Conceptual Architecture Report for GNUStep

Group 3: Gamers Never Give Up

Zoya Zarei-Joorshari — 19zzj@queensu.ca

Meagan Mann — 20mmm27@queensu.ca

Jawad Ahmed — 19jaa14@queensu.ca

Kim Hyun Bin — 24vln2@queensu.ca

Ripley Visentin — 22rknv@queensu.ca

Ethan Nguyen — 20hen@queensu.ca

February 18, 2025

Contents

Abstract

This report presents a conceptual software architecture of GNUStep, a free and open-source software implementation of the NeXT libraries and APIs such as the OpenStep specification. Like its predecessors and influences, it is an object-oriented application programming interface for GUI development, providing frameworks and tools for robust cross-platform software development. It is supported for Unix-like Operating Systems and Microsoft Windows.

GNUStep fits squarely under the object-oriented architectural style. Its reliance on component interactions such as drag-and-drop elements and OS interactions allows for a modular and encapsulated system where objects can be reused and extended. This is supported by the fact that GNUStep uses frameworks like the Foundation Kit and Application Kit which provides wrapper classes and classes oriented around GUI capabilities respectively. Additionally, it is designed to be compatible with other object-oriented frameworks like Cocoa. The major components of GNUStep are: Libs-back, Libs-base, Libs-corebase, Libs-GUI and Gorm. Each of these components serve a critical role to ensure robust development, cross-platform compatibility, functionality, and ease-of-use.

This report aims to provide insights into the conceptual architecture of GNUStep and its impact on software design, portability, and usability. Its uses for cross-platform development and object-oriented style illustrate its modular design and cements its vital role in the history of Objective-C framework development.

1 Introduction and Overview

Application development is a complex process that requires a high degree of planning and expertise. There have been many frameworks created to simplify the development process: a notable one being Apple's Cocoa for macOS based on NeXT's OpenStep framework [1]. What had started off as an implementation of NeXTSTEP evolved when it became OpenStep, an ancestor of Cocoa. The Cocoa API has expanded to become the basis for modern macOS and iOS systems. Though, Cocoa is not portable as it is specifically designed for macOS, leaving room in the market for application development frameworks for other platforms. GNUStep tries to solve this problem by being an application development platform that is available on a variety of different platforms [13].

GNUStep is an open source framework created for development of GUI applications. Written in Objective-C, it is a free, object-oriented, cross-platform, development environment. It's core framework is an open source version of Apple's Cocoa APIs and it strives to be as portable as possible [13].

We derived the conceptual architecture for GNUStep by reasearching the core components, reading documentation provided by developers, and researching related systems (ex. OpenStep, Apple CoreFoundation frarmework). We provide detailed analysis of each components and several other aspects of GNUstep such as control and data flow and concurrency.

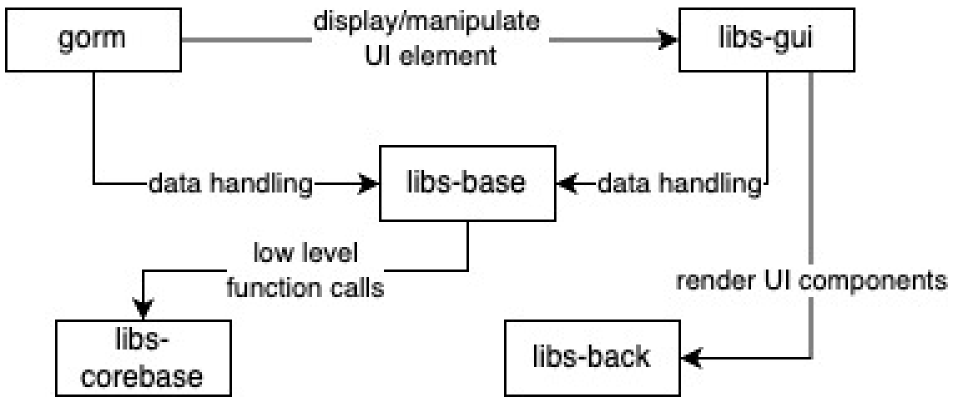
We also analyzed on the relationships and interactions between different libraries to deduce an architectural we believe was used to build GNUstep. Further, we use sequence diagrams to explore the two use cases: ”Modernizing Legacy OpenStep applications” and ”Cross-Platform Development with Objective-C”.

In addition, we also researched on the evolution of the systems through various versions and laid out any updates or changes implemented. Towards the end, we end off by laying out the division of responsibilities, data dictionary, naming conventions and the lessons learned while researching GNUstep.

2 Architecture

2.1 Architectural Style: Box and Arrow Diagram

We have identified the Object Oriented Style as GNUstep's main architectural style as shown below. This could be inferred from various sources and by taking a look at the codebase and its overview [13]. GNUstep consists of multiple different components, each acting as a library and in charge of specific functions and information body. Each component has multiple different operations and are encapsulated. However, each of the components interact with each other, invoking different function calls to call onto each other. Overall, we observed that it is a system with a collection of autonomous interacting agents.



We have also thought of a layered architectural style for GNUstep as it was mentioned sparingly throughout the documentation [10]. However, we believe that instead of hierarchical layers, with each layer interacting only with layers on top and below, these components are interacting in various ways with multiple other different components without any layered restrictions. Therefore, we decided to pursue an Object Oriented Style instead.

2.2 System functionality

GNUstep is modular and is made up of several interacting components that collectively offer a comprehensive development toolkit. The five core components of GNUstep include libs-base, libs-corebase, libs-GUI, libs-back, and Gorm, which is a visual GUI builder. These components work together to allow developers to build cross-platform applications efficiently. Libs-base is the foundational framework that offers core functionalities which libs-GUI builds to create graphical interfaces. The backend, libs-back, ensures that these interfaces are rendered correctly on various platforms. Gorm enhances the system as it has tools for visual interface design, while libs-corebase offers additional low-level utilities to support the application development.

2.2.1 libs-back

The libs-back component is commonly referred to as the GNUstep GUI backend, and is an integral part of the architecture that connects the graphical front end (libs-GUI) with the display mechanisms of the underlying operating system. Responsible for rendering and windowing tasks, Libs-back ensures GNUstep can operate across different platforms without modifications to the application source code. GNUstep GUI backend acts as a bridge between the graphical interface and the rendering systems of different operating environments for a reliable graphical experience[11].

For platform-specific rendering, libs-back translates the high-level graphical commands from libs-GUI into the unique rendering instructions of each platform [13]. It supports various operating environments, so GNUstep can run on X11, Windows, Unix-like systems, etc. Libs-back uses a rendering engine that emulates the Display PostScript (DPS) functions required by the front-end GUI components [11]. DPS is a feature that originated in the NeXTSTEP and was later used in the MacOS's Quartz graphics systems [12]. This emulation ensures that graphical elements are consistently rendered across all platforms. Another key functionality of libs-back is that it supports multiple backends through the package gnustepback. This package provides Cairo- default using the 2D graphics library, winlib- default for Microsoft Windows, art- an old backend for Unix-like systems, xlib- another old backend for X11 [13].

The AppKit, libs-GUI, provides the graphical elements, while libs-back translates them into the platform-specific drawing operations. This ensures that the GUI elements are rendered appropriately on each platform. While libs-back primarily handles rendering, it still relies on the Foundation framework for core services such as memory management, event processing, and file operations. When interfaces are being designed in Gorm, the resulting UI layouts must be interpreted and rendered by libs-back to ensure it appears as intended across platforms.

2.2.2 libs-base

The libs-base library is perhaps the most fundamental component of the GNUstep software, providing classes for essential system-level services necessary for application development in the GNUstep environment. Libs-base implements the object-oriented runtime required by Objective-C, handling object creation, message dispatching, and method resolution. It includes infrastructure such as memory management, data structures, and utility functions such as date and time manipulation, file handling, and network communication. It also forms the base of the Foundation Framework, the non-graphical portion of Apple's Cocoa frameworks. GNUstep's base library forms the backbone of the software, providing data storage and manipulation functionality that the higher-level libraries such as libs-appkit and libs-gui. It is essential for compatibility and porting from Cocoa.

Libs-corebase is an extension of libs-base, providing more specialized low level functionality, usually dealing with deep system interactions like concurrency and networking. In comparison, libs-base has more basic tools used in application development. Its implementation of these basic tools is what libs-corebase relies on to provide its extended and higher-level functionality.

2.2.3 libs-corebase

The libs-corebase component in the GNUstep project is a foundational library that provides a variety of general-purpose, non-graphical C objects and utilities [7]. It was designed to serve as an open-source alternative to Apple's CoreFoundation framework [6], and it plays a crucial role in software development by offering low-level data structures, memory management, event handling, and interprocess communication. As an open-source alternative, the library is an attempt to implement the non-graphical portion of Apple's CoreFoundation framework. By doing so, it bridges the gap between GNUstep and Apple's development ecosystem, allowing for cross platform compatibility.

The primary purpose of libs-corebase is to provide core services that support application development, enabling seamless integration with other components of the GNUstep environment. There are some key features to highlight. libs-corebase implements its own set of fundamental data types, including strings, numbers, and binary data, which can be efficiently passed to C routines. It also provides various collection types such as arrays and dictionaries, allowing developers to efficiently organize and manage data. Additionally, as memory management is a crucial aspect of any software system, libs-corebase ensures efficient allocation and deallocation of memory to optimize performance and prevent leaks. It also offers mechanisms for event-driven programming through run loops, enabling tasks to be scheduled and executed asynchronously. On top of these functions, it facilitates communication between different processes, ensuring smooth data exchange and coordination. With any systems, file management is a big issue too and this library provides robust support for file management and working with URLs, essential for modern application development. It also includes functionality for handling dynamically loaded code bundles, which enhances modularity and flexibility in software design.

2.2.4 libs-gui

In GNUStep, the libs-gui is the component responsible for the Graphics User Interface and the implementation of the AppKit API. This is the framework that would provide rendering for windows, buttons, menus, dialogs, and all other UI elements, so developers could create user-friendly applications.[8]

The libs-gui framework interacts closely with other GNUStep components, in particular libs-back, which provides much of the necessary backend functionality such as data structures, networking, and multithreading [8]. libs-gui communicates with windowing systems, such as X11[13] for Linux systems, to draw graphics and handle user interactions. It follows the Model-View-Controller paradigm where it is primarily a View layer that processes user events and dispatches them to controllers to be handled properly. Events, such as button clicks and keystrokes, are dispatched through an event loop, which uses the responder chain to determine which component should handle each interaction.

Over time, libs-gui has evolved from OpenStep into a more Cocoa-compatible API, which enables much easier porting of macOS applications[13]. Updates are regular and increase the capabilities of this framework while sticking to the open nature of OpenStep. These will be further documented in the evolution section. NSRunLoop is used in libs-gui for concurrency, which effectively runs UI event handling, simultaneously providing multithreading support [8]. However, the developer has to manage threads as there will be a risk of updates not pushing to the main thread, preventing inconsistencies and crashes.

The responsibility is well structured within the GNUStep project. Core maintainers will be working on keeping better compatibility with OpenStep and Cocoa APIs while improving stability and performance of frameworks. Contributors will be working on bug fixes and feature enhancements for specific GUI components, while application developers depend on libs-gui to build software with a functional and interactive UI[8].

2.2.5 Gorm

Gorm (Graphical Object Relationship Modeller) is used to design graphical interfaces for applications. This module is meant to make designing GUI quick and easy through drag and drop elements [3]. It is comprised of GormCore, InterfaceBuilder, GormObjCHeaderParder libraries/frameworks, the Gorm application, and gormtool [9].

Within the application itself, when a document is loaded, editors (NSMenu objects) are attached to each drag and drop element (NSWindow objects) in the user interface: these editors are responsible for the handling selection of their respective edited object [5]. Editors need to be alerted when their corresponding object is selected: this lets the editors know when they become the active editor. The main application looks out for notifications (IBSelectionChangedNotification) sent by editors when changes in selection occur to keep track of who has the selection [5]. To update the display, the editors managing the drag-and-drop operation calls [NSApp -displayConnnectionBetween:and:] to tell Gorm to update its display: this returns the window that the object is being displayed in and the rectangle enclosing the object [5]. Further, Gorm utilizes libs-gui to display UI elements, and libs- gui uses libs-back handle rendering of interfaces designed within the program. This allows for the portability of Gorm across several platforms.

2.3 Control and data flow

The main components of GNUstep work together to provide a cross-platform development environment for Objective-C applications. To better understand how GNUstep operates across different platforms, we will discuss control flow and data flow.

Control flow in GNUstep follows a hierarchical structure where separate components trigger actions in response to application requests. When a user runs GNUstep, the application first initializes the Foundation framework (libs-base). If the application has graphical elements, then it initializes the Appkit framework (libs-GUI) which manages UI components. Once GNUstep is up and running, libs-GUI receives requests to create these UI components, libs-GUI sends rendering instructions to libs-back, libs-back translates these commands into rendering instructions, then the operating systems display system executes the drawing operations and the graphical elements are rendered on the screen [13]. Input from the user are captured by the operating system, processed by libs-back, sent to libs-GUI which ultimately triggers the required logic.

Data flow describes how the different parts exchange information. When an application requires UI components, libs-GUI stores the properties such as size, position, etc., which libs-GUI send to libs-back for rendering. Libs-back converts these properties into a format suitable for the platform's graphics engine and sends it to the operating system. The operating system manages the user input, which travel through libs-back and libs-GUI before the appropriate application logic is triggered. Gorm allows UI design by storing the layouts in .gorm files which are loaded when run to build the interface.

2.4 Concurrency

GNUStep supports concurrency through classes like NSThread and NSOperationQueue found in the Framework kit. These classes which inherit from the NSObject base class, handle thread management and high-level abstractions for managing concurrency respectively.

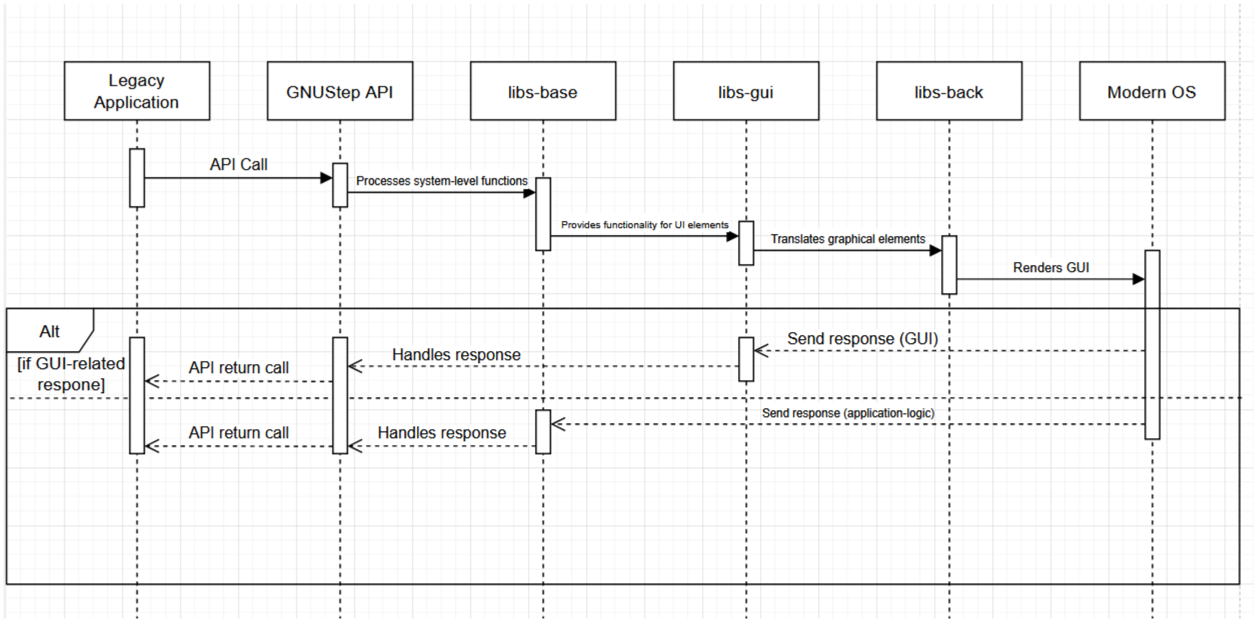
The NSThread class encapsulates OpenStep threading. Each process begins with a main thread and additional threads can be created using NSThread. The GNUstep implementation of OpenStep has been carefully designed so that the internals of the base library do not use threading (except for methods which explicitly deal with threads of course) so that you can write applications without threading. The NSOperation queue class manages a number of NSOperation objects, scheduling them for execution and managing their dependencies. Depending on the configuration of the queue, NSOperation objects may be executed concurrently or serially.

3 External Interfaces

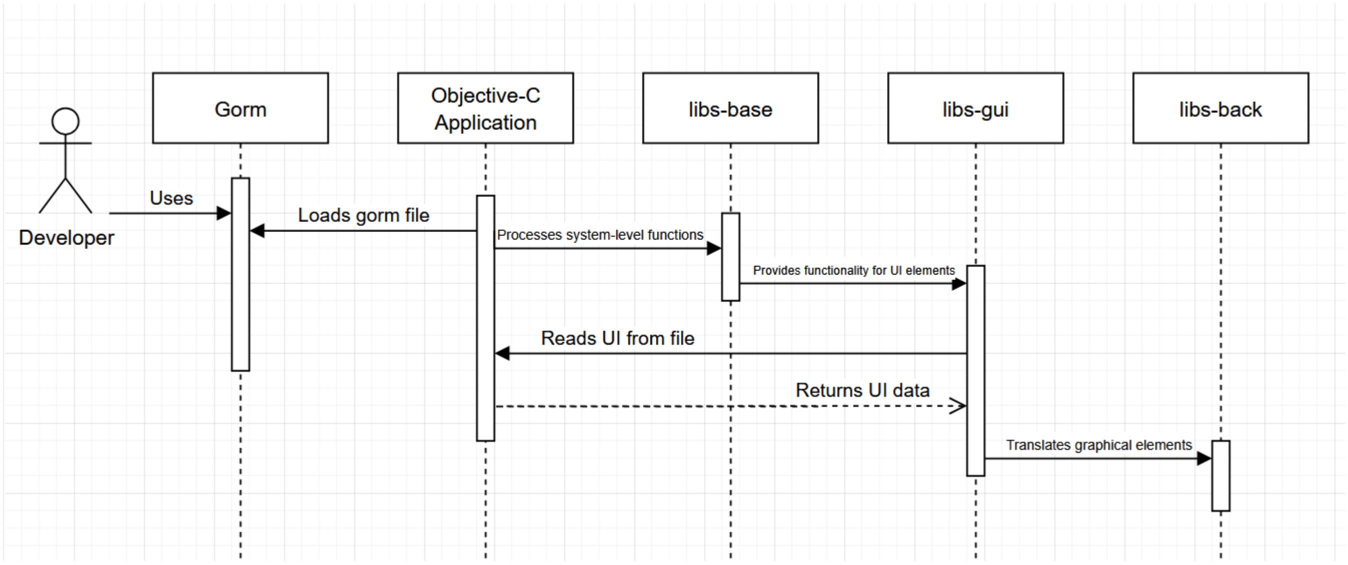
GNUStep communicates the clients with the system and services through a number of interfaces that facilitate the transfer and receipt of data. The primary external interface, the GUI, allows users to communicate with and receive data from applications developed on the GNUStep platform. The system allows data exchange through files, preserving user preferences, configurations, and application states. Furthermore, it is capable of communicating with services on remote networks through message-oriented protocols, thereby facilitating application-to-application communication, distributed object management, and data exchange with web services. GNUStep also communicates with databases for long-term storage of data, although the framework does not expose the inner structure of stored data. Collectively, these interfaces provide a strong environment for autonomous and distributed applications.

4 Sequence Diagrams

4.1 Use case 1: Modernizing Legacy OpenStep Applications



4.2 Use case 2: Cross-Platform Development With Objective-C



5 System evolution

5.1 libs-base [2]

The first and only released version of libs-base is 1.30.0. In this version weak references became available in portable code. Weak references became supported in NSMapTable, NSHashTable, and NSPointerArray. Further, support for being built with `-asan yes` command line option or the `GNUSTEP_WITH_ASAN=1` environment variable was added. This version also allowed libs-gui to work with recent versions of autoconf.

5.2 libs-gui [4]

The first released version of libs-gui is 0.9.3. In this version the spell checker daemon was re-implemented using libaspell, and NSComboBox was re-implemented. Code for NSToolbar was also redone to make the class for functional.

In version 0.10.0 the interface version was updated. Keyed encoding support was added to many classes and Nib loading was improved.

In version 0.11.0 keyed encoding was added to all gui classes and Nib loading was implemented. There was also better support for color schemes and themes added.

In version 0.12.0 the Tooltip implementation was added. This version also added new methods, ivars, classes, and changed some existing components.

In version 0.13.0 added the font attribute to allow for output of glyths not currently present. There was also several new methods added.

In version 0.14.0 a new class for glyth generation was added. Also, an old NeXT method was added for compatibility.

Version 0.16.0 was a stable release. This version added several methods for Nib loading, as well as methods to support document types with multiple allowed file extensions. Document auto-saving was also implemented.

In version 0.17.0 there was many bug fixes and improvements to the toolbars.

Version 0.18.0 was another stable release that has many Windows specific improvements and added better compatibility wit Mac OS X.

Version 0.20.0 was also a stable release with improvements for Nib loading, documents, and document controls. There was also improvements made to theming.

Version 0.22.0 was a major stable release. This version added support for drawing GUI with a scale factor, a character panel, a color picker tool (for selecting colors from parts of the screen), and support for many new image formats.

Version 0.23.0 was also a major stable release. This version added many bug fixes and tweaks to the UI to improve look and feel.

Version 0.24.0 was another major stable release. This version includes theme improvements and many bug fixes and tweaks.

In version 0.25.0 the GIF library was updated to support new versions and there was improvements to theming.

In version 0.26.0 includes many compatibility improvements.

In version 0.27.0 there was many bug fixes and compatibility improvements.

In version 0.28.0 support was added for modern XIB files.

The newest released version of libs-gui is 0.29.0. In this version support for modern XIb files was added as well as many new classes and bug fixes. There was also some improvements made to compatability.

5.3 Gorm [5]

The first released version of Gorm is 0.0.1. In this version, the basic framework was created. The ability to load and manage palettes was added along with four palettes. The Window palette was expanded to allow the creation of windows and panels. Drag-and-drop functionality for elements from these palettes into a window was introduced, along with cut/copy/paste support. A pre-document editor object was added to track icons representing drag-and-drop elements, as well as a special inspector for handling empty or multiple selections.

In version 0.1.0, more palettes and inspectors were added for number formatters, browsers, and tableViews. Further custom views were introduced, along with the ability to change the font of some objects.

In version 0.2.0, custom class support was added along with more views for editing. Improvements were made to test mode support, drag-and-drop functionality, and sound/image support.

In version 0.3.0, preferences were introduced. A class for inspectors was added, allowing users to choose between an outline view or the inspector for editing new classes. Additional inspectors were also included.

In version 0.4.0, new Menu and Menu Item inspectors were added. Users could now specify the Services and Windows menus in the menu inspector. The ability to create custom subclasses was also introduced.

In version 0.5.0, support for upgrading old .gorm files was added.

In version 0.7.5, the reparent feature was added, allowing users to change the class hierarchy within Gorm. Support for NSFontManager was also implemented.

In version 0.8.0, the file version for Gorm was updated. Full custom palette support was added, along with two experimental features:

- Standalone view – Allows users to treat a view as a top-level object.
- File section – Helps users manage file compatibility with GNUStep.

In version 0.9.0, support for drag-and-drop of images and sounds was added. Users could now add methods to non-custom classes. The date and number formatter inspectors were fully implemented, and cut/paste support was improved.

The newest released version of Gorm is 0.9.2, which is primarily a bug-fix release.

6 Division of Responsibilities

The development of GNUstep involves many contributors who take up various aspects of the system. The overall maintainers oversee the entire project, including adhering to OpenStep and Cocoa conventions and being responsible for significant updates and structural changes. Those responsible for GNUstep Base (libs-base) develop the core system functionality like memory management, file I/O, networking, and multithreading, thus ensuring the framework has a strong base to construct applications. The libs-gui group is responsible for the development and upkeep of graphical user interface-related items, such as widgets, event handling, and window management, in accordance with the Model-View-Controller (MVC) pattern. Meanwhile, the developers engaged in GNUstep Back (libs-back) develop backends for rendering that allow GNUstep applications to be executed on various platforms and consequently interface with system graphics libraries like X11, Windows GDI, or Cairo. Moreover, application developers do their part by developing applications on the framework, submitting bugs, as well as adding functionalities, whereas documentation and community contributors assist in the writing of guides, tutorials, and updating the knowledge base of the project. This split guarantees an orderly and concerted development process that allows GNUstep to develop and at the same time be stable and portable on a variety of platforms.

7 Data Dictionary

Bug Fix - fixing errors within code. **Conceptual Architecture** - abstract structure of a system.
GNUStep - Open source app development environment.
Gorm - Used to design graphical interfaces for applications using drag and drop elements.
Method - used to manipulate data in objects.
Object - an entity that contains data and has methods to manipulate said data.

Object-Oriented Style - an architectural style in which the central issue is indetifying and protecting related bodies of data.
Portable - when system/software can work on multiple different platforms.

8 Naming Conventions

GNU - GNU's Not Linux
Gorm - Graphical Object Relationship Modeller
GUI/UI - Graphical User Interface

9 Lessons Learned

One of the main lessons learned from researching GNUStep and working on the report is the need to comprehend its object-oriented architecture and its implications on the system architecture. We discussed viewing GNUstep using a layered model; however, we found that emphasizing its object-oriented concepts gave us a better insight into the interaction among its components. GNUstep is heavily based on OpenStep and NeXTSTEP concepts that employ class hierarchies, encapsulation, and delegation, which makes Objective-C's dynamic runtime an essential part that helps in its flexibility. The impact of inheritance and polymorphism on the system's extensibility was neglected, especially in libs-gui, where UI elements adhere to reusable class frameworks. If we were to do this project over, we would place even more focus on the role that design patterns, such as Model-View-Controller (MVC) and delegation, play in making the code more maintainable. What made understanding the structure and class interactions difficult was the lack of updates on documentation, leading us to directly examine the source code. In the future, a more structured approach, such as tracing class dependencies before exploring broader system interactions, would be more efficient in studying object-oriented frameworks.

10 Conclusion

In this report we derived the conceptual architecture for GNUStep. Further, we explored subsystems of GNUStep: libs-back, libs-base, libs-corebase, libs-gui, and Gorm. GNUStep's functional architecture is designed to provide a robust foundation for developing Objective-C applications under NexT-like frameworks. Libs-base serves as the core library, handling essential data structures and system-level functions while libs-corebase provides low-level functionality and interoperability. Gorm interacts with libs-gui to display and manipulate UI elements while also interacting with libs-base for data handling. Libs-gui interacts with libs-back to handle rendering UI components, and libs-base interacts with libs-corebase for low-level function calls. These interactions collectively contribute to a cohesive and extensible system

Our analysis of these subsystems and their interactions led us to determine that GNUStep follows an Object-Oriented architectural style. This style is well-suited for GNUStep, as it enables modularity, encapsulation, and polymorphism—allowing components to seamlessly communicate while maintaining abstraction from underlying implementations. This is particularly beneficial for GNUStep's structure, as it ensures flexibility and scalability while supporting future enhancements. This style also allows for the components to easily interact with each other without needing to know any underlying implementation, which works in the favor of GNUStep and its subsystems. This analysis provides a comprehensive review of GNUStep and its subsystems for future developers.

Looking ahead, the future of GNUStep will likely focus on enhancing cross-platform compatibility, improving frameworks to align with modern design trends, and increasing Objective-C support for better interoperability with contemporary software development tools like Cocoa/MacOS. This analysis provides a comprehensive foundation for developers looking to understand GNUStep at the conceptual level.

References

[1] Cocoa (api).

[2] GNUstep base release notes.

[3] GNUstep Developer Tools: Gorm.

[4] GNUstep GUI release notes.

[5] Guide to the gorm application.

[6] Apple development team. AppleCoreFoundation, 2025. Accessed: 2025-02-14.

[7] Github contributors. GNUcorebase, 2025. Accessed: 2025-02-14.

[8] Github contributors. GNUStepGUI, 2025. Accessed: 2025-02-14.

[9] Gnustep. Gnustep/apps-gorm: Gorm is a clone of the cocoa (openstep/nextstep) 'interface builder' application for GNUstep.

[10] GNUstep contributors. GNUstep Information, 2025. Accessed: 2025-02-14.

[11] GNUstep Project. Back - GNUstep, 2025. Accessed: 2025-02-10.

[12] Wikipedia contributors. Display PostScript, 2025. Accessed: 2025-02-10.

[13] Wikipedia contributors. GNUstep, 2025. Accessed: 2025-02-10.

https://www.gnustep.org/
https://www.gnustep.org/information/aboutGNUstep.html https://www.gnustep.org/resources/docum
http://gnustep.made-it.com/Guides/History.html
libs-gui:
https://github.com/gnustep/libs-gui
https://www.gnustep.org/resources/documentation/Developer/Gui/Reference/index.html
https://www.gnustep.org/resources/documentation/User/GNUstep/faq_1.html#SEC33

https://github.com/gnustep/libs-base
https://www.gnustep.org/resources/documentation/Developer/Base/Reference/index.html