https://youtu.be/XF3X_TlexBo

Big Hero 6 Team

Team Members and Roles

Team Big Hero 6 - Group 30

- Terence Jiang - Group Leader, Use Cases
- Kevin Chen - References, Lessons Learned
- Alexander Zhao - Presenter, Abstract, Slide Presentation
- Carter Gillam - Presenter, Reflexion of High Level, Data Dictionary
- Kavin Arasu - Conclusion, Introduction & Overview, Concrete Subsystems & Interactions
- Daniel Gao - Reflexion of Gorm, Concrete and Conceptual Architectures of Gorm

Gorm Concrete Architecture

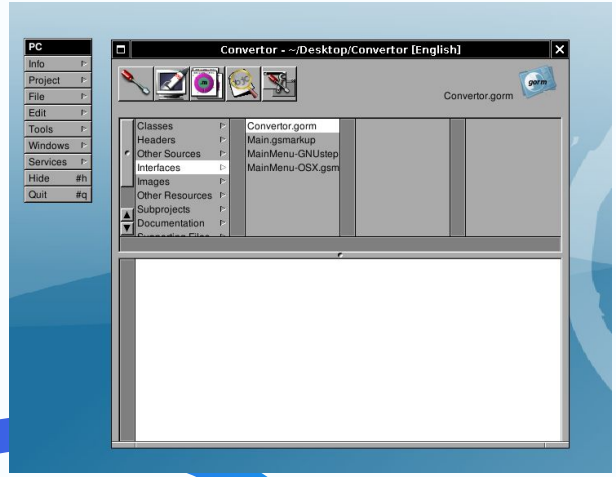
March 14, 2025

Abstract

Concrete Architecture Analysis of GNUstep and Gorm

The GNUstep system was looked at in our last report to determine its conceptual architecture. This report will examine the concrete architecture of the GNUstep, by analyzing its structure through the use of the Understand tool and comparing it to the conceptual architecture that we've updated from our previous report based on group 20's conceptual architecture. After investigating and recovering the actual system dependencies, we've identified the interactions between each subsystem and their interactions, finding many differences from our conceptual model. Furthermore we will touch upon a detailed study of the GORM subsystem, exploring its internal architecture, interactions and alignment with conventional conceptual expectations. Finally, we will provide a rationale for every identified divergence, examining constraints and trade offs. Finally, in order to illustrate real-world cases, we have provided two sequence diagrams for non-trivial use cases.

Introduction & Overview



Derivation Process:

- Use the Understand tool to examine dependencies.
- Group top-level entities into subsystems and define their interactions.
- Refine the architectural model based on findings.

Comparative Analysis:

- Identify key discrepancies between conceptual and concrete architectures.
- Provide rationale for divergences and document necessary modifications.
- Consider influences from other groups' work where relevant.

In-Depth Subsystem Analysis:

- Focus on ‘Gorm’, examining its internal structure and dependencies.
- Compare conceptual vs. concrete views of this subsystem.

Sequence Diagrams for Non-Trivial Use Cases:

- Illustrate interactions for loading/parsing an Objective-C header file in Gorm.

Derivation Process

Reevaluating Conceptual Architecture:

- Initial conceptual architecture did not accurately represent GNUstep's actual structure.
- Contained irrelevant subsystems not part of core GNUstep.
- Based on professor's recommendation, adopted Group 20's interpretation.
- Group 20's model was simpler, clearer, and focused on key subsystems.

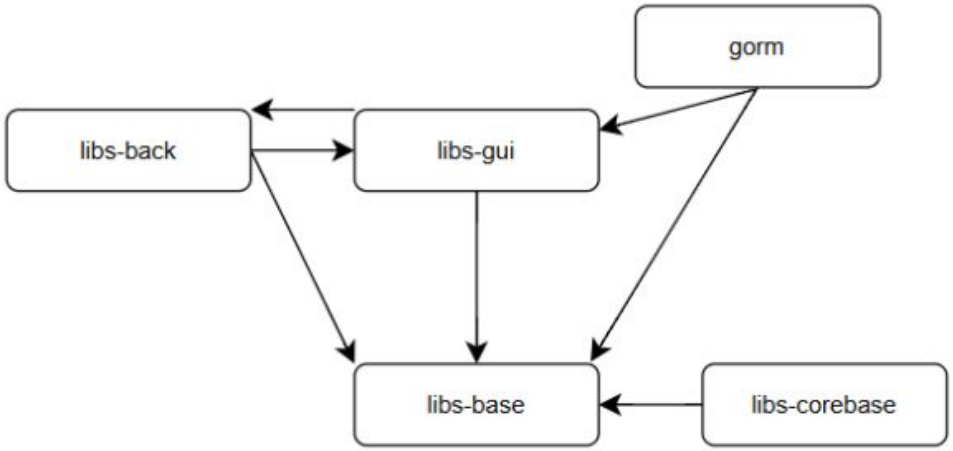
Deriving the Concrete Architecture:

- Used Understand tool to analyze and visualize dependencies.
- Imported GNUstep's source code to examine subsystem interactions.
- Avoided manually parsing source code, allowing for a more efficient analysis.

Key Differences Between Conceptual and Concrete Architectures:

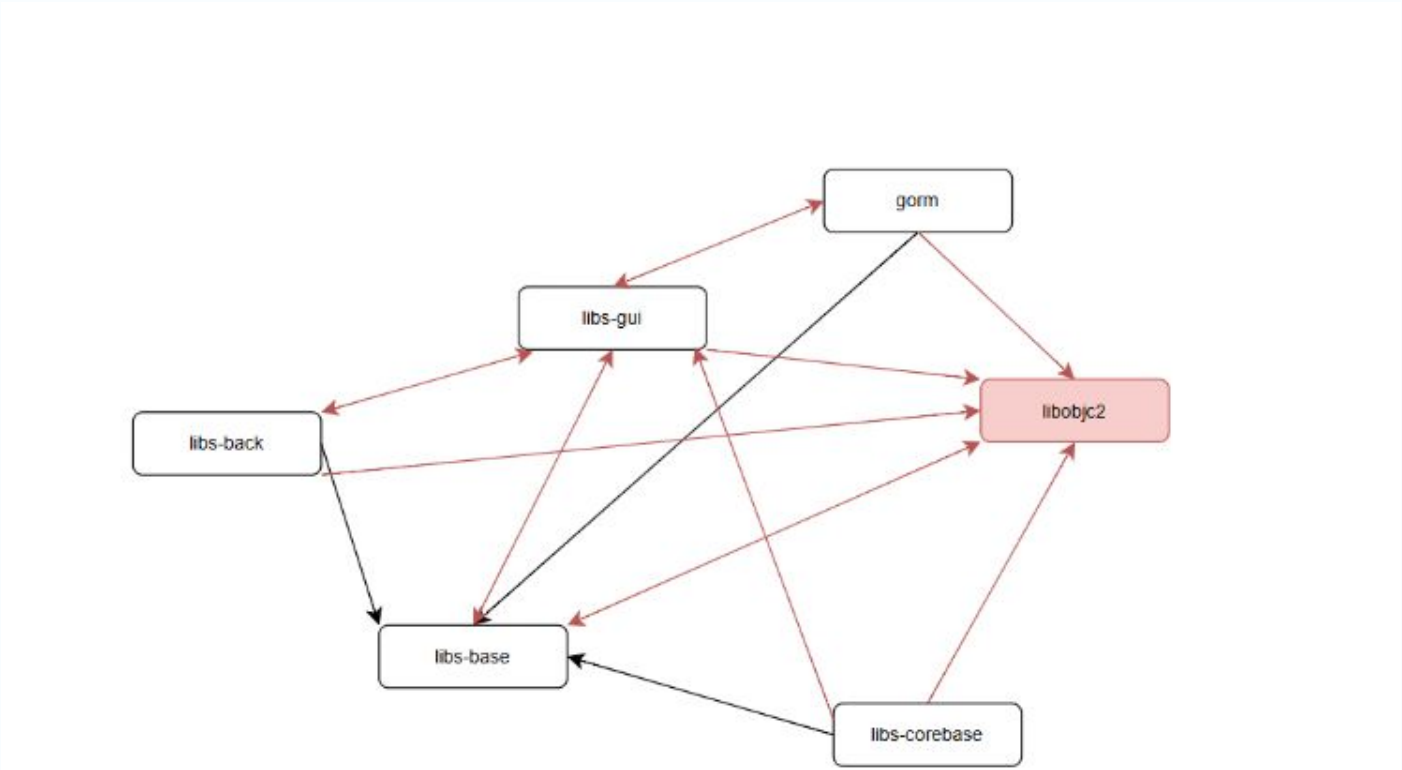
- Extra subsystem ('libsobjc2') found in concrete architecture.
- Subsystem interactions were more complex than initially expected.
- Differences will be explored in later sections of the report.
- Despite differences, the concrete architecture confirms the object-oriented design of GNUstep.

Conceptual Architecture (High level)



Conceptual Architecture of GNUstep Based on Group 20's A1

Concrete Architecture (High level)



Concrete Subsystems and Interactions

New Subsystem: libobjc2

- Objective-C runtime used by GNUstep
- Manages dynamic features: message passing, method resolution, class/object data
- Compatible with Apple's runtime, supporting:
  - Automatic Reference Counting (ARC)
  - Blocks
  - Optimized method dispatching
- Enables dynamic loading, reflection, and runtime manipulation
- Essential for GNUstep's modular and flexible architecture

libs-corebase

- Minor but important interaction with libs-base
- Extends functionality for CoreFoundation compatibility

Subsystem Interactions:

- **libs-base**
  - Strong dependency on libobjc2 for class management, method dispatching, and object interactions
  - Depends on libs-corebase for CoreFoundation compatibility
- **libs-gui**
  - Builds on libs-base for core functionalities
  - Provides graphical user interface capabilities
  - Bidirectional dependency with libs-back for rendering
- **libs-back**
  - Works with libs-gui for rendering backend
  - Ensures GUI compatibility across different windowing systems
  - Interacts with libobjc2

Reflexion Analysis - New Dependencies

apps-gorm → libs-base (7665 dependencies)

- Expected to rely on libs-gui but has a strong dependency on libs-base.
- Likely due to performance optimizations or historical design choices.
- Leads to tighter coupling, making future modifications more complex.

libs-gui → libs-base (24,009 dependencies, 13 primary)

- Expected to mainly depend on libs-back but heavily relies on libs-base.
- GUI operations depend on system-level functions like window management.

libs-back ↔ libs-corebase (29/3 dependencies)

- Unexpected bidirectional dependency suggests rendering relies on CoreFoundation.
- Likely due to platform compatibility concerns, increasing architectural complexity.

libs-base ↔ libs-corebase (475/294 dependencies)

- Originally intended as independent, now have a bidirectional dependency.
- Indicates shifting system responsibilities, reducing flexibility and increasing maintenance effort.
-

Reflexion Analysis - Removed Dependencies

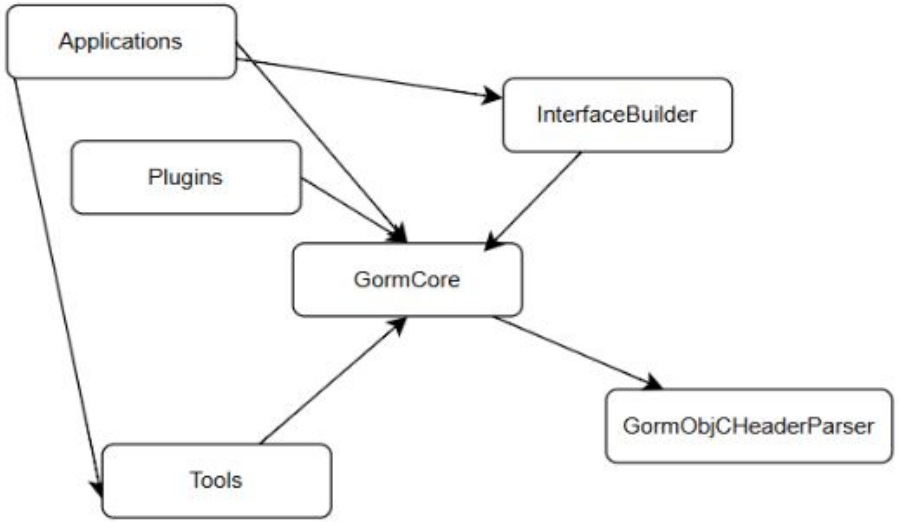
apps-gorm → libs-gui (Weaker Than Expected – 5235/3 dependencies):

- Expected **strong dependency** for UI construction, but it's significantly weaker.
- Instead, **apps-gorm interacts more heavily with libs-base**.
- Suggests **some UI logic is handled at the system level** rather than in libs-gui.
- **Makes GUI behavior harder to modify and maintain**, as UI logic is now spread across layers.

libs-gui ↔ libs-corebase (Weaker Than Expected – 10/35 dependencies):

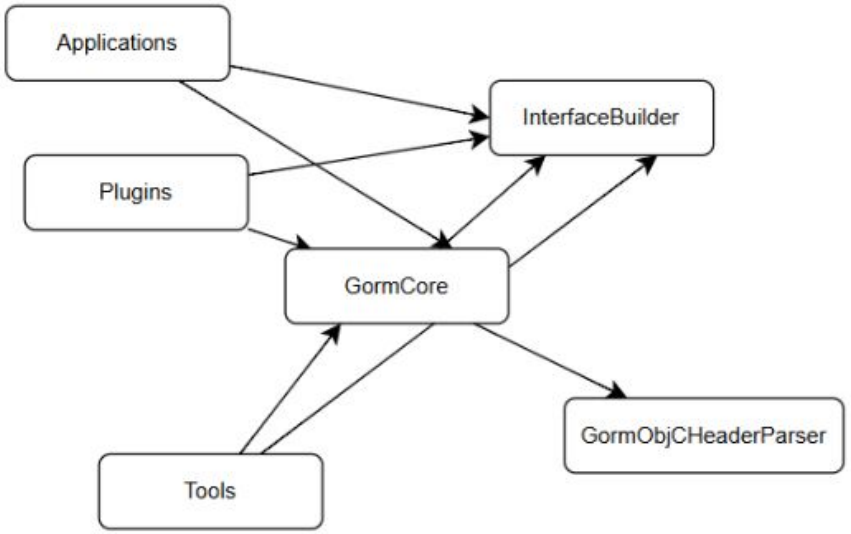
- Expected **strong reliance on CoreFoundation functions**, but actual dependency is weak.
- Indicates that **libs-gui is more autonomous** than initially thought.
- **Enhances modularity**, but the conceptual model **overestimated libs-gui's reliance on CoreFoundation utilities**.

Conceptual Architecture (Gorm)



Conceptual Architecture of Gorm

Concrete Architecture (Gorm)



Redrawing of Gorm's Concrete Architecture from Understand

Gorm Subsystems and Interactions

GormCore: Central system for database management and business logic.
InterfaceBuilder: Handles UI components, used for designing interfaces.
GormObjCHeaderParser: Parses Objective-C headers to integrate objects with GormCore.
Applications: Depend on GormCore for database interaction and InterfaceBuilder for UI design.
Plugins: Extend GormCore's functionality and use InterfaceBuilder for custom UI elements.

Tools: Utilize GormCore for database utilities and InterfaceBuilder for UI components.
GormCore Dependencies: Relies on InterfaceBuilder for UI generation and GormObjCHeaderParser for parsing.
Architecture Style: Repository-style, with GormCore as the central system and other components loosely coupled.

Gorm Reflexion Analysis - New Dependencies

Plugins → InterfaceBuilder

- Originally expected to depend only on GormCore.
- Requires direct access to InterfaceBuilder for UI component management.
- Ensures integration with external GUI standards and the broader GNU ecosystem.

GormCore → InterfaceBuilder

- Needed for handling nib files and managing UI layouts.
- Centralizes UI-related operations, improving maintainability.

Tools → InterfaceBuilder

- Initially expected to rely only on GormCore.
- Requires direct access to InterfaceBuilder for UI debugging and advanced inspection.
- Provides developers with essential UI development utilities.

Gorm Reflexion Analysis - Removed Dependencies

Application → Tools Dependency Removed

- Originally relied on Tools for debugging and inspection.
- Now accesses necessary functions directly from GormCore.
- Simplifies the system, reduces conflicts, and improves efficiency.

InterfaceBuilder → GormCore Dependency Removed

- Initially depended on GormCore for UI management.
- Found unnecessary in real implementation.
- Allows InterfaceBuilder to be maintained independently, avoiding unintended side effects.

Use Case #1

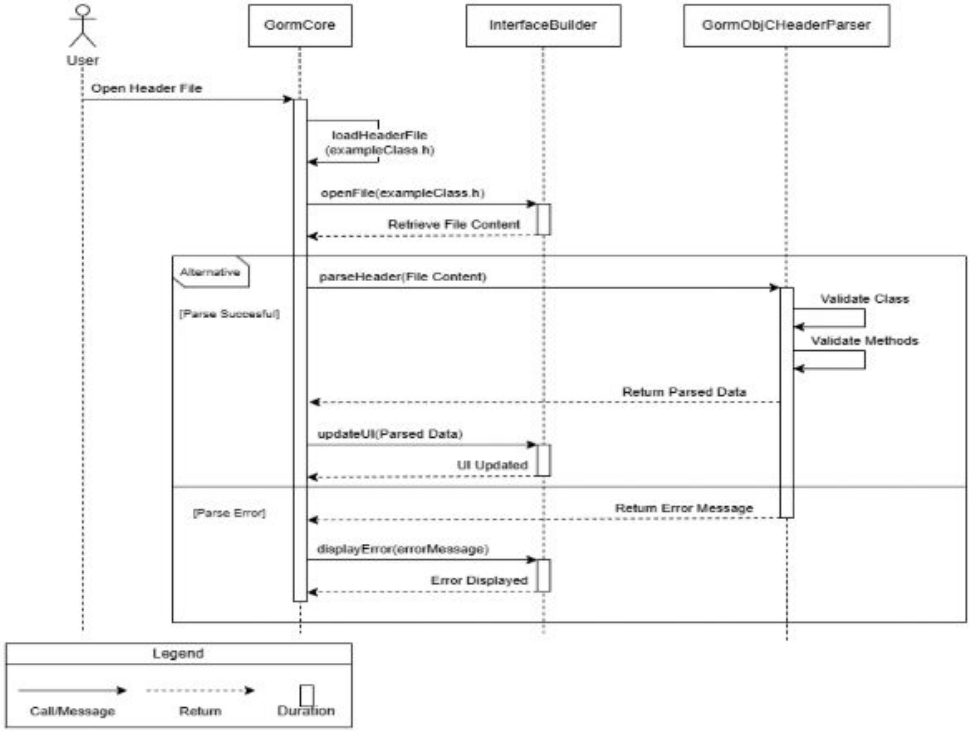


Figure 1.

Use Case 1 Loading and Parsing an Objective-C Header File

Use Case #2

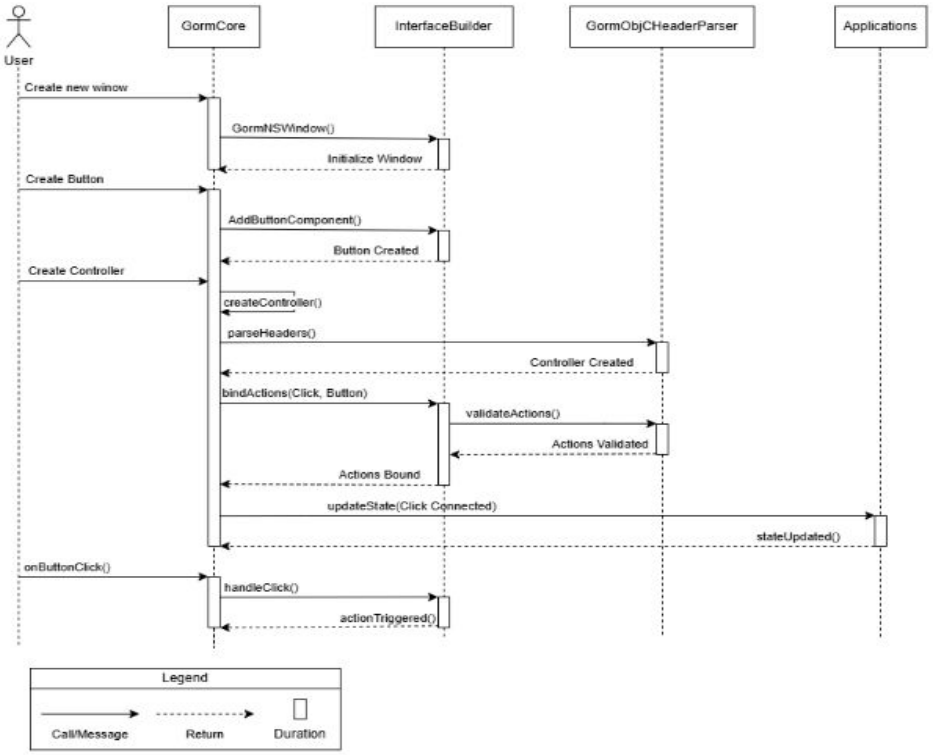


Figure 2.

Use Case 2 Connecting Button Component to an Action

Lessons Learned & Conclusion

Key Insights from GNUstep and Gorm Analysis

- **Conceptual vs. Concrete Discrepancies**: Expected clear subsystem interactions, but real dependencies were more complex.
- **Importance of Understand Tool**: Helped uncover hidden dependencies and visualize module interactions.
- **Complexity of Concrete Architecture**: More intricate than anticipated, with unexpected relationships.
- **Team Collaboration**: Shared workload on difficult tasks improved efficiency.
- **In-Person Meetings**: Enabled faster problem-solving and better workflow.