A3:
Architectural Enhancement Proposal for Kodi

Tuesday, December 5th, 2023

Group 23: Cache Me Some Waves

Erin Atacan - 20eea@queensu.ca
Eli James - 19ewj@queensu.ca
Trevor Withers - 17tkw1@queensu.ca
Joanna Bian - 18rb36@queensu.ca
Dallas Zhihao Nie - 20zn@queensu.ca
Cheng Qili - 18cl48@queensu.ca

Table of Contents

1. Abstract

2. Motivation

3. Proposed Enhancement
    - 3.1. Overview
    - 3.2. Interactions with other features
    - 3.3. Impacts of enhancement on high- and low level
    - 3.4. Impacted directories

4. Current Cache System
    - 4.1. Current State of Kodi
    - 4.2. Current System of Clearing Cache
    - 4.3. Enhanced Cache Management as a Solution

5. Use Cases and Diagrams
    - 5.1. Cache cleaning while playing movie
    - 5.2. Recommendations

6. Software Architecture Analysis Method
    - 6.1. Implementations
    - 6.2. Identifying Major Stakeholders
    - 6.3. Identify Stakeholders' Non-Functional Requirements
    - 6.4. Two Approaches' effect on Stakeholders' NFR
    - 6.5. Overall Best Approach

7. Potential Risks

8. Data Dictionary

9. Naming Conventions

10. Lessons Learned

11. Conclusion

12. References

1. Abstract

In this paper, our group, Cache Me Some Waves, discuss the development and comparison of two innovative cache cleaning mechanisms for Kodi. One is based on metadata/tags and the other will implement Artificial Intelligence. Our primary focus is to assess and contrast these implementations in terms of ease of implementation and overall effectiveness. We will aim to provide a comprehensive analysis of how each system operates within the existing concrete architectural framework of Kodi, examining their interaction with other subsystems and evaluating the distinction of their integration. Furthermore, we will highlight the respective advantages and potential limitations. By offering a comparison between the metadata/tag-based and AI-based approaches, our paper will guide the decision making process for enhancing Kodi's cache management, aiming to improve user experience and system efficiency.

2. Motivation

Kodi's user experience can be hindered due to its slow performance at times [1]. Some causes of slow speeds can be mitigated by the user, such as downloading a VPN or having faster internet speeds, but others could be improved on the development side. One such enhancement would be automatically cleaning up the cache on startup. As is, Kodi users can manually clear their cache if they notice throttled speeds, but in this paper, group 23 will propose a cache management system which automatically clears items from the cache based on how likely they need to be in the cache.

Kodi's cache serves the purpose of providing faster access to data recently used. Accessing data previously used is much quicker if it lies in the cache, but if the cache becomes full, access to needed items in the cache is slowed, and the capacity to add new things to the cache is worsened. As heavily relied on as the cache is, it seems prudent to automate the maintenance of the cache to optimize its speed, forgoing the need for the user to clear it once they notice their Kodi start to slow. We propose two (similar) options.

Our first approach is a subsystem which interprets the metadata of the multimedia and cleanses the cache of items that least need to be there, based on the type of content and access times of the cached items. The second approach would similarly purge the cache of items least needed to be there, but instead does so based on an AI's recommendation of which content in the cache is least likely to be accessed again.

An added advantage of this cache clearing system is that it has architectural similarities to a content recommender. . Whether achieved with AI, or pattern-matching metadata tags on multimedia content, either option is effectively taking the user's recommendations into account in order to delete items from the cache they are least likely to need again. So, while we are at it, why not extend the subsystem's capability to a multimedia content recommender to aid in the browsing and selection of new multimedia for the user?

3. Proposed Enhancement

3.1. Overview

As mentioned in the introduction, The first approach to automate cache maintenance is the implementation of a metadata-based cache cleanser. Its aim is to remove items least likely to be needed again. Content in the cache mustn't be there if there isn't a reasonable probability that it is going to be needed again in the short term, and should therefore be removed from the cache. The primary consideration when implementing this subsystem would be to consider how to characterize this probability. The removal of unwanted items will be a function of the types of multimedia in the cache, as well as the duration of time the content has been in the cache. We can assert with reasonable probability that items that have been sitting in the cache for a long time can be removed from the cache, and we can also reasonably assume that a user's need for a specific type of content is fairly homogeneous. Suppose, for example, a user streams music through a plugin on Kodi; in this case, it would be reasonable to remove cached items pertaining to movie and TV show types, as they are less likely to be accessed next.

Further, it is crucial to note that automating purging poses little hindrance to the performance of the cache as a whole, over time. Specifically, Kodi offers no current automated cache maintenance. Thus, removing the least-likely to be accessed content from the cache is purely advantageous to the user's experience of Kodi's speed, and incorrectly removing a cached item that *in fact* was going to be accessed again in the cache poses little negative performance in speed as noticeable by the user, as that content still lies in the device's main memory.

An item "deserves" the least to be in the cache if it is the oldest, and least similar type of content compared to other items. Conversely, an item ought to be in the cache no matter what if its type is the type of multimedia most common in the cache, or if it is the new/newer/newest in the cache. These observations offer a best and worst case need for what content needs to be in the cache, but metadata tagging will offer little guidance when deciding between the merit of staying in the cache of 2 items nearer to the middle along the spectrum of these two extremes. This could be feasible with the proposed system, but would be highly dependent on the comprehensiveness and standardization of metadata on the content from different sources. Consider for example, two plugins with the same multimedia available in both. If a user downloaded a movie from both of them, and watched both, and some time later went to rewatch them, how would we compare the merit of the two titles belonging in the cache on the startup's automated cache cleansing? They have similar “newness” in the cache, and have the same metadata about what type of multimedia they are, but there is no guarantee that metadata about the cast, director, studio, genre are standardized or even available. In section 3.2 we will investigate a remedy to this case, but regardless, we have concluded that any cache maintenance is better than none, so in the worst case only deleting the oldest/most dissimilar content and keeping the newest/most similar is a great improvement on no cache pruning.

To this point, we have described a high-level overview of a cache maintenance system which runs on startup and aims to keep Kodi performance as fast as possible. We achieved this with a metadata approach which prunes the cache of items as a function of their metadata tags and time spent in the cache. In the introduction, we made mention of a recommendation system which can be derived from the cache maintenance system. Although not the primary focus of this paper, we will discuss the recommendation system in brief here, and its interactions in section 3. 2..

We could extend the use of metadata interpretation to the use case of providing the user with multimedia content recommendations. The metadata tag system could be used to provide content with similar tags to the content in the cache to the user. This would be difficult for the aforementioned reason of there being no standardization of metadata attached to multimedia between different plugins. Although unlikely to be fruitful, we could consider an alteration in the implementation of the cache cleaning subsystem in such a way that we could more easily extend its capabilities to also serve as a content recommender. That implementation would of course be using an AI of sorts. This system would
A) Remove cached items that ought to be pruned based on the user’s past usage and B) Recommend content to the user based on past content consumed. In the former case however, we would need the cache maintenance system to have some memory of past trends and user data, as well as needing to run a more computationally expensive operation on startup in order to clean the cache, which is not ideal and somewhat defeats the purpose of trying to speed up the user experience. The second reason we make mention of this recommender system, but only in terms of theory, is that implementing a content recommender which can interface with any plugin would be a very complicated feat. Instead, the recommender would have to be a ‘widget’ or overlay of sorts, which based on user data and metadata from the plugins’ content, recommend multimedia outside of the plugin itself.

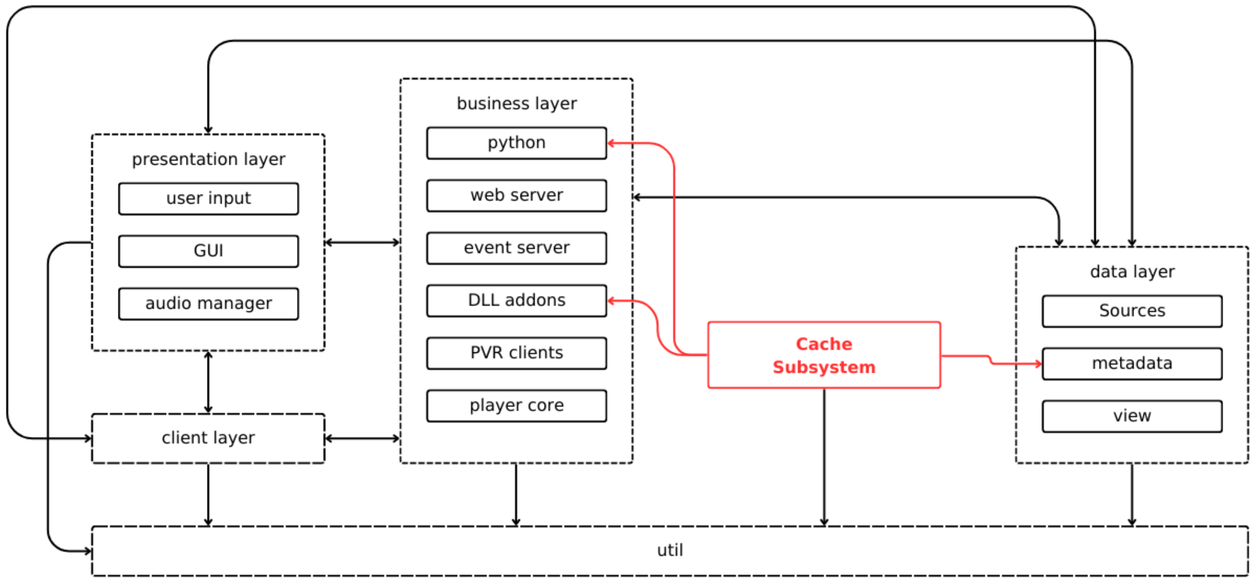
Therefore, the rest of the paper will make little mention of the extension of the cache maintenance system into a content recommender.

3.2. Interactions with Other Features

- Cache: the cache maintenance system is dependent on the cache, and will read and remove the cached items based on its algorithm.
- Metadata: metadata of files are used in the cache maintenance system, as the system reads tags to determine the 'newness' of the file and whether or not it will appear on the recommendation list.

The proposed enhancement would best be implemented as its own subsystem between the business and data layers. This provides connections no longer than 1 layer to both the plugins within the business layer and the metadata/cache which reside in the data layer.

3.3. Impacts of Enhancement on High- and Low-level



The proposal of a metadata tag reader to be used for automating cache cleanup would have minimal impact on the low-level performance of Kodi. Whether the cleanup happens on startup or in another instance like just before streaming content, it still only occurs infrequently and at an algorithmic level would be implemented with simple pattern matching between tags.

At a high level, consistent faster access to recently accessed multimedia may provide tangibly faster performance speeds as observed by the user.

3.4. Impacted Directories

- kodi\xbmc\filesystem\CacheStrategy.cpp (Data Layer)
- kodi\xbmc\addons\*.cpp
- Python/Plugins
- Web Server/Plugins
- Metadata/Tag Readers

4. Current Cache System

4.1. Current State of Kodi

Kodi, as a media center, boasts significant capabilities, but it's currently hindered by several critical issues that adversely affect the user experience. The user interface is a primary concern, often described as clunky and outdated. This makes navigation through menus feel unintuitive and sluggish, a particular problem for new users unacquainted with the layout. The aesthetic, although functional, lacks the modernity and sleekness of Kodi's competitors, potentially alienating users seeking a more contemporary media experience.

Performance inconsistency is another significant issue, particularly pronounced in older or less powerful hardware. Users frequently report problems like lag, crashes, and prolonged loading times, all of which are magnified on devices with limited capabilities. This variability in performance is largely attributed to inefficient resource management and the tendency for the software to become increasingly bloated, especially when numerous add-ons are installed.

The add-on ecosystem, while one of Kodi's most appealing features, offering extensive customization and functionality, also presents notable drawbacks. It creates a fragmented experience with varying levels of quality and reliability in add-ons. This situation is further complicated by the prevalence of broken, outdated, or unofficial add-ons that may pose security or legal issues.

Additionally, the management of these add-ons, along with other settings and preferences, is far from straightforward. The interface for this task is non-intuitive, often necessitating users to seek help from forums or guides. The absence of a centralized and streamlined management process for add-ons and settings further complicates the user experience.

Kodi's approach to cache management is yet another area in dire need of enhancement. Currently, clearing the cache is a manual and time-consuming process fraught with the risk of inadvertently deleting essential files. The lack of an efficient, automated system for cache management leads to decreased performance over time, necessitating frequent manual intervention by the user.

4.2. Current System of Clearing Cache

The existing process for clearing cache in Kodi is both cumbersome and inefficient, leading to user frustration and latency issues. Users are required to navigate through a maze of menus to the 'File Manager' to locate the cache directory, a task that is particularly challenging for those not technically inclined. The varying cache locations based on the operating system add an extra layer of complexity.

Upon reaching the cache directory, users are faced with the risky and tedious job of deciding which files to delete. The threat of accidentally removing important files is significant, compounded by the lack of clear guidance on differentiating disposable cache files from essential data. While cache management add-ons are available, they introduce additional complexity and potential compatibility problems, not to mention issues with reliability and user-friendliness.

The need to restart Kodi after clearing the cache adds another layer of inconvenience to the process. This entire procedure, requiring regular repetition for maintaining optimal performance, starkly highlights the user-unfriendliness of the current system and underscores the pressing need for improvement.

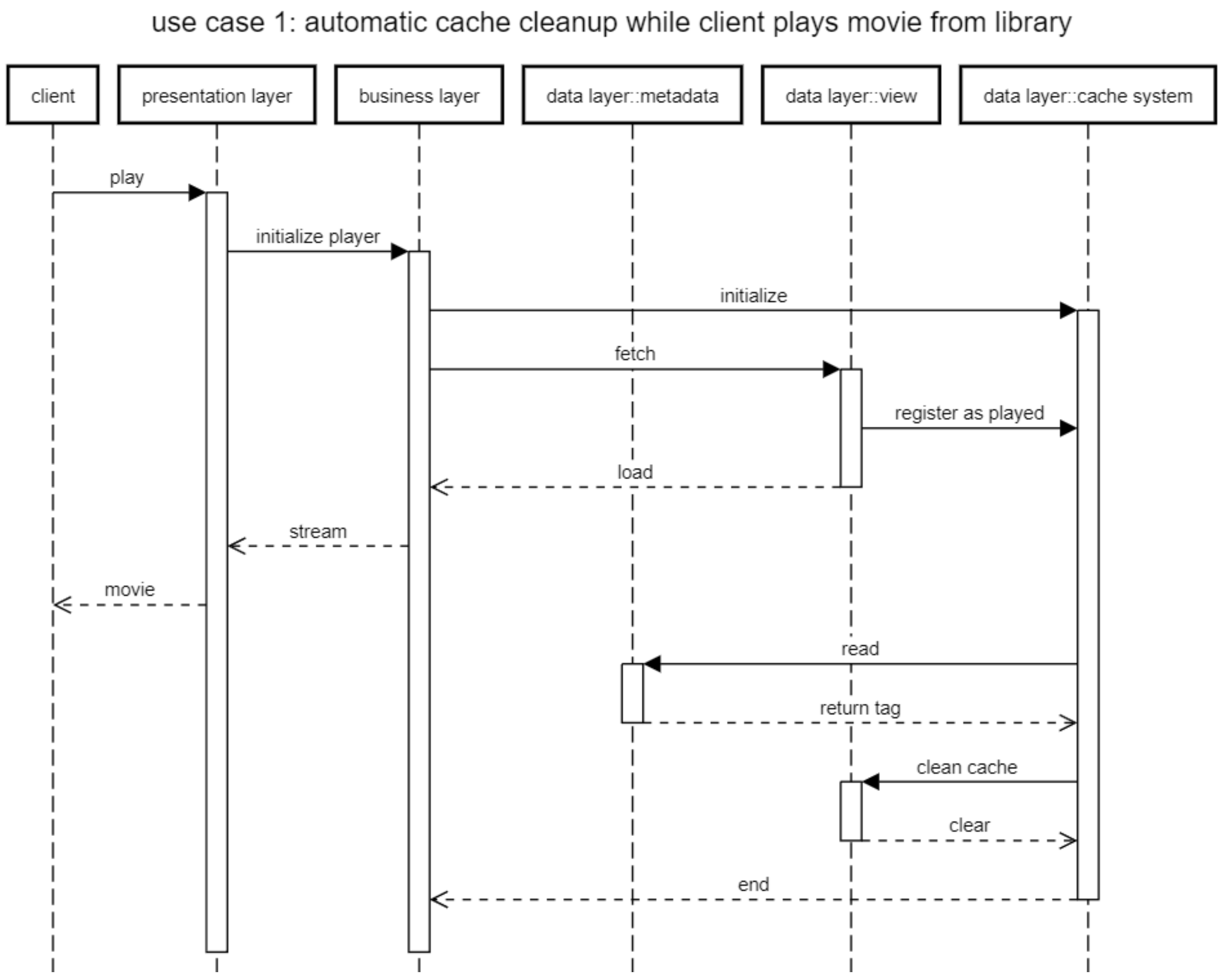
4.3. Enhanced Cache Management as a Solution

Implementing a sophisticated cache management system in Kodi could substantially address its existing challenges. Streamlined cache management would directly improve the responsiveness of the software and enhance the user interface experience. Users would experience less lag and quicker loading times, leading to more efficient navigation and interaction, particularly beneficial for those using lower-end hardware.

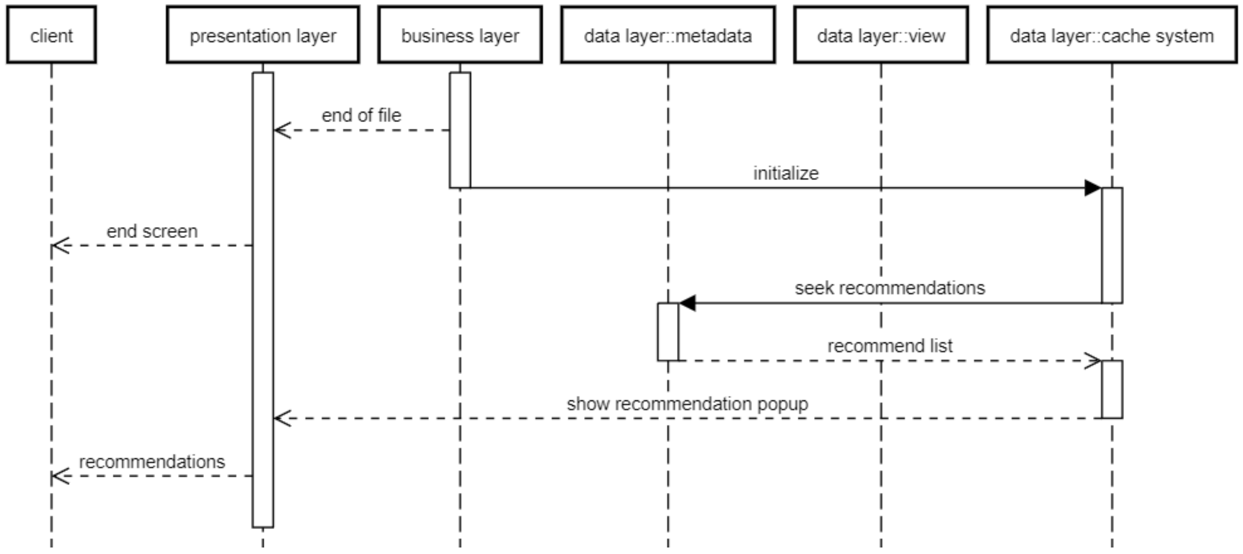
Kodi's performance issues and inconsistencies, often exacerbated by the use of multiple add-ons, are largely related to inefficient cache usage. An automated cache management system could mitigate these issues by regularly eliminating unnecessary data, promoting smoother and more stable performance. This improvement is crucial for ensuring the functionality of various add-ons, which can suffer significantly from a bloated cache.

5. Use Cases and Diagrams

5.1. Cache cleanup while playing movie



use case 2: end screen movie recommendations



6. SAAM Analysis

6.1 Implementations

Implementation 1: Use metadata pertaining to the duration of time an item has been in cache and the frequency of its data type to automatically remove content from the cache. Specifically, items that have been in cache the longest, and those with the least common data type will be removed at every system start up.

Implementation 2: On top of the metadata tagging approach, use AI based on users' past usage to develop the pattern of the cache that is saved or cleared. Based on the cache that is not cleared, develop a recommender system for users.

6.2. Major Stakeholders

The stakeholders of Kodi are the Users and Developers. Users are concerned with how an enhancement will affect their experience with the system. Developers are more concerned with how the addition of a subsystem or code will affect the system as a whole.

6.3. Stakeholders' Non-Functional Requirements

User:
Performance: The enhanced system should be more responsive and quicker.
Security: Users want their data and metadata to be protected.

Availability: Users do not want any maintenance to interrupt their usage of the system
Usability: Users want any maintenance to the system to be done with as little work as possible. Users want the system to be able to be used by one with any level of coding knowledge.

Developers:
Maintainability: Developers want this software addition to require the least amount of maintenance
Scalability: Since Kodi is a multi-media player platform, the developers would want an automatic cache management system to grow with its platform
Evolvability: Developers want the enhancement to not have any negative consequences on the rest of the system lest it change. It should evolve with the rest of it.
Testability: The performance should be easy to test and not have an ambiguously positive effect in order to justify it.

6.4. Two Approaches' effect on Stakeholders' NFR

NFRs	Implementation 1	Implementation 2
Performance	Since this approach applies a rule based to determine what part of the cache is deleted, the improvement in performance is limited. However, since part of the cache is being deleted, there will always be improvement in the response time of Kodi.	The AI approach could be more accurate than implementation 1, because the AI has a higher ability to model the user's preference. This leads to slower speed and larger file size, but more accurate cache deleting mechanism. It could increase the speed at last for Kodi.
Security	This approach does not require any external plug-ins. Through transparent documentation, users can know the exact data that the algorithm uses.	AI's approach would be less transparent compared with the first one, since it's more like a black box to find patterns. Although the user can know the module is dealing with cache, they don't know the exact situation of this module.
Availability	Since this algorithm would run at startup, the user would have to wait a certain period of time (depends on its time complexity) in order to access	It's similar to the first implementation, but the user might wait longer for the AI model to process the data.

	Kodi.	
Usability	The enhancement that this approach offers requires no intervention from the user.	It’s similar to the first approach. There is no intervention from the user.
Scalability	The metadata used to determine which content is removed from cache is determined by the rules in the algorithm. More rules can be added by the developers without any consequences.	AI’s approach will require the system to take in larger models with higher accuracy or smaller models with less accuracy. Those models can be firstly trained out of the system and deployed to the system later.
Maintainability	If the developers desire to update the rules, it may require occasional maintenance.	This approach is harder to maintain as it is less transparent than the first approach.
Evolvability	As Kodi grows, so may this algorithm, to include more data types, or new features that would use the cache.	As Kodi grows, with more data exposed, the AI model will evolve and improve.
Testability	Easy to test.	Hard to test as it takes time for the AI to learn the pattern and due to the level of transparency of this approach.

6.5. Overall Best Approach

Overall, the best approach would be the first. The simple rule based algorithm offers performance improvements for the users while requiring little maintenance for the developers and interference with other parts of the system. It has great evolvability because we are not limited to the data types mentioned above for cache pruning; more data types or features can be added in the future. This is an easy, non-intrusive way of improving the system’s performance. As the second approach with an AI model, it is slower and needs more memory. Plus, with the complexity and low transparency of the model, it is hard to evolve, maintain, and test.

7. Potential Risks

The two ways for cache cleaning might increase cache miss due to deleting some important cache. For the metadata tag method, the tag could be insufficient to describe or evaluate which parts of the cache should be cleared. The performance of this way is highly dependent on the tag's accuracy, which is hard to control and test. For instance, if the cache cleaning mechanism uses file type tag for cache cleaning, it could work for some users who use Kodi mainly for listening to music or watching movies, but would fail for users who use it for multiple purposes. The effect of the cache cleaning mechanism is also hard to test, since it's hard to simulate the user cases and guess their preferences for rewatching old movies.

For the second AI tag method, although it might locate the cache needed to be deleted better compared with the metadata tag method, it would be cumbersome and slow due to the data size and manipulation speed. And as discussed in 3, this method does not work on some devices, such as raspberry pi machines.
For security of users, this module is a file-level deleting module. It could delete some important files with a similar path. If the module is not well established and tested, system-level problems might occur.

This module is dependent on the data layer and business layer. It would lower maintainability that, if the system's hierarchy needs to be changed in the future, the cache deleting module would need to be modified as well.

Those are the main considerable potential risks for the new cache clearing module.

8. Data Dictionary

Cache - a software component that stores some data so that future request for that data could be faster
Metadata- data describing other data, for example, file type of one file
Subsystem- a system of another larger system
Recommend system- a system uses data to recommend products to the user
Performance -The speed and accuracy of the program
Stakeholders - people or group affected by the program project
Maintainability - the hardness of maintaining the system
Scalability - the ease to extend the system to a larger data size or add new subsystem into it
Plugin - a software component that adds specific features to Kodi
Compatibility - the ability of a software system to work with other products/systems

9. Naming Convention

SAAM - Software Architecture Analysis Method
AI- artificial intelligence, intelligence of machine or software
VPN-Virtual Private Network, a digital connection between your computer and a server owned by a VPN provider
NFR - Non Functional Requirement, the criteria used to judge the system that not related to its specified functional behavior
User Interface (UI) - the means through a user interacts with a software application

10. Lesson Learned

For systems that might save much cache, there should be one cache cleaning mechanism. And this should be clearly documented that even someone who does not know a lot about computer science should be able to manipulate cache to speed the system up. The development of Kodi is highly technical, so the documents take the user as highly technically inclined people. However, it's hard for everyone to find the file path and right format for developing a cache manipulating method. We need to take care of users who have low technical skills.

For cache manipulating strategy, we need to consider enough perspectives for evaluating the effectiveness and potential risks before setting it up. The accuracy and speed may conflict with each other, like the AI approach is more accurate but the metadata approach is faster. We weigh the pros and cons and get the conclusion that the metadata tag would be more suitable for Kodi.

11. Conclusion

This paper explored the development and comparison of two types of cache cleaning mechanisms for Kodi. One which leveraged metadata/tags, and the other using Artificial Intelligence. We assessed these approaches in terms of ease of implementation and effectiveness, highlighting their integration into Kodi's architecture and interaction with other subsystems. Our analysis revealed the potential of these systems to significantly enhance Kodi's performance and user experience. While the metadata-based approach offers simplicity and ease of implementation, the AI-based system presents a more advanced but complex solution.

We concluded that any form of automated cache maintenance represents a substantial improvement over the current manual system. Despite the challenges, particularly in standardizing metadata and ensuring compatibility across various plugins, these enhancements are pivotal in evolving Kodi's capabilities, offering a more responsive, efficient, and user-friendly media center.

12. References

Garland, Ian. “Why Is Kodi so Slow? Here’s How to Increase Kodi Speeds.” *Comparitech*, Comparitech, 6 Nov. 2023, www.comparitech.com/kodi/kodi-running-slow/.