CISC 322 – Assignment 1

Conceptual Architecture of GNUstep

February 14th, 2025

Group 10:

Chuka Uwefoh – 22khl2@queensu.ca

Lore Chiepe – 21lpc2@queensu.ca

Henry Chen – 19hc38@queensu.ca

James Wang – 22jw16@queensu.ca

Ryan Jokhu – 21raaj@queensu.ca

Zachary Stephens – 20zs46@queensu.ca

Table of Contents

Abstract

This report attempts to explain the conceptual architecture of the application GNUstep, a development environment. Through research in the GNUstep documentation and its wiki as well as other sources it was determined that GNUstep operates using a layered style.

GNUstep is an open-source development environment that implements Apple's Cocoa (formerly known as OpenStep) APIs, it main feature is that it enables developers working with it to create cross-platform applications with a minimal amount of code that is specific to said platform. This report will present a comprehensive analysis of GNUstep's conceptual architecture, covering its design principles, component interactions, as well as some other notable factors like programming convention and primary use cases.

The framework's core libraries; libs-base, libs-corebase, libs-gui, and libs-back work together to form a hierarchical structure that focuses on maintainability and extensibility. This architectural approach enables independent development of components while preserving overall system stability. This report will feature GNUstep's concurrency model, implementing Grand Central Dispatch (GCD) and traditional threading mechanisms, which facilitates responsive user interfaces and efficient background processing. The system's build infrastructure,

Introduction and Overview

Purpose

This comprehensive architectural report aims to explore and summarize GNUstep's internal structure and design principles into a 10 to 15 page document that can be understood by developers easily. This report focuses on the framework's conceptual architecture using pre-existing documentation rather than implementational details based on the source code and aims to provide details on how GNUstep achieves its goals of cross-platform compatibility and developer productivity.

Historical Context and Development of GNUstep

In 1997, GNUstep was written based on the OpenStep specification, originally developed by NexT Computer (later acquired by Apple). Being based on OpenStep would influence its architectural decisions, particularly its emphasis on object-oriented design and composition. Over time, GNUstep was gradually developed to maintain compatibility with Apple's Cocoa framework while also developing its own identity as a cross-platform development environment.

Throughout the years, the framework's development history spanned multiple computing paradigm shifts, from single-threaded desktop applications all the way to modern, concurrent, distributed systems. Throughout these changes, GNUstep's architecture continuously changed and accommodated new features and capabilities while maintaining backward compatibility.

Report Organization

This report is structured to provide a systematic exploration and explanation of GNUstep's architecture, while also covering auxiliary information that pertains to understanding the architecture itself, including diagrams, dictionaries, use cases, and so forth.

1. Abstract
2. Introduction and Overview
3. Architecture: Examines the major components and their interactions, focusing on the layered structure.
    i. Component Analysis: Details each major subsystem's responsibilities, interfaces, and evolution patterns:
        - Core Libraries
        - GUI Framework
        - Development Tools
        - Cross-Platform Layer
    ii. Subsystem Interaction and Workflow
    iii. Use Cases
    iv. Evolution
    v. Concurrency
4. External Interfaces
5. Dictionary
6. Conclusion

Derivation Process

To determine the architecture of GNUstep, we relied on the course-provided sources. We felt that these resources were sufficient in information, so other resources were not necessary. After constructing a textual description of the conceptual architecture, we created diagrams to visually represent the component connections, helping us better understand the system's structure and interactions.

Architecture

GNU step is an open-source, object-oriented, cross-platform development environment that is compatible with Apple's Cocoa framework. Its conceptual architecture is designed to provide a robust and flexible environment for developing both non-graphic tools and graphical applications.

3.0.1 Core System Components

GNUstep is built around a modular architecture that separates its core functionality into distinct layers, enabling developers to create advanced GUI desktop applications and server applications that are portable across multiple platforms. The architecture is divided into several key subsystems, each serving a specific purpose in the development process.

GNUstep's architecture can be divided into several key subsystems:

Core Libraries

libs-base: This is the core library for non-graphic applications and serves as the foundation of the GNUstep framework. It provides foundational classes for handling strings, collections, file I/O, and networking, making it essential for developing command-line tools and other non-graphic applications. GNUstep-base includes key Objective-C classes such as NSString, NSArray, NSDictionary, and NSFileManager, which are used for data manipulation, storage, and system interactions. This library is designed to be lightweight and efficient, ensuring that developers have access to robust and reliable tools for building applications. GNUstep-base also plays a critical role in supporting graphical applications by providing the underlying data structures and utilities that libs-gui relies on.

libs-corebase: This library extends the functionality of libs-base by providing additional core utilities and abstractions. It is designed to support more advanced features and optimizations, making it a critical component for both non-graphic and graphical applications. libs-corebase works closely with libs-base to provide a comprehensive foundation for application development.

Graphical Development Framework

libs-gui: This library is the graphical counterpart to libs-base and libs-corebase. It provides classes for creating windows, menus, buttons, and other UI elements,

enabling developers to build rich graphical applications. libs-gui handles event management, drawing, and printing, making it the backbone of GUI development in GNUstep. It is designed to be compatible with Apple's Cocoa API, ensuring that applications can be easily ported between GNUstep and macOS environments.

libs-back: This library serves as the backend for graphical rendering, abstracting the underlying platform-specific graphics and windowing systems. It ensures that libs-gui can function seamlessly across different operating systems, such as GNU/Linux, Windows, and UNIX-based systems. libs-back plays a crucial role in maintaining GNUstep's cross-platform compatibility.

Development Tools:

Gorm is a visual GUI builder that allows developers to design user interfaces visually. It generates Objective-C code that can be integrated into applications, streamlining the process of creating complex UIs. Gorm works closely with libs-gui to ensure that the generated code is compatible with GNUstep's graphical framework. It is an essential tool for developers who want to create sophisticated desktop applications with minimal manual coding.

GNUstep-make is the build system for GNUstep applications. It automates tasks such as dependency resolution, linking, and installation, ensuring that the necessary libraries and resources are correctly integrated into the final application. GNUstep-make interacts with libs-base, libs-corebase, and libs-gui to compile and link applications, making it a critical component for both non-graphic and graphical development. It also integrates with development tools like Gorm and ProjectCenter to provide a seamless build process.

Cross-Platform Layer:

Cross-Platform Compatibility: GNUstep's cross-platform layer ensures that applications can run on multiple operating systems without requiring significant modifications. This layer abstracts away platform-specific details, such as file paths, windowing systems, and graphics rendering, allowing developers to write portable code. It works in conjunction with libs-back and libs-gui to provide a consistent development experience across different platforms.

3.0.2 Subsystem Interaction and Workflow

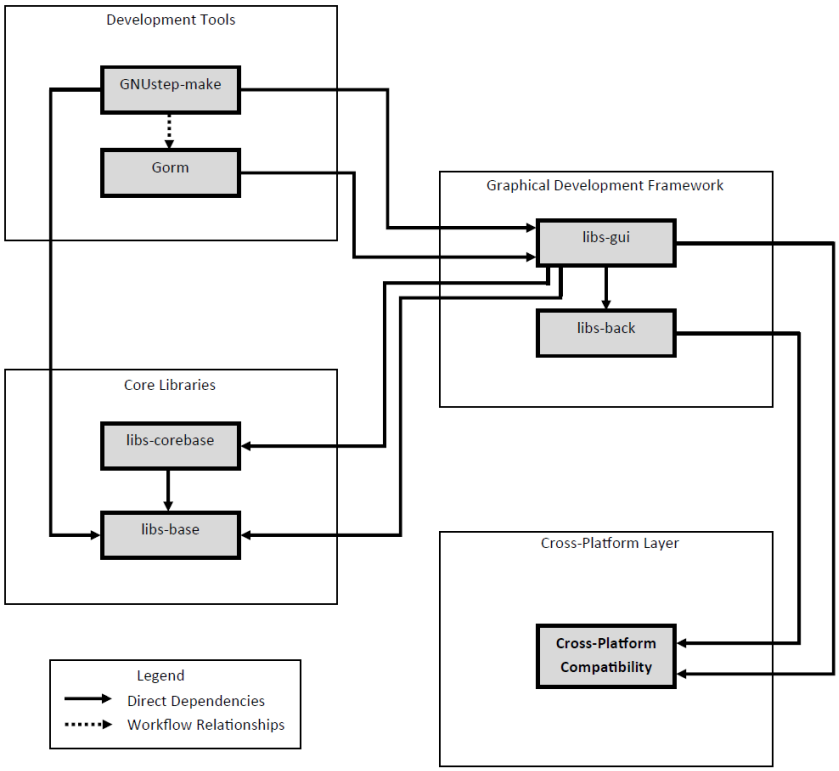

Figure 1: Software Architecture of GNUstep

The core components of GNUstep—libs-back, libs-base, libs-corebase, libs-gui, and Gorm—work together to create a cohesive system for developing and running both non-graphic and graphical applications. Their interactions ensure seamless functionality and cross-platform compatibility.

libs-base provides the foundational functionality for all applications, handling tasks like data manipulation, file I/O, and networking. In graphical applications, libs-base works with libs-gui to process data for display. libs-corebase extends libs-base with additional utilities and optimizations, enhancing the system's capabilities for complex tasks.

libs-gui builds on libs-base and libs-corebase to create windows, menus, and other UI elements. It relies on libs-back for platform-specific rendering and window management. libs-back abstracts the underlying graphics systems, ensuring libs-gui works across different operating systems. For example, when libs-gui renders a window, libs-back translates the request into platform-specific calls (e.g., X11 on Linux or Win32 on Windows).

Gorm, the visual GUI builder, simplifies UI development by generating Objective-C code for libs-gui. Developers design interfaces in Gorm, and the generated code is integrated into

the application. GNUstep-make then compiles and links the application, ensuring all components work together. This interaction between Gorm, libs-gui, and GNUstep-make streamlines the development process.

Finally, the cross-platform layer ensures applications run on multiple operating systems by abstracting platform-specific details. It works with libs-back and libs-gui to provide a consistent experience across platforms.

In summary, libs-base and libs-corebase provide the foundation, libs-gui and libs-back enable graphical rendering, Gorm simplifies UI design, and GNUstep-make automates the build process. Together, these components create a robust, portable development environment.

3.0.3 Use cases

Use Case 1

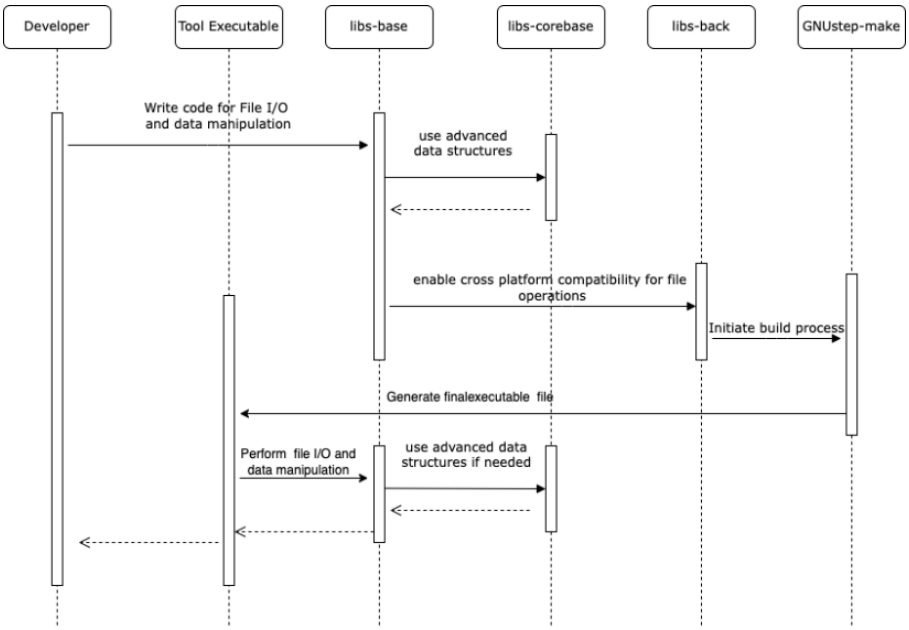

Figure 2: Command Line Application Use Case

In this use case, a developer creates a cross-platform command-line tool that reads data from a file, processes it, and outputs the results to another file. The developer uses libs-base for file I/O and data manipulation, libs-corebase for advanced data structures and optimizations, and libs-back to ensure cross-platform compatibility for file operations. GNUstep-make compiles and links the tool with these libraries, generating the final executable. During testing, the tool provides feedback (e.g., logs or errors) to the developer, ensuring it works correctly on different platforms. Once verified, the tool is deployed in a

production environment, where it performs file operations and data processing as designed, leveraging libs-base, libs-corebase, and libs-back to maintain functionality and portability.

Use Case 2

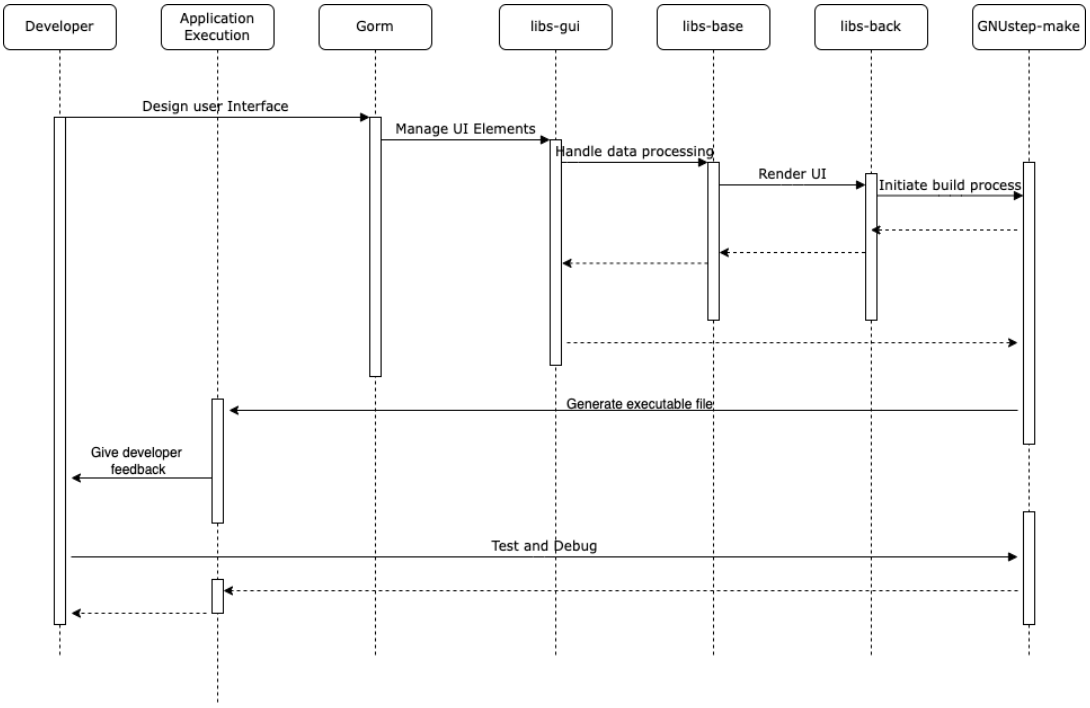

Figure 3: GUI Application Use Case

In this use case, a developer creates a cross-platform graphical desktop application for managing a to-do list. The developer designs the UI using Gorm, which generates Objective-C code for the interface. libs-gui handles the graphical elements and user interactions, while libs-base manages the underlying data processing. libs-back ensures the UI renders correctly across different platforms by abstracting platform-specific graphics and windowing systems. GNUstep-make compiles and links the application with these libraries, generating the final executable. After testing on multiple platforms, the application is deployed, allowing users to interact with the to-do list seamlessly.

3.0.4 Evolution

The evolution of GNUstep focuses not only on improving its performance but also on increasing its accessibility across various platforms. By refining core libraries, such as `libs-base` and `libs-gui`, the system continues to maintain compatibility with modern development practices. One of the critical areas of improvement has been in enabling

better integration of graphical and functional user interface elements, ensuring developers have a robust framework for both UI and logic implementation.

Concurrency and thread management are also key elements in GNUstep's evolution. The incorporation of Grand Central Dispatch (GCD) allows for efficient multi-threading, enabling applications to scale effectively and perform resource-intensive tasks asynchronously. This progress in concurrency models ensures that GNUstep remains relevant in a competitive landscape, where performance under load is crucial.

Additionally, the developer tools that accompany GNUstep, such as ProjectCenter and Gorm, continue to evolve alongside the system. These tools help streamline the development process, making it easier for developers to create complex applications within the GNUstep ecosystem. ProjectCenter, for example, offers an integrated development environment (IDE) that assists in project management, while Gorm facilitates rapid GUI design. Both tools are updated with each release to enhance usability and incorporate newer features that improve workflow.

The adaptability of the framework is also a priority. GNUstep is continually refining its support for multiple operating systems, ensuring that applications built using the framework can run seamlessly on various platforms. As more devices and environments emerge, the ability to port and deploy applications efficiently across platforms becomes even more significant. This cross-platform functionality underpins the framework's ability to cater to a global developer audience.

Lastly, the broader GNUstep community plays a critical role in the evolution of the system. Ongoing collaboration among developers and contributors helps prioritize new features and bug fixes, allowing the system to evolve dynamically. Community-driven development means that feedback from users and developers is often incorporated into each release, ensuring that the system adapts quickly to new challenges in the software development space.

3.0.5 Concurrency

Concurrency is a critical aspect of modern software systems, especially for applications that require real-time performance or need to handle multiple tasks simultaneously. GNUstep, being a framework for developing advanced GUI desktop applications and server applications, incorporates concurrency mechanisms to ensure efficient and responsive performance. The framework supports multi-threading through POSIX threads (pthreads) and Objective-C threading APIs, allowing developers to create and manage threads for tasks such as handling user input, processing data, or performing background

computations. For example, libs-base provides utilities like NSThread, which enables developers to offload time-consuming tasks (e.g., file processing) to background threads, ensuring the main thread remains responsive to user interactions.

In addition to traditional threading, GNUstep implements Grand Central Dispatch (GCD), a concurrency model that simplifies the management of concurrent tasks. GCD allows developers to define tasks as blocks of code and dispatch them to queues, which are managed by the system. libs-corebase provides support for GCD, enabling developers to easily parallelize tasks without manually managing threads. For instance, a developer might use GCD to dispatch multiple tasks, such as downloading files or processing data, to concurrent queues, improving the application's performance. This approach is particularly useful for tasks that can be broken into smaller, independent units of work.

In graphical applications, the main thread plays a crucial role in handling user interface updates and event processing. To ensure a responsive UI, long-running tasks must be offloaded to background threads. libs-gui manages the main event loop, which processes user input (e.g., mouse clicks, keyboard input) and updates the UI. Developers must ensure that UI updates are performed on the main thread to avoid rendering issues. For example, if a task like loading data from a file is performed on a background thread, the results must be dispatched back to the main thread to update the UI. This separation of responsibilities ensures that the application remains responsive and provides a smooth user experience.

GNUstep also supports asynchronous operations, which are essential for tasks such as network requests or file I/O that can take a significant amount of time. libs-base provides classes like NSOperation and NSOperationQueue for managing asynchronous tasks. Developers can define tasks as operations and add them to a queue, which executes them concurrently. For instance, a developer might use NSOperationQueue to download multiple files in the background while keeping the UI responsive. This approach allows the application to perform complex tasks without blocking the main thread, ensuring that the user interface remains interactive.

Finally, GNUstep emphasizes thread safety, ensuring that its core components can be used safely in multi-threaded environments. Developers must be cautious when accessing shared resources (e.g., data structures) from multiple threads to avoid race conditions. libs-base provides thread-safe classes and synchronization mechanisms, such as NSLock and NSRecursiveLock, to help developers manage shared resources. For example, if multiple threads need to access a shared data structure, such as a to-do list, the developer can use NSLock to ensure that only one thread accesses the data at a time. This focus on thread safety ensures that GNUstep applications can handle concurrency effectively while maintaining stability and reliability.

Use Case 3

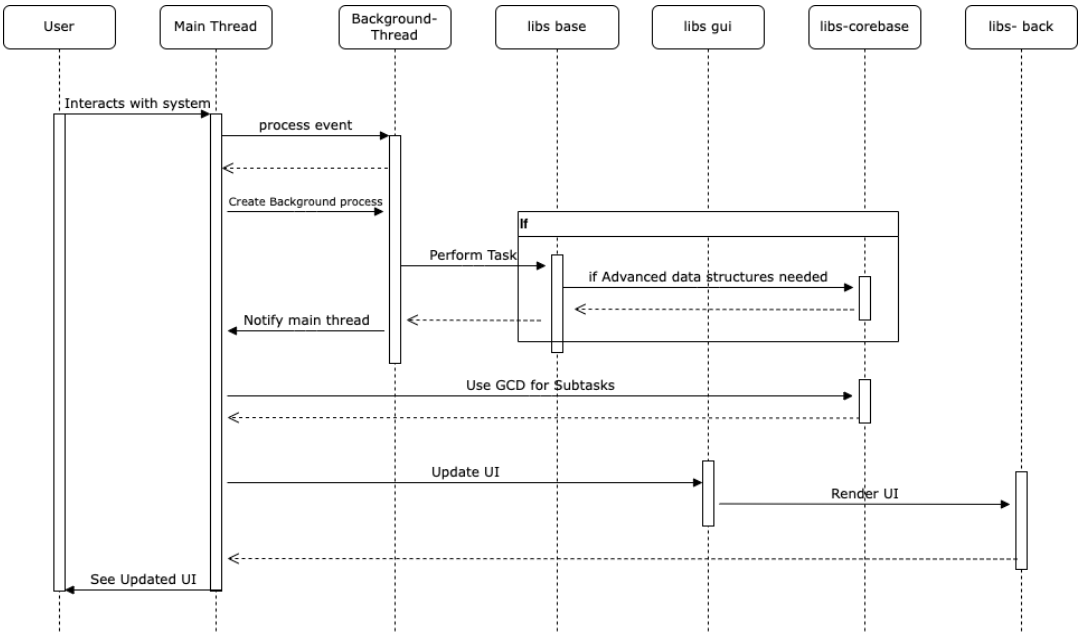

Figure 3: Concurrency Use Case Diagram

This use case demonstrates how GNUstep manages concurrency to maintain a responsive user experience. When the user initiates a task (e.g., downloading a file), the Main Thread processes the input and delegates the task to a Background Thread using libs-base. The Background Thread performs the task, optionally using GCD (via libs-corebase) for parallel subtasks. Once the task is complete, the Background Thread notifies the Main Thread, which updates the UI using libs-gui. Finally, libs-back renders the updated UI on the screen, ensuring platform compatibility. This workflow ensures that time-consuming tasks do not block the UI, keeping the application responsive and providing real-time feedback to the user.

External Interfaces

GNUstep interacts with external systems and platforms to enable seamless application functionality. Below is a concise breakdown of the information transmitted to/from the system through its key external interfaces:

1. File System Interaction
    - Information Transmitted:
        i. File data (e.g., configuration files, logs, user data).
        ii. File paths and metadata (e.g., file size, permissions).
    - Purpose:

        i. Read, write, and manage files to ensure data persistence and accessibility.

2. Network Communication
    - Information Transmitted:
        i. Data packets (e.g., HTTP requests/responses, file downloads).
        ii. Network status (e.g., connection success/failure).
    - Purpose:
        - Enable communication with remote servers for data exchange and real-time updates.
3. Graphical Environment Interaction
    a. Information Transmitted:
        i. User input (e.g., mouse clicks, keyboard input).
        ii. UI state (e.g., window positions, button states).
    b. Purpose:
        i. Render graphical elements and handle user interactions.
4. Hardware Device Interaction
    a. Information Transmitted:
        i. Input events (e.g., mouse movements, keyboard presses).
        ii. Device status (e.g., printer availability, scanner readiness).
    b. Purpose:
        i. Capture user input and interact with hardware devices.
5. Inter-Process Communication (IPC)
    a. Information Transmitted:
        - Messages and notifications between processes.
        - Shared data (e.g., configuration settings, task status).
    b. Purpose:
        i. Enable communication between applications or processes running on the same system.

Data Dictionary:

- API- Application Programming Interface
- IPC – Inter Process Communication
- GCD - Grand Central Dispatch
- Main Thread - primary thread responsible for handling UI updates and event processing.

- Background Thread - secondary thread used to perform time-consuming tasks without blocking the main thread.
- Class Names:
    - NSFileManager (class for file operations).
    - NSWindow (class for creating windows).

Conclusions

The architectural design of GNUstep is in a layered style of modular components that can communicate with each other. This allows each component to be updated and new ones to be added without much trouble, which is especially important for open source projects that typically struggle to find volunteers to deal with integration of new and old components.

GNUstep has 4 main components, a number of secondary optional components, as well as some external connections. Communication between these components is handled with both asynchronous requests and multi-threading through POSIX and Objective-C.

GNUstep is currently still under development by the open source community with the last major release being released on February 12 of 2025. The current roadmap has the focus of the development to be allowing GNUstep to be more usable on Windows as well as general maintenance to ensure GNUstep continues to work with new updates to a variety of operating systems.

Lessons Learned

Analyzing the architecture of GNUstep, was challenging due to the complexity of the system. While GNUstep is well-documented, understanding its architecture required extensive research into its subsystems through a large number of sources. Starting the project earlier would have given us more room to further explore these resources.

Visual tools such as sequence diagrams and component diagrams played a significant role in visualizing, keeping track of, and understanding GNUstep's interactions. By modeling key processes, such as a developer creating a type of tool or app, we were able to refine our understanding of how data flows within the system.

Overall, this process demonstrated importance of architectural analysis in open-source projects. Unlike proprietary systems, GNUstep has evolved through community contributions. This means that it is essential for contributors to examine historical design choices and their impact on maintainability and future additions.

References

[1] Wikipedia contributors, "GNUstep," *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/GNUstep. [Accessed: Feb. 14, 2025].

[2] GNUstep Project, "GNUstep Official Website," GitHub Pages. [Online]. Available: https://gnustep.github.io/. [Accessed: Feb. 14, 2025].

[3] GNUstep MediaWiki, "Main Page," GNUstep Wiki. [Online]. Available: https://mediawiki.gnustep.org/index.php/Main_Page. [Accessed: Feb. 14, 2025].

[4] GNUstep Project, "Developer Documentation," GNUstep. [Online]. Available: https://www.gnustep.org/developers/documentation.html. [Accessed: Feb. 14, 2025].

[5] GNUstep Project, "Developer Tools," GNUstep. [Online]. Available: https://www.gnustep.org/experience/DeveloperTools.html. [Accessed: Feb. 14, 2025].

[6] GNUstep Project, "Project Center," GNUstep. [Online]. Available: https://www.gnustep.org/experience/ProjectCenter.html. [Accessed: Feb. 14, 2025].

[7] GNUstep Project, *Gorm Manual*, PDF document. [Online]. Available: https://www.gnustep.org/resources/documentation/Gorm.pdf. [Accessed: Feb. 14, 2025].

[8] A. D. Duchowski, *GNUStep Manual*, Clemson University, PDF document. [Online]. Available: http://andrewd.ces.clemson.edu/courses/cpsc102/notes/GNUStep-manual.pdf. [Accessed: Feb. 14, 2025].

[9] NeXTSTEP/OpenStep, *OpenStep Development Tools*, PDF document. [Online]. Available: https://cdn.preterhuman.net/texts/computing/nextstep-openstep/802-2110%20-%20OpenStep%20Development%20Tools.pdf. [Accessed: Feb. 14, 2025].

[10] GNUstep Project, *OpenStep User Interface Guidelines*, PDF document. [Online]. Available: https://www.gnustep.org/resources/documentation/OpenStepUserInterfaceGuidelines.pdf. [Accessed: Feb. 14, 2025].

[11] GNUstep MediaWiki, "OpenStep," GNUstep Wiki. [Online]. Available: https://mediawiki.gnustep.org/index.php/OpenStep. [Accessed: Feb. 14, 2025]..