Proposal for Mobile Platform Integration

April 4th, 2025

Group #8: 500 - Internal Server Error

Qiu Xing (Nathan) Cai - 21qxc@queensuca

Wanting Huang - 20wh18@queensu.ca

Haoxi Yang - 21hy11@queensu.ca

Yuda Hu - 21yh29@queensu.ca

Nicholas Wang - 22nw3@queensu.ca

Amethyst Shen - 21YC121@QueensU.ca

Table of Contents

1.0 Abstract
2.0 Introduction and Motivation
3.0 Current Architecture Overview
    3.1 Overview of Existing System Design
    3.2 Limitations of Current Architecture
4.0 Proposed Enhancement: Mobile Platform Support
    4.1 Feature Description
    4.2 Value and Benefits
    4.3 Interactions with Existing Features
5.0 Use Cases and Sequence Diagrams
6.0 Impact Analysis
7.0 Alternative Architectural Designs
8.0 SAAM Analysis
    8.1 Stakeholder Identification
    8.2 Non-Functional Requirements
    8.3 Evaluation of Alternatives
    8.4 Selected Alternative
9.0 Architectural Modifications
    9.1 High-Level Architectural Changes
    9.2 Low-Level Component Changes
    9.3 Affected Files and Directories
10.0 Risk Analysis and Mitigation
11.0 Testing Strategy
12.0 Conclusion
13.0 References
14.0 Lessons Learned
15.0 Naming Conventions
16.0 Reference

1. Abstract

This report presents an architectural enhancement for the GNUstep software system aimed at providing support for enriched mobile platforms namely iOS and Android. While GNUstep is very solid for desktop development, it has limited support for mobile rendering, touch input, and lifecycle management. To this end, we propose adding mobile-specific backends—using Core Graphics on iOS and SurfaceFlinger on Android—as well as touch event handling inside libs-back and libs-gui. These extensions fit into GNUstep's layered architecture, yet allow cross-platform development.

You can find in that document the high-level and component level changes to be made in order to support the enhancement as well as a sequence diagram of the mobile use case. We apply SAAM (Software Architecture Analysis Method) and it compares two implementations, including one that is directly integrated and one that demonstrates plugin based design. We analyze their implications on various stakeholders, and assess their non-functional requirements, like maintainability and portability. This leads us to the conclusion that we believe the plugin-based approach is the better one in that it allows for greater modularity, and therefore scalability. This improvement expands GNUstep's reach into mobile ecosystems without sacrificing architectural purity.

2. Introduction and Motivation

GNUstep is a free and open-source implementation of Apple's Cocoa (formerly OpenStep) framework, designed primarily for desktop platforms such as Linux, Windows, and macOS. While it offers a mature set of APIs for building GUI applications using Objective-C, it is fundamentally limited to desktop environments. Its architecture assumes traditional windowing systems and relies heavily on mouse-and-keyboard input models. As a result, it lacks native support for mobile operating systems such as iOS and Android, including the absence of support for mobile rendering backends or touch-based event handling. These architectural constraints limit GNUstep's applicability in today's mobile-driven software ecosystem.

Introducing support for mobile platforms would significantly expand GNUstep's relevance and utility. Developers could leverage the same Objective-C codebase to create applications for both desktop and mobile devices, improving code reuse and reducing maintenance overhead. For users, this enhancement would enable access to applications across more device types and operating systems. Additionally, enabling mobile support aligns GNUstep with modern development expectations, where cross-platform compatibility is a standard requirement.

This report explores a proposed architectural enhancement to bring mobile platform support to GNUstep. We begin by analyzing the current system design and its limitations, then describe the proposed enhancement in detail. The report includes

use cases and sequence diagrams to illustrate mobile interaction flows, an impact analysis covering quality attributes such as maintainability and performance, and two alternative architectural design options to support the enhancement. We conclude with a comparative SAAM evaluation, risk analysis, and testing strategies.

3.0 Current Architecture Overview

3.1 Overview of Existing System Design

GNUstep is organized into five key components:

libs-base: Provides fundamental system services, such as memory management, file I/O, and networking.

libs-corebase: Acts as a compatibility layer between GNUstep and Apple's Core Foundation framework.

libs-gui: Supplies user interface elements, event handling, and application-level input control.

libs-back: Handles rendering tasks and bridges UI elements with platform-specific graphics systems.

Gorm: A visual GUI builder that generates UI files for use with libs-gui.

These components work together in a layered model. At the bottom, libs-base and libs-corebase form the foundation. Above them, libs-gui manages the application logic and interface. libs-back renders UI components, while Gorm sits at the top, used during development to build interface layouts. Most user-facing applications built with GNUstep depend heavily on libs-gui and libs-back.

This design enables applications to run across Unix-like systems and Windows with minimal changes. Each layer exposes a clear interface to the one above, and hides platform-specific complexity from the application developer.

3.2 Limitations of Current Architecture

While the current system is portable across desktop platforms, it does not support mobile environments like iOS or Android. Several parts of the architecture assume the presence of desktop-style windowing systems such as X11 or Win32. libs-back is tightly coupled to these backends and lacks support for mobile rendering systems like Core Graphic (iOS) or SurfaceFlinger and WindowManager (Android).

Event handling also assumes a mouse-and-keyboard interaction model. Touch input, gesture recognition, and other mobile-specific controls are not built into libs-gui. Additionally, screen layout, resolution handling, and power management features needed for mobile platforms are absent.

Because of these design assumptions, porting GNUstep to mobile platforms is not straightforward. Subsystems like libs-back would require new backends, and libs-gui would need updated controls and event models. Despite this, the modular design means some adaptation is possible without replacing the whole system.

4.0 Proposed Enhancement: Mobile Platform Support

4.1 Feature Description

We propose adding mobile platform support to GNUstep by extending its rendering and input handling capabilities to support iOS and Android. This enhancement would involve creating new backends in the libs-back subsystem that interface with mobile rendering APIs—Core Graphics for iOS and SurfaceFlinger and WindowManager for Android. To support mobile input, libs-gui would be updated to handle touch events, gestures, and mobile system notifications. These features would coexist with the existing desktop input model but would be activated conditionally based on the platform. In addition, the application lifecycle model would be adapted to match mobile standards. This means adding support for lifecycle events such as “pause,” “resume,” and “terminate,” which are required for integration with mobile OSes. Finally, adjustments to the build system would be required to support compilation into mobile app packages—IPA for iOS and APK for Android. These changes would be localized and optional so that developers can continue to build desktop-only applications without modification.

4.2 Value and Benefits

This feature would extend GNUstep’s reach to modern mobile platforms, making it more relevant for current application development. Developers would be able to reuse existing Objective-C codebases for mobile platforms and build cross-platform applications that target desktop and mobile from a unified framework, all while maintaining compatibility with Apple’s Cocoa conventions.

GNUstep would also become more appealing to new contributors and projects seeking to support both desktop and mobile environments. It would provide a great open-source alternative to commercial cross-platform frameworks like Flutter or Xamarin. For existing GNUstep users, mobile support would not interfere with desktop projects, as the new code paths would only activate when targeting mobile platforms.

4.3 Interactions with Existing Features

The proposed mobile support would mainly affect libs-back and libs-gui.

- **libs-back**: New rendering backends would be added alongside the existing ones. For example, a Core Graphic backend for iOS and a SurfaceFlinger and WindowManager Android backend would be implemented using native drawing APIs. These would plug into the same internal interfaces used by current backends like X11 and Windows GDI.

- **libs-gui**: Touch and gesture input would be handled through conditional logic based on platform. Existing desktop event code would remain untouched. UI components might be extended to support mobile layout rules (e.g., scaling, dynamic resizing).

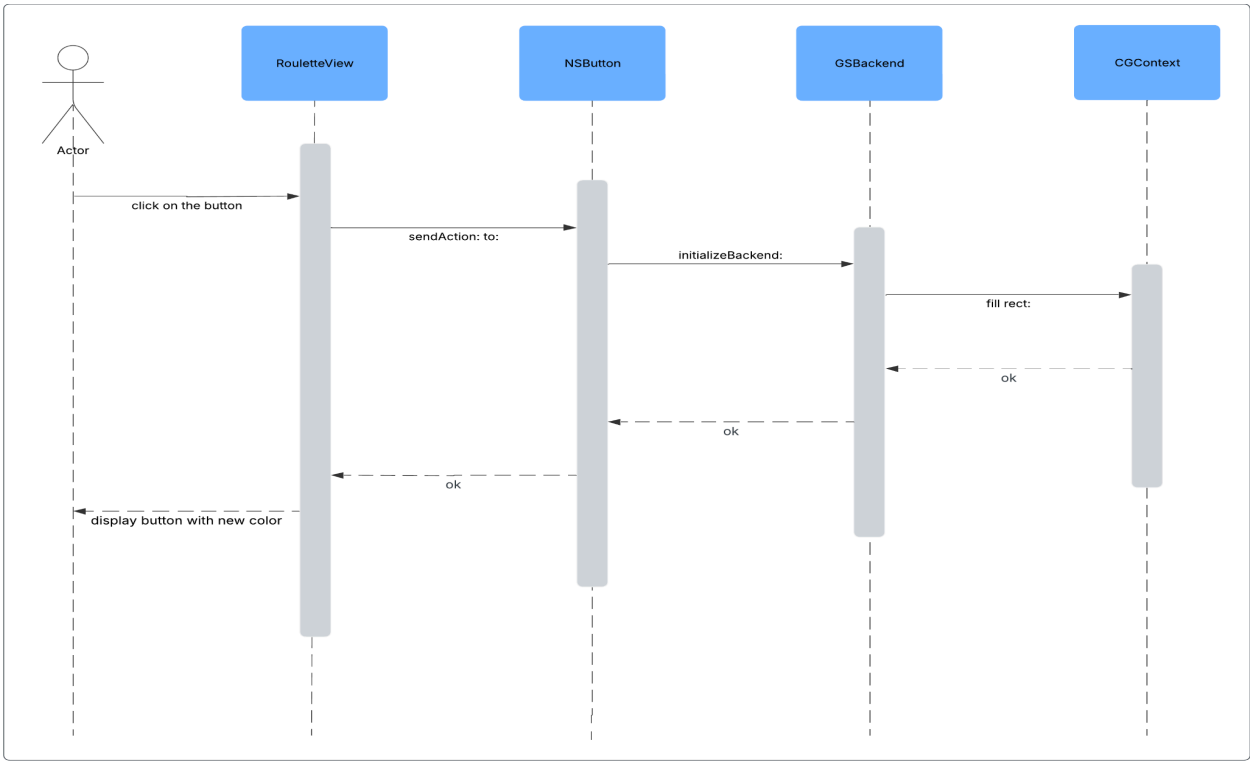
- **libs-base** and **libs-corebase**: Minor adjustments may be needed to support mobile file paths, resource access, or power management, but no major architectural changes are expected here.

- **Gorm**: In its current form, Gorm focuses on desktop UI layout. It could optionally be extended in the future to offer mobile-friendly UI templates, but this is not part of the core enhancement.

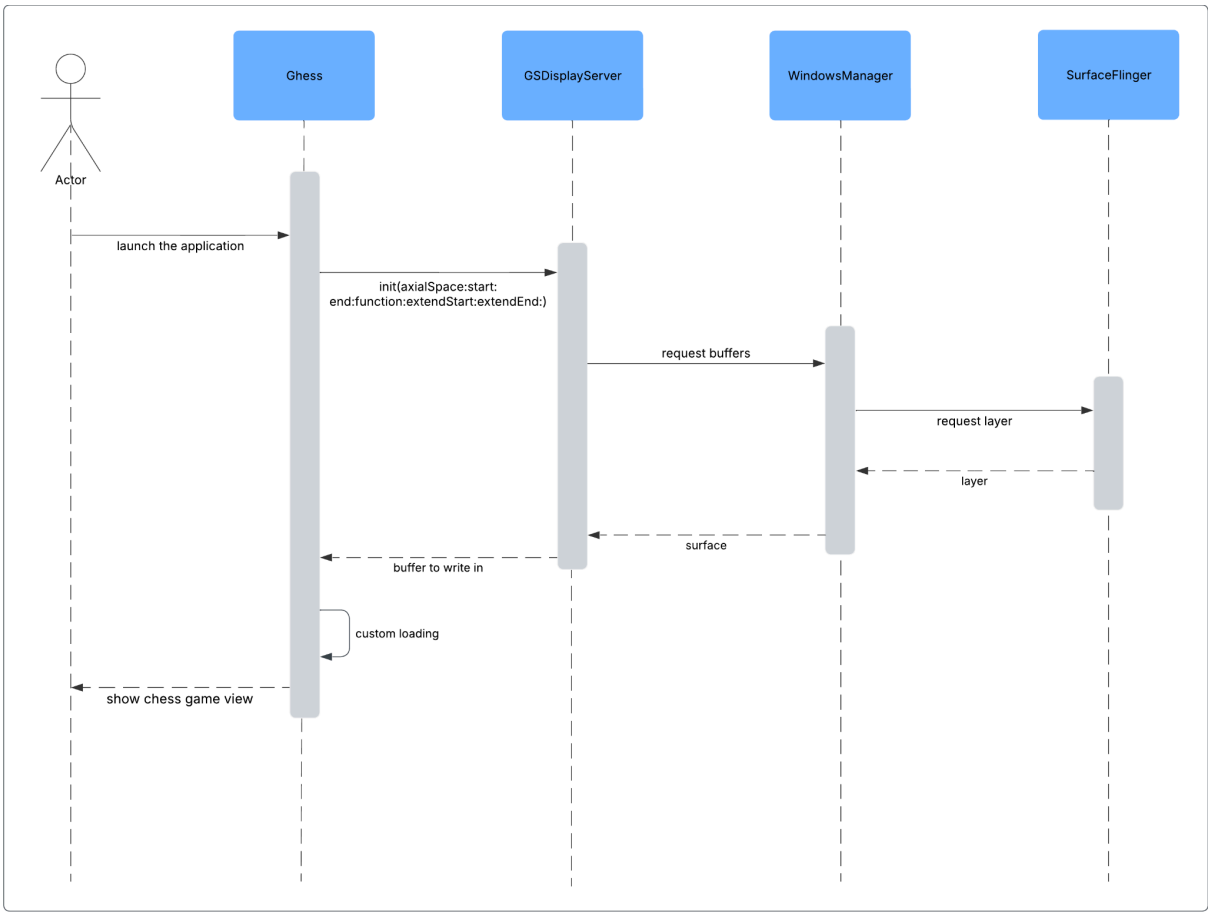
This enhancement respects the existing layered architecture. It introduces platform-specific support in a way that keeps mobile features modular and well-isolated from current functionality.

5. Use Cases & Sequence Diagrams

Use case 1: An app built with GNUstep has an easter egg page "Roulette" in which there is a button that changes its color every time it is touched.



Use case 2: “Ghess” is a chess game built with GNUstep for Android. To achieve better graphic performance, the developer decides to obtain the window buffer from the system and directly write into it at launch time.



6. Impact Analysis

The maintainability of such massive features will be significant. GNUStep will need to support two more operating systems, Android, and iOS. Its maintainers will need to support compatibility between mobile operating systems as well as support the existing operating systems of Windows, Mac, and Linux. Supporting two entirely new and different operating systems may cause complications about ensuring existing and new features remain supported between all five.

Our new features provide many avenues for evolvability. The maintainers of GNUStep will need to support the latest versions of iOS and Android and whatever new features and new APIs are available. For example, with the new iPhone 14 series and newer, Apple created new APIs for app developers to use the dynamic island, transitioning away from the iconic notch. That is one example where GNUStep would need to step up its efforts in maintainability and supporting the newest features available.

GNUStep will also face significantly more rigorous testing than the current ones when incorporating two new OS. GNUStep will need significantly more work on ensuring consistent behaviour and compatibility when transitioning from supporting three to five OS.

One way GNUStep can test our new features is to make an application that uses tools and APIs offered by GNUStep and making sure that the application's test suite works in not just three, but five OS.

One impact on performance is that GNUStep would need to ensure that its apps don't consume too much power, as unlike laptop and desktops, they don't have as large of a battery or a constant source of power.

GNUStep offers a unique advantage compared to other cross-platform mobile frameworks like React Native and Flutter. GNUStep is written Objective-C which is the native programming language of not just Mac, but also iOS, this can simplify the implementation of this feature at least for iOS and increased performance as there is less overhead compared to for example React Native which has overhead between JavaScript and the native OS.

7. Alternative Architectures

7.0 Alternative Architectural Designs

To realize mobile platform support in GNUstep, we have considered two distinct architectural approaches. ***Option A*** involves integrating the new functionality directly into GNUstep's existing GUI and backend layers, extending the current layered architecture to handle mobile-specific requirements. ***Option B***, in contrast, proposes a more modular design: a plug-in based backend that can be added to GNUstep to support mobile rendering and input, isolating mobile-specific code outside of the core libs-back. Each option leverages a different architecture style (direct layered integration vs. plugin-oriented extension), with its own benefits, risks, and alignment with GNUstep's design principles. Below, we describe each option in detail and discuss their trade-offs and how they fit with GNUstep's established layered, Model-View-Controller (MVC) design.

Option A: Direct Integration into Core Libraries

In this option, mobile platform support is added directly into GNUstep's core libraries by modifying the existing ***libs-back*** and ***libs-gui*** components. Mobile rendering backends such as Core Graphics for iOS and SurfaceFlinger and WindowManager for Android would be incorporated directly into ***libs-back***, while ***libs-gui*** would be extended to handle mobile input, including touch gestures and multi touch

interactions. This approach leverages GNUstep's existing layered architecture and retains the current flow of GUI-to-backend communication.

The benefit of this approach is architectural continuity. It maintains the familiar layered model and reuses existing code paths, which may reduce the learning curve for contributors and avoid the need for a plugin management system. It also enables tight coupling between the GUI and rendering layers, potentially reducing latency in event processing and simplifying debugging. However, this method introduces increased complexity in the core libraries, which could affect maintainability and make the system harder to evolve. Mobile-specific logic would be intermingled with desktop functionality, increasing the risk of regressions. Additionally, this tightly coupled structure would require frequent updates to the core whenever new mobile platform changes occur. Overall, this approach is suitable for teams seeking a direct, integrated solution with limited initial restructuring, but it trades flexibility for immediacy.

Option B: Plugin-Based Backend Extension

This approach introduces a modular plugin-based system for mobile support. Instead of modifying the core libraries directly, platform-specific rendering and input handling are implemented as external plugins that conform to a well-defined interface provided by ***libs-back***. At runtime or compile-time, the appropriate plugin—such as one for iOS or Android—is loaded to provide the necessary backend functionality. The core libraries remain mostly unchanged, acting as an abstraction layer between GUI components and platform-specific implementations.

This modular design enhances maintainability and scalability. Platform-specific logic is isolated within its respective plugin, which reduces the likelihood of affecting desktop code and simplifies testing. New platforms can be added by developing new plugins without altering the core system. This model aligns with the open-closed principle, allowing the framework to grow without major refactoring. The downside is the initial complexity of designing a robust and flexible plugin interface. Indirect calls between components may also introduce minimal performance overhead. Still, the benefits of clean separation, better testability, and future-proofing outweigh these concerns. Option B is ideal for teams aiming for long-term extensibility and minimal impact on the core architecture.

8. SAAM Analysis

8.1 Stakeholder Identification

The stakeholders in our feature enhancement proposal are broad, including any company or developer seeking to expand into the mobile market and the users who would benefit from cross-platform applications. Supporting mobile platforms would transform GNUstep into a more powerful and versatile framework, extending its utility

beyond traditional desktop systems. This expansion would require substantial architectural changes, and thus involves various parties. The maintainers of GNUstep would need to introduce new subsystems and modify existing ones to support mobile-specific features. Developers of third-party GNUstep libraries might also face the decision of whether to extend their tools for mobile compatibility.

8.2 Non-Functional Requirements

A key non-functional requirement (NFR) for this enhancement is maintainability—it must remain feasible to implement and sustain without compromising the stability or ongoing development of other parts of the framework. For the developers using GNUStep to write applications, another NFR is for there to be a good developer experience writing applications in this framework on mobile platforms. Additional stakeholders include app distribution platforms like the Apple App Store and Google Play Store. Cooperation between GNUstep developers and application publishing platforms would be necessary to ensure compatibility with each ecosystem's requirements, such as API support, app signing, and packaging formats. This introduces NFRs like stability, security, and platform compliance. End users are also crucial stakeholders. With this enhancement, users would have access to more applications built using GNUstep, alongside existing frameworks like Swift, Kotlin, Flutter, and React Native. Accordingly, GNUstep applications must be performant and resource-efficient to meet the constraints of mobile devices, which typically have less processing power, memory, and battery life than desktop systems.

8.3 Evaluation of Alternatives

There are two architectural approaches to implementing mobile support. The first is inspired by React Native, which using React Components, renders native UI components or widget corresponding to the OS. ReactNative also relies on a bridge to handle communication between JavaScript and native code, which comes at a compromise of the performance of a React Native app. This method enables native look and feel on each platform while maintaining performance that is acceptable for most applications.

The second approach is modeled after Flutter, which uses a custom rendering engine. Originally it was Skia but later on in version 3.27 it was replaced by default with a newer render engine Impeller. Flutter uses Dart, which is compiled ahead-of-time into native ARM code. Therefore, it doesn't use the native UI Widgets of the respective mobile OS, Flutter implements it themselves through their rendering engine which means its more performant than React Native. But it comes at the cost of increased complexity for the maintenance of Flutter and larger app sizes since it contains its own rendering engine.

Creating a new rendering engine for GNUstep from scratch would likely exceed the capacity of its current development team. Since most mobile applications—outside of

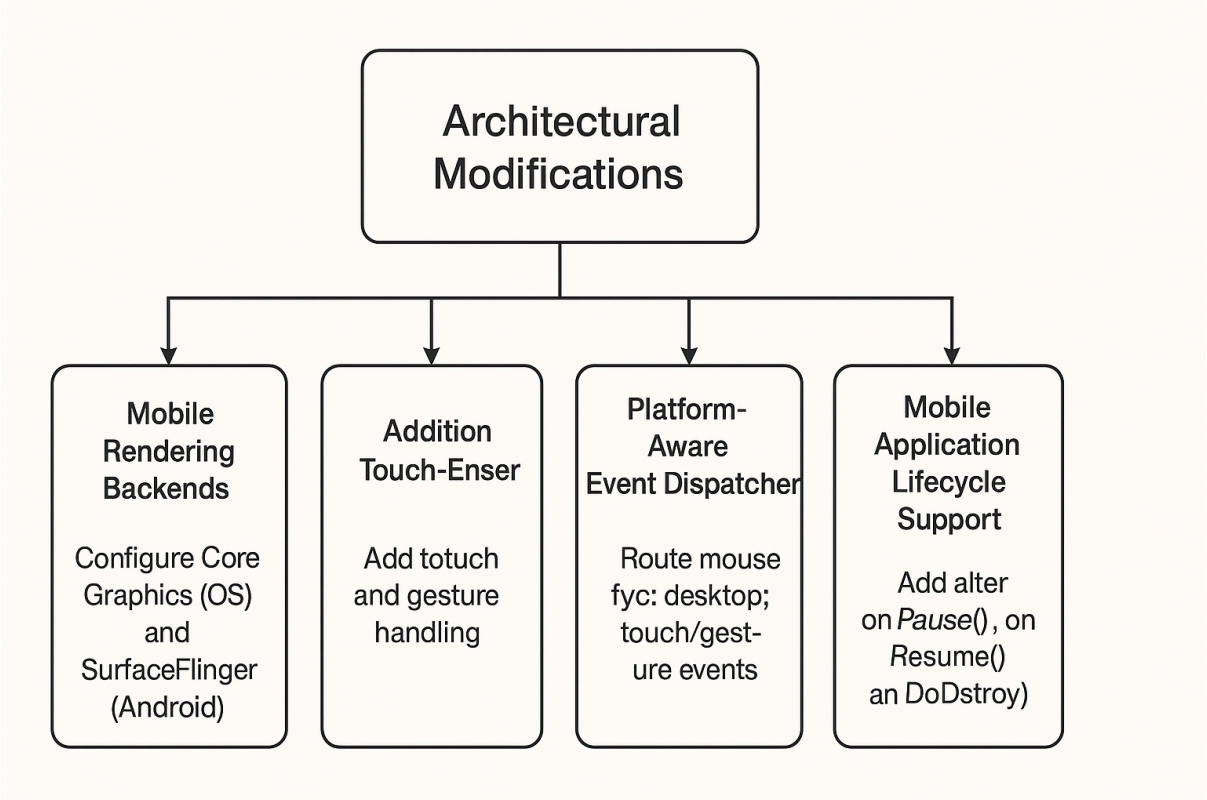
performance-heavy gaming—do not require the highest levels of rendering speed, the performance trade-off of using a native-bridge model like React Native is generally acceptable.

8.4 Selected Alternative

GNUstep's usage of the Cocoa API, mapping UI elements to the native graphics systems (like X11 or Windows APIs) elements makes its architecture more aligned with React Native's approach than Flutter's. Notably, on Apple platforms, GNUstep can directly leverage Objective-C and Swift without needing a language bridge. Given these considerations, we believe the React Native-inspired model is the more realistic and maintainable choice for integrating mobile support into GNUstep. It balances performance, simplicity, and alignment with existing architecture, making it a practical enhancement strategy for the project.

9. Architectural Modifications

9.1 High-level changes



We propose a modular architectural modification that adds rendering and event-handling capabilities for mobile while preserving GNUstep's existing desktop design, thus supporting mobile platforms such as iOS and Android. The changes have kept the layered architecture and maintained platform-specific logic at encapsulated level.

We provide a high-level overview of the mobile rendering backends added on the libs-back, including one for Core Graphics for iOS and another for SurfaceFlinger for

Android platform. We also have to modify libs-gui to create a mobile input layer for gesture-based input and touch interaction. Implement a platform aware event dispatcher to route mouse/keyboard events for desktops and touch/gesture events for mobile. The latter one is adding mobile-specific application lifecycle support, allowing for more mobile OS-friendly onPause(), onResume(), and onDestroy()-like functions. This enhancement detach mobile functionality while still being compatible with legacy features and sets up the system for cross-platform development.

9.2 Low-level changes

Mobile rendering and input support in GNUstep requires some low-level changes, primarily at the libs-back and libs-gui levels. This will involve a new rendering backend for iOS based on the use of Apple's Core Graphics framework (Quartz 2D), providing us with lightweight, high-fidelity 2D rendering. This backend will use various Key Core Graphics classes, including CGContext for drawing, CGLayer for offscreen rendering, and CGPath for vector graphics. It should be implemented via new source files such as CGContext. m and CGLayer. Similarly, AndroidSurface for an Android-specific backend will be introduced. enabling direct communication with SurfaceFlinger and native window buffers. In GSDisplayServer. m and GSBackend. h that the selection logic and rendering interfaces for platforms will be updated and are planned for use on these mobile backends.
On the input side, new modules will be introduced in libs-gui like the GSTouchEventHandler. m and GSGestureRecognizer. m to handle mobile events like tap, swipe, and pinch gestures. Existing files like NSEvent. m and NSApplication. Instead, m will be adapted to differentiate between desktop and mobile input, and will also include lifecycle states; pause, resume, terminate, etc. These changes are meant to be modular and conditional, as well as being able to retain backward compatibility with the desktop environment. Moreover, several small improvements in libs-base and libs-corebase will support file paths and system-level information (like power state, device metrics) that are specific to mobile devices. In a nutshell, these changes at the component level make room for mobile support whilst not interfering with desktop functionality in the existing GNUstep system.

9.3 Affected directories or files

This architectural enhancement involves multiple files and directories across the GNUstep framework (primarily in the libs-back, libs-gui, libs-base and libs-corebase modules). New files are created in libs-back, such as CGContext. m and CGLayer. m will be introduced to implement Core Graphics-based rendering backend for iOS AndroidSurface. m will enable rendering on Android through SurfaceFlinger. Files including already exist, for example GSDisplayServer. m and GSBackend. h will need to be altered to support registering, and dispatching to, these mobile-specific backends. On the UI side, new components will be added to libs-gui, such as GSTouchEventHandler. m and GSGestureRecognizer. m for dealing with touch

and gesture inputs. Additionally, NSEvent. m and NSApplication. m will be amended to differentiate source types and incorporate mobile lifecycle events. Simple, but required changes in libs-base, mainly NSPathUtilities. m, for compatibility with sandboxed mobile file systems. Similarly, SystemInfo. m in libs-corebase will be expanded to surface mobile-relevant system metrics including battery level and screen resolution. These changes reveal the architecture is mobile-capable without losing backward compatibility with the current desktop-focused architecture.

10. Risks and Mitigation Strategies

Extending GNUstep to support mobile platforms introduces several notable risks that must be addressed to ensure long-term stability and usability. One of the primary concerns is compatibility with existing applications. Since GNUstep was originally designed for desktop environments, introducing new mobile-specific backends and input models could potentially interfere with existing desktop workflows. To mitigate this, we propose keeping mobile support modular and conditionally compiled. Mobile backends and gesture-based input handlers will only be included when targeting iOS or Android platforms, ensuring that legacy desktop behavior remains unaffected. Comprehensive regression testing will also be performed to verify that existing applications continue to function as expected.

Another major concern is performance on mobile devices. Unlike desktop systems, mobile platforms are more constrained in terms of processing power, memory, and battery life. Porting desktop-centric rendering logic and event processing without adjustment could result in sluggish UI responsiveness or high resource consumption on mobile devices. To address this, the proposed enhancement leverages native graphics APIs such as Core Graphics on iOS and SurfaceFlinger on Android to take advantage of hardware acceleration. Asynchronous processing techniques and platform-specific optimizations will be used to keep the user interface responsive and efficient.

Finally, expanding GNUstep to support new platforms significantly increases maintenance overhead. Supporting two additional operating systems means that future development must account for greater variation in system APIs, device configurations, and OS updates. This added complexity may challenge maintainers and contributors. To alleviate this, our proposal favors a plugin-based architecture for mobile support. By isolating mobile-specific code into self-contained modules that interface with the existing system, we can reduce coupling and enable contributors to focus on maintaining individual platform backends. In addition, the adoption of automated cross-platform testing and thorough documentation will help streamline development workflows and reduce the long-term maintenance burden.

11. Testing Strategy

- Device compatibility: using virtual machine providers like Android Virtual Device (AVD) or Xcode, the developer can test their software on a variety of different devices with different CPU, GPU, screen size, resolution, refresh rate, aspect ratio, and so on, to ensure that the software works even under uncommon devices or corner cases of operating systems.
- Automated testing: GNUStep could implement a testing library like Jest for React Native which tests the behaviour of functions and components of the application for unit testing.
- Performance profiling: performance counters can be introduced into the GNUstep project to better monitor that what piece of code will be the bottleneck of the whole program so that it can be optimized

12. Conclusion

In this report, we proposed a significant enhancement to GNUstep: support for mobile platforms such as iOS and Android. This feature addresses the growing demand for cross-platform development and aligns GNUstep with modern software ecosystems. By introducing mobile-specific rendering backends and touch-based input handling, our proposal preserves the existing layered architecture while extending GNUstep's capabilities in a modular and scalable way. Between the two design options considered, the plugin-based approach offers greater long-term flexibility and maintainability, allowing new platforms to be added without disrupting the core system. Our SAAM analysis supported this choice by showing improved separation of concerns and reduced risk of regressions. While this enhancement introduces architectural complexity and maintenance overhead, careful design, testing, and documentation can mitigate these challenges.

Overall, mobile platform integration enhances GNUstep's relevance, broadens its user base, and lays a strong foundation for future extensibility without compromising its original design principles.

14.0 Lessons Learned

Working on this enhancement proposal taught our group a lot about the real-world challenges of software architecture. We realized that even though GNUstep has a clean layered structure, adding mobile support isn't just about writing new code, it affects rendering, input handling, app lifecycle, and more. We also learned that the best solution isn't always the most powerful one; sometimes the more practical option is what fits the team's capacity and the system's design.

Throughout the process, we practiced balancing technical ideas with realistic constraints, and learned how to evaluate trade-offs between flexibility, performance, and maintainability. Collaborating as a team also helped us improve our communication and decision-making, and each member contributed unique skills that shaped the final result. Overall, it was a valuable experience that gave us a clearer understanding of how architecture impacts real projects.

15.0 Naming Conventions

NS: Class and method names in GNUstep often begin with the "NS" prefix.
Platform-Specific Files: New source files introduced for mobile support follow descriptive naming patterns, which indicate their platform and function.
Touch/Gesture: Modules related to mobile input handling are named using intuitive prefixes like GSTouch or GSGesture to clearly reflect their functionality.
NFR: Non-Functional Requirements
SAAM (Software Architecture Analysis Method): A structured method used to evaluate architectural design decisions based on their impact on key quality attributes and stakeholder concerns.

16.0 Reference

Partly inspired by: GNUstep. (n.d.). *libs-mobile* [GitHub repository]. GitHub. https://github.com/gnustep/libs-mobile