Conceptual Architecture of Kodi

CISC 322 Assignment 1 Report

October 22, 2023

Nicholas Falconi (falconi.nicholas@queensu.ca) 20124757

Karina Verma (20kv15@queensu.ca) 20287311

Arda Ozdemir (arda.ozdemir@queensu.ca) 20370479

Jasmine van Leeuwen (20jjvl@queensu.ca) 20256908

Hashir Sami (hashir.sami@queensu.ca) 20285281

Anishka Barran (20ar14@queensu.ca) 20292268

Table of Contents

1 Abstract

This report provides an in-depth examination of the Software Architecture of Kodi, beginning with its primary interactions with end-users and extending to the underlying technologies and frameworks that ensure its functionality. Kodi's architecture is primarily characterized by a layered design, incorporating the client, presentation, business, and data layers. It also embraces an interpreter style for user-driven add-on development and a repository style demonstrated by the use of GitHub as a central data structure. Future changes and adaptability are discussed, with a focus on modularity through extensions and plugins. The report emphasizes the vital components and external interfaces that contribute to Kodi's flexible and user-friendly multimedia experience. Kodi's architecture showcases the power of open-source collaboration, enabling a dynamic and adaptable platform for multimedia entertainment.

2 Overview of Kodi

The Kodi Media Player is a testament to the innovations achievable through collaborative digital efforts. Developed by the XBMC Foundation and enriched by contributions from a multitude of international developers, Kodi has firmly positioned itself as a prominent platform within the entertainment hub sector. This open-source multimedia player features an architecture that is able to seamlessly integrate various media types, including video, audio, and images.

Kodi's origins lie in its initial development for the XBOX game console, known then as the XBOX Media Center (XBMC). Its evolution from a game-centric software to a full-fledged media center led to its rebranding as the “Kodi” Entertainment Center in 2014. True to its open-source development ethos, Kodi remains freely available to its large user community.

At the heart of Kodi's operations lies its collaborative development process. With its large developer base, and constant list of ongoing developments maintained by GitHub issues, Kodi provides ways for developers of all skill abilities to involve themselves in the open source software.

Notable updates, such as version v15 'Isengard', released in 2015, showcase Kodi's ongoing growth. This update reflected the Foundation's commitment to addressing user feedback, by implementing highly requested user components, performance updates, and quality of life updates. Furthermore, version v20, named Nexus, showcases Kodi's commitment to progress. Nexus has accumulated over 4,600 commits, introducing key features such as multiple binary add-ons, AV1 video support, and standardized subtitle formats. As Kodi continues to evolve, it never strays from its dedication to user experience and technical advancements.

3 Architecture Style

Though Kodi uses several architectural styles, layered architectural style is the primary classification of the software. In addition, Kodi utilizes a repository and interpreter style for subclasses within the main

architecture. In addition, throughout the project, Kodi makes use of Event Driven architecture to ensure responsiveness.

3.1 Layered Style

The primary architectural style of Kodi is layered. By definition, the layered style focuses on organizing similar functioning components into horizontal layers, where each layer performs a specific role. Kodi divides its architecture into four distinct layers: the client layer, the presentation layer, the business layer, and the data layer.

The client layer centers around the end-user's device, and encompasses the essential functionalities and support for systems Kodi is able to run on. The most up-to-date iteration of Kodi extends compatibility to Apple tvOS, Android 5.0 and above, as well as iOS devices equipped with XCode.

The presentation layer is primarily concerned with the packages and files that play a pivotal role in rendering information to the user. It includes files and packages which are in charge of the user information display, including but not limited to user input, view management, GUI elements, and audio management.

The business layer encompasses files and packages associated with server connectivity, add-ons, and also furnishes a client that establishes connections with streaming servers. It is responsible for the retrieval and presentation of video and audio content, transforming raw data into a seamless and user-friendly viewing experience. This layer also integrates the Python module, allowing for custom add-on integration. In essence, this module handles all aspects pertaining to content delivery to the end-user.

The data layer focuses on file and content management. Its primary uses can be found in streaming media from an external server and transforming data to FTP, RSS and HTTP files. These use cases are divided into three different elements: Sources, Views, and Metadata.The Sources element organizes the media sources, views element focuses on the user interface for the content, and the metadata element focuses on additional information that can help improve the user experience.

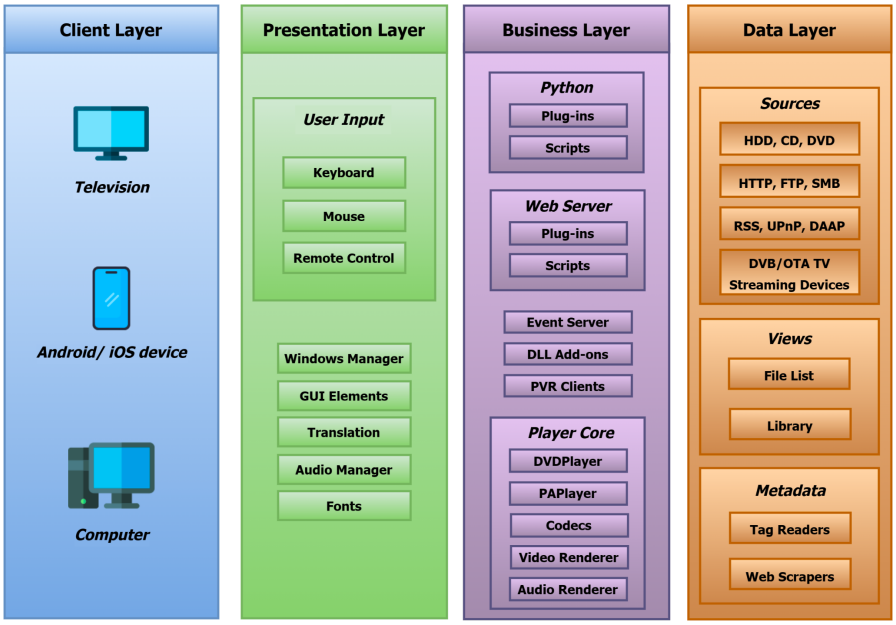


Fig. 1 Kodi's Official Breakdown Architecture Style

As observed, from the architectural layers, each layer services the layer above it. The data layer is responsible for all of the information necessary for the web server, player core, and the plug-in functionality in the business layer. The business layer is the layer that interacts with the presentation layer to execute tasks based on user input, while also using the data from the data layer to function. Finally, the presentation layer parses the input and provides relevant output, which the client visually sees and interacts with from the client layer.

3.2 Interpreter Style

Primarily used in the business layer, the interpreter style is important for the integration of the Web Server and Python add-ons/scripts. Although technically speaking, this interpreter style is not necessary for the functionality of the whole application, it provides a gateway for a customizable user experience.

The implementation follows a simple execution style. A Python script written by a user is compiled and executed using the embedded Python interpreter. It then uses an API created specifically for plugin/add-on script compatibility that interacts with the C++ core (without modifying it). If compiled successfully, the script will run as intended, allowing for cross-language communication and development. To see more information, reference the Python module in section 4.2.1

3.3 Repository Style

By definition, Kodi also utilizes the repository style. Since Kodi uses GitHub, which can be seen as a central data structure, and also creates sub-components (observed through layered architecture style), it can be said that the project uses some implementations of repository style.

Kodi uses CMake to automate, package, test, and build the C++ codebase. Additionally, Github is used as a central repository for the project, which services many different builds for the application. In terms of the main dependencies required for the project, Kodi uses Neptune and Platinum. Platinum is a modular UPnP cross-platform SDK library for C++, which depends on Neptune, a C++ runtime library. These packages are built and maintained on the Github repository, and are an important part of data integration with the rest of the platform.

4 Components

4.1 Presentation Layer:

This layer encompasses files and packages responsible for user information display, including fonts, language packs, user input handling, and view management.

4.1.1 User Input Module

Manages and processes user input from input devices such as keyboards, touchscreens, and remotes. This module handles the communication between these input devices and the software application, ensuring that user input is captured accurately and processed appropriately.

4.1.2 Windows Manager Module

A component within the software responsible for managing the graphical user interface (GUI) elements, including windows, buttons, and dialog boxes. Kodi, as a multimedia player, does not have its own built-in system for managing these interface elements, it relies on the libraries and functions provided by the operating system where it is installed to handle tasks related to window management. Kodi utilizes the built-in capabilities of the operating system to handle these actions relating to window management, which helps ensure consistency and compatibility across different platforms.

4.1.3 GUI Elements

GUI elements encompass a variety of visual components including windows, buttons, dialog boxes, and lists. These elements make up the actual user interface, where the windows management module deploys them, based on the specific system. They are crucial for crafting the program's visual layout, enabling users to navigate and interact with the platform's content effectively.

4.1.4 User Preferences

Composed of several modules, including the translation module, which manages the translation of the Kodi application. Allows users to select language preferences for display. The Audio Manager Module manages audio settings on the platform. Through the Font Module, the user can customize the appearance of text by adjusting the font and font size, and text styling. They are also able to customize the interface for specific system components, such as buttons and menus..

4.2 Business Layer

Comprises files and packages associated with server interactions, external libraries, and add-ons. It includes a client responsible for establishing connections with streaming servers. Additionally, this layer handles the reading and display of video and audio files utilizing codecs like DIVX and AC3. These codecs are essential for transforming raw data into a user-friendly viewing experience.

4.2.1 Python Module

Kodi's Python interpreter acts as a bridge between the user and Kodi's core functionalities, allowing users to develop custom add-ons (scripts and plugins) that enhance Kodi's functionality and user experience.

Scripts, in Kodi, are user-invoked sequences of instructions written in Python. Users can directly run scripts to perform specific tasks or actions within Kodi. A script can display a custom message, automate a series of tasks, or interact with external services. Scripts provide a way for users to customize their Kodi experience by executing specific functionalities upon request.

Not meant to be directly invoked by users, plugins are automatically invoked by Kodi when the user enters a designated virtual folder or triggers a specific event. Plugins do not introduce entirely new functionalities to Kodi; rather, they provide a simple method to present content listings within Kodi's native GUI interface. For instance, a plugin might scrape a website for streaming video links and seamlessly integrate these links into Kodi's interface, allowing users to access external content without leaving the Kodi environment. Kodi's Python interpreter supports various modules, such as xbmc, xbmcgui, xbmcplugin, xbmcaddon, xbmcvfs, and xbmcdrm, each serving specific classes and functions related to media manipulation and user interface management. Users can also install third party modules that are not a part of the source code through the root folder of the created addon.

4.2.2 Web Server Module

Grants users the capability to remotely manage their multimedia content through a web browser or other applications. Allows users to access their multimedia library even when they are not directly interacting with the Kodi application on their device. Multiple users can access and manage the same multimedia content simultaneously if they are connected to a shared local network.

4.2.3 Player Core Module

The Player Core Module is the vital component within the software system responsible for managing, processing, controlling, and playing multimedia content. Serving as the intermediary between multimedia files and the hardware or software necessary for playback, its primary objective is to deliver a seamless and high-quality multimedia experience to users of the application. The player core decodes multimedia files, ensuring they are in a format which is recognizable to the system. It then synchronizes these files, guaranteeing no delays between the audio and video components, a crucial aspect for maintaining a coherent viewing experience. The Player Core also oversees the entire playback flow, allowing users to pause and resume multimedia content at their convenience.

4.3 Data Layer

This layer is responsible for managing all files, functions, and content related to multimedia operations. It includes tasks such as saving files, organizing them on disks, streaming content from external servers, and transforming data into formats including FTP, RSS, and HTTP files.

4.3.1 Source Element

Manages and organizes the different multimedia input streams within the application. This element allows users to access a variety of storage devices such as HDD, CDs and DVDs, allowing them to directly play the multimedia content through Kodi. Furthermore, Kodi supports many network protocols including HTTP, FTP, SMB, RSS, UPnP and DAAP. These protocols enable users to access media sources over local and global networks. HTTP allows streaming media files from the internet, while FTP enables files from FTP servers. SMB facilitates sharing files within local networks, and RSS feeds can be used for streaming podcasts or news videos. UPnP and DAAP protocols help Kodi connect with other devices on the home network, allowing smooth sharing and streaming of multimedia content. Kodi is also able to integrate with streaming devices such as DVB (Digital Video Broadcasting) and OTA (Over-the-air) which allows users to live stream channels and even set up scheduled recordings.

4.3.2 View Element

Allows users to customize how their multimedia data is organized and displayed. Users can opt for a more traditional folder structure, where media and files are categorized and stored similarly to how they are organized on a computer. Users can also opt for a library structure, where media is automatically organized into different media types, including movies, shows and music.

4.3.3 Metadata Element

Provides additional information about the user's multimedia content. This includes details such as title, release date, description, genre, language, and other relevant information associated with the media. The component is made up of two subcomponents: tag readers and web scrapers. Tag Readers are responsible for reading and extracting metadata directly from the media files themselves. Multimedia files often contain embedded information, known as metadata or tags, which include details like the title of the content, artist, album, genre, etc. Tag Readers extract this embedded metadata and use it to correctly organize and display the content in the media library. Web Scrapers are responsible for obtaining additional metadata from online sources, such as websites dedicated to maintaining supporting information about various multimedia. Web Scrapers utilize search algorithms to extract relevant data from these online sources. For instance, when a user adds a movie to their library, Kodi's Web Scraper might connect to an online movie database, search for the movie title, and retrieve detailed information such as plot summary, cast, ratings, and cover art.

5 External Interfaces

Kodi maintains several external interfaces which can be used to interact with, or retrieve information from, the system. Kodi provides several interfaces for external control, including a powerful web server which can be used with any browser, the JSON-RPC interface, and the EventServer.

Kodi's web interface allows users to control and interact with their Kodi installation through a web browser, enabling remote control, library management, visual feedback, and more. When enabling the web interface, the user may specify a custom port number and an optional username and password for HTTP's Basic Access Authentication. Without an authorization protocol of sorts, it is possible to exploit Kodi's web interface to run malicious code in the installment location. To avoid this, users should not publicly expose their web interface directly to the internet, and should always use a basic form of authentication for their web interface.

The JSON-RPC is a HTTP socket-based interface for communicating with Kodi. It replaced the deprecated HTTP API, and offers a more secure and robust mechanism in the same format. Each method in the interface can have different security needs, allowing different clients to have different permissions in manipulating and writing data. JSON-RPC is designed so that most methods should behave roughly the same, and maintain consistency while hiding the mechanics of Kodi from the client creator.

The JSON-RPC has four functionalities that creators can choose from in order to interact with the system via transports, which have connectivity information that specifies the URL of the HTTP server, the user credentials and the maximum number of requests.. The response functionality should be present in every transport, as it enables a JSON-RPC request to be responded to with a valid JSON-RPC response, including both error messages and actual responses. The notification functionality includes server-side and client-side notifications, which are valid JSON-RPC requests with no id property. In Kodi, server-side notifications are used to inform clients of certain events, and relieve them of continuously checking to see if those events have occurred. The Direct file download functionality is the ability to directly download files from Kodi by calling Files.Download. Direct in this scenario, signifies that the download occurs within the JSON-RPC response of the Files.Download request. The final functionality is the Redirected file download, which enables the ability to indirectly download files from Kodi by calling Files.PrepareDownload, then using the response data to download the file over a different protocol, or another socket.

EventServer is the portion of Kodi that accepts remote device input on all platforms. The EventServer API is used to program event clients and simplifies interfacing input devices with Kodi, accepting commands from clients such as joysticks and iPhones. The EventServer listens for commands from event clients, which are items that can communicate using User Datagram Protocol.

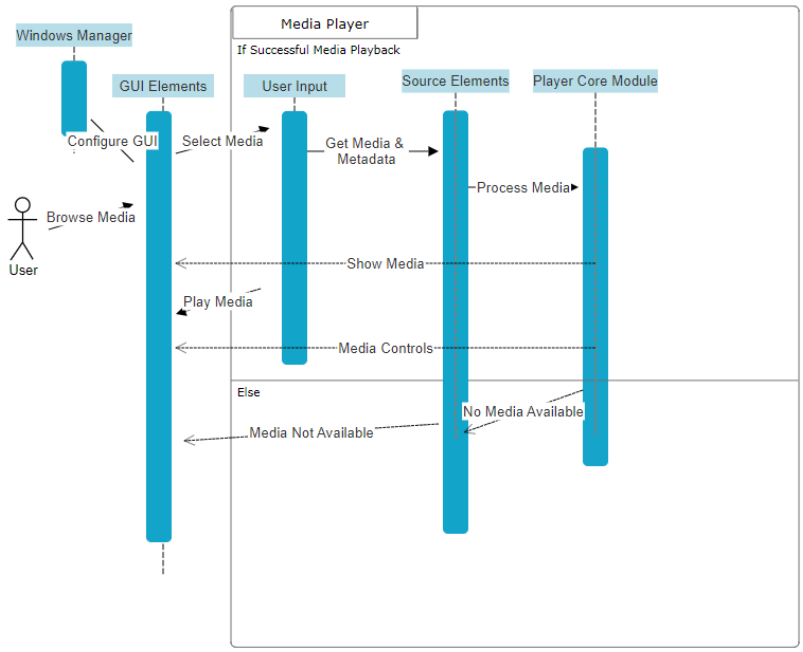
6 Use Cases

Kodi is a multi-platform multimedia center that is designed to operate across various supporting operating systems including Windows, macOS, Linux, Android and IOS. Its layered, modular design between a

client layer, presentation layer, business layer and data layer along with external interfaces allows for a robust multi purpose media-platform with a variety of use cases. To show the conceptual architecture a couple core use cases will be crucial in describing how the components interact with each other and the general flow of the platform. The two primary use cases would be first for Media Playback and then Media Library Management. The first use case highlights the flow from when a user selects a media file while the second highlights how a user can add or organize new media.

6.1 Use Case 1: Media Playback

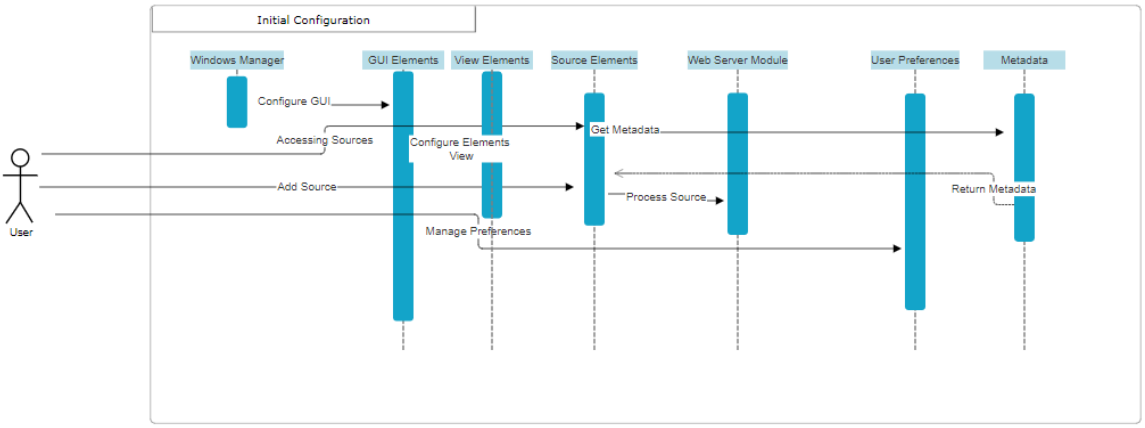
The Windows Manager Module manages the graphical user interface (GUI) elements, including windows, buttons, and lists. GUI Elements provide the visual components for the user to browse media. The user navigates through the list of movies using the input device. They select a specific movie title indicating their choice through the User Input Module. The media is accessed through the Sources Elements Component. The Player Core Module decodes the multimedia file of the selected movie using appropriate codecs to make it compatible with the system. The Player Core Module synchronizes the audio and video components. The module handles the playback flow. The Player Core Module outputs the video to the screen and audio to the speakers, displaying the movie to the user.



6.2 Use Case 2: Media Library Management

The Windows Manager Module manages graphical user interface (GUI) elements like windows, buttons, and dialog boxes according to device and OS. GUI Elements provide the visual components for the user to navigate and interact with the interface. The user accesses the "Sources" section within Kodi. Sources Elements manage and organize different multimedia input streams within the application. Users can explore various sources, including HDD, CDs, and network protocols like HTTP and DAAP. 4. The

user decides to add a new media source, initiating the process. The Web Server Module allows remote management through a web browser. User Preferences, including the Translation Module and Audio Manager Module, might come into play if the user needs to customize language preferences or audio settings while adding the source. When the user adds a new media source, Metadata Elements come into play. Tag Readers extract metadata embedded within media files, providing essential details like title, artist, and genre. Web Scrapers connect to online databases, extracting additional metadata such as plot summaries, cast, and cover art from the internet. Views Elements allow users to customize how their multimedia data is organized and displayed. Users can choose between a traditional folder structure or an automated library structure, where media is categorized based on types such as movies, shows, and music.



7 Supporting System Evaluation

The modular and add-on supported approach for application design helps ensure the support of future changes to the project. However, because Kodi primarily uses a layered architecture style, future-proofing the project can produce some challenges. Because changes in layers can impact subsequent layers, it may prove to be expensive and costly in order to implement changes. For example, if there is a change required in the data layer, the business layer must also be refactored in order to accommodate said change. This can potentially make future changes complicated.

On the other hand, Kodi's modularity helps isolate component changes. This is specifically applicable for the implementation of the add-on and plugin system. As these features are mostly independent (as a result of being built in Python and interpreted separately from the rest of the project), future changes to the project should not affect these extensions, assuming the changes are not targeting plug-in implementation. As a result, extensions and plugins are very future proof.

8 Division of Responsibilities across Developers

Seeing as Kodi is an open-source software, it follows that its developers are spread around the globe, with some contributing significantly more than others. Kodi is also in a stage far enough along that its contributions are specialized across several different areas, ranging from actual development to simple translations and add ons. Kodi utilizes the “issues” component of GitHub to facilitate division of responsibilities across developers. Developers are able to submit new issues to be fixed, as well as take on an issue to fix it themselves. At the time of this report, Kodi has 629 issues which are open, and 3,017 closed issues.

Once developers believe that they have resolved a particular issue, they can submit a pull-request, which would merge their code into the next version of Kodi. Kodi has entered a stage of development where it is more focused on resolving existing issues, rather than introducing new content and functionalities. Therefore, there haven’t been issues with division of responsibilities, as issues are isolated within themselves, and do not interact with each other. If a developer doesn’t fix an issue that they started, it is easy for another developer to pick up that issue and resolve it. All in all, Kodi’s stage of development, combined with it’s robust use of GitHub issues, ensure that there are no problems with the division of responsibilities among developers.

9 Conclusion

Kodi Media Player stands as a remarkable example of collaborative digital innovation in the realm of multimedia entertainment. Our report has offered an extensive exploration of Kodi's architectural underpinnings, from user interactions to its core functionality.

Kodi's architecture is primarily characterized by its layered approach, dividing the application into four distinct layers: the client layer, presentation layer, business layer, and data layer. Kodi's commitment to an open and flexible design is further exemplified through its use of an interpreter style, allowing users to enhance its capabilities through Python scripting. The repository style is another facet of Kodi's architecture, with the platform relying on GitHub and essential packages like Neptune and Platinum for code management. This collaborative approach is instrumental in maintaining a well-organized codebase, ensuring that Kodi remains adaptable to evolving user needs and preferences.

The report also discusses the potential challenges associated with future changes, given the interdependencies between different layers. Nevertheless, Kodi's modular and add-on approach provides a path for adapting to evolving requirements, particularly through extensions and plugins. The primary components of Kodi's architecture, including the presentation layer, business layer, and data layer, emphasize their respective roles and functions. This report also highlighted the significance of external interfaces for controlling and interacting with Kodi while underscoring the critical aspect of security considerations.

In conclusion, Kodi's architectural design is a testament to the power of open-source collaboration and innovation in delivering a versatile and user-friendly multimedia player. Its layered structure, support for

custom add-ons, and a range of external interfaces provide a flexible and extensible platform for multimedia entertainment, making Kodi a revered choice for users worldwide.

10 Lessons Learned

While completing our deliverable, we faced several challenges. Many of these challenges stemmed from the overwhelming amount of information available for Kodi. We found it difficult to sort through and portray the most relevant information due to the sheer amount available. To overcome this, we first created categories then sorted our notes accordingly. From there, it became easier to tell which sections did not have sufficient information and which sections did not require any more research. This also helped us stay organized while writing our report, and facilitated easy division of responsibilities among our team.

We also had several successes in completing our deliverable. By dividing up the information we retrieved, we were easily and efficiently able to split up the work among ourselves. We were also able to set strict deadlines for ourselves, which held us accountable and facilitated continuous progress. We also had success in our asynchronous collaboration. Over reading week, though we could not meet in person, we were able to easily communicate with each other, and conduct progress check-ins. All in all, we will take into account our challenges, implement ways to resolve them, and continue our habits of success for the next report.

Data Dictionary

Xbmc: Original name of KODI
Xbmcgui: Module used to manage the graphical user interface
Xbmcplugin: Module used to manage the plugins
Xbmcaddon: Module used to manage the addons
Xbmcvfs: Module used to manage the virtual file system
Xbmcdrm: Module used to manage Kodi’s DRM class

Naming Conventions

AC3: Digital audio coding technique that reduces the amount of data needed to produce high-quality sound.
DAAP: Digital Audio Access Protocol
DIVX: Digital Video Express
FTP: File Transfer Protocol
HTTP: Hypertext Transfer Protocol
JSON-RPC: JavaScript Object Notation (JSON) Remote Procedure Call (RPC)
RSS: Rich Site Summary or Really Simple Syndication
SMB: Server Message Block
UPnP: Universal Plug and Play

XBMC: XBOX Media Center

References

"Architecture." Kodi Wiki, n.d., kodi.wiki/view/Architecture https://kodi.wiki/view/Architecture

"Syncing and sharing." Kodi Wiki, n.d., kodi.wiki/view/Syncing_and_sharing

"Skinning." Kodi Wiki, n.d., kodi.wiki/view/Skinning

"About Add Ons." Kodi Wiki, n.d., kodi.wiki/view/About_Add-ons#Addon_system_using_python

"Archive:Kodi v15 (Isengard) FAQ" Kodi Wiki, n.d., kodi.wiki/view/Archive:Kodi_v15_(Isengard)_FAQ#Why_has_this_version_been_released_so_soon

Dreef, Kaj, Menno van der Reek, Koen Schaper, and Maarten Steinfort. "Architecting Software to Keep the Lazy Ones On the Couch." DelftSWA, April 23 2015, delftswa.github.io/

“Kodi 20.0 "Nexus" - Release” Kodi. n.d., kodi.tv/article/kodi-20-0-nexus-release/

Kodi. "xbmc." 2023, github.com/xbmc/xbmc