Group 4 - Conceptual Architecture of GNUstep GUI Framework

Friday, February 14, 2025

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)

0. Abstract

Cross-platform GUI development is particularly difficult since it needs to maintain the native look and feel of the host operating systems. Traditional solutions have involved maintaining several codebases and complex rendering pipelines. GNUstep solves these issues by being an open-source reimplementation of the OpenStep specification, which provides a standard application development environment for command-line and GUI applications. Its modular architecture employs core libraries like libs-base and libs-corebase for core functionality (collections, memory management, threads, networking) with libs-gui and libs-back collectively handling user interface concerns and backend rendering on several platforms (X11, Windows GDI, etc.).

With Objective-C being its native language, GNUstep is rich in APIs for component reuse. libs-gui's MVC framework splits business logic and presentation, making it easy to maintain and extend. libs-back, however, hides low-level rendering details so that UI code can be written once and deployed on several systems with little or no effort.

This paper describes GNUstep's conceptual architecture, with particular emphasis on how its constituent subsystems and modules interact, control and data manage, provide concurrency support, and change. At high levels of abstraction, dependencies between subsystems are charted, and sequence diagrams record key use cases in operation. We uncover in our analysis a portability-driven, modularity-and-maintainability-motivated architecture, making GNUstep an attractive choice for cross-platform software development. Ultimately, learning GNUstep's design concept is worth it for new contributors, introducing them to its fundamental building blocks, relationships, and future directions.

1. Introduction and Overview

Creating cross-platform applications that will run exactly the same way on various operating systems has long been a challenge to developers. Having to maintain platform-specific codebases while delivering a consistent look and feel can result in disjointed development processes and exorbitant maintenance. GNUstep resolves these concerns by providing an open-source application framework compliant with the OpenStep specification, enabling applications to be developed once and then, with little or no modification, run on any number of platforms. It is realized using Objective-C as its programming language and consists of a clearly defined application development framework that incorporates core libraries along with user interface toolkits into a unifying framework.

This report describes the conceptual framework of GNUstep, in the hope of providing new contributors and interested readers with an overview of system organization, more than an analysis of implementation specifics. In developing this framework, special attention has been given to explaining how the different GNUstep libraries interact, how control and data are exchanged between different levels of the framework, and how concurrency and modularity are accommodated. The main aim is to illustrate how GNUstep's framework continues to be portable and extendable, the same characteristics inherent in a cross-platform framework that has to conform to evolving operating systems and users' needs.

While describing the conceptual model, the report summarizes the key subsystems of GNUstep, including foundation libraries offering runtime services, memory management, and data structures, and graphical libraries that implement a Model-View-Controller strategy to building user interfaces. Though some of these libraries provide support for command-line application development, others focus on the domains of graphics rendering and user interaction abstraction so that it becomes possible to write programs in a platform-independent way. By highlighting these complementary but different functions, it becomes easier to understand how GNUstep operates as a unified system despite being modular.

The document is structured to lead the reader from high-level architectural overview down to detailed information and use cases. It begins with the subsystems and illustrates how they communicate in a high-level box-and-arrow diagram, describes the data flow and control within the framework, and remarks on how GNUstep manages updates and additions of new features over time. Concurrency is also briefly discussed to indicate how the framework supports multithreading, with the intent of indicating how its core libraries and GUI libraries synchronize when doing things in parallel. Use cases are then employed to specify daily usage patterns, e.g., initializing a GNUstep application or drawing a window; each usage pattern is backed up with a sequence diagram illustrating function calls and data flow among subsystems.

The architectural view above leads to several definite conclusions about GNUstep. One observation of particular importance is that the modular and layered nature of the framework not only enables it to support a wide range of operating systems, but also simplifies the porting of GNUstep to future environments. A second observation is that the strong separation of concerns—particularly the utilization of Objective-C and the Model-View-Controller pattern—simplifies the maintenance and comprehension of the code for developers. Architectural choices in GNUstep also demonstrate a strong focus on reusability and flexibility, such as in the manner in which its drawing backends can be exchanged without necessitating complete rewritings of the GUI logic. Finally, witnessing the community-based development of GNUstep over the years demonstrates that this architecture, though mature, remains responsive to contemporary demands by embracing new technology, incorporating or enhancing libraries, and embracing contributions from a wide range of developers and users.

In exploring these elements of GNUstep's conceptual architecture, this report is presented both as a beginner's introductory primer and as an analytic guide for those considering evaluating or contributing to the framework. Although the architecture outlined herein offers a top-level roadmap, it also constitutes a basis for additional, more in-depth explorations. Additional work could address real implementations of particular libraries, investigate particular design patterns in more depth, or even suggest refactorings for better adherence to more modern software engineering best practice. In providing a shared vision of the internal organization of GNUstep, this report hopes to illustrate the timelessness of the OpenStep philosophy and the value of open-source collaboration in the development of complex, portable, and well-engineered software systems.

2. System Overview & Architecture

2.1. Subsystems and Components

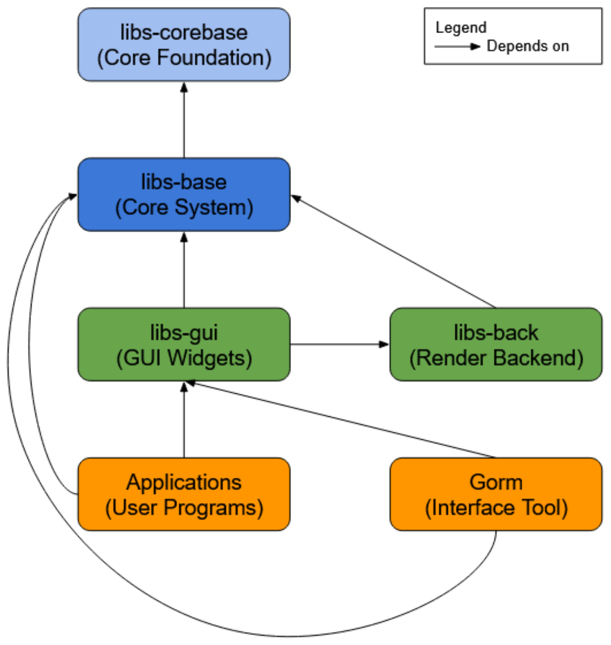
GNUstep is a cross-platform, open-source framework designed to allow the easy development of both graphical (GUI) and non-graphical applications. The GNUstep Foundation library is based on the OpenStep specification, upon which the Cocoa framework for macOS is also based. This means that, by writing code against the GNUstep Foundation, you can more easily port your code to other platforms, such as macOS and Windows (using the Cocotron compiler). Being a modern, object-oriented framework, GNUstep is written in Objective-C, which is a superset of the C programming language that integrates features from Smalltalk. The GNUstep framework is broadly divided into four major parts.

First the libs-base, this is the core library, the foundation on which the other parts of the framework are built. libs-base includes the classes NSObject, NSString, NSArray, NSDictionary, NSData, and others. The naming convention is identical to Apple’s Foundation framework, allowing you to port code from macOS to GNUstep much more easily. Furthermore, the libs-corebase, this is a part of GNUstep that is missing from Apple's Foundation library. The libs-corebase corresponds to Apple's Core Foundation library, which includes fundamental utilities, such as low-level data structures (arrays, dictionaries, sets, and strings), system utilities, and the event loop handling on the main thread. Afterwards, the libs-gui, this part of the GNUstep framework is equivalent to Apple's AppKit. The NS prefix is used for classes in this part, such as NSWindow, NSView, and NSButton. Lastly, libs-back is the rendering backend for GNUstep applications. The back-end renderer can be switched to take advantage of platform-specific rendering mechanisms. For example, on macOS, GNUstep uses the Quartz renderer for rendering, while on Windows, GNUstep uses the Win32 API and the X11 renderer on Unix-based systems.

GNUstep also includes several additional tools, such as the graphical user interface builder called Gorm. Gorm is the GUI builder that will allow users to design their applications with ease, similar to Apple's Interface Builder. The difference is that Apple's Interface Builder produces XML-based .nib or .xib files, while Gorm produces .gorm files. These files contain the layout and the connections to the objects in your program. You can load .gorm files dynamically at runtime, so that you don't need to hardcode the interface elements in your program.

2.2. Component Interactions

GNUstep's components work together through a series of well-defined interactions. These interactions are crucial for understanding how the system functions as a whole.



The module dependency graph reflects the following hierarchy: libs-corebase at the bottom, libs-base which builds on libs-corebase, libs-gui and libs-back which build on libs-base, and all user GUI applications which build on libs-gui (and libs-base).

- Everything builds on libs-base. And these are just basic but necessary methods for handling text, numbers, files in a way that is very handy for developers. So, if you are writing/developing any app you will end up using it.

- libs-gui will need to talk to libs-back for actually showing them actual things on the screen. Think of libs-gui as an architect saying put a button here, and libs-back is like a contractor, who actually builds it. This separation is very important, now you can swap out an entirely different libs-back for different systems without changing your design.

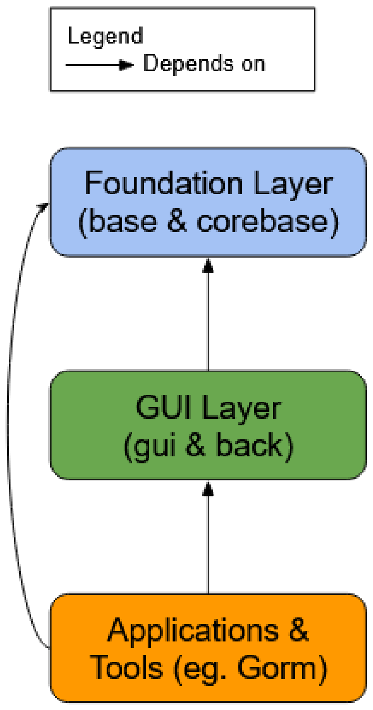
- gorm is a design tool that sits on top of all these above components. It's like saying you design your house, how it looks – same way you use gorm layer to design your interface. gorm generates .gorm files, which your application can load at runtime to know the interface design to be created.

2.3. Architectural Style

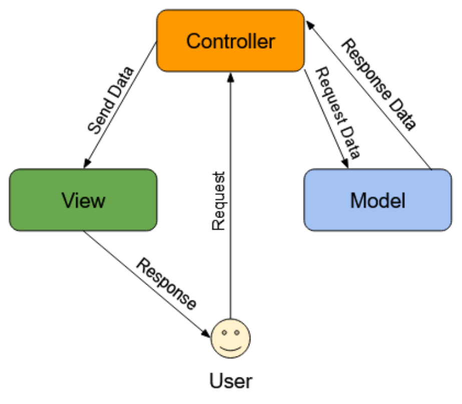
The layered design is a key pattern of GNUstep architecture that helps organize our code into a series of components, or layers, stacked on top of each other. Up to this point, this setting might remind you of a delicious layered cake. We can certainly build the layered cake without any particular structure. But we do end up with a big mess. It will be very hard for the developer to put the cake back to normal shape.

The layered architecture can help us organize our code better:

- It is easier to understand. We can look at any component and have a rough idea where to expect the rest of the components.
- If something goes wrong, we can easily zone in on the layer level, where the problem is.
- If we need to add new features, we know the layer where the new functionality should reside. Furthermore application developers usually only need to worry about the bottom layer.


Legend
Depends on
Foundation Layer (base & corebase)
GUI Layer (gui & back)
Applications & Tools (eg. Gorm)

Furthermore, a common design pattern used in GNUstep applications, to further improve maintainability, is the Model-View-Controller (MVC) pattern. Although not strictly enforced, GNUstep's class structure encourages separation of concerns. You can keep data and business logic in model classes (using libs-base for fundamentals), and place UI logic in views/controllers (provided by libs-gui). The frameworks promote patterns like delegation, notifications, and target–action, all of which support MVC-like architecture in practice.



GNUstep also incorporates the object-oriented, and event-driven/pub-sub architectural styles. At its core, GNUstep follows the OpenStep APIs, which are fundamentally object-oriented. Classes in libs-base (NSString, NSArray, NSDictionary) and libs-gui (NSWindow, NSButton, NSTableView) encapsulate data and behavior, and developers work primarily by sending messages to objects.

Much like in macOS/iOS or other windowing systems, the GUI portion (libs-gui and the backends) is event-driven. The main run loop waits for user input (mouse events, key presses, etc.) or system events, then dispatches them to the appropriate handlers in the application's objects.

3. Control and Data Flow

GNUstep evolves through a modular architecture that allows new features and updates to be integrated seamlessly while maintaining system stability. The development process is driven by both community contributions and structured updates, ensuring that applications remain compatible across different platforms. By separating core components from additional frameworks, GNUstep enables developers to introduce new functionality without disrupting existing features. This modularity is crucial for maintaining a system that is both flexible and scalable.

One of the defining characteristics of GNUstep's evolution is its adherence to the object-oriented programming paradigm, which emphasizes encapsulation and abstraction. This approach ensures that core functionalities remain independent, making it easier to modify or enhance specific components without affecting the entire system. The use of well-defined application programming interfaces and standardized message-passing mechanisms further supports this modularity, allowing developers to extend functionality without introducing unintended dependencies.

Cross-platform compatibility is another important aspect of GNUstep's design. By implementing an abstraction layer that decouples platform-specific operations from application logic, GNUstep ensures that code written for one environment can run consistently on others, such as macOS and Linux. This approach minimizes the need for developers to make extensive modifications when porting applications across different operating systems. Additionally, the system supports dynamic linking, which allows new components to be loaded at runtime without requiring a full system rebuild. This capability enhances flexibility and makes it easier to introduce updates and new features in a non-disruptive manner.

GNUstep provides robust support for multithreading, enabling applications to perform tasks concurrently and improve overall performance. This is particularly important for graphical user interface applications and background processing, where responsiveness is key. The system is designed to handle concurrent execution efficiently, ensuring that multiple components can interact without causing conflicts or performance bottlenecks.

Multithreading in GNUstep is managed through mechanisms such as operation queues and dispatch models, which allow tasks to be scheduled and executed asynchronously. This approach helps distribute computational workloads more efficiently, reducing the risk of a single task monopolizing system resources. Additionally, GNUstep is compatible with POSIX threads, offering developers finer control over thread management when needed.

To ensure that concurrent processes do not interfere with each other, GNUstep employs various synchronization techniques. These mechanisms prevent multiple threads from modifying shared resources simultaneously, reducing the likelihood of race conditions and system instability. Synchronization ensures that applications remain reliable even when executing complex, multi-threaded operations.

A key aspect of GNUstep's concurrency model is its event-driven architecture, which enables components to communicate asynchronously. This is particularly beneficial in UI applications, where long-running background tasks must not block user interactions. By handling tasks asynchronously, GNUstep ensures that applications remain smooth and responsive, even when performing computationally intensive operations. The balance between concurrency and system stability allows GNUstep to efficiently manage performance without sacrificing reliability.

4. Use Cases and Sequence Diagrams

4.1 Opening a GNUstep application (e.g., launching a GUI app).

To launch a GNUstep application, multiple subsystems have to work together to initialize the application, set up the user interface, and render the window. This process involves the operating system, the window manager, the GNUstep application, and core GNUstep libraries (libs-gui and libs-back).

Actors:

User – Initiates the application launch.

Operating System – Manages the execution of the application.

Window Manager – Handles window-related tasks.

GNUstep Application – The main program being launched.

GNUstep GUI Library (libs-gui) – Manages UI elements.

GNUstep Backend Library (libs-back) – Handles low-level rendering.

Display Server – Renders the final graphical output on the screen.

Preconditions

The application is installed and all required GNUstep libraries are available.

The operating system and window manager are functioning correctly.

Main Flow

The User requests to launch the GNUstep application.

The Operating System notifies the Window Manager to handle the process.

The Window Manager starts the application process.

The GNUstep Application loads the required GUI libraries (libs-gui).

libs-gui initializes the user interface components.

The application loads the Backend libraries (libs-back), which are responsible for rendering.

The Backend library initializes backend components.

The Application requests UI component rendering.

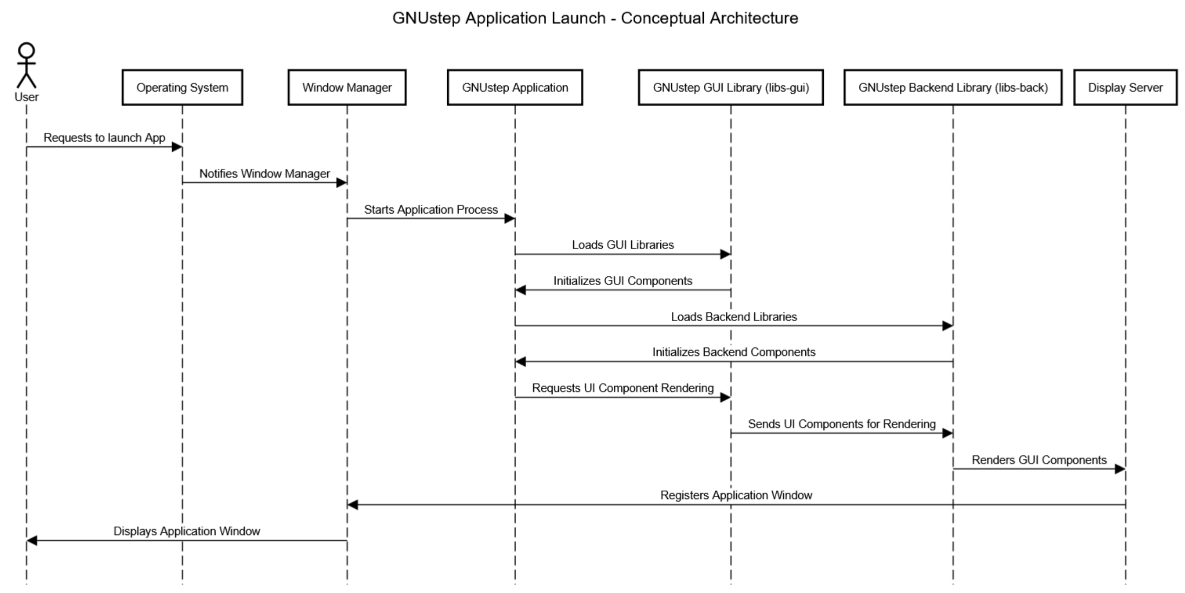
libs-back sends the UI components to the Display Server for rendering.

The Display Server registers the application window and renders the UI.

The User sees the application window displayed on the screen.

Sequence Diagram

The following sequence diagram displays the process:


GNUstep Application Launch - Conceptual Architecture
User
Operating System
Window Manager
GNUstep Application
GNUstep GUI Library (libs-gui)
GNUstep Backend Library (libs-back)
Display Server
Requests to launch App
Notifies Window Manager
Starts Application Process
Loads GUI Libraries
Initializes GUI Components
Loads Backend Libraries
Initializes Backend Components
Requests UI Component Rendering
Sends UI Components for Rendering
Renders GUI Components
Registers Application Window
Displays Application Window

4.2 Rendering a Window Using GNUstep.

The process for a window to be created and rendered within a running GNUstep, involves requesting a new window, managing rendering, and displaying the window on screen using the GNUstep GUI and backend libraries.

Actors

User – Performs an action that requires a new window.
GNUstep Application – Manages window requests.
GNUstep GUI Library (libs-gui) – Handles UI components and layouts.
GNUstep Backend Library (libs-back) – Processes drawing and rendering requests.
Window Manager – Manages the positioning and visibility of windows.
Display Server – Handles low-level rendering and display operations.

Preconditions

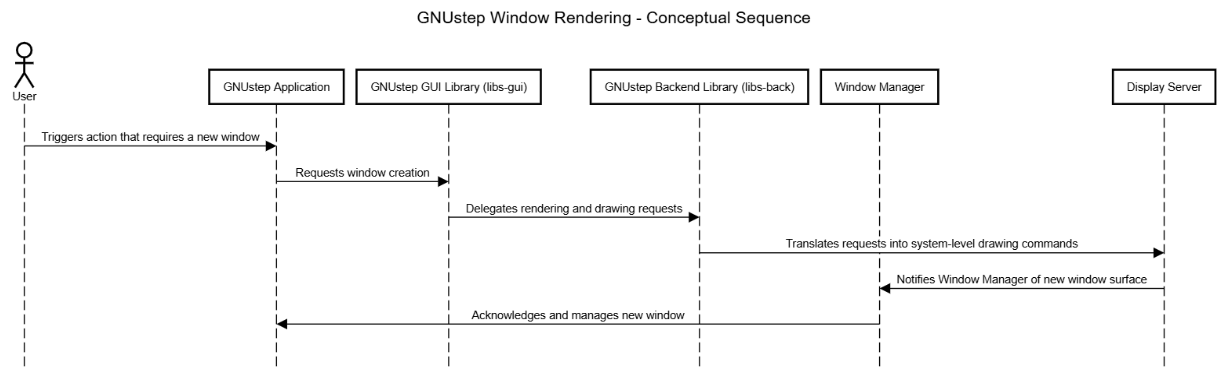
The GNUstep application is already running.
The GUI framework (libs-gui) and rendering backend (libs-back) are initialized.

Main Flow

The User triggers an action that requires a new window.
The GNUstep Application requests window creation from libs-gui.
libs-gui delegates rendering and drawing tasks to libs-back.
libs-back translates the request into system-level drawing commands.
The Window Manager is notified of the new window surface.
The Display Server renders the window on the screen.
The Application acknowledges and manages the new window.

Sequence Diagram

The following sequence diagram displays the process:


GNUstep Window Rendering - Conceptual Sequence
User
GNUstep Application
GNUstep GUI Library (libs-gui)
GNUstep Backend Library (libs-back)
Window Manager
Display Server
Triggers action that requires a new window
Requests window creation
Delegates rendering and drawing requests
Translates requests into system-level drawing commands
Notifies Window Manager of new window surface
Acknowledges and manages new window

5. Derivation Process

Our analysis of GNUstep's conceptual architecture was derived through a combination of reviewing official documentation, exploring repository structures, and analyzing source code. Given the limited up-to-date architectural documentation available, our approach involved a mix of top-down and bottom-up investigation.

5.1 Research Sources

To reconstruct GNUstep's architecture, we relied on:

- Official GNUstep Documentation: High-level overviews of key libraries and their intended roles.
- Source Code from GNUstep Repositories: Examining folder structures, class definitions, and dependencies.

5.2 Reverse Engineering the Architecture

Understanding GNUstep required reconstructing its layered structure by tracing dependencies between libraries:

- Identifying Core Components: We examined libs-base and libs-corebase, noting their relationship with other libraries.
- Tracing GUI Interactions: By following function calls between libs-gui and libs-back, we mapped out how UI elements are rendered.
- Sequence Diagrams for Key Operations: We analyzed how a GNUstep application launches and how windows are rendered, which helped confirm the roles of various subsystems.

5.3 Architectural Insights Gained

Through this process, we uncovered:

- A layered architecture with clear separation of concerns (core system, GUI, and rendering backends).
- The Model-View-Controller (MVC) pattern being central to libs-gui, enforcing separation of logic and presentation.
- The importance of platform abstraction via libs-back, ensuring compatibility across different operating systems.

This derivation process allowed us to piece together GNUstep's conceptual model without relying solely on existing architectural diagrams, reinforcing our understanding of its modular and extensible design.

6. Design Patterns and Key Insights

GNUstep’s conceptual architecture uses well-established software design principles. This ensures the software is modular, extensible, and platform-independent. There are multiple architecture patterns used to improve maintainability, facilitate code reuse, and support cross-platform compatibility

6.1 Architectural Design Patterns in GNUstep

One of the fundamental patterns that shapes GNUstep is the Model-View-Controller (MVC) paradigm. In GNUstep, MVC is used to help the framework manage user interface components and application logic. The Model encapsulates the core data structures and system logic, mostly implemented within libs-base. Next, the View is responsible for graphical rendering which is defined in libs-gui. To ensure cross-platform support, libs-gui interacts with libs-back. Finally, the Controller deals with user interactions and application behavior which links the Model and View layers together. The way GNUstep uses a separation creates a well-structured approach to UI development, this ensures that modifications to the user interface does not affect any underlying application logic, and vice versa.

Another pattern that is used extensively by GNUstep is the Observer pattern, which is particularly useful in its GUI event-handling mechanisms. Components in libs-gui are designed to respond dynamically to changes in underlying application states, reducing the need for manual UI updates. This pattern is crucial for event-driven programming, as it allows views to subscribe to changes in data models and update when needed.

The Factory pattern is another strong design choice which enabled the creation of UI elements dynamically without exposing object instantiation details to the developer. This approach is particularly beneficial when creating reusable UI components, as it ensures flexibility and consistency in object creation. GNUstep allows different UI components to be generated efficiently based on runtime conditions by abstracting instantiation logic.

A defining characteristic of GNUstep is its use of layered architecture. This enforces a separation between different subsystems so they do not have to interact with each other. This makes it so specific details are contained within specific layers. An example of this is that libs-base provides fundamental system services, libs-gui manages the graphical interface, and libs-back abstracts the rendering backend. This approach ensures that platform-specific details are contained within libs-back, allowing the rest of the system to remain unaware of the underlying operating system.

6.2 Alternative Architectural Approaches and Rationale for Rejection

There are several other architecture models that were considered in the creation of GNUstep that were overall rejected for the current design of the framework. Initially, a monolithic architecture, which is an architecture where libs-gui and libs-back are tightly integrated, was considered for the main architecture. This approach had the ability to simplify system interactions, at the expense of reducing flexibility and maintainability. The architecture used is modular which allows GNUstep to swap rendering backends without modifying GUI logic, making it a more sustainable and long-term solution.

Direct OS API calls were another potential alternative for rendering, bypassing libs-back altogether. This would have benefits such as improving performance by removing an abstraction layer, however, it would have caused issues with cross-platform compatibility. GNUstep ensures that UI code remains independent of platform-specific implementations, greatly reducing development overhead by using libs-back as an intermediary.

Another change that was considered was a transition from Objective-C to a language such as C++, since C++ is so widely used. However, this created issues since C++ is incompatible with the foundation of GNUstep, OpenStep. This change would have caused the need to extensively rewrite existing libraries. The decision to stick with Objective-C allowed the use of OpenStep and maximized compatibility with macOS development environments.

6.3 Key Architectural Insights

When analysing GNUstep's conceptual architecture, there are several key insights on the structure and design philosophy of the framework. The use of a modular design ensures that individual components can be developed, tested, and maintained independently, allowing scalability and long-term sustainability. Since there is a strict separation between UI logic and rendering, it gave room to future adaptations to new graphics technologies without major code overhauls and enhanced the maintainability.

Furthermore, the abstraction of platform-specific functionality in libs-back reinforces the importance of portability in software architecture. GNUstep maintains flexibility, enabling applications to run across multiple operating systems with minimal modifications by isolating OS-dependent behavior. The use of well-established design patterns, such as MVC and Observer, gives the ability of code reusability and reduces maintenance complexity, reinforcing the architectural strength of the system.

Overall, the design decisions for GNUstep focuses and prioritizes on modularity, cross-platform compatibility, and software longevity, making it a strong and flexible framework for GUI application development.

7. Conclusion and Future Considerations

7.1 Summary of Findings

Throughout this report, we have explained how the layered architecture of GNUstep, written on top of the OpenStep spec, enables cross-platform development. By breaking down its core subsystems—libs-base, libs-corebase, libs-gui, libs-back, and supporting tools like gorm—into their constituent parts, we have shown how they contribute to its portability and modularity. The framework follows well-known design patterns like Model-View-Controller (MVC), which ensures clean separation of logic from presentation. In short, GNUstep has proven itself to be a universal and well-ordered framework; however, some details need modernization for better functionality.

7.2 Potential Areas for Improvement

Though GNUstep has a sound foundation, there are some significant areas that require attention. One major limitation is the dependence on outdated rendering engines such as X11 and Windows GDI that could be updated to incorporate newer graphics APIs such as Vulkan or Metal to make it perform better and more efficiently. Also, while GNUstep does have multithreading support, its concurrency model might not be designed to make the UI remain responsive in computationally intensive applications. Documentation is also very important—additional onboarding documentation for developers, particularly those without prior Objective-C or OpenStep experience, could make GNUstep easier to use and encourage more contributors.

7.3 Implications for Concrete Architecture

As we move forward from studying the conceptual structure to considering actual realization (A2), we will be witnessing to what degree the code adheres to the idealized plan. While we may wish to see the same broad form working itself out, we do expect to observe some optimizations, drifts, or hacks having been introduced due to performance requirements. Indicating where these deviations exist will tell us where we need to polish the framework while preserving its strength. Lastly, this analysis will further clarify how GNUstep operates in real-life situations and how it can be made to remain current and effective.

8. External Interfaces

GNUstep interacts with various external systems through well-defined interfaces:

- Graphical User Interfaces (GUIs): Applications interact with users via windows and controls managed by libs-gui and rendered through libs-back.
- File System: Configuration and resource files (e.g., .plist, .gorm) store application settings and UI layouts. Furthermore, libs-base offers utilities for interacting with files (NSFileManager, NSFileHandle, etc).
- Networking: libs-base provides networking utilities for HTTP requests, sockets, and inter-process communication.

This modular design ensures flexibility while maintaining a consistent interface across different operating systems.

These revisions should better reflect how you derived the architecture and keep the external interfaces section concise. Let me know if you need any refinements!

9. Data Dictionary

Term	**Definition**
OpenStep	A set of APIs originally developed by NeXT, forming the foundation for macOS's Cocoa framework.
GNUstep	An open-source reimplementation of the OpenStep API for cross-platform application development.
libs-base	The core system library providing fundamental data structures and runtime utilities.
libs–corebase	A lower-level foundation library that provides additional system utilities.
libs-gui	The user interface library containing UI components and widgets.
libs-back	The rendering backend library responsible for drawing UI elements using platform-specific graphics APIs.
Gorm	GNUstep's interface builder, similar to Apple's Interface Builder.
MVC	Model-View-Controller, a software design pattern that separates application logic from the UI.
X11	A windowing system commonly used on Unix-based operating systems.
GDI	The Windows Graphics Device Interface, used for rendering on Windows systems.

10. Naming Conventions

GNUstep follows a strict naming convention derived from its OpenStep roots. The conventions ensure consistency and compatibility with macOS development practices.

Class Naming:

- NS Prefix: Most classes in GNUstep use the NS (NeXTSTEP) prefix, e.g., NSString, NSArray, NSWindow.
- GS Prefix: GNUstep-specific classes use the GS prefix, distinguishing them from macOS counterparts.

Method Naming: Methods in Objective-C follow camelCase notation.

Example: - (void)loadApplicationWindow;

File Naming:

- Header files (.h) are named after their respective classes.
- Implementation files (.m) contain the method definitions for Objective-C classes.
- Resource files (.gorm, .plist) store UI layouts and configurations.

11. Lessons Learned

One of the biggest challenges we faced in uncovering GNUstep's architecture was the lack of centralized, up-to-date documentation. While there are various resources available, including official GNUstep documentation, mailing lists, and source code repositories, there is no single comprehensive source outlining the full architectural structure. This meant that we had to piece together information from multiple sources, often verifying our findings by inspecting the actual codebase.

Another challenge was the complexity of GNUstep's modular design. Unlike monolithic frameworks, GNUstep follows a layered approach, with clear separations between foundation libraries, UI components, and rendering backends. While this makes GNUstep flexible and portable, it also meant that understanding interactions between components required tracing dependencies across multiple libraries (e.g., libs-gui calls libs-back for rendering).

What we would do differently:

- Start by mapping dependencies before diving into code: Initially, we focused on reading documentation and source code separately, but in hindsight, tracing dependencies between libraries first (e.g., through class references and function calls) would have helped build a clearer mental model earlier in the process.

- Use tools to analyze source code structure: Instead of manually exploring repositories, static analysis tools (e.g., Doxygen, dependency graph generators) could have sped up the process of identifying key class interactions.
- Look at real-world GNUstep applications earlier: Examining how existing applications use the framework (e.g., their import patterns and calls to GNUstep APIs) would have provided quicker insights into how GNUstep's components interact in practice.

12. References

[1] GNUstep Project, "GNUstep Documentation," Available: https://www.gnustep.org/developers/documentation.html. [Accessed: Feb. 7, 2025].

[2] GNUstep Project, "GNUstep Official Website," Available: https://gnustep.github.io/. [Accessed: Feb. 13, 2025].

[3] GNUstep Project, "Developer Tools Overview," Available: https://www.gnustep.org/experience/DeveloperTools.html. [Accessed: Feb. 9, 2025].

[4] GNUstep Project, "GNUstep GUI Design Guide," Available: https://mediawiki.gnustep.org/index.php/Main_Page. [Accessed: Feb. 7, 2025].

[5] GNUstep GitHub Repository, Available: https://github.com/gnustep. [Accessed: Feb. 13, 2025].

[6] GNUstep Project, "OpenStep Specification," Available: https://mediawiki.gnustep.org/index.php/OpenStep. [Accessed: Feb. 8, 2025].

[7] Wikipedia, "OpenStep – Overview and History," Available: https://en.wikipedia.org/wiki/OpenStep. [Accessed: Feb. 11, 2025].

[8] Preterhuman Archive, "OpenStep Development Tools," Available: https://cdn.preterhuman.net/texts/computing/nextstep-openstep/802-2110%20-%20OpenStep%20Development%20Tools.pdf. [Accessed: Feb. 12, 2025].

[9] GNUstep Project, "OpenStep User Interface Guidelines," Available: https://www.gnustep.org/resources/documentation/OpenStepUserInterfaceGuidelines.pdf. [Accessed: Feb. 13, 2025].

[10] UML Diagrams, "UML 2.0 Overview and Documentation," Available: https://www.uml-diagrams.org/uml-2.0.html. [Accessed: Feb. 9, 2025].