GNUstep Conceptual Report

Group 14: ArchiTECHS

Kamana Chapagain 21kc77@queensu.ca

Sareena Shrestha 21ss377@queensu.ca

Yashasvi Pradhan 21yp26@queensu.ca

Justin Li 21jjl13@queensu.ca

Saachi Singh 22ss62@queensu.ca

Shreya Menon 22sm47@queensu.ca

CISC 322/326

Assignment 1: Conceptual Architecture Report

February 14th, 2025

Table of Contents

1. Abstract 3
2. Introduction and Overview 3
3. Derivation Process 4
4. Subsystems 5
- Libs-Base 5
- Libs-CoreBase 5
- Libs-GUI 5
- Libs-Back 6
- Gorm 6

5. System Evolution 7
- History of GNUstep 7
- Platform Support and Compatibility 7
- Changes in Core Libraries 7
- Developer Tools and Community Contribution 7
- Modern Development Practices 7

6. Control and Data Flow 8
7. Concurrency 9
- Thread Management 9
- Synchronization 9
- Concurrency in GUI Applications 9
- Implications for Developers 10

8. Implications for the Division of Responsibilities 10
- Layered and Modular Development 10
- Collaboration and Dependency Management 10
- Scalability and Maintainability 11

9. Use Cases 11
- Sequence Diagram to Run the Application 11
- Designing a User Interface with Gorm 12

10. External Interfaces 12
- Graphical User Interface (GUI) 12
- Operating System (OS) 13
- File System Interaction 13
- Database Systems 14
- Plugins and External Libraries 14

11. Data Dictionary 14
12. Naming Conventions 14
13. Conclusion 14
14. Lessons Learned 15
15. References 15

1. Abstract

This report reviews GNUstep's architecture, a cross-platform framework for building desktop and server applications. We found that GNUstep has a modular, object-oriented structure, which makes it flexible, scalable, and easy to maintain. The system is divided into layers: libs-base for core system services, libs-corebase for low-level utilities and optimizations, libs-gui for managing user interfaces, and libs-back for rendering on different platforms. This clear separation of responsibilities helps keep development and debugging straightforward.

GNUstep uses a layered and object-oriented architecture to structure components efficiently. Its modular design allows components to function independently, improving scalability and maintenance. The framework supports multi-threading, enabling background tasks without slowing the user interface (UI). Libs-base manages threads and synchronization, which ensures smooth and safe resource access. With this clear structure and strong concurrency support GNUstep enables fast, responsive and scalable cross-platform applications.

GNUstep is a modular, cross-platform framework that integrates with external systems and supports Unix-like and Windows environments. It handles file operations, memory management, and networking while adapting its rendering with Cairo, OpenGL, and Winlib. Modern software practices improve its performance, usability, and stability. GNUstep also provides tools like Gorm for UI design and Renaissance for XML-based layouts, simplifying cross-platform development. Additionally, it supports external databases and plugins, allowing developers to extend application functionality as needed.

In summary, GNUstep provides a robust and flexible framework for developers looking to create cross-platform applications. Its object-oriented, layered architecture, combined with its powerful tools and modularity, makes it a viable choice for developing maintainable, high-performance applications across diverse operating systems.

2. Introduction and Overview

GNUstep is a cross-platform framework designed for desktop and server application development. It provides an open-source implementation of the OpenStep API, which serves as the foundation for Apple’s Cocoa framework (Gui, n.d.). By leveraging Objective-C and a rich set of reusable components, GNUstep allows developers to build applications efficiently across different operating systems, including Unix-like platforms and Windows (Wikimedia Foundation, n.d.). Originally based on NeXTSTEP and OpenStep, GNUstep maintains compatibility with Apple’s Cocoa API, making it an alternative for developers who want to create portable GUI applications without being locked into Apple’s ecosystem. It provides a set of modular libraries and tools that support both graphical and non-graphic development (GNUstep, n.d.). GNUstep is made up of numerous major components, each of which plays a distinct role in the system's operation. The components explored include libs-base, libs-corebase, libs-gui, libs-back and GORM.

GNUstep's architectural approach is primarily based on object-oriented architecture with some elements of layered architecture style. The object-oriented architecture style uses techniques like encapsulation, inheritance, and polymorphism to increase modularity and reusability. It is based on a root class called NSObject and includes a complex hierarchy of classes such as NSString and NSArray which contain data and action (GNUstep, n.d.). The layered architecture style presents itself in the way the libraries are structured. There is a base layer called libs-base that provides fundamental non-GUI functionalities. Then follows the GUI layer that is called libs-gui which is built on top of the base layer, this layer provides the user interface components. The backend layer called libs-back is what summarizes drawing and rendering, interacting with system-specific graphic subsystems. Lastly, the development tools layer, which uses tools like GORM sits at the top and facilitates visual development and project management.

Overall, this report will analyze how these components interact, their dependencies, and how they function together to provide a cohesive development framework. It will also review GNUstep's architectural style in further detail, and review system behaviours such as control and data flow, concurrency management, and external interfaces. Furthermore, this report will also discuss the broader implications of GNUstep's design choices, specifically how they support cross-platform compatibility, modularity and reusability. The conclusion will highlight GNUstep's effectiveness as a flexible, modular framework and reflect on its continued significance in facilitating the creation of cross-platform development.

3. Derivation Process

Our research began by exploring official GNUstep documentation, including the GNUstep website, MediaWiki, and source code on GitHub. We focused on breaking down the system into its core components: libs-corebase, libs-base, libs-gui, libs-back, and Gorm. To get a clearer picture of how everything fits together, we created a dependency diagram, which helped us see that GNUstep is structured in layers. At the bottom is libs-base, which handles essential system-level tasks like memory, files, and networking. Built on top of that is libs-gui, which deals with user interface elements like buttons and windows. Rendering is handled by libs-back, which works with different platforms like X11, Windows, and Cairo to draw everything on the screen. Finally, Gorm sits alongside these as a design tool, letting developers visually create their interfaces, feeding into libs-gui.

As we worked through these relationships, it became clear that GNUstep's architecture is driven by two main styles. The first is Object-Oriented since each part is built as a modular, reusable component that interacts with the others through clear interfaces. The second is Layered, because each library builds on the one below it, with a clear separation of concerns.

We also noticed some event-driven and publish-subscribe patterns, particularly in how libs-gui handles user input and updates parts of the interface. However, we decided these were more about the internal behaviour of the GUI library, and not reflective of GNUstep's overall architecture.

In the end, we concluded that GNUstep's architecture is best described as Object-Oriented with a Layered structure. This approach supports GNUstep's core goals—modularity, cross-platform compatibility, and flexibility—while making it easier for developers to build and maintain applications across different systems.

4. Subsystems

Libs-Base

The libs-base library provides the core non-graphical functionality that GNUstep applications rely on. It includes classes for handling strings, collections, file I/O, networking, threading, event loops, notifications, and distributed objects (Gui, n.d.). It's modelled after Apple's Foundation framework, making it easier for developers to work in Objective-C while keeping their applications portable across Unix-like systems and Windows. Since libs-base handles low-level operations like memory management, system events, and data processing, it serves as the foundation for other key components. Both libs-gui, which manages user interfaces, and libs-back, which handles rendering, rely on libs-base for the system-level services it provides.

Libs-CoreBase

libs-corebase adds a set of low-level, C-based system utilities to GNUstep, focusing on memory management, string handling, and data structures (GNUstep, n.d.). While libs-base offers these features with an Objective-C implementation, libs-corebase works behind the scenes to handle platform-specific optimizations, making sure GNUstep can perform efficiently across different systems. This makes libs-corebase useful for applications needing reliable, low-level system operations. It's not typically used on its own; instead, it works together with libs-base, enhancing performance and compatibility by providing these system-level improvements.

Libs-GUI

The libs-gui library provides the graphical user interface (GUI) framework for GNUstep applications. Written entirely in Objective-C, it includes common UI elements like buttons, text fields, menus, popup lists, browser views, and window management. libs-gui follows Apple's Cocoa framework, implementing an event-driven model that supports modern GUI development. The architecture is divided into two parts: a front-end, which is independent of the platform, and a back end which handles platform-specific rendering. This design allows applications to adapt to the look and feel of the operating system without requiring changes to the application code. libs-gui relies on libs-base for non-graphical operations and also depends on libs-back for rendering its graphical elements on the screen. It is also used by Gorm, GNUstep's interface builder, to supply the visual components that developers arrange when designing application interfaces.

Libs-Back

Libs-back is the rendering engine for GNUstep, responsible for turning abstract drawing commands from libs-gui into actual graphics on the screen. It supports multiple display backends, including X11, Windows GDI, and Cairo, which allows applications to run across different operating systems without changes to the code. While libs-gui deals with UI components, libs-back handles the low-level drawing and rendering that makes those components appear on the screen. Developers don't typically work with libs-back directly; it works behind the scenes with libs-gui to ensure everything is displayed properly, regardless of the platform. It also uses a DPS emulation engine, which helps manage platform-specific graphics calls consistently across different systems. To do its job, libs-back relies on libs-base for low-level system services. libs-gui depends on libs-back to render its user interface elements, making libs-back an essential part of the GNUstep graphics pipeline.

Gorm

Gorm is a drag-and-drop GUI builder designed to simplify interface development. Inspired by NeXT's Interface Builder, Gorm allows developers to design complex user interfaces without writing code. It includes prebuilt UI elements such as buttons, labels, sliders, tables, and text fields, which can be resized, repositioned, and linked to application logic using mouse interactions. Gorm also supports interactive testing, allowing developers to preview and fine-tune their interfaces before fully integrating them into applications. When developers finish building a UI in Gorm, it generates interface description files that libs-gui can dynamically load at runtime, making the process of designing and implementing UIs faster and more efficient.

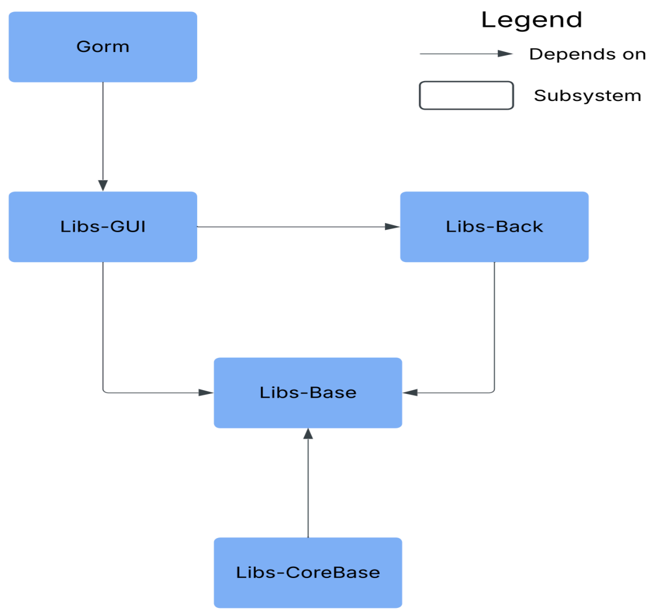

Figure 1. Dependency Diagram of GNUstep's Subsystems

5. System Evolution

History of GNUstep

GNUstep has been evolving since its creation, initially designed as an open-source version of NeXT's OpenStep API to ensure compatibility with macOS's Cocoa framework. Over time, contributions from the open-source community have enhanced its features, performance, and compatibility, allowing it to stay modular, portable, and compatible with both OpenStep and Cocoa APIs. Regular updates have helped maintain a modern and efficient core structure, ensuring GNUstep remains a valuable framework for cross-platform Objective-C development.

Platform Support and Compatibility

A major goal of GNUstep is to run on multiple platforms. It was first built for Unix-like systems, but its Windows support has improved a lot. Enhancements include better Windows GDI-based rendering, improved compatibility with the Win32 API, and stronger support for native Windows system calls. GNUstep also now works better on ARM-based devices, increasing its reach across different hardware types.

Changes in Core Libraries

Libs-base: GNUstep's foundation has been upgraded with better memory management using automatic reference counting (ARC), improved data structures for faster performance, and expanded networking support for modern protocols like IPv6 and TLS.
Libs-gui and Libs-back: The GUI library now supports high-resolution displays, vector-based UI elements, and dynamic themes. The libs-back component has been enhanced to improve rendering with backends like Cairo, Windows GDI, and OpenGL for better graphics performance.
CoreBase Integration: Adding libs-corebase made GNUstep more portable and improved low-level system functions. It also strengthened inter-process communication (IPC), making GNUstep applications work better in multi-threaded and distributed environments.

Developer Tools and Community Contribution

GNUstep's tools and developer support have improved with the following.
Gorm (Graphical Object Relationship Modeller): Inspired by NeXT's Interface Builder, Gorm now has better drag-and-drop UI building, improved XIB/NIB file support, and smoother Objective-C integration.
Renaissance: An XML-based UI layout system that allows for flexible and adaptable interface design.
Better Build Systems: GNUstep now uses modern build tools like CMake and Ninja, which make compiling, managing dependencies, and deploying across platforms more efficient.

Modern Development Practices

GNUstep now follows modern software development practices to make it more reliable and easier to maintain.

Better Version Control: Switching to Git has made development more flexible, with better branch management and collaboration.
Continuous Integration and Automated Testing: GNUstep uses automated testing to ensure updates do not introduce problems.
Stronger Security Features: Improvements in memory management, better encryption, and sandboxing make GNUstep more secure.

Overall, GNUstep continues to improve by refining its architecture, tools, and platform support. These updates keep it a strong choice for developing Objective-C applications that need modularity, cross-platform compatibility, and long-term stability.

6. Control and Data Flow

Control flow in GNUstep focuses on how execution moves through the system. This is driven by event-based interactions, function calls, and system responses. When a GNUstep-based application is launched, the execution begins with NSApplication, a class that provides the central framework of every application's execution (GNUstep, n.d.) This initializes the necessary environment and event loop.

When a user interacts with the application such as through clicking a button or typing in a text field, control is passed to the event handling system, specifically libs-base, where threads wait for events and processed input to determine the appropriate response. Once detected, events are queued and processed asynchronously. The control flow then moves to NSResponder in the AppKit framework, where it determines which object should handle the event. This follows the Responder Chain (Fedor, 2008), where events pass through a hierarchy of objects, allowing various elements such as NSWindow, NSView, and NSControl to process or pass the event as needed. Events are dispatched to UI elements such as NSButton or NSTextField, triggering their corresponding methods to execute necessary actions (e.g., changing text, opening dialogues, or triggering animations).

If any operation requires system-level functionality, such as file access, graphics rendering, or sound processing, then the control is passed to libs-back, which interacts with lower-level system libraries such as Cairo or Windows API. This backend layer abstracts OS-specific operations, ensuring platform-independent behaviour. This layer is responsible for drawing UI elements, managing system events, and handling device-specific operations such as printing or network communication. Once an operation is complete, control flows back upwards through the UI framework, allowing the GUI and elements to refresh, provide visual feedback, or update data representations dynamically. This ensures efficiently managing re-rendering cycles and avoids unnecessary computations.

In addition, GNUstep follows a structured approach to data flow, primarily using the Model-View-Controller (MVC) design pattern. This design separates the software component into three pieces: a model, a view, and a controller, to organize data flow effectively. The model layer stores and processes data using objects like NSArrayController and NSUserDefaults. The view layer displays information to users through NSView, NSTableView, and other UI elements. The controller layer connects the model and view, ensuring that updates are properly communicated and changes are reflected dynamically in the UI.

7. Concurrency

GNUstep uses a multi-threaded architecture to handle concurrency; essential for model applications that require responsiveness and resource efficiency. The libs-base library provides necessary tools for managing threads and synchronization, allowing applications to perform multiple tasks simultaneously without any issues.

Thread Management

The component libs-base allows applications to create multiple threads, each of which can execute independently. This can be useful for tasks that require background processing, such as network requests or file I/O. To optimize resource usage, libs-base also supports thread pooling, where a fixed number of threads are reused for multiple tasks, reducing the overhead of creating and destroying threads.

Synchronization

Locks and semaphores are synchronization mechanisms that libs-base provides to ensure that shared resources are accessed safely by multiple threads. This prevents race conditions and checks for data integrity. For event handling, libs-base supports event-driven programming, where threads can wait for specific events (e.g., user input or network responses) before proceeding. This allows for applications to remain responsive while waiting for external events.

Concurrency in GUI Applications

In GUI applications, the main thread is responsible for handling user interactions and updating the UI, and the libs-gui is responsible for all UI updates to be performed on the main thread to avoid conflicts. Long-running tasks, such as file downloads or complex computations, are typically performed on background threads to prevent the UI from freezing. Once the task is complete, the results are passed back to the main thread for display.

Implications for Developers

To account for thread safety, developers must ensure that shared resources are accessed safely, using the synchronization mechanisms provided by libs-base. Since long-running tasks are performed on background threads, developers can offload this to ensure that the UI remains responsive, providing a better user experience.

8. Implications for the Division of Responsibilities

GNUstep's design, based on object-oriented and layered design principles, naturally divides responsibilities among developers based on the modular structure of its core components. The implications for the division of labour are clear role specialization, successful parallel development, and better maintainability. GNUstep design promotes cooperation among developers by ensuring modular growth for concurrent activity and maintenance.

Layered and Modular Development

GNUstep's layered structure allows teams to work on individual components of the system separately. Each layer is a separate unit that interacts with other layers via specified interfaces, so modifications to one module would not impact others. Modularity facilitates specialization within the developers:

Foundation & Core Libraries Developers (libs-base, libs-corebase)**:** Develop core non-graphical functionality such as memory management, network programming, event handling, and data structures. They ensure compatibility with Apple's CoreFoundation while maintaining cross-platform portability.

User Interface Developers (libs-gui, Gorm): Develop graphical functionality, event-driven programming, and UI rendering. They ensure the UI is intuitive and responsive while maintaining compatibility with Apple's Cocoa framework.

Rendering & System Integration Developers (libs-back): Provide rendering and display support that bridges the gap between GNUstep's UI building blocks and the operating system graphical system.

Tooling & Interface Designers (Gorm): Design drag-and-drop tools to graphically construct an interface, making it easy for end-users to develop applications.

Collaboration and Dependency Management

While teams work independently, coordination is necessary to deliver seamless integration between components:

APIs and Interface Design: Developers must adhere to strict API contracts to maintain interoperability among layers.

Version Control & Testing: Continuous integration practices ensure that modifications in a single module do not cause conflicts in dependent modules.

Documentation & Code Standards: It allows developers to contribute substantially to different subsystems by having organized documentation and coding standards.

Scalability and Maintainability

The object-oriented design in GNUstep encourages code modularity and reuse, which simplifies maintenance. The design is also scalable as new functionality or optimizations can be introduced without interrupting the system.

9. Use Cases

Sequence Diagram to Run the Application

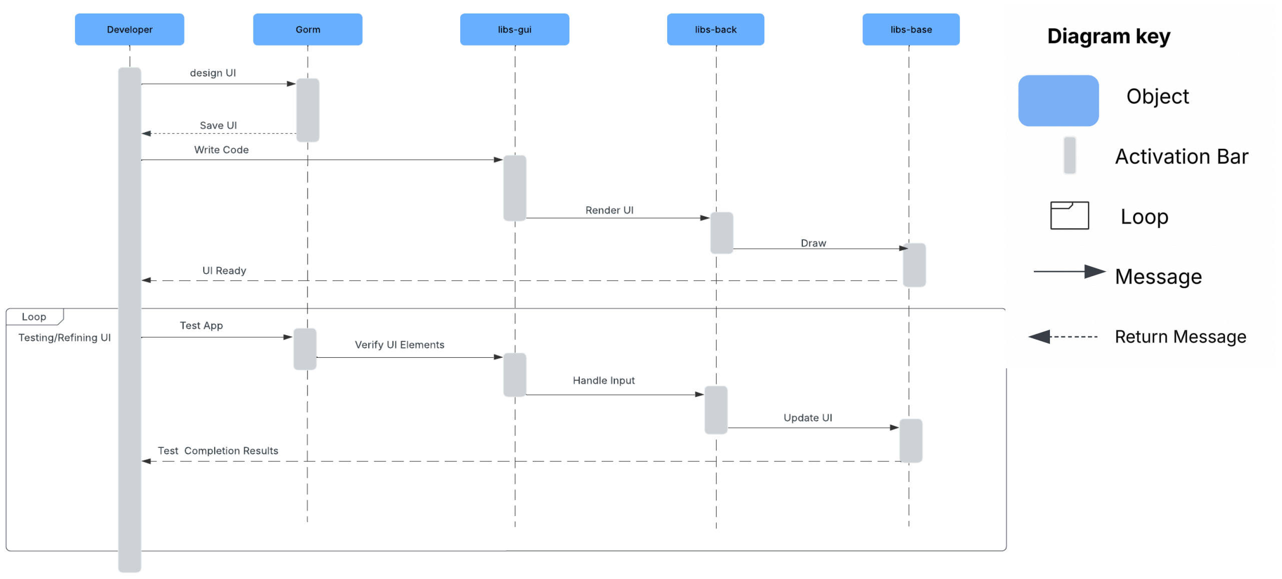

Figure 2. Sequence Diagram to Run GNUstep

When the End User opens a GNUstep program, the system initializes using libs-base for fundamental services like memory management and libs-gui to configure the graphical user interface. libs-gui creates and configures UI components, whereas libs-back renders these components on the screen by transforming drawing commands into platform-specific actions. When the End User interacts with the UI (by clicking a button or entering text), libs-gui evaluates the input and modifies the interface accordingly. libs-back then updates the display to reflect the changes, offering a smooth and responsive user experience. Finally, the program replies to the user's actions to close the interaction loop. This layered design, with explicit separation of duties between libs-base, libs-gui, and libs-back, guarantees modularity, cross-platform compatibility, and scalability for applications developed within the GNUstep ecosystem.

Designing a User Interface with Gorm

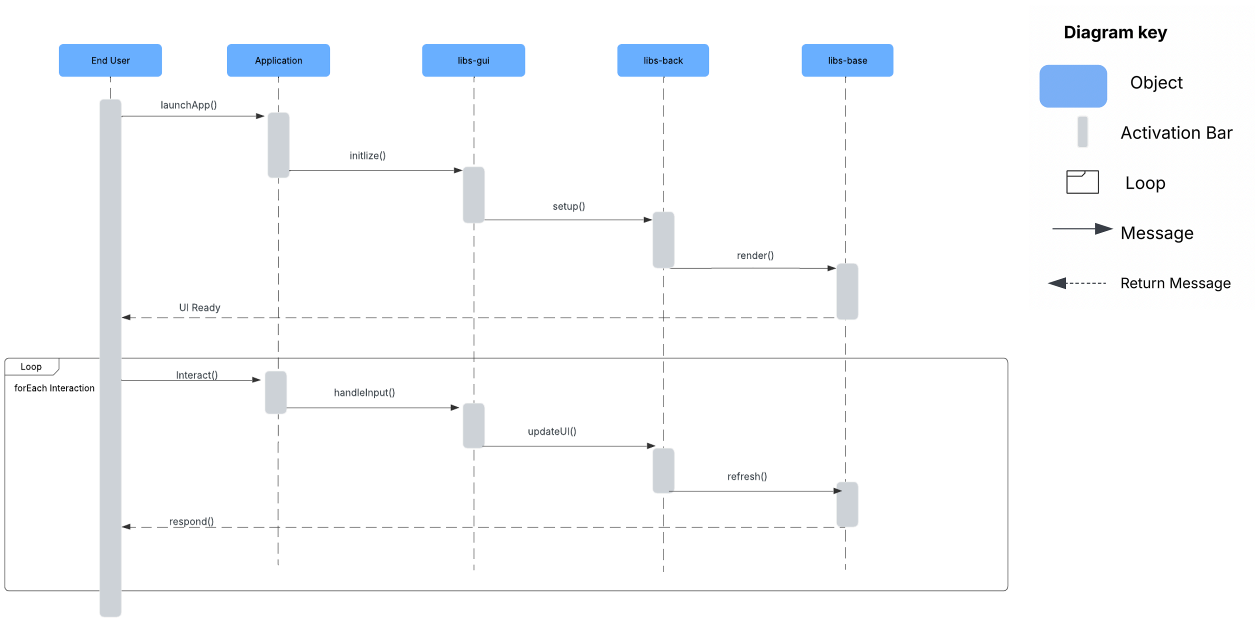

Figure 3. Sequence Diagram for Designing a User Interface with Gorm

Gorm is used only in the design phase. The Developer first designs the UI using Gorm, which saves the layout and connects it to application logic. During the rendering phase, libs-gui handles UI components, libs-back manages rendering, and libs-base provides low-level system support. Once the UI is ready, the testing phase begins, where the Developer interacts with the UI, verifies elements, processes input, and updates the UI as needed. This iterative loop continues until the UI behaves as intended.

10. External Interfaces

GNUstep interacts with several external components to facilitate its functionality as a cross-platform application development framework. It provides an object-oriented approach to GUI development, modeled after the OpenStep API. These interactions are mainly managed through its core libraries, which handle communication with the operating system, graphical display systems, and other external resources.

Graphical User Interface (GUI)

GNUstep’s libs-gui library manages the GUI components of applications. Its applications interact with different windowing systems, depending on the operating system, through the libs-back library (GNUstep’s backend system):

- Cairo backend (default): Uses the Cairo 2D graphics library for rendering.
- Winlib backend: Uses Windows API for rendering on Windows systems.
- X11 backend (deprecated): Older backend for X11-based systems.
- Library backend (deprecated): Used a PostScript-like 2D vector rendering system.

The information exchanged includes:

- Input Events: User interactions such as mouse clicks, keyboard input, and window resizing are captured by the display system and passed to libs-gui for processing.
- Rendering Commands: libs-gui sends abstract drawing commands, from the Application Kit (AppKit), which provides GUI components like buttons, windows, and menus, to libs-back, which translates these commands into platform-specific rendering operations (Wikipedia).
- Display Updates: After rendering, libs-back updates the display with the rendered graphical elements, and ensures that the application's UI is displayed correctly on the screen.

Additionally, GNUstep provides tools for GUI design. One is Gorm, which is a visual interface builder, similar to Interface Builder in macOS. Another is Renaissance, which is an XML-based system to define UI layouts that offer portability and theme support. Lastly, user inputs such as mouse clicks, and key presses are captured through event handling mechanisms provided by AppKit.

Operating System (OS)

GNUstep's libs-base and libs-corebase libraries interact with the operating system to provide essential system services. The information exchanged includes:

- File I/O: libs-base handles file operations such as reading and writing files, which involve communication with the OS's file system.
- Memory Management: libs-corebase manages memory allocation and deallocation, and interacts with the OS's memory management system so that the resources are used efficiently.
- Networking: libs-base provides networking capabilities that allow applications to send and receive data over the network. This involves communication with the OS's networking stack (TCP/IP, DNS, HTTP).

GNUstep also supports interactions with web APIs through NSURLConnection (Handles HTTP and HTTPS requests), Sockets (Allows low-level network communication), and Distributed Objects (DO) (Supports inter-process communication over networks).

File System Interaction

GNUstep supports file operations through its Foundation Kit, which provides high-level file-handling APIs. User preferences and configurations are stored in a default database, typically located at ~/GNUstep/Defaults/, relative to the user's home directory. Applications interact with the host operating system's file system through the Foundation's NSFileManager class. Configuration data is stored in property list (plist) format, supporting both text-based and binary serialization (GNUstep.org).

Database Systems

While GNUstep itself does not include a built-in database system, it can interact with external databases through third-party Object-C libraries and its networking and file I/O capabilities. The information exchanged includes:

- Database Queries: Applications can send queries to external databases (e.g., SQL databases) using networking libraries provided by libs-base.

- Data Retrieval: The database system returns query results to the application, which can then process and display the data using libs-gui.

Interactions with external components include: use of key-value storage for application settings and preferences, Objective-C binding to interact with SQL databases, and XML-based formats for structured data storage (Wikipedia).

Plugins and External Libraries

GNUstep supports the use of plugins and external libraries to extend its functionality. The information exchanged includes:

- Plugin Initialization: When a plugin is loaded, GNUstep communicates with the plugin to initialize it and integrate its functionality into the application.
- Data Exchange: Plugins may exchange data with GNUstep components, like passing configuration settings or receiving input from the user interface.
- Event Handling: Plugins can register to handle specific events (e.g., mouse clicks or keyboard input) and receive notifications from GNUstep when these events occur.

11. Data Dictionary

Open-source: A code base that is publicly accessible to be used or contributed to.

12. Naming Conventions

GUI: Graphical User Interface
UI: User Interface
Gorm: Graphical Object Relationship Modeller
API: Application Programming Interface
I/O: Input/Output
XML: Extensible Markup Language
IPC: Inter-Process Communication (IPC)
ARC: Automatic Reference Counting

13. Conclusion

GNUstep's architecture shows a well-structured framework built around object-oriented and layered styles. The many sets of components that make up the system, namely libs-base, libs-corebase, libs-gui, libs-back, and Gorm, work together to create efficient application development, from low-level system functionality to high-level GUIs. In addition, analyzing the architecture shows its modularity and cross-platform compatibility, allowing GNUstep applications to run on various operating systems while maintaining a native look and feel. The system uses a model-view-controller design for control and data flow, ensuring clear separation between UI elements, logic, and data management. Concurrency is managed through multi-threading and synchronization mechanisms in libs-base, which improves responsiveness and efficiency. We are interested to see how future optimizations and enhancements of GNUstep will further strengthen its role in the software development community.

14. Lessons Learned

One of the biggest things we took away from this project was just how important good communication is when working as a team. For a few of us, this was our first time working with a larger group on a school assignment, and it definitely took some adjusting. We had to learn how each person works, things like their pace, how they like to get tasks done, and when they're most available. Figuring all that out helped us work better together and made sure everyone felt like they were contributing. We also realized pretty quickly that we underestimated how much time and research this project would actually take. If we could go back and do it again, we'd definitely begin researching and pulling our ideas together sooner, instead of feeling rushed later on. That said, we're proud of what we accomplished. We learned a lot, not just about the topic we were researching, but also about working as a team, managing our time better, and making sure everyone's voices were heard.

15. References

Fedor, A. (2008, January 1). 8. Event handling. https://www.ict.griffith.edu.au/teaching/7421ICT/archive/resources/documentation/Developer/Gui/ProgrammingManual/AppKit_8.html

GNUstep Base Library. Free Software Directory. (n.d.). https://directory.fsf.org/wiki/Gnustep-base

GNUstep. GitHub. (n.d.). https://github.com/gnustep

GNUstep. GNUstep.org. (n.d.). https://www.gnustep.org/

Gui. GNUstep. (n.d.). https://mediawiki.gnustep.org/index.php/Gui

Marcotte, L. (n.d.). *Programming under GNUstep--an introduction*. ACM Digital Library. https://dl.acm.org/

Mockus, A., Fielding, R. T., & Herbsleb, J. (n.d.). Two case studies of Open Source Software Development. https://mockus.us/papers/mozilla.pdf

MVC. MDN Web Docs. (n.d.). https://developer.mozilla.org/en-US/docs/Glossary/MVC

Wikimedia Foundation. (n.d.). *GNUstep*. Wikipedia. https://en.wikipedia.org/wiki/GNUstep#Software_architecture