Chrome: Conceptual Architecture

CISC/CMPE 322

Mackenzie Furlong
Alex Golinescu
David Haddad
William Melanson-O'Neill
Michael Reinhart
Lianne Zelsman

Agenda

- Architecture Overview
  - Derivation Process (Alternatives)
  - Subsystems
  - Design & Structure
  - Developer Implications
  - Tradeoffs
- User Login Use-Case
- Research Process
  - Limitations
  - Lessons Learned
  - Conclusion

Chrome

Web browser developed by Google based on the open-source Chromium project

Architecture

Subsystems

Network

Connects to internet with FTP & HTTP

Rendering Engine

Parses HTML/CSS and prepares DOM

UI

How the user interacts with the browser

User Accounts

Collects continuous data from users

Browser Engine

Represents the top-level browser window

Derivation Process

Layered Architecture Style

We considered it because:

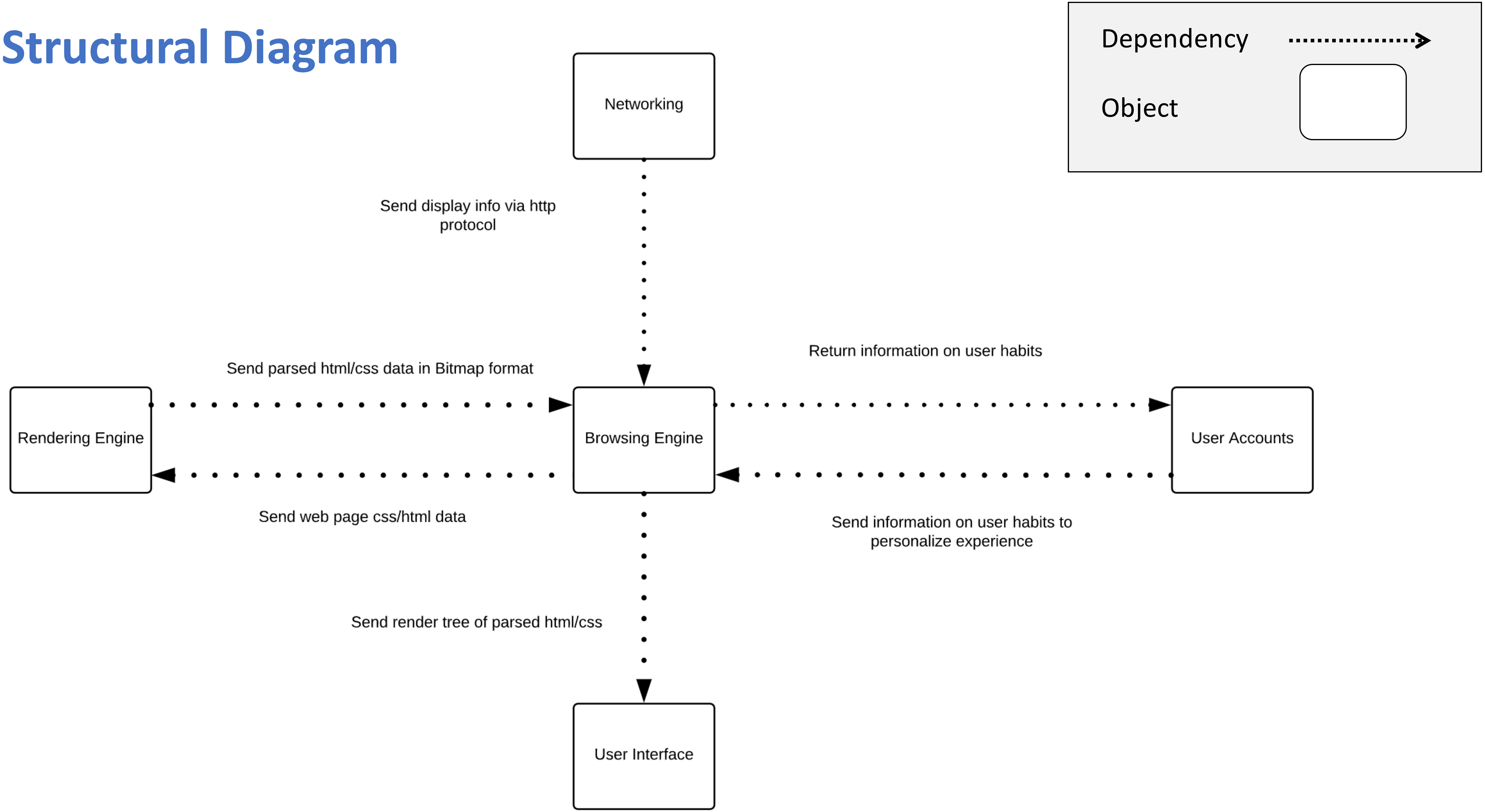
- Makes reuse easy
  - Different implementations of the same layer can be used interchangeably
- Makes evolution easy
  - Changing one layer won't affect the entire system

We decided against it because:

- Entire system could not be structured in a layered way
  - Relationships and interactions between Chrome subsystems are too complex for basic layering

Object Oriented Style

Structural Diagram



Control & Data Flow

Inter-Process Communication (IPC) – How the browser, renderer, & plugin processes communicate

Multi-Process Architecture – Many processes communicating with each other

Pipe – Main inter-process communication primitive

IPC in the Browser: Communication with the renders is done in a separate I/O thread

IPC in the Renderer: Each renderer has a thread that manages communication

Concurrency

- Each tab in Chrome runs its own instance of the rendering engine
  - Allows tabs to **operate independently** and concurrently from one another

Evolution

- Modularity of objects provides building blocks for easy **scalability** and **evolution**
- New subsystems can be added and tested individually before being implemented into the pre-existing system

Developer Implications

- Easy **work division** based on separate objects
- Easy to **test**
- Teams have **autonomy** to change internals of their subsystem without affecting other parts of the system

BUT… developers must understand the way subsystems interface with each other and how changes to one object could affect a dependent object (side effect)

Tradeoffs

- Difficult to make changes to object oriented components
  - Every dependent subsystem needs to know the state of all other subsystems
- Object Oriented design can be **less efficient** and use more CPU

Use Cases

User Log In

Sequence Diagram

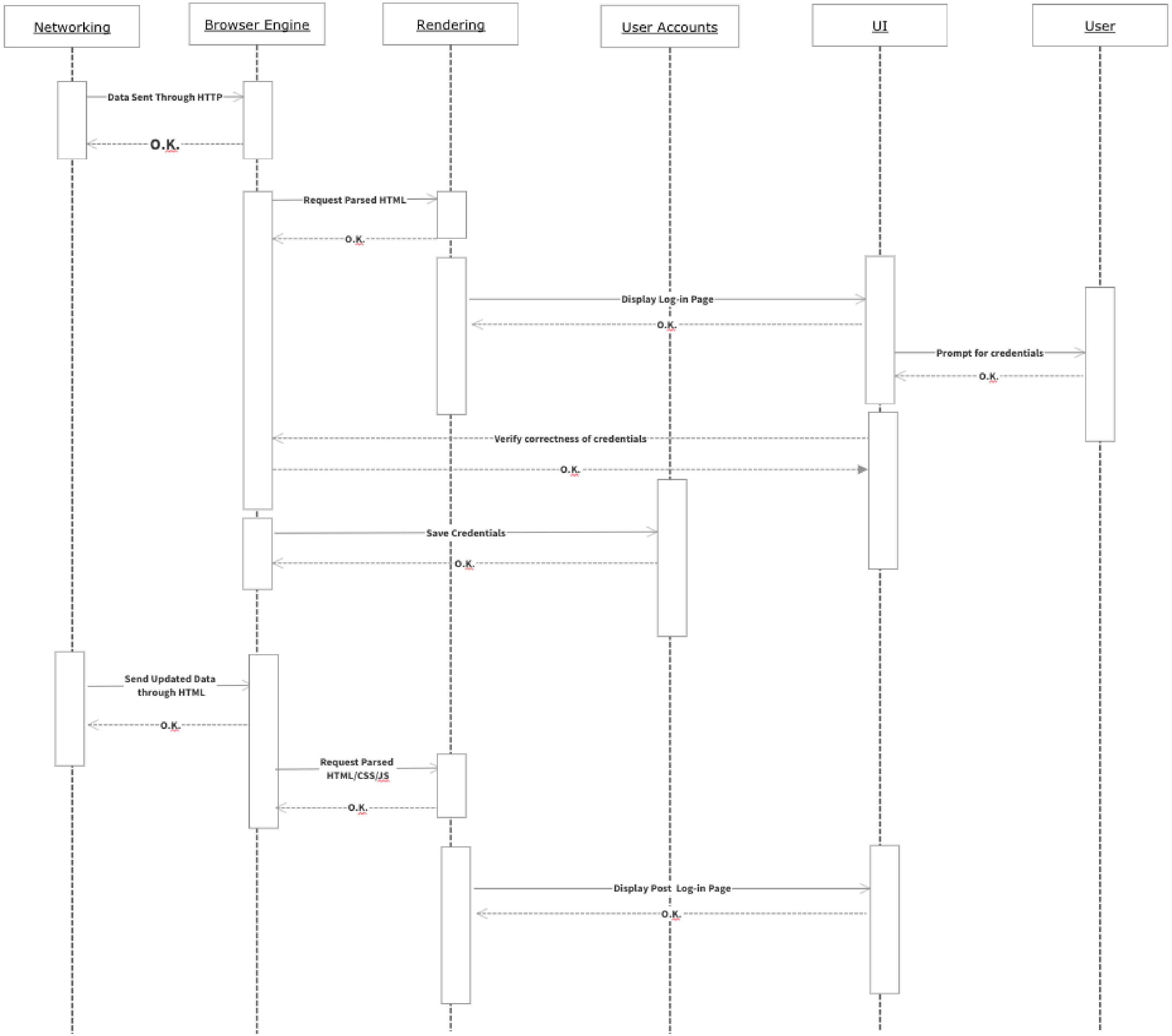
Message

Reply

Lifeline

Object

Activation



Design Process

Limitations

- Research based on **Chromium**, which has slight differences from **Chrome**
- Many sources over 5 years old
  - Information may be outdated
- Third party sources could be inaccurate

Lessons Learned

- Chrome architecture is **vast**
  - Difficult to simplify into 5 subsystems
  - Can’t deep-dive into everything
- Subsystem relationships are **complex**
- Research and analysis of low-level design will be difficult

Conclusion

- Object oriented architecture helps with complexity of system
- Despite tradeoffs, design makes development feasible
  - Easy to test
  - Easy to evolve
- Efficiency and dependency problems must be factored in carefully

Questions?