GNUstep: Conceptual Architecture Overview

February 14th, 2025

Group 17: Architecture Addicted Aficionados

Group Members:

Sungmoon Choi	22sc@queensu.ca
Dayna Corman	21drc13@queensu.ca
Michael Cox	21mwc3@queensu.ca
Aydan Macgregor	21ajm32@queensu.ca
Samantha Mak	21yssm@queensu.ca
Kylie Wong	21kycw@queesnu.ca

Abstract

GNUstep is an open-source software framework used to develop desktop applications. It provides tools, libraries, and APIs that enable developers to build cross-platform applications, primarily using Objective-C. GNUstep is based on the same foundation as Apple's Cocoa framework and utilizes many of the same functionalities as OpenStep, Cocoa's predecessor. So GNUstep is optimal as an open-source alternative to Apple's environment. This report will look at this open-source framework and analyze its conceptual architecture. We will be interested in five main libraries, apps-gorm, libs-back, libs-base, libs-corebase, and libs-gui. These main components will provide a unique insight into GNUstep and its architecture. Its architecture can be viewed from 3 main styles, GNUstep is organized with a layered architectural style, object-oriented components for the GUI, and is connected with a publisher/subscriber event system. The layered style allows developers to focus on their applications and let lower-level information be handled by the operating system. This can be seen best in the data flow diagrams, which showcase the control flow of information. GNUstep is optimized for response time as it utilizes many concurrency techniques, such as threads and event-based concurrency for GUI input, user input, and file management. Its overall architecture can be exemplified in the use case diagrams near the end of the report which showcase GNUstep's actions for various real-world use cases of GNUstep.

1. Introduction

As developers sought cross-platform development environments for Objective-C applications, GNUstep emerged as a powerful open-source framework which is capable of creating complex, modern applications. GNUstep was initially developed under the philosophy of open-source development and with the ideals of cross-development in mind. It was meant to copy the OpenStep API, an object-oriented framework for building applications in Objective-C. GNUstep has become the best open-source, Objective-C development application tool, mainly for Windows systems.

This report aims to study and analyze the conceptual architecture of GNUstep and its major subsystems. The report includes an overview of how five main libraries (apps-gorm, libs-back, libs-base, libs-corebase, libs-gui) interact with each other and other frameworks within GNUstep that help create the core functionalities of the application.

We will examine the five main libraries and their architecture, sharing what each section is responsible for and their interactions with each other to create a unified experience in GNUstep. An analysis of the data flow between the components will also provide insight into how the libraries communicate with each other and share information. Information sharing will also be discussed in the context of concurrency, through which GNUstep makes great use of concurrent operations through user input, GUI, and File processing. With this information provided, we will also discuss the implications of GNUsteps conceptual architecture for developers looking to use

such a system for their use. Finally, we will show scenarios such as creating new windows and saving an application using use case diagrams, providing a visual example of GNUsteps conceptual architecture.

We have decided on three main architectural styles that best represent GNUstep. GNUstep organizes its major libraries into layers, each with its purpose and function that will pass information further down the chain. For example, the main program (GORM) will pass data and actions to AppKit (libs-base, libs-gui), which itself passes information to the backend (libs-back). The GUI components are modelled by an object-oriented architecture style that is heavily inspired by OpenStep. GNUstep uses many of the same objects for functions such as event creation, buttons, and I/O. For most of the data flow and the event system used to connect it all, a publisher/subscriber architecture is used.

1.1. Derivation Process

The architecture described in the report is derived through individual research, group discussion, and summarizing. We began by researching GNUstep's vision and the fundamentals of the five major components. With this foundation, we examined GNUstep's official documentation, source code, and historical development to understand its structure better. Additionally, we explored community-driven resources such as wiki pages and developer forums to gain insights from contributors and users.

Utilizing the research and our knowledge, we identified the relationship between the major components and how they interact. We also considered alternative architectural styles. While a repository style could be used for centralized data management, it is not ideal for GUI frameworks that require event-driven interactions. The interpreter style offers dynamic script execution and flexibility; it is less suited for GNUstep, which is designed to implement OpenStep APIs through a more structured approach. Ultimately, object-oriented and layered styles are the most effective for maintaining cross-platform compatibility, independent development, and modularity, allowing components to evolve without disrupting the system.

2. Architecture Overviews

GNUstep is organized with a layered architectural style, with object-oriented components for GUI, connected with a publisher/subscriber event system. This layered system allows application developers not to worry about which operating system/windowing system a user is using when they develop the application, as the precise implementation of the lower layers can be changed to handle the user's operating system without making any difference to how the application's code interacts with AppKit.

2.1 Components

GNUstep's architecture can be split into five major components across three layers. These are:

Application: This is the program the user creates using a GNUstep-powered user interface. Gorm is an example of one such application. These applications are allowed to be platform-independent by the supporting architecture below. The application layer is the outermost layer of the hierarchy.

AppKit: The AppKit is a collection of user interface elements, event-related classes, and classes designed to help develop tools, wrappers, and interfaces with web servers. It is the layer directly below the application layer and is what the programmer of the application mainly interacts with to ensure the platform independence of their application. This layer contains two major components:

- **Libs-gui:** This part handles all GUI functionality, and contains classes and methods useful for all kinds of elements like buttons, select menus, and the like.
- **Libs-base:** This is the part of the AppKit that handles all other functionalities that are useful to a developer, like interfacing with web servers to utilities for tools.

Back-end: The layer below the AppKit layer is the backend. This is where GNU-step starts to get very particular on the specific implementation of the operating system and windowing system that the end-user is using. The functionality is contained in backend modules, like in the case of our defined scope, libs-back, specifically for X-systems or Windows. This layer interfaces directly with the operating system and the windowing system of the machine it is running on.

Shared Library: Aside from the three main layers, a side layer contains code used by the above layers. This is the function of libs-corebase, which provides abstractions to many common data types in programming and helps to add additional features to Objective-C, which GNUstep is coded in.

2.2 Component Interactions

Application: The application layer only really interacts with two main parts of GNUstep, the AppKit, and the shared library. This is so the application can be programmed in a platform-independent way using bindings from the AppKit so the developer does not have to worry about the differences between systems.

AppKit: The AppKit interacts with the shared library, and the back-end, and is interacted with by the application. The AppKit uses shared bindings to interact with the back-end, so the back-end can be swapped out with a back-end with the same bindings but handling a different operating system, without creating any major differences for the AppKit and application.

Back-end: The back-end interacts directly with the windowing system and operating system and therefore must be coded specifically for the windowing system that the machine it is running on is using. This means the back end has several different versions that can be swapped out

depending on the OS/Windowing system. It interacts with the shared library as well for some functionality. It is dependent on the AppKit, and by association, the application, although the application barely interacts with the backend.

3. External Interfaces

GNUstep uses Objective-C for its interfaces. This enables GNUstep to provide vast classes and methods for developing applications, primarily inspired by OpenSteps classes and methods. Although Objective-C is not the only programming language available for GNUstep, it also supports Java and Ruby. The JIGS (Java Interface to GNUstep) is a library that allows the use of Java for development with GNUstep. Java programmers can use all of the GNUsteps libraries from Java. This can be done through the fact that Objective-C and Java are very similar languages so it makes it possible to use GNUstep Objective-C classes with Java. RIGS is the other language interface which allows Ruby developers to use GNUstep similarly to how Java programmers use GNUstep. GNUstep-Guile also provides functionalities for the scripting language GUILE to be used for things such as task automation and batch processing. Furthermore, as previously discussed in 2.1 Components and 2.2 Component interactions, libs-back interfaces with the operating system on the user's machine. Which helps with event handling from user input, interfacing with external graphics libraries, rendering, a much more. For database management, GNUstep offers GDL2 (GNUstep Database Library 2) which provides a set of libraries to interface with relational database management systems. It can map Objective-C object rows within GNUstep to rows within the relational database management system it is interfacing with.

4. Control & Data Flow

This section describes the control & data flow among the five key components for a GNUstep application, i.e. libs-base, libs-gui, libs-back, apps-gorm (Gorm) & a user-developed GUI application namely apps-User.

4.1 Control Flow

The control flow among these five components is shown in the layered diagram and use case diagram below.

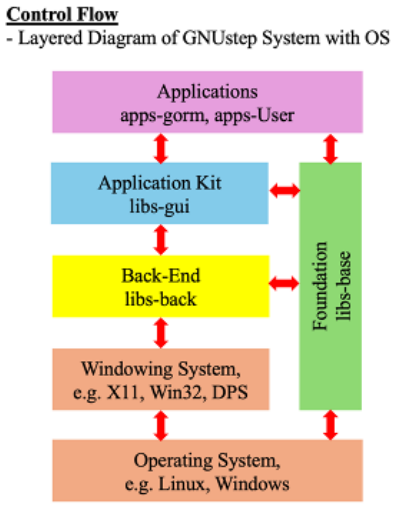


Figure 4.1.1 Layered Diagram

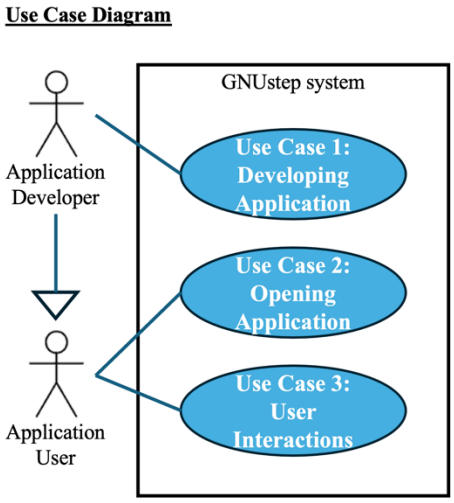


Figure 4.1.2 Use Case Diagram

This is a layer diagram with bi-directional control flows. As in its top-down control flow, layer view shows GNUstep applications at the top will use libs-gui & libs-base services only instead of underlying libs-back & OS. The Back-End libs-back hides details of underlying OS & Windows systems to provide platform-independent services to libs-gui for GNU elements. Similar libs-base does for all other GNUstep components for non-GNU elements. The opposite direction, bottom-up, in the bi-directional control flow, indicates that methods in upper layers can be invoked too by a lower layer, such as leveraging the Notifications feature in GNUstep as indicated in Use Case 3 User Interactions in this section.

There are two types of actors in the Use Case Diagram, i.e. Application User & Application Developer. Application User is a more generic user actor using GNUstep applications like apps-User. Application Developer is a specific application user using apps-gorm to develop other GNUstep applications, which is described in Use Case 1 below.

Use Case 1: Developing Application – developing the GNUstep application (apps-User)

Application developers can use GNUstep tool Gorm, which is also a GNUstep application apps-gorm, to develop a GUI application, said apps-User. Using apps-gorm, an application developer is also an application user of the GNUstep system, as shown in the use case diagram above.

The developer uses apps-gorm to design and configure GUI components from libs-gui for apps-User. The developer can drag and drop GUI elements from palettes, manipulate their properties, define layouts, and interactively test the apps-User's design within apps-gorm. The developer provides custom application logic of apps-User in a user-defined class's method, and defines the method as the event handler for libs-gui's events like mouse clicks and text edits.

Use Case 2: Opening Application - opening the GNUstep application (apps-User) to run

There are several ways to use a compiled GNUstep application. For example, the user can use the command "*openapp apps-User*" in the OS command prompt to open the GNUstep apps-User application. Then GNUstep will initialize its system by loading the libs-base & libs-gui libraries, getting system preferences & user defaults, determining libs-back library from the GSBackend preferences and loading it, loading the application & resource files like those created by apps-gorm (Gorm) during the application development phase, drawing graphical elements on the screen per the application, and finally waiting for user inputs.

Use Case 3: User Interactions - handling user inputs by the GNUstep application (apps-User)

After the application is started, the application user interacts with the application with user inputs like a mouse click, text edit, drag & drop on GUI controls in libs-gui, e.g. button, text field, tableview, image, window, etc. libs-gui defines an NSEvent type for representing mouse and keyboard actions while libs-base provides a notification framework with NSNotifications & NSNotificationCenter classes.

libs-back captures user inputs from the underlying operating system and passes them to GUI controls. GUI controls in libs-gui post notifications with libs-base's support, e.g. editing text in text control, user selection changes or the reordering of columns or rows in tableview. Objects in apps-User can register notifications in their interests and perform actions per user defined application logic in apps-User.

4.2 Data Flow

Below are data flow structure/view of all three use cases.

Use Case 1: Developing Application – developing the GNUstep application (apps-User)

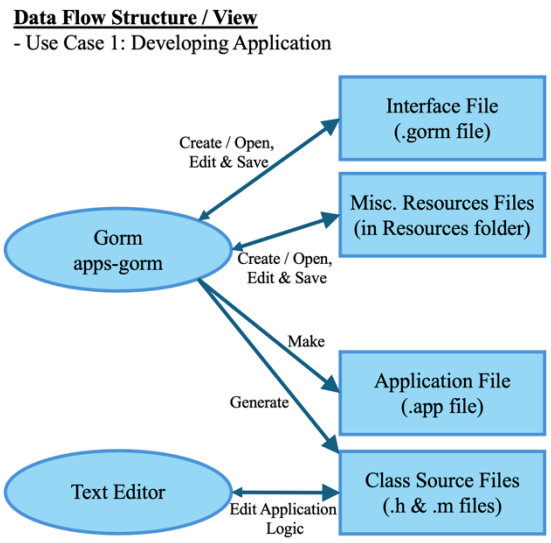


Figure 4.2.1 Data Flow Structure/View of Use Case 1: Developing Application

Resource Files are any sort of resources that a GNUstep application will need to operate, including interface files and any icons, images, data, etc. that the application uses. They are stored in the 'Resources' directory in the application's app wrapper. The most common resource files needed are the interface file and the application's property list (Info-gnustep.plist).

The application developer creates/opens, edits & saves the interface file, .gorm file, with the apps-gorm (Gorm). After the application developer creates a custom class in apps-gorm, apps-gorm can generate the initial class source files, and the developer can edit application logic in the .m file by using any text editor. After the developer completes the development, he can make the application executable (.app file) with apps-gorm & make utilities.

Use Case 2: Opening Application - opening the GNUstep application (apps-User) to run

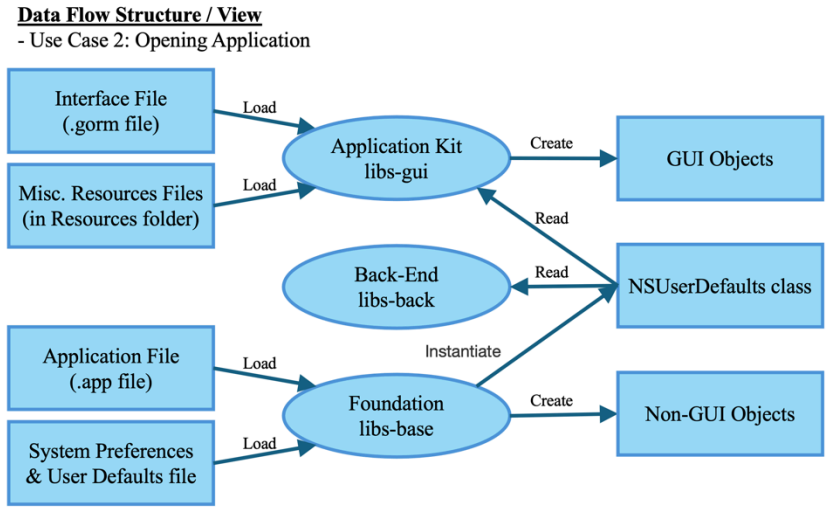


Figure 4.2.2 Data Flow Structure/View of Use Case 2: Opening Application

libs-base provides the NSUserDefaults class as an interface to the defaults system, which allows application access to global and/or application-specific defaults set by the user. These system preferences & user defaults are stored in a file system and loaded by libs-base when an application is opened.

For the application execution, GNUstep will create GUI & non-GUI objects in the memory with libs-gui & libs-base per apps-User's application definition.

Use Case 3: User Interactions - handling user inputs by the GNUstep application (apps-User)

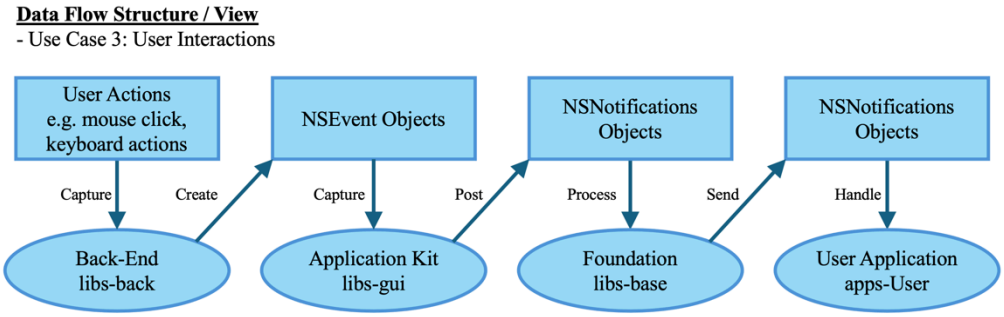


Figure 4.2.3 Data Flow Structure / View of Use Case 3: User Interactions

libs-back captures user actions, e.g. mouse click, keyboard actions, from the underlying OS, and creates NSEvent objects to be captured by libs-gui. Then libs-gui creates a NSNotifications object which is processed by the libs-base's notification center. The notification center will send out the NSNotifications object to all objects registered on the notification, which allows the apps-User to handle the notification by the user-defined application logic.

5. Concurrency

GNUstep is a framework suite for GUI applications based on OpenStep API. The framework uses single-threaded event loops for GUI operations and Apple's Grand Central Dispatch (GCD) with NSThread and NSOperationQueue to implement concurrency. These concurrency techniques in GNUstep improve application responsiveness by allowing tasks to be run in the background without blocking the main thread. This section will go into detail on the key areas of concurrency and control mechanisms found within GNUstep for managing its concurrency.

5.1 Key Areas of Concurrency

To handle various areas of concurrency GNUstep uses libraries like libgnustep-base lib-thread that are a part of the GNUstep-base. These libraries help to run multiple tasks in the background and ensure simultaneous execution and smooth response times. Some examples where these libraries are required include:

- **GUI Processing:**
  As a framework for GUI applications, Users must be able to interact with their applications inside their desktop environment without interruption. As such response time becomes essential to ensure a smooth user experience. By offloading tasks such as file handling during GUI processing, we can allow the main thread to focus on the user input or any GUI updates to ensure ample response time

- **Input Handling:**
  Input from the keyboard and mouse are used to manipulate the GNUstep desktop environment and must be run with appropriate response time. Input handling itself is typically executed on the main thread where events can be dispatched and processed.

However, tasks like file I/O, data processing, or network communication are offloaded as background threads during the user interaction to prevent blocking the main thread.

- **File Manipulation:**
  While GNUstep does not have its model for dealing with concurrency through file management, it handles files synchronously and asynchronously through NSFileManager and NSFileHandle. To avoid blocking the main thread GNUstep uses NSThread to read and write to files in the background asynchronously while using NSData to read files synchronously.

5.2 Concurrency Control Mechanisms

GNUstep utilizes the GNUstep Base library to manage multiple concurrent processes efficiently, ensuring synchronization across its functions. Some examples of where these libraries are applied are as follows:

Thread-Based Concurrency:
In GNUstep, when the user interacts with the framework environment, multiple threads must be able to run concurrently. For example, when opening multiple GUI applications, each task must run without blocking the main thread. This is done through the NSThread and Apple's Grand Central Dispatch, where background tasks can execute asynchronously, allowing the main thread to focus on user input or GUI updates. This form of concurrency ensures that long-running tasks such as data processing or file handling never interrupt the main thread.

Event-Based Concurrency:
Within GNUstep's framework, user input and networking are critical asynchronous tasks that use event-based concurrency to ensure a low response time. These factors rely on NSRunLoop to prevent blocking the main thread, allowing these operations to be performed asynchronously while maintaining responsiveness. This approach ensures that the application can handle multiple tasks simultaneously without delaying or interrupting the user interface.

GNUstep utilizes many forms of concurrency across GUI processing, input handling, and file manipulation to ensure these tasks do not block the main thread. The GNUstep framework uses the NS library to provide both thread and event-based concurrency to simulate these tasks simultaneously and give the effect of parallelism.

6. Implications for Developers

As GNUstep began as a way to port HippoDraw from NeXTSTEP, it borrows many interface and design choices from NeXTSTEP, which any developers familiar with Objective-C and NeXTSTEP's interface can adapt to GNUstep easily. Despite this, learning to use GNUstep can still be a learning curve due to the niche nature of Objective-C and GNUstep's framework can

prove to be a challenge to developers starting out. To combat this issue, GNUstep provides a graphical application to manage projects easily; ProjectCenter and Gorm.

ProjectCenter is an integrated development environment (IDE) for managing project files and workflows. This allows the developer to create applications, tools, libraries and bundles with the ability to easily add, remove, edit and search files which further aids developers working together on the same project to focus on their work with its well-sorted and categorized overview of the files.

Gorm is a graphical interface builder that designs and tests GUIs for GNUstep. Using its simple and intuitive drag-and-drop actions, UI developers/designers can focus on improving UI development and its efficiency.

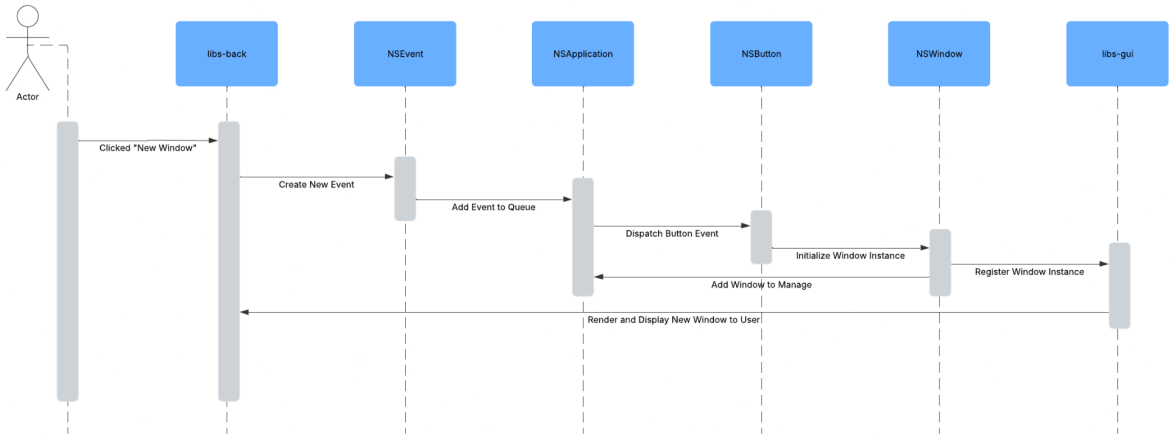
Using these GNUstep developer tools, participating developers can divide and assign their roles and goals as a team effectively with two development tools that are specialized in their needs. While in comparison with many modern applications, they lack in special features, but these tools stick to the basic and essential needs that get the job done.

The developer documentation for GNUstep lists out GNUstep coding standards to ensure consistency and uniformity across multiple works of many developers. This extensive list of documentation includes, but not limited to using basic classes, distributed objects, intro to GNUstep and Gorm, optimizing objective-C code, GNUstep coding standards, etc. Documentation is crucial as GNUstep doesn’t contain as many debugging tools as compared to other mainstream frameworks requiring the developers to fully understand and utilize the GNU debugger effectively.

Despite the challenges that participating developers may face, GNUstep provides plenty of information and details available for them, with different tools specializing in their needs to allocate their contribution to the project effectively.

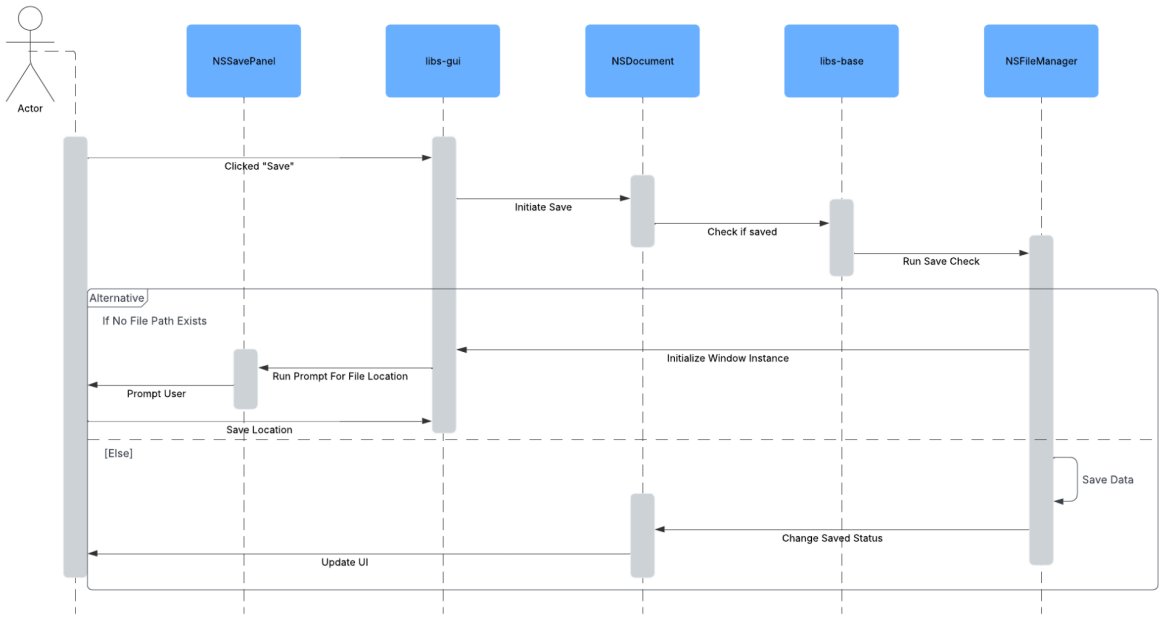
7. Use Case Diagrams

7.1 Use Case: The user creates a new window by selecting “New Window”



This is a common use case for GNUstep users as the action creates a new window for the application being developed. Once the user clicks New Window, libs-back will take the input and create a new NSEvent object to hold the action. This object will then be added to the queue in NSApplication to await a response. Once it's dequeued it is sent to the object that took in the user event, which is NSButton. NSButton then dispatches the event to NSWindow to create the new window, and libs-gui will index the new instance of a window. The window is then displayed to the user.

7.2 Use Case: The user saves the application they are working on



Another common use case for any user is saving the application they made. Once the user saves their work, it is sent to libs-gui, which initiates the saving function by sending the event to NSDocument, libs-base checks for previous saves and NSFileManager will either save the window instance automatically or prompt the user for a file location then save the application.

Legend for both use case diagrams:

8. The Evolution of GNUstep

GNUstep is an open-source implementation of the OpenStep specification, and its development has been influenced by the need for a cross-platform object-oriented framework compatible with Apple's Cocoa. The evolution of GNUstep has been driven by the need for flexibility, allowing it to adapt to emerging technologies, user needs, and various platforms over time.

8.1 GNUstep Vision (Early 1990s)

Inspired by OpenStep, an object-oriented framework developed by NeXT in 1989, the GNU Project aimed to create a free and cross-platform environment that could provide the same features and functionalities for developers. This vision led to the start of GNUstep, which quickly became a pivotal project for bringing the NeXT ecosystem to the open-source world.

8.2 Early Development (Mid 1990s)

GNUstep development started dually. Andrew McMullin and Paul Kunz made significant contributions by independently working on two essential components of the framework. McMullin started writing the foundation framework, which provides essential functionalities that support system-level operations. Meanwhile, Kunz had been working on objcX, an AppKit implementation responsible for GUI elements. These parallel efforts allowed GNUstep to be developed efficiently, as the core system and graphical interface were built simultaneously.

8.3 Consolidation and Expansion (Late 1990s)

By the late 1990s, GNUstep had progressed significantly, with the Foundation and AppKit frameworks becoming more stable and functional. The independent efforts of developers evolved into a more structured and unified project. As GNUstep gained recognition as an open-source alternative to OpenStep, it attracted more contributions, which became a challenge in achieving

full compatibility with OpenStep due to differences in implementation and evolving system requirements.

8.4 Modernization and Adaption (2000s onward)

In the 2000s, GNUstep faced new challenges as Apple transitioned OpenStep into Cocoa, introducing proprietary enhancements where changes were not made available under open-source licenses. In response to these changes, GNUstep developers focused on improving their own implementation and modernizing the framework while retaining its cross-platform capabilities. On April 9, 2001, GNUstep released the first official version 1.0.0, marking a significant milestone in the project's development. The project adapted to evolving system requirements, enhancing Objective-C support and refining the graphical user interface. Ongoing community contributions helped sustain and expand the project, ensuring its continued relevance as an open-source alternative to OpenStep.

9. Conclusions & Lessons Learned

We have analyzed the conceptual architecture of GNUstep, focusing on its structure and key subsystems, data flow, and concurrency. GNUstep follows a layered architecture comprising five main object-oriented components across three layers: the Application layer as the outermost layer, followed by the AppKit and the back-end. Layered and object-oriented architecture style provide flexibility, scalability, and efficiency in event handling within GNUstep. We explored how components like GUI rendering, event handling, and data management interact to ensure seamless functionality. Concurrency plays a role in handling user interaction and system events, ensuring simultaneous execution. In addition, we examine the structured control and data flow between layers, maintaining efficient state management and communication across the architecture. Overall, GNUstep provides an open-source, cross-platform software framework for Objective- C applications, and its structured architecture ensures efficient resource management and a consistent user experience across different platforms.

One of the takeaways from this project was the importance of flexibility in the system to adapt to changes. GNUstep demonstrated the necessity of flexibility in a system as it has continuously adapted to changing requirements to maintain compatibility with modern platforms. This project gave us valuable insight into large-scale software architecture and the development process. We gained experience analyzing and breaking down open-source software, collaborating and communicating findings effectively within our group, and piecing together our research to understand the system comprehensively.

References

- GNUstep's Wiki Main Page: https://mediawiki.gnustep.org/index.php/Main_Page
- GNUstep Base Programming Manual http://andrewd.ces.clemson.edu/courses/cpsc102/notes/GNUstep-manual.pdf
- GNUstep GUI Programming Manual https://www.gnustep.org/resources/documentation/Developer/Gui/ProgrammingManual/AppKit.pdf
- GNUstep Base API Reference https://www.gnustep.org/resources/documentation/Developer/Base/Reference/index.html
- GNUstep GUI API Reference https://www.gnustep.org/resources/documentation/Developer/Gui/Reference/index.html
- Guide to GORM application https://www.gnustep.org/resources/documentation/Gorm.pdf
- Building Apps with GORM https://gnustep.github.io/Guides/App/index.html
- GNUstep development tools: a basic tutorial https://www.gnustep.org/experience/PierresDevTutorial/index.html
- User Defaults Summary for GNUstep Libraries https://www.gnustep.org/resources/documentation/User/Gui/DefaultsSummary.html
- User Defaults Summary for GNUstep Backend https://www.gnustep.org/resources/documentation/Developer/Back/General/DefaultsSummary.html
- GNUstep Configuration Guide http://gnustep.made-it.com/Configuration/index.html
- GNUstep System Overview https://gnustep.made-it.com/SystemOverview/index.html#AEN67
- GNUstep Develop Tools https://www.gnustep.org/experience/DeveloperTools.html