From Java to Flutter: Evaluating Modern Frameworks in Mobile App Development with Semantic Editor

Balachandran Nair Sumadevi Sarada

November 18, 2024

From Java to Flutter: Evaluating Modern Frameworks in Mobile App Development with Semantic Editor

SARADA BALACHANDRAN NAIR SUMADEVI

Decentralized Bigdata Team, *RIKEN*, *Tokyo*

Abstract —**This study investigates the transition from Java to Flutter in mobile app development, using the Semantic Editor application as a case study. It evaluates the benefits of adopting Flutter over traditional Java methods in the fast-evolving mobile technology landscape. The research starts with an overview of Java's historical significance in app development and introduces Flutter as a contemporary framework designed to tackle issues like platform fragmentation and code redundancy. Versions of the Semantic Editor were developed in both Java and Flutter to compare aspects such as development efficiency, user interface quality, and overall performance. The findings demonstrate that Flutter significantly enhances development speed, interface adaptability, and user experience when compared to Java. Additionally, a theoretical comparison with React Native was performed, ultimately favoring Flutter for its superior performance and UI capabilities. This study offers valuable insights for development teams considering a switch to Flutter, providing guidance for strategic decision-making in future mobile app projects. The results suggest that Flutter's cross-platform approach and advanced features make it an attractive choice for modern mobile app development, especially for applications demanding complex user interfaces and real-time collaboration functionalities.**

Index Terms— **Java, Flutter, React Native, Cross-Platform Development, Mobile App Development, User Interface, Performance, Code Reusability.**

I. INTRODUCTION

In recent years, the rapid growth of mobile technologies has significantly increased the demand for mobile applications. This surge has necessitated crucial decisions regarding the choice of technology stacks used in application development, particularly for the Android operating system. Historically, Java has been the primary language for Android development since the OS's inception in 2008, being widely recognized for its robustness and versatility. Java's features such as object-oriented structure, platform independence, automatic garbage collection, type safety, and multithreading support have established it as a foundational language in mobile development.

However, despite Java's capabilities and its pivotal role in the evolution of mobile applications, it presents several limitations such as substantial memory consumption, performance overhead, longer "cold start" times, and limited control over memory and hardware. These constraints have propelled developers to seek more efficient and flexible alternatives.

Enter Flutter, introduced by Google in 2016, which has quickly garnered attention in the mobile development realm. Unlike traditional frameworks that relied on web views or native components, Flutter utilizes a unique approach by employing a rendering engine to generate custom interfaces directly. This method ensures a highly native-like user experience across different platforms. Flutter's advantages include cross-platform development capabilities, a rich set of customizable widgets, efficient rendering engines, and the innovative "Stateful Hot Reload" feature, which accelerates development cycles by allowing developers to instantly see changes in the app.

Transitioning from Java to Flutter

Motivation and Background

The shift from Java to Flutter is not merely a change of tools but a strategic response to evolving development needs. This transition is motivated by several factors:

Enhanced Development Speed: Flutter's hot reload feature significantly shortens the development loop by allowing immediate visual feedback for changes, enhancing developer productivity.

Cross-Platform Efficiency: With Flutter, developers can maintain a single codebase for both Android and iOS platforms, which simplifies the development process and reduces time and resources.

Expressive and Dynamic UIs: Flutter's widget-based architecture allows the creation of highly responsive and aesthetically pleasing user interfaces, which can be more cumbersome to achieve with Java.

Growing Community and Ecosystem: Flutter's increasing popularity has fostered a vibrant community and a growing ecosystem of tools and libraries, providing robust support for developers.

Resource Optimization: By utilizing a single codebase, Flutter reduces the need for platform-specific teams, cutting down on the overhead associated with managing separate development streams for Android and iOS.

II. BACKGROUND

The development of mobile applications has undergone significant transformations over the past few decades, driven largely by advances in programming languages and development frameworks. Java, introduced by Sun Microsystems in 1995, has been a staple in the development community, not least because of its adoption as the official language for Android development when Google launched the operating system in 2008. Java's architecture, which emphasizes security, portability, and high performance, made it an ideal choice for the burgeoning mobile app market (Oracle, "Java Language and Virtual Machine Specifications," Oracle, 2021).

Despite its strengths, Java has faced criticism for its verbosity, memory consumption, and sometimes sluggish performance on less capable devices, which are critical considerations in mobile environments. These limitations have prompted the exploration of more versatile and efficient technologies (Lindholm, T., Yellin, F., & Walrath, K., "The Java Virtual Machine Specification," Addison-Wesley, 1999).

In response to these evolving needs, Google introduced Flutter in 2017, initially as an open-source mobile application development framework. Flutter allows developers to build high-performance, natively compiled applications for mobile, web, and desktop from a single codebase. Unlike traditional frameworks that require separate views for each platform, Flutter employs a unique approach by using a proprietary rendering engine to draw UIs, thus offering excellent performance and visual consistency across platforms (Google, "Introducing Flutter," Google, 2017).

The use of Dart, a language optimized for client-side development, as Flutter's programming language further simplifies the development process, providing features like just-in-time compilation and hot reload that significantly enhance developer productivity (Google, "Dart Language Overview," Google, 2021).

As mobile technology continues to advance, the shift from Java to more dynamic frameworks like Flutter represents not just a technological evolution but also a response to the global market's demand for more responsive and visually appealing applications.

Why Flutter not React Native ?

In this comparative analysis [12], a fair and comprehensive evaluation was conducted between Flutter and React Native across several performance metrics, including CPU usage, memory usage, and janky frames, under different test conditions. Overall, Flutter demonstrated superior CPU performance with an average CPU usage of 43.42%, compared to React Native's higher average of 52.92%. Although Flutter exhibited higher initial spikes in CPU usage, it stabilized more efficiently over time. In terms of memory usage, React Native had a slight edge with an average usage of 7.85%, while Flutter's average was marginally higher at 8.06%. However, Flutter excelled in maintaining smoother performance with fewer janky frames, averaging 2.6 compared to React Native's 3.6.

When evaluating the performance of accessing hardware functionality through the camera implementation, the results indicated mixed outcomes. React Native showed better CPU efficiency with an average usage of 49.06%, significantly lower than Flutter's 81.63%. Conversely, Flutter outperformed React Native in memory usage, averaging 5.62% compared to React Native's 7.76%, and in reducing janky frames, with Flutter averaging just 0.8 compared to React Native's substantial 52.6.

Additionally, in evaluating the application's search feature, Flutter consistently demonstrated superior performance across all metrics. The average CPU usage for Flutter was 11.34%, markedly lower than React Native's 36.21%. Similarly, Flutter had better memory efficiency with an average usage of 4.91%, compared to React Native's 5.77%. Flutter also maintained smoother performance with fewer janky frames, averaging 2, whereas React Native averaged 6.6 janky frames.

This fair comparison has highlighted Flutter's consistent superiority in CPU efficiency and reducing janky frames across various conditions, although React Native occasionally exhibited slightly better memory usage. These insights have been instrumental in choosing Flutter over React Native for its overall performance advantages.

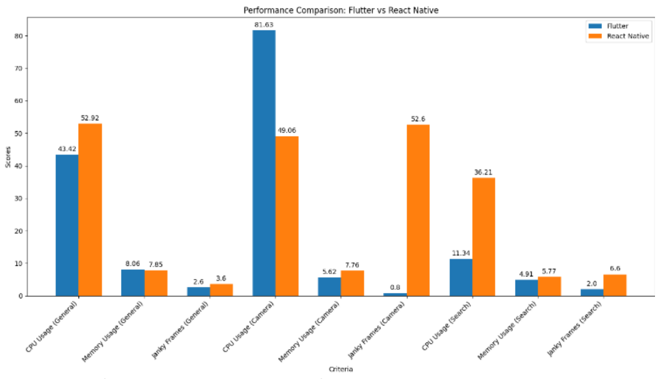


Fig 1. Performance Comparison of Flutter and React Native Across Multiple Metrics

III. Problem Definition

This section revisits pertinent previous studies to set the stage for this research, which aims to critically evaluate and compare the user interface (UI) quality and performance of the Semantic Editor developed in Java and Flutter. This comparison aims to determine which development framework better suits the needs of modern application design, particularly in terms of efficiency and user experience.

i. Purpose

The Semantic Editor has been implemented using both Java and Flutter to explore which platform offers superior support for creating intuitive and effective user interfaces. Despite the foundational role of these technologies in application development, a systematic comparison of their capabilities in actual use scenarios, particularly in UI responsiveness and overall performance, has been lacking.

To address this gap, a series of experiments were conducted focusing on analyzing and comparing the UI design capabilities and performance metrics of the Semantic Editor across these two platforms. The experiments aimed to assess aspects such as the ease of use, responsiveness of the interface, and the smoothness of interactions within collaborative environments.

Expected outcomes from this comparative analysis include detailed insights into the advantages and limitations of Java and Flutter in application development. Additionally, a theoretical evaluation of React Native was conducted to broaden the perspective. Based on this evaluation, Flutter was chosen over React Native for its superior performance and UI design capabilities. These findings will help identify which platform better meets the criteria for effective UI design and performance and will guide future enhancements to the Semantic Editor. Ultimately, this research contributes to broader discussions on choosing appropriate technologies for developing high-performance, user-friendly applications.

ii. Hypotheses

The general hypothesis of this study is:

The Semantic Editor developed in Flutter will overall outperform the Java version in terms of development efficiency, user interface quality, and end-user satisfaction, demonstrating the advantages of modern cross-platform frameworks in creating superior application experiences.

To comprehensively test this general hypothesis, we have delineated it into the following three specific hypotheses, each addressing distinct aspects of the application's performance and user interaction, as outlined earlier:

Hypothesis 1: Flutter will exhibit faster development cycles and reduced time-to-market for the Semantic Editor compared to Java, owing to its single codebase approach and hot reload capabilities.

Hypothesis 2: The Semantic Editor developed in Flutter will provide a more responsive and visually appealing user interface than its Java counterpart, due to Flutter's rich set of customizable widgets and inherent high-performance rendering engine.

Hypothesis 3: Users will experience greater satisfaction and higher efficiency when collaborating using the Semantic Editor developed in Flutter compared to Java, primarily because of Flutter's streamlined UI components and better performance consistency across platforms.

These hypotheses are crafted to explore and validate the various benefits of using Flutter over Java in different phases of software development and user engagement, providing a structured framework for our experimental evaluation.

IV. Introduction to the System

i. Semantic Editor Overview

Semantic Editor [11] is an innovative application specifically designed to facilitate the practice of the Diagrammatic Semantic Authoring (DSA) standard. This tool is pivotal for those engaged in the creation and manipulation of semantic graphs through collaborative efforts.

Core Features include,

1. Graph-Based Editing System:

The Semantic Editor offers a sophisticated graph-based editing system, enabling users to interactively create, move, delete, and modify nodes and links within a graphical interface.

Each link is annotated within the DSA framework, signifying the semantic relationships between nodes, which are crucial for interpreting the underlying structure and meaning of the data.

2. Collaborative Editing Capabilities:

The application is engineered to support real-time collaborative editing, allowing multiple users at different locations to work concurrently on the same document.

It achieves this through a cloud-based architecture that integrates a graphical synchronization system, ensuring that all changes are reflected instantly across all users' views.

3. Security and Data Sharing

Personal Life Repository (PLR)[11]: Semantic Editor incorporates the Personal Life Repository (PLR) to enhance communication security. PLR is a decentralized, secure, low-cost, and scalable Personal Data Store (PDS) that facilitates the social sharing and utilization of personal and other data based on the intentions of the data subjects.

End-to-End Encryption: Users, including individuals and organizations, can securely share their data directly, bypassing any intermediaries, thanks to end-to-end encryption.

4. Cost-Efficiency: The operational cost for both application/service providers and end-users is minimized as the platform supports the use of common cloud storage solutions like Google Drive and OneDrive for storing shared data.

ii. Functionalities

RDF-Graph Composition: Users have the capability to dynamically interact with the RDF graph by creating, moving, deleting, and modifying nodes and links. Nodes are versatile, capable of containing text such as simple sentences or phrases, which adds another layer of data representation.

Real-Time Collaboration: The editor is designed to support seamless, real-time collaboration among multiple users, facilitated by robust data synchronization through public clouds. This feature is essential for teams that operate in fast-paced environments where immediate feedback and iterative changes are common.

Semantic Relationships

The tool utilizes an ontology of discourse and other relationships to comprehensively address and manage the interactions among nodes. This ontology is crucial as it defines the basic semantic and pragmatic relations, which are fundamental for maintaining the integrity and utility of the semantic graphs created within the editor.

iii. Technical Architecture

The technical architecture of semantic editor in both Java and flutter is discussed in this paper.

1. Java Architecture

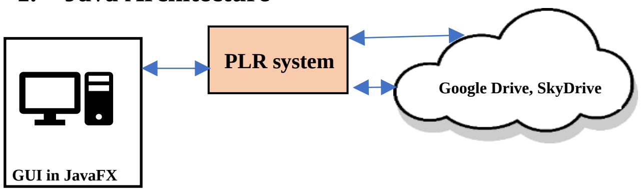


Fig 2: Java Architecture of Semantic Editor

The system architecture revolves around the Semantic Editor application, which is designed using JavaFX, a robust platform for creating desktop applications with sophisticated graphical user interfaces (GUIs). JavaFX serves as the backbone of the Semantic Editor, enabling developers to craft an intuitive and visually appealing interface for users to interact with. Through its seamless integration with the PLR library, JavaFX ensures that the Semantic Editor's functionalities align with the PLR standard, facilitating compatibility and interoperability with other PLR-compliant applications. This integration allows users to seamlessly create, edit, and manage public land records data within the Semantic Editor's user-friendly interface, leveraging JavaFX's rich set of UI controls and layout options.

Furthermore, JavaFX's integration with the PLR library extends to crucial aspects such as user identification, security management, and encryption, ensuring a secure and seamless user experience within the PLR ecosystem. Overall, JavaFX plays a pivotal role in empowering users to interact with public land records data efficiently and securely through the Semantic Editor application.

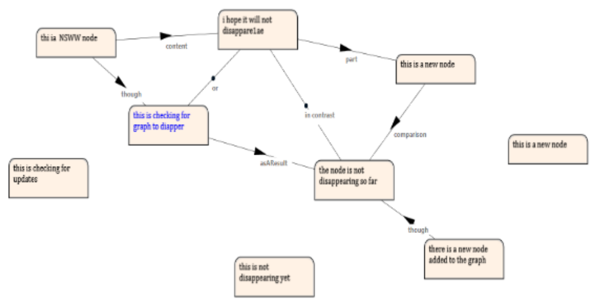

thi ia NSWW node
content
i hope it will not disappare1ae
part
this is a new node
though
or
in contrast
this is checking for graph to diapper
comparison
this is a new node
this is checking for updates
asAResult
the node is not disappearing so far
though
there is a new node added to the graph
this is not disappearing yet

Fig 3: Java UI of Semantic Editor

In our current system architecture, which relies heavily on JavaFX for the development of the Semantic Editor's graphical user interface (GUI), we've encountered notable drawbacks that have significantly impacted user experience and system performance.

- Lack of robust optimization options within JavaFX: JavaFX's limited optimization options make it difficult to fine-tune the application's performance for better responsiveness and efficiency. The UI components were very limited compared to flutter which made it difficult to implement the requirements.
- Continued performance bottlenecks and instability despite optimization efforts: Despite attempts to optimize the application's performance, performance bottlenecks persist, leading to unstable behavior and inconsistent user experience. When multiple threads were running in the backend the application freeze most of the time which affected the usability of the application.
- Slower performance compared to native GUI frameworks: JavaFX's performance is slower when compared to native GUI frameworks, affecting the overall responsiveness and speed of the application. Each time the screen refresh was called when a new node is added which caused freezing for longer time when the number of nodes were more than 10.
- Negative impact on user satisfaction and application adoption: The performance issues and limitations negatively impact user satisfaction and may hinder the adoption of the application by users, leading to reduced productivity and efficiency.

1. **Flutter Architecture**

Flutter as the primary technology for developing the graphical user interface (GUI) of our Semantic Editor application. This strategic decision was made in response to the significant performance challenges and limitations experienced with JavaFX, including frequent application hang-ups, sluggish behavior, and optimization difficulties. By leveraging Flutter's fast rendering capabilities, optimized performance, and ability to compile down to native code, we have effectively addressed these issues.

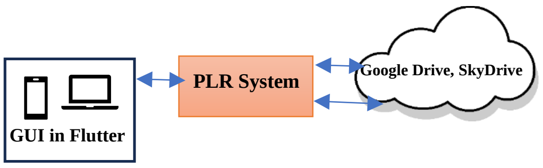
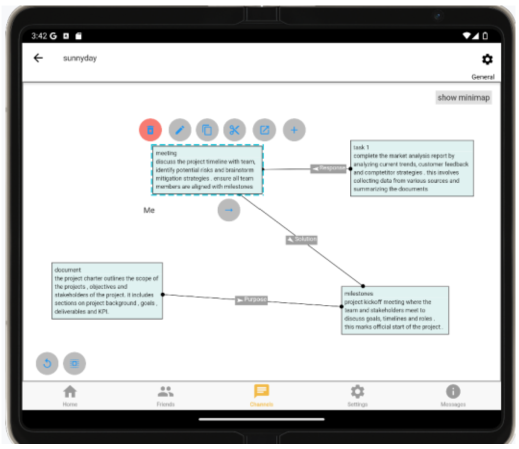


Fig 4: Flutter Architecture of Semantic Editor

The transition to Flutter has resulted in a marked improvement in performance, with smoother and more responsive user interactions, thereby mitigating the performance bottlenecks and instability that persisted with JavaFX. Additionally, Flutter's robust optimization options have empowered our development team to fine-tune the application's performance effectively. This successful transition not only ensures a seamless and efficient user experience but also aligns with our commitment to delivering high-quality software within our system architecture.



meeting
discuss the project timeline with team, identify potential risks and brainstorm mitigation strategies . ensure all team members are aligned with milestones

task 1
complete the market analysis report by analyzing current trends, customer feedback and competitor strategies . this involves collecting data from various sources and summarizing the documents

document
the project charter outlines the scope of the projects , objectives and stakeholders of the project. it includes sections on project background , goals , deliverables and KPI.

milestones
project kickoff meeting where the team and stakeholders meet to discuss goals, timelines and roles . this marks official start of the project .

Fig 5: Flutter UI of Semantic Editor

Moreover, the adoption of Flutter has not only resolved performance issues but has also significantly improved the graphical user interface (GUI) compared to traditional applications, providing users with a more visually appealing and intuitive experience.
Indeed, the transition to Flutter has brought about significant enhancements to the user interface (UI), offering a splendid view across various devices, including mobile and desktop platforms. Flutter's flexibility allows for seamless adaptation to different

screen sizes and resolutions, ensuring a consistent and visually appealing experience across devices. Additionally, Flutter's extensive set of features, such as the addition of a minimap, further enhances user experience by providing additional navigation options and improving overall usability. These advancements underscore our commitment to delivering a top-notch user interface that exceeds expectations and enhances user satisfaction.

IV. FUTURE WORK

As part of our future work, we are excited to explore the integration of generative AI technologies into the Semantic Editor application built on Flutter. By harnessing the power of generative AI, we aim to revolutionize content creation within the application. This will involve implementing AI-driven tools that can assist users in generating and customizing content such as textual descriptions, graphical elements, and visualizations. These AI-powered features will not only streamline the content creation process but also unlock new possibilities for creativity and innovation.

Furthermore, we plan to leverage generative AI to enhance the user experience in novel ways. For example, we envision incorporating AI-generated suggestions and recommendations to assist users in their tasks within the Semantic Editor. This could include intelligent prompts for optimizing document layouts, suggesting relevant keywords or tags, or even generating design variations based on user preferences and project requirements.

In addition, we see potential in using generative AI to automate repetitive tasks and accelerate the creation of complex content. By training AI models on existing datasets and user interactions, we can develop smart algorithms capable of autonomously generating content elements tailored to specific user needs. This will not only save time and effort but also enable users to focus on higher-level creative tasks while the AI handles mundane and repetitive aspects of content creation.

Overall, the integration of generative AI technologies into the Semantic Editor application represents an exciting opportunity to push the boundaries of what is possible with Flutter. By combining the power of Flutter's versatile UI capabilities with generative AI's creativity and automation, we aim to deliver a groundbreaking user experience that empowers users to create, customize, and collaborate on content in innovative ways.

V. REFERENCES

[1] Mirghani Hassan, A. (2020). "JAVA and DART programming languages: Conceptual comparison." *Indonesian Journal of Electrical Engineering and Computer Science*, 17(2), 845-849. DOI: 10.11591/ijeecs.v17.i2.pp845-849

[2] Singh, G., & Sagga, S. (2018). "A Review Paper on Developer's Choice: Java or C++." *JETIR*, 5(10).

[3] Martinez, M., & Mateus, B. G. (2022). "Why Did Developers Migrate Android Applications From Java to Kotlin?" *IEEE Transactions on Software Engineering*, 48(11).

[4] Robillard, M. P., & Kutschera, K. (Year). "Lessons Learned in Migrating from Swing to JavaFX."

[5] Sharma, S., Khare, S., Unival, V., & Verma, S. (Year). "Hybrid Development in Flutter and its Widgets." *Proceedings of the International Conference on Cyber Resilience (ICCR)*.

[6] Kurale, R., & Bala, K. (2021). "A Comparative Study of Flutter with other Cross-Platform Mobile Application Development." *International Journal of Computer Science and Information Technologies*, 9(12).

[7] Oracle. (2021). Java Language and Virtual Machine Specifications. Retrieved from https://www.oracle.com/java/technologies/javase/vm-spec.html

[8] Lindholm, T., Yellin, F., & Walrath, K. (1999). The Java Virtual Machine Specification. Addison-Wesley.

[9] Google. (2017). Introducing Flutter. Retrieved from https://flutter.dev/docs/get-started/flutter-for/android-devs

[10] Google. (2021). Dart Language Overview. Retrieved from https://dart.dev/guides/language

[11] Hasida K. Personal life repository as a distributed PDS and its dissemination strategy for healthcare services. In2014 AAAI Spring Symposium Series 2014 Mar 22.

[12] Gustav Tollin,Marcus Lidekrans(2023) React Native vs. Flutter: A performance comparison between cross-platform mobile application development frameworks.
Linköping University | Innovativ programmering HT 2023 |