

[https://www.youtube.com
/watch?v=KJvFHsUGKaE](https://www.youtube.com/watch?v=KJvFHsUGKaE)

GNUstep

ConcreteArchitecture

Rodrigo Del Aguila | Lucas Ferber | Robert He | Vlad Kukov | Enqi Liang | Luc Robitaille (group leader)

TABLE OF CONTENTS

01

Overview

Abstract, general
Introduction

02

Derivation Process

Derivation Process

03

Architecture

Concrete Architecture and
Subsystems

04

1. Discrepancies

Between Concrete &
Conceptual
Architecture

05

Use Cases

Use cases, data dictionary,
naming convention

06

Conclusion

Lessons learned,
Conclusion and references

Abstract

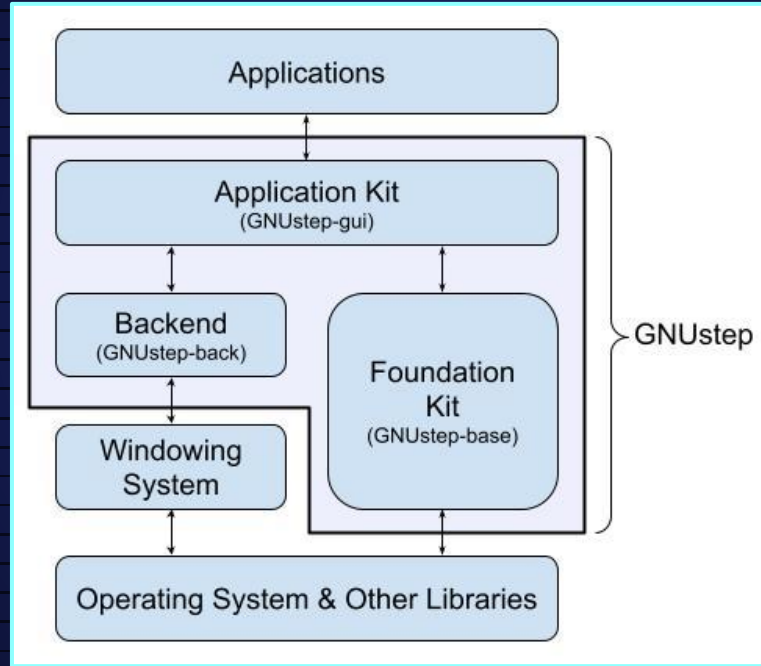
The report examines GNUstep's top-level concrete architecture and the libs-base subsystem in depth—focusing on its key parts: documentation, examples, config, headers, source, and tools. It begins by outlining the group's derivation process and reviewing the conceptual top-level architecture before presenting the concrete version and its architectural style. The report then analyzes both the conceptual and concrete architectures of the libs-base subsystem, discusses reflexion analyses at different levels, showcases two use cases with sequence diagrams, shares group lessons learned, and concludes with a data dictionary and key insights.



Introduction

GNUstep is an open source, object-oriented development environment derived from the OPENstep API and NEXTstep framework. It began in 1995 with three core subsystems—GNUstep-base, GNUstep-back, and GNUstep-gui—which have since expanded to include additional components like GORM and GNUstep-corebase. Over 30 years, its concrete architecture has evolved significantly from its original design, now incorporating Openstep functionality and many new Cocoa classes, with ongoing updates to meet modern standards.

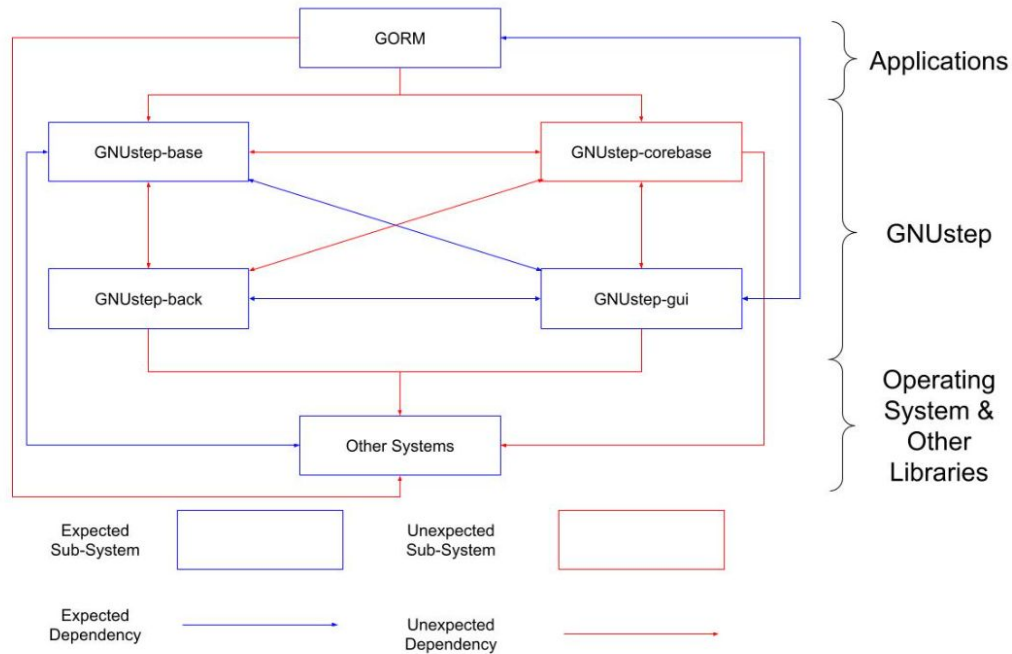
Review of Conceptual Architecture



Concrete ARCHITECTURE

The concrete architecture uses a mix of object-oriented and layered styles, with the object-oriented approach enhancing modular interactions and cross-platform compatibility.

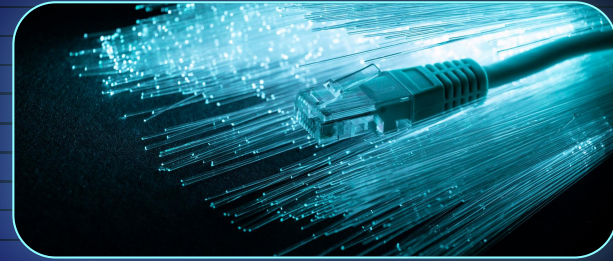
Concrete ARCHITECTURE (contd.)



Subsystems

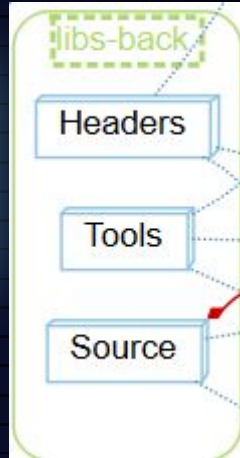
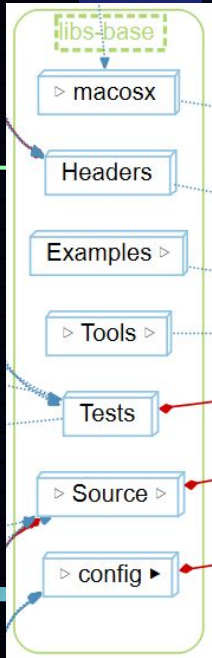
Manages non-graphical Objective-C classes (like strings, byte streams, notifications) and supplies reusable code for data handling, relying on the external Objective-C library.

GNUstep-base



GNUstep-back

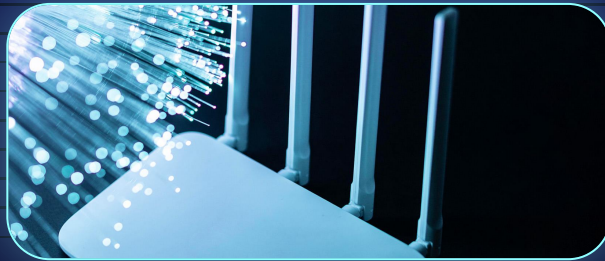
Renders raw graphical data into a displayable format and manages events, relying on GNUstep-core and GNUstep-corebase.



Subsystems

Provides graphical classes for UI elements (fonts, menus, windows, etc.) and depends on GNUstep-back for rendering and GNUstep-base/corebase for core functions.

GNUstep-gui



libs-base

Examples

Tools

macosx

Source

config

Headers

Tests

Documentation

Subsystems

libs-corebase

Headers

Tests

Source

Documentation

Examples

GNUstep-corebase

A cross-platform utility library implementing Apple's Core Foundation API, offering data type abstractions and utilities essential for all subsystems.

Gorm

A GUI design application that leverages GNUstep-gui for rendering and GNUstep-base/corebase for underlying non-graphical operations.

apps-gorm

Tools

Documentation

InterfaceBuilder

GormObjCHeaderParser

Plugins

Applications

GormCore

Conceptual Architecture:GNUStep

GNUstep-base's architecture mirrors the OpenStep Foundation Kit due to limited specific GNUstep documentation, and it centers on the crucial NSObject class, which provides essential Objective-C hierarchies and methods.

GNUstep-base SubSystems

Documentation

Offers license details, installation instructions, and tutorials, operating independently of other components.

Examples

Demonstrates GNUstep-base usage with sample functions, relying on Headers, Source, and the Objective-C library.

Config

Runs tests to verify component functionality and cross-OS compatibility, using the Objective-C library and the operating system.

GNUstep-base SubSystems

Headers

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

Source

Provides the source code for all non-graphical Objective-C classes, offering reusable functionality.

Tools

Builds command line utilities for GNUstep, ensuring proper dependency compilation, placement, and multi-language support.

MacOSX

Creates a library with GNUstep-specific classes and extensions not found in the MacOSX foundation framework.

Discrepancies

- Investigations reveal significant differences between GNUstep's conceptual and concrete architectures.
- Reflection analyses at both the high-level and 2nd-level subsystems justified these discrepancies.
- Key changes include the addition of corebase and tools-make/gnustep-make, driven by 30 years of evolving development and new computing challenges.

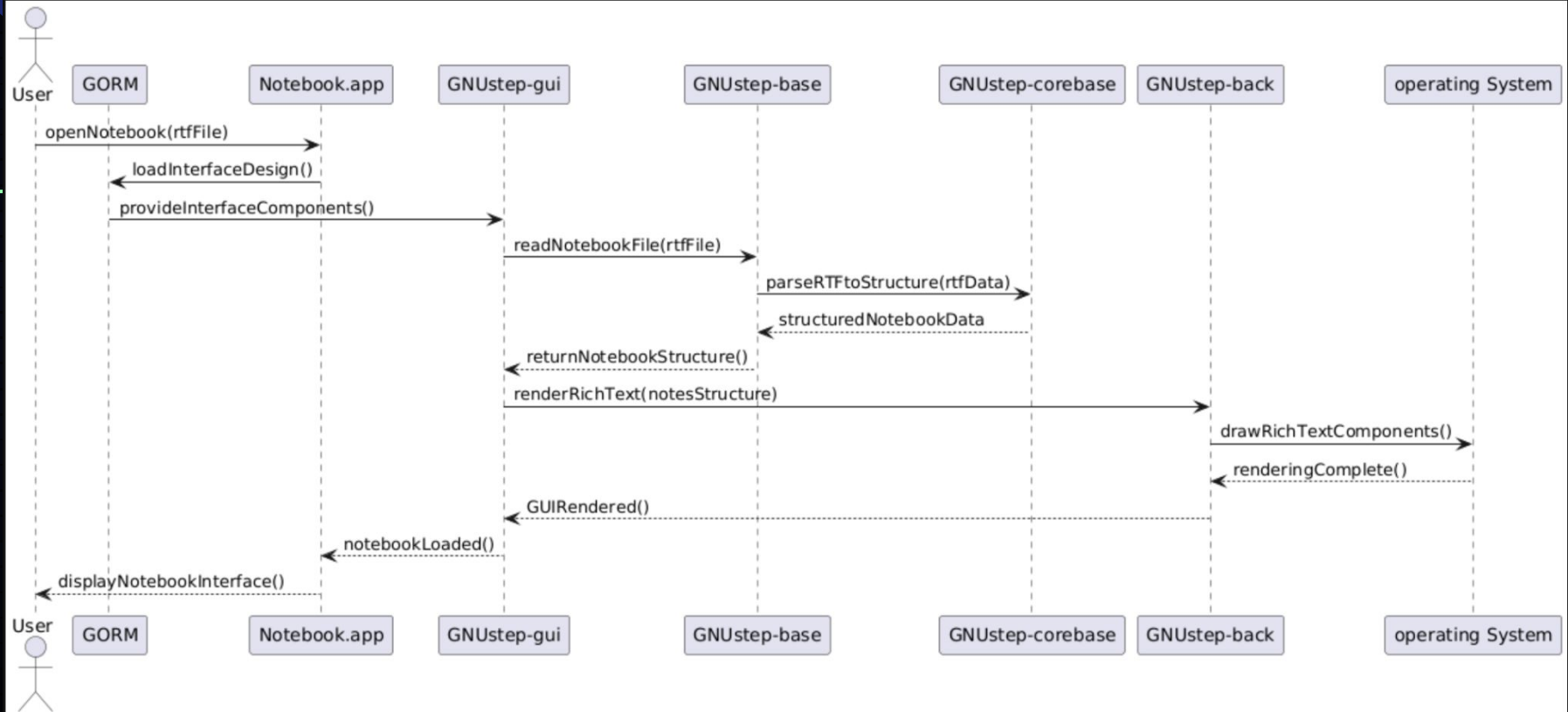
High-level architecture changes

During development, new features sometimes don't fit into existing libraries, leading to the creation of additional ones like `libs-corebase`. This library, written in C, implements general-purpose non-graphical objects similar to `GNUstep-base` but exists separately due to language differences. While a layered system would ideally restrict applications like GORM to interacting only with the application layer (graphical objects), practical needs—such as calling non-graphical methods like `isEqual`—have resulted in dependencies on `GNUstep-base` across subsystems.

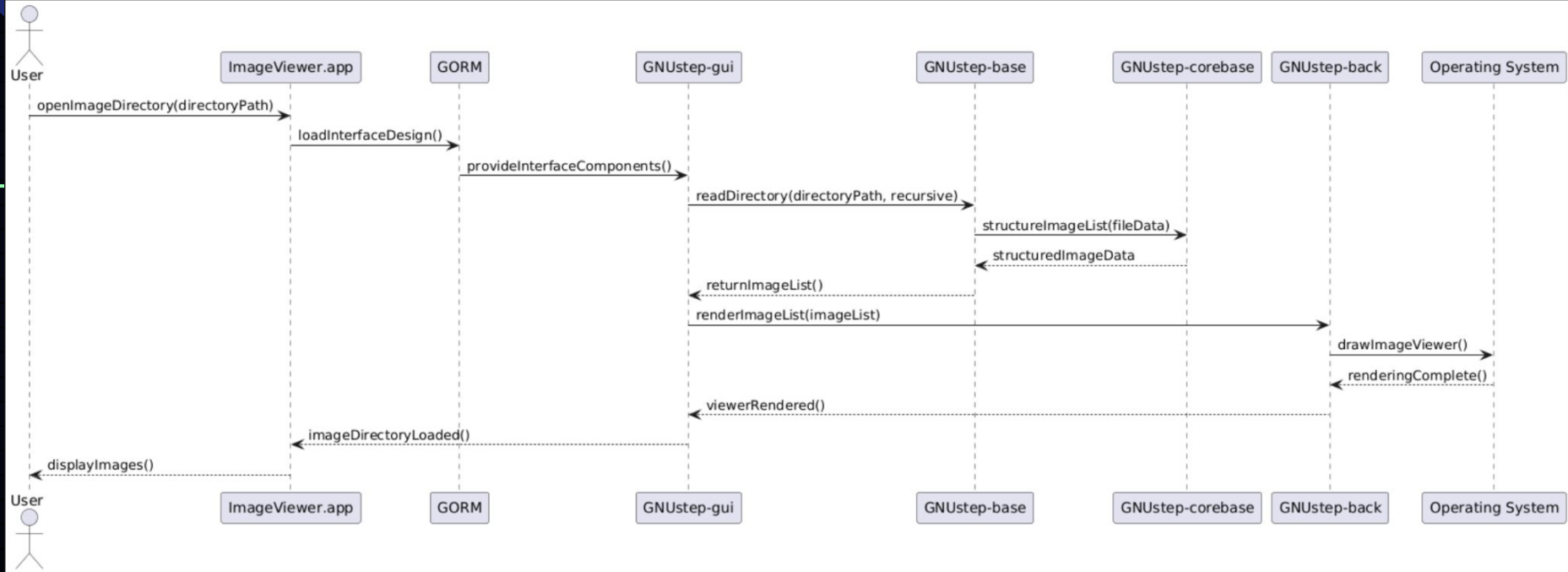
Subsystem architecture changes

GNUstep aims for cross-platform compatibility, typically handled by GNUstep-back for code reuse across systems. However, due to major differences between Unix and Windows, a dedicated win32 folder in GNUstep-base contains rewritten code—such as separate versions of GSFileHandle.m—to ensure Win32 compatibility.

USE CASES: Notebook



USE CASES: ImageViewer



Lessons Learned

The analysis revealed that GNUstep's concrete architecture can differ notably from its conceptual model, partly due to its complexity and reliance on outdated OpenStep documentation. We found that reflexion analysis and visualization tools greatly simplified understanding the intricate source code and component interactions. Additionally, effective group work requires that all team members gain a comprehensive understanding of the entire system rather than specializing in isolated parts, ensuring a cohesive and high-quality report.



Data Dictionary

**Concrete
Architect
ure**


**Conceptual
Architecture**

Dependency

Subsystems

Object-Oriented

**Reflexion
Analysis**



Layered

Library



CONCLUSION

Exploring GNUstep's concrete architecture alongside its base library deepened our understanding of its complex structure and evolution. By combining our conceptual model with implementation data, we noted that while GNUstep employs both layered and object-oriented designs, differences exist—such as the addition of the corebase component that wasn't anticipated in the conceptual model. Use case diagrams for Notebook.app and ImageViewer.app further illustrated how various subsystems and dependencies interact. Overall, the analysis highlighted the benefits of using tools like Understand for examining concrete architecture and the challenges of maintaining a static conceptual model in a continually evolving system, thereby paving the way for potential future improvements.

