Adding Dynamic Theme Management to GNUstep

Architectural enhancement of GNUStep

Authors:

- Seyed Ebrahim Haghshenas (21seh22@queensu.ca)
- Masih Hashemi (masih.h@queensu.ca)
- Kiarash Mirkamandari (kiarash.mirkamandari@queensu.ca)
- Bahar Rahimivarposhti (23vs27@queensu.ca)
- Kiarash Soleimani Roozbahani (21ksr5@queensu.ca)
- Andrew Ternopolsky (21at103@queensu.ca)

1. Abstract

This report suggests and discusses a Dynamic Theme Manager for GNUstep that enables users to change UI looks (light/dark modes, individual color schemes) at runtime. We present the case for an independent theming subsystem and explain how it fits into GNUstep's current layered architecture (libs-base, libs-gui, libs-back). With architecture diagrams and a Software Architecture Analysis Method (SAAM) comparison of design options (plugin-based versus core integration), we show how the new capability can add usability, extensibility, and maintainability without disrupting core aspects. Risks and performance effects are addressed, along with high-level testing methods to ensure confidence for reliable upgrades. Our results show that a notification-based, modular theming is a good fit for the GNUstep architecture, delivering a modern user experience and facilitating future development.

2. Introduction

GNUstep is an open-source implementation of the OpenStep specification that combines object-oriented libraries (Foundation, AppKit equivalents) with platform-specific backends. Though basic visual theming is all that is supported, current users want complete, on-the-fly theming. This paper resolves this mismatch by presenting a new Dynamic Theme Manager. After a brief description of the current GNUstep architecture, we describe how the manager consolidates theme data, communicates with existing components, and supports various realizations of features. We compare each of the alternate designs with a SAAM analysis, highlighting the performance, complexity, and user experience compromises. The goal is to illustrate one feasible approach to changing the look and feel of GNUstep without shattering its already well-established layered design.

3. Architecture

This section outlines how the proposed Dynamic Theme Manager feature integrates into (and modifies) the existing GNUstep architecture, leveraging details from our previous analysis (see A2 report). We begin by summarizing the relevant parts of the current design, then describe the new components, interfaces, and interactions required for dynamic theme switching. We also highlight the impact of these changes on maintainability, testability, performance, and evolvability.

3.1 Overview of the Existing Architecture

From our prior study, GNUstep's current concrete architecture can be understood as a layered set of libraries, each responsible for different aspects of the OpenStep-inspired framework:

1. **libs-corebase**
    - Provides low-level C-based features (partial equivalents to Apple's CoreFoundation).
    - In practice, not all calls go through libs-corebase; some system operations bypass it for performance or lack of implementation.
2. **libs-base** (Foundation)
    - Core Objective-C classes (e.g., NSObject, NSString, NSArray) and runtime support (e.g., NSRunLoop, NSThread).
    - Also provides OS-integration features such as file I/O, networking, and concurrency.
    - Often interfaces directly with the operating system if libs-corebase is incomplete.
3. **libs-gui** (AppKit)
    - Contains core GUI classes (e.g., NSWindow, NSView, NSButton), event handling, the NSApplication event loop, and higher-level UI abstractions.
    - Implements the fundamental Model-View-Controller (MVC) patterns that separate data from user interface code.
    - Relies heavily on libs-base for concurrency and system services.
4. **libs-back** ("gnustep-back")
    - Bridges libs-gui with platform-specific rendering (e.g., X11 on Linux, Win32 on Windows).
    - Receives high-level draw and display requests from libs-gui, then translates them into OS-specific calls (e.g., XCreateWindow on Unix-like systems or GDI calls on Windows).
    - Multiple backends exist for different platforms or rendering libraries (e.g., Cairo, Win32).
5. **libobjc2**
    - The modern Objective-C runtime powering GNUstep.

    - Provides message dispatch, memory management behaviors, categories, and reflection capabilities.
    - Heavily used by libs-base and libs-gui to enable dynamic patterns like delegation, target-action, etc.

Despite a clear conceptual layering, pragmatic shortcuts have developed over time (e.g., direct OS calls from libs-base, incomplete coverage in libs-corebase). Nevertheless, the layering approach remains recognizable and is especially visible in the division between libs-gui (platform-agnostic UI) and libs-back (platform-specific rendering).

3.2 Rationale for a Dynamic Theme Manager

Currently, GNUstep supports only rudimentary theming. While users can modify certain interface elements (fonts, colors, etc.) via configuration files, there is no *comprehensive or runtime-adjustable* mechanism for switching between a "light" theme, a "dark" theme, or any other custom appearance set on the fly. In a modern environment, dynamic theming:

- Improves usability: Allows users to quickly switch to high-contrast or dark modes to reduce eye strain.
- Facilitates accessibility: Helps accommodate different visual needs without requiring restarts or manual code changes.
- Increases extensibility: Developers can create custom look-and-feel "skins" or brand-specific themes for their GNUstep applications.

Therefore, our enhancement introduces a new Dynamic Theme Manager that:

1. Centralizes theme-related data and settings.
2. Provides an API to query, load, and apply new appearance settings at runtime.
3. Coordinates with libs-gui to propagate those updates through all relevant UI elements.
4. Optionally interacts with libs-back if low-level rendering changes are required (e.g., if a user picks a custom color scheme demanding subtle changes in draw operations).

3.3 High-Level Architectural Modifications

The new Theme Manager (TM) will be introduced as either an extension of libs-gui (Option A: Core Integration) or as a separate module that plugs into libs-gui (Option B: Plugin-Based). In either approach, maintaining the existing layered structure is key:

- **Location of Theme Manager Logic**
    - **Option A (Core Integration)**:
        - The TM becomes part of libs-gui, inheriting direct access to core UI classes (NSView, NSWindow) and hooking into the event loop.

        - Pros: Less overhead on event dispatch and direct synergy with existing GUI classes.
        - Cons: Harder to isolate theming logic for testing or updates, risk of more invasive changes to libs-gui.
    - **Option B (Plugin-Based)**:
        - The TM lives in a separate library (e.g., libs-theme) but communicates with libs-gui through a well-defined interface.
        - Pros: Clear modular boundary, easier to test/replace the theming subsystem, minimal risk to the stability of libs-gui.
        - Cons: Slight performance overhead from additional layer of indirection, requires adding or extending an interface that can handle theme events.
- **API Changes**
    - A new set of Objective-C classes or protocols for reading, storing, and applying theme data.
    - Potentially modifies existing classes in libs-gui (e.g., NSWindow, NSApplication) to accept "theme update notifications" or to query the TM for color/font updates.
    - A method (or set of methods) to trigger theme changes at runtime—for instance:

      [NSThemeManager.sharedManager applyTheme:@"DarkModeTheme"];

    - If the user modifies the theme, the manager broadcasts notifications, prompting each visual component to refresh itself.
- **Configuration Storage and Integration**
    - The existing .plist infrastructure (in libs-base) or other custom JSON/XML files can store theme definitions.
    - The TM's **configuration submodule** loads theme data from these sources.
    - Integration with system-level preferences (if desired) to allow an entire desktop environment's theme to sync with GNUstep applications.
- **Interaction with libs-back**
    - If theming changes require advanced rendering (e.g., switching from a "flat" theme to a "textured" one, or toggling certain anti-aliasing options), the TM must send commands to libs-back.
    - The hooking may remain minimal if theming is purely color-and-font-based, but the design must allow for deeper customizations that might affect the backend's rendering pipeline.

3.4 Detailed Subsystem View

3.4.1 libs-base

- **Before Enhancement**:

    - Provides fundamental data structures (NSString, NSDictionary) and concurrency features.
    - Acts as the foundation for higher-level frameworks.
- **Changes**:
    - **Optional**: A “Theme Preferences” class in libs-base that reads/writes .plist or other config files (since theming data must be persistently stored).
    - Minor changes to concurrency if theme updates are triggered off the main thread. Typically, UI updates should occur on the main thread in a Cocoa-like environment, but the manager might need background loading for performance reasons.

3.4.2 libs-gui

- **Before Enhancement**:
    1. Primary library for windows, views, and event handling.
    2. Manages user interactions, dispatching them to UI objects.
- **Changes**:
    1. **Theme Manager Integration**:
        - Incorporate a new NSThemeManager class (or “manager” object) capable of:
            - Storing a list of installed themes.
            - Broadcasting notifications (e.g., NSThemeDidChangeNotification) to all UI components.
        - Each UI component (e.g., NSWindow, NSView, NSButton) listens for these notifications and calls a method like - (void)applyCurrentTheme to update colors, fonts, and other style attributes.
    2. **Event Loop Hooks**:
        - If the user triggers a “Switch Theme” action from the menu, the event loop calls a theme-switching function in the manager.
    3. **MVC Adjustments**:
        - The Model (data) for the theme resides within the new manager or in configuration files.
        - The View (e.g., NSWindow) references that model whenever it draws or re-renders.
    4. **Extensibility for Third-Party Apps**:
        - Provide an official “theme interface,” enabling external developers to define custom theme bundles that the manager can install at runtime.

3.4.3 libs-back

- **Before Enhancement**:
    - Translates libs-gui requests into OS-specific drawing commands.

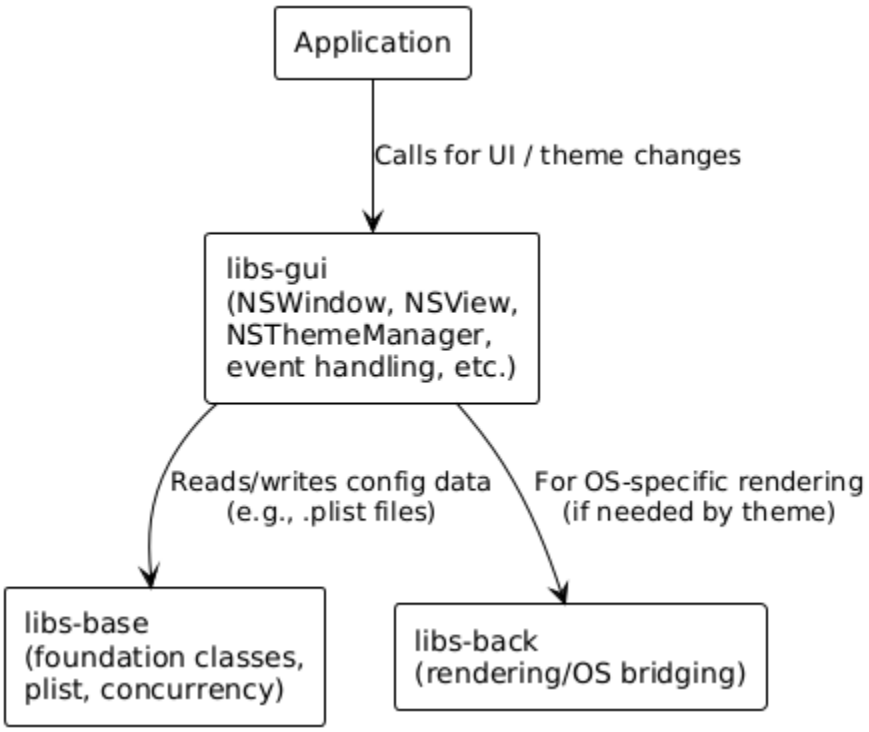
    - May incorporate multiple rendering backends (Cairo, Win32, X11, etc.).
- **Changes**:
    - Potentially minimal if the new theming is restricted to color and font changes—these can be handled by libs-gui classes before calling libs-back.
    - For more elaborate themes (e.g., different window chrome, custom shadows, additional texture-based rendering), new hooks or method calls to libs-back might be introduced:

      [NSBackendManager.sharedBackend updateWindowShadowStyle:self.window withTheme:currentTheme];

    - If advanced theming is desired (custom window shapes or transparency), libs-back code must be extended to handle the relevant OS-level APIs.

3.5 Updated Dependency Diagram

Below is a high-level sketch of how the proposed TM integrates into the existing architecture



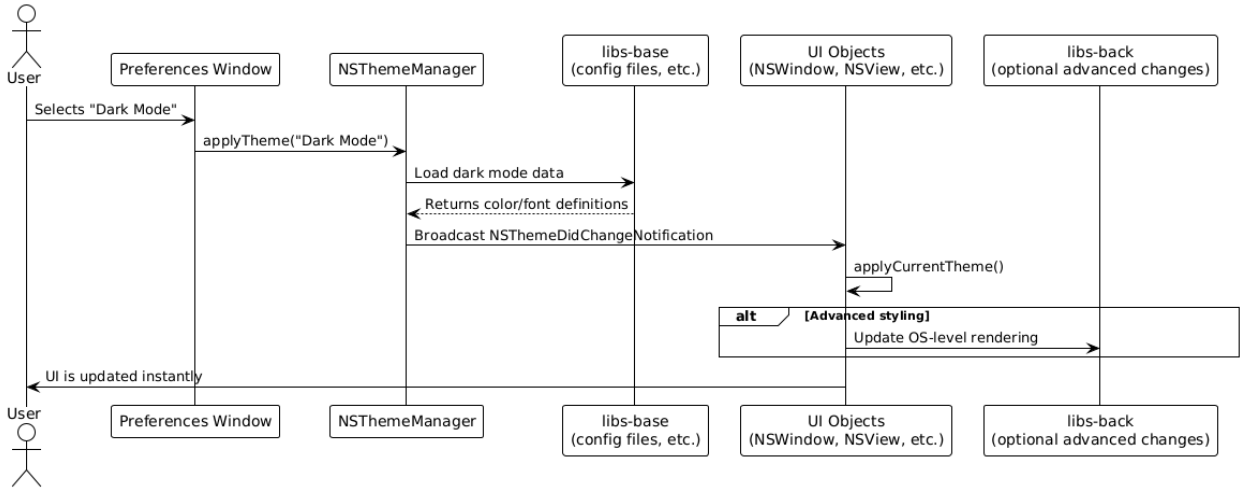
1. **Application Layer** calls libs-gui—for instance, the user clicks “Switch Theme.”
2. **libs-gui** calls the new Theme Manager object (within the same library or as a separate module) to apply or load themes.
3. **libs-gui** broadcasts theme updates to each UI element, possibly tapping into libs-base for reading .plist files or user defaults.

4. **libs-back** is invoked only if certain theme changes require low-level OS rendering adjustments.

3.6 Architectural Impacts

1. **Maintainability**
    - Introducing a dedicated module or class for theming supports cleaner separation of concerns.
    - If integrated natively into libs-gui, code changes will be more invasive but might be simpler to maintain in one place.
    - A plugin-based approach requires a stable, well-documented interface but can reduce the churn within libs-gui.
2. **Evolvability**
    - Future expansions (e.g., advanced transitional animations, additional color profiles) become simpler if theming logic is centralized.
    - A modular design allows new theme bundles or more robust custom styling (e.g., theme designers who are not core GNUstep developers).
3. **Testability**
    - The theme manager can be tested in isolation, mocking out libs-gui calls.
    - UI snapshots or “visual regression tests” can confirm that theme changes are correctly applied.
4. **Performance**
    - Theme changes at runtime trigger redraw calls. On large applications, repeated or frequent theme switching could create a momentary overhead.
    - Caching of frequently used theme assets (colors, images, icons) is recommended to mitigate performance hits.
    - If concurrency is used for loading theme data in the background, potential race conditions must be carefully managed.
5. **Risks**
    - An incomplete or buggy theming API could destabilize the entire UI if unhandled color updates or partial loads lead to inconsistent rendering states.
    - Overly invasive changes in libs-gui might introduce regressions in existing applications that assume static theme data.

3.7 Example Flow: Applying a Theme at Runtime



1. User selects a new theme (e.g., Dark Mode) in the Preferences Window (part of libs-gui).
2. The NSThemeManager loads data from libs-base (such as a .plist or configuration file).
3. The manager then notifies all UI elements (windows, views) to refresh themselves with the new style.
4. If additional OS-level changes are needed (e.g., window shadows, custom shapes), the UI objects call into libs-back.

3.8 Impacted Files and Directories

To implement the Dynamic Theme Manager, multiple new files must be added, and some existing components must be updated. These following files and directories are expected to be impacted:

New files:

- **Headers/gui/NSThemeManager.h:** Declares the interface for the Theme Manager.
- **Source/gui/NSThemeManager.m:** Implements a runtime theming logic, event handling, and API functions.
- **Resources/Themes/DarkTheme.plist, LightTheme.plist, etc.:** Sample or default theme definitions stored as configuration files.
- **Source/base/NSUserDefaults+ThemePrefs.m:** Provides convenience methods for storing and retrieving theme preferences. (Optional.)

Modified Files:

- **Headers/gui/NSWindow.h, NSView.h, NSButton.h:** Add a method such as **-(void)applyCurrentTheme** and notification registration.
- **Source/gui/NSWindow.m, NSView.m, NSButton.m:** Implement the theme application logic responding to **NSThemeDidChangeNotification**.

- **Source/gui/NSApplication.m:** Integrate theme change hooks (e.g., from preferences menu or system signal).

Directories:

- **gnustep/core/gui/:** Primary location for new theming code and modified UI components.
- **gnustep/core/base/:** (Optional) Theme preference storage extensions.
- **gnustep/core/back/:** May require updates for low-level rendering if advanced theming is implemented (e.g., shadows, transparency).

These changes will keep the original architectural layering and add the ability to use modular integration of theming capabilities without significantly disrupting other system features.

4. External Interfaces

Although the new Dynamic Theme Manager is primarily an internal GNUstep feature, it must still interact with several external entities:

1. **Configuration Files and User Preferences**
    - **Plist or JSON Files**: The Theme Manager loads theme data (colors, fonts, layout attributes) from .plist or JSON files stored in the user's configuration directory. These files define all appearance-related parameters (e.g., light/dark variants, custom color palettes) and can be updated or replaced without recompiling GNUstep.
    - **User Defaults**: Where relevant, the manager can store user preferences (e.g., last-selected theme) in the existing user defaults system. This ensures that the chosen theme persists across application restarts.
2. **Operating System / Window Manager**
    - **Rendering Calls**: When a theme change includes alterations to window decorations, shadows, or transparency, the Theme Manager instructs libs-gui to pass updated rendering instructions to libs-back. This may involve additional OS or window manager calls for advanced visual features.
    - **System Theme Signals (Optional)**: On some platforms, the OS or desktop environment can emit global theme-change signals. If GNUstep chooses to integrate with these, the Theme Manager would listen for such signals and automatically switch the application's appearance accordingly.
3. **Third-Party Extensions**
    - **Theme Bundles or Plugins**: Developers outside the core GNUstep project can create custom theme bundles. These bundles communicate with the Theme Manager via a documented interface (e.g., providing color assets or custom drawing code). The manager integrates these bundles into its theme registry, making them available to end users.

5. Use Cases

The other major use case is described in Section 3.7.

Use Case: Customizing Theme Settings

Goal/Context:
A user or developer wishes to modify the appearance of GNUstep's UI (e.g., colors, fonts, layout attributes) to create or update a custom theme.

Primary Actors:

- **End User:** An individual customizing the interface for aesthetic or accessibility reasons.
- **Theme Manager (System):** The new Dynamic Theme Manager component responsible for reading, storing, and broadcasting theme preferences.

Preconditions:

- GNUstep is running, and the user has access to some form of Preferences or Settings interface.
- A default or previously selected theme is already applied.

Main Success Scenario (Typical Flow):

1. **User Opens Preferences:** The user navigates to the application's Preferences panel (hosted by libs-gui).
2. **Selects "Theme" Section:** They choose a "Theme" or "Appearance" tab to view current UI customization options.
3. **Modifies Appearance Settings:** The user changes specific parameters (e.g., window background color, default font style, accent colors) via sliders, dropdowns, or color pickers.
4. **Theme Manager Updates Data:** Upon confirmation (e.g., clicking "Save" or "Apply"), the Preferences panel calls the Theme Manager's API (within libs-gui or a dedicated module).
5. **Configuration Storage:** The Theme Manager writes or updates the user's custom settings in a .plist or JSON file through libs-base I/O operations.
6. **Preview/Immediate Feedback (Optional):** If the design allows, the user sees a live preview or partial update of the new style.
7. **Saved Custom Theme:** The updated theme is now saved and can be referenced later (e.g., under a new theme name or overriding the current one).

Extensions/Variations:

- **Invalid Color or Font:** If a user selects a color/font that does not exist or is unsupported, the Theme Manager falls back to a default option.
- **Partial Save:** The user might discard changes mid-way. The manager reverts to the last valid theme or does not commit the modifications to disk.
- **Programmatic Customization:** Advanced users/developers might edit .plist or JSON files directly, bypassing the Preferences UI. The Theme Manager picks up changes the next time the app refreshes or restarts.

Postconditions:

- The user's custom theme is stored on disk.
- These new settings are now available for future use, either immediately if the system applies them instantly or upon the next theme-switching action.

6. Data Dictionary

- **Dynamic Theme Manager**
  A subsystem or module responsible for loading, storing, and applying visual styling data (colors, fonts, images, etc.) across GNUstep applications at runtime.
- **Theme**
  A defined set of appearance attributes (e.g., primary color, secondary color, default font, window background) that can be applied to the user interface. Themes may be stored in configuration files for flexibility.
- **libs-gui**
  The main graphical library in GNUstep, providing classes for windows, views, event handling, and high-level rendering. Integrates with the Theme Manager to propagate theme changes to on-screen elements.
- **libs-base**
  The foundation layer of GNUstep, containing core Objective-C classes (e.g., NSObject, NSString) and utilities (file I/O, concurrency). Used by the Theme Manager for reading and writing configuration data.
- **libs-back**
  A platform-specific rendering library that translates draw commands from libs-gui into OS-level calls. May also adjust window decorations or other low-level visuals when theme changes demand it.
- **Configuration File**
  A .plist or JSON file containing key-value pairs of appearance settings (colors, fonts, image paths). Dynamically loaded by the Theme Manager to apply user-selected or default themes.

- **NSWindow / NSView**
  Core user interface components in GNUstep, each subscribing to theme change notifications and refreshing its visual properties as instructed by the Theme Manager.
- **NSThemeDidChangeNotification**
  A broadcast event triggered by the Theme Manager when a new theme is applied or modified. Listeners (UI components) receive this message and update their appearance accordingly.
- **Preferences Panel**
  A user-facing settings window allowing customization of theme attributes. It calls the Theme Manager's APIs to store or apply changes immediately or at a later time.

7. Naming Conventions

- **Class and File Names**
  - GNUstep follows Apple's Cocoa convention of prefixing classes with NS (e.g., NSWindow, NSView).
  - For theming-related classes, a similar convention can be used (e.g., NSThemeManager).
  - Source files typically match the class name (e.g., NSThemeManager.m, NSThemeManager.h).
- **Method and Variable Names**
  - Use camelCase for Objective-C method names (applyCurrentTheme), with descriptive prefixes (set, get, apply) as needed.
  - Member variables or instance variables generally follow an underscore or leading lowercase pattern (e.g., _currentTheme, currentTheme).
- **Configuration Files**
  - Themes stored in .plist or JSON typically use key-value pairs with meaningful, hierarchical keys (e.g., Theme.BackgroundColor, Theme.DefaultFont).
  - File names often reflect the contained theme (e.g., DarkTheme.plist, CustomTheme.json).

8. Lessons Learned

- **Significance of Modularity over Core Integration**
  By wrapping theme logic within a separate subsystem (either as a plugin or libs-gui extension), we restrict potential core instability. This strategy demonstrated that modular architecture not only promotes maintainability but also simplifies testing as well as future development (e.g., additional theme bundles).
- **Architectural Trade-Offs for Real-World Constraints**
  The current GNUstep codebase has developed through performance-driven "shortcuts" and partial coverage in libs-corebase. Though complicating a strictly layered approach,

such divergences also reflect pragmatic necessities in balancing ideal architecture against real-world performance and historical rationale. Our theme manager needs to be resilient enough to handle these divergences without introducing meaningful complexity.

- **Proper Use of Notifications and MVC**
  Relying on the existing event-driven approach (i.e., posting an NSThemeDidChangeNotification to all views interested) elegantly leverages Objective-C's dynamic dispatch. It helps to confirm well-established patterns, like broadcast notifications and the MVC split as still providing a good structure for large-scale changes, such as updating an entire UI at once.
- **Future-Proofing Through Clear Interfaces**
  Recording public APIs (i.e., how plugins would register custom themes) makes it easier for us to make GNUstep evolve without breaking existing apps. Similarly, reading and writing theme data in a human-readable format (.plist or JSON) makes it possible for newcomers and power users to define or edit themes without having to rebuild the system.
- **Value of Incremental Updates**
  Adding functionality piecemeal, beginning with simple color and font customizations, then moving forward toward more control over rendering, lessens the danger of gigantic regressions. It is feasible to test each step along the way against current GNUstep programs for stability and enhancing user experience.

The conclusions are that strong architectural designs have to be both principled (maintaining the layering as separate and decoupled) and pragmatic (capable of supporting historical weaknesses or performance-sensitive direct calls). By emphasizing a modular, notification-based strategy, the Dynamic Theme Manager enhances contemporary UI requirements without compromising GNUstep's signature portability and layered architecture.

9. Conclusion

With the provision of a powerful Dynamic Theme Manager, GNUstep is able to address modern UI demands without compromising maintainability and portability. Theming logic centralized in a self-contained subsystem makes updating and testing easier and allows consistent user experience with on-the-fly appearance changeover. Architectural strategy, core-integrated or plugin-based, illustrates how new features can expand GNUstep's patterns already established (e.g., notifications, MVC) without accumulating excessive technical debt. Down the line, incremental steps like more possibilities for customization or coordination with system-wide themes can branch out from here, making GNUstep adaptable and friendly to use within modern computing environments.

10. References

1. **GNUstep Project — Developer Documentation**
   Retrieved from https://www.gnustep.org/developers/documentation.html
2. **GNUstep Project — User Experience Documentation**
   Retrieved from https://www.gnustep.org/experience/documentation.html
3. **GNUstep Official GitHub Repositories**
   Retrieved from https://github.com/gnustep/
4. **Wikipedia Contributors. (2025, March 9). GNUstep.**
   In *Wikipedia, The Free Encyclopedia*. Retrieved from https://en.wikipedia.org/wiki/GNUstep