# The Concrete 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

18 November, 2023

## Abstract

This report serves as a resource for those who wish to learn more about the Kodi architecture. Kodi's architecture follows an Implicit Invocation or Publish-Subscribe for its main architecture style. Through broadcasting events created within the Skinning Engine, one of the top level subsystems of Kodi, the individual top level subsystems are capable of interacting. The report outlines the top level subsystems that Kodi consists of are Content Management, Interfaces, Player Core, Skinning Engine, and Utilities. Each of which play separate roles that contribute to the overall functionality of Kodi. The Content Management component handles storage and retrieval of media and libraries, Interfaces deals with web related functionality and add-ons the user may incorporate, Player Core is responsible for fundamental functionality of media playback, Skinning Engine serves as the interface that the user sees/hears as well as track user input, and finally Utilities houses essential tools and support functionality that assist in the operation of Kodi. Each of these top level subsystems alongside the Video Player subsystem is examined and has reflexion analysis performed on it. Our report also covers the derivation of the concrete architecture, where many new subsystems were identified and created. These discrepancies (new subsystems) include Utilities, Stream Clients, Retro Player, External Player, and Audio Manager. Through our derivation process, the rationale behind the creation of each of the subsystems is examined as well as our thought process when building the concrete architecture.

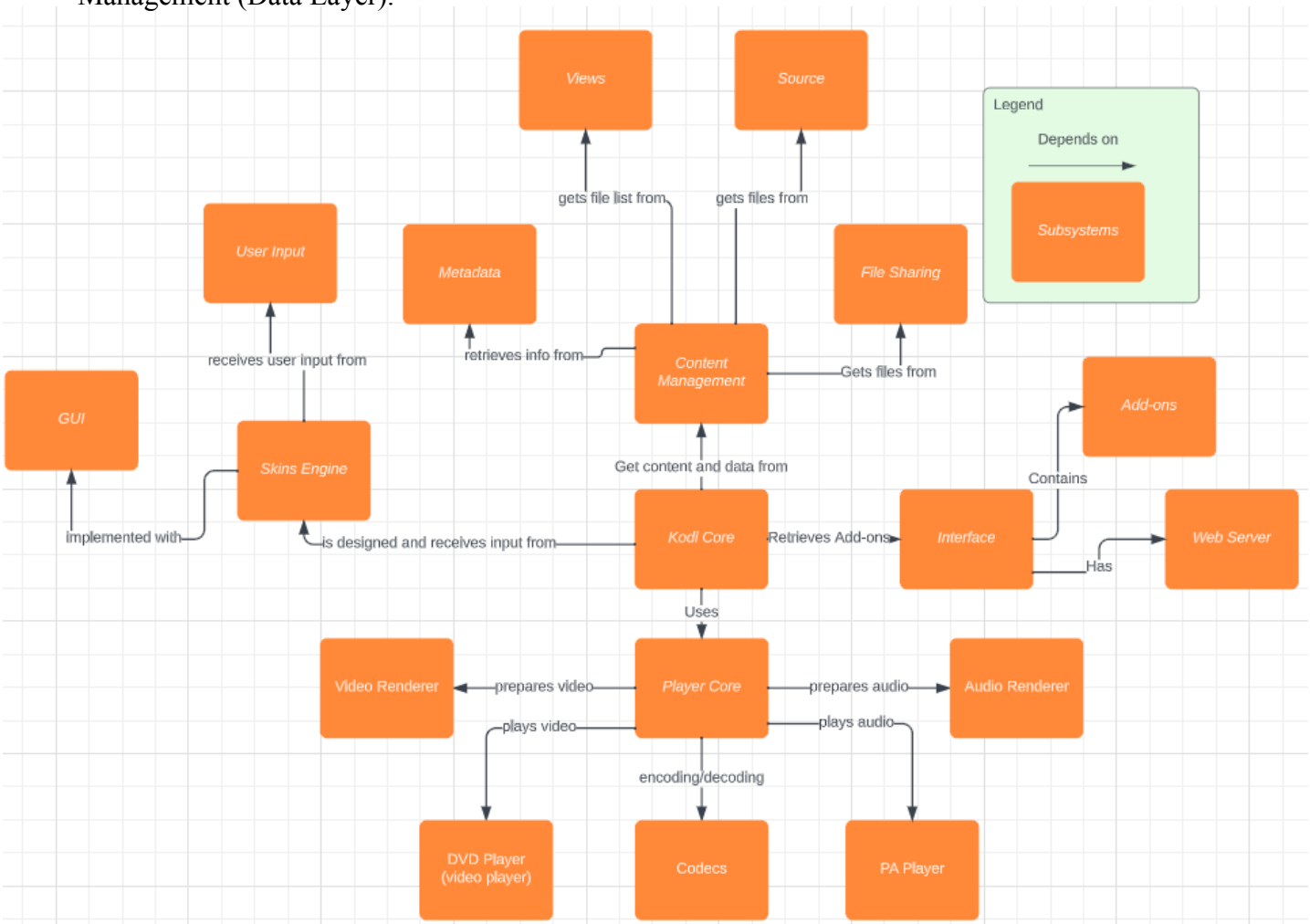

## Introduction

In this extensive architectural analysis, we delve into the transformation of Kodi's architecture from its conceptual blueprint to the creation of a detailed concrete architecture. The report sheds light upon the thought process, rationale, and critical reflections that shaped each architectural decision. This analysis is structured to offer a comprehensive insight into the transformation from the conceptual to concrete architecture, revealing the intricacies of Kodi's subsystems, their functionalities, and interactions. The analysis of each subsystem unravels the intricacies of Kodi's inner workings, clarifying divergences between the conceptual blueprint and the final concrete architecture. The report outlines the nuanced reasons behind additions and relocations from the conceptual architecture to the concrete architecture, addressing the reasons behind divergences and the implications on Kodi's functionality. The discrepancies noted are derived from the analysis of Kodi's inner subsystems, their functions, and their interactions. The examination delves deeply into the inner workings of the 5 subsystems within Kodi's architecture: Content Management, Interfaces, Player Core, Skinning Engine, and Utilities. Within the Content Management subsystem, the report dissects the subsystem's pivotal role in encapsulating functionalities pertaining to user data, stored content, and specific platform libraries. This in-depth exploration further isolates Content Management into Media Sources, Metadata, and Views. Similarly, the Interfaces subsystem is explored, highlighting its role in managing web-related functionalities and user-customizable features. This subsystem is separated into Add-ons, Stream Clients, and Web Server. The Player Core is responsible for orchestrating fundamental aspects of media playback within Kodi, housing specialized subsystems such as the Audio Renderer, Codecs, DVD Player, External Player, PAPlayer, RetroPlayer, and VideoRenderer. Each of these components collaborate to ensure seamless playback experiences across numerous forms of media content. Moreover, the report navigates the Skinning Engine, revealing its significance as the interface layer dictating Kodi's aesthetic

appeal, user interaction, and overall user experience. Within this subsystem lies the Audio Manager, GUI, User Input, and Window Manager, each playing a pivotal role in manufacturing Kodi's visual aesthetics and user interaction specifications. Finally, the report navigates the Utilities subsystem, clarifying its role as a repository for foundational tools, testing modules, and diverse utilities that fortify Kodi's operational framework. In addition to the analysis of top-level subsystems, through an in-depth exploration of the VideoPlayer subsystem and its interactions within Kodi, the report reveals the intricacies of video playback functionalities and their dependencies within the architecture. Detailed sequence diagrams are located throughout the report to illustrate the inner workings of the concrete architecture in various use cases. From the derivation process, reflexion analyses, and detailed inner architecture discussions, this report aims to provide an insightful exploration of Kodi's concrete architecture.

## Changes to Conceptual Architecture

Before creating our concrete architecture of Kodi, we first made some changes to how we represented the conceptual architecture. We wanted to show some of the inner components of some of the subsystems that we already had in our first version of the conceptual architecture. In the Player Core subsystem we wanted to show the Audio Renderer, Video Renderer, DVD Player (also known as Video Player), PA Player, and Codecs subsystems/components. The rationale behind this choice was to better show the different components and interactions within the Player Core. This allows for us to see which part of the Player Core depends on, or is depended on by other parts of the architecture. We also decided to move the File Sharing subsystem into the Content Management subsystem. The reason behind this decision was to better match the four layer design for the conceptual architecture since the inner components are part of the Content Management (Data Layer).



Descriptions of Top Level Components and Architecture

The interactions within Kodi thoroughly demonstrate its concurrency and publish-and-subscribe architecture embedded within it. When actions are completed through peripherals, the SkinsEngine through GUI components creates events. The events are listed and appropriate subscribers are distributed the event and handle it through internal components. The use-case of the pause functionality explains how the left click is an action/event and is then handled by the application which determines its a pause for the video player.

Within Kodi there are also subsystems that resemble other architecture styles. For example, the DVD Player (Video Player) matches the pipe and filter architecture style which allows for concurrent processing. The Stream Clients subsystem is similar to the Client-Server architecture style as it distributes data hosted on servers to clients.

Content Management is the subsystem that encapsulates the functionality revolving around user data, stored content, and additional libraries pertaining to specific platforms and functionality around media. Within Content Management there are 3 subsystems: Media Sources, Metadata, and Views. Media sources handle the storage and retrieval of content such as movies or songs, including media that is stored on another networked PC. The Metadata subsystem which has the functionality for database wrappers, tags, and info scanner, all of which are related to grabbing and storing information about the media. The Views subsystem has the libraries related to specific platforms, dynamic link libraries, and other processes.

Interfaces is the subsystem that handles web related functionality as well as additional functionality the user can add. There are 3 components that Interfaces is composed of: Add-ons, Stream Clients, and Web Server. Add-ons allow the user to incorporate add-ons to customize and change the way Kodi operates. Stream Clients has the functionality that allows the user to stream live television and the recording of it. The last component, Web Server handles all web/network related functions.

The Player Core subsystem is responsible for handling and managing the fundamental aspects of media playback. It is composed of various specialized modules that collaborate with each other to ensure quality playback experiences for different types of media content. The Player Core encompasses 7 subsystems: Audio Renderer, Codecs, DVD Player, External Player, PAPlayer, RetroPlayer, and VideoRenderer. The Audio Renderer manages the rendering and output of audio streams, Codecs handles the decoding and encoding of different media formats, the DVD Player deals with various video-specific functionalities, the External Player facilitates integration with external media players, the PAPlayer provides alternate playback capabilities, the RetroPlayer focuses on gaming-related functionalities, and the VideoRenderer manages the rendering and display of video content.

The Skinning Engine within Kodi serves as the interface layer responsible for the look, feel, and user interaction aspects of the media player. It consists of 4 essential subsystems that manage the user interface and input functionalities, ensuring a customizable and engaging user experience: Audio Manager, GUI, User Input, and Window Manager. The Audio Manager oversees audio-related settings within the user interface, the User Input subsystem manages user interactions and input devices, and the Window Manager controls the creation, display, and management of windows within the user interface.

The Utilities subsystem in Kodi's architecture serves as a repository for various essential tools, resources, and support functionalities that contribute to the smooth operation and maintenance of Kodi. It consists of various files and 8 subsystems: Commons, Contrib, Cores,

Power Management, Test, Threads, Utils, and Weather. Commons holds common functionalities that are frequently used across different components within Kodi, Contrib contains mathematics algorithms vital for Kodi's function, Cores houses core functionalities and data structures, Power Management includes tools to manage power-related features, Test encompasses various testing tool and frameworks, Utils consists of utility modules and tools that provide auxiliary functions, and Weather contains tools and APIs related to weather information services.

## Discrepancies of High Level Architecture

Alongside the initial changes we made to the conceptual architecture, there were also additional subsystems that were added. As we looked through the directories and files, we realized that there were groups of directories/files that had a common purpose that could be grouped together into a subsystem. These subsystems added are the divergences between our conceptual and concrete architecture.

In the top-level of the concrete architecture there is one discrepancy where we added a utilities subsystem. After organizing the rest of Kodi's files into the concrete architecture, there were files that were related to settings, math, threads, power management and more. As a result, our group determined there should be a Utilities subsystem that contains these files.

There were also some discrepancies for some of the inner subsystems starting with the Interfaces subsystem. Within this subsystem we created a new subsystem called Stream Clients. The Stream Clients subsystem handles the streaming of live television through the PVR feature. The rationale behind this decision is that we realized the live streaming was more related to the networking and transferring of the content rather than rendering which would be Player Core.

When constructing the concrete architecture for the Player Core, two divergences that can be noted are the addition of the External Player and the Retro Player subsystems. These divergences arose due to a newfound understanding of Kodi's Player Core's inner workings when analyzing the XBMC source code through Understand. Kodi users may prefer external players or devices to handle specific types of content, which is something that we did not account for in our conceptual architecture.[1] Kodi's codebase contains an external player, which possessed a functionality unique enough from the other Player Core subsystems that we believed it needed to be its own separate subsystem. Similarly, the introduction of the Retro Player subsystem stems from the potential need for gaming functionalities within Kodi, as providing support for gaming features could cater to a broader audience interested in gaming alongside media playback.[2] The RetroPlayer directory found in the XBMC source code handles this potential need, and we believed that the tasks the Retro Player performed inside Kodi's Player Core were unique enough from the other subsystems to warrant RetroPlayer to be its own subsystem in our concrete architecture.

In regards to the Skinning Engine, two additional second level subsystems were added. Inside the Kodi codebase there were files related to how audio is handled on the user side. But within the Skinning Engine there we did not create any subsystem to handle audio. So on top of the GUI the user sees and interacts with, audio must also be handled on the user end of the software. Thus the Audio Manager was added to the Skinning Engine, which is responsible for translation and audio related settings within the user interface. Additionally, the window manager

[1] https://kodi.wiki/view/External_players
[2] https://kodi.wiki/view/Category:RetroPlayer

was implemented to control the creation, display, and management of windows within the user interface.

Concrete Architecture

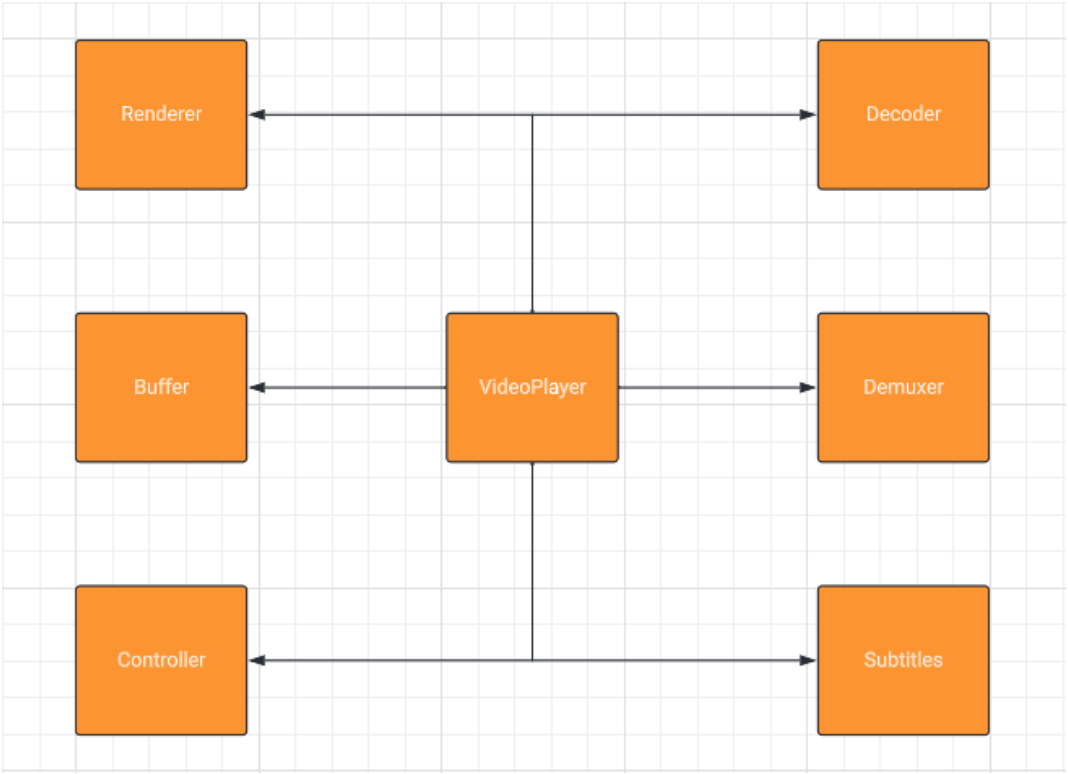

Inner Architecture of the Video Player

From our created architecture, derived from the architecture in the Kodi documentation, it is generally understood that the PlayerCore in the Business Layer is essentially synonymous as the subsystem primarily responsible for video playback. From the source code, it is identifiable that it is responsible for controlling the playing of different media, and within it, rendering, codecs and more, which are each integral to the functionality of video playback. Within the player core there must be a dedicated video player, which is the selected subsystem for analysis.

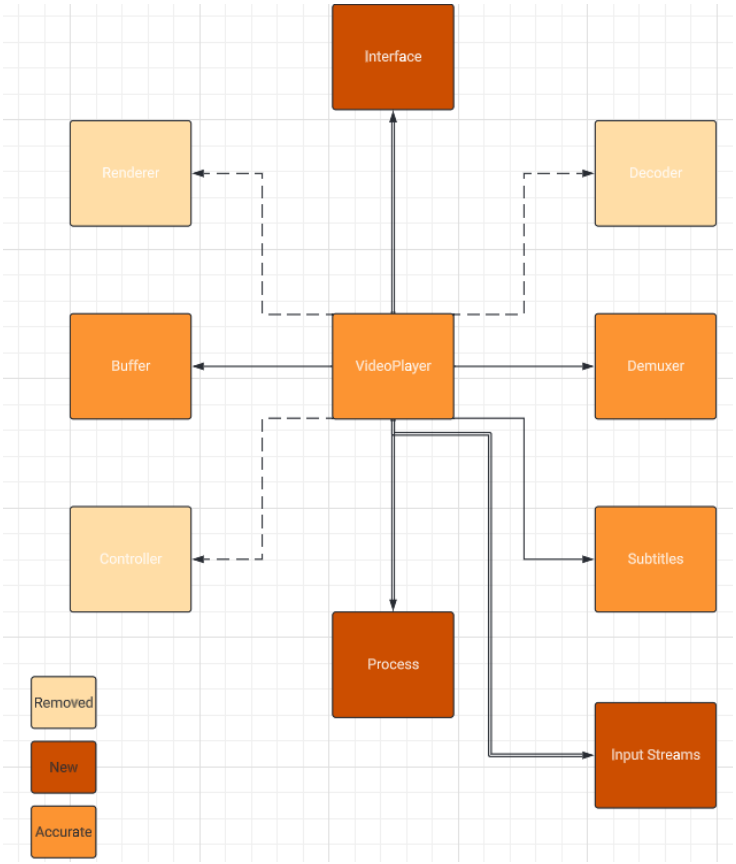
Before going into the concrete architecture created by Understand, the conceptual architecture will be presented. The conceptual architecture was created based on what a video player would generally need in order to function, with considerations from our previous architecture.

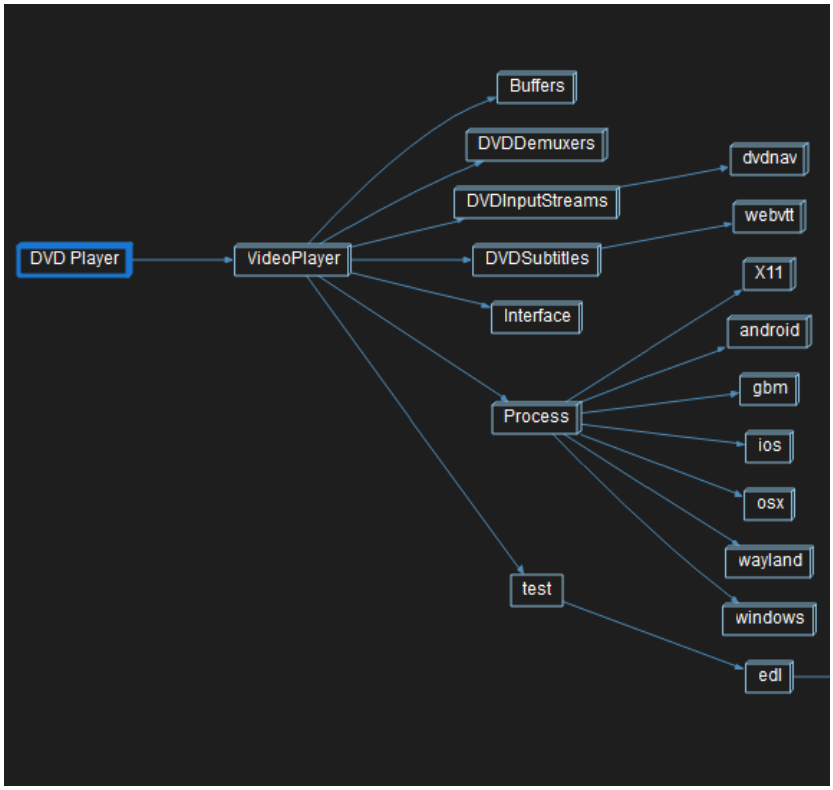
For a fully-functioned video player, for example, these are the following components needed: source, demuxer, decoder, renderer, buffer, controller, network, and subtitle support. It is important to note that all these components can be considered the base case for video playback capabilities. The focus of our subsystem is the actual playing of the content within the Player Core. Kodi already provides the source and network capabilities from other components, thus the video player needs the remaining features.

In terms of interactions within the conceptual architecture, the demuxer would separate the stream for subtitles and decoder, and then render, by the renderer, for the buffer and video player. Finally, the controller would allow playing and pausing of the video player.




From Understand, a proper concrete architecture was able to be created. Between the two architectures, there are some discrepancies. Firstly, there is no renderer. This makes sense as the renderer is actually under the PlayerCore and not the DVDPlayer (VideoPlayer). This can be because of other media content being supported. Secondly, the video player has support for multiple OSs and architectures under Process and an Interface, which is something the conceptual architecture did not consider. Thirdly, the conceptual architecture assumed the controller would be defined within VideoPlayer, however it is not visible under VideoPlayer, but rather somewhere else. The interaction process is similar to how it is described in the conceptual architecture. Finally, the decoding and encoding is left to Codecs, most likely, however the handling of input streams is additional. Though test is a separate entity present in the architecture, it is assumed that a dependency is not used during runtime, but rather as validation of the functioning video player.




One odd feature of the architecture is the relation between DVD Player and VideoPlayer. Despite a difference in naming convention, it seems as though DVDPlayer is now called VideoPlayer, but the internal components have not been changed in name. This is most likely due to evolution.

Regardless, the chosen subsystem to analyze is the VideoPlayer itself. From the diagram above, it depends on many different components in order to properly function: Buffers, DVDDemuxers, DVDInputStreams, DVDSubtitles, Interface, and Process. From the component names, it is easily identifiable what role they each play.

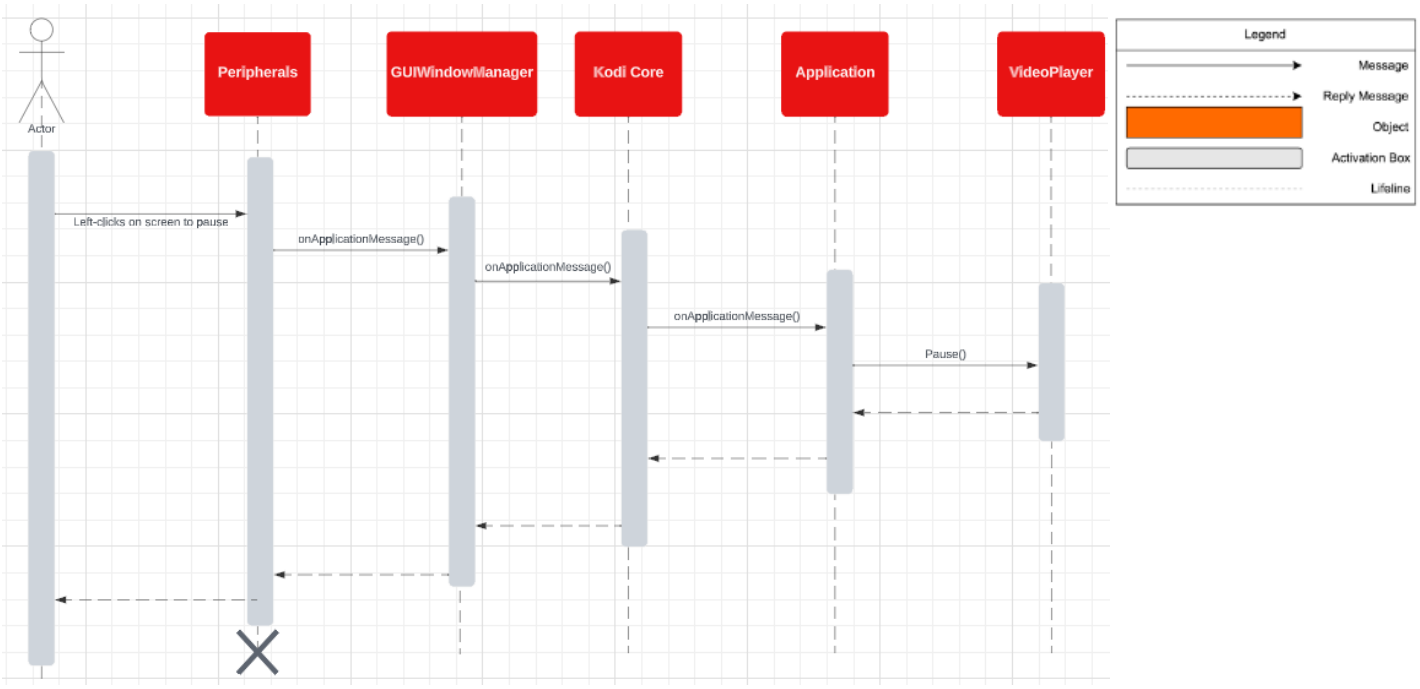
The VideoPlayer subsystem is the central module for video playback in Kodi, coordinating interactions between demuxers, input streams, and buffers. It manages the entire playback process and integrates demuxing, input stream handling, buffer management, and subtitle rendering seamlessly. The modular structure allows for easy maintenance and customization without affecting the overall system.

The Buffer is responsible for managing video buffers, crucial for achieving smooth video playback. It includes code for handling different types of buffers, their allocation, and synchronization to ensure a continuous and uninterrupted streaming experience. DVDDemuxers focuses on the process of separating multimedia streams. It contains code for handling various demuxing formats. This directory ensures that multimedia content from diverse sources can be appropriately separated and processed. DVDInputStreams manages different types of input streams, and the dvdnav subdirectory specifically deals with DVD navigation streams. It includes code for handling Bluray, file-based, and memory-based input streams, as well as navigation-specific streams for DVDs. The DVDSubtitles directory is dedicated to managing subtitles during video playback. Interface contains code defining interfaces for various components within the VideoPlayer subsystem. Files like DemuxCrypto.h and DemuxPacket.h

indicate the presence of interfaces for cryptographic operations and packet handling, respectively. Process contains platform-specific code for different operating systems and windowing systems. Each subdirectory, such as android, gbm, and windows, includes code tailored to the specific requirements of that platform.Test is dedicated to unit testing. The edl (Edit Decision Lists) subdirectory, allows developers to ensure that changes or additions to the codebase do not introduce unintended issues and that the system behaves as expected.

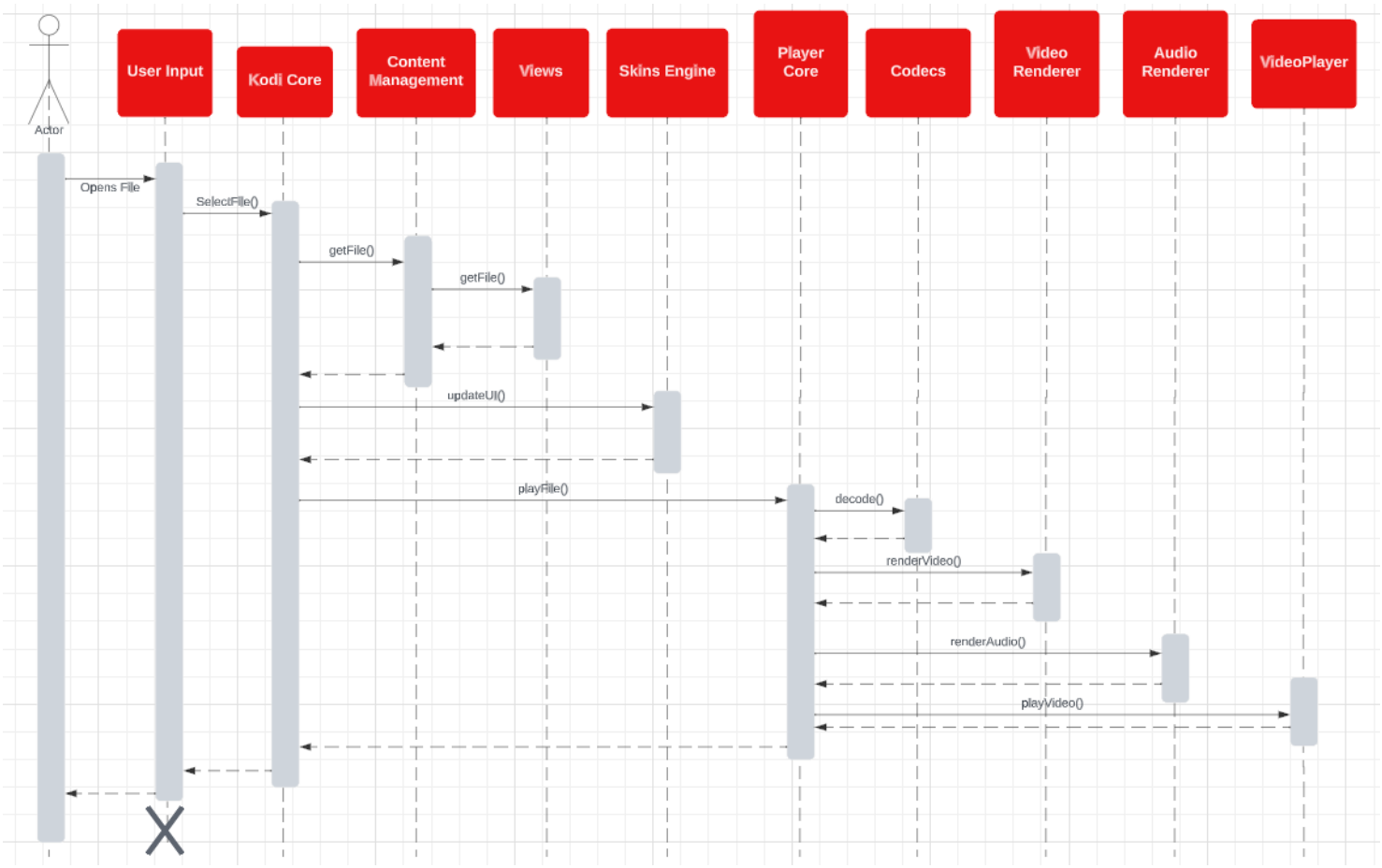
The VideoPlayer subsystem in Kodi collaborates with higher-level subsystems. In content management, it interacts with media and storage for access and management, potentially utilizing infoscanner and Metadata for file information. On the interface front, VideoPlayer communicates with interfaces and networks for input compatibility and network functionalities. Within Player Core, it directly interacts with audioEngine and playback and leverages Codecs for stream decoding. In the Skinning Engine, it engages with GUI and guiwindows for UI during playback, and with the utils directory for task-related utilities. This cohesive integration ensures a flexible media playback experience within Kodi.

## Use Cases




The sequence diagram above highlights the scenario when the user left clicks on their mouse while a video is playing in order to pause it. Note that while these components are being listed to exemplify the main interactions going on in order to provide this functionality, there are other minor components at work, or possibly different scenarios in which the sequence is a little different. This could include if the left click was on the pause button or on the screen, whether the screen is fullscreen, and more. The general case is that Peripherals acknowledges an action done by the user and sends the action to the GUIWindowManager. The GUI portion identifies which element the left click is for, and assigns it to Kodi Core. Here the action is further transported as an action for pausing. The application then recognizes it was for the video player,

and then the player pauses the video. The return is typically a boolean as to whether it was successful or not.




The case shown above highlights how the system interacts with each component when the user opens a media file. The user will perform the action of opening a file through the presentation layer, in which then the Kodi core calls upon Content Management to retrieve the file. The Content Management will retrieve the file's metadata and content. Kodi Core might communicate with the Skins Engine, in order to update the UI accordingly based on the file or user interaction. Finally, the Kodi Core will need to use the Player Core component to decode and render.

## Derivation Process

In our final concrete architecture, Kodi possesses 5 subsystems, which are Content Management, Interfaces, Player Core, Skins Engine, and Utilities. Each of the subsystems can be further broken down into smaller subsystems created by our group during the architecture building process. The subsystems of Content Management are Media Sources, Metadata, and Views. The subsystems of Interfaces are Add-ons, Event Server, Stream Clients, and Web Server. The Player Core's subsystems are the Audio Renderer, Codecs, DVD Player, External Player, PAPlayer, RetroPlayer, and VideoRenderer. The

subsystems of the Skins Engine are the Audio Manager, GUI, User Input, and the Window Manager. Finally, the utilities subsystem is unique, as it does not contain any subsystems that were created by our group. All of the subsystems of Utilities are XBMC system and utility directories that interact with the other subsystems in various ways to enhance their overall capabilities.

Regarding the derivation process for the Content Management subsystem, the first subsystem that XBMC directories were transferred to is Media Sources.

File Sharing is a third-level subsystem created to store any directories pertaining to the sharing of files. While it was listed as a separate subsystem in our developmental view of Kodi's conceptual architecture, upon further reflection we believed it to best fit in the 'Media Sources' subsystem. The XBMC directories, namely 'messaging', 'filesystem', 'listproviders', 'media', 'recordings', and 'storage', are inherently linked to the retrieval, organization, and management of diverse media content from various sources within Kodi's architecture. These directories and other files in the Media Sources subsystem encompass functionalities ranging from accessing file systems, providing catalog-like structures of available content, managing media playback, handling recorded materials, and optimizing storage mechanisms. Their placement in the Media Sources subsystem underscores their roles in enabling Kodi to handle a wide array of multimedia content, making them integral to Kodi's overall content management capabilities.

The 'dbwrappers', 'infoscanner', and 'tags' directories are inherently linked to the acquisition, processing, and organization of metadata within Kodi's architecture. These directories and associated files collectively contribute to enriching XBMC's functionality by sourcing and managing metadata such as information about media content, enabling efficient organization and retrieval within the Kodi environment. They were placed in the Metadata subsystem due to their important roles in enhancing content management through metadata manipulation and utilization.

The directories 'dllLoader', 'favourites', 'imagefiles', 'jobs', 'music', 'pictures', 'platform', 'playlist', 'process', 'profiles', and 'video' and other files placed within the Views subsystem are files and libraries that collectively contribute to aspects directly related to the organization, representation, and manipulation of visual and auditory content within Kodi. The directories contain code responsible for managing different types of media content representations, user preferences, tasks, and configurations within different contexts offered by Kodi.

Moving on to the Media Sources subsystem, the 'addons' and 'interfaces' directories alongside their associated files play integral roles in extending and enhancing the functionality and user interaction capabilities within Kodi. Their placement within this subsystem is due to their collective role in facilitating customization, expansion, and augmentation of Kodi's features through various extensions and interfaces. They help extend the capabilities of Kodi, enhancing its flexibility and usability through modular and customizable components.

The 'events' and 'threads' directories along with other files are key in managing event-driven mechanisms and concurrent processes within Kodi. Their placement within the Event Server subsystem is due to their collective role in handling and orchestrating various events, triggers, and concurrent operations that facilitate communication and coordination among different components of the software.

The 'pvr' directory, was placed within the Stream Clients subsystem due to its role in handling and managing TV and streaming functionalities within Kodi. It plays a pivotal role in facilitating the streaming client capabilities, encompassing components responsible for managing TV channels, electronic program guides, and streaming functionalities, making it a suitable addition to the Stream Clients subsystem due to its focus on enabling streaming and live TV features within Kodi.

The 'network' directory and associated files were placed within the Web Server subsystem due to their roles in managing networking functionalities specifically related to web services and communication within Kodi. This directory contains code responsible for handling web-related protocols and functionalities related to serving web-based content or managing network-related aspects within the software.

In regards to the Player Core's subsystems, the 'audioEngine' directory was placed within the Audio Renderer subsystem due to its role in managing and orchestrating audio rendering functionalities within Kodi. This directory contains code responsible for handling audio playback, processing, and rendering tasks specific to Kodi's audio functionality.

The 'DVDCodecs' and 'cdrip' directories along with associated files were placed within the Codecs subsystem due to their roles in managing codec-related functionalities specific to DVD playback and CD ripping within Kodi. These directories contain code responsible for handling encoding, decoding, or processing audio/video codecs specifically tailored for DVD playback and CD ripping tasks.

The 'videoPlayer' directory is renaming the DVDPlayer as that directory can no longer be found.

The 'playercorefactory' and other individual files were placed in the External Player subsystem, as they manage and configure external player functionalities within Kodi. The 'playercorefactory' directory contains code responsible for handling the configuration and management of external player options for specific media formats or sources within Kodi.

The 'paplayer' directory was placed inside the PAPlayer subsystem for its role in managing and handling audio playback functionalities specific to PAPlayer within Kodi. This directory encapsulates the implementation and management of the PAPlayer module, which focuses on audio playback capabilities specific to PAPlayer.

The 'RetroPlayer' directory was made into its own unique subsystem, due to its role in the handling of visual and auditory playback, processing, and buffering capabilities for games. The directory contains code that is unique for games, which is why we deemed it necessary for RetroPlayer to be its own separate subsystem under the Player Core.

Finally, the 'videoRenderers', 'buffers', 'playback', and 'rendering' directories went inside the VideoRenderer subsystem due to their roles in managing and handling video rendering functionalities within Kodi. These directories contain code responsible for various aspects related to video rendering, including handling video buffers, implementing playback control mechanisms, and dealing with rendering processes specific to video output within Kodi.
Moving on to the Skins Engine subsystem, the 'audio' directory was placed within the Audio Manager subsystem due to its role in managing and handling audio-related functionalities specific to Kodi's Skinning Engine.

The 'application', 'dialogs', 'guibridge', 'guicontrols', 'guilib', 'guiplayback', 'guiwindows', and 'programs' and any other GUI related files are located within the GUI subsystem, due to their roles in managing and handling various graphical user interface (GUI) elements, controls, interactions, and functionalities within Kodi.

The 'controllers', 'input', 'peripherals', and 'speech' directories were placed within the User Input subsystem for their roles in managing and handling various input-related functionalities within Kodi. These directories contain code responsible for defining input controllers, managing input interactions, and handling peripheral or input devices.

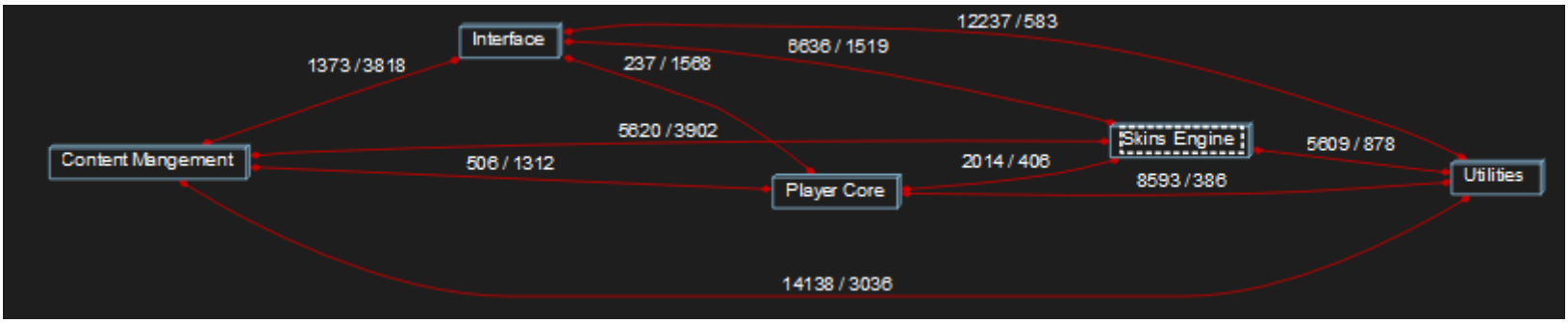
Finally, the 'windows' directory and certain individual files are located within the Window Manager subsystem, due to their roles in managing and handling window-related

functionalities within Kodi. This directory contains code responsible for defining and managing various aspects of windows within the user interface of Kodi.

Looking at the final top-level subsystem of the concrete architecture, the 'commons', 'contrib', 'cores', 'powermanagement', 'test', 'threads', 'utils', and 'weather' directories along with certain individual files find their place within the Utilities subsystem due to their nature of providing foundational and supportive functionalities to the entire Kodi system. The utilities subsystem serves as the backbone of Kodi, offering essential tools, common functionalities, testing modules, thread management, and miscellaneous utilities. Overall, these directories and files make up the Utilities subsystem, contributing to Kodi's overall functionality and robustness.

Due to the implicit invocation architectural style of Kodi, the events directory of XBMC's source code was moved under the Kodi Core, which is the highest level of the concrete architecture in which all subsystems extend from. The events directory directly handles event scheduling, therefore its location in the concrete architecture makes sense due to the Kodi Core being the component that announces events, with the 5 main subsystems (Content Management, Interfaces, Player Core, Skins Engine, and Utilities) being the components that register interest in an event.

Understand Dependency Diagram




Conclusion and Lessons Learned

In short, our exploration of Kodi's architecture aimed to provide a comprehensive understanding. We evolved from our initial conceptual framework, refining it to better align with Kodi's structure, emphasizing subsystems and adjusting component placements. We dove into the significance of the VideoPlayer subsystems key components to see how it reflects with the higher level subsystems. Lastly, our reflexion analyses highlighted discrepancies between conceptual and concrete architectures, offering valuable insights. The main lesson drawn from this architectural exploration is the inherent difficulty in accurately predicting the intricate web of dependencies within a program's architecture. Despite meticulous planning in our conceptual architecture, the XBMC source code revealed significantly more dependencies and interconnections than we had previously assumed. The analysis acknowledges uncertainties. We made assumptions about certain architectural aspects, marking these as potential limitations. Additionally, the dependency on tools, particularly Understand for code analysis, is highlighted, emphasizing the need to consider any inherent limitations or blind spots associated with the chosen tool. These factors underscore the importance of transparency and awareness of potential constraints in the overall analysis. Through the use of Understand we learned more about the

interconnections within Kodi. Understand provided the tools to see the dependencies and how the architecture was utilized. We also learned more about the prevalence of concurrency and threading within Kodi and events which lead to the publish and subscribe architecture. In conclusion, the journey from conceptual to concrete architecture for Kodi exemplified the importance of reflexion analysis in understanding the inner workings of a program's source code.

## Data dictionary

Codec: A program that encodes and decodes data

Demuxing formats: the process of extracting individual signals from stream

Implicit Invocation: An architectural style where the main idea is that a central system broadcasts events for individual components to register for and complete.

## Naming Conventions

API: Application Programming Interface

GUI: Graphical User Interface

PC: Personal Computer

PAPlayer: Psycho-acoustic Audio Player[3]

UI: User Interface

XBMC: Xbox Media Center (Old name for Kodi)

[3] https://kodi.wiki/view/Archive:PAPlayer

References

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

Category:retroplayer - official Kodi Wiki. (n.d.-a). https://kodi.wiki/view/Category:RetroPlayer

External players - official Kodi Wiki. (n.d.-b). https://kodi.wiki/view/External_players