GNUstep Conceptual Architecture

Jason Wei, 22jw23@queensu.ca
John Zhou, 22jz10@queensu.ca
Winston Chu, 22wjc@queensu.ca
Aidan Kim, 21ak191@queensu.ca
Vamiq Valji, vamiq.valji@queensu.ca
Alexander Marinkovich, 21agm26@queensu.ca

CISC 322
Group 28
February 14th, 2025

Table of Contents

1. Abstract

GNUstep is a free, open-source framework designed for cross-platform application development, providing compatibility with Apple's Cocoa framework (formerly OpenStep). The architecture of GNUstep follows a layered model, ensuring modularity, maintainability, and scalability. Gorm functions as a visual GUI builder at the highest level, streamlining interface design. The application interface layer consists of libs-gui and libs-back, handling user interactions and rendering. Below this, the foundation layer includes libs-base and libs-corebase, providing essential data structures, system utilities, and bridging compatibility with Apple's Core Foundation API. This structured approach minimizes dependencies on the underlying operating system, enhancing portability.

GNUstep components follow well-established design patterns, including the MVC paradigm, target action for event handling, delegation for adaptability, and drag-and-drop for user interaction. These patterns contribute to an organized, scalable, and maintainable development environment. The system's control flow begins with user inputs processed through libs-gui, which communicates with libs-back for rendering, while libs-base and libs-corebase manage fundamental system operations. The framework supports concurrency through an event loop and thread management, allowing efficient parallel execution.

From an architectural perspective, the layered approach is well-suited to GNUstep, ensuring clear separation of concerns and structured communication between components. While an alternative implicit invocation (publish-subscribe) architecture could allow greater parallelization, it introduces challenges in dependency management and fault tolerance, making the layered model the preferable choice.

Overall, GNUstep's architecture balances flexibility, maintainability, and cross-platform support, making it a robust framework for GUI application development. The separation of responsibilities among subsystems ensures efficient development, testing, and scalability, while the system's architectural choices enable long-term adaptability and seamless integration with various platforms.

2. Introduction and Overview

This report provides an abstract overview of the software architecture of GNUstep, an open-source framework for developing cross-platform applications compatible with Apple's Cocoa framework. GNUstep enables developers to create portable applications across macOS, Unix-like systems, and Windows, while also maintaining legacy OpenStep-based software. The report focuses on GNUstep's layered architectural style, which ensures modularity, scalability, and maintainability while allowing efficient GUI development.

The report is structured into several sections. It first introduces GNUstep's core components, including libs-back, libs-base, libs-corebase, libs-gui, and Gorm (a visual GUI builder). Each component plays a critical role in providing graphical rendering, foundational system utilities, UI frameworks, and GUI design tools. GNUstep follows the MVC paradigm, ensuring a structured approach to data management, UI interactions, and application logic. Additionally, design patterns such as delegation and target action enhance the framework's flexibility.

A key focus of the report is how these subsystems interact. Libs-back handles rendering operations, libs-base provides core Objective-C utilities, libs-corebase ensures compatibility with Apple's Core Foundation, libs-gui manages the graphical user interface, and Gorm facilitates GUI development through a visual interface. The control and data flow within GNUstep follows a structured event-dispatch model, where user inputs propagate through the system in a clear hierarchy. The report also explores architectural considerations, including system evolution, concurrency, and developer responsibilities. GNUstep's modular nature enables easy adaptation to new Cocoa features while ensuring backward compatibility. Concurrency mechanisms in libs-base provide threading, synchronization, and event-driven execution. Developer responsibilities are divided based on subsystem specialization, ensuring efficient collaboration.

The report compares alternative architectural styles, specifically the Implicit Invocation (pub-sub) model, which offers parallelization but lacks the structured dependency management of the layered architecture. The analysis concludes that the layered architecture is the best fit for GNUstep, ensuring clear separation of concerns, maintainability, and efficient data flow.

Lastly, the report examines two primary use cases that illustrate GNUstep's functionality with corresponding sequence diagrams: rendering a GUI window and handling user input (button click). For rendering a GUI window, the process starts when an application initializes and requests a new window. Gorm creates the UI components, which are then processed through libs-gui and libs-back, with libs-base handling resource allocation. The OS then renders and displays the window. Alongside that, for handling user input, the process starts when a user clicks a button, libs-gui detects the event and forwards it to libs-base, which then routes it to the appropriate controller. The system processes the event, updates the application state, and re-renders the UI as needed via libs-back and the OS.

The report provides a high-level understanding of GNUstep's architectural structure, component interactions, use cases, and design rationale, providing a strong foundation for exploring the detailed technical discussions in the subsequent sections.

3. Architecture

GNUstep is a free, open-source framework used for developing advanced GUI desktop or server cross-platform applications based on and integrated with Apple's Cocoa framework (formerly

OpenStep). Written in Objective-C and implementing the OpenStep API, GNUstep enables developers to write portable code working on various operating systems between Macintosh, Unit, Unix-like, and Windows. In addition, it allows developers to maintain legacy OpenStep-based software, and port existing Cocoa applications to Unix-like operating systems. GNUstep libraries are used in various applications, notable examples include Window Maker, TeXShop, Zipper, and TextEdit, showing its flexibility and applications.

GNUstep strives to provide developers with a mature and user-friendly environment offering a rich set of libraries and components that streamline application development. It enables Cocoa developers to build sophisticated commercial applications in weeks or months, dramatically reducing development time compared to traditional methods which might take years or never.

To achieve this functionality, GNUstep is broken into various subsystems and interacting parts. However, due to its complexity, this report focuses on the core libraries and components: libs-back, libs-base, libs-corebase, libs-gui, and gorm (others are outside our scope of this report). GNUstep is implemented as a system on the OS and the interaction of its components adheres to the layered architectural style as described by Garlan and Shaw, with clear separation between application logic, GUI, and backend rendering. The choice of this setup creates very little dependencies between the underlying OS and GNUstep, allowing portability, maintainability and scalability. The diagram in Figure 1 illustrates the structure and dependencies of the different layers and their components in the layered architecture.

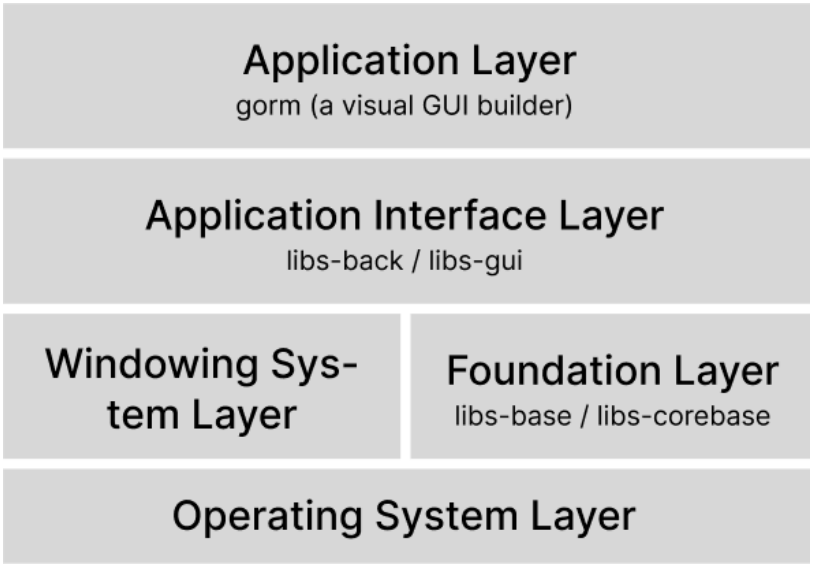

Figure 1: Diagram illustrating the layered architecture of GNUstep and its main components

Starting at the application layer, gorm serves as a visual GUI builder, allowing developers to design interfaces without writing code, simplifying UI creation. The application interface layer

consists of libs-gui and libs-back, where the layer enables user interaction through graphical components and manages to render on different platforms. Below this, the foundation layer includes libs-base and libs-corebase, where libs-base implements fundamental data structures, memory management, and system utilities, serving as the equivalent of macOS's Foundation framework, while libs-corebase bridges GNUstep with Apple's core foundation API for compatibility. In the same layer there's the windowing system layer that acts as an abstraction between the graphical user interface and the underlying display environment, ensuring compatibility with different window servers. Finally, the operating system layer which serves as the foundation that enables portability and system-level interaction across different platforms.

While the overall architecture of GNUstep follows a layered architectural style, ensuring modularity and separation of concerns, the individual components within this structure also adhere to well-established design patterns that enhance flexibility, maintainability, and usability. MVC paradigm is central to libs-gui, where the model (libs-base) manages data, the view (libs-gui) handles user interface elements, and the controller processes user interactions, ensuring a clear separation of concerns. The target-action pattern is extensively used in UI event handling, allowing user interface elements, such as buttons, to trigger actions in a loosely coupled manner. Additionally, GNUstep implements drag-and-drop, enabling seamless interaction between GUI elements by allowing users to transfer data between views or applications. Lastly, the delegation pattern is also widely used, particularly in libs-gui, where objects delegate responsibilities to other components, reducing tight coupling and improving adaptability. These patterns collectively contribute to a well-structured framework, ensuring that GNUstep applications remain scalable, extensible, and easy to maintain.

The following sections provide a detailed breakdown of each component, exploring how they specifically interact with each other and how they contribute to the system's overall functionality.

Libs-back

The libs-back system handles front-end and back-end rendering for GNU Step applications. It can be broken down into two defined systems; rendering libraries and display servers. The graphics rendering libraries include Xlib, Cairo, Opal, XDPS, Art, and Winlib. And the supported display servers are Win32, X11, and Wayland. There is also support for headless development. Using these settings, the subsystem can handle output into any of the supported platforms and rendering options. It takes in input from libs-gui for what to render, and with the objects/classes it defines for output options and from libs-base or libs-corebase, it can effectively render the programmed system.

Libs-base

Libs-base is a library of general-purpose, non-graphical Objective C objects. Strings, object collections, and byte streams are just a few of many object abstractions implemented by this

library. It can be used throughout GNUStep as a basis for objects used within the library and is the component that has the most involvement with external sources like the operating system to handle the responsibilities of the application.

The library itself is expansive on everything needed for an application to be processed by the OS. It handles time, authentication, networking, concurrency (i.e. processes and threads handling), networking, as well as standard classes such as strings, hash tables, numbers, arrays, etc. It is used by libs-gui and libs-back as the Foundation package for the standardization of communication between them and the OS.

Libs-corebase

Libs-corebase is a library of general-purpose non-graphical C objects including types for strings, collections, notifications, network ports, event loops, and more (low-level data management utilities). It provides functionality to implement the non-graphical part of Apple's CoreFoundation framework. In other words, the library serves as a compatibility layer between GNUstep's Foundation framework and Apple's Core Foundation. 'libs-corebase' is designed to connect the gap between Objective-C-based Foundation APIs and Core Foundation (C-based) APIs, which makes it easier to write portable code between GNUstep and macOS.

Libs-corebase is structured into many key components, each providing different uses. It has components that handle string and data management. It can manage string objects, including text processing, localization, and structured data storage. This interacts with GNUstep applications and libs-base. Libs-corebase also includes event handling and the application lifecycle. This manages event-driven execution and makes sure that the application runs smoothly. This component relies on libs-base for Objective-C messaging and libs-gui for UI event propagation.

The library includes cross-platform compatibility, making sure that there is standardized Core Foundation functionality across all software (Linux, Windows, and macOS). It connects GNUstep's internal APIs with Apple's Core Foundation functions. Libs-corebase also has file and URL handling, where it provides shorthand methods and abstraction for filesystem operations. This makes sure that file paths, URLs, and networking resources are handled continually. This works with libs-base, Foundation file APIs, and other OS-specific implementations.

Libs-corebase acts as a bridge between GNUstep's Foundation (libs-base) and lower-level Core Foundation functionalities. 'Libs-corebase' gives Core Foundation compatibility in GNUstep, making sure the interaction between GNUstep's Foundation framework (libs-base) and macOS-like Core Foundation APIs is seamless. The components work together with libs-base to maintain compatibility and portability across platforms.

Libs-gui

Libs-gui provides and manages the graphical user interface framework for GNUstep applications. This includes elements such as buttons, text fields, tables, and more. Libs-gui handles events through Objective-C's delegation model, allowing the user to assign actions to objects such as clicking, typing, scrolling, etc. Using graphics rendering libraries such as the ones listed under libs-back, the user can also draw and render custom images for use in the GUI.

Libs-gui relies on libs-base for its foundation as it provides important data structures such as strings and arrays. It also uses libs-back for its rendering of graphics as the GUI doesn't directly handle low-level rendering.

Gorm (a visual GUI builder)

Gorm, also known as GNUstep Object Relationship Modeler, is GNUstep's visual GUI builder. By using Gorm, users can design and modify user interfaces while saving them as .gorm files. One convenient function is the drag-and-drop function, where the user can drag elements such as buttons, text fields, etc. from a dropdown menu onto the page. Components can also be linked to actions in Objective-C code, creating functionality for the GUI without the hassle of writing code manually.

Gorm uses libs-gui for its GUI components, as all of its features are provided and found in libs-gui. It also uses libs-base to load or save the generated .gorm files, since libs-base can deal with encoding/decoding the objects.

4. Architectural Considerations: Evolution, Concurrency, and Responsibilities

System Evolvement

As GNUstep aims to be an open-source implementation of Cocoa, it must be ready to support any new features Apple rolls out. With its modular architecture, there are many different layers, each with separate functions. For example, the Foundation Layer (libs-base, libs-corebase) lays out a core foundation for the application. In the event of a future change, it can extend to support any new Objective-C runtime features. As for the Application Interface Layer (libs-gui, libs-back), libs-gui can be updated to introduce any new UI elements added (animations, icons) and libs-back can integrate any new technologies to help improve its rendering. Finally, at the Application Layer (Gorm), the builder can be updated to support any new models.

While evolving to support new features, GNUstep also strives to keep previous versions from breaking down with the changes. This backward compatibility ensures that users do not have to worry about having to constantly check for updates and modify their code accordingly. For example, .gorm files from older Gorm versions will always be loadable without errors, as long as it's being loaded into a version after its own.

A large reason why GNUstep can keep up with the improving technology is due to how it is fully open-sourced. The community is always working to propose changes to implement new technology. Being completely community-driven also helps GNUstep be flexible, as it isn't limited to following the technology of a few vendors. Any rising new technology can be implemented, allowing constant innovation.

Control and Data Flow

The control flow of the GNUstep application begins with the event-dispatch where user inputs interact with the GUI. These inputs include actions such as keyboard usage or clicks with the pointer. These inputs are registered by the operating system and then directed to libs-back, the backend library. The backend library converts inputs into GNUstep events that the libs-gui can understand. libs-gui, essentially the main controller of the front end, is built to handle the control flow of the interface logic. It controls how or what window to present to the UI and can call relevant methods in the Objective-C classes from the backend components. Underneath libs-back is libs-base, which maintains the event loop for the GNUstep program. Finally, below all of this is libs-corebase, which provides the fundamental Objective-C operations backing the data structures that build libs-base.

Most of the data flow of GNUstep takes place in libs-base, where a lot of the classes that involve the storage and manipulation of data are built on top of libs-corebase. libs-corebase is very important in direct relation to libs-base as it contains lower-level C-based structures such as CFString or CFArray. The CF stands for Core Foundation, which are C-based data structures and system functionality that higher-level components such as libs-base can use. In libs-base, there exists classes such as, for example: NSData, NSFileManager, or NSHashTable, which are higher-level abstractions that build on top of existing C-based foundations. The NS stands for NeXTSTEP, referencing the historical origin of these classes from the NeXTSTEP operating system. Other components participate in certain sections of the data flow. libs-gui uses the data structures in libs-base to manage the creation and updates of the user interface. It also communicates graphical data (windows, text, etc.) back to libs-back, where libs-back can translate this graphical data into GNUstep platform-native operations. Finally, gorm's role in the data flow is to act as a blueprint for the user interface. In .gorm files, it informs the libs-gui about what objects to create, how to connect them to the Objective-C classes in libs-base, and what properties they should contain. At runtime, libs-gui loads these files to instantiate and link UI elements with the underlying data structures, ensuring everything stays in sync. Therefore, while libs-base remains the primary hub for data flow, libs-gui, libs-back, and Gorm each contribute to how information is represented and exchanged throughout the application's lifecycle.

Concurrency

Concurrency is present throughout the system. Files of concurrency are found in the libs-base component. Central to all concurrency files is the GNUstep event loop found in `NSRunLoop`,

which isn't concurrent by itself, rather, it enables concurrency and event-driven execution. This section can monitor and respond to various input sources without blocking the main thread. Then there is the explicit concurrent thread management inside the `NSThread` file where multiple threads can be created, started, synchronized, and torn down depending on the application's needs. These threads can be controlled manually in parallel, giving a lot of control to the developer. For these concurrent threads, resources such as locks (example: NSLock) and condition variables (example: NSCondition) are usable to prevent race conditions. Higher-level abstractions such as NSOperationQueue are tools that allow the developer to execute threads automatically with optional dependencies so that they don't have to micromanage every single concurrently running thread.

Responsibilities among Developers

The division of responsibilities among developers is important because of GNUstep's modular architecture. The system consists of multiple subsystems ('libs-') that have distinct roles, and that influences how tasks are distributed among developers. Some key implications would include subsystem-based specialization, concurrency, testing, code access, and cross-platform considerations.

Developers typically specialize in their preferred libraries and frameworks which can cause differences in levels of competencies in what they are building and how that fits together. Dividing developers to their strengths is very important. For example, for the foundation (libs-base) developers work on core functionalities like memory management, data structures, and networking. For the GUI (libs-gui), developers are responsible for UI components, widgets, event handling, and rendering. For the backend (libs-back), developers deal with rendering graphics and other OS-specific tasks. Lastly, GORM has developers focusing on GUI tools like drag-and-drop.

Regarding concurrency, the modular design and dividing developers between tasks promotes parallel development. Teams can work on separate libraries without significant conflicts. But of course, this needs to be well planned because API contracts and clear documentation are necessary to ensure smooth interaction among components and to keep developers happy.

Testing responsibilities extend to unit, integration, and GUI testing. For unit tests, each subsystem must have its own suite of tests to ensure a reliable program. Developers must make sure there is compatibility between components (e.g. libs-gui interacting with libs-base). GUI testing would require some extra test cases for usability and responsiveness across different software and machines.

Regarding code access, GNUsteps is an open-source project, so contributions come from independent developers. Code reviews, version tracking (Git), and modular ownership are

absolutely necessary in order to maintain code quality and grow the project. Documentation and architecture guidelines must be clear to all developers and new contributors while keeping onboarding efficient.

Lastly, cross-platform considerations need to be accounted for by developers so people can use the project across different machines and software versions. Platform specific adjustments can be handled by dedicated maintainers for different operating systems as needed.

5. Derivation and Alternatives

Our choice of layered architecture stems from the adjacently connected nature of the underlying system.

Gorm can be used to serve as the interface for the user to develop applications, which makes it fall under the application tier, which focuses on what the end user interacts with and views.

Libs-back and Libs-gui lie in the application interface layer, the layer dedicated to performing tasks regarding visualizing the respective application. Gorm will communicate to libs-gui what to visualize, and then libs-gui will communicate with libs-back for output for the particular graphics library and platform to render it on the given data.

After the application interface layer, two adjacent layers play a crucial role in the system's functionality. The Windowing System layer, which manages the windows and how it is represented on the computer screen regarding window resizing and positioning. Alongside it, the Foundational layer consists of libs-base and libs-corebase – two libraries that contain fundamental data structures, algorithms, and policies that work with the OS. These systems lie in the middle between the OS and the application infrastructure layer due to their necessity of the system to communicate with the operating system for resource management, data, processing, etc. in order for the overall system to operate.

This architecture alludes to the connection between these systems can be purely implemented with adjacent connections, with each layer facilitating the inputs and outputs from the user to the OS, and back. Thus, layered is the ideal choice for the implementation of these five libraries.

An alternative to this proposed architecture could be the implicit invocation style, which is defined by components that publish their completed events and subscripts to the other components for their completed events. This relationship between components is why this architectural style is also called pub-sub. This works by having every component publish its events to all of its subscriber components, and then those very subscribers process those events. This has the advantage of parallelization as each component can reside on its own thread/process. We can develop an architecture diagram as shown below:

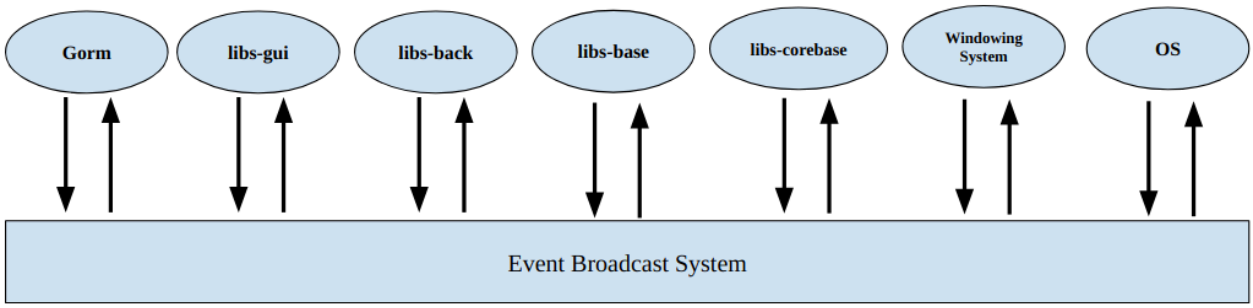

Figure 2: Diagram illustrating Implicit Invocation architecture of GNUstep and it's main components

Note we include the OS and Windowing system here for comparability with the layered-based architecture.

The disadvantages here are that concrete connections between modules like libs-back and libs-gui for handling this system are not directly supported by this architecture. Gorm will always depend on libs-gui, which will require libs-back for data regarding rendering and both will require libs-base and/or libs-corebase for communication with the OS. If a single component ever fails, its subscribers will not know it has failed and will continue to wait until the system is terminated by the user and, depending on implementation, will not return any errors or crash by the inherited design. Although this architecture style is possible for GNUSteps for its ability to run its components in parallelization, it is not fault-proof and thus is our second choice in architecture style.

6. Diagrams

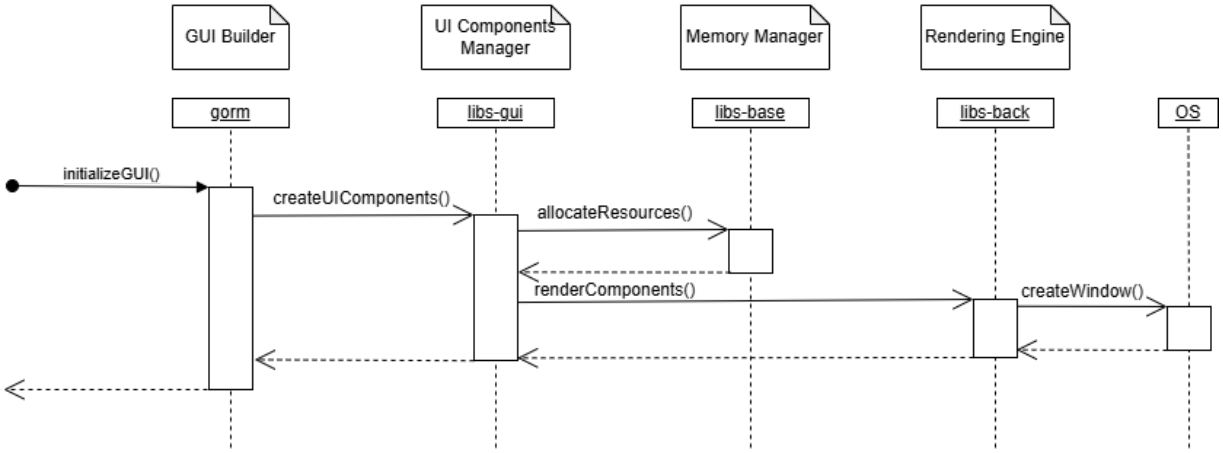

Figure 3: Sequence diagram for use case 1, rendering a GUI window

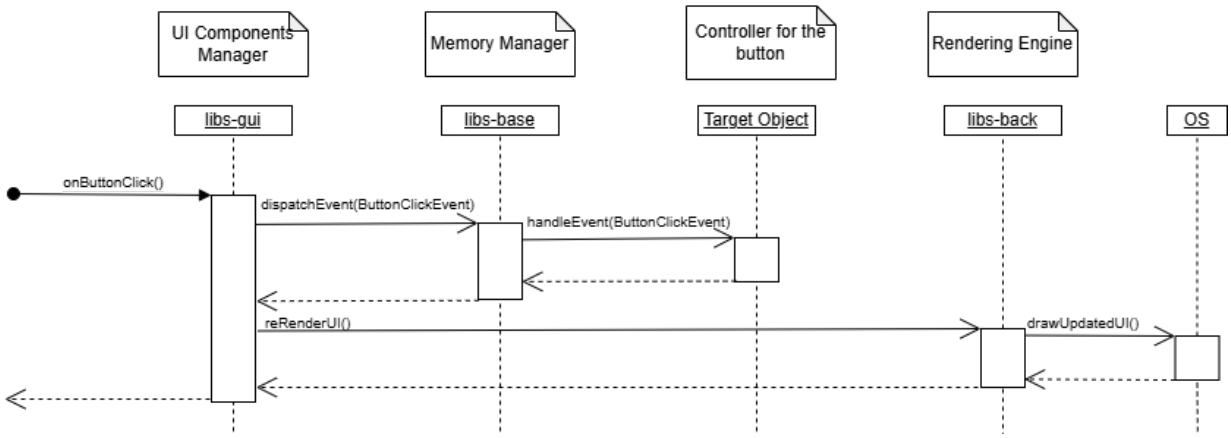

Figure 4: Sequence diagram for use case 2, handling user input (button click)

7. Use Cases

Rendering a GUI Window, Use Case 1

This use case describes the process of initializing and displaying a graphical user interface (GUI) window. The user starts the application, and the GUI framework (gorm) creates the necessary UI components using libs-gui. The libs-base module allocates resources and sends rendering requests to libs-back, which handles the actual drawing operations. The OS then creates and displays the window.

Handling User Input (Button Click), Use Case 2

This use case describes what happens when a user clicks a button in the GUI. The event is first detected by libs-gui, which forwards it to libs-base for processing. libs-base then dispatches the event to the appropriate Target Object (in this case, it is a controller that handles a button's clicks). After executing the logic, libs-base sends a UI update request to libs-gui, which calls libs-back to re-render the UI. If needed, libs-back sends a request to the OS to update the screen.

8. Conclusion

In conclusion, GNUstep is a modular, open-source framework designed to replicate Apple's Cocoa environment while providing cross-platform compatibility. Our architectural analysis has demonstrated that GNUstep follows a layered architecture, which ensures modularity, scalability, and maintainability. Each of its core components: libs-back, libs-base, libs-corebase, libs-gui, and Gorm, play a distinct role in facilitating application development. The system effectively separates concerns, with libs-base and libs-corebase managing fundamental operations, libs-gui handling UI elements, libs-back ensuring graphical rendering, and Gorm streamlining interface design.

Our findings emphasize the portability and extensibility of GNUstep. The framework's modular design enables portability and adaptability to Apple's evolving ecosystem while maintaining

compatibility with legacy OpenStep applications. Its event-driven architecture and concurrency mechanisms optimize performance, while the layered approach minimizes OS dependencies for seamless cross-platform execution.

Future improvements could enhance macOS compatibility through support for modern APIs, multi-threading, and GPU acceleration. Expanding documentation and IDE integration would further streamline development. With ongoing community-driven innovation, GNUstep continues to be a powerful and adaptable alternative to Apple's proprietary Cocoa framework.

Overall, GNUstep remains a powerful and adaptable framework for cross-platform Objective-C development. With continuous community-driven innovation, it has the potential to further evolve as a robust alternative to Apple's proprietary Cocoa ecosystem.

9. Lessons Learned

We wish we had a deeper understanding of libs-gui and libs-back. Libs-back, in particular, feels quite abstract, as it primarily formats data for the chosen graphics library and platform, but its exact role is not immediately clear from the source code. It seems logical that these two components are closely interdependent. Libs-gui determines what to render, while libs-back ensures it is displayed correctly. However, much of this understanding is based on interpreting source file headers rather than comprehensive documentation. Reviewing the GNUstep manual and additional documentation would have provided a clearer picture of their interaction and helped solidify my understanding of how these components function together.

Another lesson learned is the importance of thoroughly understanding GNUstep's architecture before diving into specific components. Initially, the complexity of its layered structure made it challenging to grasp how different libraries interact. A more structured approach to studying dependencies and data flow earlier in the process would have made the report easier.

10. Data Dictionary

Cocoa (Framework) – Apple's proprietary framework for developing macOS applications. GNUstep aims to be an open-source alternative.
Core Foundation (Framework, C-based) – Apple's low-level C-based framework, which libs-corebase implements to ensure compatibility with GNUstep.
OpenStep (Framework, Historical) – The predecessor to Cocoa, was developed by NeXT and later adopted by Apple. GNUstep is an implementation of OpenStep.
Objective-C (Programming Language) – The primary language used for developing applications in GNUstep.

11. Naming Conventions

API – Application Programming Interface

C – C Programming Language
CF – Core Foundation (Apple's C-based framework)
GNU – GNU's Not Unix
GUI – Graphical User Interface
MVC – Model-View-Controller
NS – NeXTSTEP (Prefix for classes, e.g., NSString, NSArray)
OS – Operating System
UI – User Interface

12. References

GNUstep SystemOverview. (2025). Made-It.com.
http://gnustep.made-it.com/SystemOverview/index.html

Programming under GNUstep--An Introduction (2025). Acm.org.
https://dl.acm.org/doi/fullHtml/10.5555/640534.640540

GNUstep: Introduction. (2025). Gnustep.org.
https://www.gnustep.org/information/aboutGNUstep.html

GNUstep. (2025, 22 January). Wikipiedia.org
https://en.wikipedia.org/wiki/GNUstep

GNUstep. (2025, February 12). GitHub.
https://github.com/gnustep

Gorm Manual - GNUstepWiki. (2025). Gnustep.org.
https://mediawiki.gnustep.org/index.php/Gorm_Manual

Botto, F., Frith-Macdonald, R., Pero, N., & Robert, A. (n.d.). *Objective-C Language and GNUstep Base Library Programming Manual*. Retrieved February 10, 2025, from http://andrewd.ces.clemson.edu/courses/cpsc102/notes/GNUStep-manual.pdf