Conceptual Architecture of GNUstep

Team GNUtered

Logan Philip - 21lfp1@queensu.ca
Alex Nguyen - 22thn@queensu.ca
Ryan Waterson - 22rdw@queensu.ca
William Shaver - 21wgs6@queensu.ca
Nguyen Nguyen - 22nkn@queensu.ca
Gia Nguyen - 21hghn1@queensu.ca

Abstract

GNUstep is an open-source, cross-platform framework designed to replicate and extend the functionality of Apple's Cocoa (formerly OpenStep) environment, enabling the development of portable desktop applications.

Originating from the OpenStep specification published by NeXT in 1994, GNUstep's architecture is structured into five modular layers: the Operating System Layer, Windowing System Layer, Foundation Layer, Interface Layer, and Applications Layer. These layers integrate the Model-View-Controller (MVC) pattern with platform-agnostic subsystems, ensuring a clear separation of concerns while maintaining cross-platform consistency.

Key components of GNUstep include libs-base (core utilities, data management), libs-gui (graphical interface components), libs-back (low-level rendering abstraction), libs-corebase (CoreFoundation compatibility), and Gorm (visual UI builder). The framework's layered dependencies prioritize modularity, allowing the components, like libs-gui and libs-back, to collaborate seamlessly while abstracting away platform-specific complexities.

GNUstep provides use cases such as designing interfaces with Gorm or parsing property lists via libs-corebase demonstrate GNUstep's capabilities and interoperability. By utilizing modular, Cocoa-compatible architecture, GNUstep ensures long-term application compatibility, simplified maintenance, and efficient cross-platform development.

Introduction and Overview

GNUstep is an open-source, cross-platform framework for desktop application development, designed to replicate and extend the functionality of Apple's Cocoa (formerly OpenStep) environment. The OpenStep specification was published by NeXT in a first draft in the summer of 1994. Apple acquired NeXT on February 4, 1997, and subsequently used OpenStep as the foundation for macOS [1].

GNUstep's layered architecture integrates the Model-View-Controller (MVC) pattern with modular, platform-agnostic subsystems, ensuring a clear separation of concerns. It consists of five layers. At the base, the **Operating System layer** provides core system services for Linux, Windows, macOS, and BSD. Above it, the **Windowing System** and **Foundation Layer** handle low-level rendering (X11, Win32) and provide essential libraries for data management, cross-platform consistency, and build automation (GNUstep Base, GNUstep Make). The Application **Interface layer** (GNUstep GUI and GNUstep Back) bridges system operations and user applications, abstracting UI rendering to ensure consistency across platforms. Finally, the **Applications layer** hosts software built on GNUstep, ensuring long-term compatibility and easy maintenance [2].

Under subsystems we discuss the following major GNUstep components: GNUstep's architecture includes several key components that make development smoother. **libs-back**[3] handles low-level rendering, working with different windowing systems like X11 and Win32. **libs-base**[4] provides essential utilities for data handling, file I/O, and cross-platform support. **libs-corebase**[5] extends this functionality with additional Cocoa-compatible features. On the UI side, **libs-gui**[6] powers the graphical user interface, offering widgets and layout tools. To simplify interface design, **Gorm**[7] acts as a visual GUI builder, letting developers create and edit application interfaces without writing code from scratch.

Derivation Process

To begin learning about GNUstep, we first started with the provided Wikipedia page for more general information to create a foundation of what GNUstep is, how it's used, and what it's used for. We then went to GNUstep's website to gain a deeper understanding of GNUstep, including its different layers, like the Foundation and Interface layers. We looked into a page containing a system overview of GNUstep that detailed GNUsteps layers and their functions within the system. This was a great insight into the functions of their layers, their interactions with each other, and their relation to the computer and the user. The GNUstep documentation also provided us with information on the various libraries GNUstep uses, such as the Base, CoreBase, and GUI libraries.

To further understand the architecture of GNUstep, and to help us create the diagram of the conceptual architecture, we consulted the GNUstep GitHub repository. The GitHub repositories allowed us to directly look at the workings and dependencies of GNUstep, and cross-reference them with the documentation to produce what we believe is an accurate representation of GNUsteps architecture. When researching use cases, we first looked into the documentation for Gorm, and with that we were able to get a rough idea of how it interacts with the other components. Based on our knowledge of the dependencies for each component, we were able to construct a sequence diagram that respects the functionalities and dependencies of each component.

However, looking for information regarding the CoreBase library (libs-corebase) was more difficult than the others, but we were determined to include each component at least once between our two use cases. By looking through libs-base's GitHub repository, we were able to get some idea of how it functions and were able to construct a minor use case that displays an interaction with its dependency, libs-base.

Conceptual Architecture

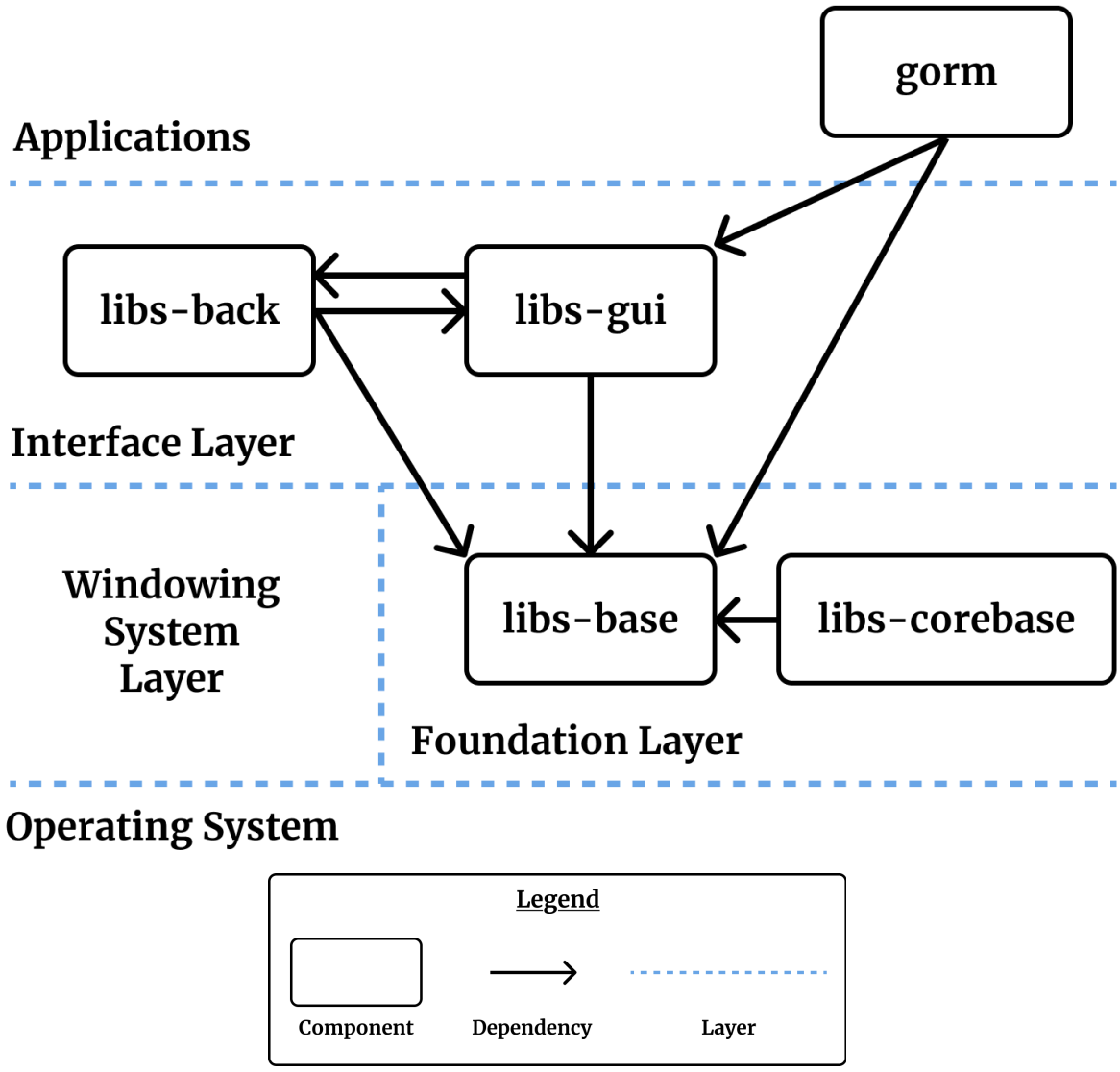

Figure 1. The proposed conceptual architecture of GNUstep.

Above is our proposed conceptual architecture of GNUstep. The architecture style is a hybrid of layered and modular object-oriented styles.

Our architecture consists of five distinct layers: the Operating System Layer, the Windowing System Layer, the Foundation Layer, the Interface Layer, and the Applications Layer. The Operating System Layer handles low-level platform-specific tasks like file systems, memory, and hardware access. The Windowing System Layer serves as a bridge between the operating system and the interface, managing graphics and input events. The Foundation Layer builds on these, providing essential classes and data structures. The Interface Layer focuses on rendering graphics and managing user interaction through libs-back and libs-gui. Finally, the Applications Layer

contains user-facing applications like Gorm, which provides tools for building and managing user interfaces.[2]

The modular portion of the architecture is seen in how libs-gui and libs-back work together. libs-back provides backend support for rendering and graphics, while libs-gui builds on that to offer complete user interface components. This allows Gorm, the application for designing user interfaces, to function without dealing directly with low-level system operations.

An alternative architecture we considered was the Client-Server Architecture, where libs-gui and applications like Gorm would act as clients requesting services from libs-base and libs-back, which would act as servers. While this design would improve modularity and scalability, we decided against it because it would add unnecessary complexity and reduce efficiency for a local system like GNUstep. The layered approach provides a simpler, more direct structure, better suited for local environments without the overhead of networked communication.

Subsystems

GNUstep is a free software framework that implements the OpenStep specification and supports Apple's Cocoa API. Its modular architecture ensures scalability and flexibility, including the following key components:

1. libs-base (Foundation Library)

libs-base is the core building block for Objective-C applications in GNUstep. It provides essential data structures, utilities, and system interaction features to make development smoother. The fundamental classes for this libs will consist of NSArray, NSDictionary, NSString, and NSNumber. Their main point is that managing data becomes intuitive and efficient. File system tasks are handled through NSFileManager, allowing us to navigate, create, and modify files easily. For multitasking, NSThread, NSOperationQueue, and Grand Central Dispatch (GCD) will help to manage parallel execution efficiently. Networking is taken care of by NSURLSession, making HTTP requests straightforward, while built-in serialization formats like PLIST and JSON simplify data encoding. Additionally, NSDate and NSCalendar provide robust tools for handling dates and times.

2. libs-gui (AppKit Library)

libs-gui is the graphical user interface (GUI) library for GNUstep. At first, it was designed to help build macOS-style applications. It offers essential UI components like NSWindow for managing windows and NSView for rendering content. There are also interactive controls, including NSTableView, NSButton, and NSTextField, to create rich user interfaces. The library ensures smooth user interactions with NSResponder and NSEvent handling events in a way that feels natural. On the other hand, advanced rendering is powered by Display PostScript, allowing vector-based

graphics. Moreover, these tools will also provide other useful features, including NSMenu and NSToolbar for navigation and built-in drag-and-drop support to facilitate seamless data transfers between apps.

3. libs-back (Display Backend and Window System)

libs-back is the backbone of cross-platform compatibility for GNUstep applications. Its main purpose is to ensure that they run smoothly regardless of the underlying system—whether it's X11, Wayland, or Windows. This tool acts as an abstraction layer, handling the heavy lifting required to adapt applications to different graphics environments. One advantage it has is that it doesn't require developers to rewrite their code for each platform. libs-back relies on Cairo and OpenGL for rendering, enabling hardware acceleration and delivering high-performance visuals. This means that applications benefit from enhanced animations, graphics and rendering, even on low-end systems. Another essential feature is multi-monitor support. Whether the application is launched on a single monitor, multiple monitors, or high-resolution setups, libs-back adapts animations to the different screen sizes and resolutions available to each. This adaptability ensures that GNUstep applications remain responsive and visually consistent, regardless of the environment.

4. libs-corebase (CoreFoundation Compatibility)

libs-corebase is the bridge that helps GNUstep work more seamlessly with Apple's CoreFoundation framework. Even though it is an optional package, if one is developing for both macOS and GNUstep, this library makes life a lot easier by reducing the amount of code we need to rewrite when switching between platforms. libs-corebase provides essential low-level utilities for managing memory, working with strings, handling collections, and storing data [8]. It also includes support for important CoreFoundation APIs like CFString, CFDictionary, and CFRunLoop, which means that existing macOS code can often be reused with minimal tweaks. For developers, this leads to less time struggling with compatibility issues and more time focusing on building features. Instead of re-implementing common functionality from scratch, developers can rely on libs-corebase to handle the heavy lifting.

5. Gorm (Interface Builder Alternative)

Gorm is GNUstep's answer to Apple's Interface Builder. It's a visual design tool that lets developers create user interfaces through an intuitive drag-and-drop environment, instead of manually writing code to lay out our user interface. One can easily arrange UI elements, manage object relationships, and connect components to controllers—all without writing code manually. Once the design is complete, Gorm generates Objective-C code that integrates seamlessly into an application, making the development process faster and more efficient. It is a beneficial tool for simplifying UI design while maintaining full control over the underlying code.

Dependencies

GNUstep's architecture is built in layers, with each part depending on the ones below it to function smoothly. At the foundation, libs-base acts as an essential system utilities and higher-level programming features. On top of that, libs-back handles rendering and graphics, working closely with libs-base to manage system interactions. On the other hand, libs-gui builds on both libs-base and libs-back, providing the user interface components needed for applications. These two, libs-gui and libs-back, have a mutual dependency, ensuring that UI elements and the rendering backend work together seamlessly. At the application level, Gorm, the UI design tool, depends on libs-gui for visual components and libs-base for core logic. While libs-corebase exists as an optional package that can extend functionality, nothing in the core system strictly requires it. This layered approach keeps the system organized and efficient, making it easier to develop and maintain while ensuring that each component has a clear role.[9]

Control Flow

The control flow in GNUstep is built on both Object Oriented Architecture (OOP) and a Layered Architecture. When a user interacts with a GNUstep application, the event is processed by the libs-gui component. This GUI layer, along with Gorm, is responsible for capturing user interactions and structuring UI elements.

According to the design diagram, the event is then forwarded to the relevant objects for processing once it has been registered. If the action involves application logic, such as fetching data or executing commands, it is handled by libs-base, which provides core utilities like memory management, collections, and file handling. If system-level interactions are required like file I/O, networking, or ensuring compatibility with Cocoa API, the libs-corebase component acts as the bridge between GNUstep and the underlying OS services. Besides, if the event affects the user interface, such as displaying updated content or rendering new elements, the event will be forwarded to the libs-back components which ensure the correct rendering through platform-specific backends like X11, Win32, or Quartz. This layered approach helps GNUstep applications remain portable and consistent across different operating systems.

After processing the event, the UI is updated, and control returns to the main event loop, waiting for the next user action. According to the GNUstep document, GNUstep also supports asynchronous execution and concurrency, allowing background tasks such as file operations and network requests to run without blocking the main application.

Division of Responsibilities

As GNUstep is an open source project, anyone interested in contributing can access GNUstep's GitHub repository and follow the development guidelines outlined in its documentation. The project follows a structured format that components are in a different GitHub repository, allowing contributors to freely choose the part they want to work on.

Contributors usually begin by reviewing the GNUstep documentation and identifying areas where they can contribute. The GNUstep mailing list and IRC channels serve as communication hubs where developers discuss bugs, propose enhancements, and collaborate on ongoing tasks. Every pull request then will be reviewed by the project maintenance team and once it is approved, the changes are merged into the repository.

While contributions are open to all, critical system components, such as interoperability with Cocoa APIs or major changes to platform backends, of the project require contributors to discuss with the project maintainer before implementation. The Gorm GUI builder also follows strict design guidelines to ensure compatibility with GNUstep applications.

Evolution of GNUstep

Originally, in 1993, GNUstep was designed to replicate the NeXTSTEP environment and later adopted the OpenStep specification in 1994. The early versions of GNUstep focused on implementing the fundamental classes of the OpenStep API, including the Foundation Kit, which is the libs-base library now, for its core functionalities such as object management, file handling, and inter-process communication.[10]

In the later versions, the libs-gui library was introduced to handle user interface elements with the goal of mirroring NeXTSTEP's AppKit. Additionally, libs-corebase was developed to bridge GNUstep with system-level features, ensuring compatibility with macOS's Cocoa framework.[11] This allowed applications built with GNUstep to maintain partial interoperability with Apple's ecosystem while remaining open and cross-platform.

In the 2000's, GNUstep expanded beyond Unix-based systems by introducing the libs-back component, which enabled rendering on different platforms such as X11, Win32, and Quartz.[10] Around the same time, Gorm was introduced which was modeled after Apple's Interface Builder with the goal of reducing the need for manual UI coding.[12]

Overtime, there were multiple other releases with minor changes such as bug fixes, improved UI, multithreading enhancement, and miscellaneous improvements and optimizations. Notably, the Objective-C runtime within GNUstep has also seen major updates, bringing it closer in line with modern Objective-C features. This includes

support for Automatic Reference Counting (ARC), which simplifies memory management and reduces the burden on developers.[10]

External Interfaces

GNUstep relies on well-defined external interfaces to deliver cross-platform functionality.

Graphical User Interface (GUI):
The libs-gui library and Gorm (GUI builder) form the primary interface for user interaction. Developers design applications using drag-and-drop tools in Gorm, which generates Objective-C code for windows (e.g., NSWindow), buttons (NSButton), and other UI elements. These components interface with libs-back to render graphics on platforms like X11 (Linux/BSD), Win32 (Windows), or Quartz (macOS). For example, a button click in Gorm triggers an NSAction event, processed by libs-gui and relayed to application logic via libs-base.

Operating System Abstraction:
The libs-base and libs-corebase libraries abstract OS-specific tasks. They translate GNUstep's APIs into POSIX calls (Linux/BSD), Win32 (Windows), or Cocoa (macOS) for file I/O (NSFileManager), memory management (NSAutoreleasePool), and threading (NSThread). For instance, NSFileManager uses CreateFile on Windows and open() on Linux to read files, ensuring developers work with a unified interface.

Windowing System Backends (libs-back):
libs-back serves as the bridge to native windowing systems. It converts high-level NSView rendering commands into platform-specific operations: X11/Wayland: Draws UI elements via Xlib or Cairo; Win32: Uses GDI for window management and Direct2D for graphics; macOS: Leverages Quartz for pixel-perfect rendering.This allows a single NSWindow declaration to adapt to any OS without code changes.

Graphics and Rendering Libraries:
Complex visuals are delegated to Cairo (vector graphics) and OpenGL (3D acceleration). For example, NSTextView uses Cairo for anti-aliased text, while animations in Gorm leverage OpenGL shaders. These libraries ensure consistent output across hardware, whether rendering a gradient on Windows or a transparent overlay on macOS.

Build and Dependency Management:
The GNUstep Make tool interfaces with compilers (GCC, Clang) and package managers to automate builds. It resolves dependencies like libobjc2 (Objective-C runtime) and libffi (foreign function calls), enabling cross-platform compilation. Developers write a single GNUmakefile, and the tool adapts it for Linux RPMs, Windows DLLs, or macOS frameworks.

Input Handling:
User inputs (mouse clicks, keystrokes) are captured by the native windowing system (e.g., X11 events) and converted into NSEvent objects by libs-back. The libs-gui layer processes these through NSResponder chains, allowing a Linux touchpad gesture and a Windows mouse click to trigger the same mouseDown: method in an application

Use Cases

Use Case 1: Designing an Interface with Gorm

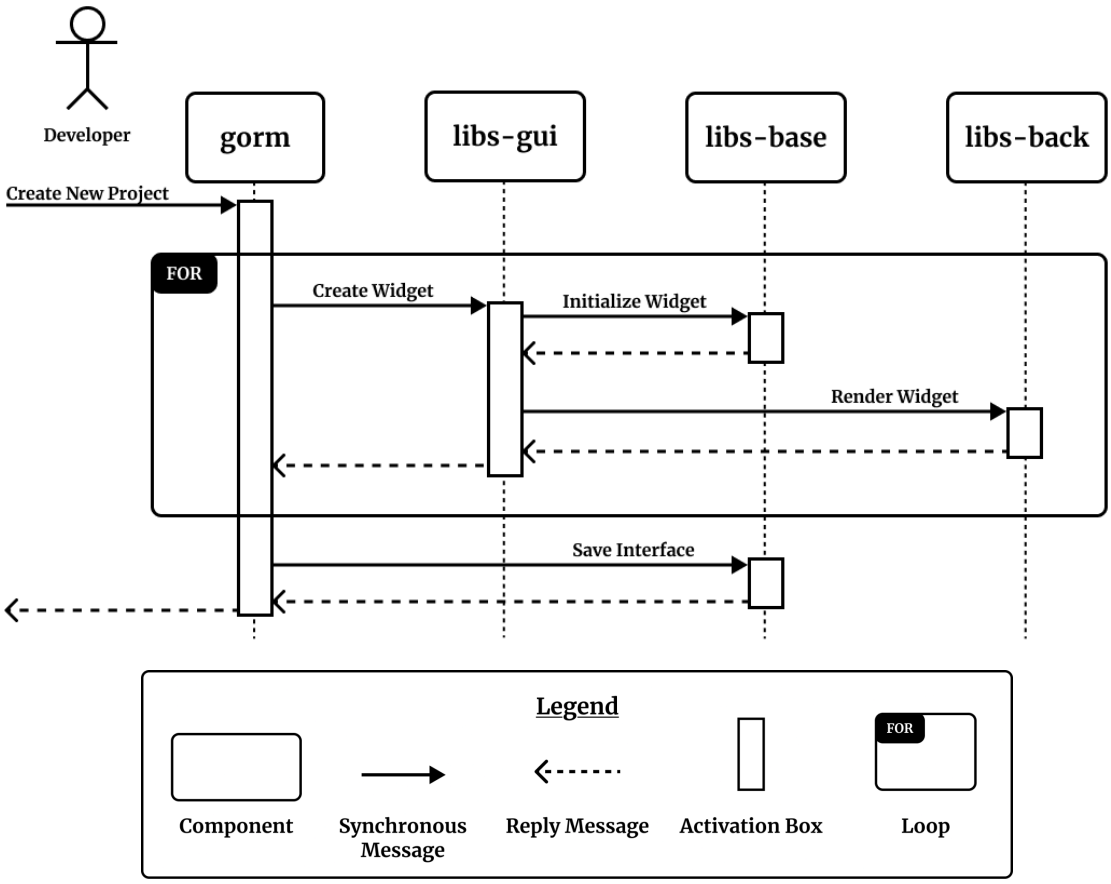

Figure 2. A sequence diagram for the use case of designing an interface using Gorm.

This is a use case involving a developer designing an interface with widgets using Gorm. This use case was chosen as using Gorm to create GNUstep apps is one of the most common methods, and this use case displays the sequence of steps as we understand them based on the documentation.

Using Gorm, the user starts by creating a new project. Then, for as long as they decide, the user can create new widgets for the interface. To create a widget, Gorm sends a request to libs-gui, which in turn goes to libs-base to initialize the new widget. After

returning the newly initialized widget to libs-gui, it is then passed to libs-back where it is rendered to the display. The user can repeat this with any kind of widget they need to add to the interface until they are satisfied. When finished, a request to save the interface is requested by Gorm to libs-base.

Use Case 2: Parsing a Property List with libs-corebase

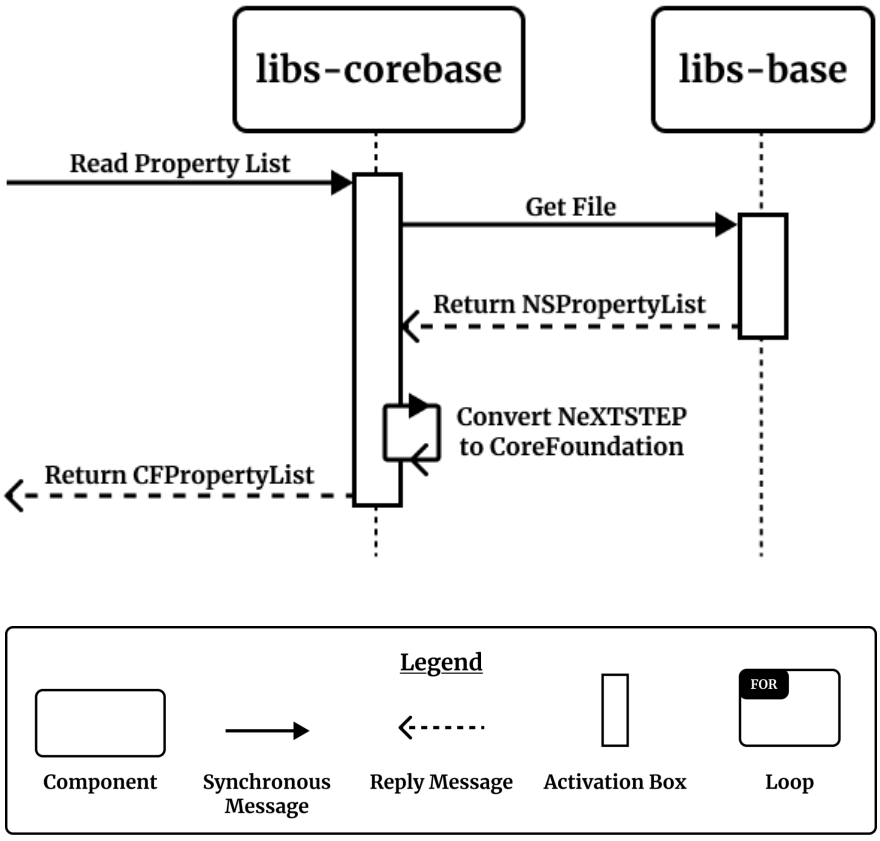

Figure 3. A sequence diagram for the use case of designing an interface using Gorm.

The second use case shows how libs-corebase can be used for interoperability with Apple's Core Foundation framework by parsing a property list. This use case was chosen to demonstrate a potential application of the CoreBase library, which may otherwise be an uncommon occurrence.

Upon a GNUstep application using the Core Foundation framework's request to read a property list from a file, libs-corebase turns to libs-base to retrieve the file and return an NSPropertyList. The property list as returned by libs-base is a NeXTSTEP Object however, and through libs-corebase, is converted to its corresponding CoreFoundation counterpart to be returned to the original application.

Data Dictionary

Conceptual architecture: A conceptual architecture shows the structure of a system and how the different parts of that system interact with each other. The conceptual architecture provides an overview of how the system works, but not its implementation.

Windowing System: A windowing system is a graphical user interface that manages different parts of display screens. [13]

Foundation Layer: The foundation layer is a layer in GNUsteps architecture that provides needed libraries for things like data management, cross-platform consistency, and build automation.

Interface Layer: The interface layer is a layer in GNUsteps architecture that connects user applications to the system.

Applications layer: The applications layer is a layer in GNUsteps architecture that hosts the software made through GNUstep.

Quartz: The core graphics and windowing system in macOS, responsible for rendering 2D graphics, managing windows, and visual effects. [14]

Widget: Widgets are small dynamic displays of information on an app or website.

Open source: Software that is open source is publicly available and can be used by anyone for free.

Naming Conventions

Application programming interface/API: An application programming interface is a set of rules that allows different pieces of software to interact and communicate with each other.

Model-View-Controller/MVC: A software design pattern for implementing user interfaces

Graphical user interface/GUI: A graphical user interface is what allows the user to interact with the system without having to rely on text commands.

User interface/UI: User interface is how a user interacts with a computer.

Operating system/OS: An operating system is the software that handles resource management and the hardware for a computer.

Gorm: “Graphical Object Relationship Modeller”, GNUstep’s GUI builder.

NS: “NeXTSTEP”: Nextstep is an object-oriented, multitasking operating system that has since been discontinued.

CF: “CoreFoundation”, CoreFoundation is a framework to provide software services to applications, and their services and environments. It can also be used for abstracting some data types, along with some other utilities. [15]

libobjc2: A package that enables objective c runtime [16]

libffi: A library for foreign function interface. Foreign function interface allows code written in one language to call on code that has been written in another language. [17]

Conclusions

In conclusion, GNUstep is a powerful open-source framework for developing cross-platform desktop applications. It achieves this by following a hybrid of layered and modular object-oriented architectural styles, ensuring flexibility and long-term compatibility. The combination of these architectural styles allows GNUstep to maintain a well-structured and scalable system, where each layer and subsystem plays a distinct role without interfering with others. This design also ensures cross-platform consistency by abstracting platform-specific details through the Windowing System Layer and libs-back, making applications portable across various operating systems.

As shown in our research and analysis, GNUstep’s core components (libs-base, libs-gui, libs-back, and libs-corebase) work together seamlessly to provide essential services such as data management, UI rendering, and platform abstraction. Tools like Gorm simplify interface design, making GNUstep a practical solution for developers aiming for Cocoa-compatible applications on multiple platforms. By identifying the relationships between these components and understanding their dependencies, we were able to propose and visualize GNUstep’s conceptual architecture, highlighting how it ensures both flexibility and efficiency for future development.

Lessons Learned

While formulating this report, our group encountered several challenges when designing the conceptual architecture for GNUstep. One of the main difficulties was understanding how the different layers and components fit together within GNUstep’s hybrid layered and object-oriented architecture. The provided documentation on GNUstep was helpful in outlining individual components like libs-base, libs-gui, and libs-back, but it was harder to figure out how these components interact and how to place them within the correct layers. Additionally, learning how to use a new tool to design our conceptual architecture diagram was time-consuming, as we had to experiment and iterate multiple times to represent the dependencies clearly.

To better understand the architecture, we explored the GNUstep documentation in more depth to confirm how the components depend on each other. This helped us clarify the relationships between key subsystems, such as libs-gui depending on libs-back for rendering and libs-corebase extending functionality from libs-base. After several group discussions and weekly in-person meetings, we finalized our

conceptual architecture, determined how each component fit into the system, and completed our diagram and descriptions with more confidence.

Overall, our group worked well together and communicated regularly through our group chat and during our weekly meetings. However, we could have done a better job planning ahead and organizing tasks early on. Miscommunication at times slowed progress, especially when merging different sections of the report. Setting soft deadlines helped us stay on track, but better initial planning would have reduced confusion and improved efficiency.

An area we could improve on is collaborating more at the start, particularly on conceptual architecture. Although we assigned sections early, not everyone had a complete picture of how the components worked together, which caused some confusion later. In the future, collaborating on the architecture design as a full group before writing the report would help avoid these issues. Additionally, having all group members contribute to both the report and presentation would ensure a more cohesive final product and give everyone a stronger understanding of the project.

References

[1] Rand, Paul, et al. "NeXT." *Wikipedia*, https://en.wikipedia.org/wiki/NeXT.

[2] "GNUstep System Overview." *GNUstep Library*, http://gnustep.made-it.com/SystemOverview/index.html.

[3] "libs-back Repository." *GitHub*, 11 February 2025, https://github.com/gnustep/libs-back.

[4] "libs-base Repository." *GitHub*, https://github.com/gnustep/libs-base.

[5] "libs-corebase Repository." *GitHub*, https://github.com/gnustep/libs-corebase.

[6] "libs-gui Repository." *GitHub*, https://github.com/gnustep/libs-gui.

[7] "GNUstep Developer Tools: Gorm." *GNUstep*, https://www.gnustep.org/experience/Gorm.html.

[8] "GNUstep." *Wikipedia*, https://en.wikipedia.org/wiki/GNUstep.

[9] "GNUstep Suite." GNUstep Wiki, https://mediawiki.gnustep.org/index.php/GNUstep_Suite.

[10] "GNUstep History." *GNUstep Library*, https://gnustep.made-it.com/Guides/History.html.

[11] "The OpenStep Story." *GNUstep.org*, https://www.gnustep.org/information/openstep.html.

[12] "Developer Documentation." *GNUstep.org*, https://www.gnustep.org.

[13] Wikipedia contributors. (2024, October 19). Windowing system. *Wikipedia*, https://en.wikipedia.org/wiki/Windowing_system.

[14] Wikipedia contributors. (2024, September 18). Quartz (graphics layer). *Wikipedia*, https://en.wikipedia.org/wiki/Quartz_(graphics_layer).

[15] "Core Foundation." *Apple Developer*, https://developer.apple.com/documentation/corefoundation.

[16] "libobjc2 Repository." *Github*, https://github.com/gnustep/libobjc2.

[17] "libffi Repository." *Github*, https://github.com/libffi/libffi.