Analysis of ScummVM's Concrete Architecture

Video Link: https://www.youtube.com/watch?v=Ivv0Ha_3_N4

Arda Utku and Oliver Macnaughton

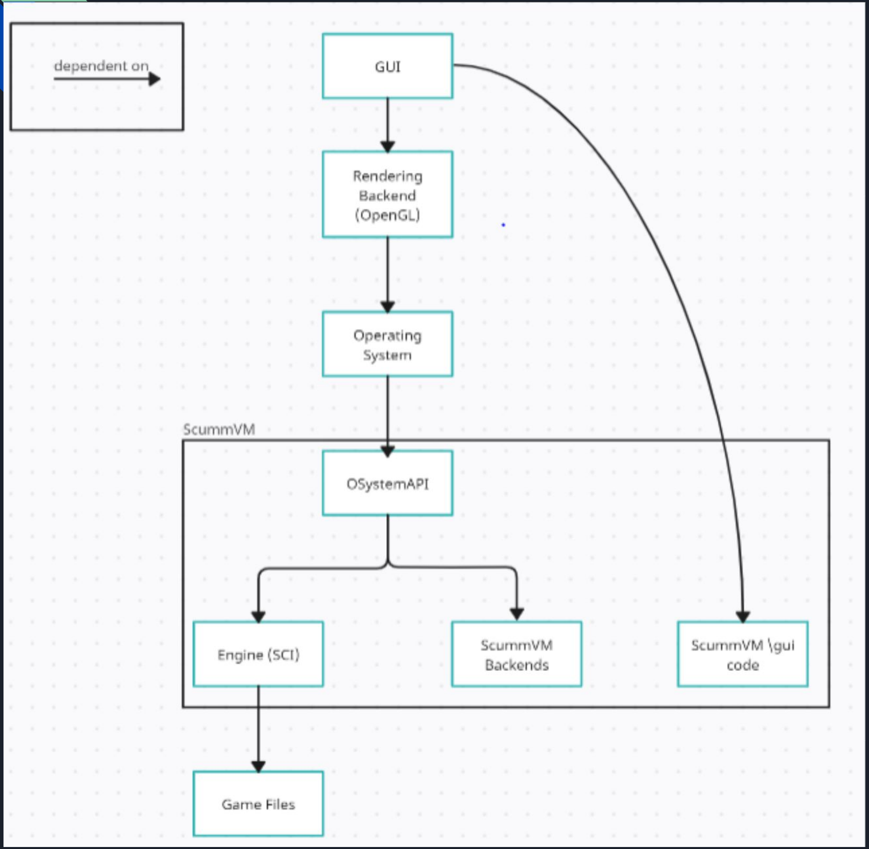
Team Member:

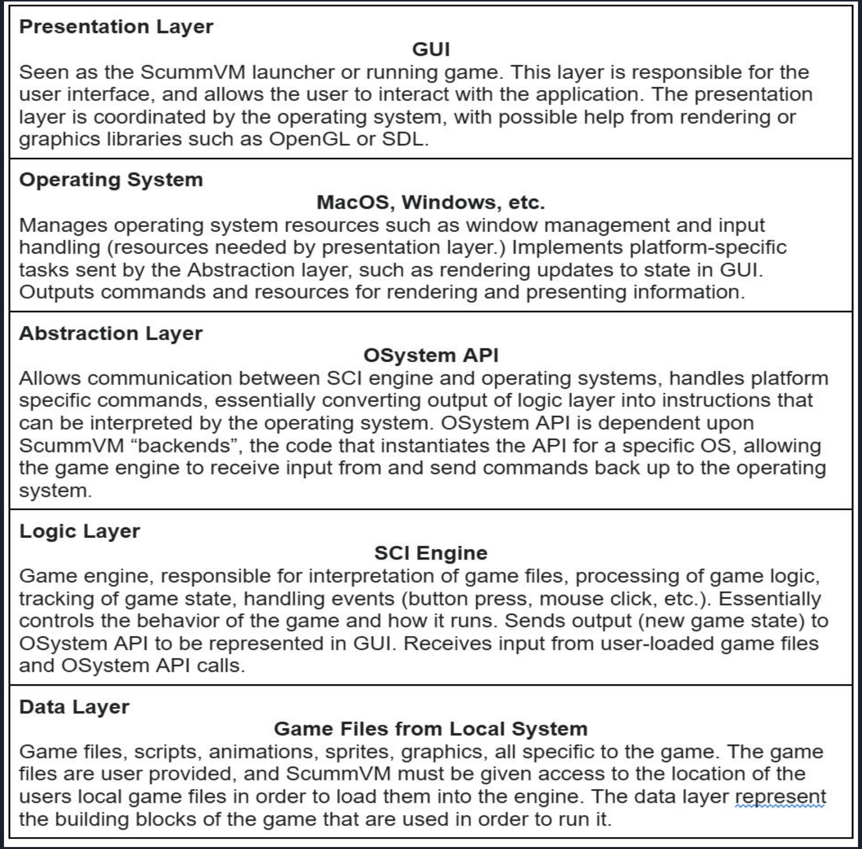
- Arianne Nantel (Team Lead, Participated in research, wrote Abstract and conclusion, use case and edited the report)
- Kyra Salmon (Participated in research, wrote Introduction and lessons learned, use case and edited the report and made reference list)
- Andrew Clawson (Participated in research wrote about the SCI engine in the report)
- Daniel Dousek (Participated in the research, wrote about the external interfaces work together)
- Arda Utku (Presentation, participated in research and wrote about the SCI engine)
- Oliver Macnaughton (Presentation, participated in research and wrote about ScummVm)

Presentation Overview

- Conceptual vs Concrete architecture of ScummVM
- Reflexion analysis of ScummVM
- Derivation and considered alternatives
- Conceptual vs Concrete architecture of SCI Engine
- Reflexion analysis of SCI Engine
- Key takeaways

Conceptual vs. Concrete: Conceptual Architecture





Presentation Layer

GUI

Seen as the ScummVM launcher or running game. This layer is responsible for the user interface, and allows the user to interact with the application. The presentation layer is coordinated by the operating system, with possible help from rendering or graphics libraries such as OpenGL or SDL.

Operating System

MacOS, Windows, etc.

Manages operating system resources such as window management and input handling (resources needed by presentation layer.) Implements platform-specific tasks sent by the Abstraction layer, such as rendering updates to state in GUI. Outputs commands and resources for rendering and presenting information.

Abstraction Layer

OSystem API

Allows communication between SCI engine and operating systems, handles platform specific commands, essentially converting output of logic layer into instructions that can be interpreted by the operating system. OSystem API is dependent upon ScummVM “backends”, the code that instantiates the API for a specific OS, allowing the game engine to receive input from and send commands back up to the operating system.

Logic Layer

SCI Engine

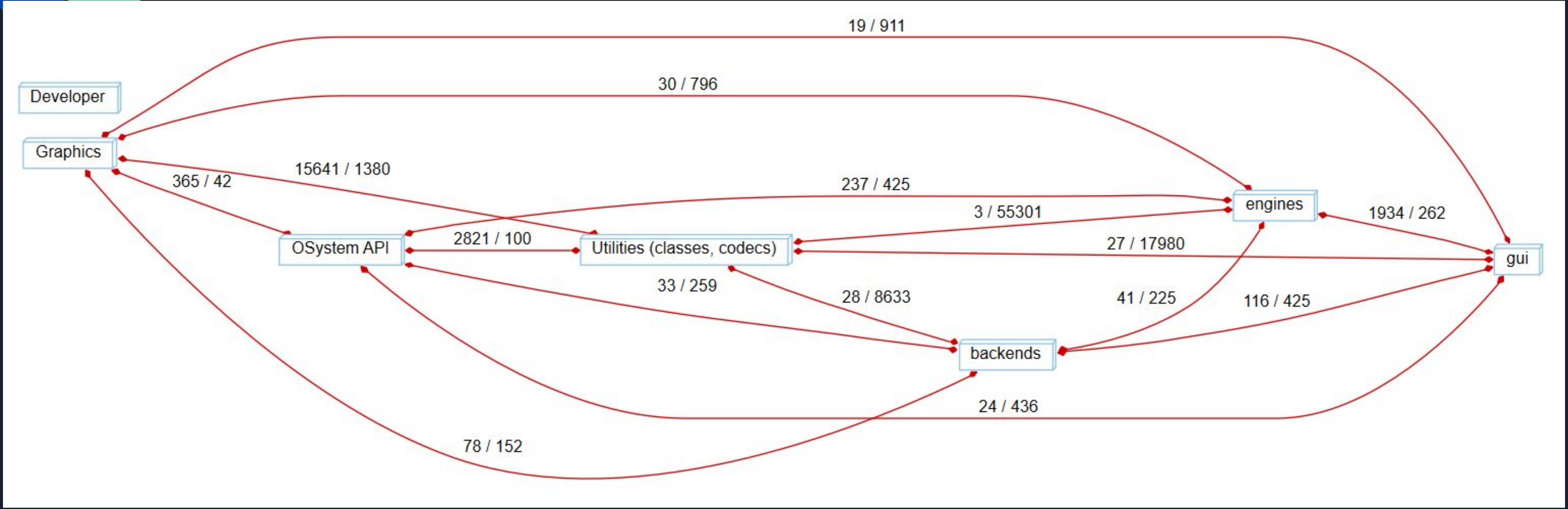
Game engine, responsible for interpretation of game files, processing of game logic, tracking of game state, handling events (button press, mouse click, etc.). Essentially controls the behavior of the game and how it runs. Sends output (new game state) to OSystem API to be represented in GUI. Receives input from user-loaded game files and OSystem API calls.

Data Layer

Game Files from Local System

Game files, scripts, animations, sprites, graphics, all specific to the game. The game files are user provided, and ScummVM must be given access to the location of the users local game files in order to load them into the engine. The data layer represent the building blocks of the game that are used in order to run it.

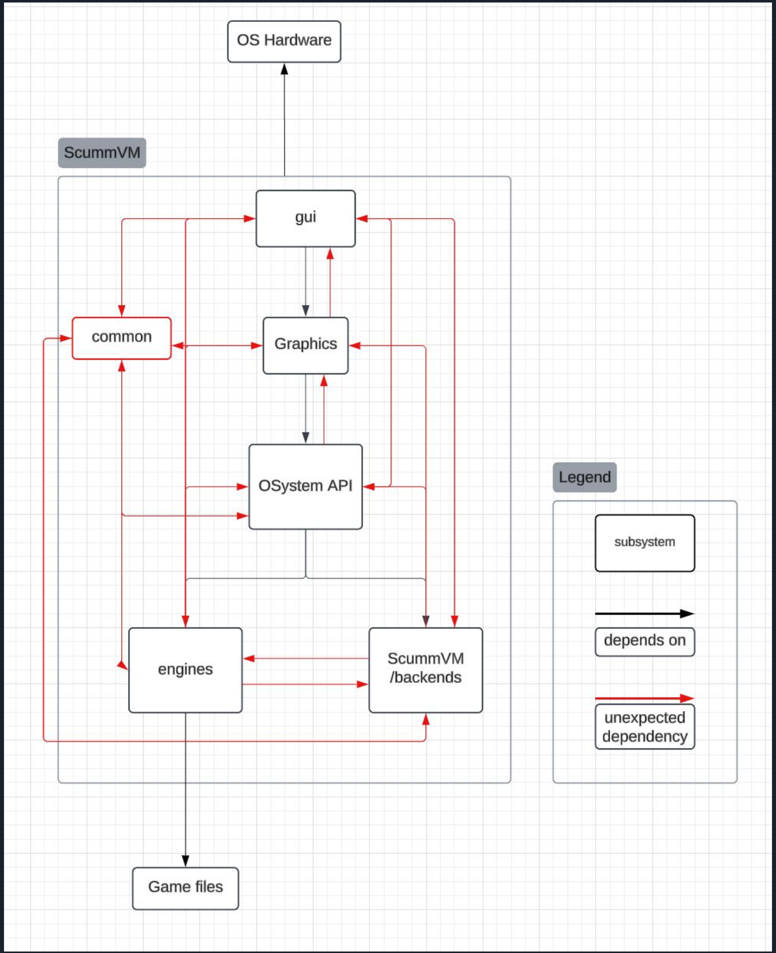
Conceptual vs. Concrete: Concrete Architecture



Conceptual vs. Concrete: Reflexion Analysis

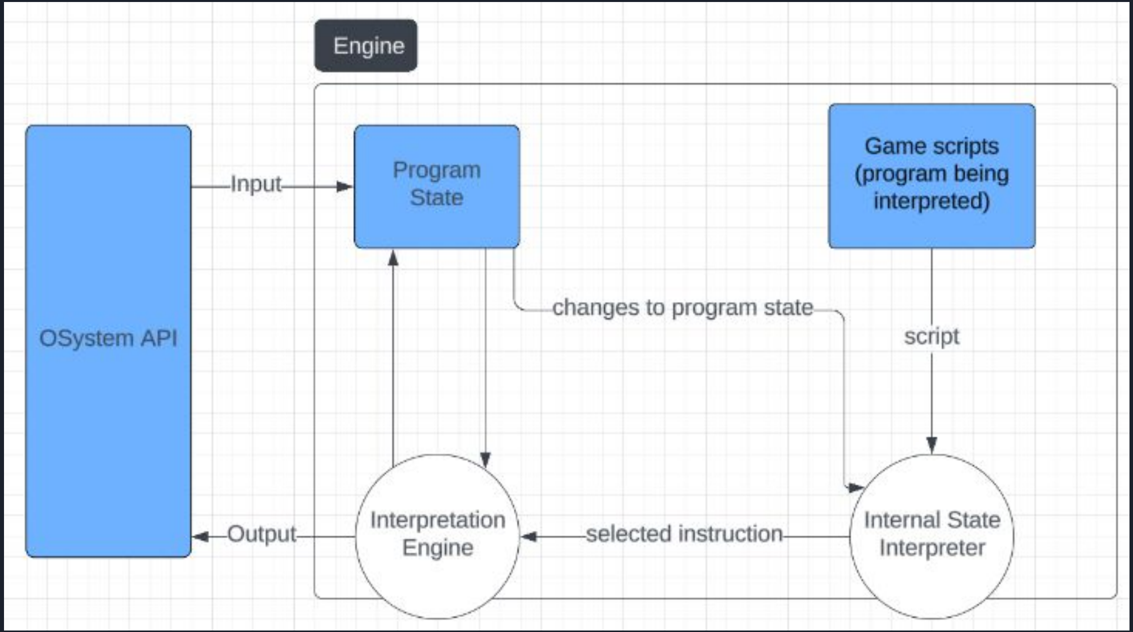
Missing Dependencies:

- Graphic -> {GUI, Engines, Backends}
- GUI -> {Engines, OSystemAPI, Backends}
- Engines -> {GUI, Backends, Graphics, OSystemAPI}
- Backends -> {Engines, OSystemAPI, Graphics, GUI}
- OSystem API -> {Graphics, GUI}

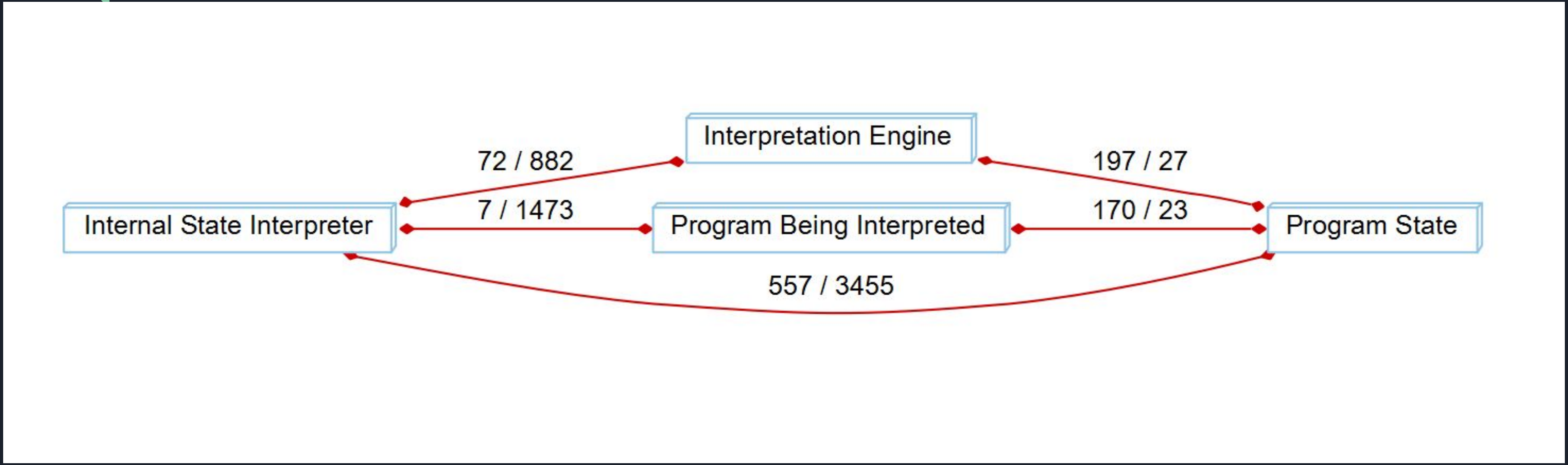


SCI Engine: Conceptual Architecture

- Scripts from the game sent to the Internal State Interpreter
- Direct input from OSystem API (preceding layer in our layered hierarchy)
- Internal State Interperter -> Interpretation Engine -> Program State



SCI Engine: Concrete Architecture

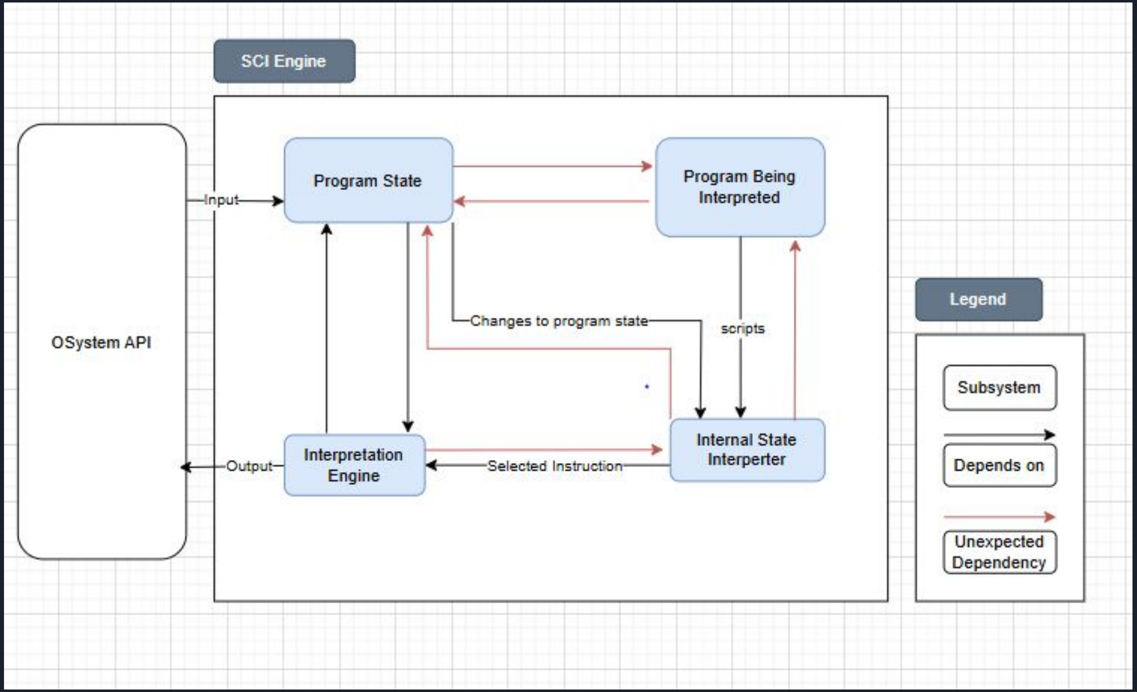


SCI Engine: Reflexion Analysis

- Every dependency is two way, some to a greater degree than others

- Strong dependency between Internal State Interpreter and Program State

- Initialization of Program State dependent upon the program being interpreted



Conclusion

Key Takeaways:

- System is much more interconnected than previously expected (communication between non-adjacent layers) - both Scumm and SCI
- ScummVM as a layered architecture: more dependencies than previously estimated
- SCI Engine as interpreter architecture: more dependencies than previously estimated
- Possible need for more clearly defined modules and encapsulation to improve clarity of system functionality

Thank you For Listening