlights the need for isolation from the rest of the architecture. Although an add-on may fail to work, KODI as a whole will still run.

The Player Module (Player Core) concerns the reading and displaying of audio and video files using codecs. These codecs are needed to convert rough data files into what is displayed on the user’s monitor. These are imported from external libraries and developed by third parties (Architecting, n.d.), which once again can be a liability if not for the relative isolation provided by the use of modules.

Other Modules in the Business Layer may contain the Web Server module for web environments to provide the user extra ways to manage and configure add-ons and options of KODI. Finally, the Business Layer contains essential libraries which are crucial to overall features of the software.

*The Data Layer* manages media sources (files) that can be played in KODI. Source module handles the content of multimedia files: its files, functions, and packages store and locate files on HDDs, streams from external servers. Metadata module scrapes data from network protocol sources, and finds multimedia using tags or metadata (Architecting, n.d.). Finally, the View module manages the organization of file lists, and automatically creates lists or collections based on the information in the file.

As a result of the sharp boundaries between each module and layer, concepts such as scalability and maintainability are smoothly integrated into KODI’s development practices (Architecting, n.d.). Its minimization of dependencies allows changes and failures to remain isolated, making way for scalability as new servers are deployed and the number of users increases.

The system is also made easier to maintain due to its ease of modification (Module 03, n.d.). The risk of changes to the software having an adverse effect across the architecture at large are lessened. This minimized risk welcomes the safe, streamlined development of new features in the future. As such, modularity makes way for scalability, modifiability, and maintainability, which allows KODI to respond and adapt to technological advancements, such as the introduction of newer platforms, in its environment.

There are many parts of the system that are performance critical, minor cases like longer load times can cause inconvenience to the user’s experience with the software, but it rarely develops to a level that halts the software as a whole. Overall media playback process and add-ons management are some of the most performance-critical parts of the system. During playback of media, If the read speed of the file is not fast enough for a continuous play, it will cause bottlenecks that will slow the overall system; and in turn, will make the software unusable. Ineffective Add-on management, too many Add-ons, or large scale Add-ons may cause unintentional errors or inconsistent performance.

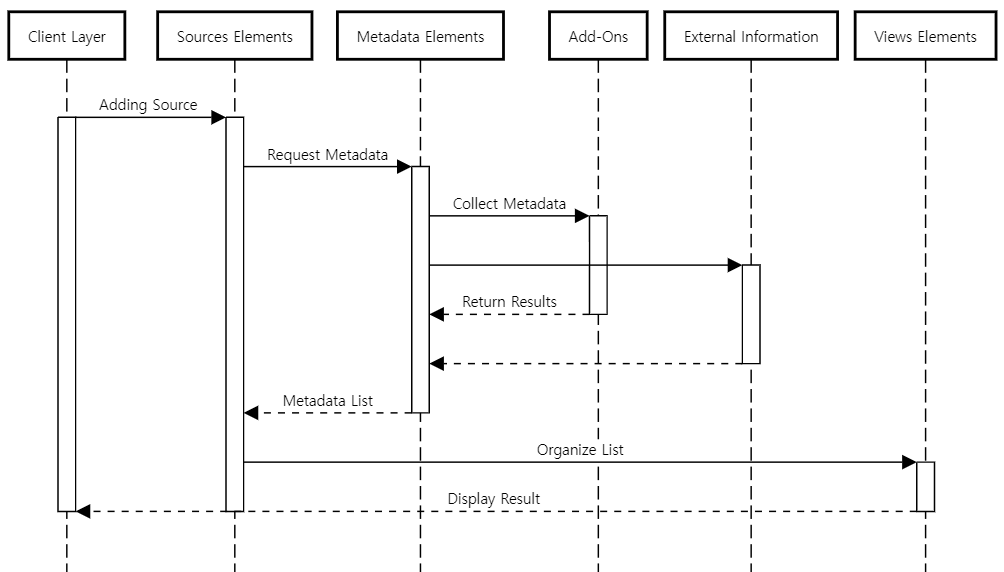
**External Interfaces:**

Since KODI is mainly used for loading media sources from user specified paths, any kind of software that manipulates and manages files might be seen as an external interface. Mainly, there are two kinds of external interfaces. As mentioned in Architecture, the Client Layer is the only place for lower layers to intercommunicate. Since all additional information and requests entered by the user are imported only through the Client Layer, it can be seen as the most crucial part of the overall data flow of the software.

Any kind of file management software could be seen as an external interface to KODI, although it is external software rather than modules or layers existing in the structure of KODI, it can indirectly interfere with media sources managed by Data Layer. Modifying file information and metadata from sources could be a few examples that can lead to the addition or change of information inside the software.

**Use Cases**

1. Adding Local Media Source



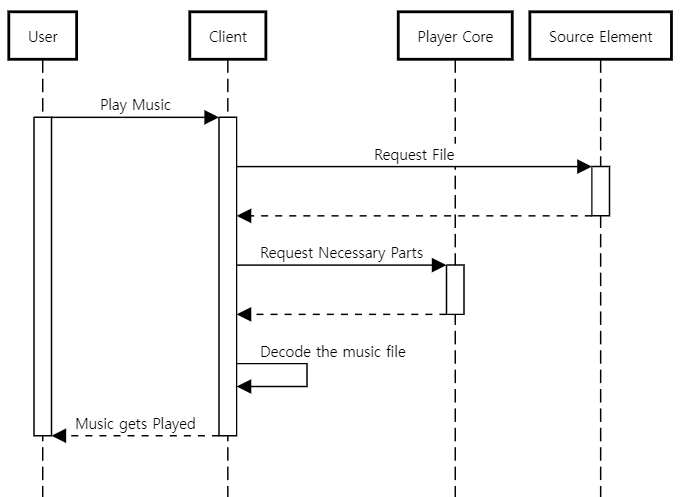
In this Use Case, we will define and illustrate the process of the user adding new local media sources. In the software UI, this is divided into Adding Source and Set Content, but most of the time, both features are used to add media sources. Thus in the current Use Case, we will define the process by which the user calls the media source after going through the two processes above.

When a user requests addition by specifying the path of a new media source as well as the name of the source through the client, the Adding Source ends after the Source Elements saves a specific path and names the path. If we move on to the next step and talk about Set Content, it is essentially a UI that allows the user to conveniently manage and utilize the metadata in the media that is contained in the media source. This section covers the collection and automatic application of metadata.

When a user makes a request by specifying information provider from the UI, the Source Elements side will request metadata from the Metadata Elements about the media existing in the current media source. Metadata Elements uses Web Scrapers and Add-ons to collect metadata from requested media at a location designated by the user, and returns the metadata to Source Elements, which applies the returned metadata to the media.

After going through the above processes, when the user calls the media source, Source Elements sends various information about the media to Views Elements, and Views Elements organizes it in the way specified by the user, and finally displays the media on the UI.

1. Music gets played from the local source



In this Use Case, we will define the situation where music is played from a local source. Once a user plays the music from a local source, the client requests user-picked music from Source Element. After the file is received from Source Element, Client requests the necessary codec / library from Player Core. After Client receives all the necessary information, Client decodes the music file on the fly then provides it to the user via hardware.

**Data Dictionary**

Metadata: Information describing the characteristics of data (Metadata, n.d.).

Codec: Hardware or software-based process used to compress and decompress large amounts of data. They are often used to play, create, and send media files over a network (Gillis, 2022).

Add-ons: Supplementary piece of the code or software that expands or improves the features of original software.

Emulator: Software that lets certain computer systems behave like other systems.

Plug-in: Software that can be added to a system to provide extra features and functions (Plug-in, n.d.).

Scalability: How well a solution to some problem will work when the size of the problem increases (Module 02, n.d.).

Modifiability: Measures how easy it may be to change an application to cater for new non-functional requirements (Module 02, n.d.).

Modularity: Technique for breaking down a complex system into smaller components (Modular, n.d.).

Open-Source: Software with source code that anyone can inspect, modify, and enhance (What is open source?, n.d.).

Web-scraper: Software that extracts content and data from a website (Scraping, n.d.)

Skins: Enable users to change a program’s overall look and feel without affecting its functionality (Computer, 2022).

**Lessons Learned**

In the process of researching the conceptual architecture of KODI, our team faced several obstacles that can be improved for future work. Although distribution of tasks were clear and concrete, we found inconsistencies in the material written by each individual. There seemed to be different methods of defining KODI’s architecture with varying levels of abstraction. In order to have consistent flow throughout the report, we would have to discuss and agree on the representation of architecture that the team should portray. To accomplish this, we must work together on a larger scale and organize all of our data on the same document.

Much of the content in the report depended on other sections to have a combined understanding of KODI’s architecture. More collaboration was needed to construct various elements of the conceptual architecture. Nonetheless, as a team, we were on track to have a detailed understanding of the conceptual architecture of KODI.

**Conclusions**

In conclusion, KODI is a Layered Architecture, employing several individual layers that each contribute to various modules. The modular design of each layer is optimal for adding new functions in the future, or for external developers to add different third-party functions through the creation of add-ons.

Furthermore, by facilitating communication between layers only at the topmost layer, the dependence of the modules in each layer was limited to the content directly inside of the layer, making the modules relatively independent.

These qualities are responsible for the admirable potential for future scalability and maintenance of KODI’s open-source software.

References

*Architecting software to keep the lazy ones on the couch*. Kodi. (n.d.). https://delftswa.github.io/chapters/kodi/

*Architecture*. Architecture - Official Kodi Wiki. (n.d.). https://kodi.wiki/view/Architecture

*Computer Skins*. Computer Hope. (2022, May 3). https://www.computerhope.com/jargon/s/skinz.htm#:~:text=Alternatively%20known%20as%20skinz%20or,program%20without%20affecting%20its%20functionality.

Garlan, D., & Shaw, M. (1993). An introduction to software architecture. *Advances in Software Engineering and Knowledge Engineering*, 1–39. https://doi.org/10.1142/9789812798039\_0001

Gillis, A. S., &amp; Lazar, I. (2022, February 15). *What is a codec?*. Unified Communications. https://www.techtarget.com/searchunifiedcommunications/definition/codec#:~:text=Codecs%20are%20used%20in%20applications,well%20as%20compression%20and%20decompression.

*Metadata - glossary:* CSRC. COMPUTER SECURITY RESOURCE CENTER. (n.d.). https://csrc.nist.gov/glossary/term/metadata

*Modular Design*. Isaac Computer Science. (n.d.). https://isaaccomputerscience.org/concepts/soft\_design\_modular?examBoard=all&amp;stage=all

*Module 02: Non Functional Requirements (NFR) – Quality Attributes*. CISC 322 Software/Game Architecture. (n.d.). https://onq.queensu.ca/d2l/le/content/820310/viewContent/4911384/View

*Module 03: Architecture Styles*. CISC 322 Software/Game Architecture. (n.d.). https://onq.queensu.ca/d2l/le/content/820310/viewContent/4911384/View

*What is open source?.* Opensource. (n.d.). https://opensource.com/resources/what-open-source

*Plug-in*. Techopedia. (n.d.). https://www.techopedia.com/definition/4324/plug-in

*Scraping*. Imperva.com. (n.d.). https://www.imperva.com/learn/application-security/web-scraping-attack/