Conceptual Architecture Report: GNUstep

Queen's University
CISC 322
Instructor: Prof. Bram Adams
Winter 2025

Cameron Jenkins 20358084 21cj32@queensu.ca
Christine Ye 20354325 21cy27@queensu.ca
Evelyn Lee 20334769 21esl5@queensu.ca
Jihyeon Park 20100373 17jp53@queensu.ca
Liam Beenken 20364179 22ltb1@queensu.ca

Table of contents

1. Abstract

In this paper, we researched and analyzed GNUstep, an open-source implementation of Cocoa frameworks. Our goal is to understand GNUstep's functionality, software architecture, system components, and compatibility with modern development environments. We examine its object-oriented framework, cross-platform capabilities, and programming language support like Objective-C, Java, Ruby, etc. To gain a better understanding, we explored official documentation, and studied real-world applications using GNUstep. Through this research, we outline the architectural structure of GNUstep, key modules and ultimately its role in modern software development.

2. Introduction

As mentioned before, GNUstep is a free and open-source implementation of the Cocoa (formerly OpenStep) Objective-C frameworks, widget toolkit, and application development tools for Unix-like operating systems and Microsoft Windows. As part of the GNU Project, it provides a cross-platform, object-oriented development environment with an integrated IDE. The project is valuable for developers who seek an open-source environment compatible with macOS development practices, enabling cross-platform software creation in addition to its default Objective-C interface, GNUstep supports bindings for Java, Ruby, GNU Guile, and Scheme, enhancing its versatility. GNUstep and Cocoa have the common origin in NeXTSTEP and OpenStep, which were the software frameworks created by NeXT (founded by Steve Job after Apple). However, GNUstep was developed first by GNU contributors to replicate NeXTSTEP's features, while Cocoa emerged later when Apple adapted NeXT's technology for macOS. Moreover, designed for portability and extensibility, GNUstep continues to serve as an independent and powerful tool for cross-platform software development, particularly for applications requiring OpenStep/Cocoa-like environments outside of macOS.

This paper primarily focuses on the architecture of the GNUstep and its key components. We will dive more into detail of its modular framework, core libraries, and development tools in greater detail, examining how they interact to create a cohesive development environment. Additionally, we will analyze its evolution, compatibility with Cocoa, concurrency management, and real-world applications. Through diagrams and structured discussions, this report aims to provide a comprehensive understanding of GNUstep's design and its significance in cross-platform software development.

Architecture

3. **System Functionality**

GNUstep is a free, object-oriented, cross-platform development framework that closely follows Apple's Cocoa APIs. It gives developers the ability to create advanced graphical user interface (GUI) desktop applications and server applications across a variety of platforms. GNUstep is structured into several interacting components, each serving a specific role in the framework

Development Tools:

ProjectCenter

The official integrated development environment (IDE) for the GNUstep framework. It allows developers to create and manage a variety of projects such as tools, libraries, and applications. ProjectCenter automatically generates the necessary makefiles to simplify the editing, compilation, packaging and debugging of your programs. With its intuitive graphical user interface, this tool allows developers to easily start new projects and manage their project files efficiently.

Gorm (Graphical Object Relationship Modeller)

An easy-to-use tool that helps developers design the graphical user interfaces of their applications. This tool allows developers to create, arrange, and link interface elements without the need for manual coding. For example, developers can use the drag-and-drop elements such as buttons and sliders and link them to functions and variables, allowing them to focus on the application’s core functionality.

Components:

GNUstep Base Library

A set of basic, non-graphical Objective-C classes that provide crucial functionality for applications. Along with other essential tools, it supports managing strings, collections, byte streams, and notifications. This library makes sure that data management and system interactions run well. It was initially based on the OpenStep standard and mimics the non-graphical features of Apple's Cocoa frameworks, giving additional components a strong base upon which to grow.

GNUstep CoreBase Library

This component offers basic data structures and utilities provided by the non-graphical C-based objects including strings, collections, byte streams, notifications, and event loops. For applications that require a lower-level alternative for Objective-C, the library acts as an implementation of Apple's CoreFoundation framework, guaranteeing compatibility and offering essential data-handling features.

GNUstep GUI Library:

The GUI Library contains graphical user interface elements written in Objective-C, based on Apple's Cocoa framework and the OpenStep specification. It provides essential graphical objects such as buttons, text fields, and windows, along with support for handling events, fonts, colors, and images.

GNUstep GUI Backend:

A component made to manage tasks that depend on the display system. The GUI library is divided into a back-end that controls specific calls to systems such as the X Window System and a platform-independent front-end. Without changing the application code, this design enables GNUstep applications to mimic the appearance and feel of the underlying display system.

Interactions

GNUstep's various components work closely together to make development easier and more efficient. The Base Library provides essential features that applications and the GUI system use to handle data and communicate with the system. The GUI Library depends on the Base Library for basic services and works with the Backend to display the graphical interface on different operating systems. The Backend acts as a connection between the GUI system and the computer's windowing system, ensuring that everything appears correctly on different platforms. Finally, tools like Gorm and ProjectCenter help developers by making it easier to design interfaces, organize code, and test applications. Source code, resources, and project structure are all organized in ProjectCenter while Gorm is used to design and refine the GUI elements of the program. The .gorm files which include the GUI definitions and object relationships, are added into the ProjectCenter project after the interface design is completed. During the build process, these files are loaded by the application at runtime so that the user interface appears as designed.

GNUstep allows information to be transmitted and received through different external interfaces. GUIs are the most popular method by which users interface with the system, which allow them to input data using buttons, text fields, and other controls while also displaying output like text, images, and notifications. Additionally, the system is file-based, enabling applications to read and write data for purposes including document storage, configuration loading, and data exchange between programs. GNUstep also supports networking through its Base Library, which contains utilities for managing internet connectivity. Similar to how macOS apps use Apple's Foundation architecture, applications can transmit and receive data over networks utilizing protocols like HTTP and TCP/IP. This makes it possible for GNUstep programs to communicate with distant

servers, get data, and connect to online services. The system also supports distributed objects, which enables network communication between several applications. Overall, these components interact to form a single development environment that effectively incorporates data transmission between applications, the operating system, and outside resources, project management, interface design, and application functionality.

4. **Evolution**

Over the years, GNUstep has evolved through continuous adaptation to new hardware, software architectures, and user requirements, driven by technological advancements and its role in the open-source ecosystem. Its evolution is reflected in various enhancements across versions, maintaining adherence to the OpenStep specification while adapting to modern computing environments and ensuring compatibility with contemporary operating systems.

GNUstep history began back in the early 1990s when NeXTSTEP was developed. NeXTSTEP is an operating system developed by NeXT and with an object-oriented programming environment based on Objective-C, it introduced the AppKit and Foundation frameworks. NeXT then partnered with Sun Microsystems to create OpenStep which is an standardized API based on NeXTSTEP that could be implemented on multiple platforms. OpenStep laid the foundation for GNUstep and it defined how graphical applications should be structured, making it an important foundation for future development. We discussed NeXTSTEP and OpenStep because GNUstep would not exist without OpenStep since it was created to be an open-source reimplementation of that API. GNUstep was created to implement OpenStep APIs, allowing developers to write applications with OpenStep-like functionality on Unix-like systems and windows. While Apple used OpenStep as the foundation for Cocoa (Apple's native framework for building macOS applications) in macOS, the GNU Project planned to create an independent, cross-platform, open-source alternative.

In its early stages, GNUstep was primarily built to copy OpenStep but it did not fully support newer Apple technologies like Cocoa or modern GUIs. In other words, GNUstep could run older OpenStep-like applications but was not fully compatible with newer macOS apps or features. Consequently, as GNUstep evolved, developers worked to enhance compatibility with Cocoa by implementing newer APIs and modernizing its graphical user interface components. Over time, GNUstep incorporated features from Apple's macOS frameworks, while still maintaining cross-platform functionality. Efforts were made to improve Objective-C support, such as Objective-C 2.0, which included garbage collection (later replaced by ARC), properties, and fast enumeration. It also refined the AppKit framework and introduced tools that made GNUstep more practical

modern development, more specifically the GUI toolkit which was updated to better match Cocoa's look and feel. This included improvements to the theme and UI components to resemble macOS.

To expand on the implementation of core frameworks in the early development, GNUstep focused on implementing GNUstep Base and GNUstep GUI. GNUstep base is a framework that is similar to Apple's Foundation framework which provides essential functionalities such as memory management, file handling and data structures. Furthermore, GNUstep GUI is a framework that is equivalent to Apple's Appkit, responsible for rendering graphical interfaces, handling user input, and managing windows and UI components. These frameworks helped applications that were developed using OpenStep APIs to function within the GNUstep environment, allowing legacy NeXTSTEP and OpenStep applications to be reused.

During the mid 2000s to 2010s, GNUstep became more compatible with different operating systems like Linux, BSD-based systems and Windows with some limitations. To improve how applications looked and performed, developers enhanced the graphics systems by adding Cairo which is a graphics library that made UI rendering smoother and improved the appearance of vector-based graphics. This ultimately made GNUstep applications look better and feel more responsive. GNUstep also updated the Objective-C runtime during this time. It adopted improvements from Apple's Objective-C updates, allowing it to work better than modern compilers like Clang which allowed developers to use newer programming features and ensured better performance across different systems.

Since the 2010s to present time, GNUstep has focused on improving compatibility with Apple's Cocoa framework and keeping up with evolving technologies. Developers have added more Cocoa features to make GNUstep resemble macOS, allowing macOS applications to run efficiently with minimal modifications. It now supports Automatic Reference Counting (ARC), blocks, and other modern Objective-C features, making development more seamless. Additionally, GNUstep has been updated to fully support 64-bit systems, ensuring compatibility with modern computers. Improvements in package management and development tools have also made installation and usage more straightforward for developers.

GNUstep is an open-source project, allowing developers from around the world to collaborate and improve it. This global contribution plays a crucial role in keeping GNUstep updated, secure, and relevant for various applications. Developers frequently update GNUstep to align with modern programming trends and security requirements. With contributors from different regions, bugs are quickly identified and fixed, enhancing

stability and reliability. As a result, GNUstep continues to expand its capabilities, supporting applications in software development, education, and productivity, broadening its potential uses.

5. Control and Data Flow

Derivation Process:

The information and analysis in the following section was acquired through researching the official GNUstep documentation and experimenting with the software components on a personal machine. These methods allowed for a wide scope on the GNUstep project, providing information on implementation details, intended use cases, overall architecture, and each component's role in the application building process.

Findings:

Development with GNUstep begins at the ProjectCenter, which provides an IDE and tools to manage existing projects. ProjectCenter accesses project files from the user's local filesystem, and performs operations on the data such as sorting, highlighting, and recommendation. With this connection, the user can either open and edit past projects, or create a new one. The connection to the filesystem persists while the integrated IDE is in use, as reads and writes will occur frequently. ProjectCenter is also dependent on both the base, and GUI libraries. When the program is run, control will switch to the GUI library to call the functionality used for placing the UI components of the application. The GUI library itself will call the backend library to determine the rendering details on the given machine. The backend library views the UI elements in the project and inspects the environment the app is attempting to run in, accessing data from the machine. Using this data, it can create the proper rendering sequence and make any necessary modifications. It will also communicate with the window manager of the machine it is running on, making system calls to display the application and its components. Here, control and data are passed between the operating system, and the GNUstep GUI backend. GUI backend also references base to use the higher level data structures it provides, so we have even more control switching when that functionality is used. Once the initial elements are rendered, control switches back to the ProjectCenter to await user input and handle actions. Since ProjectCenter's implementation also uses classes defined in gnustep-base, control will switch to base when we instantiate objects, or call on the library's functionality.

When starting a new project, the programmer will include the Foundation and AppKit modules in their project to access the higher level data structures and UI components that the base and GUI libraries provide. How those libraries are used then depends on the application being built. Projects that use an NSArray for example, would create an instance of the class, referencing the required functionality from the base library. Then, in compilation, the relevant code would be referenced in the executable, adding the NSArray functionality to your project. Due to Objective-C's dynamic linking capabilities, control will be switched to the base library at runtime, rather than directly linking it in the executable. The sequence diagrams at the bottom of this document detail control switching in various different use cases, such as maintaining an open source GNUstep project, and working with GORM.

Takeaway:

Given the interaction, control flow, and data flow of these components, we can observe elements of both Object-Oriented, and Layered architecture. The components call on each other for functionality, passing control and data through their messages, and the base/core-base/GUI libraries are built to facilitate code reuse, and abstract lower level concepts. These are all features and benefits of the Object Oriented Style. We also notice that much of the GNUstep project is built on top of existing components, in a hierarchical fashion. Core-base and base enable quicker, more enjoyable development for both end users and the GNUstep team. The GUI library, for example, is heavily dependent on the functionality of base, and it then provides functionality to higher level components like ProjectCenter. This pattern is consistent with the layered style.

6. Concurrency

In GNUstep synchronization between user and application is formed by the loop structure of application and automatic creation of autorelease. When the application is running, the program initially enters the waiting step for the user's command or any method of communication with the program. At the end of the loop, the program automatically creates and deletes autorelease so that the user's manual work is reduced. After autorelease events, the program comes back to waiting state.

For scheduling concurrent processes or tasks, NSLock and NSOperationQueue classes are used. These classes inherit from the root class NSObject in the GNUstep base library.

NSLock is used for synchronization of multiple threads created with NSThread. When a thread obtains a lock, the other threads must wait until the thread that has the lock does not hold the lock. The lock can only be locked once, and must be unlocked before a thread obtains it.

The NSOperationQueue is used for scheduling multiple NSOperation by dealing with dependencies and priorities. Depending on the environment, the operations can be executed concurrently.

7. Divisions of Responsibilities

Developing a complex software system like GNUstep requires formalized and organized roles for developers to ensure it becomes efficient, maintainable, and scalable. GNUstep is the open-source version of the OpenStep specification and includes many related components like primary frameworks, development tools, scripting interfaces, and theming options. Since the system is made up of different components, work must be divided among the contributors in such a way that teams can work independently but ensure the consistency of the project.

Modular Design and Team Structure

One of the most crucial elements of dividing tasks in GNUstep is making sure development teams are structured according to the modular design of the software. There are several main frameworks in GNUstep, such as:

- gnustep-base (Foundation)
- gnustep-gui (AppKit)

Each of these frameworks has diverse functions, and developers who code them must possess unique skills in areas such as:

- Memory management
- Event-driven programming
- User interfaces that become responsive

By assigning each framework to a unique team, the project will be easier to develop and maintain. Developers working on one component can concentrate solely on enhancing its functions without interruptions from other teams. This specialization allows teams to improve their work without needing to take into account all dimensions of the broader system, as long as they adhere to standard interfaces and architectural conventions.

Ensuring Coordination Between Teams

While modularity allows autonomy, it is also important to preserve coordination between teams to avoid integration problems. GNUstep must run perfectly on a variety of platforms, meaning core library developers must cooperate closely with those responsible for platform compatibility and system integration on an ongoing basis. If they did not communicate, changes in one part of the system could introduce unexpected regressions in others. To counter this, a group of senior developers is typically responsible for

keeping the whole system in balance. They keep an eye on important design choices so that changes in one part of the system don't cause problems in another part of the system.Good documentation is also extremely important for cooperation. By having clear coding standards and keeping proper API documentation, teams can minimize confusion and duplicated effort.

Open-Source Contributions and Developer Roles

The open-source nature of GNUstep affects the division of responsibilities because core developers as well as outside contributors work on it. To coordinate this fluid situation, the project uses formal contribution guidelines in which roles for different kinds of developers are specified. Code is reviewed and merged by core maintainers while new developers usually start by:

- Fixing small bugs
- Improving documentation
- Then moving on to more complex tasks

In this way, new developers can contribute to the project and take on responsibilities without interfering with work in progress. Feature proposals require much discussion within the community. Project maintainers or community managers manage these discussions and decide which contributions align with GNUstep's long-term vision.

Concurrency and Performance Optimization

Another crucial factor when working with responsibility management in GNUstep is concurrency and performance optimizations.The majority of GNUstep programs react to events and run in multiple threads, so there has to be diligent planning to make different parts of the system work together seamlessly. People who know about threading models and optimization of performance usually deal with such problems. They work hand in hand with both the Foundation and GUI groups to:

- Improve the way different tasks run simultaneously
- Make the system more responsive

Additionally, as the system grows, it may be necessary to implement new optimizations so that it keeps working efficiently with new hardware and software. This usually means working together with lead developers, system developers, and performance experts so that the software continues to be quick on different platforms.

Quality Assurance, Documentation, and User Support

Beyond development work that is technical in nature, keeping the project operational smoothly and keeping the community involved takes more people who focus on:

- Quality tests
- Developing guides
- Supporting users

Some developers test new functionality, find bugs, and ensure updates are within GNUstep's design principles. Documentation groups write detailed guides and references so new and experienced developers can understand the system. These documentation tasks are crucial for an open-source project like GNUstep. New contributors might not be able to communicate directly with veteran developers, so they must rely on written sources to learn in a short time. Additionally, because GNUstep is an open-source project, the onus of managing user feedback and external contributions typically falls on community managers, who:

- Moderate discussions on mailing lists
- Manage feature requests
- Recruit new developers to the project

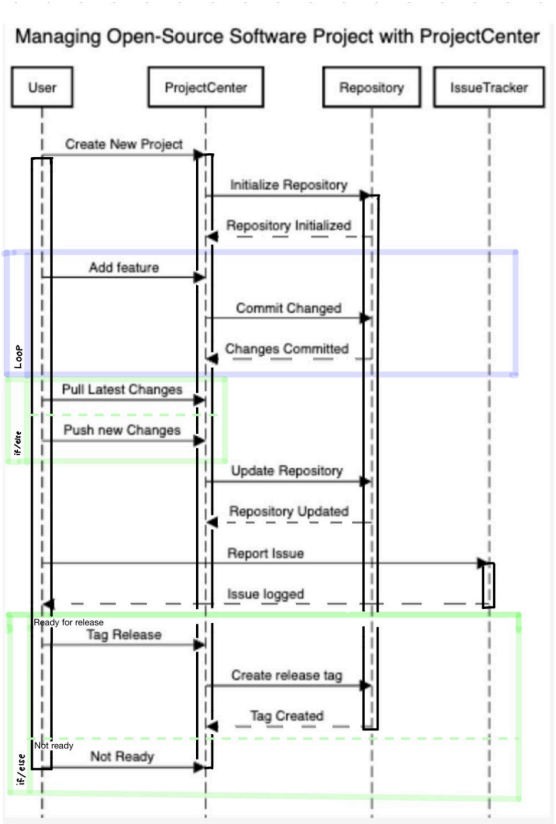
Maintaining Compatibility with OpenStep and macOS APIs

Another key task is maintaining compatibility with existing OpenStep applications and macOS APIs. GNUstep wants to create a free OpenStep alternative, so it needs to keep its APIs in sync with macOS's framework. This requires careful work by API compatibility and regression testing specialists, who:

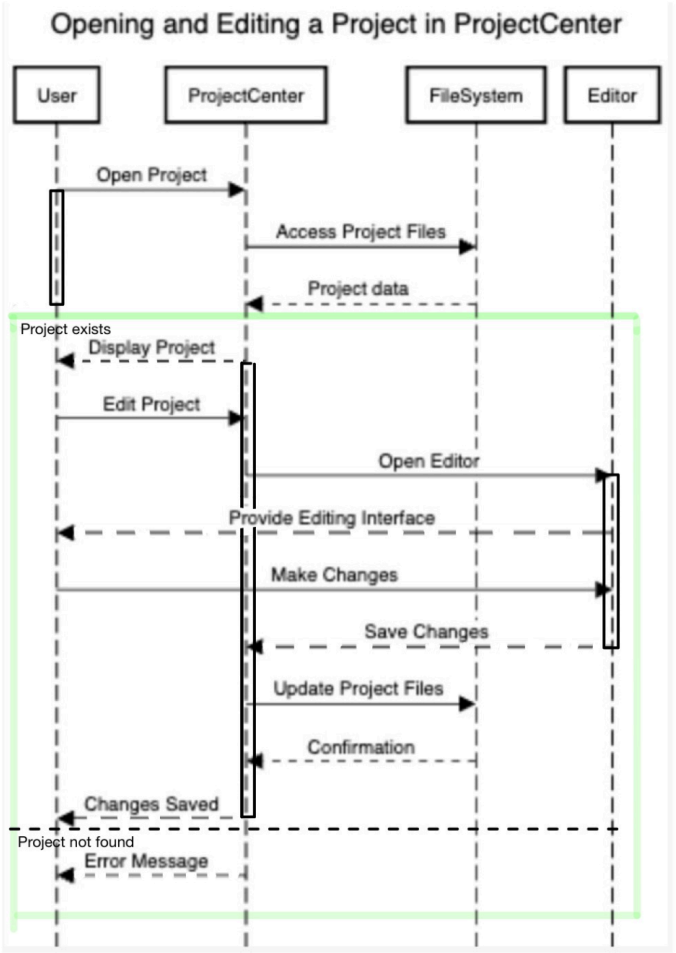
- Review Apple's changes
- Compare them with GNUstep implementations
- Figure out whether and how to add similar features

Sometimes this involves creative problem-solving to bring macOS features into the open-source GNUstep environment.

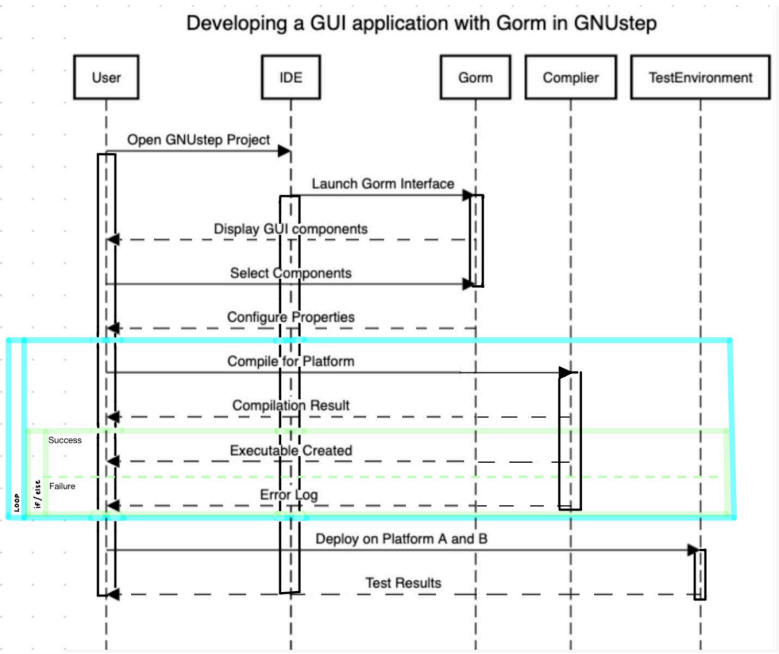
8. Sequence Diagrams



↑Figure 1.1 A sequence diagram illustrating the management of an open-source software project with ProjectCenter. It outlines interactions between the user, ProjectCenter, the repository, and the issue tracker, covering repository initialization, feature addition, version control updates, and release tagging. The highlighted sections represent iterative development loops and conditional checks for release readiness.



↑Figure 1.2 A sequence diagram illustrating the process of opening and editing a project in ProjectCenter. The diagram shows interactions between the user, ProjectCenter, the file system, and the editor, covering steps from accessing project files to making and saving changes. The highlighted section represents the project verification phase, where the system checks if the project exists before allowing edits or displaying an error message if the project is not found.



← Figure 1.3 - A sequence diagram illustrating the process of developing a GUI application with Gorm in GNUstep. The diagram outlines interactions between the user, IDE, Gorm, compiler, and test environment, covering steps from opening a GNUstep project to deploying and testing the application. The highlighted section represents the compilation phase, including success and failure conditions.

9. Data Dictionary

Glossary of Key Terms

GNUstep - An open-source implementation of Apple's Cocoa (formerly OpenStep) framework for cross-platform development.
Cocoa - Apple's native macOS application framework, derived from OpenStep.
OpenStep - An API by NeXT and Sun Microsystems that influenced Cocoa and GNUstep.
NeXTSTEP - The operating system and development environment that evolved into OpenStep and Cocoa.
Objective-C - The primary programming language for GNUstep.
Java, Ruby, GNU Guile, Scheme - Alternative languages supported via bindings.
Cross-Platform - Software compatibility with multiple OS environments like Unix and Windows.
ProjectCenter - The official GNUstep IDE for managing and compiling projects.
Gorm (Graphical Object Relationship Modeller) - A GUI designer using drag-and-drop functionality.
GNUstep Base Library - Provides fundamental Objective-C utilities like string handling, collections, and notifications.
GNUstep CoreBase Library - A low-level C library mirroring Apple's CoreFoundation.
GNUstep GUI Library - Implements Cocoa-style GUI components.
GNUstep GUI Backend - Handles OS-specific graphical rendering.
NSLock - Prevents simultaneous resource access.
NSThread - Manages parallel execution.
NSOperationQueue - Schedules concurrent tasks

10. Conclusion

GNUstep remains a vital open-source framework for cross-platform development, rooted in the OpenStep and Cocoa traditions. Its modular, object-oriented architecture enhances code reusability and adaptability, making it easier for developers to build and maintain applications. Over the years, GNUstep has evolved to improve compatibility with modern operating systems and technologies, ensuring its continued relevance in software development. While challenges like optimizing multi-threaded interactions persist, ongoing community contributions help refine its capabilities. Ultimately, GNUstep demonstrates the power of open-source collaboration in fostering innovation and expanding software accessibility across diverse computing environments.

11. Lesson learned

Throughout our exploration of GNUstep, we have seen how this open-source implementation of the OpenStep and Cocoa frameworks continues to evolve by integrating modern technologies and constantly enhancing its functionality. Initially developed to replicate OpenStep, GNUstep has steadily advanced, refining its features, improving compatibility with contemporary software development, and adapting to Apple's evolving ecosystem. Over the years, the platform has overcome significant technical challenges, evolving beyond traditional desktop environments to support cross-platform development.

Likewise, in the fast-changing landscape of technology, continuous learning and adaptation are essential. Our research into GNUstep has provided valuable insights into software development, highlighting the crucial role of user experience and its impact on simplifying the development process for programmers. By focusing on key factors such as usability, compatibility, and portability, we can drive innovation and build more adaptable and efficient software solutions that keep pace with technological advancements.

As open-source development continues to gain attention, frameworks like GNUstep are increasingly recognized for their potential to enhance intuitive, cross-platform application development. With its foundation in OpenStep, GNUstep has demonstrated how software can evolve to address larger and more complex challenges. Similarly, by building upon GNUstep or other open-source technologies, we can continue to develop innovative applications that empower developers to navigate the rapidly changing tech landscape.

12. References

GNUstep developer Tools. (n.d.). https://www.gnustep.org/experience/DeveloperTools.html

Gnustep. (n.d.-a). *Gnustep/libs-base*. GitHub. https://github.com/gnustep/libs-base

Gnustep. (n.d.-b). *Gnustep/libs-GUI*. GitHub. https://github.com/gnustep/libs-gui

Gnustep. (n.d.-c). *Releases · GNUSTEP/Libs-Base*. GitHub. https://github.com/gnustep/libs-base/releases

Gorm. GNUstep Developer Tools: Gorm. (n.d.). https://www.gnustep.org/experience/Gorm.html