CISC 322 – Software Architecture

Queen's University

School of Computing

GNUstep Conceptual Architecture

Assignment 1

February 14, 2025

Spencer Hum
21seh19@queensu.ca

Aidan Flint
21akf3@queensu.ca

Tony Luo
tony.luo@queensu.ca

Brant Xiong
21zx38@queensu.ca

Anson Jia
21aj85@queensu.ca

Zhangzhengyang Song
20zs57@queensu.ca

Table of Contents

Abstract

In this report, we study the conceptual architecture of the software project GNUstep, a set of cross-platform object-oriented frameworks written in Objective-C, along with surrounding tooling for developing desktop applications. GNUstep aims to provide a user-friendly and simple platform for developing both graphical and non-graphical software applications. As a preliminary for our study, we begin by presenting an overview of the core components, which include the base library(libs-base), corebase library(libs-corebase), GUI library (libs-gui), backend library (libs-back), and Gorm. Furthermore, we outline the concrete organization of the codebase at the source code level, providing a clear image of its organization and layout.

From GNUstep's evolution from inception in the NeXTSTEP operating system to its current configuration, the conceptual architecture are examined from five different perspectives: how components interact with each other; what are the mechanisms for control and data flow across and within components; how the system facilitates evolution, how concurrency is supported and what are the developer responsibilities. This in-depth examination offers a clear understanding of how these elements work together to shape GNUstep's functionality and style.

The report also further examines the architectural style that is embedded within GNUstep and evaluates the effectiveness of fulfilling functional and nonfunctional requirements of the system. In addition, we further explain key use cases that show the applicative uses of the system. We conclude by summarizing key findings, issues that were faced during the study, and conclusions drawn from the analysis.

1 Introduction

The goal of this report is to document the conceptual architecture of the software GNUstep through documentation available on the software website and examining its source code. GNUstep is a comprehensive system layered on top of the operating system that comes with In this report, we focus on the key components: libs-base v1.31.0, libs-back v0.32.0, libs-gui v0.32.0, libs-corebase master branch and Gorm v1.4.0. The first four libraries make up the GNUstep programming layer, while the latter application is a development tool for building

user interfaces (UIs). Due to the disparate nature of development tools versus libraries, we opt to split the analysis of the conceptual architecture into two independent streams. We propose a layered architecture for the entire GNUstep framework comprising the core libraries: libs-base, libs-corebase, libs-gui, libs-back and a repository architecture for the Gorm integrated development environment (IDE). To derive the architecture of Gorm, we will examine the Gorm IDE from a development point of view. Specifically, we will study how different subsystems of Gorm interact and function to facilitate interface programming. To derive the architecture of the GNUstep programming layer, we will scrutinize the core libraries in a runtime context, that is we will study how messages and control get passed between the core libraries in reaction to application events, focusing primarily on graphical aspects.

In this section, we provide a brief history of the GNUstep project and a survey of the project's organization at the component and source code level.

1.1 Overview/System Evolution

GNUstep is a free and open source object-oriented framework for developing desktop applications. The framework consists of two main components that collectively implement the OpenStep specification. The three components are respectively, the GNUstep base library (libs-base) and the GNUstep GUI library which we refer to as the core libraries. In addition to adhering to the OpenStep specifications, GNUstep implements many custom extensions deemed of utility and strives to support current versions of Apple's Cocoa application programming interfaces (API) which is an evolution of those defined in OpenStep.

The origin of GNUstep traces back to the development of the NeXTSTEP operating system (OS). In 1989, NeXT Computer Inc released version 1.0 of their revolutionary NeXTSTEP. Following the hype brought about by the pioneering technologies introduced, the GNU project wanted to create its own implementation of the NeXTSTEP programming environment. The primary motivation being to leverage the advantages of the Objective-C language in order to develop tooling that would allow GNU applications to run on multiple platforms with minimal effort [1]. Developments started in parallel on the foundation library and object layer. Fast forward to 1994, NeXt Computer Inc and Sun Microsystems Inc released a platform-independent object-oriented application programming interface (API) specifica-

tion based on the NeXTSTEP APIs for which other implementations can adhere to. Prior efforts on writing the foundation library and object layer converged with interests in implementing this new specification. The resulting software project became known as GNUstep, whose goal was to turn the previous scattered implementation into a real OpenStep compliant system [2]. Since then, the GNUstep project has pushed the ideas of OpenStep into different domains and grew into a powerful development environment. It has also garnered a vibrant ecosystem based around the OpenStep specification with many new libraries and tools written by contributors.

1.2 System breakdown

GNUstep is implemented as a system that sits on top of the OS. It is stratified into two layers: the application layer, and the base layer. These corresponds to the Application Kit (AppKit) and Foundation Kit (FoundationKit) libraries in the OpenStep specification respectively [3]. The GNUstep GUI library implements the AppKit. while the GNUstep base libraries libs-base and libs-corebase are alternative implementations of the Foundation Kit. The GNUstep GUI library is historically designed to be split into two parts, a frontend part that is entirely independent of the platform and display system and a backend part that fills in the necessary platform-dependent display operations. During early stages of implementing AppKit, the GNUstep developers quickly realized that the Display Postscript System (DPS) as specified in OpenStep would require significant amount of work that could hinder the completion of AppKit. As a result, they decided to split the GUI library into platform-independent and platform-dependent pieces so AppKit can be implemented without a functional DPS implementation [1]. A notable consequence of this design choice is that GNUstep application can match the "look and feel" of the underlying display system without changing their source code [4].

libs-gui is the frontend part of the GNUstep GUI library. It is completely written in the Objective-C language and provides classes for graphical elements such as buttons, text fields, popup lists, browser lists, and windows along with helper classes to work with notifications, events and handle tasks like color and font management [4]. Crucially, it does not handle drawing the graphical elements to the display system.

libs-back is the backend part of the GNUstep GUI library. It is completely written in the

Objective-C language and provides concrete the low-level methods needed to draw graphical elements as well as managing windows. There are two types of backends present in libs-back. A graphics backend which supports drawing to the screen, and a window server backend which handles the presentation and organization of windows. For each graphics backend, it implements DPS emulation engine to emulate the DPS functions expected by the frontend library [5]. For each window server backend, it creates a shim layer to bridge the interface expected by the frontend and the specific backend's interface.

libs-base is the main and actively developed implementation of FoundationKit written in the Objective-C language that follows Apple's Cocoa framework. It contains basic functions and routines for data manipulation, network and file operations, datetime handling and other primitive system functionalities. Moreover, libs-base define several important mechanisms used by the entire GNUstep system relating to object communications. It provides a powerful set of classes that permit remote messaging between objects in different processes and a distinct set of classes that permit messaging between objects within the same process called notifications. These two approaches make up the communication model in GNUstep [6].

libs-corebase is an alternative implementation of FoundationKit written in the C language that follows Apple's Core Foundation framework. It shares much of the same functions as libs-base but is grants even more low-level functions in some cases. A unique feature of libs-corebase is its interoperability with libs-base via "toll-free bridging". Toll-free bridging allows certain types in libs-corebase and libs-base to be freely interchanged while maintaining the same underlying representation [7]. At a high level, this allows for "casting" types in one library to a corresponding type in the other across language boundary. Due to the fact that the vast majority of GNUstep is centered around Objective-C, it is difficult to find nontrvial use cases of lib-corebase. Therefore we concentrate our attention on libs-base when analyzing the GNUstep base library.

Gorm is an integrated development environment for designing user interfaces for a GNUstep desktop application. It presents a visual interface to edit and create layouts for user interfaces as well as tools to connect outlets of objects to actions. The objects that compose the layout and their connections are exported to (.gorm) interface files. A desktop application can then load and instantiate these interface files in order to display them. This allows for a clean

separation between user interface layout and application logic.

The codebase for the GNUstep project is hosted on software forge GitHub under the GNUstep organization account. Libraries, applications and tools are organized into individual units and stored in repositories following a strict naming scheme. Repositories for libraries begin with the prefix `libs`, repositories begin with `apps` and so on.

Gorm is structured as a composition of palettes. The global application does not perform any work other than calling into the respective palettes. The palettes that make up the workspace are found under the Applications/Gorm directory. The work of managing and updating the interface being built is done by the GormCore library which resides in the GormCore directory. The InterfaceBuilder framework defines the model for the interface files and resides in the InterfaceBuilder directory.

For the core libraries, the appropriate subdirectory of the Headers/ directory contains all the protocols and class headers that the library exposes while the corresponding Source/ directory contains the implementation of those protocols and classes.

2 Conceptual architecture

In this section, we analyze the perspectives on conceptual architecture of the Gorm and the core libraries in parallel starting with component interactions, continuing to for system evolution, concurrency and developer responsibilities.

2.1 Component interactions

For the core libraries, libs-gui and libs-back communicate through a DPS emulation layer. This is realized by two abstract classes defined in libs-gui that perform raw display operations: `GSDisplayServer` and `NSGraphicsContext`. `GSDisplayServer` is an abstract class that provides a framework for a device independent window server. A window server manages basic control of the computer display and input. Some of these controls include window creation and destruction, event handling, icons, cursors. `NSGraphicsContext` is an abstract class that provides a framework for device independent drawing. This object contains routines to read and write raw pixel data to the display. It also offers a convenience function interfaces such

as DPS that allows drawing more complex geometric objects. Typically, the window server sets up a window and tells the context where and how to draw on the screen through the set of methods under `NSGraphicsContext` with the `GSSet` prefix. libs-back interacts with libs-gui by providing concrete subclasses of `GSDisplayServer` and `NSGraphicsContext` for various window servers and platform-dependent graphics API.

2.2 Concurrency

Concurrency is the program's ability to execute tasks synchronously without degrading the performance of individual components. A GNUstep application may use concurrency facilities to perform background tasks such as network requests and data processing without affecting the responsiveness of the main user interface. The core libraries support concurrency through multi-processing and multi-threading facilities defined in the OpenStep specification which are provided by libs-base. A GNUstep program may spawn a separate process, called a task or spawn multiple threads within the current process. Threads share data with the spawner process while tasks do not. Multi-threading is supported via NSThread, which is a high level interface for the creation and management of the different threads of the program. NSLock and NSCondition are classes that coordinate processes and prevent race conditions. NSOperation and NSOperationQueue are also provided to schedule process execution based on its dependencies, as well as supporting completion and cancellation blocks. Each thread in the program has an associated NSRunLoop, which handles tasks that must be performed repetitively such as listening for events such as timers, processing peripheral inputs, sending notifications, and firing timers. NSTimer is a class that schedules sending messages at some time in the future. NSNotifications are events that objects can subscribe to from the NSNotificationCenter. This is the main form of asynchronous communication provided by libs-base. Lastly, remote procedure calls and remote messaging are supported via the NSConnection and NSDistantObject classes.

Concurrency in Gorm is handled by the GormCore library which proxies access to the global GormDocument. Gorm itself does not employ threads or mutli-processing, thus only one subpalette is active at a given time. This means race conditions can not occur when writing and reading to the document. Furthermore, individual subpalettes do not store any state about the global document, so the view of the global document is consistent throughout

the process. This means no updates need to be propagated when a write operation is issued through GormCore. These design decisions allow Gorm to remain relatively efficient without introducing the inherent complexities of threads and tasks.

2.3 Developers responsibilities

GNUstep is an open-source framework implementing the OpenStep standard to provide a simple cross-platform development environment to easily and rapidly develop graphical applications using Gorm's graphical user interface builder or non-graphical tools by employing their large library of classes and other software components.

The GNUStep project is an open source initiative where contributors may come from volunteers or organizations with varying levels of development involvement and there are many general roles of the software development process. Other developers focus on improving and expanding the environment, tools, and GUI interface for an improved user experience. Developers work on maintenance and improving GNUStep's core libraries to ensure compatibility for cross platforms such as Cocoa. Contributors review and ensure the code adheres strictly and guarantees that all quality assurance guidelines are met and that code updates don't introduce bugs or regression in system performance. Build and deployment management is an important process where developers need to create and manage packages to ensure a stable build for smooth integration. Developers must guarantee the system's security and stability by identifying, addressing and monitoring any potential security vulnerabilities and ensuring system reliability through extensive testing. Lastly, version control is necessary for tracking, documenting and publishing updates on a changelog of the development process to maintain transparency to collaborators or users.

2.4 Architecture style

We are now ready to derive the architecture for the core libraries and Gorm.

Gorm are implemented as a collection of plugins called palettes supervised by the main application controller. Each main palette is a menu in the Gorm interface and contains subpalettes that perform various editing functionality. For example, the toplevel Control palette is a menu containing inspector subpalettes that allows viewing information about

various controls. The project being edited is represented as a GormDocument that contains all information about the interface being edited and how they are connected. This corresponds to the blackboard data structure in the repository architecture style. GormCore serve as the bridge that allow subpalettes to update the document. When an attribute of a graphical elements in the document needs to be updated, the responsible subpalette issues a call to the appropriate functions in GormCore which validate and performs the operation. The update is then propagated when subpalettes access the document again. We see that the subpalettes that access the GormDocument corresponds to knowledge sources that independently read or write to it while control is initiated by the knowledge sources. Thus we identified the three major parts of the blackboard model that is a key invariant in the repository architecture. The presented evidence leads us to conclude that Gorm follows a repository architecture.

An alternative architecture for Gorm would be an implicit invocation (pub-sub) architecture. The current architecture for Gorm places control in the hands of the subpalettes, they are the units that initiate access to the GormDocument. The advantages of this style is that access and concurrency is robust and centrally managed. The disadvantages of this style is that it is difficult to support distributed processing. Using implicit invocation inverts this control so that subpalettes subscribe to events from the main document which collects all incoming requests and update the document accordingly. The advantages of this style is that components can be loaded or swapped at runtime which facilitates system evolution and distributed processing can be easily supported. The disadvantage of the style is that concurrency becomes more complex to manage as there isn't a single consistent view of the global state. Both styles have benefits and drawbacks which must be evaluated based on the supported use case and technical consideration.

The core libraries are split into two fundamental layers: the base layer and the application layer. The base layer is made up of libs-base or libs-corebase which provides basic system functions. The application layer is made up of libs-gui and libs-back which provides graphical elements and utilities to support them. There is a clear hierarchy organization since the application utilizes data structures and protocols from the base layer. This is in the spirit of a layered system since each layer provides services to the layer above it and access services provided by the layer below it. However, we note that applications may also use the base

layer directly and the application layer's functionality does not subsume the base layer's functionality. In our opinion, this does not violate the core tenant of a layered architecture since each layer provides a substrate at a different level of abstraction. Moreover, The interchangeability of libs-base and libs-corebase reinforce our observation of clear boundaries between the two layers. The supporting evidence presented leads us to conclude that core libraries follow a repository architecture.

An alternative architecture for the core libraries would be a repository architecture. The current core libraries as required by the OpenStep specification corresponds to a retained mode GUI styled framework, one that is heavily revolved around objects and passing states. In a retained mode GUI styled framework, the graphical elements themselves know how to draw. A different style is immediate mode GUI, in which the graphical elements themselves do not know how to draw, and it is up to the application to manage the rendering. In the context of GNUstep, this means the direct DPS link between libs-gui and libs-back is broken and instead presented to the consumer. The consumer is then expected to use the classes in libs-gui for storing data only and call libs-back manually for rendering graphical elements to the screen. In this case, the buffer to render is the blackboard model which is shared between graphical elements which are the knowledge sources. Using an immediate mode GUI style is computationally more expensive as the applications needs to keep track of the display themselves, but the application logic in many cases becomes simpler. Once again, both styles have benefits and drawbacks which must be evaluated based on the supported use case and technical consideration.

2.5 Sequence diagrams

In this section, we present two use cases and the corresponding sequence diagrams of the GNUstep framework. One pertaining to rendering UI with the core libraries and one pertaining to editing interface files with Gorm.

Before we delve into the use cases, we establish a common convention of the graphics used in the sequence diagrams.

For demonstration purposes, consider a simple GNUstep application called "GuessTheNumber". This application generates a random number between 0 and 256 and stores it in a

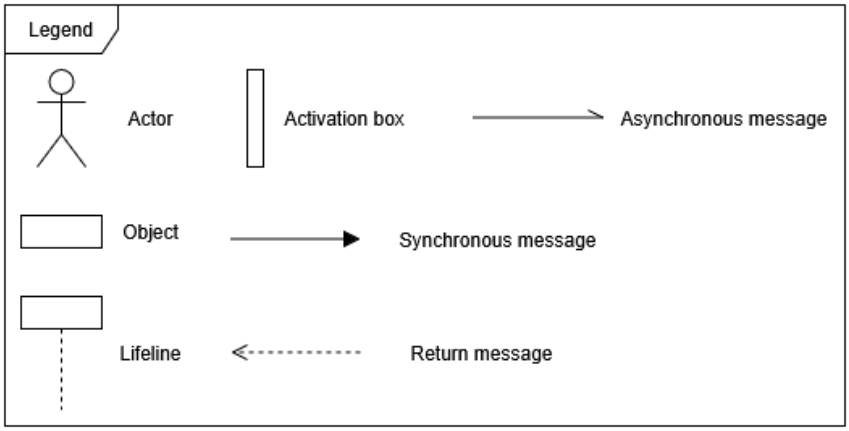

Figure 1: Sequence diagram legend

`secret` variable on startup. It presents a form with one field `number` which the user may input any integer value in the aforementioned range and a button to submit the field. Upon submission of the form, the program print whether the user input matches the secret number to screen. The use case here is the user launches the "GuessTheNumber" application, inputs a number in text field, then clicks on the button to submit the number. This use cases illustrate the activation of many parts in the GNUstep framework. In particular, the GUI rendering flow, event handling for responding to a mouse click and loading interface files. Here is the sequence diagram that shows the specific data and control flows.

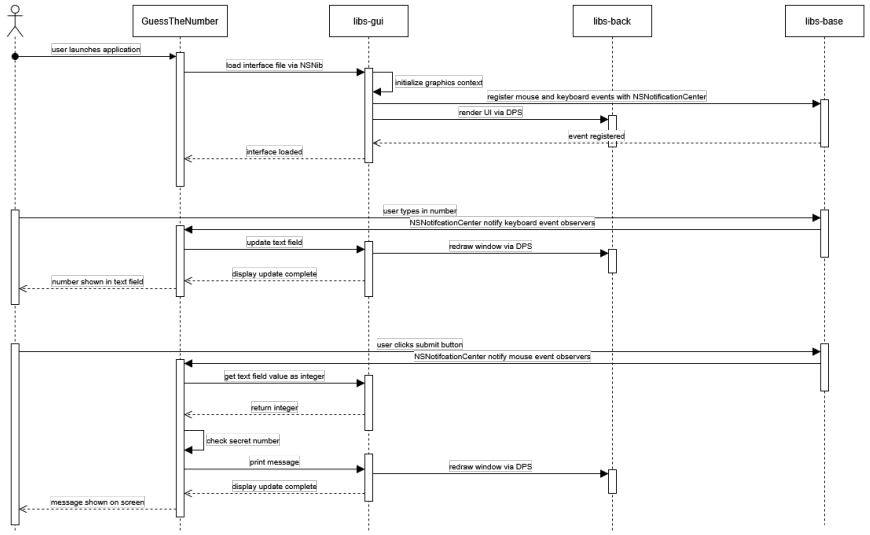

Figure 2: Sequence diagram for use case 1

Our second use case involves a developer designing the interface of a GNUstep application in Gorm. The developer use the Windows palette to drop a window into the workspace and use the Inspector palette to rename the window title to ”File Viewer”. The interface layout is then saved to the disk. For simplicity, we assume the developer has already launched the application so we may omit the startup sequence. Furthermore, we omit calls to libs-gui for rendering the graphical editor itself in order to focus on components in Gorm. This use case illustrate how palettes in Gorm work together to enable editing and how the interface files get exported. Here is the sequence diagram that shows the specific data and control flows.

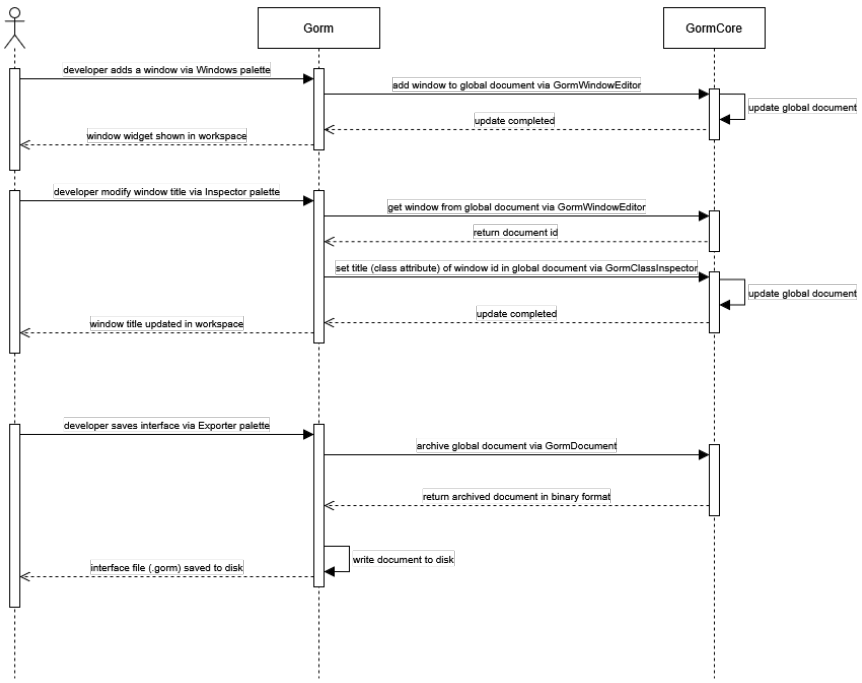

Figure 3: Sequence diagram for use case 2

3 Discussion

During this assignment, finding a way to effectively collaborate proved to be a challenge. One initial ideas was divide work between members based on the components. This task was difficult to due to the interconnected nature of the GNUstep project and our lack of experience in working in a large group. As time went on, it became increasingly clear that there was a lot of overlap and relationships between the libraries such that a global picture was needed to fully understand their function, which neither of us had by ourselves. In order to address this gap in understanding, we set up regular meetings and group work sessions to discuss the architecture and share what we have learned in our research with each other. These meetings helped us with understanding how the varying components fit within the architecture. Gradually, we transitioned to an interdisciplinary approach to research, where each group member would read up on all the components in addition to their main focus and contribute to each section. Although this lead to some duplicate work, we felt that it was beneficial in the long run as it enabled us to identify problems within the report that would

otherwise go unnoticed and validate other member's writings. Despite the advantages, we still felt that our methods of working were not ideal. We hope to explore more effective ways to collaborate in the followup assignment.

Another challenge we encountered during our research was the lack of high-level documentation. Although there were API references available for each library, they were low-level in the sense that they document individual classes, protocols and methods. From this information alone, it was very difficult to understand what the components of the system are and how they work together. We frequently needed to delve into the inner workings of the libraries through examining source code or tracing header imports to infer the dependencies and relations between units. Fortunately, the source code more than often contains comments that were essential to comprehend the architecture which we were able to utilize. If we were given an opportunity to redo this assignment, we would spend time getting GNUstep installed and work through the demos to better understand how different parts of the system functioned. This would allow us to isolate important areas of the respective libraries to focus our attention on.

4 Conclusion

The conceptual architecture of GNUstep, with its core libraries and Gorm IDE has been examined to follow a layered architecture for the core libraries and a repository style for the Gorm IDE. GNUstep does have some areas where it can be improved, the most significant one being documentation. With more variety in documentation, GNUStep could be a lot easier for the user to understand the components of GNUStep and could therefore attract a larger user base for its software. A further direction would require us to do an extensive examination of the codebase so as to accurately gauge the system and API interactions, as GNUstep has complex object relations that could not possibly be evaluated exhausted beyond the code level.

5 Resources

5.1 Glossory

GNUstep: A set of cross-platform object-oriented frameworks written in Objective-C for developing applications.
Libs-base: The fundamental library of GNUstep, providing basic functions and routines for data manipulation, network and file operations, datetime handling and other primitive system functionalities.
Libs-corebase: An alternative library, following Apple's Core Foundation framework, implemented in C.
Libs-gui: The graphical user interface library, following Apple's Core APIs, serves as the frontend part of the GNUstep GUI library.
Libs-back: The graphical user interface backend component, serves to draw graphical elements as well as managing windows.
Gorm: A graphical interface builder for GNUstep.
Application: A GNUstep program with resources.
Palette: A plugin for Gorm that extends its user interface.
Loadable bundle: A module that can be loaded by a GNUstep application for additional functionality.
Control: A simple graphical element that can be displayed on the screen.
NSThread: A high-level interface for creating and managing threads.
NSLock/NSCondition: Classes that coordinate thread access while providing an interface and protecting critical pieces of code.
NSOperation/NSOperationQueue: Classes used to schedule execution based on its dependencies and supporting completion and cancellation blocks.
NSRunLoop: A class that listens for events such as timers, network inputs and user interactions.
GSDisplayServer: An abstract class that provides a framework for a device independent window server.
NSGraphicsContext: An abstract class that provides a framework for a device independent drawing.
Display PostScript System (DPS): A device-independing imaging model.

5.2 Terminology

API - Application Programming Interface
DPS - Display PostScript System
GUI - Graphical User Interface
IDE - Integrated Development Environment
UI - User Interface

References

[1] GNUstep maintainers. "GNUstep: GNU & OpenStep," [Online]. Available: https://www.gnustep.org/information/openstep.html Accessed: 02/12/2025.

[2] "GNUstep History," [Online]. Available: http://gnustep.made-it.com/Guides/History.html Accessed: 02/12/2025.

[3] "GNUstep System Overview," [Online]. Available: http://gnustep.made-it.com/SystemOverview/index.html Accessed: 02/12/2025.

[4] GNUstep maintainers. "libs-gui," [Online]. Available: https://github.com/gnustep/libs-gui/tree/gui-0_32_0 Accessed: 02/13/2025.

[5] GNUstep maintainers. "libs-back," [Online]. Available: https://github.com/gnustep/libs-back/tree/back-0_32_0 Accessed: 02/13/2025.

[6] F. Botto, R. Frith-Macdonald, N. Pero, and A. Robert. "Objective-C language and GNUstep Base Library Programming Manual," [Online]. Available: https://www.ict.griffith.edu.au/teaching/2501ICT/archive/resources/documentation/Developer/Base/ProgrammingManual/manual.pdf Accessed: 02/12/2025.

[7] Apple Inc. "Core Foundation," [Online]. Available: https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFDesignConcepts/CFDesignConcepts.html Accessed: 02/13/2025.