Chrome: Concrete Architecture

CISC/CMPE 322

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

Agenda

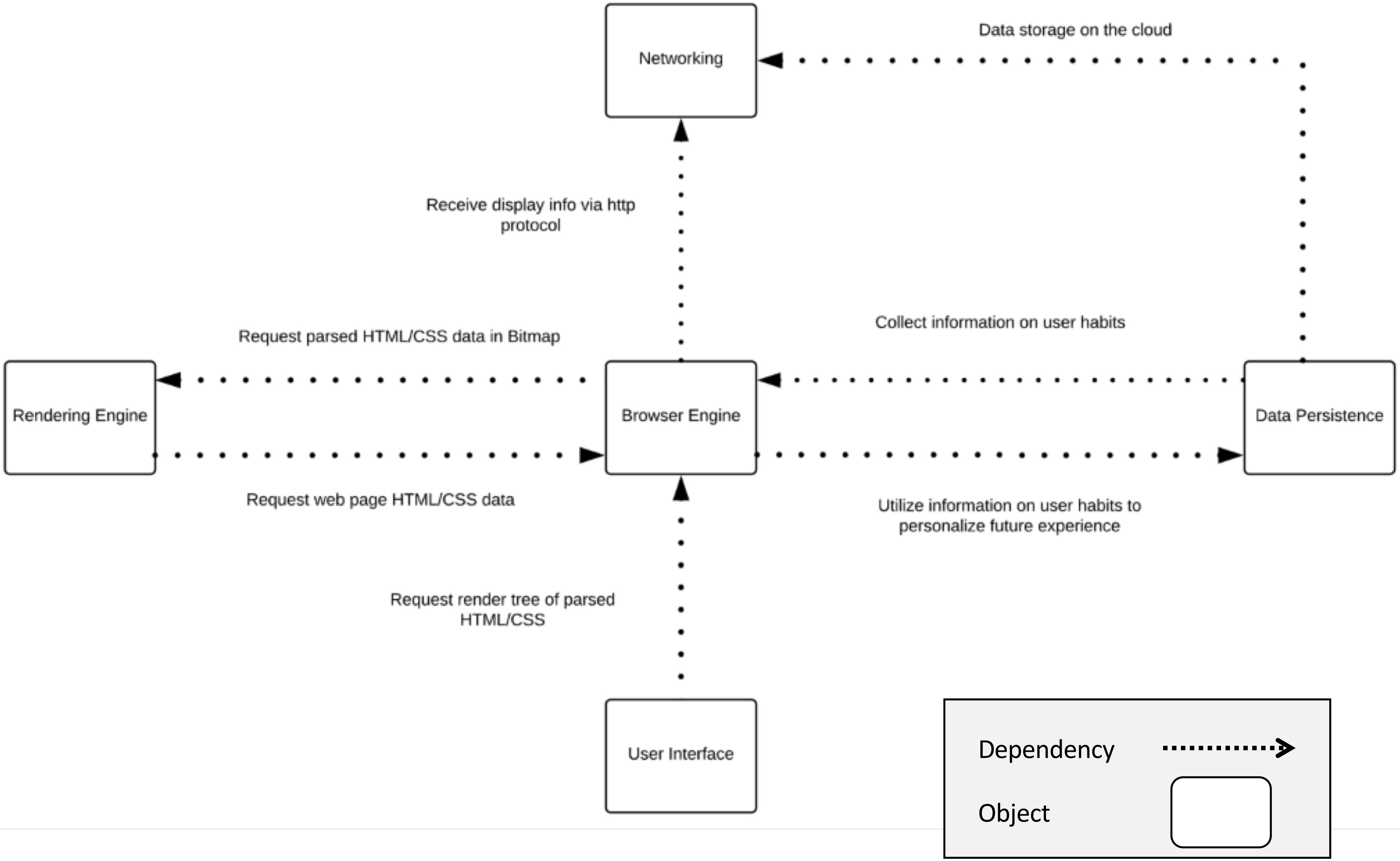
- Architecture Overview
    - Derivation Process (Alternatives)
    - Conceptual & Concrete Architecture
    - Concurrency
    - Developer Implications
- Reflexion Analysis
- User Log-In Use-Case
- Research Process
    - Limitations
    - Lessons Learned
- Proposed Enhancement (A3)

Architecture

Derivation Process

- Layered Architecture
  - Makes reuse and evolution easy
  - But, entire system could not be structured in a layered way
- Instead, used layered sub-components within object oriented system
- Understand tool used to find dependencies

Conceptual Architecture



Networking

Data storage on the cloud

Receive display info via http protocol

Request parsed HTML/CSS data in Bitmap

Collect information on user habits

Rendering Engine

Browser Engine

Data Persistence

Request web page HTML/CSS data

Utilize information on user habits to personalize future experience

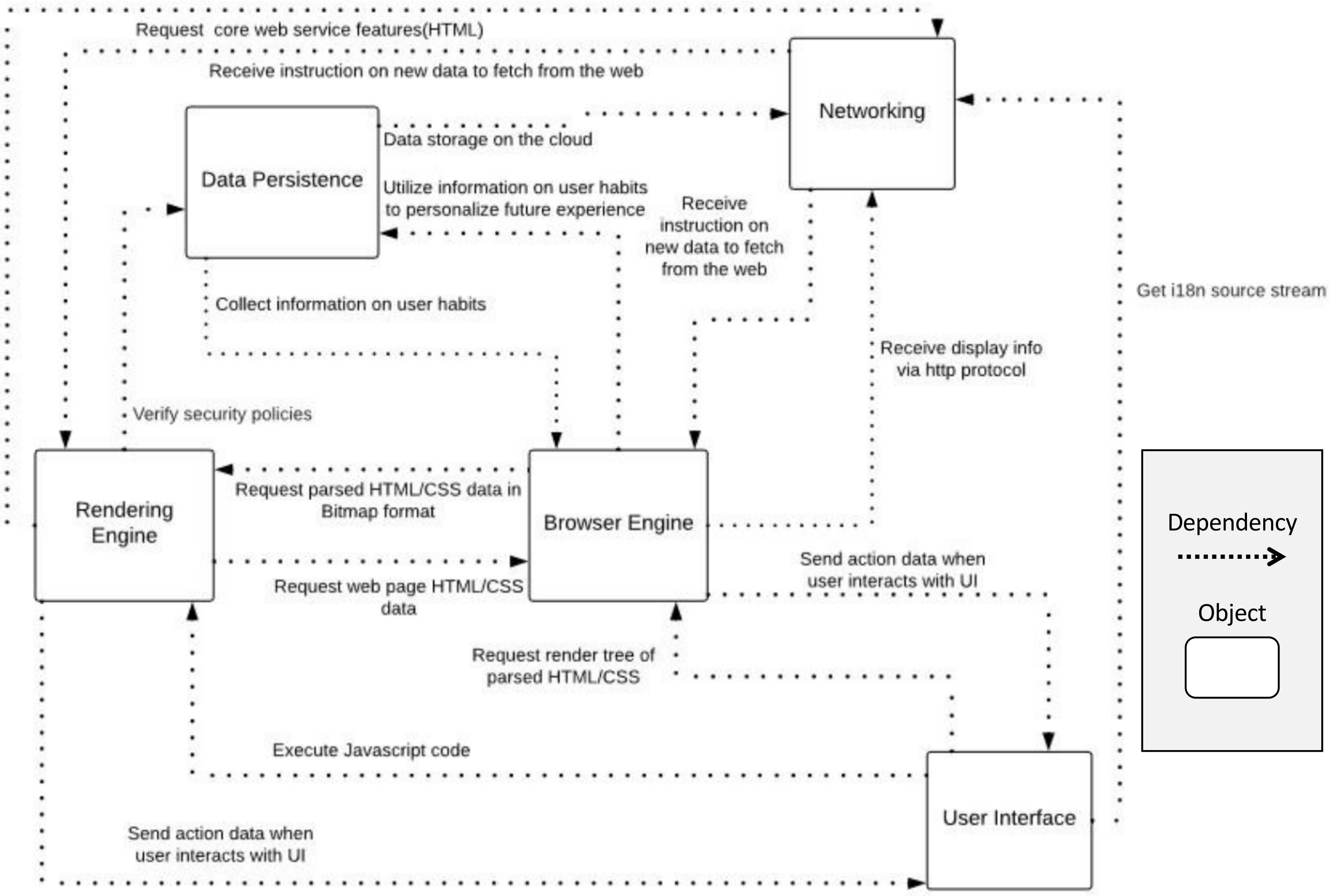
Request render tree of parsed HTML/CSS

User Interface

Dependency

Object

Concrete Architecture



Request core web service features(HTML)
Receive instruction on new data to fetch from the web
Networking
Data Persistence
Data storage on the cloud
Utilize information on user habits to personalize future experience
Receive instruction on new data to fetch from the web
Collect information on user habits
Get i18n source stream
Receive display info via http protocol
Verify security policies
Rendering Engine
Request parsed HTML/CSS data in Bitmap format
Browser Engine
Request web page HTML/CSS data
Send action data when user interacts with UI
Dependency
Object
Request render tree of parsed HTML/CSS
Execute Javascript code
User Interface
Send action data when user interacts with UI

Subsystems

Networking

Connects to internet with FTP & HTTP

Rendering Engine

Parses HTML/CSS and prepares DOM

UI

How the user interacts with the browser

Data Persistence

Collects continuous data from users

Browser Engine

Represents the top-level browser window

Concurrency

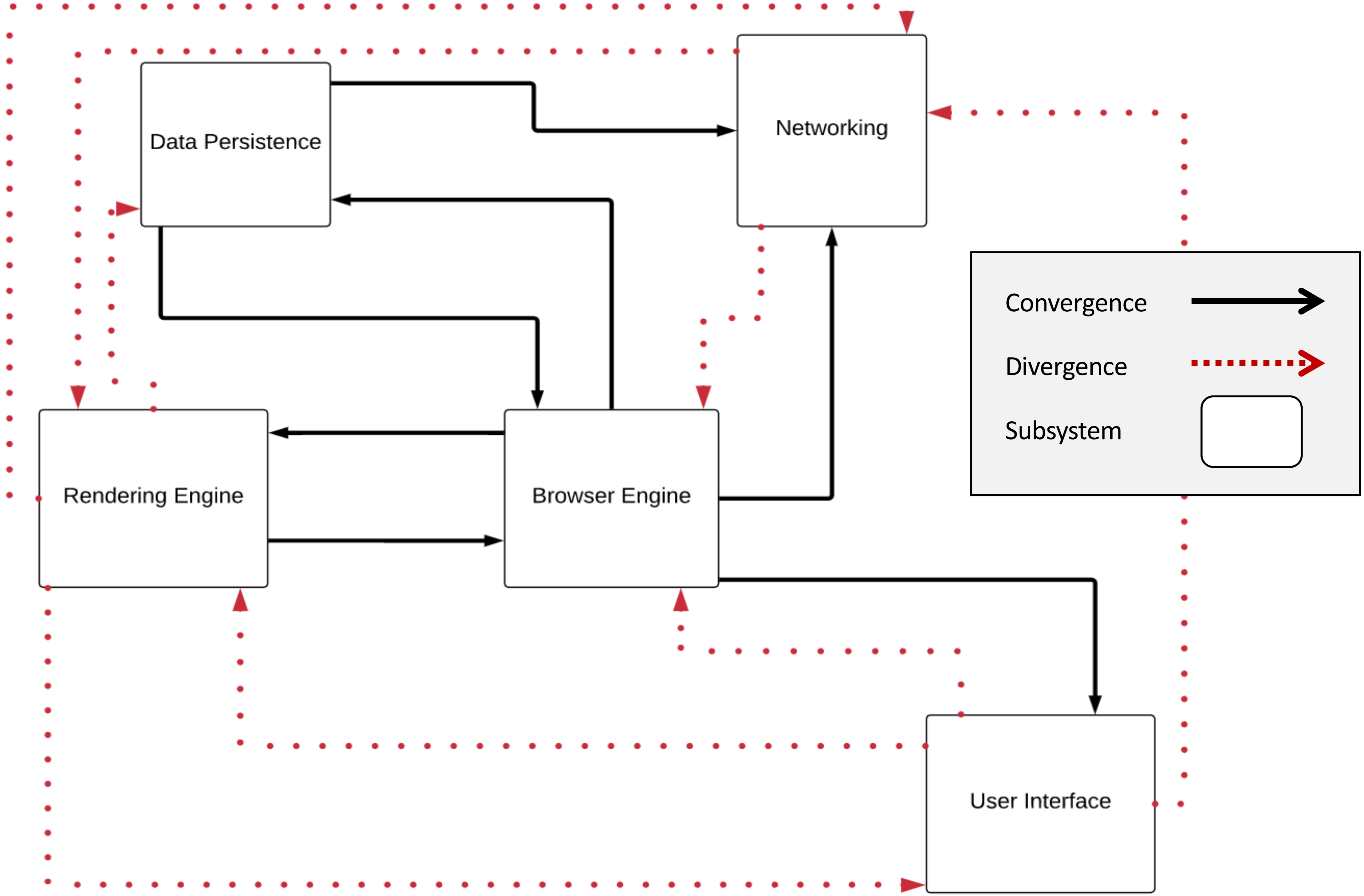
- Each tab runs **its own instance** of the rendering engine
- Allows tabs to operate **independently** and **concurrently** from one another

Developer Implications

- Easy **work division, testability & evolution**
- Teams have autonomy
- Developers must understand subsystem dependencies

Reflexion Analysis

Reflexion Model



UI

↓

Networking

- Originally assumed communication would go through the Browser Engine
- Dependency needed for window management and multi-language support in the UI

Rendering Engine

↓

Data Persistence

- Originally assumed communication would go through the Browser Engine
- Dependency needed for Rendering Engine to verify security policies

Browser Engine

↓

UI

- Originally thought UI would collect information from Browser Engine without need for back & forth communication
- Dependency needed to allow Browser Engine to adjust backend behaviour based on user interaction with UI

Rendering Engine

↑

↓

Networking

- Originally assumed all communication between Rendering Engine and Network would go through the Browser Engine
- Dependency needed to directly receive core web service features (HTML, etc)
  - Then passed to browsing engine to apply changes to the page based on user’s config

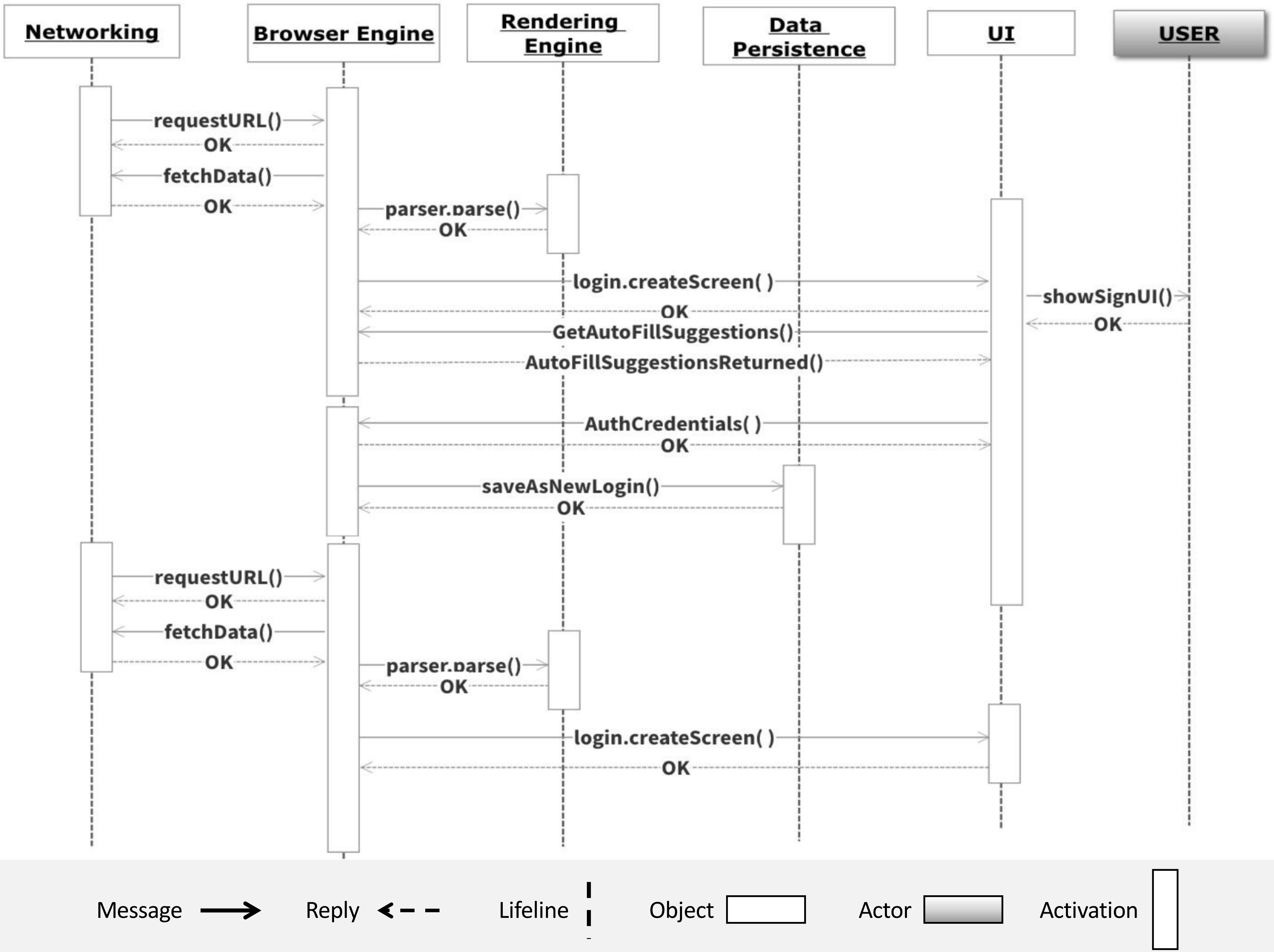
Rendering Engine

↑
↓

UI

- Originally assumed all communication between Rendering Engine and UI would go through the Browser Engine
- Dependency needed for non-static content
  - e.g. animations, different layers and third-party applications

Use Cases – User Log-In


Networking
Browser Engine
Rendering Engine
Data Persistence
UI
USER
requestURL()
OK
fetchData()
OK
parser.parse()
OK
login.createScreen( )
showSignUI()
OK
OK
GetAutoFillSuggestions()
AutoFillSuggestionsReturned()
AuthCredentials( )
OK
saveAsNewLogin()
OK
requestURL()
OK
fetchData()
OK
parser.parse()
OK
login.createScreen( )
OK
Message
Reply
Lifeline
Object
Actor
Activation

Design Process

Limitations

- Lack of documentation on certain subsets of code
- Difficulty finding correct files from source code
- Understand tool limitations
- Time restraints

Lessons Learned

- Importance of understanding dependencies
- Large code-base restrictions
- Work division difficulties
- Time management

Additional Feature (A3)

- **Facial recognition** used to
  1. Protect autofill data
  2. Bypass Chrome-prompted login requests
- Why?
  - Improve security
  - Increase speed
- How?
  - Build off of pre-existing autofill functionality
  - Use Data Persistence object as storage & Browser Engine to interface with UI

Conclusion

- Gaps in conceptual architecture
- Dependency complexity makes development more difficult
- Multiple iterations of design process required
- More research needed for A3 feature proposal

Questions?