CISC 322/326 Assignment 3
Proposal for Enhancement Report

April 4th, 2025

Group #30: Big Hero 6

Terence Jiang 21twxj@queensu.ca
Kevin Chen 21kzc2@queensu.ca
Alexander Zhao 21atxz@queensu.ca
Carter Gillam 21cng3@queensu.ca
Kavin Arasu 21kla5@queensu.ca
Daniel Gao 21dg59@queensu.ca

Table of Contents

1. Abstract

This report proposes a native window manager as a possible enhancement for the GNUstep platform. We introduce a new feature for GNUstep designed to improve the applications integration with its graphical environment by developing a dedicated window

manager to provide built in window management capabilities. This report will analyze two different implementations of the proposed feature for GNUstep. We will apply the SAAM analysis method by defining the NFRs of the enhancement, comparing the two implementations concerning the NFRs and stakeholders, and selecting the better approach. This enhancement aims to provide a more seamless and cohesive user experience for GNUstep applications, improving usability and compatibility with modern windowing systems.

2. *Introduction and Overview*

Through previous assignments, we have gained a clear understanding of GNUstep's conceptual and concrete architecture as well as its functionality. This report presents a new feature designed to improve GNUstep's graphical environment: GNUstep Window Integration. This enhancement focuses on providing better window management for GNUstep applications, through a dedicated GNUstep-native window manager. We can see straight from the GNUstep website that GNUstep does not have its own window manager. It relies on whatever window manager is provided by the OS (like Window Maker) on Linux.

This report is divided into four major parts. First, we define two possible implementations of GNUstep Window Integration. The first implementation introduces a new standalone GNUstep window manager. The second implementation enhances GNUstep's existing window handling by extending Appkit's interaction with the underlying windowing system, improving integration while still leveraging external window managers.

In the second part, we perform a SAAM analysis on both implementations, identifying key stakeholders and evaluating how each approach meets the non-functional requirements (NFRs). The SAAM analysis reveals which solution provides the best balance between maintainability, usability, and system compatibility.

The third part presents the chosen implementation in detail, illustrating its functionality through use cases and sequence diagrams. Finally, we discuss potential risks, challenges, and limitations of the enhancement while outlining key lessons learned from the analysis.

Finally, we discuss potential risks, challenges, and limitations of the enhancement while outlining key lessons learned from the analysis. This enhancement aims to make GNUstep more self-contained and improve application usability by ensuring more seamless and predictable window management behaviour.

3. *Proposed Enhancement*

The proposed enhancement introduces a dedicated GNUstep-native window manager to improve application integration with modern graphical environments. GNUstep currently relies on external window managers (e.g. Window Maker), which may not always provide optimal support for GNUstep applications. This enhancement considers two potential implementations.

1. Standalone GNUstep Window Manager: A custom window manager developed specifically for GNUstep, ensuring tight integration with its graphical framework and user interface.

2. Enhanced Window Handling in AppKit: Modifying AppKit to improve interaction with existing window managers, allowing GNUstep applications to function more seamlessly across different operating systems without requiring a dedicated window manager.

Both approaches aim to improve application usability, window consistency, and integration while balancing factors like maintainability, compatibility, and ease of adoption.

a) *Values and Benefits*

The enhancement provides several key benefits like an improved user experience. A native window manager tailored for GNUstep applications ensures more predictable window behaviour, reducing inconsistencies across different platforms. GNUstep applications will have a more cohesive appearance and interaction model, leading to a more seamless desktop experience. Enhancing AppKit's interaction with window managers allows GNUstep applications to adapt more effectively to various desktop environments.

A native window manager also eliminated dependency on external window managers, reducing the risk of compatibility issues. This dedicated solution enables more flexibility in window management behaviour, allowing for features specific to GNUstep's architecture and user needs.

b) *Overview of Changes Required*

To implement the proposed enhancement, several modifications to GNUstep's architecture are necessary.

For option 1, (the development of a standalone window manager) we would need to introduce a dedicated window manager as a separate GNUstep component, interacting directly with GNUstep's graphical environment. We would need to modify the architecture to support standard window management features (eg. window creation, focus management, resizing, stacking order) while maintaining compatibility with existing GNUstep components.

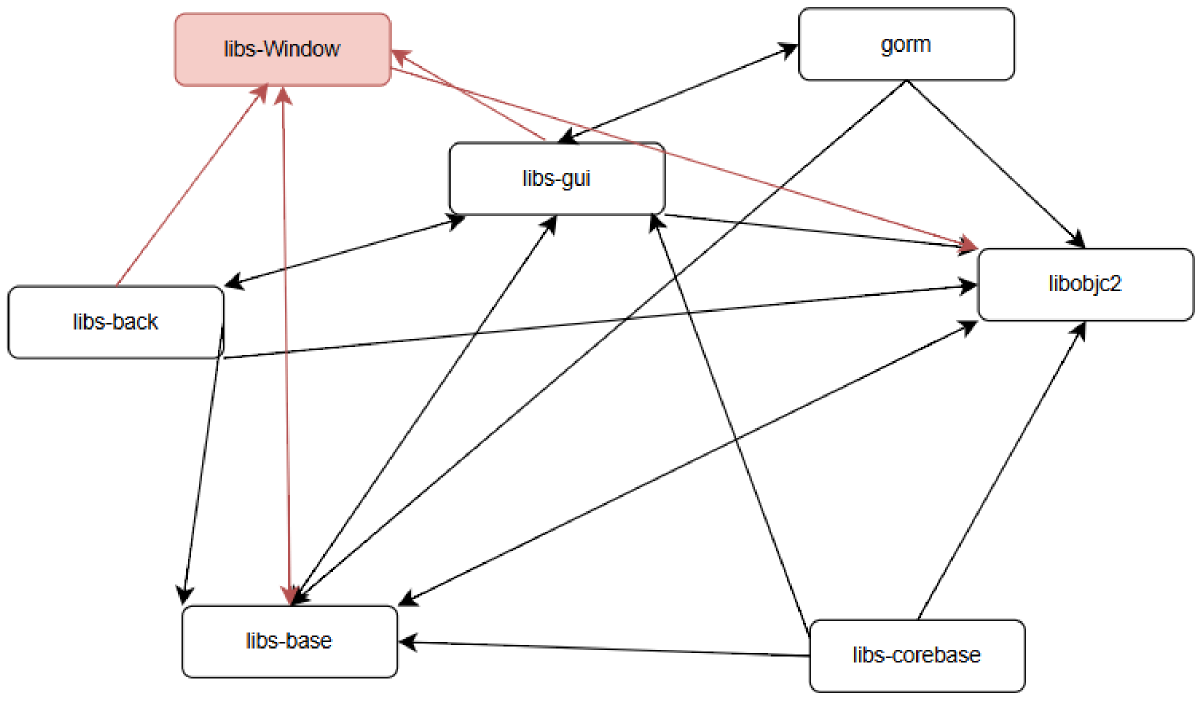
For option 2, (enhancement to AppKit's window handling) we would need to introduce a new internal module, WindowIntegration, within AppKit. We can extend key AppKit classes to delegate window-related operations in order to ensure seamless event translation between AppKit and platform-native windowing systems.

4. *Implementation 1*

a) *Conceptual Architecture*

The following modifications are necessary for a standalone GNUstep window manager: A new window manager component as a core GNUstep component. This module will handle window creation, focus management, resizing, and stacking order, ensuring compatibility with GNUstep's GUI framework.

We will also need to modify libs-gui to communicate directly with the new window manager instead of relying on external window managers. Since libs-back manages rendering and graphical backend interactions, it must be updated to work alongside the new window manager, ensuring proper coordination in drawing and event handling. The window manager will need to interact with libs-base for essential system functionalities such as application launching, even propagation and resource management. It will also rely on libobjc2 for certain low-level operations. We will call this new component: libs-Window.



Conceptual Architecture of GNUstep with a native window manager

b) High and Low Level Interactions

The new standalone GNUstep window manager (libs-Window) will act as a dedicated subsystem for window management. It handles core tasks such as window creation, focus management, resizing, and stacking order. libs-Window directly interfaces with the graphical environment of the application, allowing GNUstep to operate independently of external window managers. Libs-gui is changed directly to interact with libs-Window instead of relying on external ones. The window manager becomes responsible for managing all windows within the GNUstep environment, allowing for a consistent and multi-purpose user experience. Furthermore, the window manager coordinates with libs-back so that rendering and graphical backends are always in sync with the window.

The new window manager integrates deeply with existing GNUstep components. It uses libobjc2 for low-level operations, such as handling Objective-C events and callbacks. The manager registers for GUI events for example focus changes or window close signals; and pushes them through the event handling chain. Libs-window interacts with libs-base for system functionality like launching, event propagation, and even resource management. Furthermore it will directly handle other events in coordination with libs-back so that the visual state of windows is not disrupted.

c) *Impact of Current Directories*

The new window manager libs-window will require the introduction of a new core component into the GNUstep project. This directory will contain the main window management code, which includes all its core functionality.

Furthermore:

- Libs-gui: Will require changes to replace interactions with initial external window managers, as it will directly communicate with libs-Window for window management tasks
- Libs-back: Will require changes to ensure that the backends align with the new libs-Window logic
- Libs-base: Minor updates as the component will also communicate with the new window manager, primarily for the purposes of event propagation
- Libobjc2: libs-window will rely on this components low-level functions for event handling, so altercations to its communication structure is necessary

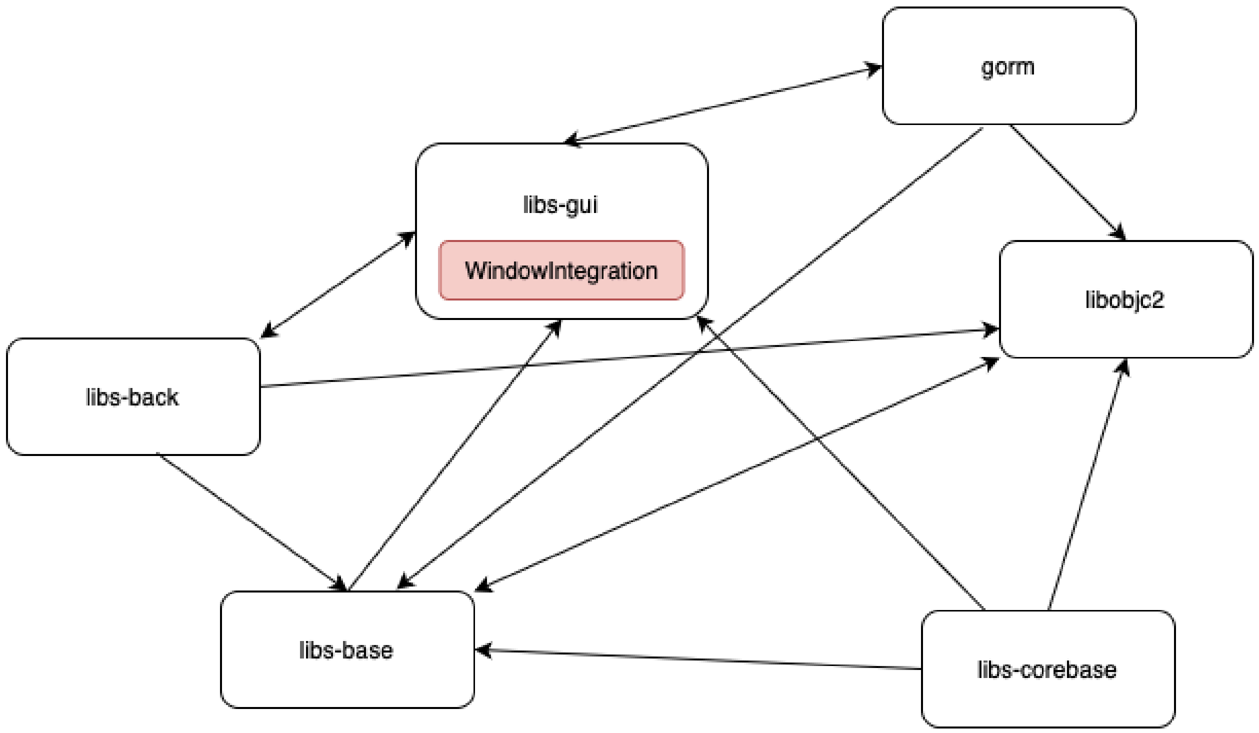
5. *Implementation 2*

a) *Conceptual Architecture*

For our second proposed implementation, we adopt an integrative approach that embeds native window manager support directly into the AppKit layer of GNUstep, rather than introducing a separate subsystem. AppKit, implemented within libs-gui, serves as the primary interface for GUI operations in GNUstep. Our enhancement introduces a new internal module called WindowIntegration, designed to extend the capabilities of key AppKit classes like NSWindow and NSApplication.

These classes are modified to internally delegate platform-specific window operations such as positioning, resizing, window decoration toggling, and stacking order to the new module. This allows GNUstep to support native window behavior while preserving its architectural boundaries and minimizing disruption to the rest of the system.

Importantly, this enhancement does not alter the overall layering of GNUstep's architecture. The WindowIntegration module resides fully within the AppKit component of libs-gui, ensuring that the Foundation layer and core logic remain unaffected. This strategy maintains architectural cohesion, promotes long-term maintainability, and reduces the risk of cross-layer complexity. All native window manager responsibilities are localized within the GUI layer, consistent with its existing responsibilities.



Conceptual Architecture of GNUstep with Implementation 2

b) *High and Low Level Interactions*

At a high level, WindowIntegration functions as a service layer within AppKit. Its purpose is to translate native window manager actions: minimizing, dragging, or closing a window, into Objective-C event patterns that the rest of GNUstep can interpret. Modifications to NSWindow and NSApplication allow these classes to route native platform signals through WindowIntegration, which in turn triggers appropriate delegate methods or UI updates.

At the low level, this module introduces wrapper functions and handlers for native windowing system APIs, primarily Xlib (for X11) and Wayland-compatible interfaces. These wrappers abstract platform-specific logic behind Objective-C interfaces, maintaining consistency across operating systems. For example, system events like focus changes or resize requests are captured at the system level, then propagated through AppKit's responder chain to ensure the application behaves predictably. Crucially, all these interactions are internal to the AppKit layer and require no changes to application-facing APIs. This maintains backward compatibility and avoids disruption to the existing event-handling pipeline. Future testing efforts would include integration tests across multiple environments to ensure WindowIntegration behaves consistently on supported platforms.

c) *Impact of Current Directories*

This implementation introduces a new directory called WindowIntegration/ within the existing gui/ directory. It contains platform-specific code for interacting with native window managers, including C and Objective-C source files that define event listeners, translation layers, and cross-platform abstractions.

Additionally, minor but important changes are made to several key classes inside gui/Classes, particularly NSWindow, NSApplication, and NSView. These changes involve internal hooks that interface with WindowIntegration and route native events appropriately. A new compile-time configuration flag (--enable-native-window-manager) is added to gui/Config, allowing the feature to be optionally enabled or disabled during build time. This ensures flexibility for developers targeting environments where native windowing is not required or supported.

While the back/ directory may require minor adjustments to synchronize rendering behavior with native window states (such as minimizing or occlusion), no changes are needed in the Foundation/ or core subsystems. The limited scope of these modifications ensures the enhancement is implemented in a controlled, maintainable fashion, delivering improved UI behavior while minimizing architectural risk.

6. SAAM Analysis

6.01 Scenarios

Five scenarios were chosen to evaluate GNUstep's architecture:

1. Cross-platform compatibility for application deployment
2. Optimizing performance for low-resource systems
3. Add a new GUI widget (e.g., a custom NSView subclass) to the AppKit framework.
4. Integrating third-party libraries
5. Security integration for sensitive data management

6.02 Stakeholders & Quality Attributes

Quality Attribute	***Stakeholders***
Maintainability	GNUstep Core Team, Maintainers
Evolvability	GNUstep Developers, External Developers
Testability	GNUstep QA Team, Open-source Contributors
Performance	End-users (especially on legacy hardware)
Security	All stakeholders (due to GPL compliance risks)

6.03 Non-Functional Requirements

Maintainability: The GNUstep framework is designed with a modular architecture, which facilitates maintainability. Modules like gnustep-base, gnustep-gui, and AppKit frameworks can be updated or replaced independently. This modularity ensures that the GNUstep Core Team and maintainers can efficiently perform maintenance. Additionally, GNUstep's open-source nature allows contributions from the public, allowing for quicker problem resolutions.

Evolvability: GNUstep's architecture supports evolvability through its modular design. It allows developers to add new features or support new platforms without disrupting the current functionality. The framework's extensibility is very important for both GNUstep developers and external developers since they can build and add new functionalities on top of the core system. GNUstep remains adaptable due to its ability to allow for easy integration of new tools and libraries.

Testability: Each component like the *gnustep-base* and *gnustep-gui* can be individually tested due to the modularity of its framework. This makes it easier for the QA team to detect and address bugs within the program. The availability of unit testing frameworks and automated testing tools ensures that open-source collaborators can contribute to an encompassing test suite.

Performance: Performance is a key consideration for GNUstep legacy users with limited resources. GNUstep's framework is designed to be flexible, allowing developers to optimize performance for specific platforms. Through modularizing various components, GNUstep ensures that performance bottlenecks (areas that slow down or impact performance) can be identified and resolved without affecting the system's functionality. Additionally, performance profiling tools allow for performance improvements, helping end-users on lower-end systems to have a smooth experience using GNUstep.

Security: The security of GNUstep is a concern for all stakeholders, especially due to the GNU General Public License (GPL) compliance risks found in open-source software. GNUstep adheres to strict security practices by ensuring that user data is securely handled and encrypted. However, the responsibility for secure development is dependent on the GNUstep developers. All stakeholders need to be on the lookout for potential security risks.

6.04 Comparison of Architectural Implementations

Implementation 1: Standalone GNUstep Window Manager
Pros:
- Evolvability: Full control over window behaviour, good for GNUstep-specific optimizations
- Performance: Optimized for GNUstep applications, reducing reliance on external managers
- Security: Sandboxed window management reduces exposure to security vulnerabilities

Cons:
- Higher maintenance, as it requires ongoing development to support new platforms

- Compatibility risks, may introduce inconsistencies when other window managers are ran at the same time

Implementation 2: Enhanced Window Handling in AppKit
Pros:
- Maintainability:Uses existing window managers to reduce GNUstep code
- Cross-platform compatibility
- No need to maintain a full window manager

Cons:
- Limited control, as it is more dependent on external window managers, which may not support all GNUstep features
- Potential Performance overhead, additional translation layer between AppKit and GNUstep window managers

6.05 Decision

Implementation 2 is the preferred choice:
1. Lower Maintenance
2. Better Compatibility
3. Easier Testing & Debugging
4. Reduced Risk

Implementation 1 remains a solid long-term option if GNUstep is able to achieve complete independence from external window managers, and if performance optimizations for low-resource systems become critical.

7. Use Cases and Sequence Diagrams

7.01 Use Case 1 Creating and Rendering a New Window

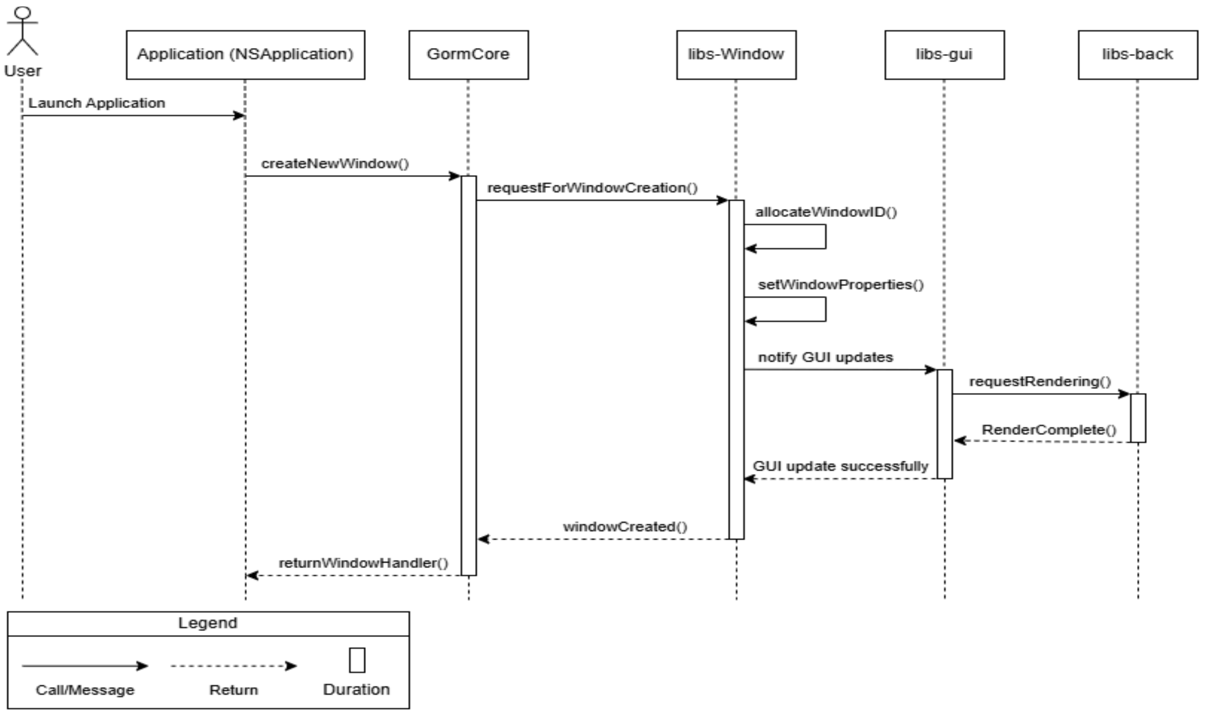


Figure 1. Use Case 1 Creating and Rendering a New Window

In this use case based off of the first implementation, the application launches and initiates the creation of a new window through the new dedicated libs-Window component. The user triggers the application (NSApplication), which delegates window creation to GormCore. GormCore then communicates with the new component libs-Window to request a window instance. libs-Window handles the low-level setup by allocating a new unique window ID and applying initial properties such as size, position, and style. After the setup, it sends a notification to libs-gui which represents the user interface responsible for translating window components into visual elements. Libs-gui then forwards a rendering request to libs-back, directly interacting with the platform's display system. Once this rendering is complete, acknowledgements are returned back, first from libs-back to libs-gui, then from libs-Window to GormCore, and finally back to the application. This flow showcases the modular, layered responsibilities in the enhanced architecture of Implementation 1, ensuring clear separation between logic, window management, UI rendering, and backend display operations.

7.02 Use Case 2 Handling a Window Focus Event with WindowIntegration

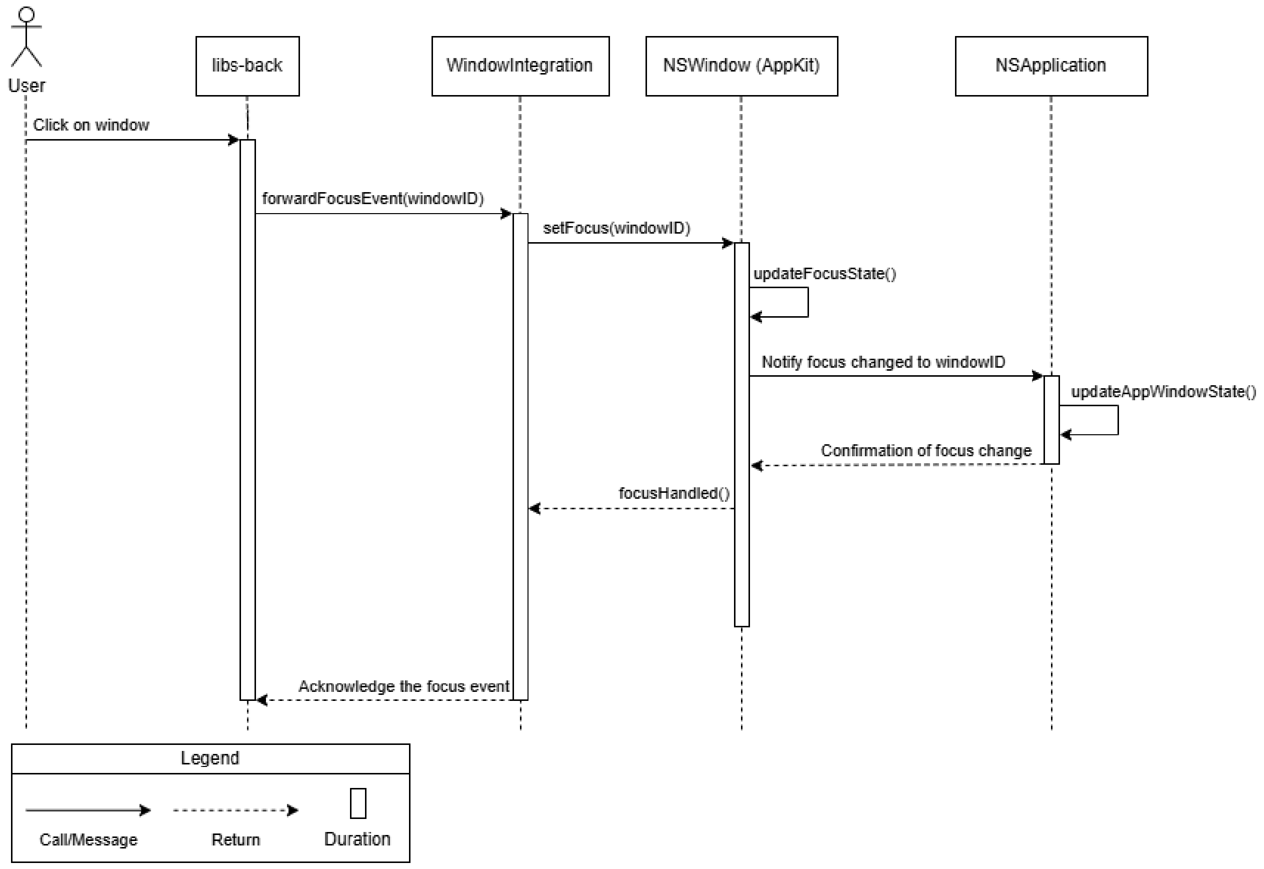


Figure 2. Use Case 2 Handling a Focus Event with WindowIntegration

This use case is based on the second implementation. In the above sequence diagram, the user clicks on an existing application window, prompting a focus event. The event first reaches the underlying platform's window system, which then forwards it to libs-back. In the modified architecture, rather than libs-back sending the event directly to NSWindow in AppKit, it now routes it through the new WindowIntegration component. This intermediary layer interprets the event and passes the focus control to NSWindow. NSWindow then updates its internal state and notifies the corresponding NSResponder chain; a hierarchical event handling structure where events are passed from the window to its responders (like NSView or NSApplication in this case). To keep the application level state in sync, NSWindow notifies NSApplication, which manages the lifecycle and states of all open windows. This modular flow allows for better platform abstraction, reduces tight coupling between AppKit and libs-back, and makes future enhancements to event handling easier.

8. *Plan for Testing*

To design practical tests for our proposal and its interaction with the GNUstep architecture, we require a set of tests satisfying compatibility, performance, usability, stability, and maintainability. We start with two main test groups to evaluate the possible implementations of the proposed enhancement: the standalone GNUstep-native window manager and the enhanced window handling in AppKit. The first set of tests will assess compatibility, usability, and stability, ensuring that GNUstep applications function correctly with the new window management approach. The second set will focus on performance, maintainability, and extensibility, ensuring that the enhancement does not create any unnecessary system overhead.

Some examples of some test cases could be:

Compatibility test: Ensure that existing GNUstep applications work correctly under the new window management system, including proper window placement, resizing, and interaction across different desktop environments

Performance test: Measure CPU and memory usage before the enhancement as well as after to make sure that resource consumption remains efficient and undisrupted.

Usability test: Check whether users experience unexpected changes in window behavior, making sure that interactions remain consistent with GNUstep's design principles. Identify any changes that could disrupt an user's workflow or cause any sort of confusion.

Stability test: Run stress tests by running and opening multiple applications simultaneously and simulating long-term usage to check for crashes, glitches, or any other unexpected behavior.

Maintainability test: Assess how easily the enhancement can be updated, debugged, and integrated into the system along with future updates in GNUstep's architecture.

These tests must consider every system and component that are significantly affected. Depending on the test results, we will then decide if there is a need to conduct broader system-wide testing or possibly focus on specific subsystems. Each test case will be documented and stored in a test suite, which will allow future developers to reference the enhancement over time if needed.

9. Potential Risk

The implementation of our enhancement could introduce several possible risks. We may encounter compatibility issues because GNUstep currently relies on external window managers, and introducing a new native window manager or modifying AppKit may lead to inconsistencies. Existing applications may misbehave and could cause display issues or unexpected interactions with other components. There are also performance concerns, as it is possible that a dedicated window manager will require more resource consumption and modifying AppKit may also add processing overhead.With this new enhancement, maintainability could become a challenge, as the new feature would require continuous updates to support different graphical standards, multi-monitor setups, and different operating systems.

10. Limitations & Lessons Learned

In previous assignments we analyzed pre-existing systems, and while we were successful in designing a new feature, it was much harder. Our first implementation was a more complicated idea which led to us creating the second implementation which was much more practical after we realized the complexities of developing the first idea. We found that without a pre-existing implementation, it was very hard to validate theories. Our key takeaway is that good software design balances creativity with practicality, there needs to be a balance or else it will be very challenging to implement.One of the more notable limitations we came across was GNUstep's reliance on X11/Wayland, that complicated how our implementations would interact with different window managers leading to potential compatibility risks we did not think of before.

11. Data Dictionary

GNUstep: An open-source implementation of Apple's Cocoa frameworks, providing a foundation for developing macOS-compatible applications.

AppKit: The central subsystem within Gorm responsible for managing database interactions and core business logic.

Xlib: A library for interacting with the X Window System, providing low-level functions for window management

Wayland: A protocol that specifies communications between a display server and clients

NSWindow: A class in AppKit that represents a window in a developed GNUstep app.

NSApplication: A class in AppKit responsible for managing app-level events and interactions that are between windows.

libs-base: A core component of GNUstep responsible for essential system utilities, including file management, threading, and object-oriented programming support.

libs-back: The rendering engine of GNUstep that manages graphical rendering and ensures cross-platform compatibility.

libs-corebase: A subsystem that extends GNUstep's compatibility with macOS CoreFoundation, providing additional system utilities.

libs-gui: The primary user interface toolkit within GNUstep, responsible for managing UI elements and event handling.

Objective-C Runtime (libobjc2): The runtime environment for Objective-C applications in GNUstep, supporting dynamic message passing, method resolution, and memory management.

12. Conclusion

This report presented a in-depth analysis of two potential implementations for enhancing the GNUstep platform: a standalone GNUstep-native window manager and an AppKit-integrated solution. Through SAAM analysis, we evaluated both approaches against key NFRs. Our findings led to implementation 2 being our preferred choice due to its lower maintenance, better cross-platform compatibility, and reduced security risk compared to the native window manager. While implementation 1 offers better control and optimization, its complex development and long-term maintenance costs make it a less viable immediate solution. Through use cases, sequence diagrams, and risk assessments, we demonstrated how our implementation integrates with GNUstep's existing architecture without disrupting its functionality. This project was proof of the importance of balancing practicality with innovation in software design. While ambitious solutions may seem appealing, real-world constraints need to be taken into consideration.

13. References

CIO Wiki. (n.d.). Software Architecture Analysis Method (SAAM). Retrieved from: https://cio-wiki.org/wiki/Software_Architecture_Analysis_Method_(SAAM)

GNUstep. (n.d.-a). GNUstep guides index. Retrieved from: https://gnustep.github.io/Guides/index.html

GNUstep. (n.d.-b). GNUstep on Windows. Retrieved from https://www.gnustep.org/windows/

GNUstep. (n.d.-c). Introduction to GNUstep applications. Retrieved from: https://gnustep.github.io/Guides/App/1_Intro/index.html

GNUstep. (n.d.-d). NSWindow class reference. Retrieved from: https://home.gnustep.org/resources/OpenStepSpec/ApplicationKit/Classes/NSWindow.html

GNUstep MediaWiki. (n.d.). AppKit. Retrieved from: https://mediawiki.gnustep.org/index.php/AppKit

Kazman, R., Abowd, G., Webb, M. (1994). SAAM: A Method for Analyzing the Properties of Software Architectures. Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.65&rep=rep1&type=pdf