Repos and Repose:
Kodi's Conceptual Architecture

Group 2: ArchAngels

Bianca Bucchino: 20blb@queensu.ca

Adam Clarke: 21asc6@queensu.ca

Christian Higham: 20csdh@queensu.ca

Omar Ibrahim: 20omha@queensu.ca

Owen Rocchi: 19omr6@queensu.ca

Aidan Sibley: 20ajs18@queensu.ca

October 22nd, 2023

Abstract

The KODI system represents a powerful open-source software solution, functioning as a versatile media player and entertainment hub. Its foundation lies in a flexible and modular architecture, comprising four core components that drive its functionality: the user interface component, platform agnosticism/cross-compatibility component, user customization component and the input handling component. KODI makes use of the coexistence of three distinct architectural styles: layered style, repository style, and publish and subscribe style. KODI's architectural hierarchy encompasses a client layer, business layer, and data layer. The system is highly extensible, allowing the development of add-ons. These additional modules enhance the user experience by introducing new features and functionalities, all while adhering to the modular structure of the architecture. Incorporating a core player, platform component, user interface, and web server enhances system performance, ensuring smooth media playback, platform compatibility, user interaction, and network sharing capabilities. External interfaces play a pivotal role in KODI's functionality, allowing information exchange with the external world. The external interfaces included are the user interface, files and media, a database and hardware. Moreover, KODI's highly portable nature allows users to enjoy its feature-rich environment on a multitude of operating systems, thereby granting accessibility without confinement to a single platform. This adaptability cements KODI as an inclusive and user-centric media system.

Introduction and Overview

Purpose of the Report

The purpose of this report is to provide a comprehensive overview of the conceptual architecture of the studied system, Kodi. Kodi is an open-source software media player and entertainment hub that can be accessed on numerous operating systems such as Linux, OSX, Windows, iOS Android and tvOS. The software allows users to play and view videos, podcasts, music, games, and other digital media files from local and network storage media, and the internet. This report was created to provide a clear and structured way to communicate the high-level design of the Kodi system to any stakeholders involved: including the developers, project managers, clients, and other team members. It provides an outline to ensure that everyone involved shares a common understanding of the system’s architecture before the implementation of the system. The conceptual architecture acts as a blueprint for the design and development process that will guide the overall construction of a system. The report explains how the systems architecture supports scalability and future growth, and provides insights to how the system can change overtime to

accommodate changing requirements and new features. Additionally, by outlining architectural considerations early in the development lifecycle, the report helps to identify and mitigate any potential risks associated with the system's design. It enables developers to address and manage any issues, design flaws or inconsistencies before they occur, which can save significant amounts of time during the implementation process. The report ensures that the proposed architecture aligns with the stated requirements and objectives of the project. This alignment is critical for meeting client expectations and for delivering a system that satisfies its initial purpose. Lastly, it provides a well-documented reference for the systems architecture which promotes transparency, accountability, and the quality of a system's architecture. The report will include Kodi's major components, their interactions and how the architecture supports the system's goals and non-functional requirements such as evolvability, testability and performance. The report aims to accomplish a clear understanding of the system's functionality, architecture style, global control flow, potential for future changes and performance considerations. To do so, the report will include relevant diagrams and use cases to aid in visualization and understanding of the system's architecture.

Organization

This report is organized into sections to provide a clearly structured and in-depth analysis of the system's conceptual architecture. Following the introduction, we have the following sections:

1 . Architecture: This section will give an overview of the overall structure of the system, describing its major components, subsystems, and database. The section is broken into 3 parts. The first part outlines the choice of architecture style for the Kodi System. It will review three different architecture styles: layered style, repository style and publish-subscribe style. The second part outlines performance-critical components of the system along with emphasis on how the architecture supports future changes. The last part outlines global control flow such as units of concurrency and methods for passing control between components.

2. Diagrams: This section will use diagrams to illustrate the system's structure. It includes a dependency diagram which showcases the systems subsystems, modules, and their dependencies. It will also include additional UML diagrams which aid in visualizing Kodi's architecture style. Clear legends are associated with each diagram for easier understanding.

3. External Interfaces: This section will list and describe the external interfaces of the system. It will enumerate the kinds of information exchanged to and from the system through graphical user interfaces (GUIs), files, databases, messages, or networks. The focus will be on information content rather than specific details.

4. Use Cases: This section will present essential use cases that illustrate how the system is used or how a proposed feature is employed. There will be two descriptions of essential use cases which will demonstrate how the architecture is activated and utilized as well as their interaction with the architecture.

5. Data Dictionary: This section will include a glossary that defines key terms used in describing the architecture.

6. Naming Conventions: This section will list any specific naming conventions and abbreviations used in the architecture.

7. Conclusion: This section will include a summary of our overall report. It will discuss key findings and proposals for future modifications or enhancements for the architecture.

8. Lessons Learned: Lastly this section will discuss any notable experiences or insights gained during the study as well as what could have been done differently in the report.

Salient Conclusions

The report will provide a detailed analysis of the conceptual architecture of the Kodi system, aiming to highlight key aspects of the conceptual architecture. Post reviewing the report, readers should have a clear understanding of the architecture styles which characterizes the overall system and its various parts, and the system components which describe the major components of the system including subsystems, modules, and their interactions. Understanding of performance considerations and non-functional requirements should be clear as well as the explanation of external interfaces used for data transmission. Through visual aids, the use cases should be understood as demonstration of how the system's architecture is activated and used in the real-world. Overall, readers should inherit comprehension about the high-level concepts and design considerations that contribute to the success of a system's architecture.

Architecture

Overall Structure

Introduction

The overall structure will lay the groundwork for conceptual functionality of the high order systems and their interactions within the context of functional and nonfunctional requirements. This will lay the groundwork for further discussion of

more specific detailed sections of lower order systems that will be built throughout this report.

Conceptual High-Level Overview

Kodi's architecture will have a layered organization of components split between three fundamental layers, each of which will serve a distinct purpose and house a set of distinct sub-architectures to deliver system wide functionality that will fulfill the desired requirements.

Client layer

The Client layer will serve as the primary architecture for delivering the means and methods of delivering the content to the user. To do this, the sub-architecture will consist of 4 main components. The **User Interface** will be the graphical user interface that the user will use for navigation and control of the software. There will be a **User Customization** component to allow the user to customize their experience within the User Interface component itself, such as language, themes, and add-ons. Speaking of **add-ons**, this will in itself be a sub module to the greater user customization component, and will consist of an architecture to allow for the addition of scripts and code to change various features of Kodi. Lastly and arguably most importantly we have the **Input handling** component, which will serve as the logic that handles the effects of user interaction within the user interface component, and permit the passage of relevant data to the next layer, the Business Layer. This component's sub architecture will be divided into different controllers or components that connect to the Business Layer at points relevant to the location of user input, and direct the input events to the relevant component in the Business Layer to handle these events.

Business Layer

The Business layer will be the layer to implement what is colloquially known as the business logic of a software solution. This layer directly interacts with the client layer via the Input handling component to deliver the computation and calculation to the client layer when requested. This architectural layer itself can be divided into its own sub-architecture composed of 5 main components. Starting off with the **Event Handling** component, which is the most important within the Business layer to disseminate user inputs in the client layer and handle those different events within the business layer. This component will directly communicate with the input handling in the client to disseminate changes. The architecture of this component will compose itself of sub-components that handle specific input cases from the user, each of these components will trigger a change in the following components that make up the broader business layer. A way to visualize the event handling component would be to view this as a wrapper on all other components, where events in other components are directed by event handling. There will also be a component to ensure **Platform Agnosticism/Cross-Compatibility**, this will ensure

that Kodi will be able to run on multiple different platforms. Moving on to the other components that are handled by the Event handling component we see that the **media playback** component controls the playback of media types. This component will react to events such as open, pause, play, seek, and more. This component will provide the logic to control these incoming requests. Next we have the **Add-on Management** component which serves the responsibility of implementing changes to the business logic in places where the addition of changes to the Add-ons in the client layer are specified by the user, when specified, the add-ons will be managed here to disseminate changes on a case by case basis. Next we have the **Streaming** component to handle the logic required to stream the videos in multiple different formats. This sub-architecture is related to the media playback component, as when specified by events triggered to the media playback component. One of the most critical sub-components of the Streaming component will be the **I/O** sub-component which will communicate with the Data layer and receive input from the Event Handling component to fetch and then relay the media to the media playback component.

Data Layer

The most fundamental of all layers to the operation of the business logic, and in particular the media playback, the Data layer will provide the underlying means of storing and retrieving content, this layer is the input output engine that drives the business logic to complete its requirements. This layer will be made of four critical components each of which will contribute to the overall functionality. We have the **Data Storage** component, which in the case of most of the operating systems will be the host computer's file system as specified by the user component Platform Agnosticism/Cross-Compatibility. Depending on the OS, this data storage component will act differently. The next component is the **data retrieval** component, which as the name suggests will take the specified file, and depending on the specifications of the Data Storage component, will be an operating system specific system call to fetch the relevant binaries. The next component will be the **File Management** component, which will act as a sub component of the data storage component, that will ensure the functionality of enabling the user do make changes to the native OS's files such as modification, addition and deletion, and in the event of no native OS, provide the basis for a file structure. Finally and arguably the most important component will be the **Streaming Controller** structure which will be responsible for the conversion of raw binary to playable content which the user eventually interacts with. This is critical as it acts as the transformation of data to human interactable/ readable format and will involve the use of industry standard video buffering, and streaming practices and algorithms.

Architecture Style

Layered style

The first and most obvious architectural style present in the Kodi architecture is the layered style. The architecture contains distinct classes of services at varied levels of abstraction, organized in a hierarchical order. On the Kodi wiki they list 4 different layers as the client layer, presentation layer, business layer, and data layer. See diagram 2 for further details. We can further classify these as tiers, where the client layer is tier 1, presentation and business layers make up tier 2, and the data layer is tier 3. Starting from the bottom, the data layer focuses on content sources and any information related to them. The data layer also contains the view module, which controls whether content is represented as a traditional file view or sorted into different types of media. The business layer provides all functionality in displaying media to the user. Different modules of the business layer allow the management of add-ons, web server functionality, and the player core module. The player core controls playback, with the emphasis on acting as the link between multimedia and the hardware/software used to play them. The presentation layer works to display all these before-mentioned functionalities. This includes the ability to change options such as language and fonts through both the translation and audio manager module. Lastly the client layer which describes the devices in use by the user to run the Kodi software. Separating the architecture's functionality into layers allows the system to be more easily developed, and additionally be more easily understood.

Repository style

A significant style used in the architecture is repository style. This immediately presents itself in the purpose of a media player's function. The file storage system functions as the repository, and the media player defines various functionality (operations) that are derived from the media files. This is a perfect solution for this product as repository style is an effective way of storing large amounts of data. The data layer is where we can consider the repository to be located, whereas the business layer implements the operations that are run on the data. Different examples of operations include streaming audio, streaming visuals, and removing or adding data to the library. Kodi displays two different libraries for videos and music to the user, but they are stored in the same location at the hardware level.

Publish and Subscribe Style

The third architectural style that presents itself in Kodi's architecture is the publish and subscribe style. This style is used specifically to implement Kodi's add-on feature, where users can install add-ons that extend existing software functionality. There exists common community developed add-ons, as well as those that are developed by third party developers. This feature sets Kodi apart from other media players because of the add-on feature, as it allows users to tailor their experience

exactly to their liking. For example, a popular existing add-on is one that automatically downloads new episodes of a show to a user's library once they're posted online. Add-ons communicate independently of one another unless they're explicitly instructed otherwise. One add-on's execution will not affect another unless it is being directly listened to by a second add-on. This is an invariant of the publish and subscribe style, where an announcer of an event does not know who is affected by these events. Kodi's design has many different events that could be listened to, and infinite possibilities to what users can implement. Add-ons are written in python, and are managed by the python module present in the business layer. In summary Kodi uses a publish and subscribe architectural style to introduce add-ons that are assigned events to both listen to and execute within Kodi's function.

Performance-Critical Components

Addons

Addons will allow developers/users to add custom Python scripts to the application. This will help foster the open source community and ideology of Kodi. Addons will be available throughout a store, so users can download other developers' addons. The pub sub ideologies need to be invoked to allow the rest of the app to remain unconcerned with the contents of these addons and enforce modular design, such that a corrupt addon does not break the whole user experience of the app.

Core Player

The Core Player will be in charge of playing/presenting all media to the user. The Core Player component should interact with the device through the platform component to ensure files are encoded and presented to the Kodi application such that it can understand local content. It will also handle content found through local network sharing, and handle the decoding of incoming information. Further, it's critical that it synchronizes all these tasks in a smooth, consistent, and relative manner. It is crucial there are no discrepancies between audio and video playback, or pauses as the application reads media from the system. The operation of this component is utterly critical to the overall user experience.

Platform

The platform component will handle all abstraction at a system/hardware level. It should act as a layer between the operating system and the application itself. This is important as it is not feasible to have high level elements worrying about the operating system, and is critical to overall function as errors at the system level can blow up and cause problems everywhere in the application. This component will

either handle requests on a case by case basis, or on invocation return a pointer to a platform appropriate library. This will, for instance, handle all file reading and writing to and from the local system.

User Interface

The GUI will sit on the client side of the application. It will interact with the platform component to handle hardware/operating system specific aspects in order to handle sizing and display. Further, it should with the platform component to extract user information. This component handles the languages and fonts presented to the user, as well as navigating the application in general.

Web Server Module

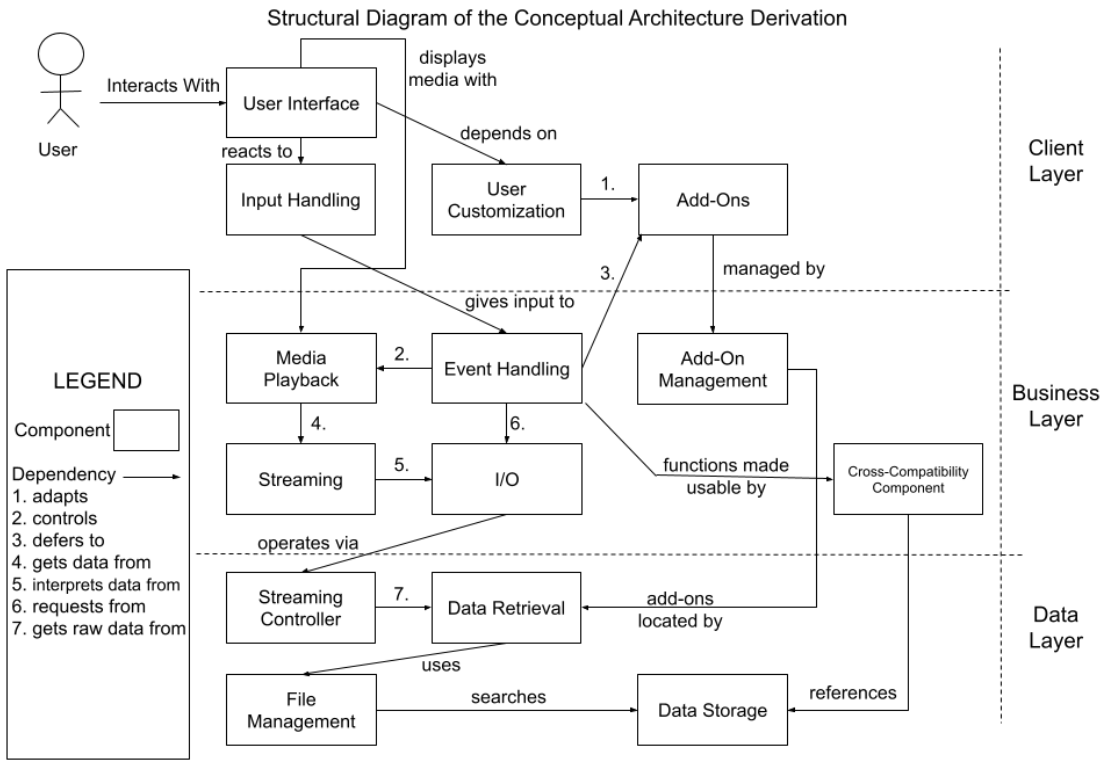
This component will handle the interaction of computers and files over local area networks (LAN) in order to support accessing content through different platforms, and streaming.

Global Control Flow

Within the client layer of the overall architecture, the User Interface and the Input handling components are responsible for the execution flow of the system. These components may be structured to operate concurrently (within different processes) to allow for responsive user experience where once a user input is collected the execution flow to the next component occurs in parallel. Within the Business and Data layer, the Streaming and Media Playback components act as a bridge between the data retrieval and streaming controllers. The event handling logic in the business layer controls the flow of execution to and from the data layer, and can be implemented using a publish subscribe model, event listener type structures, or internal method/functional calls. The data layer sends the retrieved or streamed data back to the business layer.

Diagrams

Structural Diagram



Kodi Architecture Diagram

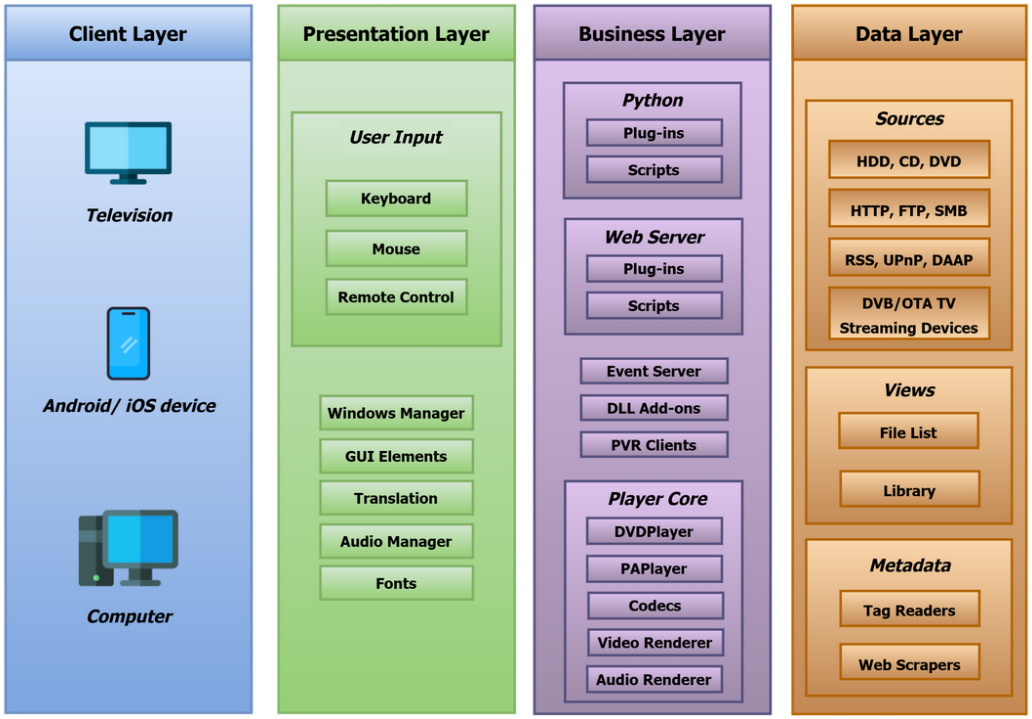


Diagram 2: View of the layers present in Kodi's architecture (Source: https://kodi.wiki/view/Architecture)

External Interfaces

While Kodi’s information is stored in a database, the precise format of the data is not a primary concern.

In terms of the user interface, the information content encompasses MP3 files, settings options, and add-on descriptions. User input is guided by expected system inputs, outlining the routes users can take to navigate different pages and the corresponding display elements.

Furthermore, files and media (i.e. the system’s capability to read various media formats) include MP4, MP3, MOV, and their associated metadata. Add-ons, particularly Python scripts, are also significant to Kodi.

Databases, arguably the most critical interface, are heavily influenced by user preferences. Namely, this is controlled by JSON settings files and other data interchange formats.

Use Cases

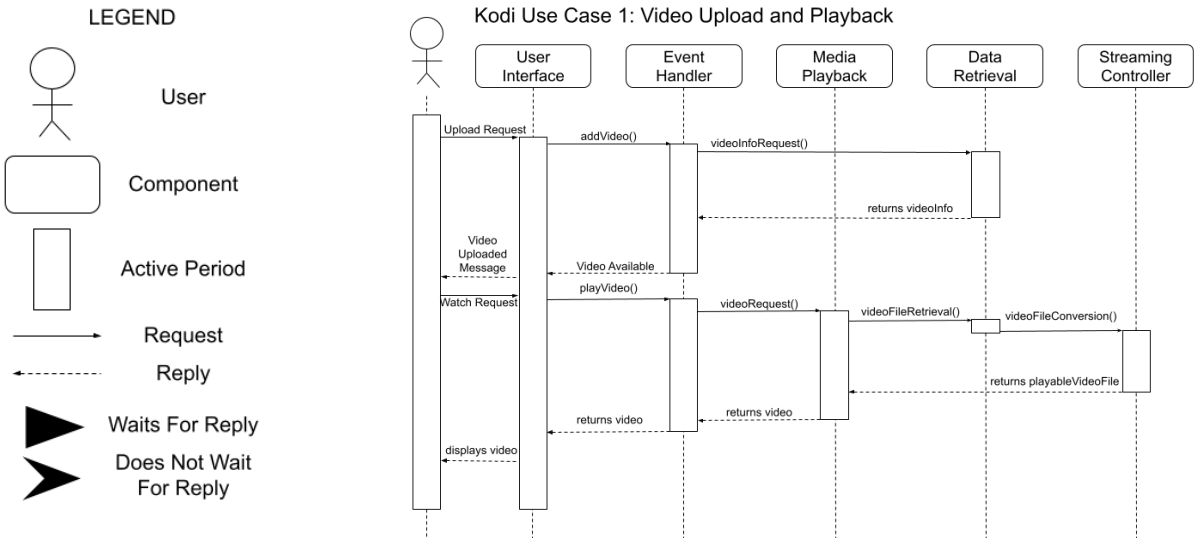
This section will focus on two main use cases, namely, this is **uploading and playing a video file** and **connecting an add-on.** These are arguably the two most fundamental use cases.

To upload and play a video file, the user engages with the client layer's interface, facilitating navigation and software control. The interface communicates laterally with the event handler, triggering "addVideo()" and "playVideo()" events, allowing relevant data passage to the business layer.

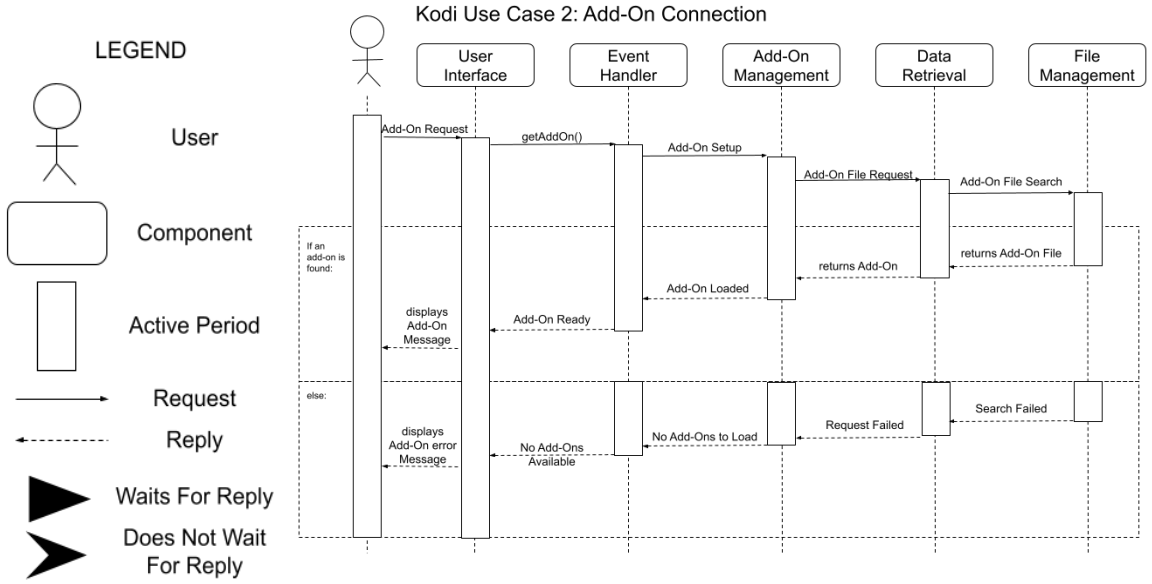
Within the business layer, the Event Handling component manages the event, interacting with the Media Playback component to handle crucial data needed for file playback (file location, resume time, subtitles, and settings). The Media Playback component communicates with the data layer to store and retrieve necessary files.

The Data Layer utilizes its data retrieval component, accessing the specified file through OS-specific system calls to retrieve binaries. These binaries are then processed by the streaming controller to convert them into playable content.

Following the layered style, data descends to the lowest (data) layer, then ascends to the highest (client) layer. Playable content from the data layer is directed to media playback and event handlers in the business layer, eventually reaching the video player in the user interface.



Connecting an add-on within Kodi mirrors the layered architecture described earlier. Notable distinctions include variations in event calls, the business layer component employing add-on management instead of media playback, and the data layer utilizing the file management component for add-on files. Despite these differences, the layered system maintains conceptual consistency, with data flowing from the bottom layer to the top layer. The user interface confirms the connected add-on through a displayed message.



Data Dictionary

Functions and Components

- **Media player:** Kodi's central function – the component that allows multimedia playback and interpretation of various digital coding formats
- **Entertainment hub:** A marketing term encapsulating the numerous entertainment features including music and audiovisual playback, live TV recording, and internet streaming
- **Add-ons:** Software installed alongside Kodi that allow the user to change the look and experience of the application
- **User interface:** the graphical interface that allows the user to interact with the application
- **Core components:** A term referring to the application's mission-critical components. Namely, this is the core player, add-on manager, platform, user interface, and web server module
- **Platform agnosticism:** The design philosophy associated with freeing software from being tied to any one particular environment
- **Input handling:** Detection and handling of real-time user inputs

Architectural Styles

- **Layered style:** One of Kodi's central architectural styles – information is encapsulated in layers where layers can only communicate to those perpendicular to it
- **Repository style:** One of Kodi's central architectural styles – information is stored in a central component and accessed by multiple entities
- **Publish and subscribe style:** One of Kodi's central architectural styles – information is broadcasted by "announcers" and consumed by "listeners"

Non-functional requirements

- **Scalability:** The ability to scale out or up to a larger capacity
- **Evolvability:** The ability to adapt and change rapidly to new requirements
- **Testability:** The ability to test the program for bugs and issues
- **Performance:** The throughput and latency of the application

---

Naming Conventions

- **GUI: Graphical User Interface**; the graphical interface that allows the user to interact with the application
- **LAN: Local Area Network**; an interconnected computer network within a small geographical area, this refers to the user's home network in this context
- **OS - Operating System**; software that facilitates a computer's basic functions including process and memory management

- **MP3 - MPEG Audio Layer III**; a coding format for digital audio
- **MP4 - MPED-V AVC;** a coding format for audiovisual content
- **MOV - Apple’s QuickTime video format**
- **JSON - JavaScript Object Notation;** an example of a data interchange formats

---

Conclusions

To sum up, the examination revealed that Kodi's core conceptual architecture follows a layered approach, supplemented by a supporting repository style for the video player and a publish-and-subscribe style for the add-ons. The detailed documentation of these architectural elements, including modules, components, and their interconnections, is available in the preceding report.

These findings shed light on how control flow operates among these components. Namely, parallel processing is employed between the user interface and input handling for a responsive experience. Streaming and media playback use data synchronization methods for data retrieval and streaming. And finally, event listening uses execution flow management to seamlessly handle broadcasted events.

In essence, Kodi stands out as a versatile and secure alternative for video playback. It empowers users to seamlessly transition their existing audio-visual content to television, offering robust support, privacy, and expandability. The chosen architectural style proves crucial for ongoing modifications, testing, and future integrations.

Lessons Learned

Crafting a conceptual architecture report involves a thorough journey of analysis, planning, and documentation. Throughout this expedition, we've absorbed a plethora of insights into the intricacies of creating such a report. Our collaborative endeavors in the realm of conceptual architecture have honed our ability to dissect a system's myriad components, organizing them into coherent sections that effectively convey goals and functionalities to stakeholders. Recognizing the significance of sketching out a "blueprint" prior to implementation has underscored the importance of achieving enhanced performance. Additionally, navigating the complexities of teamwork within a sizable group, dividing responsibilities, and meeting deadlines has polished our collaborative skills. This study has been a revelation, providing us with a priceless comprehension of the inner workings of systems architecture and the art of articulating our findings with clarity.

References

1. Kodi. Kodi, https://kodi.tv/.

2. Kodi Add-ons. Kodi, https://kodi.tv/addons/.

3. VideoLAN. VideoLAN, https://www.videolan.org/.

4. Khanna, S. and Infosys Technologies Ltd. "Internet X.509 Public Key Infrastructure: Qualified Certificates Profile." IETF, http://tools.ietf.org/html/rfc3003 .

5. Kaliski, B., et al. "Quoted Printable encoding for headers." IETF, http://tools.ietf.org/html/rfc4337.

6. "User interface." Wikipedia, https://en.wikipedia.org/wiki/User_interface.

7. Cisco. "What Is a LAN? (Local Area Network)." Cisco, https://www.cisco.com/c/en/us/products/switches/what-is-a-lan-local-area-network.html.