The Conceptual Architecture of Kodi

Aidan Wolf: 20atw@queensu.ca
Aryan Chalwa: 20acd10@queensu.ca
Marcus Secord: 20ms130@queensu.ca
Nick Levy: 18npl1@queensu.ca
Ronald Sun: 20rs80@queensu.ca
Udochukwu Ojeh: 20uwo@queensu.ca

CISC 322/326: Software Architecture
TA: Eric Lams
Professor Karim Jahed

22 October, 2023

Abstract

Kodi’s layered architecture, its modular design, and the contributions of stakeholders in its development are thoroughly covered in this report. It serves as a valuable resource for both managers and developers who want to learn more about the Kodi ecosystem’s potential for development and innovation. Kodi is a versatile open-source home theater system application that works with many operating systems and provides capabilities for media playback, customization, and remote control access. The report discusses two key observations in its architecture: Kodi’s layered architecture style, as well as a more decentralized modular design that can be observed through a development view. The architectural layers and components that make up Kodi are thoroughly examined in the report. It highlights how the client, presentation, business, and data layers interact and how third-party add-ons extend the functionality. The importance of concurrency in ensuring a responsive user experience is emphasized and the manner in which external libraries and dependencies affect Kodi's architecture is discussed. This report outlines key findings, such as the importance of efficient multitasking, the need for high-resolution visuals, and the interaction of various components in delivering media playback. It also underscores the vital role of the Kodi community, comprising both experienced developers and volunteers, in contributing to the software's development and improving user experience. Understanding the Kodi architecture provides insights into potential areas for improvement and innovation. Managers can explore opportunities for enhancing code quality, supporting emerging platforms, and integrating smart home environments. Developers can consider contributing to Kodi’s open-source codebase, creating add-ons, and improving performance. This report’s emphasis on architectural documentation and modular design highlights the importance of maintaining Kodi’s accessibility and flexibility for future growth. It serves as a valuable reference for managers and developers interested in the Kodi ecosystem, its architecture, and the collaborative efforts that drive its continued development and success.

Introduction

The purpose of this report is to try and understand the inner workings of Kodi to determine a conceptual architecture. Throughout the report key concepts such as the architectural style, its components/subsystems, and the interactions between components will be examined.

The report starts by examining the layered architectural style of Kodi. The architecture consists of 4 layers, which are the data layer, the business layer, the presentation layer, and the client layer. The Data Layer contains everything in reference to functions, files, and content management packages, and it consists of source elements, views elements, and metadata elements. The Business Layer provides a client that creates a connection with a streaming server, and it contains three modules. The Web Server Module, the Python Module, and the Player Core Module. The Presentation Layer of Kodi's architecture consists of four subsystems, the User Input Module, the GUI Elements, the Audio Manager, and the Fonts Module. All of the formerly mentioned layers lead to the Client Layer, where the user interacts with Kodi through the chosen input device.

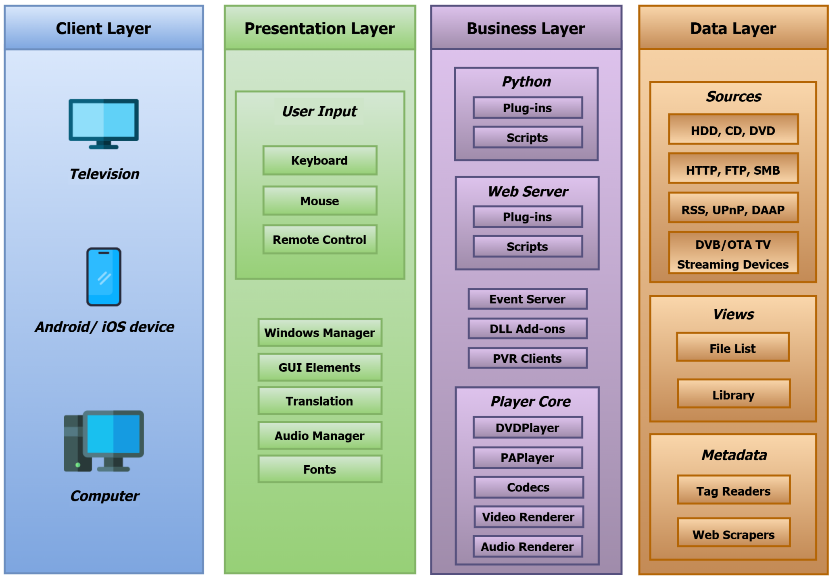
Following the architectural style is the evolution of Kodi. Despite its layered style which can be hard to evolve, there are still ways that Kodi can evolve. This section discusses the complexity of evolution as well as possible ways Kodi could evolve. Through dropping support of legacy devices, code size and complexity can be reduced and allow for new innovations to be made. The business layer can evolve through machine learning to analyze user behaviour to deliver personalized recommendations to community created features. These are just a few examples discussed within this report of how Kodi could evolve.

The next part of the architecture examined is the data flow. Using a development and functional view we can see how different components interact. Starting with the development view, Kodi can be broken down into 5 main modular building blocks that have interacting components within themselves. These blocks are skins (takes care of what the user sees and does), interfaces (API and add-ons), content management (handles media sources and related data), player core (renders and plays the media sources), and file sharing (networking). Using the functional view some interactions between blocks can be seen. For example, the content manager and player core interact as the player core needs the media source from the content manager to work.

The concurrency of Kodi is also discussed. Concurrency within Kodi plays a major role as Kodi needs to be able to play videos without constant buffering while still maintaining good resolution to ensure the user experience is optimal. This section not only discusses the concurrency of main components but also how add-ons fit into this.

The final part of the architecture examined is the way in which multiple developers collaborated in the creation of Kodi. Kodi is an open-source software and relies on developers voluntarily contributing to the project. But to ensure its guidelines are upheld, only a select group has the authority to accept push and pull requests. Not only are the roles and tasks mentioned, but also the architecture style and how it influenced the development of Kodi.

Kodi is a home theater system application for Android, Linux, BSD, macOS, iOS, tvOS and Windows operating systems.[1] It's a media player for music, movies, tv shows, photos, games, and more.[2] As a home theater system, it also allows customizability and remote control access for the user.



The Kodi wiki architecture consists of 4 layers: client, presentation, business, and data.[3] The client layer is the system in which the user is running the application from. As mentioned previously, most operating systems can run Kodi. The presentation layer handles user information display as well as user input. The user input can be via keyboard and mouse or remote control. The other modules included in the presentation layer include: the windows manager module; GUI elements, which includes the visual components and design for the program; the translation module, to support multiple languages; the audio manager module, to configure the audio settings for the system such as volume and output; and the fonts module, to configure the appearance of the text. The business layer essentially provides the core functionality for the application with the Python, web server, and player core modules. The Python module allows users to create add-ons and apply them to Kodi. The web server module allows content to be managed through the browser or other applications. The player code module provides the core functionality to the user: playing the requested media content for the user in an optimal manner. The final layer, data layer, contains the file and content management components. The three elements mentioned in this layer are source elements, view elements, and metadata elements. Source elements allow the configuration of the source for Kodi which could be a data storage, network protocol, or streaming device. The views elements allow configuring how the desired content can be played. Finally, the metadata elements obtains additional data on the content to enhance user experience which is obtained through tag readers and web scraping.

[1] https://github.com/xbmc/xbmc/tree/master#readme
[2] https://kodi.tv/
[3] https://kodi.wiki/view/Architecture

These layers, though separate, interact in order to make Kodi function. The device which the user runs Kodi - client layer - incorporates the user input in the presentation layer and the sources in the data layer. Furthermore, the version of Kodi the user sees, whether it be skin, language, or font, is determined by the presentation layer. Finally, the core functionally is presented by the business layer via the selection of media on the data layer.

The architecture for Kodi is a layered architecture, which is hard to evolve as changes in the requirements can impact all neighbouring layers, and given there are not a lot of layers, evolution is hard without redesigning. That being said, the product can evolve in each of the layers mentioned above, and more if non-functional requirements are touched on.

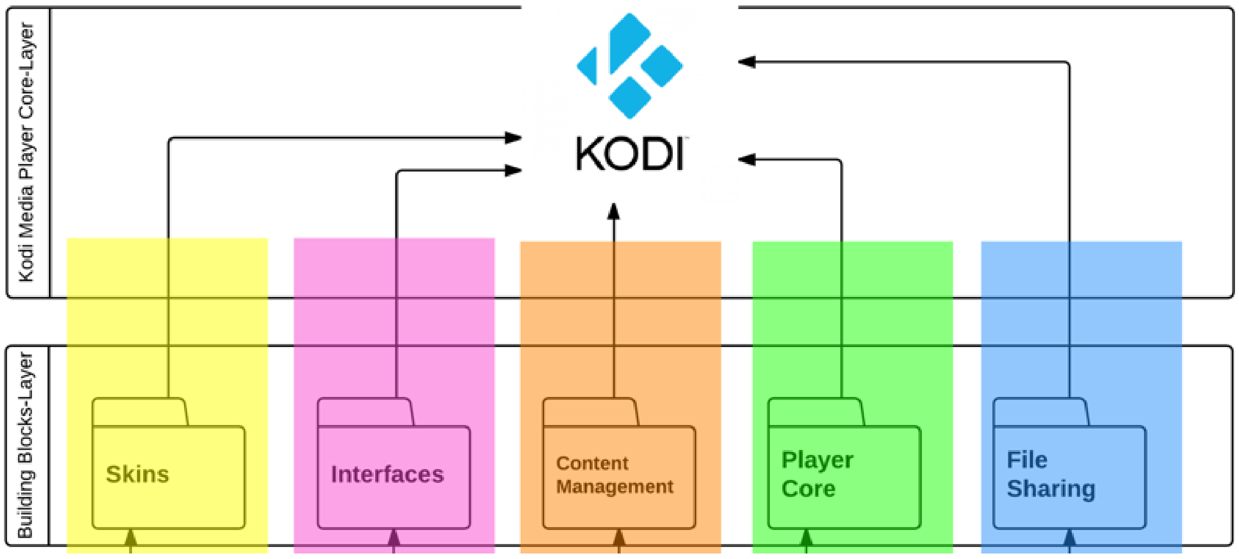
The client layer and presentation layer has a focus on enhancing support systems and user experience across all platforms, including emerging ones. From this perspective, Kodi can aim to improve code quality and stability by dropping support for legacy devices. For example, the software still supports OS like Windows 7 which is hardly used, and deprecated by most softwares nowadays. Maintaining these older softwares broadens the audience but at the cost of complexity, as such, carefully reducing code size will enhance maintainability, and room for innovation. Some of the most used third party tools can then be implemented into the core Kodi software for a better user experience, and they would not be missing out if they do not know about the add-ons.

The business layer can continuously evolve through supporting community add-ons, plugins, and scripts, as long as it can handle it. Machine learning could be a new implementation to analyze user behaviour to deliver personalized recommendation to community created features. New media types could also be incorporated, most recently the games section was added to the core. A type that has not yet been incorporated are short form media, such as Tiktok, Instagram Reels, YouTube Shorts, etc.

Kodi can look into integration with Smart Home Ecosystems as it is becoming ever more popular. Smart home systems allow users to control things like lighting, temperature, and devices like Alexa. This would require the business layer to accommodate IoT, or Internet of Things, which is defined as a network of connected devices and the technology that facilitates communication between devices. Kodi can learn from common examples like playing Spotify music through voice control on another device.

The data layer component contains the sources to allow Kodi to function. Some of the data could be expanded to cloud-based storage to enable access to media across the multiple different devices, additionally allowing users to free up space on their devices. However, this could cause security problems and privacy concerns. As such, blockchain could be implemented to verify the authenticity and integrity of the media that is being downloaded, enhancing security and privacy.

One of Kodi's general guidelines is that Kodi should still compile and run if a non-essential module/library is disabled or removed. The modules serve as building blocks for the total architecture, that are interacting with each other and distinguished based on the functional activities they perform. Each of these modules are classified into functional groups which are independent from each other. Using a development view, there are five main modular building blocks inside of Kodi's architectural layers that work together to ensure the application's functionality. These are skins, interfaces, content management, the player core, and file sharing. In regards to the **Skins Block**, the skinning engine of Kodi determines the way user input, fonts, the audio manager, the translator, the window manager, and the GUI are perceived on the users end. The skinning engine is interacted with through the users mouse, remote, and keyboard inputs/activities. Regarding the **Interfaces Block**, the event server, web server, Python, stream clients and add-ons work together to execute the HTTP API, web interface, Python scripts, and Python plug-ins. The **Content Management Block** includes media sources (hard disk, DVD/CD, HTTP, DVB), metadata (web scrapers, tag readers), and views (files and libraries). These subsystems work in unison to manage content for Kodi's usage. The DVDPlayer, PAPlayer, Audio/Video renderer, and codecs execute concurrently to make up the **Player Core Block**. Finally, the Universal PnP Server, and HTTP/FTP/RSS streaming enable the **File Sharing Block**. Each of these modular building blocks function independently from each other (there are no overlapping subsystems between blocks), but they all communicate between each other on the highest layer of Kodi's modular design, which is the Kodi Media Player Core-Layer.



Kodi is supported on 6 different (operating) systems (Linux, Windows, OSx, Raspberry Pi, iOS/Android). To support multiple operating systems, Kodi requires these operating systems to abstract communication with the hardware away. To support all these systems, sometimes external libraries are used to enable additional features. Kodi makes use of add-ons to enhance the functionality of Kodi, they are written in Python and stored in the Add-on database. Third party developers can use the Kodi API and Python to create their own add-ons. Libraries are used to store user entertainment content. This content can be stored on the user's device or an external drive, and can be streamed/played through Kodi's communication protocols. Kodi has some external dependencies that impose limitations on the architectural design, such as the power of devices it runs on, and rendering API's.

When looking at Kodi from a functional viewpoint, we can see some of the interactions between the components. Starting with the most important feature of video playback we have the player components. There are two main types of player components, video and music players. These are the components that let the user actually play/open its content. Due to the importance of this feature, the players use the accessible GUI so that it has a user friendly interface. The players also make use of other features such as the codecs. But for these players to work, they need data for them to play. There are 2 libraries that correspond to the players, the video library and audio library. These two libraries are where the video and audio data are stored. The libraries make use of communication protocols to ensure that multiple devices have the ability to access the content. The web scrapers component also uses the libraries. This component makes use of the library by storing additional information about the content the user has watched. Another component that interacts with the libraries is the recording feature for live television. Once recorded it is stored in the libraries for video playback. Live television also interacts with the 2 players and the accessible GUI so that it is user friendly.

Kodi's software is very efficient when it comes to multitasking. Although the software's building blocks are separated due to the layered style, there are still many cases where concurrency is required to achieve high responsiveness and functionality in certain components. Below we will highlight some of the areas where Kodi's components require concurrency.

User-Interface is a major component of all media platforms, and they should be responsive even when other components are being utilized in the background. For Kodi, this could include accessing the database, loading add-ons or extensions, downloading content and more. Kodi must ensure that during any of those processes, the user must be able to navigate a perfectly smooth user interface.

Kodi incorporates add-ons to increase the applications functionality and in addition, allow third party members to create their own add-ons. These are written using the Kodi API alongside Python, then are to be stored in the Add-on database. Many of these add-ons would require some sort of background operation and would need to communicate with Kodi's main system. Concurrency here would ensure each Add-on does not decrease overall performance while running, and ensure the Add-on is provided memory/system allocation. For memory, Kodi uses libraries to store its media and user information (e.x. Recently watched). To achieve top efficiency and responsiveness, Kodi ensures during any process these libraries can be accessed, written to and/or edited. This is crucial to preventing delays in the UI or software itself, as many components need to access these libraries.

As a media and streaming platform, it's imperative to provide top notch resolution and visuals for the user. Streaming tasks, buffering, decoding, and actual playback might run concurrently to ensure smooth playback experience. For instance, while a video is playing, Kodi might be buffering the next segment or processing audio in parallel. In addition, Kodi runs various services in the background, some related to core functionalities and others related to plugins. These services might include update checks, scraping new content, or maintaining libraries. Running them concurrently with efficiency ensures that they don't interfere with the main user experience.

Kodi Media Player, previously known as XBMC player, is an open-source theater/media player that XBMC Foundation developed, comprised of a large group worldwide, mainly on GitHub. Stakeholders have a strong contribution and influence as they work under the direction of board members. These board members are primarily former developers elected within the community due to their deep understanding of the architecture. They also survey the software, solve bugs and communicate with users. The stakeholder group comprises experienced developers; however, there is a very large community of high-level engineers, maintainers, translators and developers who work to improve the software voluntarily as they have other jobs to attend to. In the search to determine these stakeholders in Kodi, deep analysis of the GitHub contributions shows a select group with the authority to approve push and pull requests.

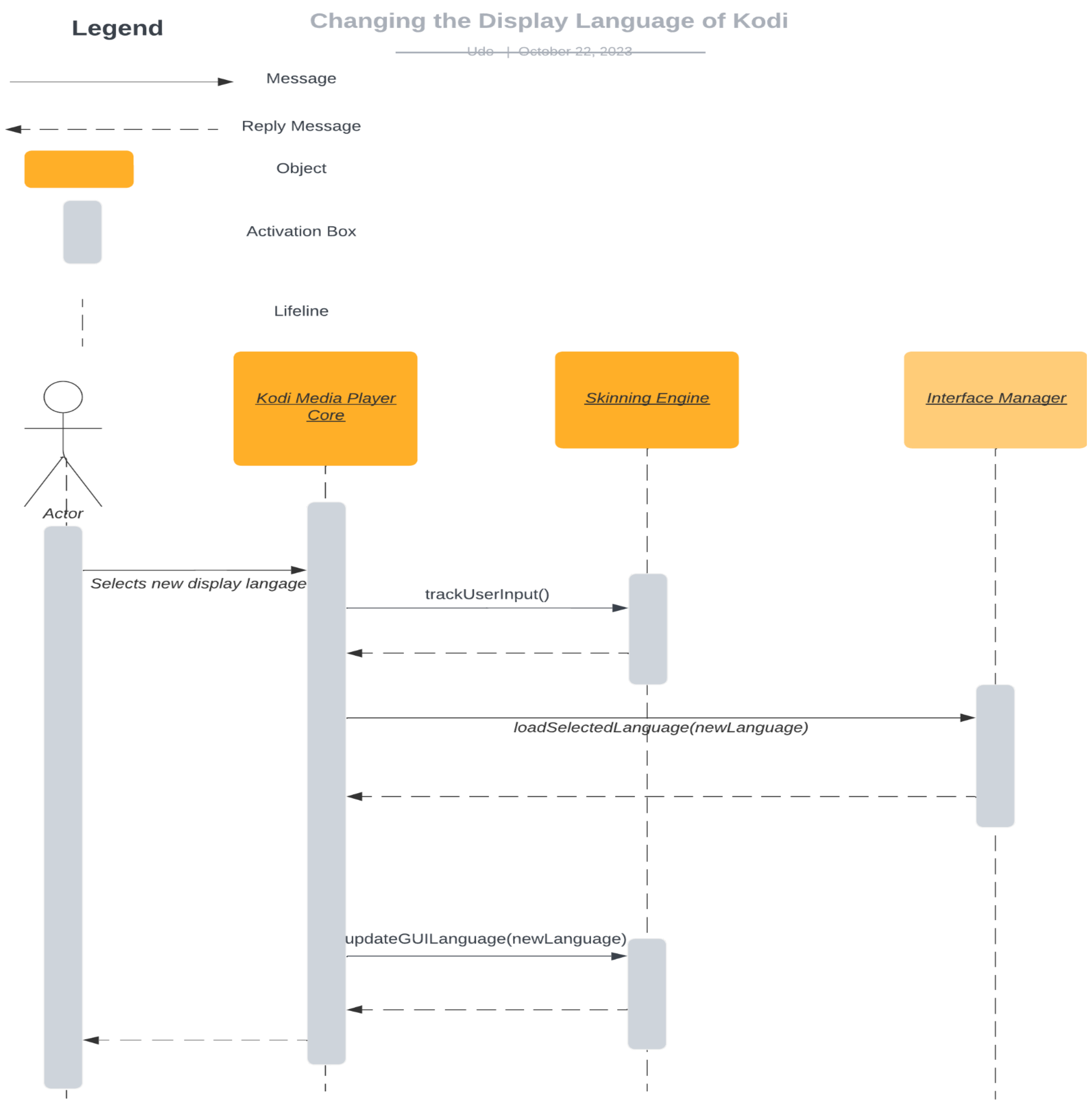
Some developers use user-provided logs to identify bugs, create tickets, fix and close the requests; they also keep frequent communication over whether or not to add additional features via the forum. Kodi's modular design improves the maintainability of the code and allows for the software to be used and developed even if a module is to fail. This is attributed to the modules all being functional groups independently. This large community of passionate tech employees has been invited to contribute to Kodi with the goal of improving the software for the greater good of the users. This tech community also makes up a big chunk of the user base. This voluntary group stands as a key factor in the development of Kodi. The accessibility to the framework allows Kodi to operate on the scale it currently does, but it is built upon strong guidelines to ensure that contribution follows some rules.

- **Code documentation**
    - This should be done by using Docbook or Doxygen (preferred)
- **Installation**
    - Be easy to install, set up and maintain, so that our valuable end-users do not get fed up with it and quit
- **Modular design**
    - Kodi should still compile and run if a non-essential module or library is disabled or removed

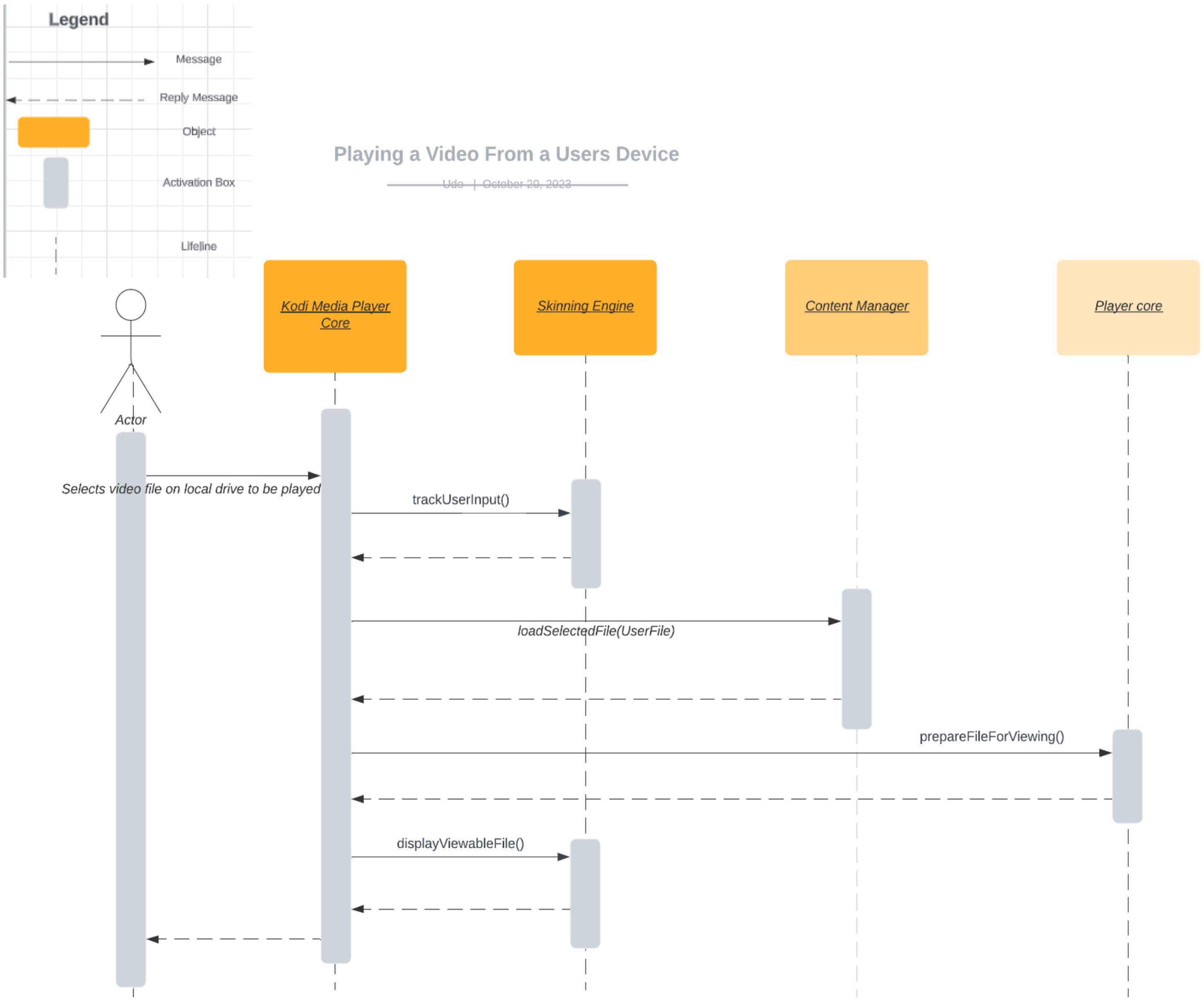
The GitHub-issue tracker is the tool Kodi uses to connect users to software developers to find and fix bugs. It is used widely across the platform to help keep tabs on the overall project and ensure technical issues are resolved.

Kodi's support for multiple platforms does not complicate development due to the clean architecture; developers with proficiency in C/C++ can work across operating systems, with add-ons primarily being Python and XML. The draw to contribute to Kodi is attributed to its open-source architecture that welcomes developers; along with a large community of users actively contributing their thoughts on potential features, add-ons, translations and the application as a whole via the development forum. Kodi operates upon a community basis of develop-user interactions that bring together to communicate in order to help and improve. This allows skilled software developers to focus on fixing issues in a large system within their focus, such as GUI or add-ons, in their free time. Due to the fact that Kodi is built upon the support of willing developers and an engaging community, they have tech companies that sponsor the platform.

Diagrams and use cases

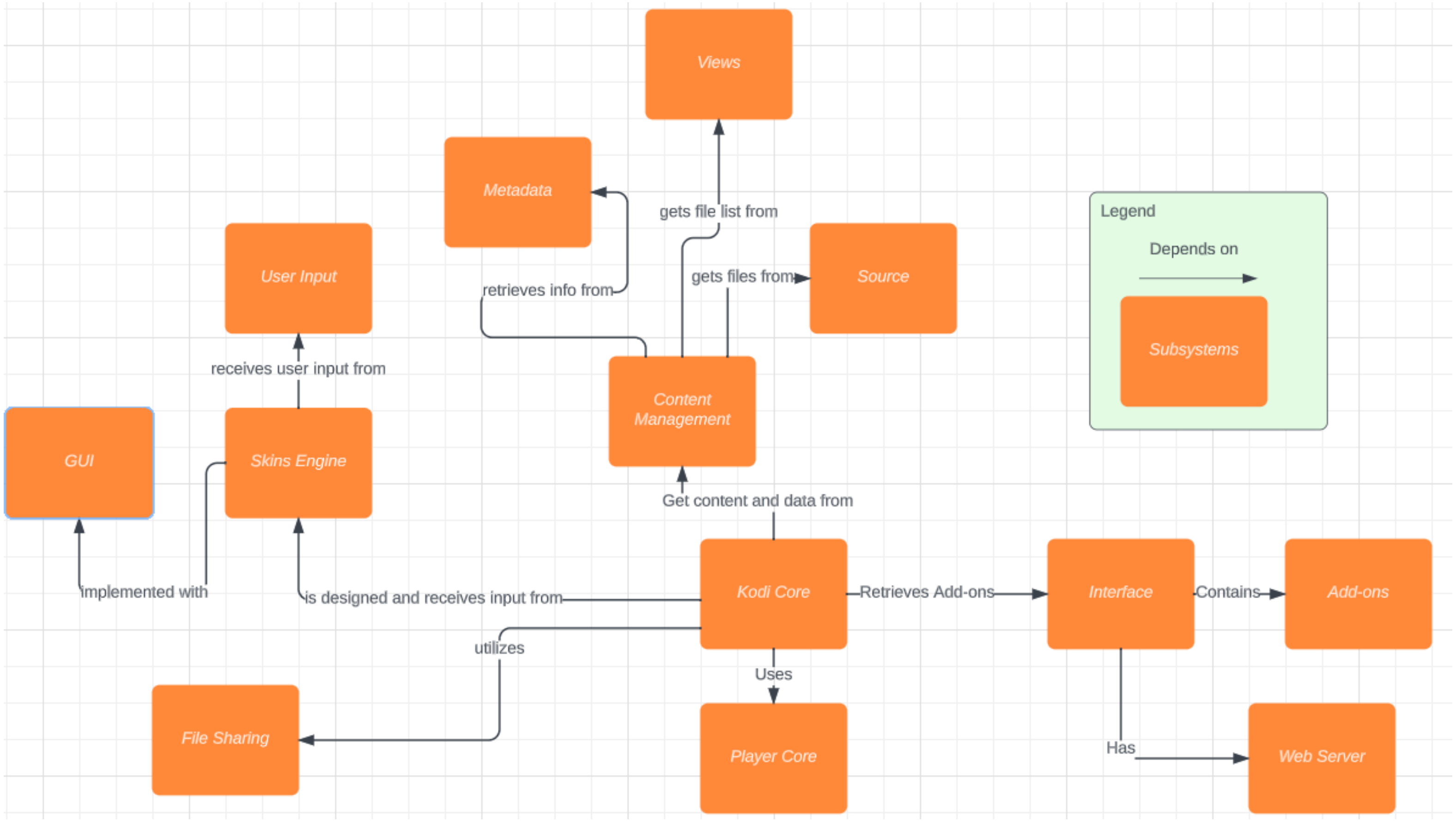

Legend
Changing the Display Language of Kodi
~~Udo | October 22, 2023~~
Message
Reply Message
Object
Activation Box
Lifeline
Kodi Media Player Core
Skinning Engine
Interface Manager
Actor
Selects new display langage
trackUserInput()
loadSelectedLanguage(newLanguage)
updateGUILanguage(newLanguage)

Kodi is first and foremost a media player. So one essential use case is playing some form of media that is stored on the user's device. The sequence diagram above represents the use case of a user selecting a file from their local drive to be played. It starts first with the actor using peripheral devices to select a file from their local drive. The Kodi Media Player Core (KMPC) uses the skinning engine, which has subsystems to track user input, to take the user's input, in this case the file they selected and returns it to the KMPC. It then uses the content manager, which is responsible for managing files, to load the file. The KMPC then uses the player core to process the file. The resulting file is then displayed using a GUI element.



Kodi has users across the world with varying linguistic necessities. The above sequence diagram represents a use case where the user wants to alter the current display language to their new preferred language. Using their peripheral input device, the user inputs the new language that they want Kodi to display. This gets passed on to the KMPC, which is the highest layer of Kodi's architectural design where all of Kodi's architectural blocks communicate with each other. The KMPC transfers these inputs to the Skinning Engine, which is responsible for tracking user inputs, which then returns the tracked information (selected display language) from the user back to the KMPC. The selected display language is then sent to the Interface Manager, which holds the language add-ons for Kodi. The Interface Manager loads the selected display language add-on data, and sends it back to the Kodi Media Player Core. Finally, The KMPC sends the new display language data to the Skinning Engine, as it is responsible for updating the GUI to the new display language. This updated GUI is then sent back to the KMPC, where it is subsequently displayed to the user in their selected language.

Dependency diagram of the overall application:



Conclusion

Overall, this report provides a comprehensive overview of Kodi's architecture, shedding light on its layered structure and the key components that make it a versatile home theater system. Kodi's architecture is a well-structured layered design consisting of four layers: client, presentation, business, and data. Each layer plays a crucial role in ensuring the smooth operation of the application, and their interactions are vital for delivering a seamless user experience. One of the notable lessons from this exploration is the challenge of evolving a layered architecture. Changes in requirements can have far-reaching impacts, and Kodi's relatively few layers make it challenging to introduce changes without extensive redesign. However, the report also highlights the potential for evolution within each layer. From a development perspective, Kodi's commitment to ensuring that the software can compile and run even if non-essential modules or libraries are disabled is a useful practice, as this modular design contributes to the software's resilience and maintainability. Kodi's support for multiple operating systems and devices is both a strength and a weakness. While broad compatibility ensures a wide user base, it can introduce complexity, such as the need to maintain support for legacy systems. The lesson here is that balancing compatibility and complexity is a continuous effort, and it's important to periodically assess the value of supporting older platforms. An additional lesson that can be extracted from this conceptual analysis is that Kodi's success is heavily dependent on its active and engaged community of developers and users. Maintaining strong community involvement is critical for the long-term sustainability and development of open-source projects. In conclusion, the architecture of Kodi is a complex yet highly functional system, and its potential for growth and adaptation is significant. The exploration of Kodi's architecture has provided insights into various aspects of software development, from architectural design to open-source community engagement and the importance of a user-centric approach.

Data Dictionary

IoT: Internet of Things, refers to the collective network of connected devices and the technology that manages communication between devices and the cloud, as well as between the devices themselves.

KMPC: The Kodi Media Player Core, which is the highest layer of Kodi’s modular design, where each of Kodi’s modular building blocks communicate with each other.

Layered Architecture: An architectural style where the main idea is that modules or components with similar functionalities are organized into horizontal layers, and each layer performs a specific role within the application.

Modular Design: A design principle that subdivides a system into smaller parts called modules which can be independently created, modified, replaced, or exchanged with other modules or between different systems.

Naming Conventions

KMPC: Kodi Media Player Core

GUI: Graphical User Interface

XBMC: Xbox Media Center

UI: User Interface

API: Application Programming Interface

DVB: Digital Video Broadcasting

HTTP: Hypertext Transfer Protocol

FTP: File Transfer Protocol

RSS: Really Simple Syndication

IoT: Internet of Things

References

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

File:XBMC architecture overview schematic.png - Wikimedia Commons. (n.d.).

https://commons.wikimedia.org/wiki/File:XBMC_Architecture_Overview_Schematic.png

Kodi Wiki. Official Kodi Wiki. (n.d.). https://kodi.wiki/view/Main_Page

Open Source Home Theater Software. Kodi. (n.d.-b). https://kodi.tv/

Xbmc. (n.d.). *XBMC/XBMC: Kodi is an award-winning free and Open Source Home*

Theater/Media Center Software and entertainment hub for digital media. with its beautiful interface and powerful skinning engine, it's available for Android, BSD, linux, macos, IOS, tvos and windows. GitHub. https://github.com/xbmc/xbmc/tree/master#readme