CptS 487
Software Design and Architecture

Lesson 16

Architectural Pattern Part 1:

Multi-layered and MVC

WASHINGTON STATE UNIVERSITY

World Class. Face to Face.

Instructor: Bolong Zeng

Overview

- Multi-layered architecture
- Model-View-Controller architecture (MVC)
  - Discussing w.r.t Lesson 6: 10 Design Principles

Key Concepts Analogy

- Subsystem
  - Mapped to: Media processor; Game console; AV output; Remote controls; etc.
- Component
  - Mapped to: each piece of hardware
- Service/Interface
  - Mapped to: functionalities/ports and cables (including wireless connections)

- There is one more piece of furniture:
  - The cabinet/TV stand

Architectural Patterns

- The notion of patterns can be applied to software architecture.
  - These are called *architectural patterns* or *architectural styles*.
  - Each allows you to design flexible systems using components
    - The components are as independent of each other as possible.

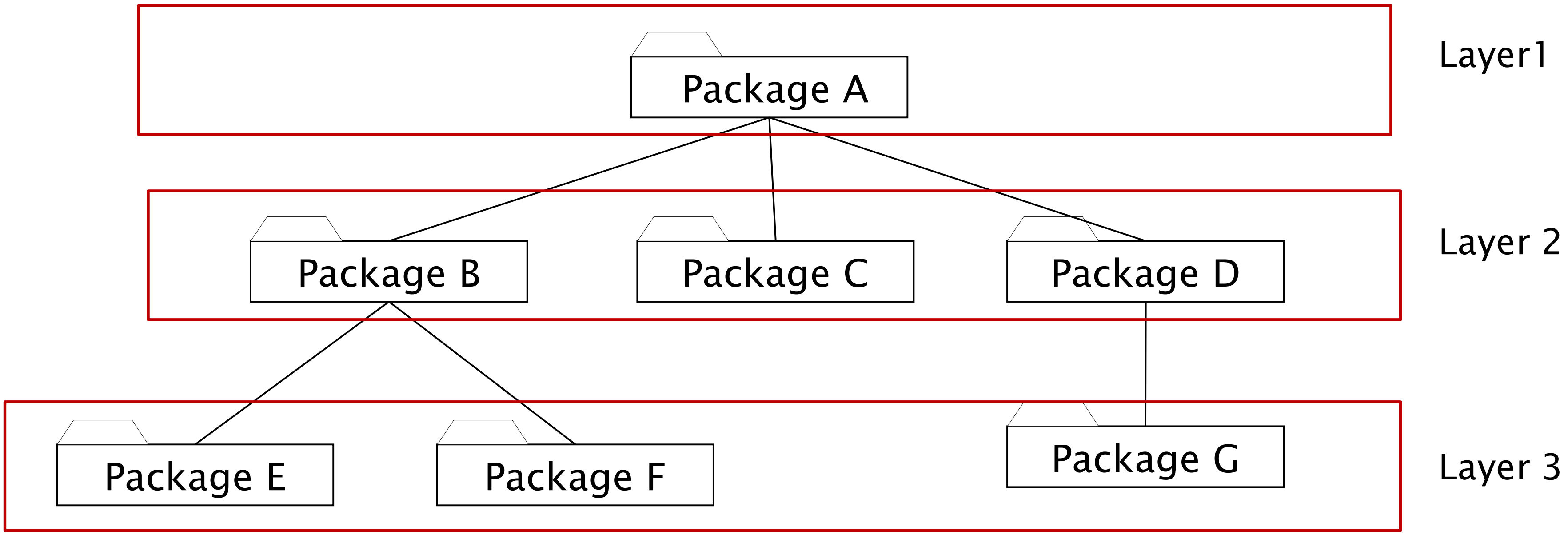
Side Note

- Architectual Pattern vs. Design Pattern
  - Architectural Pattern addresses broad and overall structures of the whole software.
  - Design Pattern addresses smaller scale part of the system, such as how classes collaborate to solve a specific problem.
  - Certain Architectural Patterns and Design Patterns are highly compatible with each other.

  - Be careful of which one is asked in a question!
    - Architecture: Multi-layer, MVC, Peer-to-Peer, Repository, Pipe-Filter etc.
    - Design: Factory, Composite, Decorator, Facade, Flyweight etc.

The Multi-Layer Architectural Pattern

- Hierarchical decomposition of the system as an ordered set of layers.
- A layer is a subsystem that provides services to another subsystem:
  - A layer only depends on services from lower layers
  - A layer has no knowledge of higher layers



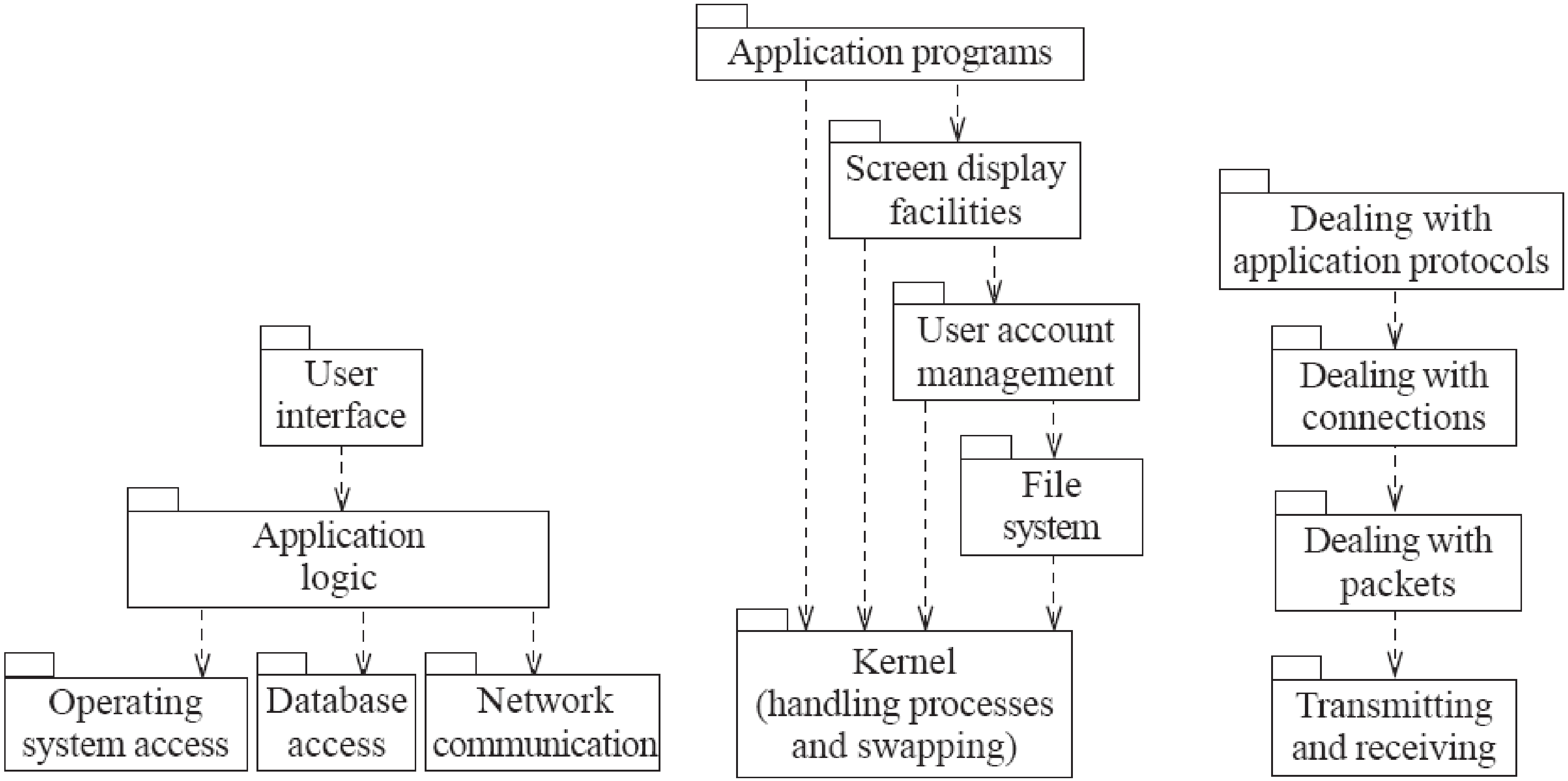
Layer Cohesion

- All the *facilities* for providing a set of related services are kept together, and everything else is kept out
  - The layers should form a hierarchy
    - Higher layers can access services of lower layers,
    - Lower layers do not access higher layers
  - The set of procedures through which a layer provides its services is the application programming interface (API)
  - You can replace a layer without having any impact on the other layers
    - You just replicate the interface (or API) for it

The Multi-Layer Architectural Pattern

- Each layer has a well-defined interface used by the layers above.
  - The higher layers see the lower layers as a set of *services*.
- A complex system can be built by superposing layers at increasing levels of abstraction.
  - It is important to have a separate layer for the UI.
  - Layers immediately below the UI layer provide the application functions determined by the use-cases.
  - Bottom layers provide general services.
    - e.g. network communication, database access

Example of Multi-layer Systems



User interface

Application logic

Operating system access

Database access

Network communication

Application programs

Screen display facilities

User account management

File system

Kernel (handling processes and swapping)

Dealing with application protocols

Dealing with connections

Dealing with packets

Transmitting and receiving

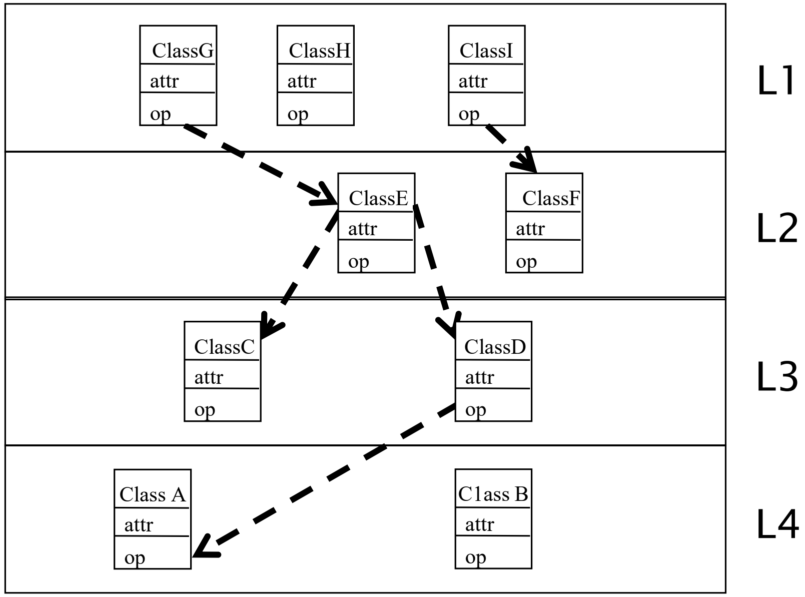
(a) Typical layers in an application program

(b) Typical layers in an operating system

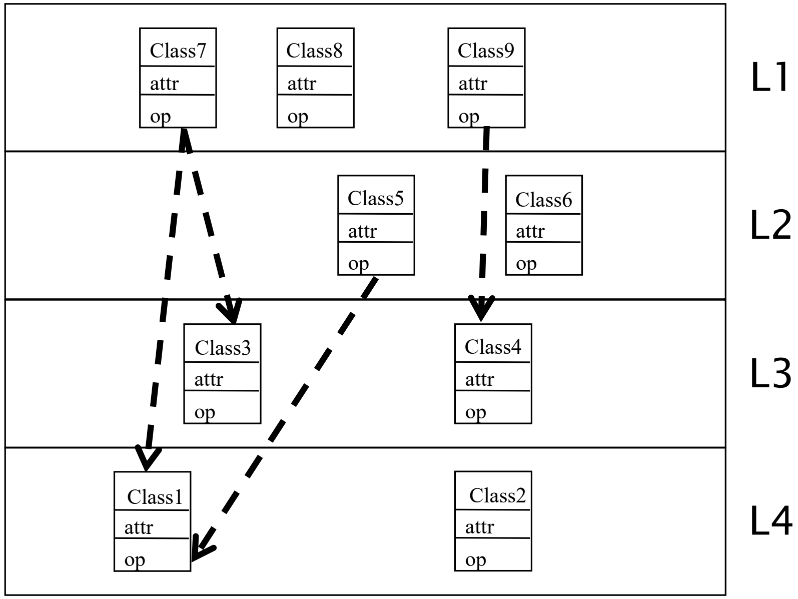
(c) Simplified view of layers in a communication system

Open vs Closed Layered Architecture

- Closed architecture: each layer communicates only with the layer immediately below it.
  - Design Goals: Maintainability, flexibility
- Open architecture: a layer can also access layers at deeper levels.
  - Design Goal: Runtime efficiency



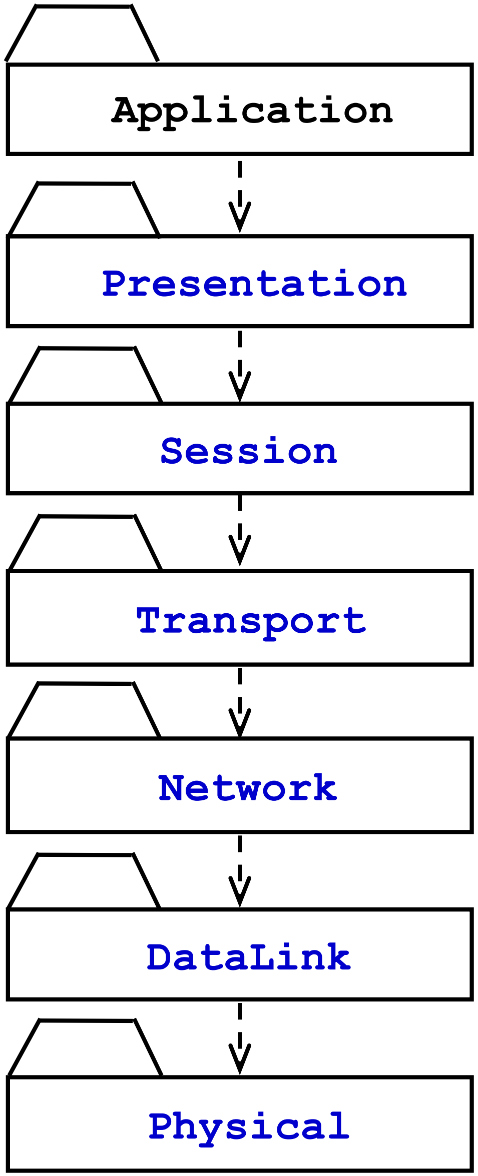
Closed Layered Architecture



Open Layered Architecture

Example of Closed Layered Architecture: OSI Model Layers and their Services

- Presentation Layer
  - Services: data transformation (encryption, byte swapping)
- Session Layer
  - Services: Initializing and authenticating a connection
- Transport layer
  - Services: Transmitting messages
    - Used by Unix programmers who transmit messages over TCP/IP sockets
- Network layer
  - Services: Transmit and route data within the network
- Datalink layer
  - Services: Transmit data frames without error
- Physical layer
  - Services: Transmit bits over communication channel



The Multi-layer Architecture and Design Principles

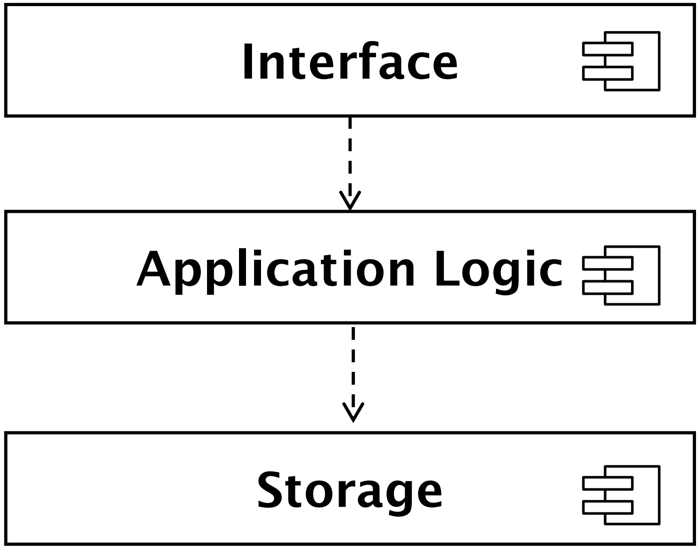
1. Divide and conquer: The layers can be designed independently.
2. Increase cohesion: Well-designed layers have layer cohesion.
3. Reduce coupling: Well-designed lower layers do not know about the higher layers and the only connection between layers is through the API.
4. Increase abstraction: You do not need to know the details of how the lower layers are implemented.
5. Increase reusability: The lower layers can often be designed generically.

The Multi-layer Architecture and Design Principles

6. Increase reuse: You can often reuse layers built by others that provide the services you need.
7. Increase flexibility: you can add new facilities built on lower-level services, or replace higher-level layers.
8. Anticipate obsolescence: By isolating components in separate layers, the system becomes more resistant to obsolescence.
9. Design for portability: All the dependent facilities can be isolated in one of the lower layers.
10. Design for testability: Layers can be integrated and tested incrementally

Three-Tier Architectural Pattern

- An application consists of 3 hierarchically ordered subsystems
  - Interface layer: user interface
  - Application logic layer: middleware
  - Storage layer: database system



Boundary objects: windows, forms, web pages, etc.

Control and entity objects: processing, rule checking, notifications

Storage, retrieval and query of persistent objects

Usually the 3 layers are allocated on 3 separate hardware nodes

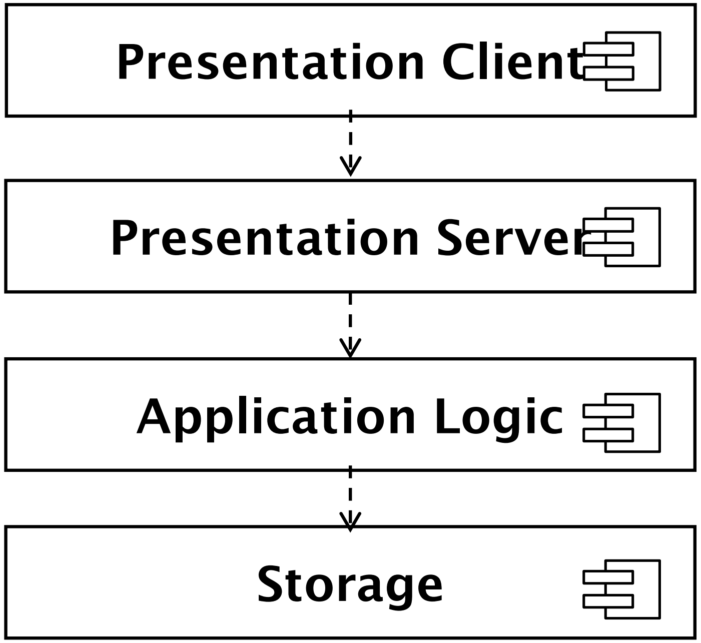
Example of a Three-tier Architectural Style

- Three-tier Architectural pattern is often used for the development of Websites:

1. The Web Browser implements the user interface
2. The Web Server serves requests from the web browser
3. The Database manages and provides access to the persistent data.

Four-Tier Architectural Pattern

- An application consists of 4 hierarchically ordered subsystems
  - Interface layer is decomposed into “Presentation Client” and “Presentation Server”
    - Presentation Client: client interface
    - Presentation Server: presentation objects serving requests from client



Web browser, application client; located on user machines

Forms; located on one or more servers

Enables more variability on the user interface style

Example of a 4-tier Architectural Style

- 4-Layer-architectural styles are usually used for the development of e-commerce sites. The layers are:

1. The Web Browser, providing the user interface
2. A Web Server, serving static HTML requests
3. An Application Server, providing session management (for example the contents of an electronic shopping cart) and processing of dynamic HTML requests
4. A back end Database, that manages and provides access to the persistent data
    - In commercially available 4-tier architectures, this is usually a relational database management system (RDBMS).

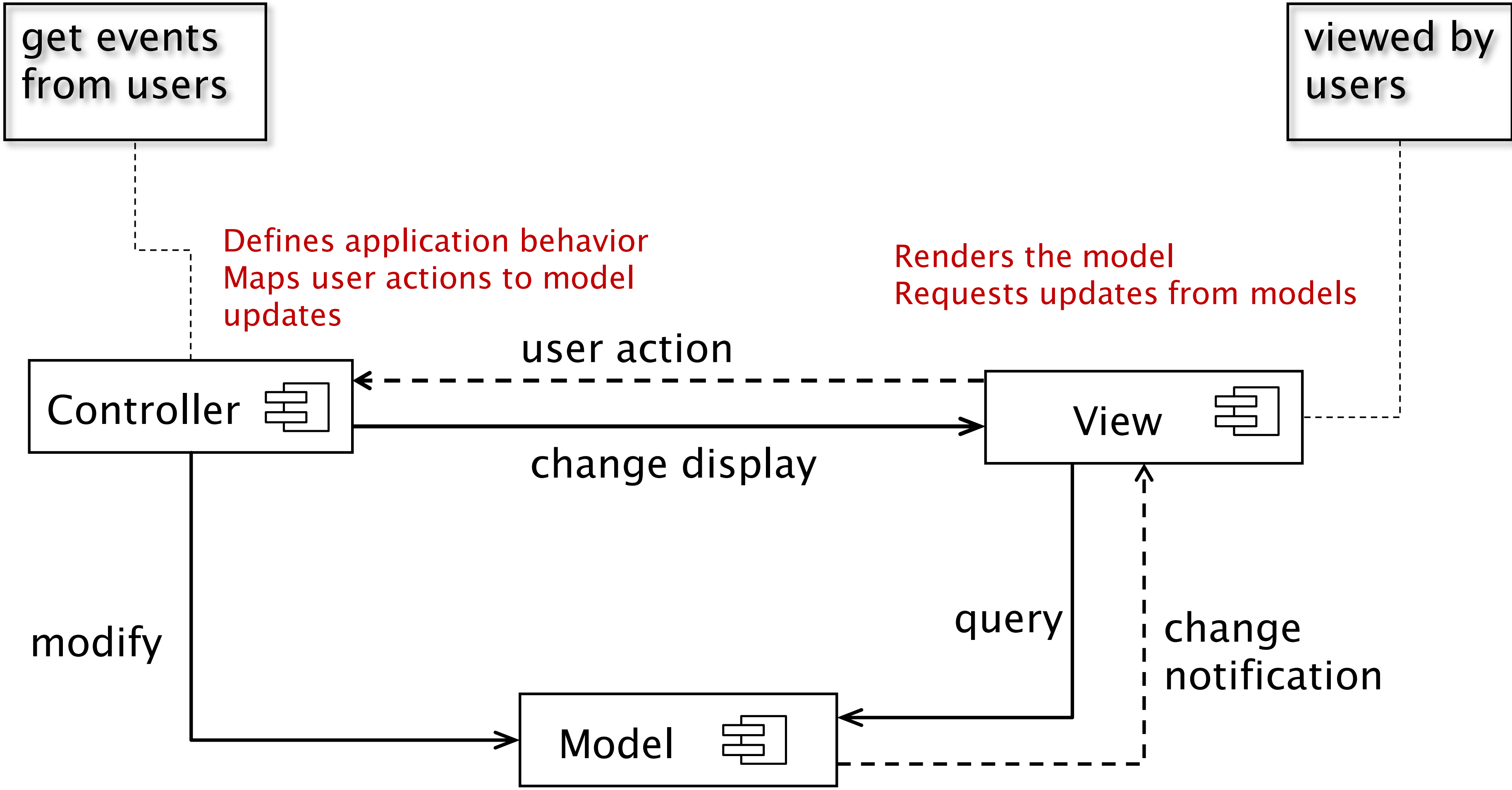
Model-View-Controller (MVC) Architectural Pattern

- Problem: In systems with high coupling changes to the user interface (boundary objects) often force changes to the entity objects (data)
  - The user interface cannot be re-implemented without changing the representation of the entity objects
  - The entity objects cannot be reorganized without changing the user interface
- Solution: Decoupling! The model-view-controller architectural pattern decouples data access (entity objects) and data presentation (boundary objects)

Model-View-Controller (MVC) Architectural Pattern

- An architectural pattern used to help separate the user interface from other parts of the system.
  - *Model* :
    - The data
    - Methods for accessing and modifying data
  - *View* :
    - Render the appearance of the data from the model in the user interface
    - When model changes, view must be updated
  - *Controller :*
    - Translates user actions (ie interactions with view) into operations on the model
    - Example user actions: button clicks, menu selections
- Rationale: user interfaces change more frequently than the system's data.

Example of the MVC architecture



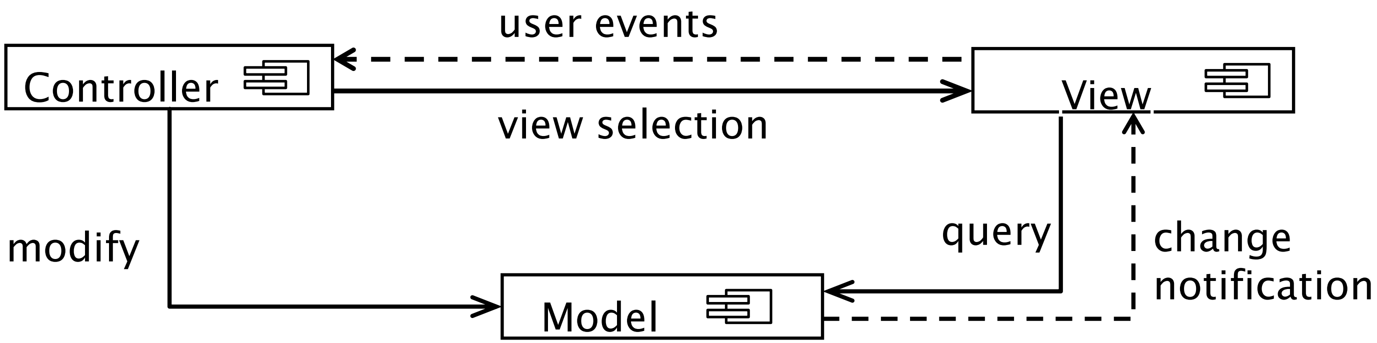
Defines application behavior
Maps user actions to model updates

Renders the model
Requests updates from models

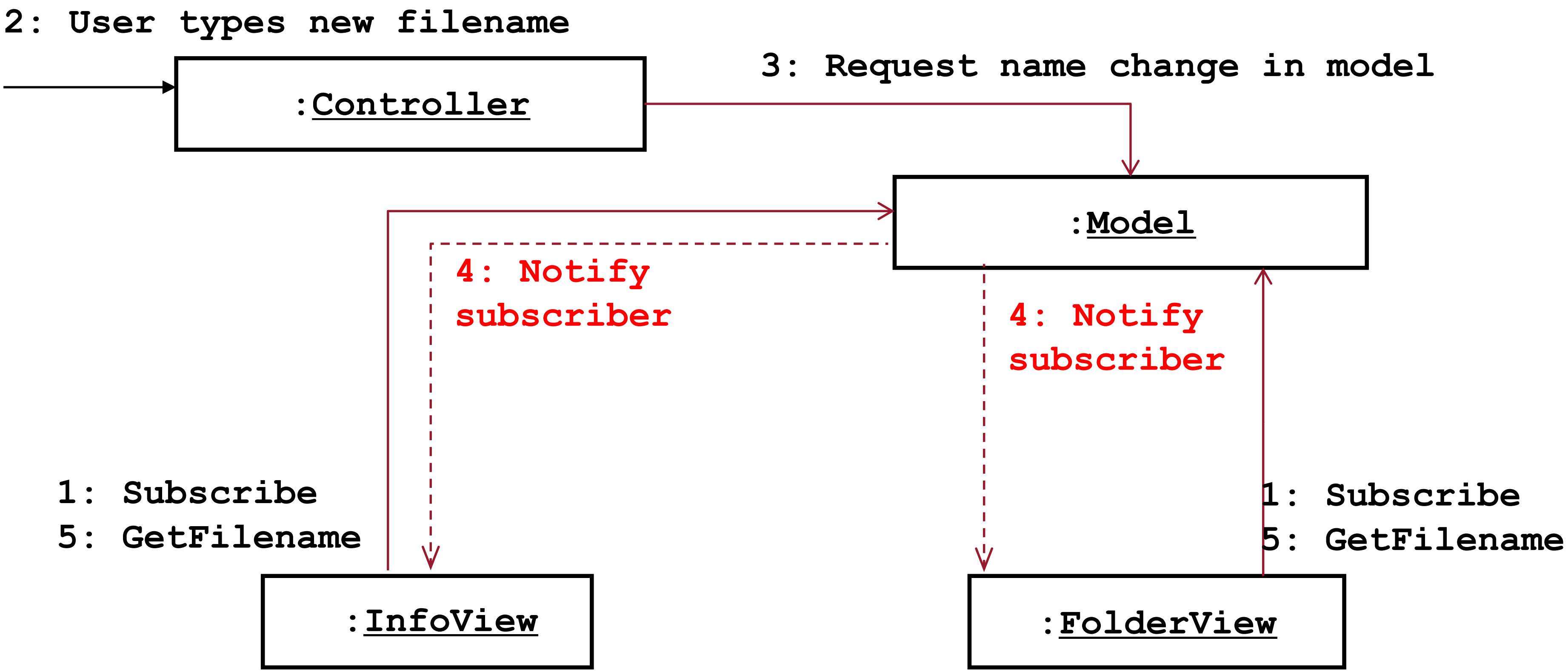
Represents application data
Responds to queries
Notifies view of changes
Allows the controller to access application functionality

Example of MVC in Web Architecture

- The *View* component generates the HTML code to be displayed by the browser.
- The *Controller* is the component that interprets ‘HTTP post’ transmissions coming back from the browser.
- The *Model* is the underlying system that manages the information.



Example: Modeling the Sequence of Events in MVC



Sequence of events in MVC (UML Communication Diagram)

- The subscription/notification functionality usually realized with Observer Design pattern

The MVC Architecture and Design principles

1. Divide and conquer: The three components can be somewhat independently designed.

3. Reduce coupling: The communication channels between the three components are minimal.

6. Increase reuse: The view and controller normally make extensive use of reusable components for various kinds of UI controls.

7. Design for flexibility: It is usually quite easy to change the UI by changing the view, the controller, or both.

10. Design for testability: You can test the application separately from the UI.