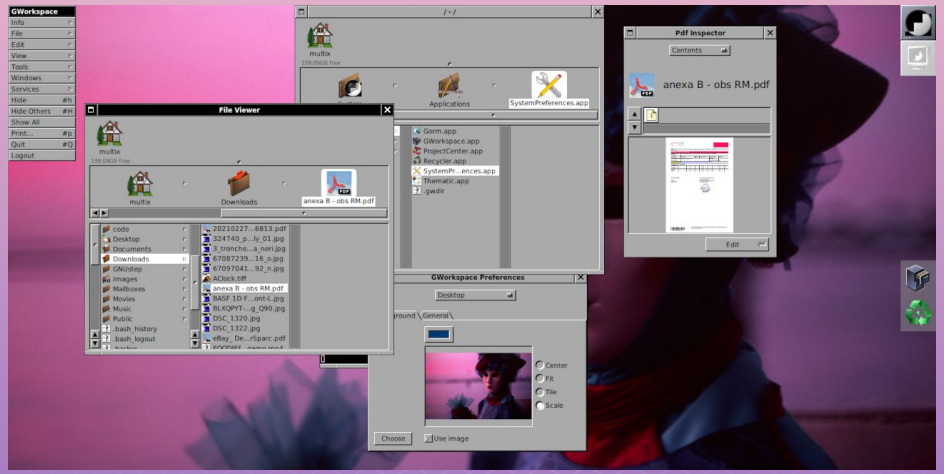
Concrete Architecture of GNUStep

GAMERS NEVER GIVE UP

Jawad Ahmed (Leader), Meagan Mann (Presenter), Ethan Nguyen (Presenter), Zoya Zarei-Joorshari, Kim Hyun Bin, Ripley Visentin

https://www.youtube.com/watch?v=IkSKwQWkjD0



CONCEPTUAL vs CONCRETE

-Blueprint
-Documentation
-Idealized developer view

-Actual architecture
-Implementation
-Examination of open-source code

Main Components

Libs-back

Handles rendering across platforms, translating graphics for different systems.

Libs-base

Provides essential system services and forms the foundation.

Libs-corebase

Adds low-level utilities like memory management and networking.

Libs-GUI

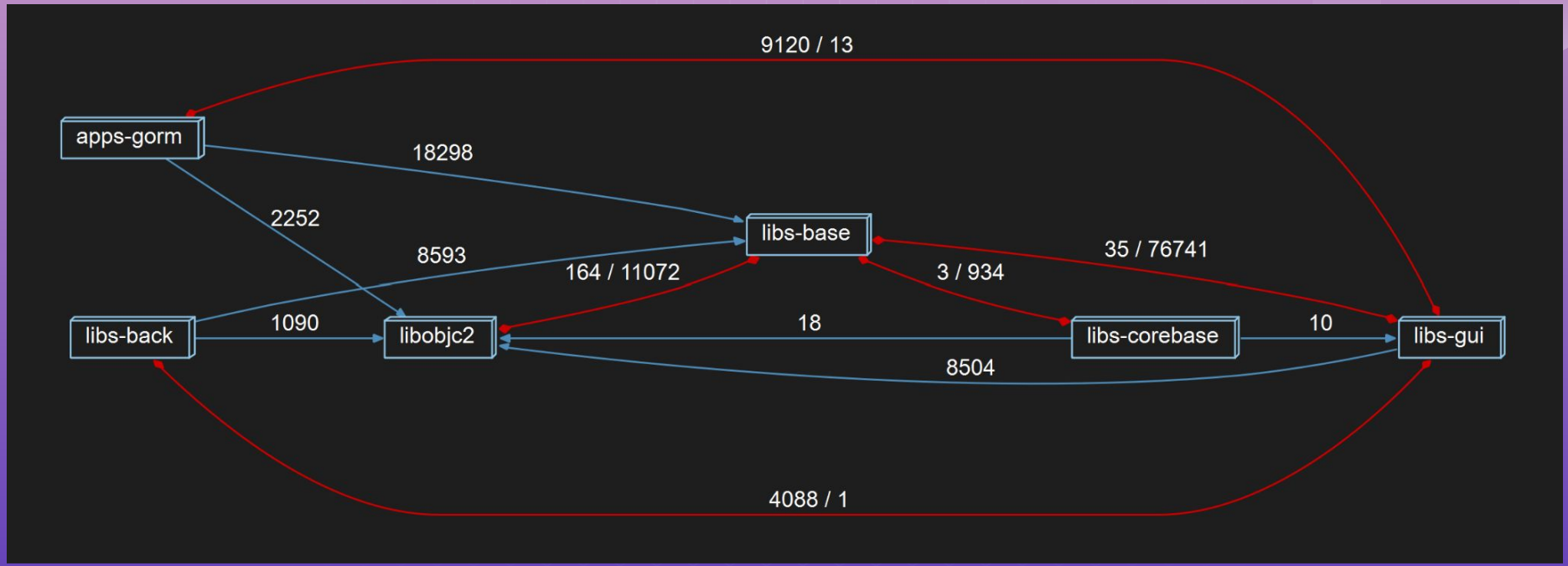
Implements the AppKit API for UI elements.

Gorm

A GUI builder for designing interfaces visually.

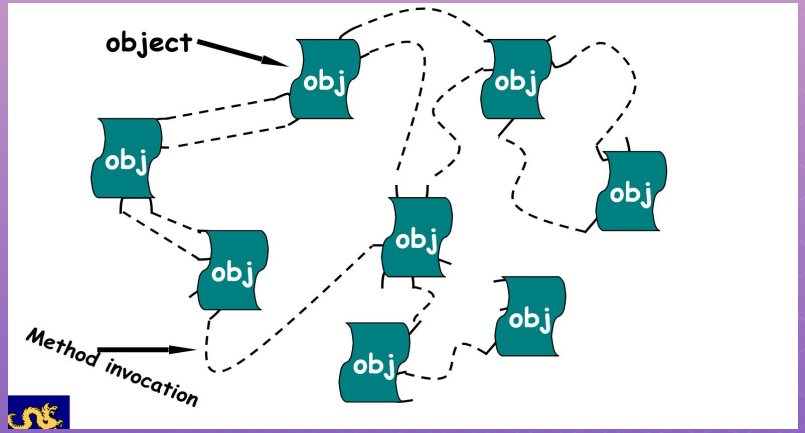
DERIVATION PROCESS

Blue arrows represent one-way dependencies
Red arrows represent two-way dependencies



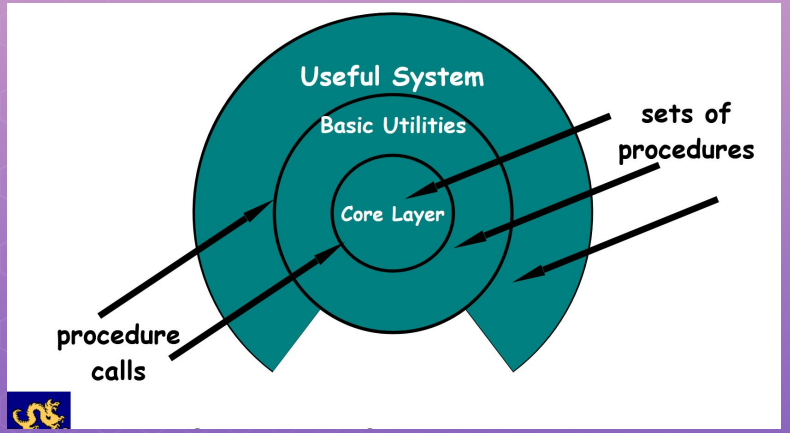
Architectural Style

Object-Oriented Style



CONCEPTUAL

Layered Style



CONCRETE

Gorm

Application

The main GUI application that developers interact with. Depends on **GormCore** for functionality and **InterfaceBuilder** for interoperability

GormCore

The core framework that handles the underlying functionality of **Gorm.** The heart of the subsystem, all components depend on it.

GormObjCHeader Parser

Allows the application to understand and link UI components to Objective-C classes. Works closely with **GormCore**.

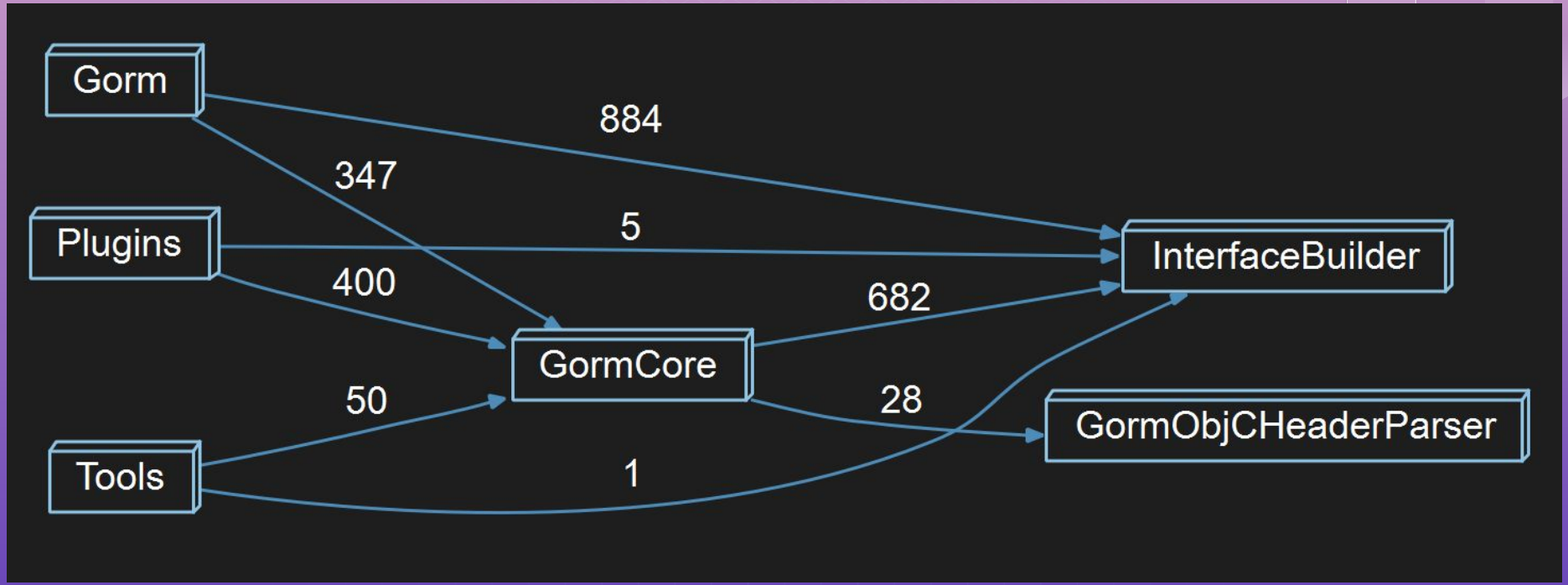
InterfaceBuilder

Originally from Legacy NeXTSTEP,. **Gorm** depends on it to ensure compatibility with different Objective-C environments

Plugins/Tools

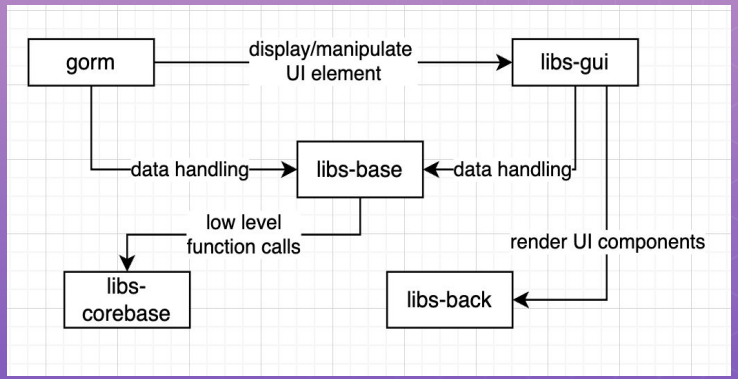
Provide and extend functionality by adding additional UI features and command-line operations respectively. Depends on **GormCore**.

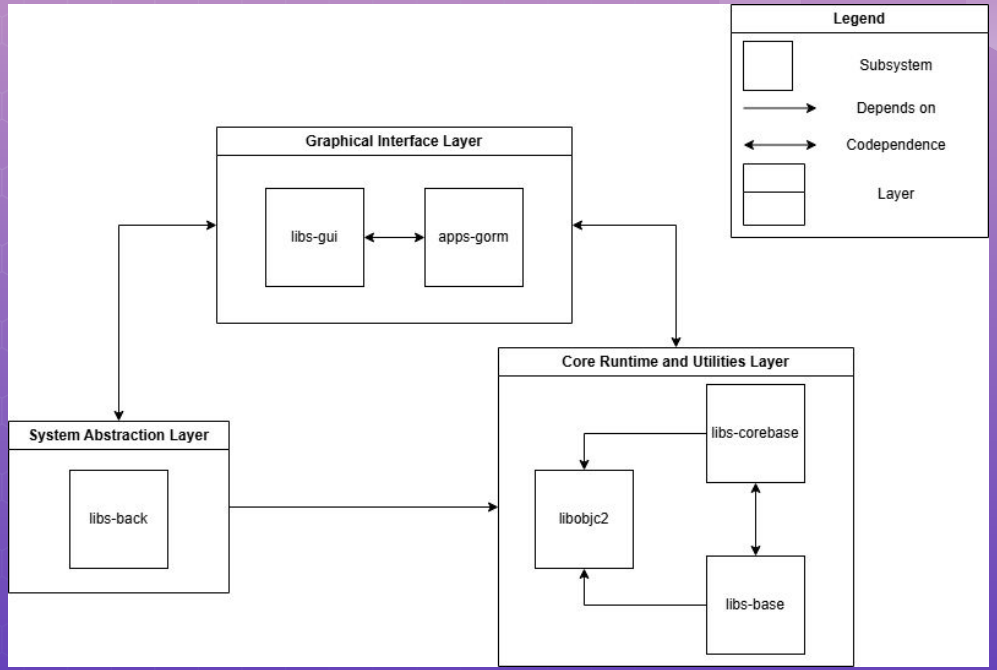
Dependency Graph of Gorm



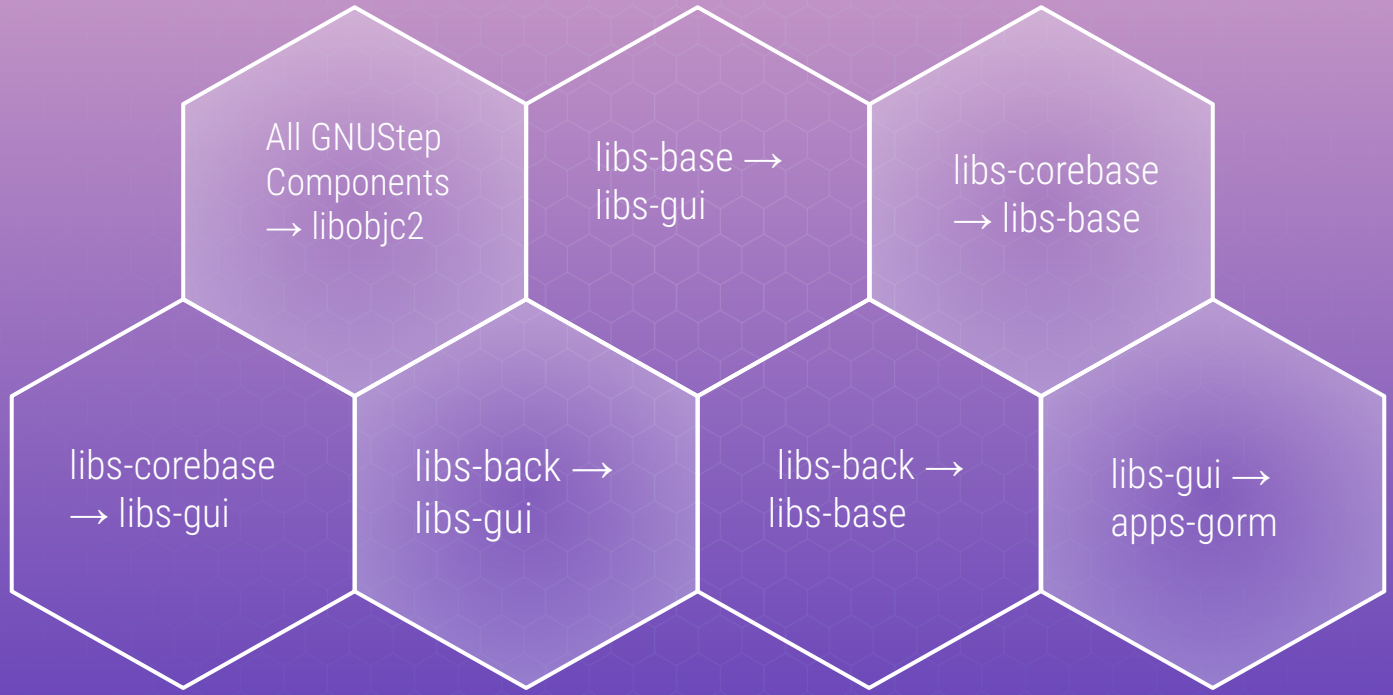
High Level Reflexion Analysis

Object Oriented Architecture → Layered Architecture





High Level Unexpected Dependencies



Subsystem Reflexion Analysis

Gorm → **InterfaceBuilder**: Gorm relies on InterfaceBuilder for cut/paste operations, palettes, and document handling.

GormCore → **InterfaceBuilder**: InterfaceBuilder provides GormCore with resource management and UI element controls.

Plugins → **GormCore** → **InterfaceBuilder**: GormCore uses plugins to recognize file types like XIB, Nib, Gorm, and GModel.

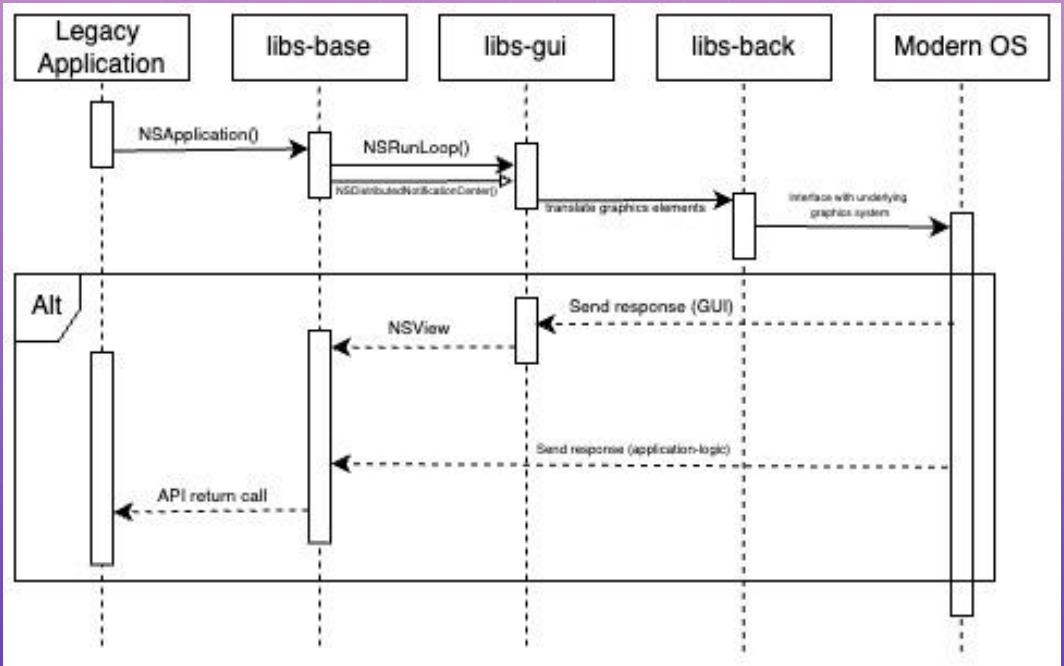
Gorm → **GormCore**: GormCore manages GUI logic, file handling, and data models for UI elements.

GormCore → **GormObjCHeaderParser**: Parses Objective-C classes for GUI design.

gormtool → **GormCore**: GormCore enables gormtool to import/export classes and process file types.

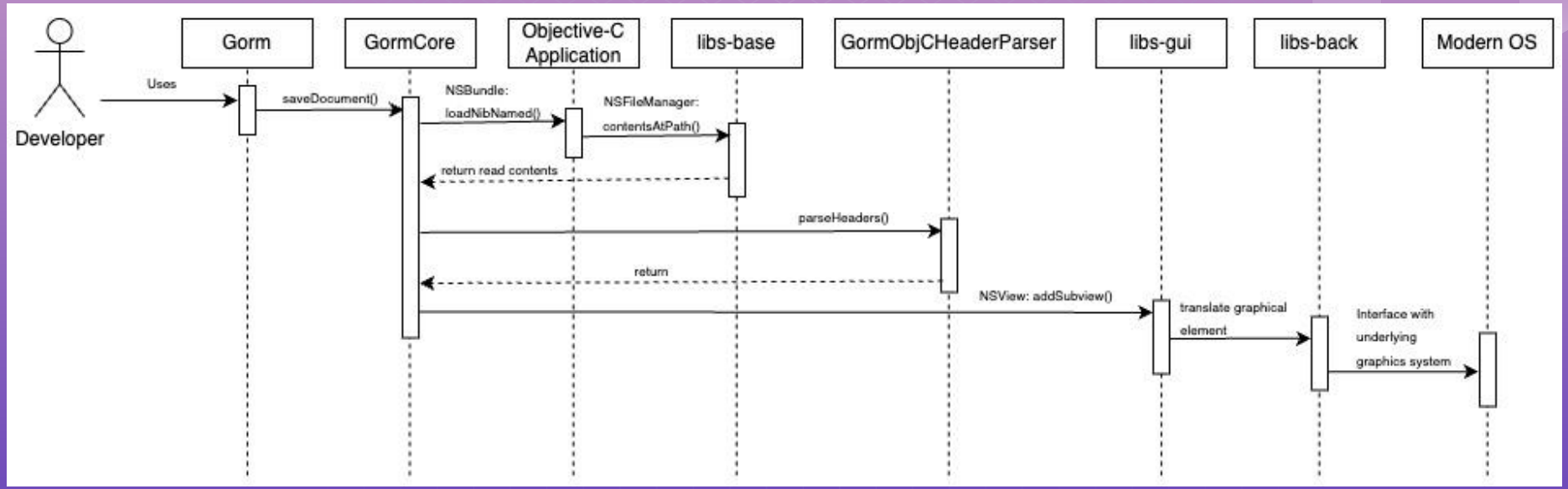
Sequence Diagrams: Use Case I

Modernizing Legacy OpenStep Applications



Sequence Diagrams: Use Case 2

Cross-Platform Development With Objective-C



Lessons Learned

1. **Conceptual vs Concrete Architecture**:
   The shift from conceptual to concrete architecture showed the difficulty of balancing ideal designs with real-world constraints, leading to a more structured Layered Architecture.

2. **Dependency Analysis**:
   Automated tools uncovered unexpected cross-layer dependencies, emphasizing the need to understand system interactions.

3. **Documenting Architectural Decisions**:
   Documenting both ideal and actual designs helped identify inconsistencies and guide improvements for scalability and maintainability.

Conclusion

Concrete Architecture Analysis: Used the **Understand tool** to examine subsystem dependencies.

Reflexion Analysis: Identified **convergences, divergences, and absences** in the architecture.

Architectural Shift: Initially designed as **object-oriented**, but transitioned to a **layered architecture**.

apps-gorm Subsystem: Detailed examination and updated **sequence diagrams** validated the benefits of the layered structure.

Thanks!

Questions?