CISC 322/326

Assignment 2: GNUstep Concrete Architecture

February 14, 2025

GNU16 Design Team
Group 16

Group Leader: Luc Robitaille - 21lar18@queensu.ca
Presenter 1: Enqi Liang - 21el32@queensu.ca
Presenter 2: Rodrigo Del Aguila Velarde - 20rdav@queensu.ca
Vlad Kukov - 21vk29@queensu.ca
Lucas Ferber - 19ljf1@queensu.ca
Robert He - 20rlth@queensu.ca

Table of Contents

1. Abstract
2. Introduction
3. Derivation process
4. Architecture
    - 4.1. Review of Conceptual Architecture
    - 4.2. Top-Level Concrete Architecture
        - 4.2.1. Architectural Style
        - 4.2.2. SubSystems
    - 4.3. Sub-System: GNUstep-base
        - 4.3.1. Conceptual Architecture: GNUstep-base
        - 4.3.2. Concrete Architecture: GNUstep-base
        - 4.3.3. GNUstep-base SubSystems
5. Discrepancies Between Concrete & Conceptual Architecture
    - 5.1. High-level architecture changes
    - 5.2. Subsystem architecture changes
6. Use Cases
7. Lessons Learned
8. Data Dictionary
9. Conclusion
10. References

1. Abstract

This report identifies and showcases the concrete architecture of the top-level architecture of GNUstep. As well as the conceptual and concrete architecture of the libs-base subsystem, with an in-depth look into its own subsystems. The most important and noteworthy subsystems within the libs-base library are documentation, examples, config, headers, source, and tools. All of the other subsystems within libs-base will not be discussed in this report. This report starts with a look into our derivation process for this entire report, describing the work decisions made by the group. Next, a quick review of the conceptual top-level architecture is given as a refresher before showcasing the concrete top-level architecture. During this section, the architectural style of the concrete top-level architecture is discussed. Then each subsystem on the top-level is described for a clear understanding of the concrete architecture. Then, an in-depth look into the libs-base subsystem is depicted in this report. Both the conceptual and concrete architecture are identified and analysed. The subsystems of the libs-base library are then described for a clear understanding of what their purpose is inside libs-base. Next the report conducts a reflexion analysis on the high-level conceptual and concrete architecture. Another reflexion analysis is done during this section on the subsystem conceptual and concrete architecture. Then this report showcases two non-trivial use cases, with respective sequence diagrams to visually demonstrate how the subsystems within GNUstep interact. Then this report discusses lessons learned by everyone in the group when working on assignment 2. Finally, this report Treport lays out some of the important new terms used in a data dictionary, and showcases our main conclusions about the concrete architecture of GNUstep.

2. Introduction

GNUstep is an open source object-oriented development environment originally derived from the OPENstep API and utilizing the NEXTstep framework that provides the user with a large range of utilities and libraries for building large, cross-platform, applications and tools. Development began in 1995, Its original conceptual architecture of 3 core subsystems, GNUstep-base, GNUstep-back, and GNUstep-gui have expanded to many more subsystems and applications. Examples such as GORM, and GNUstep-corebase. In order to reach its goals, as well as needing to keep up to modern standards. The concrete architecture has developed many discrepancies from its original conception in the last 30 years of development. Its current implementation has implemented all of Opensteps useful functionally, as well as most new classes implemented by Cocoa. Its development team is continually keeping the program up to date and seeks to do so for the foreseeable future.

3. Derivation Process

For the derivation process of the top-level concrete architecture, this section was assigned to one of the group members. As our main method of completing the report was splitting up different sections for different group members to complete. Even though specific sections were assigned, members are not restricted to only work on their sections. This resulted in multiple group members reading and checking over the determined top-level concrete architecture. The Understand tool was used on the reduced version of GNUstep assigned. The results created by the tool were analysed by group members in order to discover and agree on the concrete architecture of GNUstep. After the concrete architecture was found, an updated dependency graph was created that shows the original ideas from the conceptual architecture and the real dependencies from the concrete architecture.

For the selection of the subsystem to investigate, every group member discussed which subsystem we thought would be best. During this discussion, the group realized that we wanted to look into a subsystem that provided important foundational aspects for GNUstep. We were split between the base and back libraries as they both seemed to be important and correlate to more foundational features of GNUstep. After this discussion, we thought it would be best for the group member that was primarily working on the concrete architecture section of the report to make the final decision on which subsystem to analyse. This ended up being the base subsystem. The concrete architecture of the base subsystem was found using the Understand tool. Again, the results created by the tool were analysed by group members in order to discover and agree on the concrete architecture of the subsystem in question. The conceptual architecture of the base subsystem was found through researching on the internet, and documentation of the base subsystem. This conceptual architecture found for GNUstep-base is based on the conceptual architecture of the OpenStep Foundation Kit, as there is very little information on the internet about specifically GNUstep's base library. Due to the fact that the base library is based on OpenStep's Foundation Kit, these subsystem's conceptual architectures would be very similar, if not the same.

When comparing the conceptual architecture and the concrete architecture, reflexion analysis was used in order to provide an in-depth comparison between the found architectures.

4. Architecture

4.1 Review of Conceptual Architecture

As discussed in the previous report the conceptual architecture uses a hybrid of object oriented and layered style. (Figure 1)

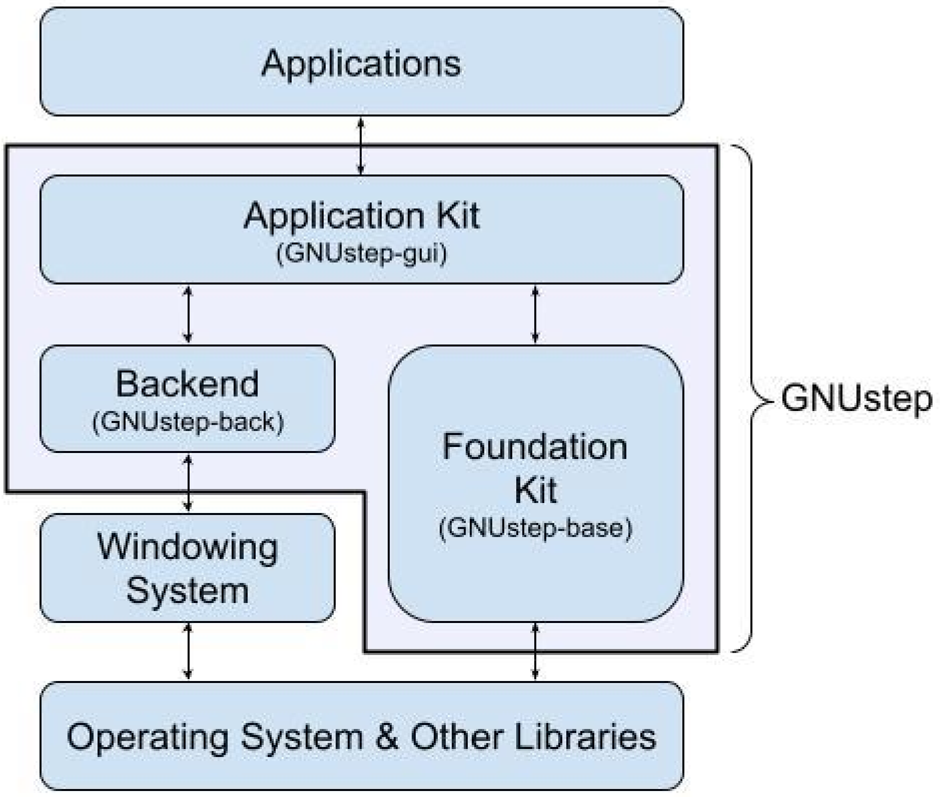


Figure 1: Diagram of the Data & Control Flows in GNUstep

4.2 Top-Level Concrete Architecture

4.2.1 Architectural Style

As with the conceptual architecture, the concrete architecture was also constructed using a hybrid of object oriented and layered architectural styles. The object oriented style is ideal for the modular interactions between GNUstep's various subsystems. This style is also ideal for the cross-platform nature of GNUstep, being required to operate on many different operating systems.(Figure 2)

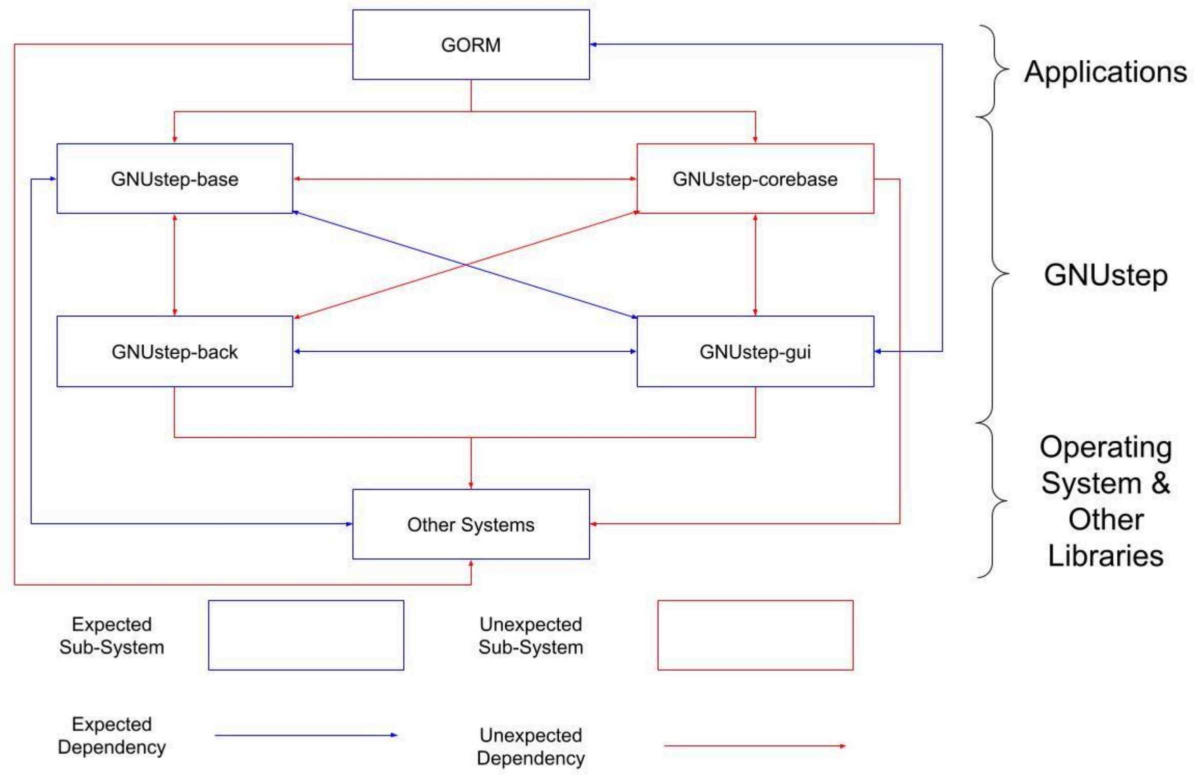


Figure 2: Object Oriented Concrete Architecture of GNUstep

4.2.2 SubSystems

GNUstep-base
This subsystem is responsible for non-graphical objective-C classes such as strings, byte streams, type coders, notifications, network ports, event loops and random number generators. GNUstep-base interacts with and relies on the Objective-C library, which is not part of GNUstep. It also provides reusable code to the other GNUstep subsystems for functions, such as data representation and storage.

GNUstep-back
This subsystem is responsible for rendering and handling events. GNUstep-back takes raw graphical data from GNUstep-gui and other subsystems and renders it allowing the gui to visually display for the user. This sub-system also relies on classes from GNUstep-core and GNUstep-corebase for file management and runtime operations.

GNUstep-gui
This subsystem graphical Objective-C classes such as fonts, events, menus, windows, buttons and sliders. This subsystem is required by all applications in order

to display the user interface. It also requires the GNUstep-back subsystem to render the graphical data which is then displayed by GNUstep-gui. It also requires GNUstep-base and GNUstep-corebase for basic non-graphical classes and runtime operations.

GNUstep-corebase
This subsystem is a cross-platform, general-purpose utility library that implements Apple's core foundation framework API. It provides functionality for GNUstep to operate on Unix-like operating systems such as Linux and Windows. This subsystem provides abstractions to common data types such as strings, numbers, arrays and dictionaries and also provides utilities for lists, run loops and more. GNUstep-corebase is required by all other subsystems for basic data abstractions and utilities.

GORM
This subsystem is an application created to design graphical user interfaces using the GNUstep system. It relies heavily on GNUstep-gui as this is responsible for providing graphical classes and rendering the gui elements that can then be displayed and edited using GORM. It also relies on GNUstep-base and corebase as these provide basic compatibility to the operating system and functionality of non-graphical classes.

4.3 Sub-System: GNUstep-base

4.3.1 Conceptual Architecture: GNUstep-base

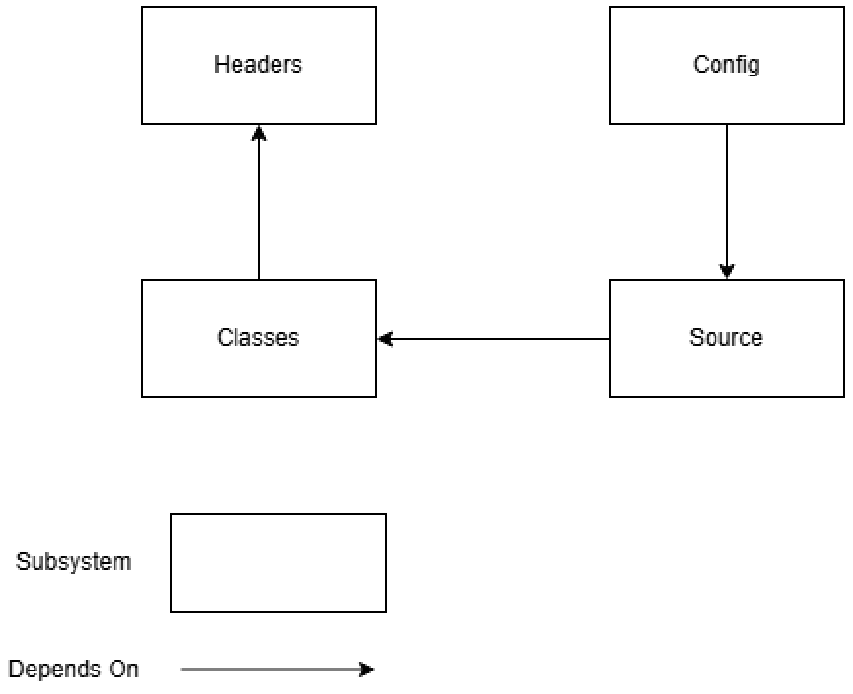


Figure 3: Conceptual Architecture of GNUstep-base Subsystem

4.3.2 Concrete Architecture: GNUstep-base

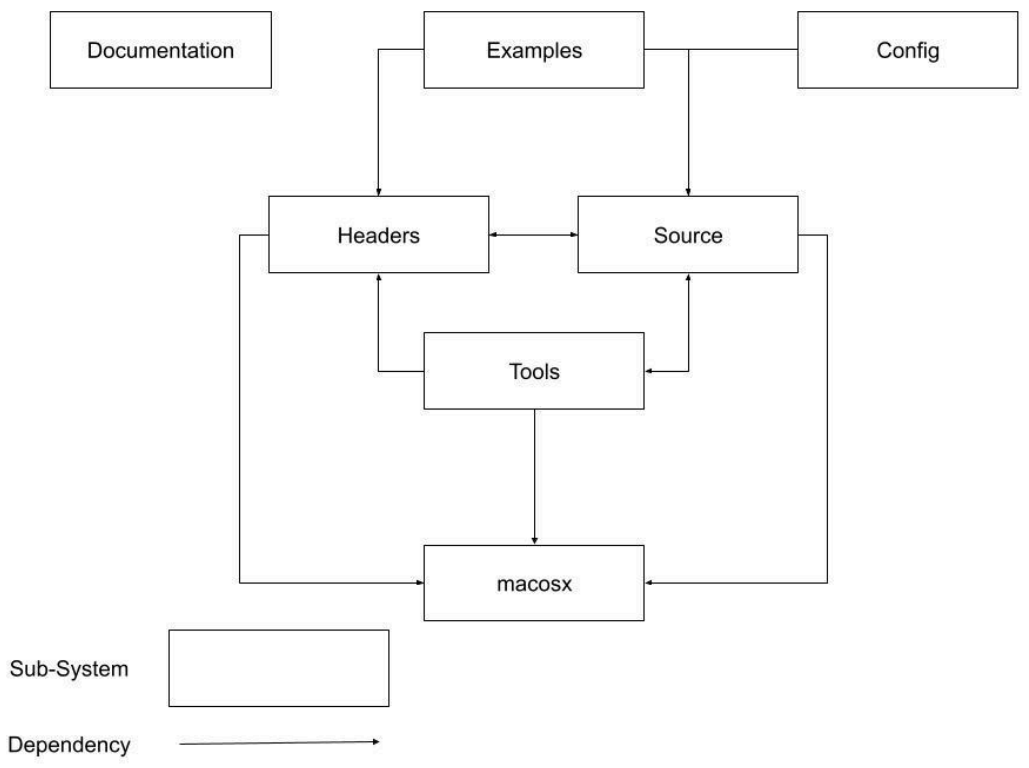


Figure 4: Concrete Architecture of GNUstep-base Sub-System

4.3.3 GNUstep-base SubSystems

Documentation
This component simply provides information about the license of GNUstep and how to properly install GNUstep-base as well as a number of tutorials to introduce software developers into the functions and dependencies of the subsystem's classes. This component is separate from all other components and shares no dependencies.

Examples
This component is included as a demonstration tool for developers to understand how to use the GNUstep-base library efficiently. It provides several example functions that acquaint developers with the capabilities of the GNUstep-base library. This component relies on the Headers and Source components as well as the Objective-C library to ensure that the example files have access to fundamental Objective-C classes required for execution.

Config
This component contains several tests and checks to determine if various parts of the GNUstep-base library are operating correctly. Relies on the Objective-C library as well as the operating system to determine if compatibility is maintained between GNUstep and the various operating systems it is designed to operate on.

Headers
Imports and defines many useful Objective-C classes from the Objective-C library which are then used by all other GNUstep subsystems.

Source
Contains the source code for all non-graphical objective-C classes which provides the reusable code to the other subsystems.

Tools
This component contains classes that build and define various command line utilities used in GNUstep. It ensures these tools are compiled with the correct dependencies, placed in the right directories, and function correctly across different operating systems. It also contains classes required for translation of the GNUstep interface to other supported languages.

MacOSX
This component builds a library that contains classes and extensions found in the GNustep base library but that are not found in the MacOSX foundation framework.

5. Discrepancies Between Concrete & Conceptual

GNUstep has faced a lot of changes. Such a fact is not surprising since as of 2025, there has been 30 years of on & off development by an ever changing open source team. And vast changes in the computing landscape have created new challenges which the GNUstep team had to solve. Discrepancies from the conceptual architecture are significant, the introduction of entire new subsystems such as libs-corebase, as well as how the subsystems are structured internally have changed over time.

5.1 High-level architecture changes

New libraries: Libs-corebase

During concrete development, it's not uncommon for features not considered during its conceptual development to be introduced. However, these new features/objects may not fit into the existing structure of the preplanned libraries. The introduction of new libraries for these objects is the obvious solution to such a problem. A key example is libs-corebase. It is the C implementation of general purpose, non-graphical objects. It is thus similar to its

Objective-C counterpart & forefather, libs-base. But due to being written in the C programming language, rather than Objective-C. It does not fit into any pre-existing library. The rationale to create the additional library libs-corebase was to create a place to store these kinds of C objects.

Structure:

Applications such as GORM, in a layered system, should have only needed to communicate with objects from the application layer. Which stores graphical objects & methods. This does seem nice conceptually until you realise that “method: isEqual” counts as non-graphical. Desiring to be able to consider if two objects were equal. The layered separation was promptly skipped over. The rationale being that it would not make much sense to go through two layers to reach the desired method. So applications that take advantage of non-graphical objects & methods like GORM, will have a dependency on GNUstep-base. GNUstep-back also has some dependencies on GNUstep-base, simply due to requiring these non-graphical objects & methods.

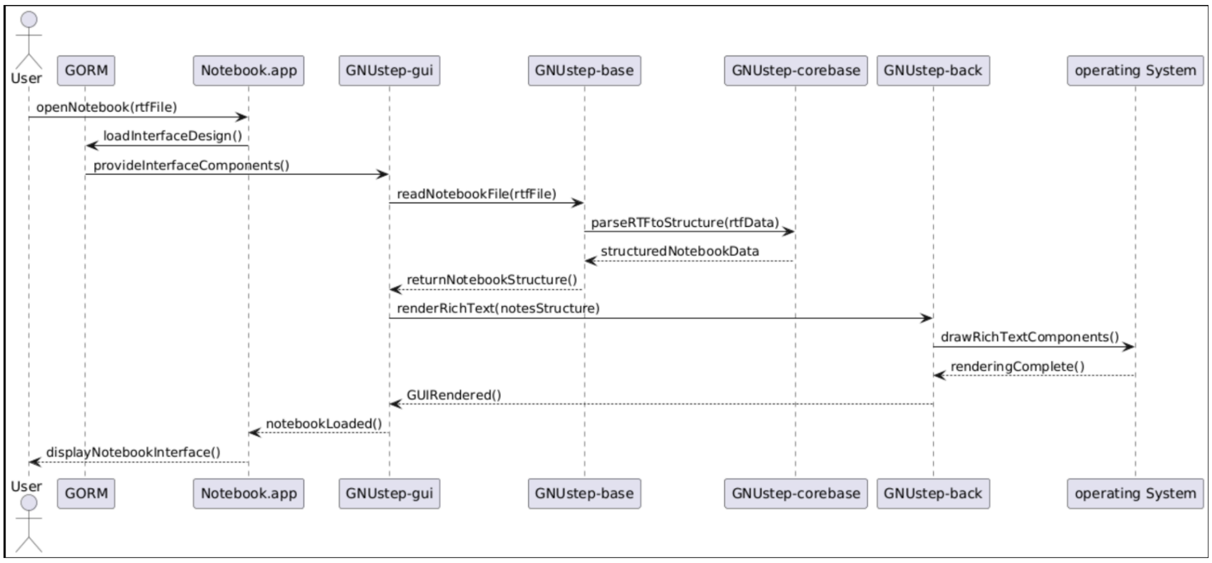
5.2 Subsystem architecture changes

GNU-base Win32:

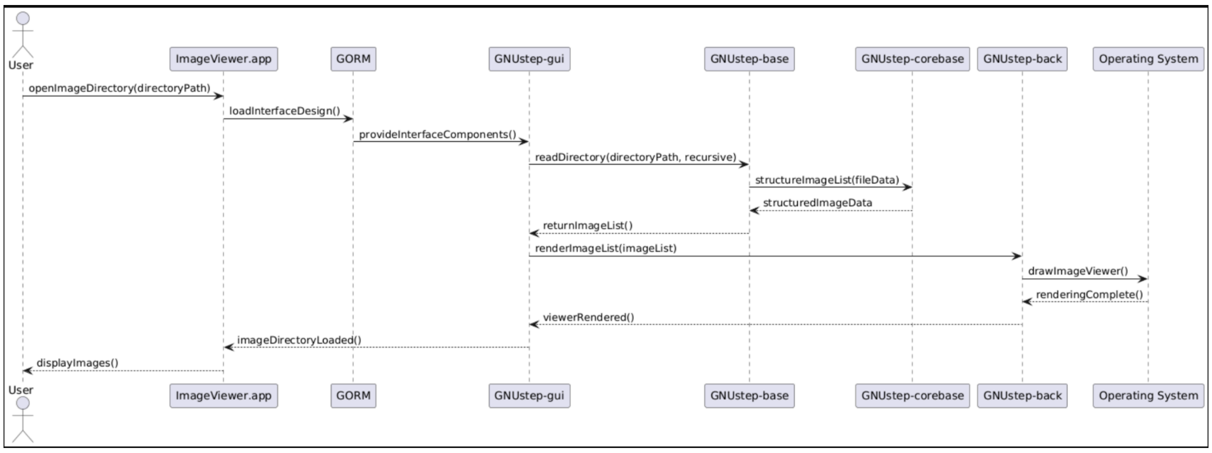
One of GNUsteps founding goals is to be compatible across all platforms. This is normally the responsibility of GNUstep-back. Which would allow for code from the other subsystems to be reused across platforms, instead of writing a new program for each platform. This code reuse would also give the benefit of applications created on one platform to be compatible with any other GNUstep supported platform. Unfortunately, this was simply not possible between unix systems and windows systems by only using GNUstep-back. Due to the significant differences in operating systems & API, you will notice a win32 folder in the Source folder of GNUstep-base. This folder contains rewritten code which is responsible for making GNUstep-base compatible with Win32. “GSFileHandle.m” is one example. Which exists twice in the GNUstep-base subsystem. One for windows systems, & one for unix like systems.

6. Use Cases

- Use Case1: The first use case is Notebook.app. First, the user opens Notebook.app and selects a notebook file to view. Notebook.app then asks GORM to load the interface design previously created. GORM provides the interface components to GNUstep-gui, which then asks GNUstep-base to read the notebook file. GNUstep-base gets help from GNUstep-corebase to organize the notes clearly. After that, GNUstep-gui sends the organized notes to GNUstep-back, which tells the Windowing System to display the notes visually on the screen. Finally, the user can see and read the notes in Notebook.app.



- Use case 2: The second use case is ImageViewer.app. The user opens the ImageViewer.app and chooses a directory containing images to view. ImageViewer.app then asks GORM to load the interface design for viewing images. Then GORM gives the graphical components to GNUstep-gui. Then GNUstep-gui requests will GNUstep-base to read all images in the selected directory and subdirectories. GNUstep-base then uses GNUstep-corebase to organize the list of images clearly. Once ready, GNUstep-gui sends this list to GNUstep-back, which instructs the Windowing System to visually display the images to the user. Finally, the user sees and can explore the images clearly within ImageViewer.app.



7. Lessons Learned

There were several lessons we learned in the analysis of the concrete architecture of GNUStep. We learned many valuable lessons about architecture design which supplemented our knowledge from the last deliverable. We discovered that concrete architecture may vary from the conceptual. Since GNUStep is a complex system, it is difficult to build a completely accurate conceptual architecture and update it as it evolves. Additionally, a significant amount of the documentation is based on OpenStep and does not capture the differences in implementation which GNUStep presents. We also learned that reflexion analysis is an effective method in analyzing the variations.

We also learned about the difficulties of examining and determining the concrete structure of a system. Although the prior knowledge of the conceptual architecture helped us determine how to approach this view, it was still somewhat difficult to understand how all of the components and dependencies of the source code worked together. We also learned to appreciate the understand tool in the analysis of the source code as it significantly simplified our grasp of the components and interactions of the concrete architecture with its visualizations. If we were to simply look at the source code, it would likely have taken us much longer to connect all the information.

Lastly, we learned some lessons about group work. Although assigning leaders to specific parts of the project may allow for a part to be written with more expertise, it may discourage others from comprehending the topic completely. Assigning members to specific parts such as concrete architecture analysis or use cases may have them learn extensively about that specific aspect of the system but not as much for the others. For a cohesive and well-structured report, it is important that everyone understands the whole system when doing their role. Even more favourably, quality would likely be maximized if everyone contributed something to every major part of the report.

8. Data Dictionary

Conceptual Architecture - High level design of a system highlighting the components of a system and their relations in an abstract form

Concrete Architecture - The actual design of a system featuring its implemented modules and their interactions

Subsystem - A component of a larger system which performs a unique function and interacts with other components to perform the larger system's functions

Dependency - A necessary requirement of a component by another component to function

Reflexion Analysis - Comparison of conceptual and concrete architectures to find any inconsistencies between design and implementation

Library - A collection of code with common utilities available to be implemented to avoid unnecessary rewriting

Object-Oriented - An architecture which treats components of a system as independent interacting objects with their own methods and data

Layered - Architecture style where services are grouped into classes and organized in a hierarchy with levels of hierarchy serving their adjacent levels

9. Conclusion

Through the exploration of the concrete architecture of GNUstep's and GNUstep base's concrete architecture we have been able to derive some useful information about the complex structure of GNUstep. Using both our previous knowledge of the conceptual architecture and the newly obtained data about its implementation we have been able to observe the changes GNUstep has experienced in its evolution.

Similarly to the conceptual architecture, we have found that GNUstep uses both a layered and object oriented architecture in its implementation. Despite this similarity, there are still several differences between the conceptual and concrete. For example, the corebase component was not present in the conceptual architecture but is present in the concrete which signifies changes not foreseen when the conceptual architecture was created were made during development.

The architecture is made of several core subsystems with many expected and unexpected dependencies. Examples of the process flow have been explored with use case diagrams of Notebook.app and ImageViewer.app which displayed the system's concrete architecture and roles of individual components in a process. These diagrams demonstrated how the subsystems work together to fulfill the functions of GNUstep.

This report brought several lessons including the benefits of using tools such as understand to examine concrete architecture and downsides of evolving code on the unchanging conceptual architecture. In this report we have added to our comprehension of GNUstep by using concrete architecture to add to our knowledge of conceptual architecture. Using the combined information, we can better understand this complex system and potentially create additions or improvements for the future.

10. References

https://en.wikipedia.org/wiki/Foundation_Kit
GNUstep · GitHub