CISC 322/326 Assignment 1
Conceptual Architecture of GNUstep

February 14, 2025

Group #30: Big Hero 6

Terence Jiang 21twxj@queensu.ca
Kevin Chen 21kzc2@queensu.ca
Alexander Zhao 21atxz@queensu.ca
Carter Gillam 21cng3@queensu.ca
Kavin Arasu 21kla5@queensu.ca
Daniel Gao 21dg59@queensu.ca

Table of Contents

1. Abstract

The goal of this report is to analyze and properly define the intricacies of the conceptual architecture supporting the GNUstep program. GNUstep is an open-source implementation of the OpenStep framework which provides a development environment for building applications. In our report, we've defined GNUstep as being constructed using a modular architecture, featuring key components libs-back, libs-base, libs-corebase, libs-gui as well as gorm.

Our analysis of GNUstep's architecture was multifaceted including aspects of functionality, concurrency, control and data flow, and interactions between subsystems. Through this, we determined that GNUstep's modular design follows an object-oriented architecture style. We derive this conclusion by examining the different libraries and components that work together to provide a cohesive development environment. To illustrate, Libs-base serves by providing the non-essential functionalities such as object-management networking, and file I/O. Libs-gui, serves as the foundation for core graphical user interfaces, which implements Cocoa API functions enabling cross-platform app development. There are just two examples of the modular style that GNUstep follows, but each of these libraries functions like an independent object with its own responsibilities and behaviors. They encapsulate specific functionalities and interact with other components to form a cohesive system, exemplifying GNUstep's object-oriented architectural style.

Further technical analysis provides insights into the communication and interactions function of key components within the framework. The report includes sequence diagrams to illustrate both the relationship between various components, as well as the flow of information. Furthermore, this report explores GNUsteps's evolution, real life use-cases and its role in modern software development. By understanding these core concepts of computing, we aim to provide a reference for future developers, and contributors looking to engage with GNUstep.

2. Introduction and Overview

The software industry has witnessed a continuous evolution over the decades, with systems becoming more powerful, cross-platform, and user-friendly. In recent years, the demand for open-source, modular, and flexible development environments has grown significantly, enabling developers to create applications that seamlessly integrate across different systems.

GNUstep was created to address this need by offering an open-source implementation of the OpenStep API, originally developed by NeXT in the 1990s. With its foundation in Objective-C, GNUstep provides a powerful development environment that enables the creation

of cross-platform applications for Unix-like systems, Windows, and macOS. The Project preserves non-functional requirements like modularity, reusability and elegance while ensuring compatibility with legacy NeXTSTEP and macOS Cocoa applications. Unlike proprietary frameworks, GNUstep remains fully open and community-driven, allowing developers worldwide to contribute, extend, and improve the system.

This report provides a comprehensive analysis of GNUstep's conceptual architecture, exploring key aspects such as component interactions, use cases, evolution, control mechanisms, and data flow.

We begin by examining the Architectural style, identifying the fundamental design principles that shape GNUstep's structure. Next, we break down the components and their interaction, detailing how different modules work together. In System evolution we explore how GNUstep has adapted over time, maintaining compatibility while integrating modern features. Following this, we analyze control and data flow, explaining how execution and information are managed across various subsystems. We then discuss concurrency, deciding whether GNUstep leverages parallelism and how it handles multitasking. The implication for division among developers section considers how the architectural design influences the distribution of work among contributors, ensuring efficient collaboration. To illustrate key concepts, we include two sequence diagrams , visually depicting the flow of two major use cases. Finally we summarize our findings, reflecting on the strengths of GNUstep's design and identifying potential areas for future improvement.

3. Components and their Interaction

The GNUstep ecosystem is built on modular, object-oriented architecture comprising various libraries and tools that work together to support application development and execution. Each component plays a specific role, from providing core functionality to handling user interfaces and system interactions. Understanding these components and how they interact is essential for leveraging GNUstep's full potential in building cross-platform Objective-C applications.

Core Libraries

GNUstep Base (libs-base) is the foundation of GNUstep, implementing the foundation framework. It provides fundamental Objective-C classes such as NSString, NSArray, and NSDictionary, along with system-level functionality for file handling, networking, threading, and memory management. **GNUstep CoreBase (libs-corebase)** extends the base library by implementing additional core foundation APIs, ensuring better compatibility with macOS applications. **GNUstep GUI (libs-gui)** provides the AppKit framework, allowing the development of graphical user interfaces. It includes UI components such as windows, buttons, menus, and text fields, making it essential for building interactive applications. The **GNUstep**

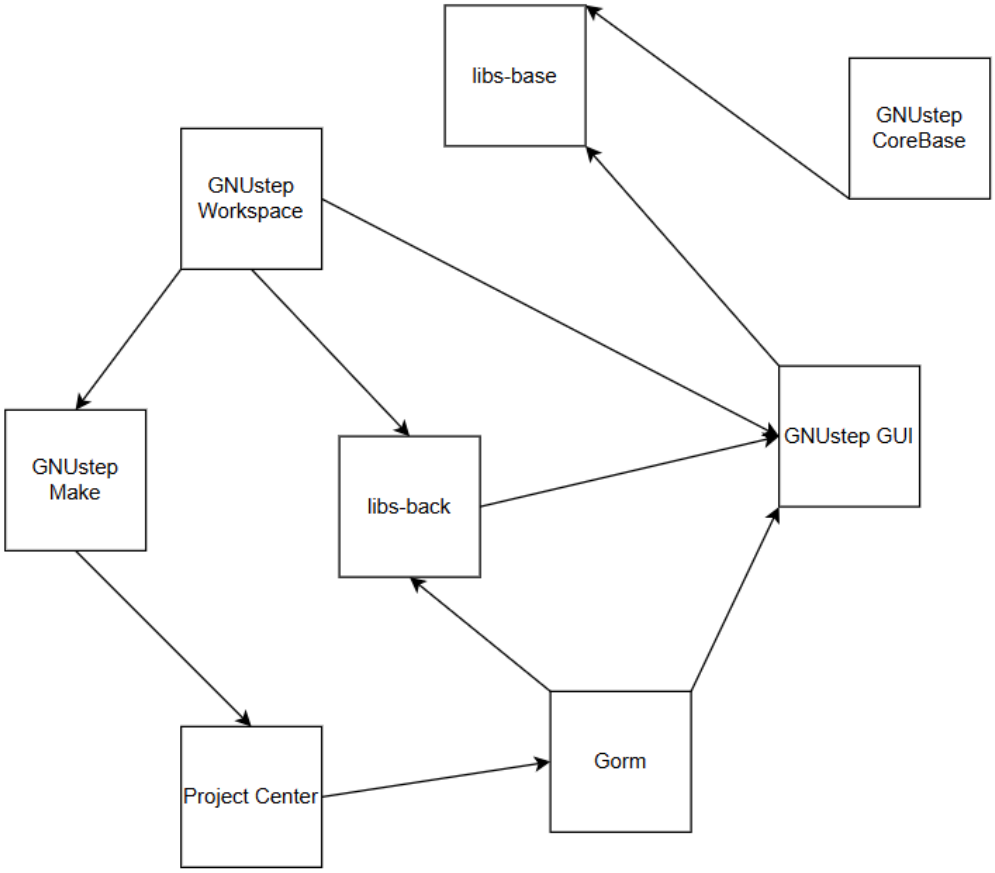
Backend (libs-back) is responsible for rendering graphics and interfacing with different windowing systems. It supports multiple backends, such as X11, Cairo, and Wayland, ensuring flexibility across platform

Development Tools
Gorm (GNUstep Object Relationship Modeller) is a graphical interface builder that allows developers to design user interfaces visually, creating .gorm files that store UI definitions.
ProjectCenter serves as an integrated development environment (IDE) for managing GNUstep projects. It provides tools for editing, compiling, and linking applications.

Additional Components
GNUstep Make is a build system that simplifies the compilation and linking of GNUstep applications, handling dependencies and platform-specific configurations.
GNUstep Workspace is a collection of tools and applications that create a desktop environment based on GNUstep, including file management utilities and system configuration tools.

By combining these components, GNUstep provides a cohesive and flexible environment for developing cross-platform Objective-C applications. Its object-oriented architecture ensures that each component plays a distinct role while interacting seamlessly with others to create a stable and efficient development environment. CoreBase depends on Base for fundamental Objective-C functionality. Libs-gui depends on Base and CoreBase for core classes and additional foundation APIs. The Backend needs libs-gui for rendering graphical elements. Gorm depends on libs-gui and libs-back for IO components and rendering. The Project center depends on Gorm for UI-related project integration. GNUstep Make depends on the ProjectCenter for project compilation and linking. Finally, GNUstep Workspace depends on libs-gui, libs-back, and GNUstep Make for UI elements, graphical rendering, and managing built applications.

The box and line diagram below visually represents these interactions, illustrating how the core libraries, development tools, and additional components connect to form a complete system. Each box represents a component, and the connecting lines indicate the relationships and dependencies between them:



4. System Evolution

The GNUstep framework originated as an open-source reimplementation of NeXTSTEP's OpenStep framework, designed to provide users with essential Objective-C libraries and a basic GUI toolkit. In its initial phase, the primary objective was focused on accurately replicating the APIs of NeXTSTEP while ensuring compatibility with the early iterations of macOS. The evolution of GNUstep introduced several key milestones, including the integration of developer tools such as Gorm and Project Center, the incorporation of contemporary Objective-C features such as Automatic Reference Counting (ARC) and blocks, as well as enhancements to cross-platform compatibility and GUI capabilities. Today, GNUstep remains true to its open-source roots, providing developers with a robust platform for creating advanced GUI applications with its replicated APIs. ("GNUstep Wiki," n.d.)

The objective of GNUstep 1.0 was to replicate the functionality of NeXSTEP's OpenStep environment and offer developers basic tools for building GUI applications in Objective-C. At this stage, GNUstep 1.0 provided a basic GUI toolkit and did not have cross-platform

compatibility, only supporting Unix-based systems. It supported basic windowing, event handling and file management, but lacked advanced features. While developers could build basic apps, GNUstep 1.0 ultimately lacked the capabilities of later versions. ("GNUstep Github," n.d.)

GNUstep 1.2 introduced key refinements that improved the framework's stability and usability. Enhancements to memory management and data structures allowed for more efficient application development, while the AppKit gained better support for event handling, window management, and graphical rendering. While these updates made GUI applications more interactive, GNUstep still lacked advanced graphics rendering, multimedia support, and cross-platform compatibility, limiting its usability beyond Unix systems. ("GNUstep," n.d.)

GNUstep 1.4 was a significant update that focused on cross-platform compatibility, enabling applications to run on Linux, macOS, and Windows. Key features included the inclusion of Gorm, an interface builder replacement for drag-and-drop GUI creation and advanced Appkit Features such as NSTableView and NSTextView for complex GUI layouts. This version was limited by its lack of support for macOS-specific APIs like Cocoa Bindings, Core Graphics, and Core Animation, which also resulted in restricted support for 3D graphics and complex animations. ("GNUstep Developer Tools," n.d.)

GNUstep 1.8 marked a significant improvement in GUI and rendering capabilities, offering enhanced support for Core Graphics-style drawing and Quartz-style graphics, which every other version lacked support for. This allowed for smoother transitions and more complex graphics and animations. However, it still lacked comprehensive support for multi-touch gestures, high-resolution displays, and newer macOS technologies like Metal and Core Animation. This made it challenging to transition applications onto Retina displays and mobile platforms. ("GNUstep Github," n.d.)

GNUstep 1.30 saw compatibility upgrades with macOS Cocoa APIs, and made significant improvements in aligning GNUstep with Cocoa to allow for straightforward porting of macOS applications to the platform. A new and more stable implementation of Cocoa Bindings allowed developers to use declarative bindings between UI components and data sources, which had been a staple of macOS development. Furthermore, performance optimizations were implemented in memory management, event handling, and layout calculations, making the framework more efficient for larger GUI applications. GNUstep 1.30 still had gaps in full macOS API compatibility, mainly for AppKit's support for even greater animation and 3d rendering effects. While the platform was able to replicate many features and accomplish many tasks Cocoa APIs could do, it was not a perfect substitute, especially for apps that relied on newer macOS frameworks. ("GNUstep Github," n.d.)

In the latest update, GNUstep 1.31 committed to expanding cross-platform support and modernizing its features for the purpose of community-driven development and maintenance. Critical updates included better integration with Linux-based operating systems and improved performance on Windows. The GUI system was overhauled and introduced updated controls and layout management, offering more flexibility and responsiveness for developers. GNUstep 1.31 optimized integration with other open-source technologies and provided developers with better support for mobile and desktop software development. ("GNUstep Developer Tools," n.d.)

5. Control and Data Flow

Within GNUStep, there reside three main components, the foundation kit (GNUStep Base), the application kit (GNUStep GUI), and the development tools which include Gorm and Project Center. GNU Base manages the fundamentals of data management, exception handling, and file systems. It is a library that provides users with the fundamental Objective-C classes. GNU GUI allows users to create graphical user interfaces and components, along with handling events. It is a library that provides objects for writing graphical applications.

The application kit manages the main event loop that handles user input and sends those events to corresponding handlers. Following a model-view-controller pattern, it separates into three sections. The Model is the data, the View represents the user interface, and the Controller shows the application's logic. This pattern is used by having the Controller moderate data flow between the data and the user interface or display. When editing a text box for a GUI, the new data imputed would be the Model. How the data is displayed would be handled by the View, and the Controller would be responsible for handling the input and updating the model, ensuring the View shows correctly inputted data.

The development tools offered by GNUstep are application tools that abstract and simplify application creation and management. The two main development tools are Gorm and Project Center. Gorm is responsible for helping users to design their UI allowing easy drag and drop element editing. The Project Center is an IDE that lets users write and compile their source code. Altogether, these components let developers create cross-platform applications with GNUstep.

Refer to Section 9 for specific use cases.

6. Concurrency

Concurrency is a cornerstone of GNUstep’s architecture, enabling its components to perform tasks in parallel, improving responsiveness, scalability, and reliability. This capability is essential for GUI desktop applications and server-side operations, where real-time performance and fault tolerance are crucial. By leveraging powerful tools within its libs-base framework,

GNUstep provides developers with effective solutions for managing concurrent tasks (GNUstep Developers, n.d.-a).

Central to GNUstep's concurrency model is the NSThread class, which allows developers to create and manage independent threads. These threads are used to isolate resource-intensive processes, such as loading large files or performing calculations, ensuring the system's main thread remains responsive to user interactions. For higher-level task management, GNUstep offers NSOperation and NSOperationQueue, which simplify scheduling and dependencies. For example, these tools enable background tasks like network requests to execute without freezing the interface. Additionally, timers and asynchronous callbacks help handle periodic updates, such as progress indicators or real-time notifications, ensuring smooth user experiences even during intensive operations (GNUstep MediaWiki, n.d.-a).

An essential benefit of GNUstep's concurrency model is its focus on fault isolation. Tasks running in separate threads or processes operate independently, meaning that if one task crashes or encounters an error, it does not disrupt the others. For instance, a failed background process will not freeze the main application thread. GNUstep also provides tools like *@synchronized* blocks and locks to prevent race conditions. Race conditions occur when multiple threads access or modify shared resources simultaneously, causing unpredictable results. By enforcing thread-safe access, these synchronization mechanisms maintain system stability even under heavy workloads (GNUstep Developers, n.d.-a).

Despite its advantages, concurrency in GNUstep also presents challenges. Developers must manage shared resources carefully to avoid race conditions and ensure that locking strategies do not result in deadlocks. Deadlocks occur when threads are stuck waiting indefinitely for access. Additionally, concurrency introduces overhead, such as increased memory usage and context switching between threads. GNUstep minimizes these issues by promoting lightweight thread management and providing developers with efficient tools to lessen these risks (GNUstep MediaWiki, n.d.-a).

In conclusion, GNUstep's concurrency framework empowers developers to create responsive and reliable software. Tools like NSThread and NSOperationQueue enable the execution of complex tasks without sacrificing system performance. Whether building GUI desktop applications or server-side processes, GNUstep's concurrency model ensures scalability, fault isolation, and smooth operation for modern software development (GNUstep Developers, n.d.-a; GNUstep Developers, n.d.-b).

7. Implication for Division Among Developers

As we've discussed, GNUstep is a project built with the object-oriented architecture style meaning it is very modularized, where components interact through well-defined interfaces.

Therefore, each module can be developed, maintained and replaced independently without much influence on the structure of the program. In an open-source, decentralized, object oriented based project like GNUstep, responsibilities are often distributed based on module goals and developer expertise. For example, some developers may contribute to lower level Foundation classes like NSString. Other developers may focus on graphical elements defined in other modules of the program. Tool developers may work to enhance Gorm to improve the development experience of GNUstep. It is widely known that object oriented programming encourages reusability and encapsulation. This allows open-source contributors to work on specific classes and methods without prerequisites of deep knowledge of the system, a significant benefit of working in an object oriented open-source project.

In an OOP system resembling GNUstep, the responsibilities of development are split into classes and objects, rather than stiff layers (found in the layered architecture style). Therefore participating developers typically work in specific modules which encapsulate well defined behaviors. However, it is worth noting that developers are held to the standard in which they do not unintentionally break other classes. Separation and modularity helps to prevent this occurrence, but developers are ultimately responsible for this issue.

Being a program that incorporates inheritance, new modules and functionality is typically added by extending existing classes. The core libraries libs-back, libs-base, libs-corebase, libs-gui, and gorm play a significant role in this process as they form the foundation of all other components within the framework. Participating developers must ensure how changes to parent classes will affect subclasses. Furthermore, they are responsible for proper documentation as well as adhering to the Cocoa API behavior rules. Documentation holds significant importance within object oriented applications as it allows other developers to understand certain uses and information about modules. Because GNUstep also incorporates polymorphism, developers must also ensure that different classes are able to interact with each other seamlessly within the GNUstep framework. This is rather difficult, as developers bear the responsibility of maintaining core functionalities while also allowing freedom for flexibility in their development.

A potential unseen responsibility that developers must be aware of are thread safeties and concurrency. Some applications may require multi-threading, meaning that developers must be wary of thread safety, including handling race conditions, resource sharing and other mechanisms.

Ultimately, the division of responsibilities in GNUstep is shaped by its core libraries, libs-base, libs-corebase, libs-gui, libs-back, and gorm, as well as its object oriented architectural style. As discussed, the implications of this framework are, modular development, consistency, and maintainability.

1. Modular Development allows developers to specialize in their areas of expertise, without needing knowledge of the whole system.
2. Consistency using Objective-C as the underlying system allows developers to easily extend and ensure consistency.
3. Each library is encapsulated allowing developers to fix bugs and extend functionality without disrupting other libraries.

8. Use Cases

Use Case 1:

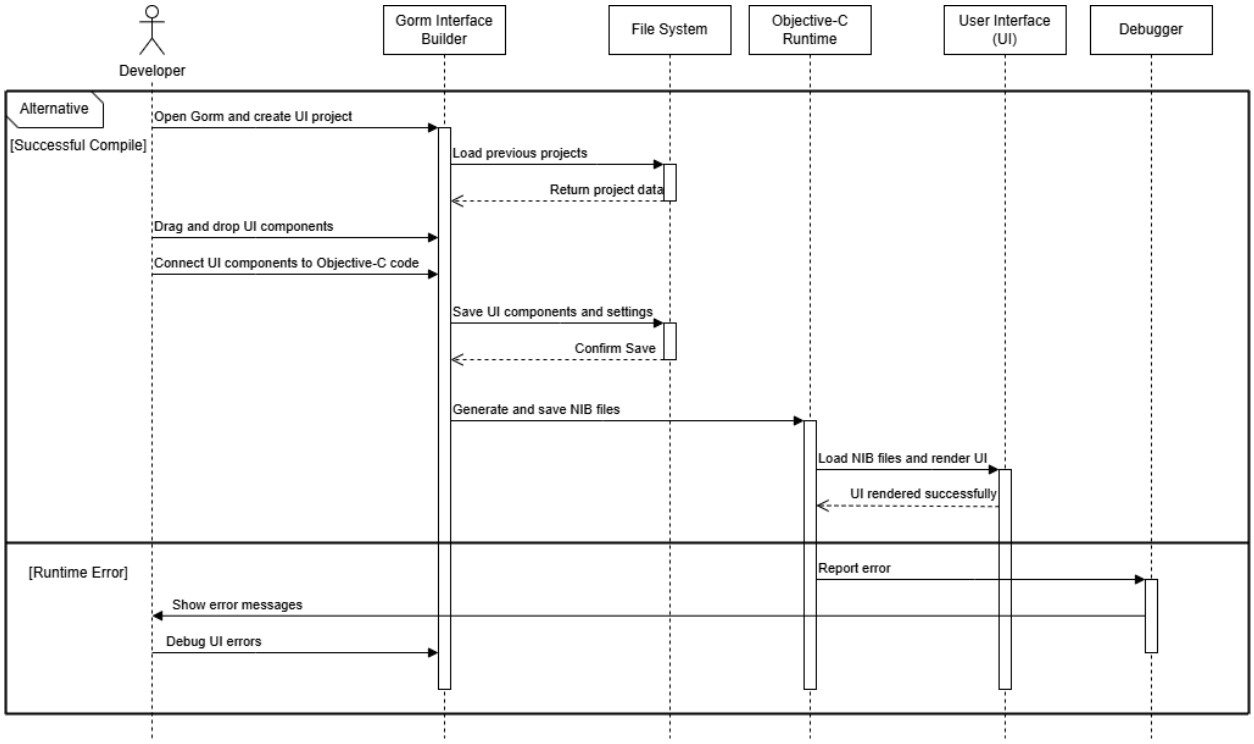


Figure 1. Use Case 1 Creating a GUI application with Gorm

This sequence diagram illustrates a developer utilizing the graphical interface builder Gorm, to design a GUI for an application built with GNUstep. When a developer first opens the interface builder to start working on an application, Gorm makes sure to load any previously saved projects from the file system. This interaction is an exchange of request and responses, where Gorm queries the file system and waits for project data before proceeding. Once loaded, the developer can visually design the UI by dragging and dropping UI components onto the canvas, connecting them to Objective-C code. When saves for the project are successfully made, Gorm generates NIB files which contain the structured UI components, sending them to the Objective-C runtime for processing. The runtime then interprets the files and attempts to render the UI onto the screen. If successful, the application displays the designed interface as expected.

However, if the runtime encounters an issue, an error report is sent to the debugger where the issues are analyzed and sent back to the developer. This is repeated until correct adjustments are made to ensure that the UI is correctly structured to avoid deployment issues.

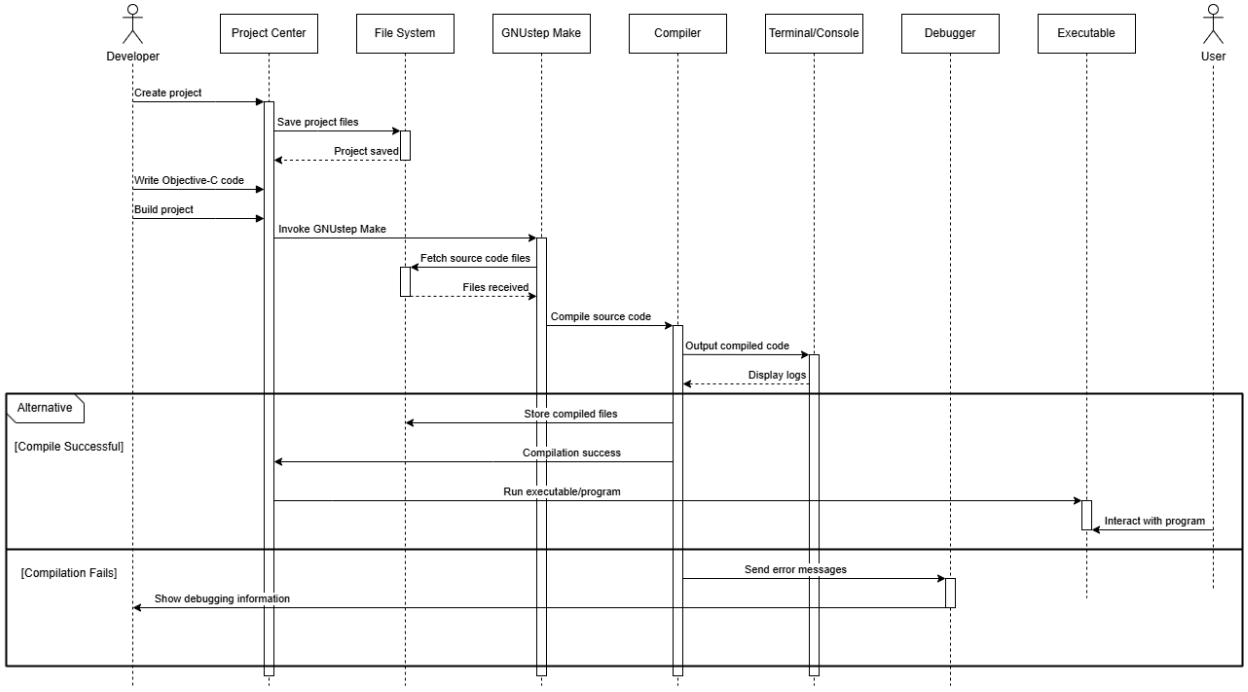


Figure 2. Use Case 2 Compiling and Running a Project in ProjectCenter

Figure 2 demonstrates a developer using ProjectCenter to create, compile, and run a project within the GNUstep environment. The developer begins by creating a new project file, prompting a request to save it within the file system. When this is achieved, the developer can start writing code directly within ProjectCenter and initiating the build process when the code is ready to be compiled. This action triggers GNUstep Make, a build system used for compiling and managing projects within the GNUstep framework. At the core of this process is the compilation sequence, which begins with GNUstep Make retrieving the project's source files from the file system. Because this is a synchronous operation unlike UI rendering in Gorm, Make has to wait for the file system to return the requested files before proceeding. The source code files are then sent to the compiler where its progress, outputs, warnings, and error messages are logged to the terminal/console. If compilation succeeds, the executable is stored in the file system, allowing the developer to run the program through ProjectCenter. If it fails the compiler forwards error logs to the debugger, which analyzes the issues and provides feedback for corrections before restarting the build.

9. Conclusion / Insights

Our deep dive into GNUstep has provided valuable lessons about designing a flexible and sustainable software framework. One key takeaway is how its modular design allows developers to work on different parts of the system without overwhelming complexity. By organizing the platform into distinct libraries and tools, GNUStep demonstrates the benefits of breaking down a

large software project into manageable, interconnected components. Additionally, its evolution from a simple OpenStep reimplementation to a robust, cross platform development environment demonstrates the importance of continuous improvement. With incremental updates in memory management, graphical rendering, and concurrency handling, the framework has steadily enhanced its performance while maintaining its core principles.

Architecture design is a crucial factor in the development and longevity of the GNUstep framework. While GNUstep's architecture remains relatively straightforward, the way its components work together follows closely with the principles of Objective-C and OpenStep. GNUstep is modular and adaptable which allows developers to build cross platform applications while staying compatible with other frameworks. The flexibility and scalability of GNUstep makes it a valuable tool for those interested in creating software using the NeXTSTEP style approach. Its object-oriented architecture style provides a strong foundation for building scalable and maintainable software. By structuring components and libraries as independent objects, GNUstep simplifies development, reduces code duplication, and enhances flexibility. This approach allows developers to easily extend functionality, adapt to new requirements, and collaborate efficiently. Ultimately, GNUstep remains a versatile and reliable platform for software development.

10. Glossary

Gorm - A graphical tool used to design user interfaces for GNUstep applications. Developers can visually arrange UI elements and connect them to their code, simplifying the process of building applications.

Make - A build system for compiling and managing projects in the GNUstep framework. It simplifies the process of building applications by providing predefined rules and configurations tailored for GNUstep development.

ProjectCenter - An open-source IDE that provides tools for managing and developing Objective-C applications within the GNUstep ecosystem.

libs-back - This library handles the connection between GNUstep and the system's graphical backend. It manages rendering and drawing operations, ensuring that GNUstep applications work across different operating systems.

libs-base - The foundation of GNUstep, this library provides essential tools for developers, including support for threading, memory management, collections, and other basic features needed for building applications.

libs-corebase - A low-level library that provides core data types and system functionalities. It supports efficient data handling and forms the backbone for higher-level GNUstep libraries.

libs-gui - The library responsible for creating graphical user interfaces in GNUstep applications. It includes components like windows, menus, and buttons, making it possible to build interactive and visually sound software.

NSThread - A GNUstep class that allows the creation and management of threads, enabling tasks to run concurrently in a program.
NSOperation - A high-level abstraction in GNUstep for encapsulating and managing individual tasks, often used in conjunction with NSOperationQueue.
NSOperationQueue - A class that executes NSOperation tasks in a controlled and concurrent manner, managing dependencies and system resources.

11. References

GNUstep Developers. (n.d.-a). NSThread and Threading. Retrieved from https://www.gnustep.org/developers/documentation.html

GNUstep Developers. (n.d.-b). Developer Tools. Retrieved from https://www.gnustep.org/experience/DeveloperTools.html

GNUstep Developers. (n.d.-c). Gorm. Retrieved from https://www.gnustep.org/experience/Gorm.html

GNUstep Developers. (n.d.-d). ProjectCenter. Retrieved from https://www.gnustep.org/experience/ProjectCenter.html

GNUstep Developers. (n.d.-e). Makefile Package. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Make/Manual/gnustep-make.pdf

GNUstep MediaWiki. (n.d.-a). Main Page. Retrieved from https://mediawiki.gnustep.org/index.php/Main_Page

GNUstep MediaWiki. (n.d.-b). Foundation. Retrieved from https://mediawiki.gnustep.org/index.php/Foundation

GNUstep MediaWiki. (n.d.-b). AppKit. Retrieved from https://mediawiki.gnustep.org/index.php/AppKit

Marcotte, L. (n.d.). Programming under GNUstep—An introduction. Linux Journal. Retrieved from https://dl.acm.org/doi/fullHtml/10.5555/640534.640540