

<https://youtu.be/V7H0UWjYGyA>

Big Hero 6 Team

Team Members and Roles

Team Big Hero 6 - Group 30

- Terence Jiang - Use Cases and Sequence Diagrams
- Kevin Chen - SAAM Analysis, Lessons Learned, Conclusion, References
- Alexander Zhao - Presenter, Slide Presentation, Data Dictionary, Implementation 1
- Carter Gillam - Presenter, Implementation 2
- Kavin Arasu - Abstract, Introduction & Overview, Proposed Enhancement, Implementation 1
- Daniel Gao - Plan for Testing, Potential Risk

Proposal for Enhancement Report on GNUstep

April 4, 2025

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.

Introduction & Overview

- We have developed a clear understanding of GNUstep's conceptual and concrete architecture, as well as its functionality, through previous assignments.
- This report proposes an enhancement called GNUstep Window Integration, aiming to improve window management for GNUstep applications through a dedicated native window manager.
- Currently, GNUstep does not have its own window manager and relies on the OS-provided one (like Window Maker on Linux).
- The report outlines two possible implementations: (1) a standalone GNUstep window manager, and (2) enhanced window handling within AppKit to improve integration with external window managers.
- A SAAM analysis will be conducted to evaluate both implementations, focusing on maintainability, usability, and system compatibility while identifying key stakeholders.
- The chosen implementation will be presented in detail, including functionality, use cases, and sequence diagrams.
- We will discuss potential risks, challenges, and limitations associated with the enhancement, as well as key lessons learned from the analysis.
- The ultimate goal of the enhancement is to make GNUstep more self-contained and improve application usability with seamless and predictable window management.

Proposed Enhancement

Core Framework Improvements for GNUstep

The proposed enhancement introduces a dedicated GNUstep-native window manager to improve application integration with modern graphical environments.

Two potential implementations are considered:

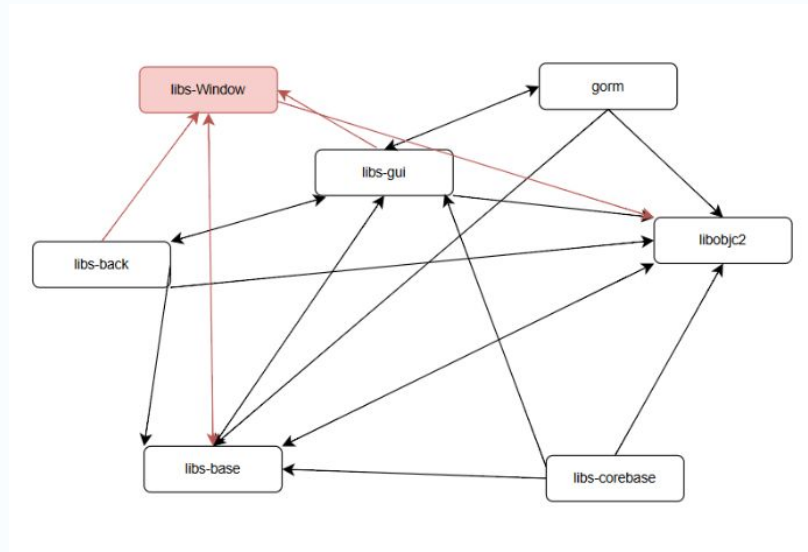
1. **Standalone GNUstep Window Manager:** A custom manager specifically for GNUstep, ensuring tight integration with its graphical framework.
2. **Enhanced Window Handling in AppKit:** Modifying AppKit to better interact with existing window managers, allowing seamless functionality across various operating systems.

Both approaches focus on improving application usability, window consistency, and integration while balancing maintainability, compatibility, and ease of adoption.

- Key benefits of the enhancement include a more predictable and cohesive user experience, reducing inconsistencies across platforms and offering a more seamless desktop experience.
- A native window manager also reduces dependency on external window managers, minimizing compatibility issues and allowing for GNUstep-specific window management features.
- Implementing a standalone window manager requires introducing a new GNUstep component that supports standard window management features while integrating with existing components.
- Enhancing AppKit's window handling involves creating a new WindowIntegration module within AppKit, extending key classes to delegate window operations and ensure compatibility with platform-native systems.

Implementation 1: Conceptual Architecture

- Introduce a new window manager component (libs-Window) as a core part of GNUstep, responsible for window creation, focus management, resizing, and stacking order, ensuring compatibility with GNUstep's GUI framework.
- Modify libs-gui to communicate directly with the new window manager, replacing reliance on external managers, and update libs-back for coordinated rendering and event handling.
- Ensure the window manager interacts with libs-base for system functionalities (application launching, event propagation, resource management) and libobjc2 for low-level operations.



Implementation 1: High and Low Level Interactions

Component Interactions in GNUstep Enhancement

- High-Level Interaction: The new standalone window manager (libs-Window) acts as a dedicated subsystem, handling window creation, focus management, resizing, and stacking order, enabling GNUstep to function independently of external window managers.
- Integration with libs-gui and libs-back: Modify libs-gui to interact directly with libs-Window instead of external managers. The window manager ensures consistent user experience by managing all windows within the GNUstep environment and coordinating with libs-back for synchronized rendering and graphical backend operations.
- Low-Level Interaction: Integrate libobjc2 for handling Objective-C events and callbacks, and use libs-base for system functionalities (event propagation, resource management). The window manager registers for GUI events (e.g., focus changes, window close signals) and coordinates with libs-back to maintain visual consistency of windows.

Implementation 1: Impact on Current Directories

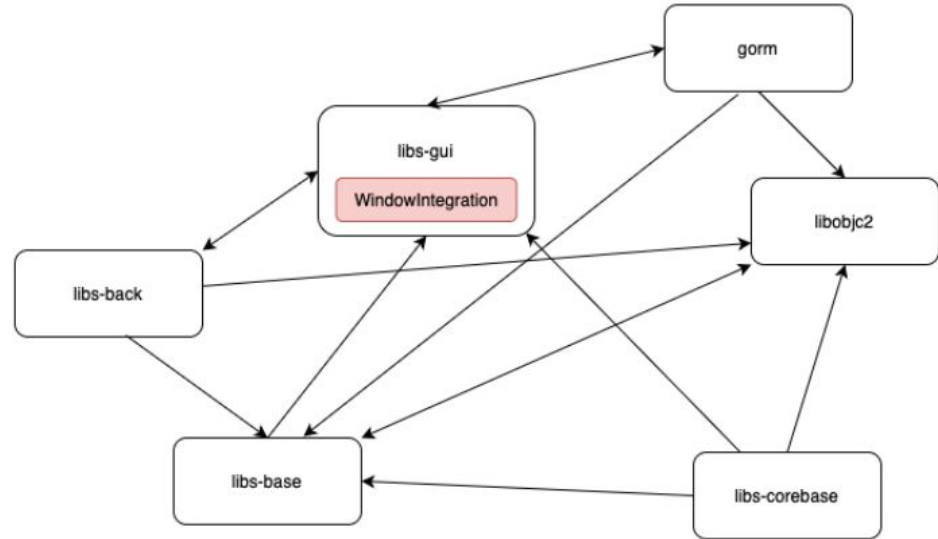
The new window manager `libs-window` will require the introduction of a new core component into the GNUstep project. This directory will contain the main window management code, which includes all its core functionality.

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 alterations to its communication structure is necessary

Implementation 2: Conceptual Architecture

- Integrative Approach: Embeds native window manager support directly into the AppKit layer of GNUstep, avoiding the need for a separate subsystem.
- WindowIntegration Module: Introduces a new module within libs-gui to extend NSWindow and NSApplication, delegating platform-specific window operations (e.g., positioning, resizing).
- Architectural Cohesion: Maintains GNUstep's architectural boundaries by localizing window manager responsibilities within the GUI layer, preserving layering consistency.
- Minimal Disruption: Ensures the Foundation layer and core logic remain unaffected, promoting long-term maintainability and reducing cross-layer complexity.



Implementation 2: High and Low Level Interactions

- **High-Level Functionality:** WindowIntegration functions as a service layer within AppKit, translating native window manager actions (e.g., minimizing, dragging, closing) into Objective-C event patterns for GNUstep.
- **Class Modifications:** Changes to NSWindow and NSApplication allow these classes to route native signals through WindowIntegration, triggering delegate methods or UI updates.
- **Low-Level Abstraction:** Introduces wrapper functions and handlers for native windowing system APIs, mainly Xlib (X11) and Wayland, abstracting platform logic behind Objective-C interfaces.
- **Event Propagation:** System events (e.g., focus changes, resize requests) are captured at the system level and routed through AppKit's responder chain, ensuring predictable application behavior.
- **Internal Operations:** All interactions remain internal to AppKit, maintaining backward compatibility and avoiding changes to application-facing APIs.
- **Cross-Platform Consistency:** Ensures consistent behavior across different operating systems by using standardized event handling.
- **Testing and Validation:** Future testing will involve integration tests across multiple environments to validate consistency.
- **No API Disruption:** Maintains the existing event-handling pipeline, preventing disruption and preserving existing application functionality.

Implementation 2: Impact on Current Directories

- **New Directory:** Introduces a `WindowIntegration/` directory within the existing `gui/` directory, containing platform-specific code (C and Objective-C) for interacting with native window managers.
- **Class Modifications:** Updates key classes (`NSWindow`, `NSApplication`, `NSView`) inside `gui/Classes` with internal hooks for native event routing.
- **Configuration Flexibility:** Adds a compile-time flag (`--enable-native-window-manager`) in `gui/Config` to enable or disable native windowing support during build time.
- **Minimal Back-End Changes:** Only minor adjustments in the `back/` directory to sync rendering with native window states; no changes to `Foundation/` or core subsystems, maintaining a controlled, maintainable enhancement.

SAAM Analysis

The Software Architecture Analysis Method (SAAM) is used to evaluate different quality attributes of software architectures based on their ability to meet a variety of non-functional requirements (NFRs). As described by CIO Wiki, SAAM is "a method used to evaluate the architecture of a software system based on scenarios of how the system is expected to be used or changed in the future" (CIO Wiki, n.d.).

SAAM Analysis Cont.

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

CISC 322/326 - Big Hero 6 - Assignment 3

<i>Quality Attribute</i>	<i>Stakeholders</i>
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)

Figure 1: Stakeholders & Quality Attributes

SAAM Analysis Cont. NFR Analysis

Maintainability:

- Modular architecture (e.g., gnustep-base, gnustep-gui, AppKit) supports independent updates and replacements.
- Open-source nature allows public contributions, enabling faster problem resolution.

Evolvability:

- Modular design supports adding new features and platform support without affecting current functionality.
- Extensibility benefits both GNUstep and external developers for building and integrating new functionalities.

Testability:

- Modularity allows individual testing of components like gnustep-base and gnustep-gui.
- Availability of unit testing frameworks and automated tools helps ensure comprehensive

SAAM Analysis Cont. NFR Analysis

Performance:

- Designed to be flexible and optimized for specific platforms.
- Modularization helps identify and resolve performance bottlenecks without impacting system functionality.
- Performance profiling tools assist in maintaining smooth performance on lower-end systems.

Security:

- Adheres to strict security practices, including secure data handling and encryption.
- Compliance with GNU General Public License (GPL) helps address open-source security concerns.
- Developers and stakeholders are responsible for ongoing vigilance against security risks.

SAAM Analysis Cont.

Implementation 1: Standalone GNUstep Window Manager

- Pros:
 - Evolvability: Full control over window behavior, ideal for GNUstep-specific optimizations.
 - Performance: Optimized for GNUstep applications, reducing reliance on external window managers.
 - Security: Sandboxed window management reduces exposure to security vulnerabilities.
- Cons:
 - Higher Maintenance: Ongoing development needed to support new platforms.
 - Compatibility Risks: Inconsistencies may arise when used alongside other window managers.

SAAM Analysis Cont.

Implementation 2: Enhanced Window Handling in AppKit

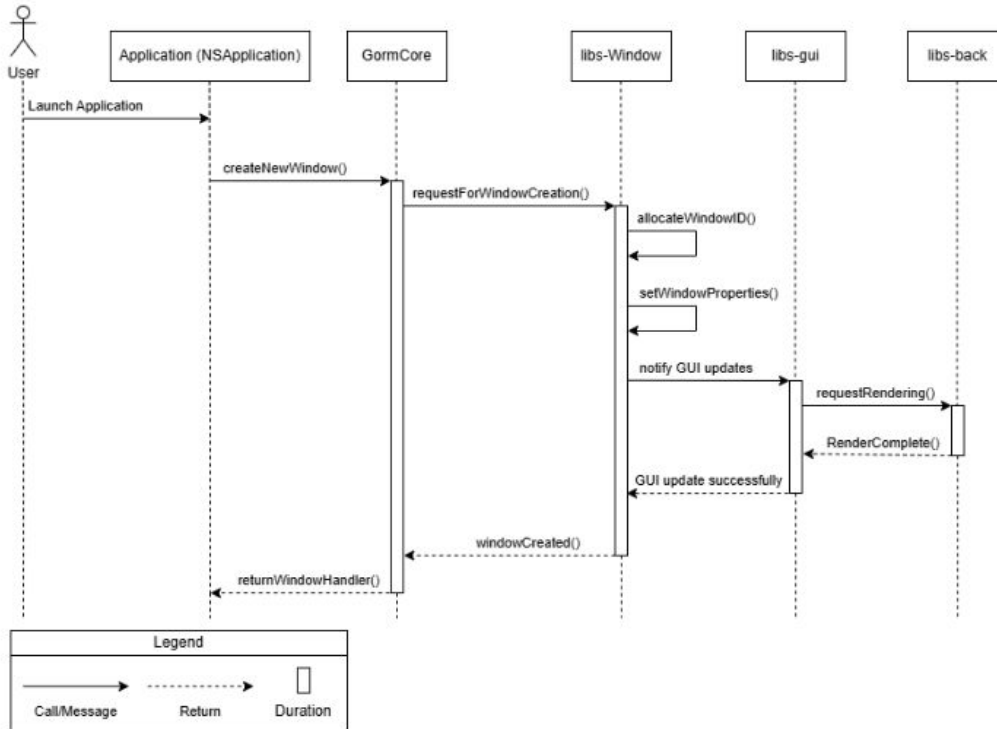
- Pros:
 - Maintainability: Uses existing window managers, reducing the need for additional GNUstep code.
 - Cross-platform Compatibility: Works seamlessly across multiple platforms.
 - No need to maintain a full window manager: Leverages external window managers without the complexity of maintaining one.
- Cons:
 - Limited Control: Dependent on external window managers, which may not fully support all GNUstep features.
 - Potential Performance Overhead: Adds an additional translation layer between AppKit and window managers.

SAAM Analysis Cont.

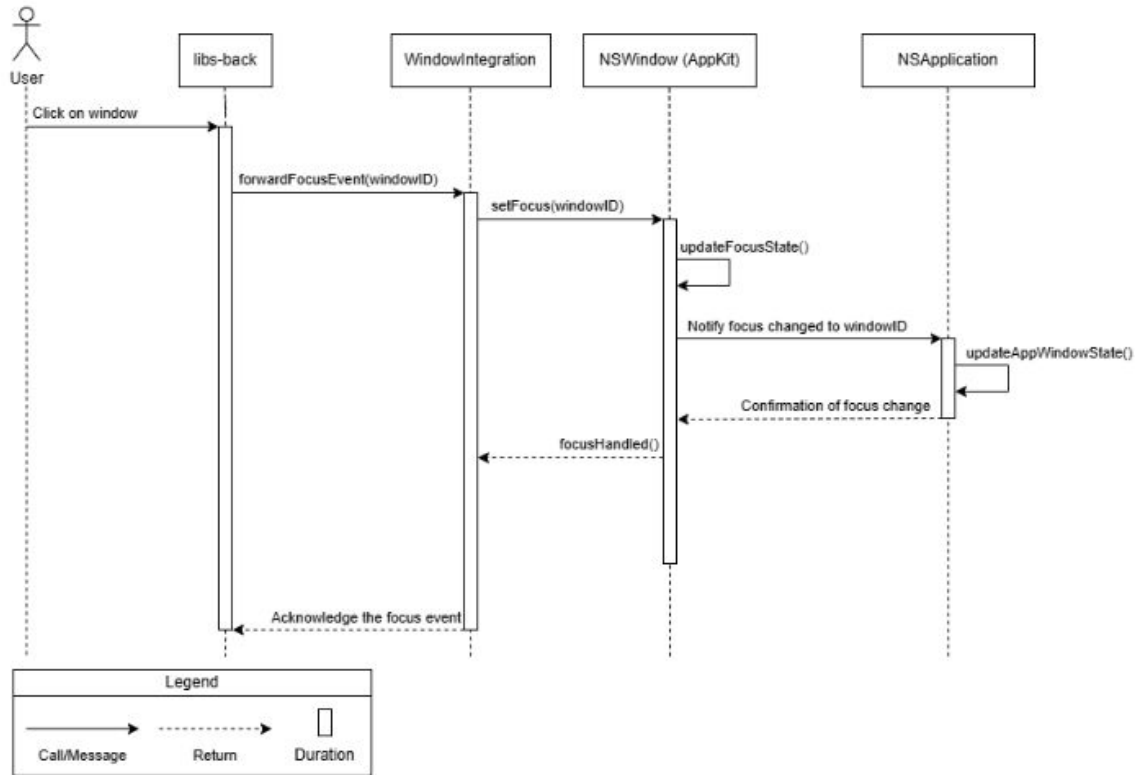
Decision

- Preferred Choice: Implementation 2
 - Lower Maintenance
 - Better Compatibility
 - Easier Testing & Debugging
 - Reduced Risk
- Implementation 1 remains a solid option for long-term use if GNUstep can become fully independent from external window managers and if low-resource performance optimizations become critical.

Use Case 1



Use Case 2



Plan for Testing

Test Design for Proposed Enhancement:

- Test Groups:
 1. Compatibility, Usability, and Stability Tests.
 2. Performance, Maintainability, and Extensibility Tests.
- Example Test Cases:
 1. Compatibility Test: Verify proper window behavior (placement, resizing, interaction) across different desktop environments with existing GNUstep applications.
 2. Performance Test: Measure CPU and memory usage before and after the enhancement to ensure efficient resource consumption.
 3. Usability Test: Ensure consistent window interactions in line with GNUstep's design principles and check for any disruptions in user workflows.

Plan for Testing Cont.

1. Stability Test: Run stress tests with multiple applications to check for crashes, glitches, or unexpected behaviors during long-term usage.
2. Maintainability Test: Assess how easily the enhancement can be updated, debugged, and integrated with future updates to GNUstep.

Test Documentation:

- Store test cases in a suite for future reference, enabling easy tracking and evaluation of the enhancement's integration over time.



Potential Risks


Compatibility Risks:

- Potential inconsistencies due to reliance on external window managers; new window manager or AppKit modifications could cause display issues or unexpected application behavior.

Performance Risks:

- A dedicated window manager might increase resource consumption, and modifications to AppKit could introduce additional processing overhead.

Maintainability Risks:

- The new feature could require continuous updates to accommodate different graphical standards, multi-monitor setups, and operating systems, adding complexity to maintenance.
- 

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.

References

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