UECS2344 Software Design: Lecture 10

Software Architecture

Software Architecture

- looks at big picture – high-level view of system
- represents the structure or organisation of a system in terms of subsystems, their interrelationships, and interactions among them

Architectural Design

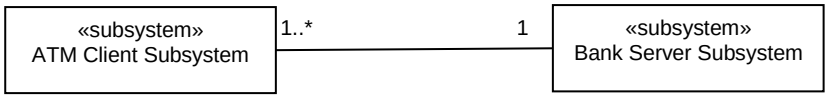
- high-level design
- decompose system into subsystems
- *separation of concerns* - each subsystem is relatively self-contained and performs a part of the overall system functionality, as independently as possible from the functionality provided by other subsystems

Advantages of decomposing system into subsystems:

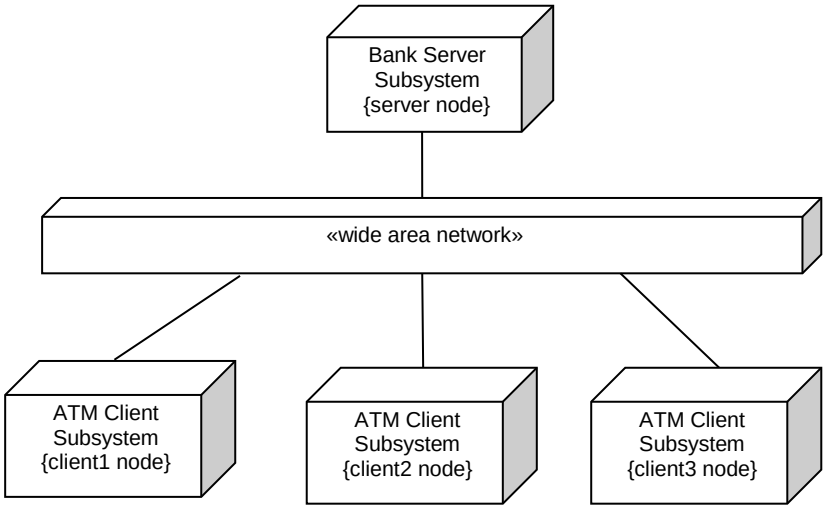
- it produces smaller units of development
- it helps to maximise reuse at the subsystem level
- it helps developers to cope with complexity
- it improves maintainability
- it aids portability

Possible Ways to Decompose System into Subsystems

- group together elements of the system that share some common properties into one subsystem
- in general, elements that are highly coupled should be placed together in one subsystem so that the subsystem is highly cohesive
- in object-oriented approach - classes that are functionally related could be placed in the same subsystem
    - example:
        - the elements of one subsystem may all focus on functional requirements or use cases
        - the elements of a second subsystem may all deal with the user interface
        - the elements of a third subsystem may all deal with database management
- a subsystem can also be determined based on physical distribution
    - example: client and server software functionality
        - client software communicates with server software to request resources or services
        - server software manages resources or provides services and responds to client software requests
        - client and server software responsibilities allocated to different subsystems: client subsystem and server subsystem
            - this is because client and server software functionalities are performed on 2 different computers – the client computer and the server machine
            - Example: ATM Client subsystem and Bank Server subsystem



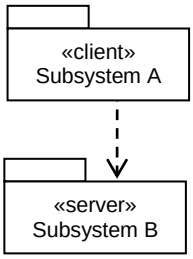
Physical Architecture of System (Implementation) can be shown in D**eployment Diagram**



Subsystem Communication Styles

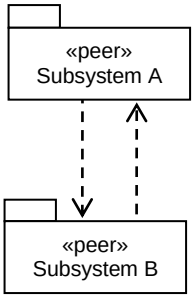
A. Client-Server

- a server subsystem provides services for multiple clients
- a client subsystem request services from a server i.e. client is dependent on the server



B. Peer-to-Peer

- a subsystem requests services from other subsystems
- it also provides services to other subsystems



Software Architectural Patterns

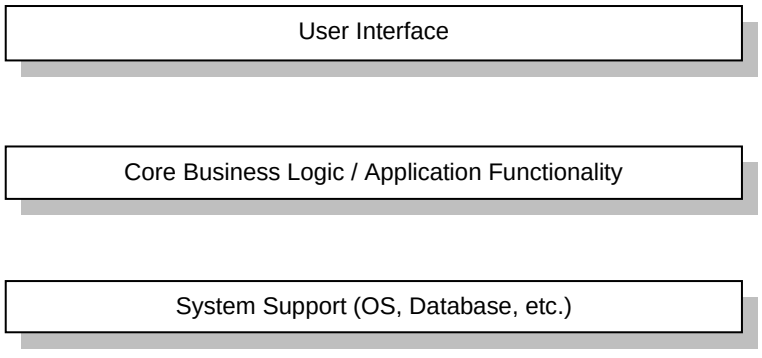
The architecture of a software system may be based on a particular architectural pattern or style. An architectural pattern captures the main aspects of a certain type of system. It provides a skeleton or template for the software architecture of that type of system.

You can think of an architectural pattern as an abstract description of good practice, which has been tested in different systems. So, an architectural pattern describes a software architecture that has been successfully used in previous systems.

A. Layered Architectural Pattern

Name	**Layered Architecture**
Description	Organises the system into layers with each layer containing components dealing with related functionality. A layer provides services to the layers above it so that the lowest-level layers represent core services that are likely to be used throughout the system.
Advantages	Allows replacement of entire layer as long as the communication requirements between the layers remain unchanged.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact with lower-level layers instead of through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

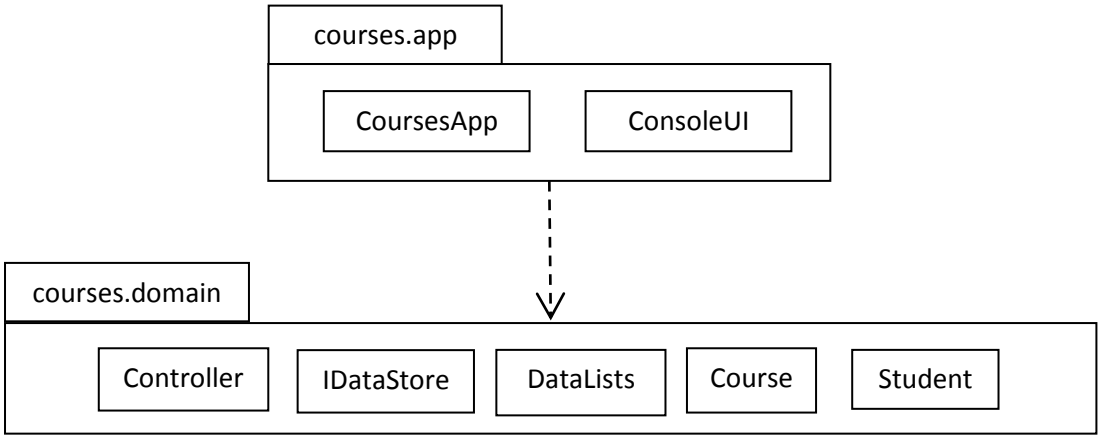
Generic Diagram



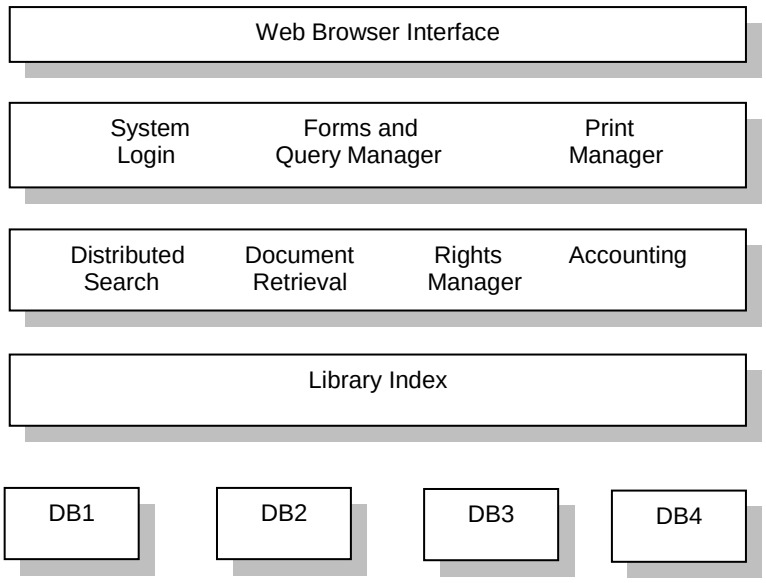
Examples of Applications using Layered Architectural Pattern

1. Example: Course Management System (*2 layers - from Practical 6*)

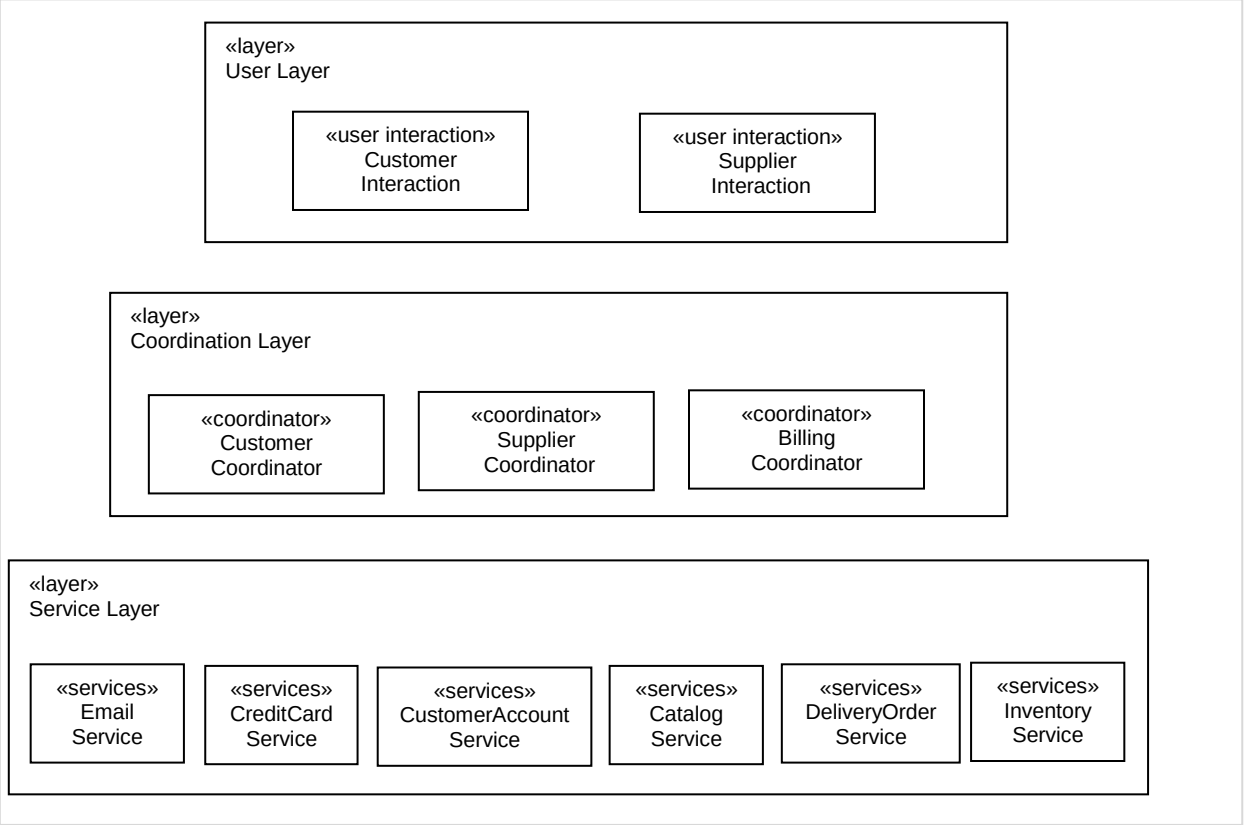
Package Diagram



2. Example: System for sharing copyright documents held in different libraries

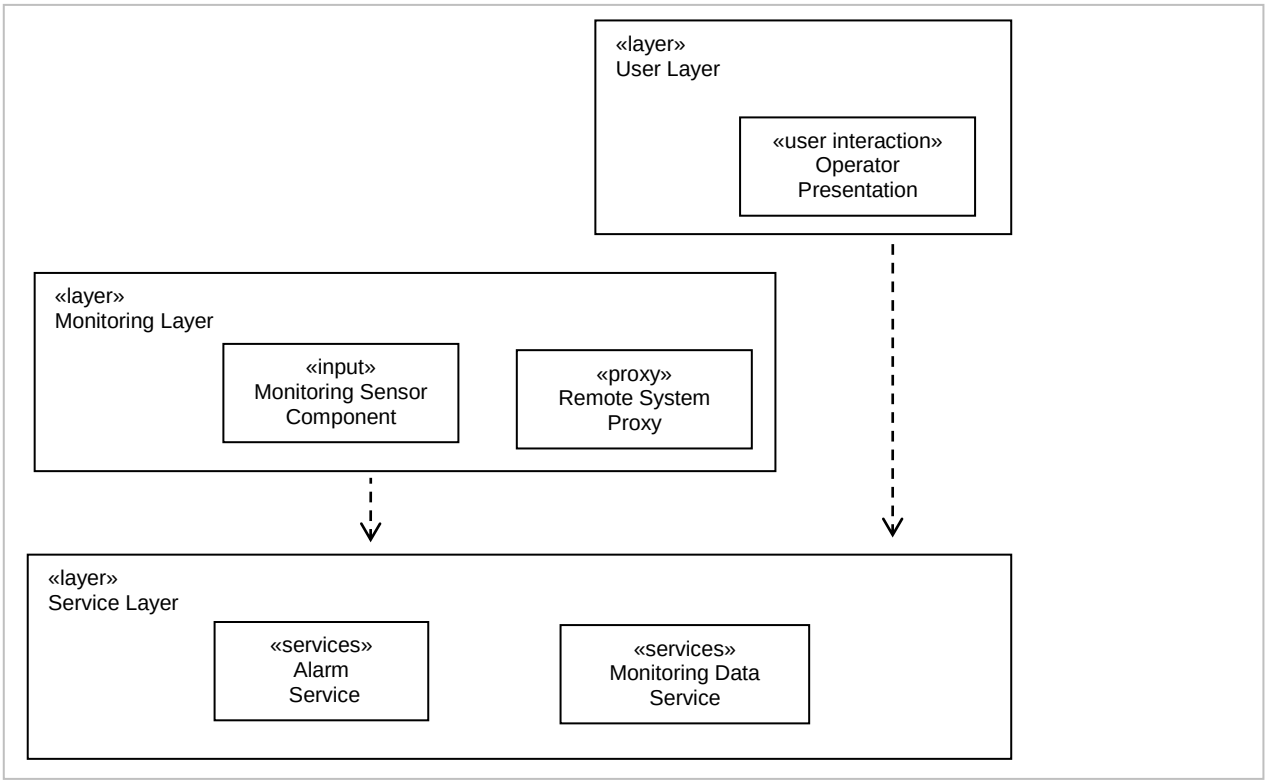


3. Example: Order processing system

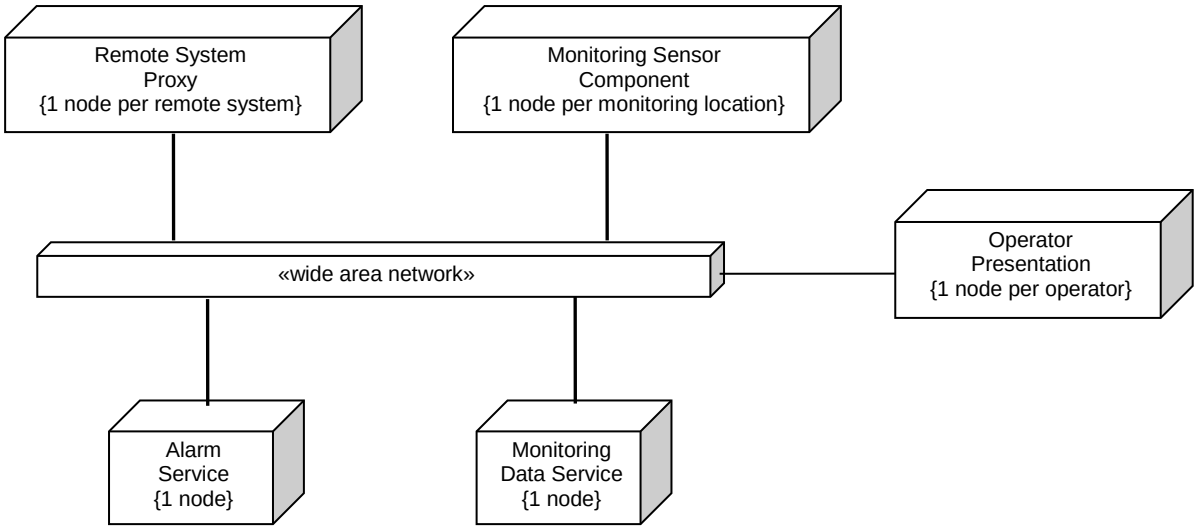


4. Example: Emergency monitoring system

Software Architecture View



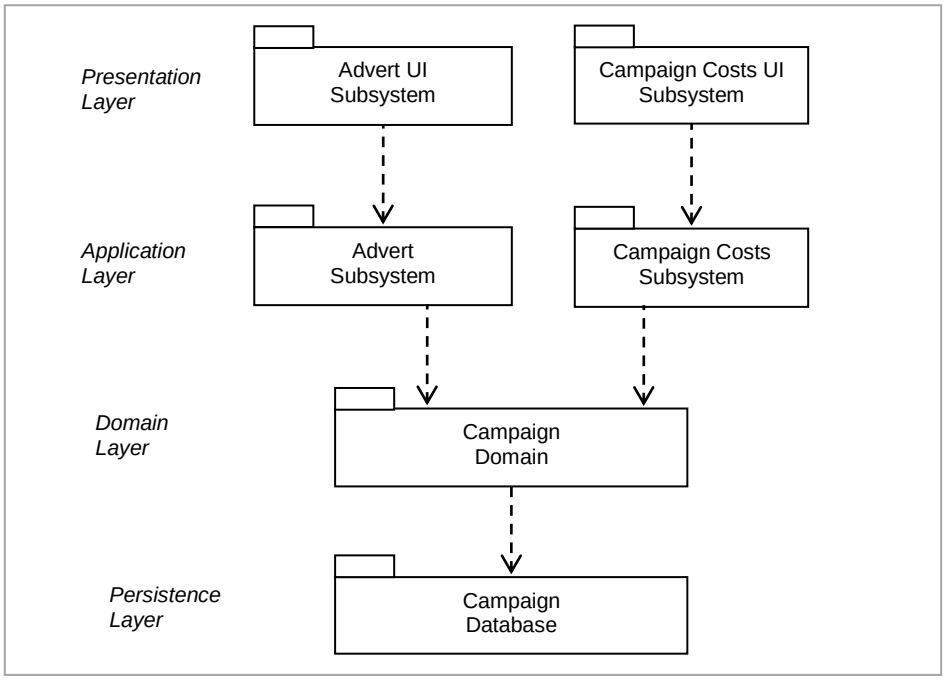
Physical Architecture View (**Deployment Diagram**)



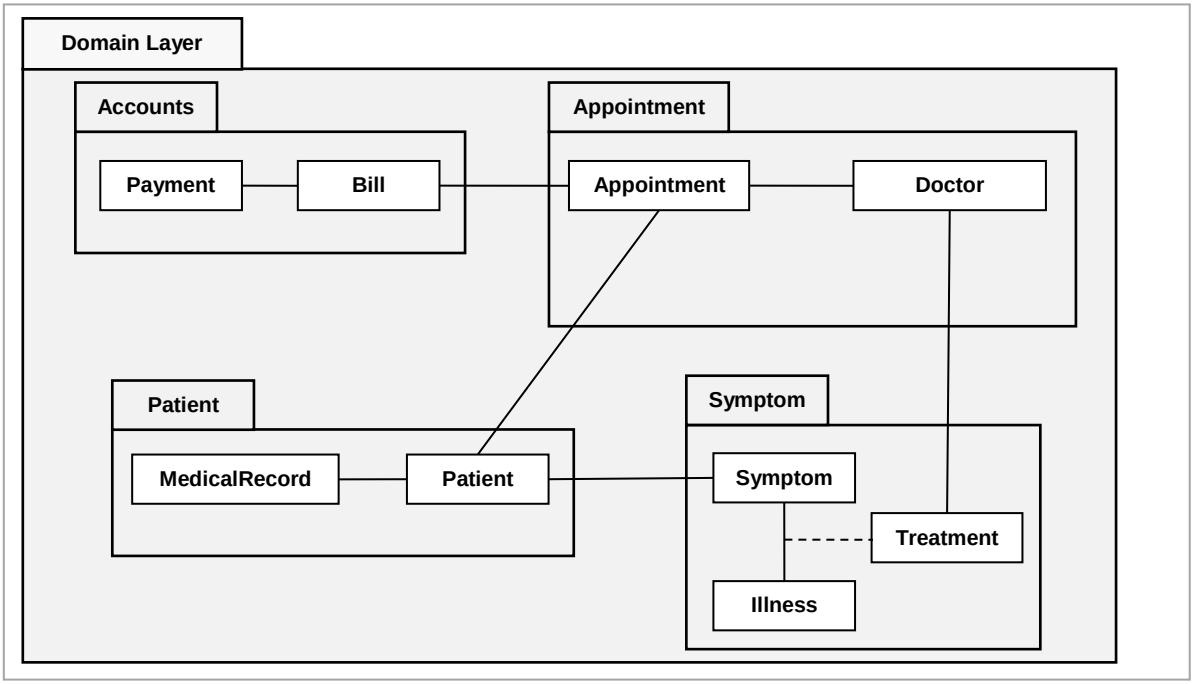
Subsystem Partitioning

- Layering involves a vertical distribution of functionality.
- Partitioning is another approach to distribution where all parts are at the same level
- Example: when a layer is complex, it can be partitioned to reduce complexity

Example: 4 Layers with 2 Partitions in presentation layer and 2 partitions in application layer



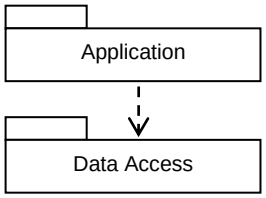
Example: Domain Layer with 4 Partitions



Note:
Layer – usually refers to a logical boundary
Tier – usually refers a physical boundary

A.1. Two-Layered (Two-Tiers) Architectural Pattern

- separate into 2 layers:
  - application layer: contains code for business logic and user interface and user interaction of the application
  - data access layer: contains code handling data persistence

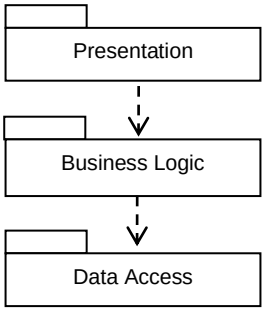


- disadvantage: user interface in application layer coupled to data representation in data access (database) layer – difficult to modify either one independently

- better to introduce middle layer to provide that is an abstract view of data representation in data access layer to the business logic

A.2 Three-Layered (Three-Tiers) Architectural Pattern

- separate into 3 layers: -
    - presentation layer: contains code for user interface and user interaction of the application
    - business logic layer: contains code for the business logic of the application
    - data access layer: contains code handling data persistence



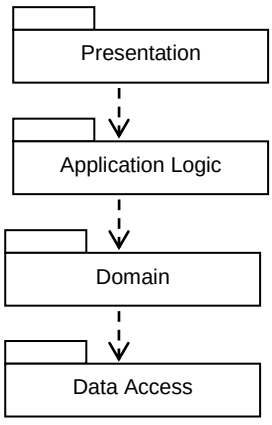
- aim is to separate business logic from the presentation and the data access layers

- provides flexibility:
    - example: when moving application currently running on desktop to Web browser, changes limited presentation layer
    - example: when changing data access layer from one type of database management system to another, changes limited data access layer

A.3 Four-Layered (Four Tiers) Architectural Pattern

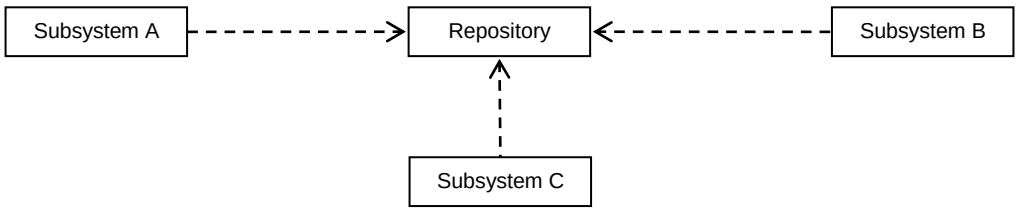
- separate business logic layer into application logic layer and domain layer

- applicable when problem domain and controller classes are complex

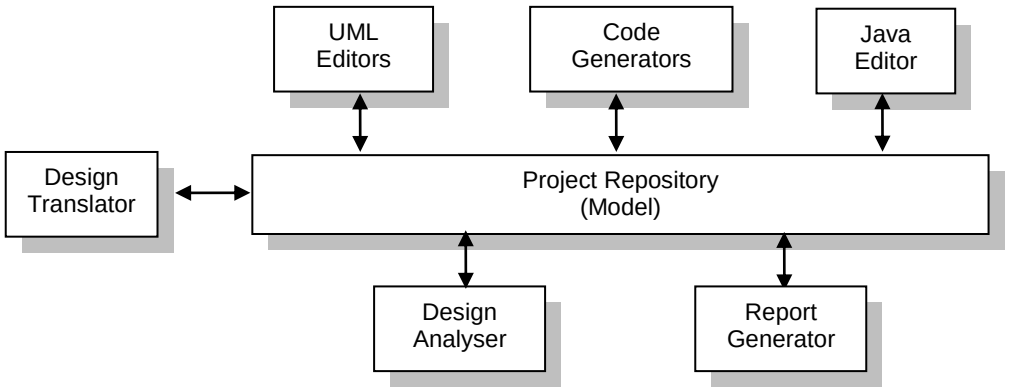


B. Repository Architectural Pattern

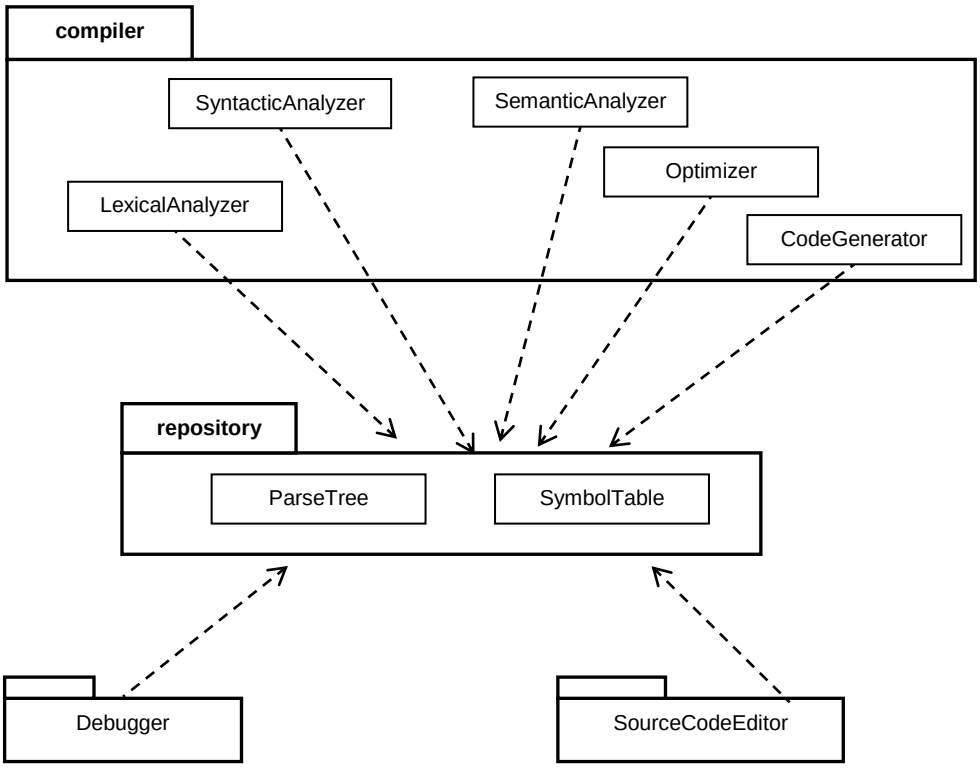
Name	**Repository Architecture (concerned with static structure)**
Description	All data in a system is managed in a central repository (data storage system) that is accessible to all subsystems. Subsystems do not interact directly with each other, only through the repository.
Advantages	Subsystems can be independent – they do not need to know of the existence of other subsystems. Changes made to the repository by one subsystem are visible to all other subsystems.
Disadvantages	Problems or faults in the repository affect the whole system. May have inefficiencies in organising all communication through the repository.



Example: System like Enterprise Architect



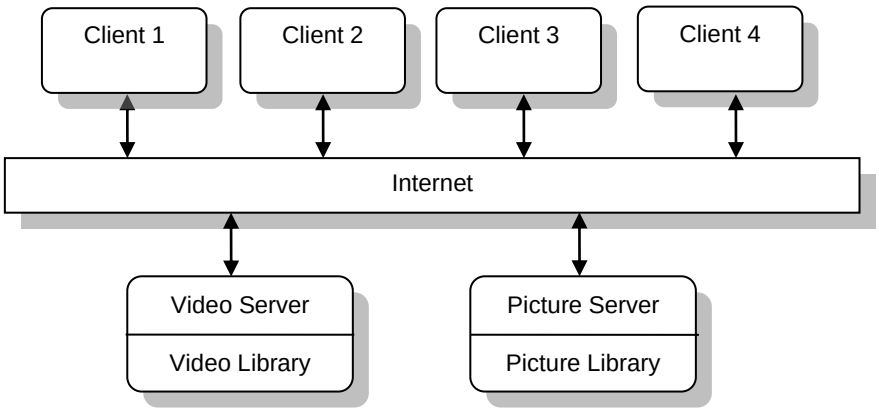
Example: System like Eclipse IDE



C. Client-Server Architectural Pattern

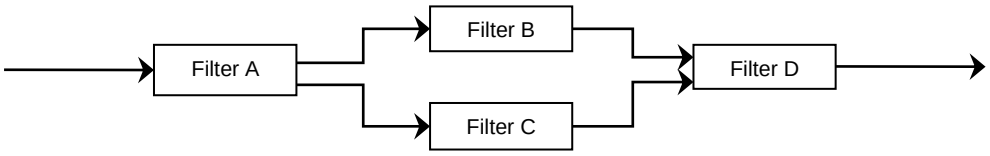
Name	**Client-Server Architecture (concerned with run-time structure)**
Description	The functionality of the system is organised into services provided by a server. Clients access the server to make use of these services.
Advantages	The clients and servers can be distributed across a network.
Disadvantages	A server failure affects all clients. The server may be open to denial of service attacks. Performance may be unpredictable because it depends on the network as well as the system.

Example: Multi-user web-based system for providing video and picture library services.

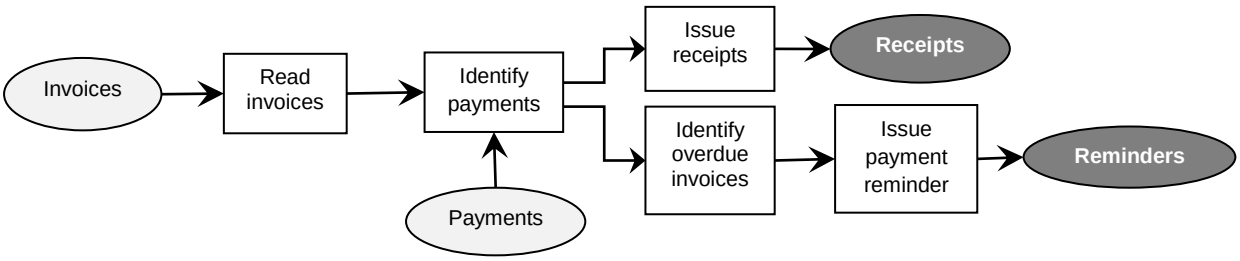


D. Pipe-and-Filter Architectural Pattern

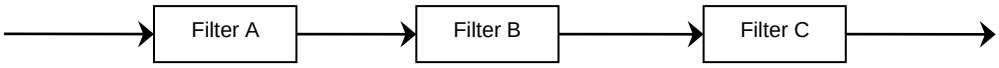
Name	**Pipe-and-Filter Architecture**
Description	The processing of data in a system is organised as a series of processing elements where each element carries out one type of data transformation. The elements are called filters and the connections that transmit data from one element to the next are called pipes. Each filter works independently from other filters. Commonly used in data processing applications where inputs are processed in separate stages to generate outputs.
Advantages	Filters can be easily replaced by another filter that performs the same processing as long as the input and output formats are the same.
Disadvantages	The format of data transfer has to be agreed upon between communicating filters.



Example: A system to process invoices.



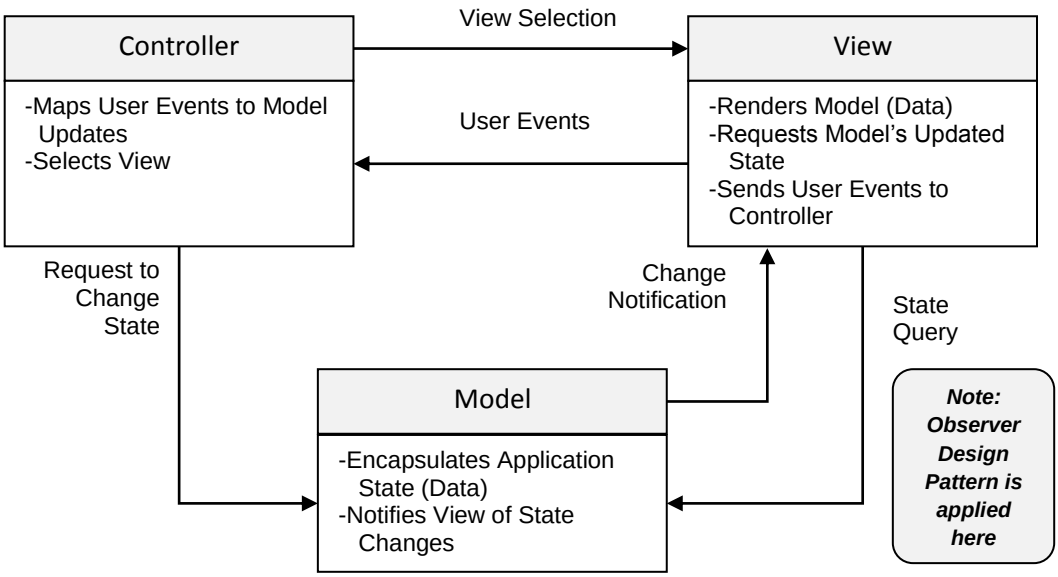
If there is only a single line of filters, it is called sequential batch processing



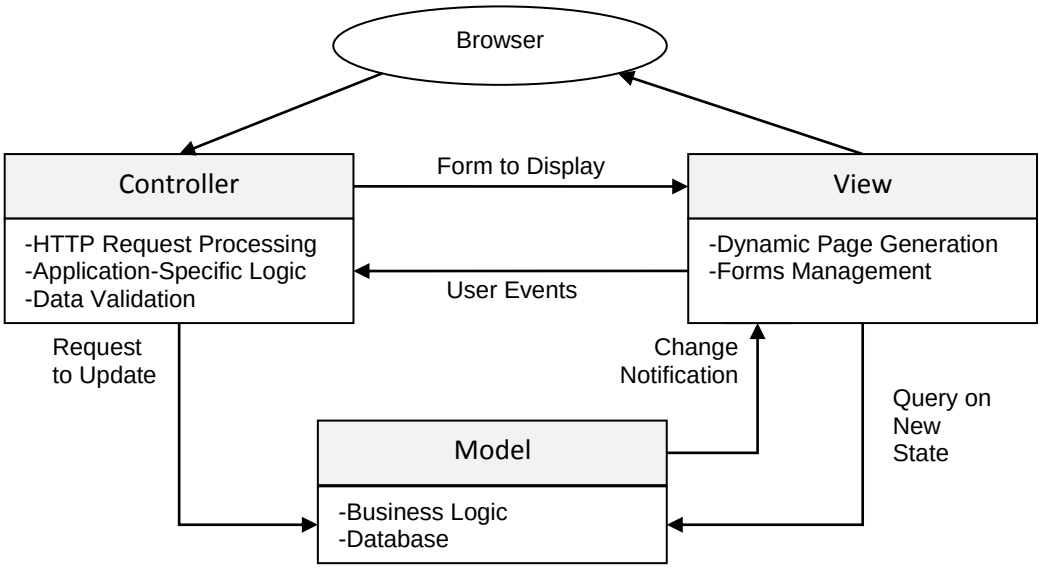
E. Model-View-Controller (MVC) Architectural Pattern

Name	**Model-View-Controller (MVC) Pattern**
Description	Separates presentation and interaction from the system data. The system is structured into three logical elements that interact with each other. The Model manages the system data and associated operations on that data. The View defines and manages how the data is presented to the user. The Controller manages user interaction (e.g. key presses, mouse clicks, etc.) and updates the Model and the View.
Advantages	Allows the structure of the data to change independently of its presentation and vice-versa. Supports presentation of the same data in different ways (views) with changes made in one view updated in all other views.
Disadvantages	Can involve additional code and increase code complexity when the data model and interactions are simple.

Generic Diagram

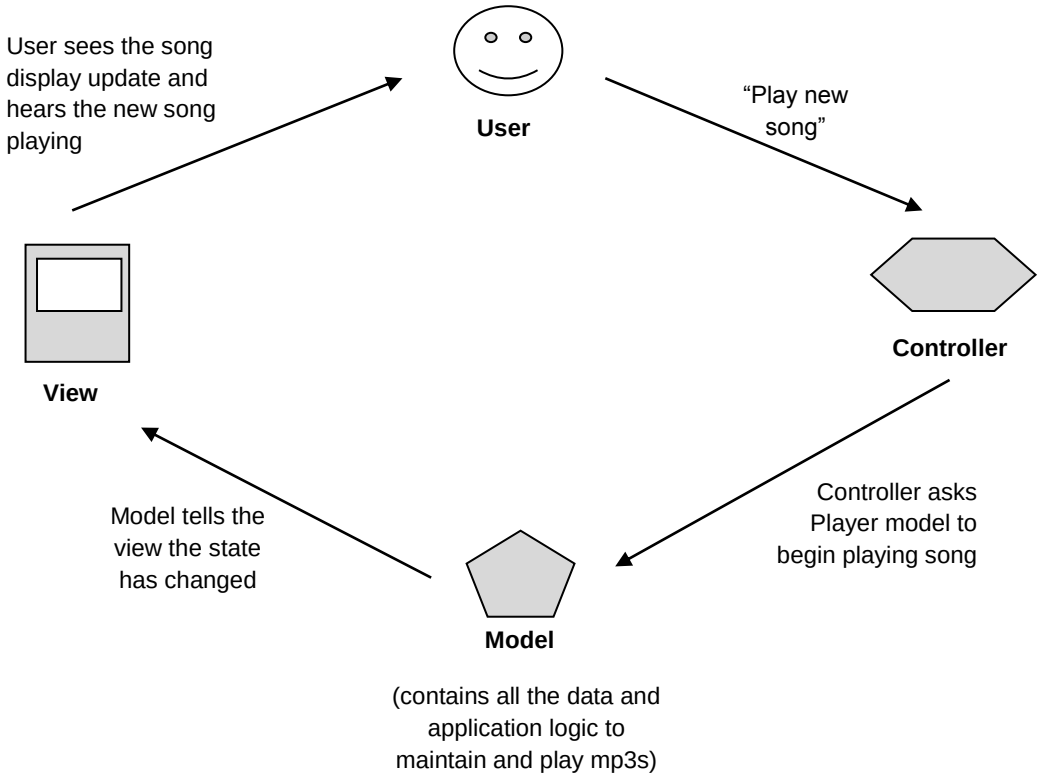


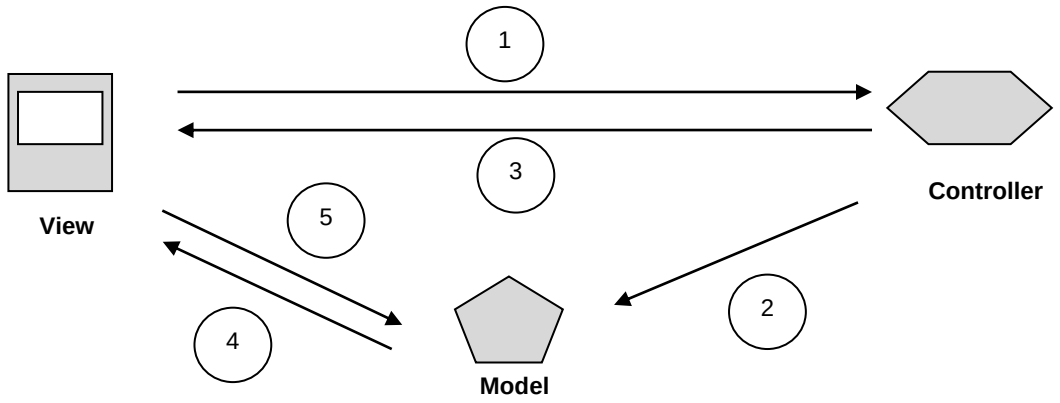
Example: Web application architecture using MVC pattern



Example (from Heads First Design Patterns (Freeman and Freeman))

Imagine you're using an MP3 player, like iTunes. You can use its interface to add new songs, manage playlists and rename tracks. The player takes care of maintaining a little database of all your songs along with their associated names, etc. It also takes care of playing the songs. While it plays the songs, the user interface is constantly updated with the current song title, the running time, and so on.

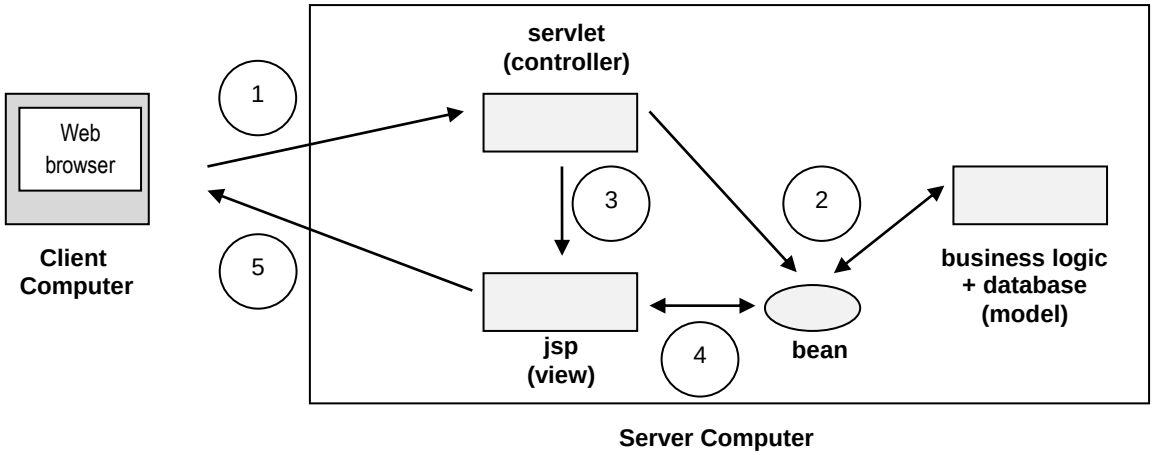




1	**User interacts with the View** When the user does something to the view (like click the Play button), the view tells the controller what the user did. It's the controller's job to handle the user's action.
2	**Controller asks Model to change its state** The controller takes the user's actions and interprets them. If the user clicks on a button, it's the controller's job to figure out what that means and how the model should be manipulated based on that action.
3	**Controller may also ask View to change** When the controller receives an action from the view, it may need to tell the view to change as a result. For example, the controller could enable or disable certain buttons or menu items in the interface.

4	**Model notifies View when its state has changed** When something changes in the model, based either on some action the user took (like clicking a button) or some other internal change (like the next song in the playlist has started), the model notifies the view that its state has changed.
5	**View asks Model for state** The view gets the state directly from the model and displays it. For instance, when the model notifies the view that a new song has started playing, the view requests the song name from the model and displays it.

MVC and the Web



1	**User makes an HTTP request, which is received by Servlet** Using a web browser, the user makes an HTTP request. This typically involves sending along some form data. A servlet receives this form data and parses it.
2	**Servlet acts as Controller** The servlet plays the role of a controller and processes the user's request, most likely making requests on the model (usually a database). The result of processing the request is usually bundled in the form of a JavaBean.
3	**Controller forwards control to View** The View is represented by a JSP. The JSP's only job is to generate the page representing the view of the model along with any controls needed for further actions.
4	**View obtains data representing Model** The view obtains data representing the model from the JavaBean.
5	**View returns a page to the browser via HTTP response** A page is returned to the browser, where it is displayed as the view. The user submits further requests, which are processed in the same manner.