Proposed Enhancement for Kodi

CISC 322 Assignment 3 Report

December 5, 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

Abstract

This report provides a detailed overview of a proposed Screen Time Monitoring enhancement to Kodi. This proposed feature would allow users to set screen time limits and would prevent media from playing if these limits are exceeded. Our report evaluates two different implementation approaches: Usage Monitoring with System Events and a Predictive Machine Learning Model.

Through a Software Architecture Analysis Method (SAAM), the report examines the potential impacts on stakeholders through a careful exploration of the Non-Functional Requirements for both Users and Developers. Once an optimal implementation has been identified, the report dives into the effects on the Non-Functional Requirements, the effects on the Software Architecture, identifies the impacted files and directories, and lastly explores the potential risks that this implementation poses. The use cases of setting a screen time limit and exceeding a screen time limit are presented to provide a clearer overview of the functionality and interactions of the enhancement. Lastly, the report concludes with final thoughts, and lessons learned throughout the creation of the report.

Proposed Enhancement

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. With the first Beta release of Kodi 21.0 “Omega”this past October, Kodi is clearly a stable platform which continues to grow within the multimedia sector [1].

As media players become more sophisticated, and as our world becomes increasingly digitized, average screen time has also been on the rise, with an alarming associated climb in anxious and depressive symptoms [2]. Increased screen time has been linked to negative effects throughout the general population. Children are more likely to suffer from behavioural problems and poor academic performance. Adults are more likely to suffer from obesity, and poor mental health [4]. In light of these escalating concerns and their pervasive impact across age groups, it is paramount for society to heed the warnings embedded in these findings and consider proactive measures to mitigate the adverse consequences of escalating screen time.

Recognizing the importance of reducing screen time, our group proposes a Screen Time Monitoring Component for Kodi. The Screen Time Monitoring component would have a predefined limit of two hours, the recommended limit for adults, and would lock the media playback function when the limit is reached [5].The user has the option to set a password which would bypass the limit, and would be able to increase or decrease the limit to fit their preferences. Kodi would also alert users of their current screen time use with a small notification in the top left corner, so as not to disturb the viewing experience. These notifications would occur 25%, 50%, 75% and 90% through the allotted screen time. This proposed enhancement allows users to continue to enjoy the Kodi experience they love, while also encouraging responsible amounts of media consumption.

Interactions with Existing Architecture

A new subsystem containing the Screen Time Monitoring Component will be needed to handle the screen time settings, the monitoring of screen time, the notifications that occur while viewing content, and the lockout functionality that is triggered when screen time has been exceeded. The new subsystem would be placed in the Interface Layer, as the Screen Time Monitoring Component relates directly to the functioning of the system.

The Screen Time Monitoring Component would need to interact mainly with the Player Core Module and PVR module, which handle media playback. The Screen Time Monitoring Component would implement locks which would effectively block access to media playback. These locks would be triggered when the screen time limit has been exceeded, and they are unlocked when the user either enters their screen time password, or a new day begins.

The Screen Time Monitoring Component would also need to interact with the Presentation Layer, specifically the GUI Elements and the User Preferences. The GUI Elements would be necessary in order to implement the screen time notifications. The User Preferences would be the home of the Screen Time Monitoring Component settings, and would be where the User sets their preferred screen time limit and adjusts their screen time password.

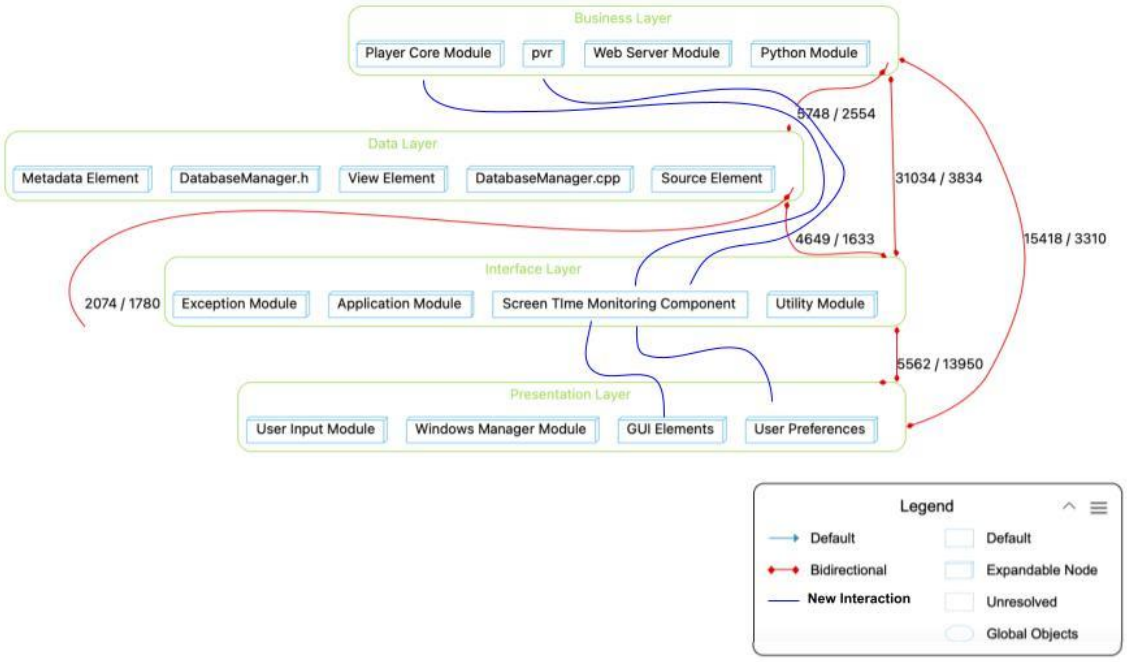

Figure 1: New Interactions from the Screen Time Monitoring Component

Options for Implementation

Option 1: Usage Monitoring with System Events

The first option for implementation of the Screen Time Monitoring Component is to implement it through usage monitoring with system events. In this option, the Screen Time Monitoring Component would system events which continuously send data about the user's watch time to the event listeners which trigger the notifications and lockout. It would also be customisable, as the event listeners can be placed to monitor for actual media playback, instead of simply application use. This allows for an accurate reflection of the user's screen time, which is crucial in ensuring that they stay under the screen time limit.

Implementing usage monitoring with system events would create a large number of interactions between many of the components. The event listeners may need to be placed in differing parts of the architecture in order to accurately perform their task. This would introduce a peer-to-peer style for the Screen Time Monitoring Component. This would allow each of the event listeners to interact with each other without needing to be contained within the same module, increasing flexibility in the implementation, as well as extensibility for future modifications.

Option 2: Predictive Machine Learning Model

The second option for implementation of the Screen Time Monitoring Component is a predictive machine learning model, which would be able to predict when a user is likely to exceed their screen time limit. By identifying relevant information, such as day of week, time of day, and type of media, the model will be able to predict the user's tendency to go over their screen time limit, as well as provide them with insightful feedback on their screen time habits. This data would be incredibly helpful for a user who was trying to cut down on their screen time use, as the insights would help them be more mindful of their viewing habits.

Implementing a predictive machine learning module would require a self contained module where the majority of model training will occur. It would also need data observers that capture the user's data across several aspects of the application. The observers would need to capture information such as when the user watches content, what content they watch, and how long their watch session is. This information would be transmitted to the self contained module and used to train the model. The architectural style of this implementation would be repository style due to the flow of data through one central self contained module. This would facilitate ease of upkeep, as small adjustments can easily be made to the model without impacting the installation of the data observers.

Software Architecture Analysis Method

The Software Architecture Analysis Method (SAAM) is introduced as a five-step approach aimed at addressing the challenges in evaluating software architectures, a crucial but often neglected aspect of software engineering research. The motivation behind SAAM stems from the inadequacy of a common language to describe diverse architectures and the absence of a clear method for understanding

architectures in the context of an organization's life cycle concerns, particularly software quality factors like maintainability, portability, modularity, and reusability. SAAM, as a solution, operates through three perspectives—functionality, structure, and allocation—allowing for a comprehensive understanding of software architectures. By establishing a clear and consistent language for describing architectures, SAAM aims to facilitate comparison and analysis across different architectural designs, emphasizing its utility in assessing and improving software quality throughout the development life cycle[6]. In the context of this report, SAAM is applied in determining the optimal implementation method for the Screen Time Monitoring Component.

Stakeholders

There are two major stakeholders that must be considered:

Users: Users can be classified as anyone who uses the Kodi multimedia player for their own personal use. Activities that a user participates in include viewing multimedia, adjusting a screen time limit, and hitting a screen time limit.
Developers: Developers can be classified as the people who work to develop and maintain Kodi. They may work alone, or in teams, and bring improvements and updates to the feature when necessary.

Identifying Non-Functional Requirements for Stakeholders

Stakeholder	*Non-Functional Requirements for Screen Time Monitoring Component*
User	*Security*: The user's viewing data should be stored safely and securely on their own installation to avoid data breaches. All screen time calculations and operations should occur on the user's machine to decrease the likelihood of a large-scale data breach of Kodi. *Privacy:* The user's data should be anonymized in all external uses. The data should not include any identifying aspects such as location of user, installation type, or user name. *Usability:* It should be easy to modify the screen time limit and view how much of the allotted time has been used. The addition of the Screen Time Monitoring Component should not impact the actual viewing experience on Kodi. The Screen Time Monitoring Component must also allow access to critical system information and should not completely shut out the user. *Performance*: The screen time monitoring must not slow down the application in any way. The lock-out feature must not lock the user out of critical parts of the application. *Availability*: The Screen Time Monitoring Component must always be able to accurately track the user's watch time, and should rarely suffer from outages. The component should not fail under normal circumstances.

	Accuracy: The recorded screen time must be within 2% of the actual screen time of the user.
Developer	*Maintainability:* Errors and bugs should be easily fixed. The component should be managed by a dedicated team to maximize efficiency and uptime. A robust test scheme should also be used to ensure stability after any changes. *Scalability:* The Screen Time Monitoring Component must be able to scale to handle multiple users on a specific installation. The metrics must be kept separate for each of the users, and the screen time of one user should not cause lockout for another user. *Performance:* The Screen Time Monitoring Component must not slow down the application in any way. The lock-out feature should ensure that the user still has access to critical parts of the application. When the user hits the screen time limit, the application should not take longer than 20 seconds to initiate the lock out process. *Evolvability:* The Screen Time Monitoring Component should support the addition of new and modern technologies in the future, without entirely rewriting the feature.

Evaluating Implementation Options & Selecting the Best One

Non-functional requirements (NFRs) are critical in Kodi as they define the characteristics that are necessary for the system's overall performance, reliability, and security. Below are the important NFRs regarding both enhancements suggested. A high rating indicates great compatibility with the NFR, while a low rating indicates that such NFR will be difficult or problematic to implement.

Non-Functional Requirements	*Usage Monitoring with System Events*	*Predictive Machine Learning Model*
Accuracy	*High:* Usage monitoring through system events would result in a high level of accuracy, as the data is directly passed between the different parts of the application, reducing latency between event and response.	*Medium:* Training a predictive machine learning model takes a significant amount of time, and it would not be immediately able to provide useful statistics and insights. The more the user interacts with the application, the more accurate the model will become over time.
Availability	*High:* With a peer to peer network implementation, this approach would experience minimal downtime. The lightweight implementation makes it unlikely to experience significant problems. Additionally, since only one	*Medium:* With the large amount of compute power required for training and using the predictive machine learning model, it is likely to experience some outages. To circumvent the likelihood of critical failures, training the model

	user can view content at a time, the system is unlikely to experience critical failure.	should occur during late night hours, when users are least likely to be viewing content through Kodi. Additionally, should there be an error within the model, it would render the entire system unusable, which would result in a critical failure.
Evolvability	*Low:* Usage monitoring through system events leaves little room for innovative new features to be introduced in the future. Seeing as the system events have only one purpose in this implementation, it would be difficult to expand on the functionality in the future.	*High:* Predictive machine learning models are incredibly flexible and continue to be refined and improved each day. It would be easy to add new functionality to the Screen Time Monitoring Component, such as a natural language processing component, that could respond to the user's queries about screen time, or adding a personalized training plan to decrease screen time.
Maintainability	*Medium:* Implementation of the usage monitoring system may vary across different installations. This would require multiple implementations of the monitoring system, increasing the amount of time that would be dedicated to maintaining the system.	*High:* The predictive machine learning model can be abstracted into a component which can be used across various platforms. This would increase the maintainability as system wide changes can be made by simply changing one aspect of the component.
Performance	*Medium:* Usage monitoring with system events would require continuous monitoring, which may begin to degrade the performance of the application. However, the actual implementation itself would be relatively light weight, as the event listeners would not be monitoring for every possible event. This would enable a quick response time after the screen time limit has been exceeded, with the potential tradeoff of a slower experience throughout the program.	*Medium:* Training and usage of a predictive machine learning model is energy intensive, as it analyzes data and provides the result to users. This may cause delays across the system as some of the compute power is used for the model. This would create a slower experience when interacting with the model, with the tradeoff of producing useful data and feedback that can aid the user in changing their screen time habits.
Privacy	*High:* This approach has more privacy because system events do not contain any identifying details, they simply trigger when a certain event occurs. Therefore, the data would already be anonymized, and any data breach would have great difficulty in gaining information specific to a particular user.	*Low:* In a machine learning model, anonymizing and de-identifying data results in a significant degradation in model accuracy [3]. Therefore, an increase in privacy would result in a significant decrease in performance, and would negate the benefits of the predictive machine learning model.

Scalability	*High:* Scaling the usage monitoring option to handle multiple users within a single installation would only require an additional data tag which states which user is currently viewing the content. The screen time could then be categorized by each user.	*Low:* Scaling the predictive machine learning model to handle multiple users within a single installation would require a large increase in compute power and memory. There would need to be a predictive machine learning model associated with each of the users, and distinctions would have to be made between each of the user's viewing habits.
Security	*Medium:* System events can be triggered and monitored within each individual installation. No data needs to be sent to an external server in order for this approach to function correctly. However, it is possible for a malicious user to gain access to a particular Kodi system and falsely trigger events which would indicate that screen time has been exceeded, locking a user out of the viewing component.	*Medium:* Though each model would be individually trained to each user, the developing team may need to extract some user data or results from models, which would require data to be stored on an external server. This would put the user's personal data at risk of a data breach.
Usability	*High:* Usage monitoring with system events has good usability, though it wouldn't involve a significant amount of direct interaction with the user. The usage of this option would be straightforward, as it would only require the user to specify their screen time limit.	*High:* Usage of the predictive machine learning model would also be straightforward, as the complexity would be abstracted from the user through the interface. The user also has flexibility in how much they interact with the model. The user can simply let the model remind them when they are likely to exceed their limit or they can take a more hands on approach by interacting with the model to see their personal trend data.

Figure 2: Table depicting the Non-Functional Requirements and their compatibility with each implementation method

Based on the aforementioned NFRs and their compatibility with both options, the best option for implementation is the Usage Monitoring with System Events. Although the Predictive Machine Learning Model provides innovative statistics and a modern interface, the scalability, availability and privacy are large points of concern. The lightweight of the Usage Monitoring with System Events as well as its security, privacy, and scalability make it the optimal option for implementation.

Effects of the Enhancement

Effects on the Non-Functional Requirements

The implementation of the Screen Time Monitoring Component through Usage Monitoring with System Events would have considerable impact on the Non-Functional Requirements of the system. The largest impact would be on the evolvability of the system. Seeing as System Events are not an innovative technology, the evolvability of the system would be negatively impacted. This somewhat dated technology, though reliable, would require a large overhaul in the future if a developer wanted to add new features to the Screen Time Monitoring Component.

The maintainability of the system would be slightly negatively impacted. Implementations of the event listeners may vary across installations, increasing the amount of time dedicated to maintaining the system. The testability of the system would be positively implemented. As observed in the Non-Functional Requirements table, this particular implementation has robust privacy and security by default, due to its ability to store all data locally. This particular implementation would also be straightforward to test, since the interactions between components are clearly defined.

Effects on the Architecture

The implementation of the Screen Time Monitoring Component would have noticeable impacts on both the high and low level conceptual architectures. The integration of new features, such as the media viewing lockout system and the continuous screen time monitoring component would create a large footprint in the application. The most noticeable effect would be the creation of the Screen Time Monitoring Component within the Interface Layer.

At a high level, the addition of the lock-out feature would require significant changes within the PVR Module, Player Core Module, and the Settings component in the User Preferences Module. These changes would facilitate the actual lockout of the user and allow them to set a screen time limit for themselves. The Screen Time Monitoring Component would also implement its own subsystems. A Driver subsystem would be created, with the main purpose of triggering the appropriate event when necessary. A Monitoring system would be created, with the main purpose of observing the event listeners. The Monitoring subsystem would interact with the Driver subsystem, calling on the appropriate driver to take a certain action, for example, locking the media function when the screen time limit is reached. The Monitoring subsystem would also need to interact with the component that stores the date and time in order to reset the screen time limit each day.

At a lower level, the implementation of the Screen Time Monitoring Component would require significant changes within the PVR Module and Player Core Module. Event observers would need to be placed strategically to monitor time spent watching content. These observers would need to interact with the Screen Time Monitoring Component to communicate their data. The Driver subsystem would analyze this data and call the appropriate action. Code would also need to be added in the Settings component within the User Preferences module in order to store the user’s screen time preferences.

Impacted Directories and Files

Business Layer : Player Core Module & PVR Module
The Player Core Module and PVR Module are impacted by the addition of the lockout feature, which would prevent users from accessing media playback if their screen time limit has been exceeded.

Interface Layer : Utility Module : test
The testing directory would be impacted by the addition of new tests to support the additional functionality added by the Screen Time Monitoring Component.

Interface Layer : Utility Module : XBDateTime.cpp
This file would have to support calls to reset the screen time each day.

Interface Layer : Application Module : Interfaces : AnnouncementManager.cpp
This file would require an addition to support the announcements that are triggered when a screen time event occurs. This includes reaching the screen time limit, or reaching any of the thresholds.

Presentation Layer : GUI Elements : dialogs : CGUIDialogKaiToast.cpp
This file mimics push notifications throughout the application. Support would need to be added for the notifications that are triggered when the user reaches certain thresholds of screentime.

Presentation Layer : User Preferences : settings
The settings and profiles must be updated to support a screen time limit. Support would also need to be added in order to have a password that would allow a user to bypass the screen time limit.

Testing the Enhancement

To test the impact of the interactions between the proposed Screen Time Monitoring Component and the existing system, various techniques will be used. A major point that we would like to avoid is having the Screen Time Monitoring Component negatively affect the performance of Kodi. To monitor and prevent this from occurring, several performance tests will be performed on Kodi before and after a proposed modification to the existing component.

The performance tests will measure how long it takes for the system to perform its main functionality: browsing, selecting, and playing media. The tests without the monitoring component will be used as a baseline. If the performance of Kodi is over 5% slower when the modification has been made, the performance test should be considered a failure. This would indicate to the development team that there is a deeper problem within the Screen Time Monitoring Component that must be explored before the modification can be deployed.

Potential Risks

The implementation of a Screen Time Monitoring Component through Usage Monitoring with System Events has potential risks to the performance and security of the application. Constant and continuous screen time monitoring can be resource-intensive, and may eat away at some of the resources which are required for the system to function. This is especially true for any game emulation that takes place on Kodi, since games tend to be resource demanding. To decrease the number of resources that may be taken up by system events, it is important to determine a baseline as for how often to send updates to the usage monitoring system. Updates that are too infrequent decrease the accuracy, while updates that are too frequent would consume too many resources.

There is also the risk of malicious users accessing a particular Kodi installation through the remote web server. These malicious users could falsely trigger events which say that the screen time limit has been exceeded, therefore locking a user out of their system, even though they have not hit their screen time limit. Additionally, if malicious users are accessing Kodi through a remote web server, more aspects than just the Screen Time Monitoring Component would be compromised. Therefore, it is important to ensure the web server is secure in order to avoid malicious web attacks.

Use Case 1: Setting Screen Time Limits

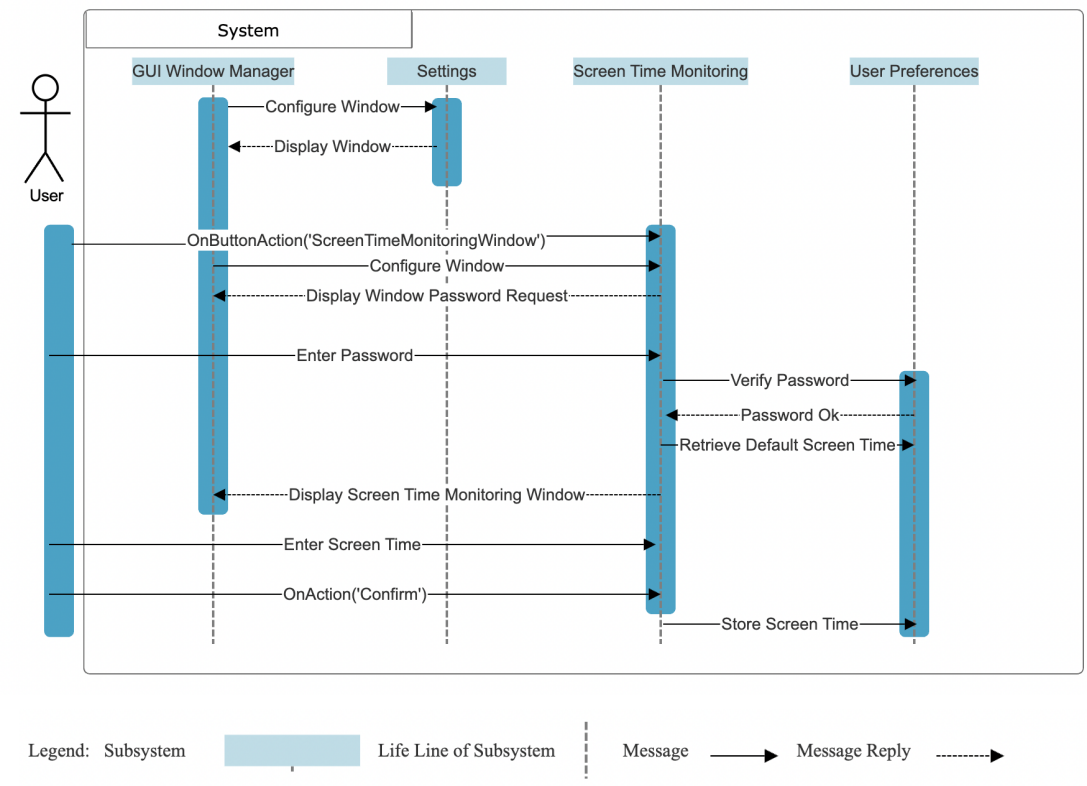

Figure 3: Sequence Diagram for Setting Screen Time Limits

The GUI Windows Manager Module manages the graphical user interface (GUI) elements, including windows, buttons, and lists. The user begins by navigating to the Settings menu from the main window and selects "Screen Time." A password prompt appears to secure access to screen time settings. The entered password is validated through a call to the User Preferences Storage. Upon correct validation, the Window Manager presents the Screen Time Monitoring Window to the user. The system then retrieves the current screen time limit from the User Preferences Storage. If the user has not previously customized the limit, the default of 2 hours is used; otherwise, the system uses the user's last preference. Using a scroll bar or by typing, the user sets their desired screen time limit and confirms their choice, by pressing the confirm button. The Screen Time Monitoring System updates the stored limit in the User Preferences Storage, confirming the successful storage of the new screen time limit data.

Use Case 2: Exceeding the Screen Time Limit

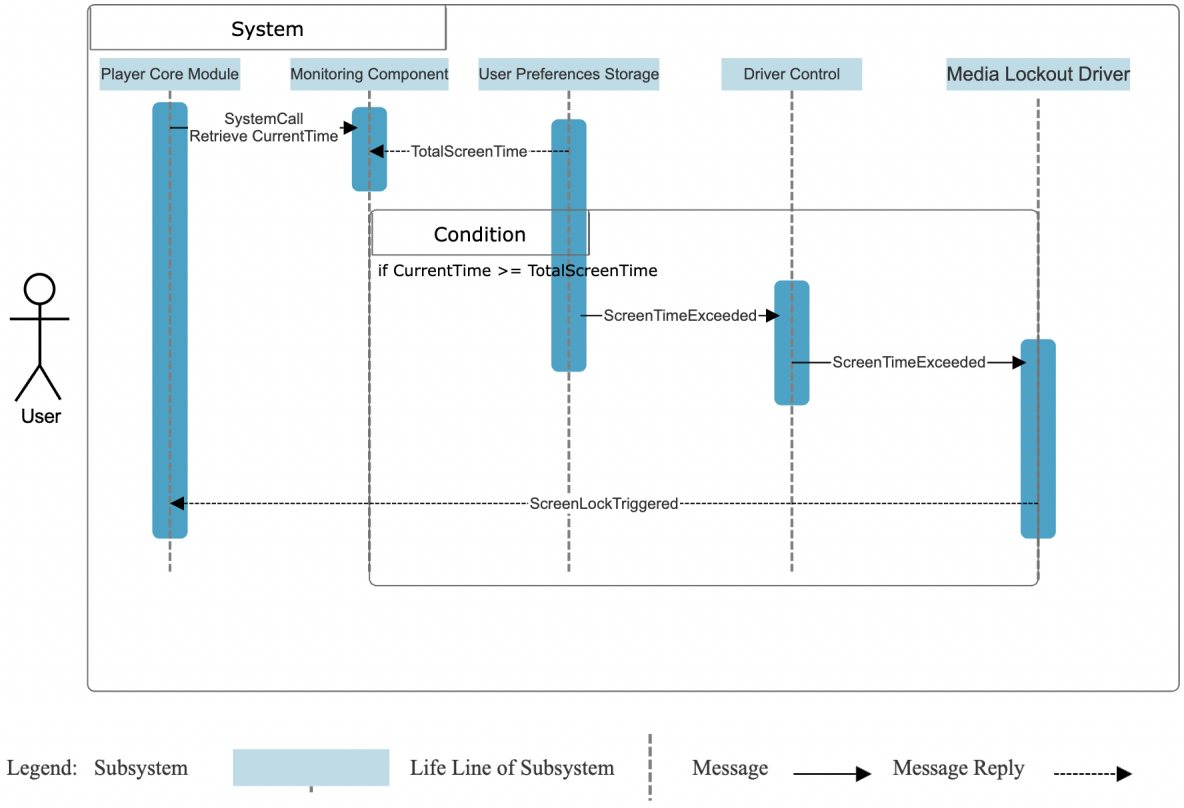

Figure 4: Sequence Diagram for Exceeding the Screen Time Limit

This scenario begins with a call made by System Events to the Monitoring Component of the Screen Time Monitoring Component. The call simply relays how much time has been spent consuming media. The Monitoring Component then compares this to the amount of total screen time that has been set for the user, retrieving it from the User Preferences. Once the Monitoring Component determines that the time spent watching media is equal to or greater than the screen time limit, the Monitoring Component calls the Driver Controller, located in the Driver subsystem of the STMC. The Driver Controller then calls on

the appropriate driver to take action, in this case, the Media Lockout Driver. The Media Lockout Driver communicates directly with the PVR Module and Player Core Module and triggers the locks. Once the locks have been triggered, the user is unable to use the media functionality of Kodi, unless they enter their screen time password.

Conclusion

To summarize, the proposed enhancement to Kodi aims to introduce a Screen Time Monitoring Component to encourage responsible amounts of media consumption. Users and developers, both stakeholders of the system, have their Non-Functional Requirements that are specific to their needs of the system. Between the two parties, the most important Non-Functional Requirements for the enhancement are performance, accuracy, and security. Two approaches for implementation were considered: Usage Monitoring with System Events, and a Predictive Machine Learning Model. After analyzing these approaches, we found implementing this feature via Usage Monitoring with System Events is optimal, due to its privacy, accuracy, and scalability.

To implement the Screen Time Monitoring Component through the selected approach, the PVR Module and Player Core Module would need significant modifications to support the new functionality. A new subsystem called the Screen Time Monitoring Component would need to be created in the Interface Layer. There are some risks with this proposed implementation, the performance could be negatively impacted due to an increase in resources, security could be impacted if malicious users were to access the installation through the remote web server, and the selected implementation may require significant refactoring in order to introduce new features to the component.

Despite these risks, a Screen TIme Monitoring Component would ultimately be a valuable addition to the application. The incorporation of this feature would demonstrate Kodi's care for the wellbeing of its own users by encouraging them to lower their screen time, following recommendations from experts.

Lessons Learned

In our second report, we did not leave enough time to review our final work as a group, and the quality of our paper suffered as a result. For the third and final report, our team started well in advance to ensure we had enough time to structure our ideas and read through to ensure it read coherently. We believe that as a result of our early start, the quality of our third paper is significantly higher than our second.

Another challenge we took on was citing more resources throughout the paper. We did end up devoting a good portion of our time to searching for and reading through relevant research papers. However, this was instrumental in forming our arguments, and was also useful in increasing our own knowledge about certain topics.

References

[1] Team Kodi, “Kodi 21.0 ‘omega’ beta 1: News,” Kodi, https://kodi.tv/article/kodi-omega-beta-1/ (accessed Dec. 3, 2023).

[2] A. Hmidan, D. Seguin, and E. G. Duerden, “Media Screen Time Use and mental health in school aged children during the pandemic,” BioMed Central, https://bmcpsychology.biomedcentral.com/articles/10.1186/s40359-023-01240-0 (accessed Dec. 3, 2023).

[3] A. Goldsteen, G. Ezov, R. Shmelkin, M. Moffie, and A. Farkash, “Anonymizing Machine Learning Models,” arXiv.org, https://arxiv.org/pdf/2007.13086.pdf (accessed Dec. 4, 2023).

[4] S. K. Muppalla, S. Vuppalapati, A. Reddy Pulliahgaru, and H. Sreenivasulu, “Effects of excessive screen time on Child development: An updated review and strategies for Management,” Cureus, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10353947/ (accessed Dec. 5, 2023).

[5] X. Wang, Y. Li, and H. Fan, “The associations between screen time-based sedentary behavior and depression: A systematic review and meta-analysis - BMC Public Health,” BioMed Central, https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-019-7904-9 (accessed Dec. 5, 2023).

[6] R. Kazman, L. Bass, G. Abowd and M. Webb, "SAAM: a method for analyzing the properties of software architectures," Proceedings of 16th International Conference on Software Engineering, Sorrento, Italy, 1994, pp. 81-90, doi: 10.1109/ICSE.1994.296768 (accessed Dec. 5, 2023).