CISC 322/326 - Winter 2025

GNUstep Conceptual Architecture Report

February 14th, 2025

Infinite Loops - Authors

Ray Huezo	21tch5@queensu.ca
Anneth Sivakumar	21as221@queensu.ca
Vedsai Pandla	21vp23@queensu.ca
Rachel Narda	22rn3@queensu.ca
Jonathan Thompson	20jmt1@queensu.ca
Princess Viernes	20pejv@queensu.ca

Table of Contents

1.0. Abstract

This study examines the conceptual architecture of GNUstep, an open-source framework for cross-platform GUI and server application development. GNUstep implements the OpenStep specification, ensuring compatibility with Apple's Cocoa API and allowing the production of portable applications across several operating systems. Through an analysis of documentation and source code, this study reconstructs the high-level architecture of GNUstep, focusing on its key components, subsystem organization, and architectural patterns.

GNUstep follows a layered object-oriented model, where core libraries provide essential system functionality. Other components also support an implicit invocation style for event-handling. The libs-base library offers fundamental Objective-C classes, while libs-corebase ensures integration with Apple's Core Foundation. The libs-gui library supports graphical user interface development, and libs-back enables cross-platform rendering. Additionally, GNUstep's notification and distributed object systems facilitate inter-process communication, while concurrency mechanisms such as NSThread, NSOperationQueue, and Grand Central Dispatch (GCD) enhance performance.

The report also explores GNUstep's subsystem organization, structured into four domains: System, Local, Network, and User, ensuring modularity and compatibility with OpenStep, NeXTstep (NS), and Unix-based file systems. By analyzing these aspects, this study highlights the role of layered architectures, modular design, and event-driven programming in GNUstep's framework.

The findings provide insight into GNUstep's scalability, adaptability, and cross-platform capabilities, demonstrating how its structured architecture supports long-term software evolution. Understanding GNUstep's conceptual design enhances knowledge of software architecture principles, distributed system interactions, and modular framework development.

2.0. Introduction and Overview

GNUstep is an open-source framework designed for the development of advanced GUI desktop applications and server applications [1]. It follows the OpenStep specification, originally developed by NeXT and eventually served as the foundation for Apple's Cocoa API [2]. By providing a cross-platform environment, GNUstep enables developers to write portable code that can run seamlessly across various operating systems, including Linux, BSD, and Windows [1]. Additionally, it offers a set of development tools that support both command-line and graphical user interface (GUI) programming, allowing for more efficient software production and management.

The objective of this report is to reconstruct the conceptual architecture of GNUstep by analyzing its documentation, source code, and design principles. Conceptual architecture refers to the structural organization of a system, focusing on its primary components and their interactions rather than specific implementation details. Understanding the conceptual

architecture of GNUstep provides valuable insight into how the framework is designed to enable software development while maintaining platform portability.

This study examines the essential components of GNUstep, which include the fundamental libraries and development tools that define its architecture. The libs-base library offers core Objective-C classes for collections, string manipulation, and system interaction, which serve as the framework foundation [15]. The libs-corebase library ensures compatibility with Apple's Core Foundation framework, bridging GNUstep with macOS-based applications [16]. The libs-gui library offers a comprehensive set of graphical user interface components that enable developers to create fully functional desktop applications [3]. Complementing this, the libs-back component serves as an abstraction layer that allows the rendering of graphical elements on different platforms, including X11 and Windows GDI, ensuring cross-platform compatibility [17]. Lastly, the Gorm tool, a visual GUI builder similar to Apple's Interface Builder, provides a graphical environment for designing user interfaces without requiring manual code implementation [4].

By analyzing these components, this study aims to identify the architecture styles utilized in GNUstep, examine the concurrency mechanisms, and evaluate its external interfaces and primary use cases. This inquiry will contribute to a deeper understanding of how GNUstep functions as a flexible and portable development framework, demonstrating its role in facilitating cross-platform program development.

3.0. Architecture

3.1. Conceptual Architecture Overview

GNUstep's architecture consists of a physical collection of subsystems that provide core functionalities towards application development. GNUstep's subsystems are structured into four domains: System, Local, Network, and Users [6]. The domains are composed of subdirectories that enable compatibility with OpenStep, NeXTstep, and OS X [1].

These domains use abstract data types (ADT), also known as objects, to encapsulate data and operations that preserve integrity and are designed as collections of autonomous interacting agents. Objects are written in Objective-C, an object-oriented programming language, and interact with other objects mainly through method invocation [1].

The system's interactions can be structured into different layers based on increasing levels of abstraction. Starting with the outermost layer, there is the application layer, then the presentation layer, the rendering layer, and at the bottom is the core layer. Objects in each layer invoke methods to interact with other objects in adjacent layers, but GNUstep permits the exception to allow non-adjacent layers to communicate directly.

Additionally, GNUstep's base library implements notification center objects that follow an implicit invocation design. An object can broadcast and/or register for notifications instead of invoking a method directly [10]. Events are handled like a stack using responder objects [14]. GNUstep primarily follows a layered object-oriented model that includes certain objects that follow an implicit invocation design.

The architecture is structured accordingly to focus on the main functional and nonfunctional requirements: portability, performance, integration, and modifiability. GNUstep also implements concurrency that increases efficiency and performance, making GNUstep a high-level system for developing applications.

GNUstep allows implementations of external interfaces that ease the interactions between its core libraries, development tools, and external systems. This report focuses on the core libraries consisting of gnustep-corebase, gnustep-base, gnustep-back, and gnustep-gui, as well as, the development tool, gorm. Other development tools that are outside of the report's scope include make and ProjectCenter.

3.2. Subsystems

The subsystems of GNUstep are four domains in which the directories of a GNUstep installation are organized into. These domains are: System, Local, Network, and Users. Each domain contains subdirectories as specified by the GNUstep Filesystem Layout. This configuration of subsystems enables compatibility with OpenStep, NeXTstep, and OS X in addition to traditional Unix file system conventions [1].

System

The System domain contains default GNUStep installation and distribution files, gnustep-make, basic GNUstep libraries such as Foundation and AppKit as well as Base and GUI libraries, essential system applications like the Workspace Manager and other system administrative task applications, developer applications such as ProjectCenter and Gorm, essential extensions, and software specific to the manufacturer of the distribution [6].

Local

The Local domain is the default location for users to install all non-standard GNUstep software. In this domain you will find third party applications and custom libraries with their corresponding header files. This domain is not intended for distributor specific software; distributors should manually override this default location to instead include their distribution specific software in the System domain [6].

Network

The Network domain consists of exported files from a network's central server, or from various workstations. This domain is by default combined with the Local domain in

the standard GNUstep filesystem layout. Users can install a separate variant of the default filesystem layout that includes a Network domain [6].

User

The User domain contains GNUstep related files whose only intended access is the user who owns them. The GNUstep default database, temporary data, and user installed applications and tools can all be found in this domain [6].

An example of the filesystem layout of a domain is included in the image below:

Figure 1

Note: GNUstep. (n.d.). [Example of the structure of a domain for GNUstep]. Retrieved February 12, 2025, from https://www.gnustep.org/resources/documentation/User/GNUstep/filesystem_5.html#SEC5

Knowing the directory structure of the domains and the associated terminology is important for discussions pertaining to the architecture of GNUstep [6].

3.3. Global Control Flow

At its core, the system consists of classes that are normally inherited from NSObject [18]. These classes that interact with other objects in a manner that parallels the interaction of objects in the physical world.

Objects primarily interact through method invocations. An object can independently function on its own when requested by other objects, but also consist of methods that act on instance variables (data) in response to messages [1]. Two objects can also respond to the same message in different ways [1].

When a message is sent, the run-time system invokes a specified method and determines which code to execute according to the object type [1]. Each message tells the object to do something or asks for information. The base library also offers distributed objects that allow communication with other objects between different processes running on the same machine or on different machines on the same network [9]. The details of this process is hidden inside the run-time system and the base library, which allows the user to concentrate on what the object does rather than how it is implemented [1].

The primary interactions between objects use method invocations, but GNUstep also supports implicit invocation. The NSNotificationCenter object is implemented to allow objects to send notifications to other objects that are registered to receive these notifications as they are posted [10]. Much like distributed objects, the NSDistributedNotificationCenter object involves inter-process communication [11].

For event-driven programming, specifically within user input, events are handled with NSResponder objects that are linked in a chain that acts like a stack [14]. When an event occurs, it's passed along the responder chain until an object handles it [14]. Users interact with the application via GUI components or CLI commands and these events are processed and dispatched by the GUI library that interacts with the backend library [3].

The system can be generalized into layers of objects where each layer provides services to the one above it and interacts with lower layers via well-defined APIs. The application layer is the outermost layer where developers create applications built on top of GNUstep using gorm [13]. Below, is the presentation layer where the GUI library provides UI components for user interactions and event processing [3]. The presentation layer interacts with the rendering layer where the backend library converts high-level UI commands into system-specific rendering instructions [3]. Lastly, the core layer includes the corebase and base library that provides fundamental classes to the upper layers [12]. It is important to note that objects are not limited to interacting only with objects from adjacent layers, however, this structure provides a hierarchical abstraction of the system's interactions.

The global control flow and interactions within GNUstep primarily is consistent with the object-oriented style, but also combines aspects from the implicit-invocation and layered styles to achieve high performance, portability, modifiability.

3.4. Requirements

3.4.1 Functional Requirements

<u>The purpose of GNUstep</u>

The purpose of GNUstep is to provide developers with an object-oriented tool development kit. Using GNUstep provided libraries, users can develop non-graphical tools in Objective-C. Users should be able to build, install, and package their tools using the provided gnustep-make package [8]. Users should also be able to create complex graphical applications using GNUstep's ProjectCenter and Gorm [7].

<u>Portable</u>

GNUstep is intended to be a cross platform development environment, allowing developers to build their tools and applications on GNU/Linux, Windows, FreeBSD, OpenBSD, NetBSD, and Solaris, alongside any POSIX compliant UNIX based operating system [8].

3.4.2 Non-functional Requirements

Performance

GNUstep's ProjectCenter and Gorm should allow users to fully develop their graphical applications in a short time (weeks or months) compared to similar development environments which might take years. This is due to GNUstep's simple API, which took decades to develop, and its utilization of an optimized Foundation library [7].

Integration

Due to GNUstep's use of the language Objective-C, users can easily integrate tools and applications made in GNUstep with C libraries and functions. Objective-C can also be interfaced with other languages such as Guile, Java, and Ruby [7].

Modifiable

Due to the hierarchy of domains in a GNUstep installation, distributors can make changes to the main system, such as including additional packages and libraries specific to their distribution, by altering files in the System domain. Users can also customize their experience by installing library extensions in the Local domain without the chance of altering core components of GNUstep's functionality [7].

3.5 Concurrency

GNUstep provides concurrency mechanisms similar to those in Apple's Cocoa framework, leveraging Objective-C, POSIX threads, and Grand Central Dispatch (GCD) where available. Concurrency in GNUstep allows applications to perform multiple tasks simultaneously, improving responsiveness and performance.

Concurrency mechanisms include NSThread, NSOperationQueue, and Grand Central Dispatch (GCD). NSThread enables explicit thread creation and manual thread management. Developers can create and control threads directly but must handle synchronization manually. NSOperationQueue provides task-based concurrency, where tasks are encapsulated as NSOperation objects and executed asynchronously in the background. It allows dependency management, ensuring that some tasks execute only after others complete. GNUstep has partial support for GCD, Apple's low-level concurrency API. GCD provides dispatch queues, which allow easy execution of tasks on background threads.

Secondly, Synchronization and Thread Safety Concurrency introduces potential race conditions and data corruption. GNUstep provides synchronization mechanisms such as @synchronized, NSLock, and Atomic Properties. @synchronized is an Objective-C locking mechanism that ensures only one thread executes a critical section at a time. NSLock provides explicit locking mechanisms for synchronization. Atomic Properties ensures thread safety for individual property accesses but does not provide full synchronization.

Thirdly, asynchronous programming in GNUstep supports event-driven programming, particularly in GUI applications, through NSNotificationCenter and NSRunLoop, allowing efficient handling of asynchronous events.

Overall, GNUstep's concurrency mechanism supports cross-platform compatibility along with other functional and non-functional requirements. Portability is accomplished with NSOperationQueue or NSThread. Thread safety is implemented with GNUstep's base library (Foundation Kit) since it is mostly thread-safe, while GNUstep's GUI library (AppKit) is not thread-safe. All UI updates must be done on the main thread. Optimized performance is done with NSOperationQueue for high-level concurrency management, but a preference to GCD if available for efficient task scheduling, and avoid excessive thread creation by using queues for workload management.

4.0. External Interface

GNUstep provides several external interfaces that facilitate interaction between its core libraries, development tools, and external systems. These interfaces allow components to communicate with one another, making program development and runtime execution possible across various platforms.

One of the most important interfaces in GNUstep is the programming interface (API). GNUstep implements the OpenStep specification and supports Apple's Cocoa API, ensuring compatibility with Objective-C-based software development [2]. The main libraries: libs-base, libs-corebase, and libs-gui, provide extensive APIs that support functionality for fundamental data structures, GUI components, and system interaction [3]. The libs-base framework includes essential Objective-C classes for collections, string manipulation, and threading [15]. Meanwhile, libs-gui offers GUI-related classes that enable the development of graphical applications that include components such as windows, buttons, and menus [2].

In addition to the API, GNUstep contains a collection of development tools that act as external interfaces for creating and managing applications. Gorm, the visual GUI builder, offers an interface for developing user interfaces using a drag-and-drop feature, similar to Apple's Interface Builder [4]. It interacts with libs-gui to generate and manage UI components programmatically. Moreover, Make and ProjectCenter serve as development tools that communicate with the GNUstep build system, allowing developers to easily compile, link, and deploy applications [5].

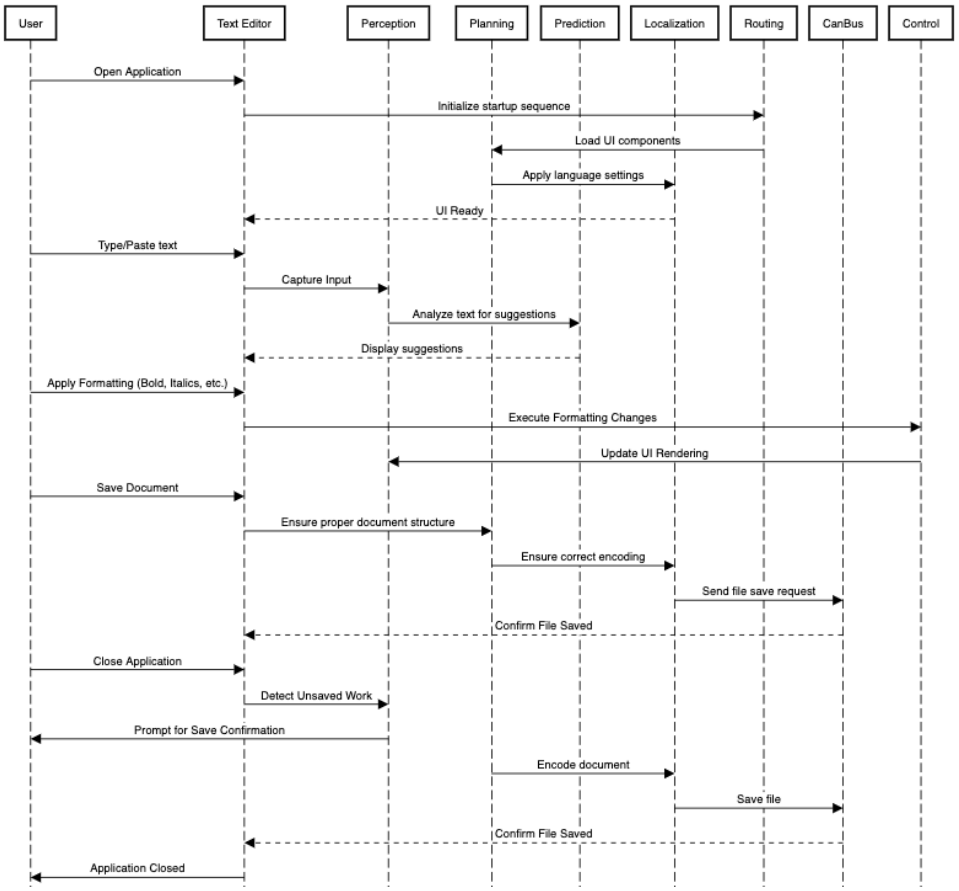
Furthermore, GNUstep's system-level integration guarantees compatibility with a variety of operating systems, including Linux, BSD, and Windows. The libs-back component serves as an abstraction layer that bridges GNUstep's GUI rendering with different backend systems such as X11 and Windows GDI [17]. This enables GNUstep applications to run on multiple platforms without modification. Additionally, GNUstep communicates with system libraries and external services to provide file system access, networking, and process management, allowing for a smooth integration with the host operating system [2].

Finally, GNUstep applications offer graphical user interfaces to end users. The GUI elements generated by libs-gui enable applications to present intuitive interfaces that interact with users via input events such as keyboard and mouse interactions [3]. Additionally, command-line tools integrated within GNUstep provide an alternative interface for configuring and managing applications, offering flexibility for both power users and developers.

By leveraging these external interfaces, GNUstep maintains excellent portability, interoperability, and development ease, ensuring that applications can be created and executed across diverse environments with minimal adaptation

5.0. Use Case

5.1. Use Case #1: Text Editor



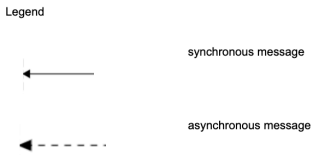
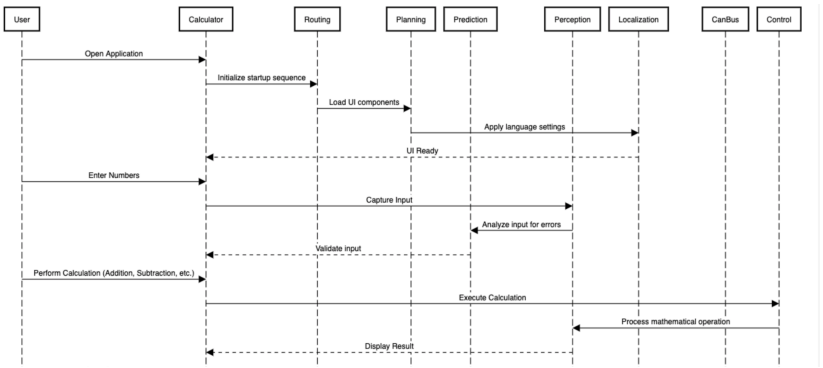


Figure 2: Sequence diagram for use case 1

This use case represents using GNUstep structured approach to text editing. When the user launches the text editor, the system initializes the startup sequence. The perception and planning modules are activated to load UI components and apply language settings. This aligns with GNUstep's object-oriented design, where UI elements are managed through object-oriented principles. As the user types or pastes text, the perception module captures the input, while the prediction module analyzes the text for grammar suggestions using its built-in rules for grammar prediction. This is all processed within GNUstep's object model, which ensures that UI responsiveness and text processing operate independently but cohesively. Using formatting actions such as bold, italics, etc. trigger updates to the UI rendering. GNUstep's architecture enables for simple modifications to select areas of the interface. Formatting instructions can modify the appearance of fundamental UI components without affecting other elements. Then, the save operation first checks for proper document structure and encoding before sending a file save request. This calls routing and control modules that manage data. Lastly, before shutting down, the system detects unsaved work and prompts the user for confirmation. If the document is saved, the encoding and file-saving processes are triggered, ensuring data integrity. This structured approach aligns with GNUstep's event-handling mechanisms and state management.

5.2. Use Case #2: Calculators



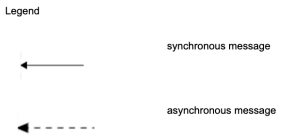


Figure 3: Sequence diagram for use case 2

This second use case displays GNUstep as a calculator. It demonstrates the modular approach to handling user interactions, UI rendering, and computational operations. When the user opens the calculator application, the system initiates a startup sequence managed by the routing module, and the planning module ensures that UI elements are properly loaded. The prediction module applies language settings, adapting the interface to system preferences for accessibility and localization. Once the interface is initialized, the system is ready for user input. When a user enters numerical values, the calculator module reads the input, and the perception module processes it. If an error is detected, an appropriate message is displayed; otherwise, the input is processed further. When the user selects a mathematical operation such as addition, subtraction, multiplication, or division, the control module verifies that the input meets the required format. The calculator module then performs the operation using the Routing module to determine the appropriate computational method. Once the calculation is complete, the control module sends the result to the UI, where the planning and routing modules format and update the output. The object oriented architecture allows for the clear separation of concerns between the UI, logic, and system interactions, making the application easier to maintain and expand.

6.0. Limitations & Lessons Learned

Limitations

While conducting this study on the conceptual architecture of GNUstep, several limitations were encountered that affected the extent of our analysis. One major limitation was the availability and organization of official documentation. While GNUstep is an open-source software with extensive resources, much of the documentation is fragmented across multiple sources, making it difficult to combine information into a unified architecture model. Some components, such as libs-corebase, have insufficient documentation that necessitates the use of source code and external references, which may result in minor inaccuracies to the actual functions of the library.

Another limitation was the complexity of GNUstep's architecture. The framework is designed to be very flexible and adaptable, resulting in interactions between subsystems that are

not necessarily explicitly described in the documentation. The availability of several backend implementations, including X11, Windows GDI, and etc., complicates establishing the exact layout of the system. Additionally, because GNUstep maintains compatibility with OpenStep and Cocoa, some design choices impact how modern components interact, making certain aspects of the architecture less intuitive.

Time constraints also posed a challenge, as the analysis was conducted within a limited timeframe. Due to the necessity to thoroughly analyze both documentation and source code, some parts may not convey the entire scope of GNUsteps' capabilities. Future research could delve more into concurrency mechanisms and how GNUstep manages multi-threading in large-scale applications.

Lessons Learned

Despite these limitations, the process of analyzing GNUstep's conceptual architecture provided valuable insights into software design, modularity, and cross-platform development. One key takeaway was the importance of architecture consistency in long-term software maintenance. GNUstep's adherence to the OpenStep standard, despite current revisions, highlights how well-defined architectural principles contribute to software longevity and adaptability across different platforms.

Another significant lesson was the importance of documentation in open-source projects. The challenges faced due to fragmented information underscored the necessity for clear and centralized documentation, especially for complex frameworks. The experience reinforced the need for developers to prioritize maintainable documentation to facilitate future development and research.

Additionally, this study provided a deeper understanding of layered and object-oriented architectural styles. GNUstep's architecture follows a structured layering approach, where different libraries provide well-defined services to the layers above them. The modular design ensures scalability and flexibility, serving as an excellent case study in software engineering best practices.

Finally, the analysis improved our ability to reverse-engineer conceptual architectures from documentation and source code. By piecing together information from various sources, we gained practical experience in reconstructing architectural models, a crucial skill in software engineering. This project demonstrated the challenges and the benefits of working with legacy architectures while adapting them to modern development environments.

Overall, while limitations in documentation and time constraints affected the depth of analysis, the study of GNUstep provided significant learning opportunities in understanding the software architecture, modular design, and platform system interactions.

7.0. Conclusion

This study examined the conceptual architecture of GNUstep, an open-source framework for cross-platform GUI and server application development. By analyzing its documentation and

source code, this study identified its key components, architectural styles, and system interactions.

GNUstep follows a layered object-oriented model, where libs-base provides essential Objective-C classes, libs-corebase ensures compatibility with Apple's Core Foundation, libs-gui supports graphical user interfaces, and libs-back enables platform-independent rendering. The framework employs layered and event-driven architectures, utilizing distributed objects and notification systems like NSNotificationCenter for inter-process communication. Additionally, GNUstep incorporates concurrency mechanisms such as NSThread, NSOperationQueue, and Grand Central Dispatch (GCD) to improve performance.

The subsystem organization within GNUstep further reinforces its modular design. Its four primary domains—System, Local, Network, and User—logically separate system-wide, third-party, network-shared, and user-specific resources, ensuring compatibility with OpenStep, NeXTstep, and Unix-based file systems.

This investigation reconstructs GNUstep's architecture and highlights its modularity, layering, and structured component interactions, which contribute to its scalability and adaptability. GNUstep's compliance to the OpenStep specification, as well as its support for distributed computing and concurrency, make it a reliable cross-platform development environment. Understanding its design provides valuable insights into software modularity, system structuring, and scalable framework development.

8.0. Glossary

8.1. Data Dictionary

API - Abbreviation for Application Programming Interface.
CLI - Abbreviation for Command-Line Input.
File System Domains – The hierarchical structure of GNUstep's file organization, divided into System, Local, Network, and User domains.
GNUstep – An open-source framework implementing the OpenStep specification, designed for developing cross-platform GUI and server applications.
Gorm – A visual GUI builder, similar to Apple's Interface Builder, used for designing user interfaces.
Grand Central Dispatch (GCD) – A low-level concurrency API that provides thread pool management and efficient task scheduling.
GUI - Abbreviation for Graphical User Interface.
libs-back – A rendering system library responsible for abstracting different backend implementations (e.g., X11, Windows GDI).
libs-base – A core library providing fundamental Objective-C classes for collections, threading, and system interaction.

libs-corebase – A compatibility layer library ensuring integration with Apple's Core Foundation framework.
libs-gui – A library providing graphical user interface components such as windows, buttons, and menus.
Make - A package system of make commands that is designed to encapsulate all the complex details of building and installing various types of projects from libraries to applications to documentation.
NSDistributedNotificationCenter – A mechanism for inter-process communication within GNUstep, allowing objects in different processes to exchange notifications.
NSLock - Provides explicit locking mechanisms.
NSNotificationCenter – A notification system within GNUstep that enables implicit invocation by broadcasting events to registered objects.
NSObject - The root class of the GNUstep base library class hierarchy. All classes should inherit (directly/indirectly) from NSObject.
NSOperation - An operation that needs to be executed.
NSOperationQueue – A task-based concurrency model that manages execution order and dependencies between tasks.
NSResponder - Methods that are overridden by subclasses for receiving events, from simple user inputs to abstract events like text selection or text modification
NSRunLoop - handles various utility tasks that must be performed repetitively in an application.
NSThread – A threading mechanism that allows manual thread management within GNUstep applications.
Objective-C – The primary programming language used in GNUstep, supporting object-oriented design and dynamic runtime features.
OpenStep Specification – A set of application programming interface (API) standards originally developed by NeXT, forming the foundation for Apple's Cocoa API.
POSIX threads - An execution model that exists independently from a programming language, as well as a parallel execution model.
Project Center - The graphical integrated development environment (IDE), integrated with Gorm.
UI - Abbreviation for User Interface.

8.2. Naming Conventions

@synchronized – A keyword in Objective-C used for thread-synchronization and locking mechanisms.
Cocoa API – Apple's API for macOS development, which GNUstep maintains compatibility with through its libraries.
libs – Used as a prefix for library names that are part of the GNU framework (e.g., libs-back, libs-corebase, libs-gui, and libs-back).

NS – Stands for NextStep, a prefix used for class names and operations inherited from OpenStep (e.g NSThread and NSResponder).
OpenStep – The API specification for cross-platform Objective-C development, originally defined by NeXT and later influencing Apple's Cocoa framework.

9.0. References

[1] Botto, F., Frith-Macdonald, R., Pero, N., & Robert, A. (2001-2004). *Objective-C language and GNUstep base library programming manual*. Free Software Foundation. Retrieved from http://andrewd.ces.clemson.edu/courses/cpsc102/notes/GNUStep-manual.pdf

[2] GNUstep. (2024). *GNUstep API documentation*. Retrieved from https://www.gnustep.org

[3] GNUstep. (2024). *AppKit documentation*. GNUstep MediaWiki. Retrieved from https://mediawiki.gnustep.org/index.php/AppKit

[4] GNUstep. (2024). *Gorm.app documentation*. GNUstep MediaWiki. Retrieved from https://mediawiki.gnustep.org/index.php/Gorm.app

[5] GNUstep. (2024). *ProjectCenter.app documentation*. GNUstep MediaWiki. Retrieved from https://mediawiki.gnustep.org/index.php/ProjectCenter.app

[6] GNUstep filesystem hierarchy document: Structure of a domain. (n.d.). Retrieved from https://www.gnustep.org/resources/documentation/User/GNUstep/filesystem_5.html#SEC5

[7] (N.d.). Retrieved from https://www.gnustep.org/information/GNUstep-brochure.pdf

[8] Introduction to GNUstep. (n.d.). Retrieved from https://www.gnustep.org/information/aboutGNUstep.html

[9] Pero, N. (2010). *1 What are Distributed Objects (DO).* GNUstep Distributed Objects. Retrieved from https://www.gnustep.org/nicola/Tutorials/DistributedObjects/node1.html

[10] Frith-Macdonald, R. and Kachites McCallum, A. (2025). *NSNotificationCenter class reference.* GNUstep. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/NSNotification.html#class$NSNotificationCenter

[11] Frith-Macdonald, R. and Kachites McCallum, A. (2025). *NSDistributedNotificationCenter class reference.* GNUstep. Retrieved from

https://www.gnustep.org/resources/documentation/Developer/Base/Reference/NSDistributedNotificationCenter.html#class$NSDistributedNotificationCenter

[12] GNUstep. (2012). *Foundation.* GNUstep MediaWiki. Retrieved from https://mediawiki.gnustep.org/index.php/Foundation

[13] GNUstep. (2024). *Gorm.* GNUstep. Retrieved from https://www.gnustep.org/experience/Gorm.html

[14] Armstrong, C. (2006). *Using the GNUstep AppKit*. Griffith University. Retrieved from https://www.ict.griffith.edu.au/teaching/2501ICT/archive/resources/documentation/Developer/Gui/ProgrammingManual/AppKit.pdf

[15] GitHub. (n.d.). *gnustep/libs-base releases*. GitHub. Retrieved from https://github.com/gnustep/libs-base/releases

[16] GitHub. (n.d.). *gnustep/libs-corebase*. GitHub. Retrieved from https://github.com/gnustep/libs-corebase

[17] GNUstep. (2024). *Back - GNUstep Wiki*. GNUstep MediaWiki. Retrieved from https://mediawiki.gnustep.org/index.php/Back

[18] Frith-Macdonald, R. and Kachites McCallum, A. (2025). *NSObject class reference.* GNUstep. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/NSObject.html#class$NSObject

[18] GNUstep examples. (n.d.). https://www.gnustep.org/experience/examples.html