Chapter 3

Architecture Styles (Pipe and Filter, Event-based, Layered etc.) and Architectural View Models)

Outline

- What is Architecture and architect role
- Architecture Patterns
  - Pipe and filter
  - Event-based
  - Layered etc.
- Architectural view models



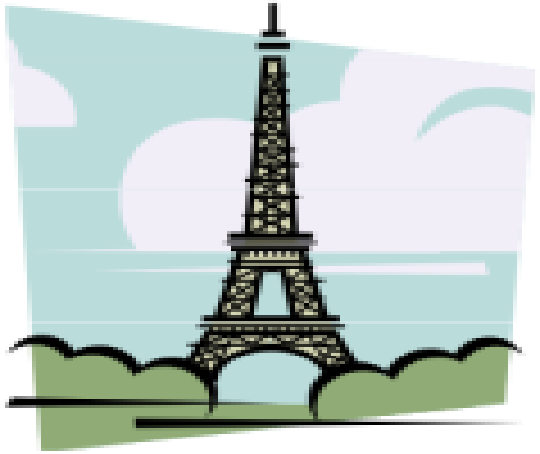

What is Software Architecture?

- **The architecture of a system consists of:**
  - **The structure(s) of its parts**
    - Including design-time, test-time, and run-time hardware and software parts
  - **The externally visible properties of those parts**
    - modules, hardware units
  - **The relationships and constraints between them**

- **Architecture** is the **fundamental organization of a system**, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

- **Architecture is:**
  - All about **communication**
  - What '**parts**' are there?
  - How do the **'parts' fit together**?

- **Architecture is Not:**
  - About development
  - About algorithms
  - About data structures



Other Definitions of Software Architecture

- Architecture **is high-level design**.
- Architecture is the **overall structure of the system.**
- Architecture is a **set of principal design decisions** about a software system
- Architecture is the **structure of the components of a program or system, their interrelationships,** and the **principles** and **guidelines** governing their design and evolution over time.
- Architecture is **components and connectors**

- **Note: Almost all of the above definitions focus on reasoning about the structural system issue.**

Fundamental concepts of Architecture

- **Three fundamental** understandings of software architecture.
  - **Every application** has **an architecture**.
  - Every application has at **least one architect**.
  - Architecture is **not a phase of development**.

Note: Architecture is a design, but every design is not an architecture.

Architecture-Centric Design

- **Traditional design phase** suggests translating the requirements into algorithms, so a programmer can implement them
- **Architecture-centric design** includes the following considerations:
  - Stakeholder issues
  - Decision about use of COTS component
  - Overarching style and structure
  - Package and primary class structure
  - Deployment issues
  - Post implementation/post deployment issues

Software Architecture (cont'd)

- **A software architecture defines:**
  - The components of the software system
  - How the components use each other's functionality and data
  - How control is managed between the components
- Software architecture is **the blueprint** for a software system's construction and evolution
- Architecture = {**components, connectors, constraints**}

Components

- A **software component** is an architectural entity that :
  - Encapsulates a subset of the system's functionality and/or data
  - Restricts access to that subset via an explicitly defined interface
  - Has explicitly defined dependencies on its required execution context
- Components typically provide application-specific services.
- **Types of components**
  - **computational**: does a computation of some sort. E.g. function, filter .
  - **memory**: maintains a collection of persistent data. E.g. data base, file system, symbol table.
  - **manager**: contains state + operations. State is retained between invocations of operations. E.g. server.
  - **controller**: governs time sequence of events. E.g. control module, scheduler .

Connectors

- A **software connector** is an architectural building block ,regulating interactions among components.
- In many software systems, connectors are usually simple procedure calls or shared data accesses
  - ✓ Much more sophisticated and complex connectors are possible!
- Connectors typically provide application-independent interaction facilities.

Configuration/Topology

- Components and connectors are composed in a specific way in a given system's architecture to accomplish that system's objective.
- An **architectural configuration**/topology, is a set of specific associations between the components and connectors of a software system's architecture

Prescriptive vs. Descriptive Architecture

- A system's **prescriptive architecture** captures the design decisions made prior to the system's construction.
  - It is the as-conceived or as-intended architecture
- A system's **descriptive architecture** describes **how the system has been built**
  - It is the as-implemented or as-realized architecture

Domain Specific Software Architecture

- Domain-specific software architectures (DSSAs):
  - Specialized for a particular task (domain).
  - Generalized for effective use in that domain.
  - Composed using a standardized topology.
- **Key benefit**:
  - maximal reuse of knowledge.
- **Key drawback**:
  - only applicable in specific domain.

Why Software Architecture is Important

1) Communication among stakeholders

- Each stakeholder (customer, user, project manager, coder, tester and so on) of a system is concerned about different characteristics of the system that will be affected by the architecture.
  - A **user** is concerned that the system is reliable and available when needed; The **customer** is concerned that the architecture can be implemented on schedule and within budget;
  - The **manager** is worried (as well as about cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways.
  - The **software architect** is worried about strategies to achieve all of those goals.
- Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a highly intellectually manageable level.

Why Software Architecture is Important

2) It helps to make early design decisions :

- Software architecture manifests the earliest design decisions about a system, which influence the system's remaining development, its deployment, and its maintenance life.
- The architecture defines constraints on implementation
- The architecture dictates organizational structure
- The architecture inhibits or enables a system's quality attributes
  e.g. If your system requires high performance, you need to manage the time based behavior of elements and the frequency and volume of inter-element communication
- **Predicting system qualities** by studying the architecture
- The architecture enables more **accurate cost and schedule estimates**

Why Software Architecture is Important…

3) It provides a transferable, re-usable model

- Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its elements work together, and this model is transferable across systems.
- In particular, it can be applied to other systems exhibiting similar quality attribute and functional requirements and can promote large-scale reuse.
- Systems can be built using large, externally developed elements
- An architecture permits template-based development
- An architecture can be the basis for training

Role of Software Architect

- Convert customer requirements into a technical design
- Lead the problem domain analysis team
- Ensure that the technical design meets quality requirement
- Perform continuous risk assessment, develop risk mitigation strategies
- Perform early prototyping aimed at mitigating major risks
- Communicate with stakeholders through detailed technical presentations
- Listen to stakeholders and build consensus
- Review developer code and ensure conformance to the architecture and good coding practices
- Serve as a mentor for analysts, designers, and developers

Architectural Design Process

- System structuring and partitioning
- Decomposition of software system into sub-systems and communications between sub-systems
- Sub-system is an independent system from other sub-systems.
- Decomposition of sub-system into modules or components
  - ✓Component (module) provides services to other component.
  - ✓Reuse component

Sub-systems, Modules and Components

- A **sub-system** is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A **module** is a system component that **provides services** to other components but would not normally be considered as a separate system.
- A **component** is an **independently deliverable unit** of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.

Architecture Style

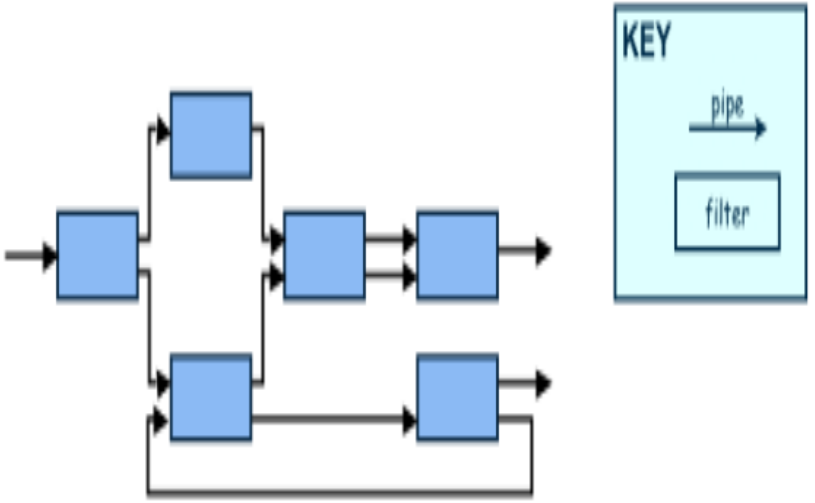
- Set of rules, constraints, or patterns of how to structure a system into a set of components and connectors.
- **An *architectural style*** defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined.

Types of Architectural Styles

- Pipes-and-Filter
- Client-Server
- Peer-to-Peer
- Event Based
- Layering
- MVC
- SOA (Service Oriented Architecture)

Pipes-and-Filter

- The system has
  - Streams of data (pipe) for input and output
  - Transformation of the data (**filter**)



- "The ***Pipes and Filters*** architectural pattern provides a structure for systems that **process a stream of data**. Each processing step is encapsulated in a filter component. Data are passed through pipes between adjacent filters. Recombining filters allows you to build families of related filters."

Cont'd

- P&F architecture consist of producer/consumer subsystems each subsystem may produce, consume, or consume/produce data and connectors (pipes) to forward the data from one filter to another involving transformation on streams of data.
- **Implementation steps:**
  - Divide the functionality of the problem into a sequence of processing steps.
  - Define the type and format of the data to be passed along each pipe.
  - Determine how to implement each pipe connection.
  - Design and implement the filters.

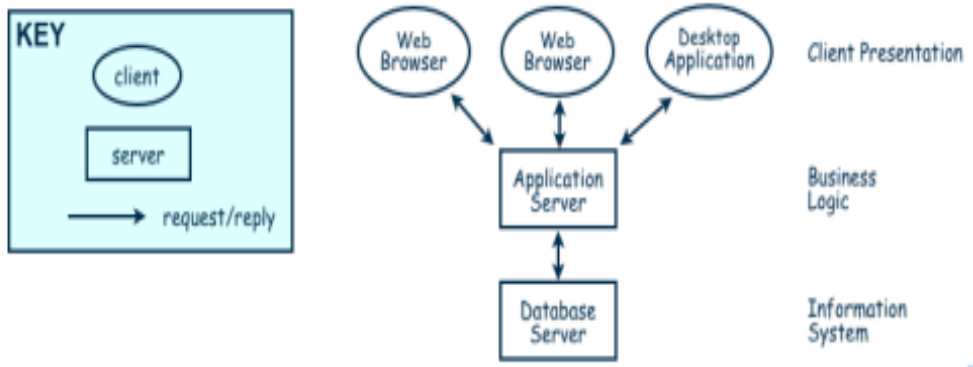
    Note: The design of a filter is based on the nature of the task to be performed and the natures of the pipes to which it can be connected.

Cont’d

- **Several important properties**
  - The designer can understand the entire system's effect on input and output as the composition of the filters
  - The filters can be reused easily on other systems
  - System evolution is simple
- **Drawbacks**
  - Not good for handling interactive application
  - Duplication in filters functions

Client-Server

- A client-server architecture *distributes application logic and services* respectively to a number of client and server subsystems, each potentially running on a different machine and communicating through the network (e.g. by RPC).
- Two types of components:
  - Server components offer services(Response/Reply)
  - Clients sends request and receive response from server.
- Client may send the server a request and server reply a response to the request. It is a kind of request and response/reply mechanism.



Cont'd

- ***Advantages***
  - *Distribution* of data is straightforward
  - Makes *effective use of networked systems*.
  - May require cheaper hardware
  - Easy to *add new servers* or upgrade existing servers
- ***Disadvantages***
  - *Redundant management* in each server
  - May require a *central registry* of names and services - it may be hard to find out what servers and services are available

Peer-to-Peer (P2P)

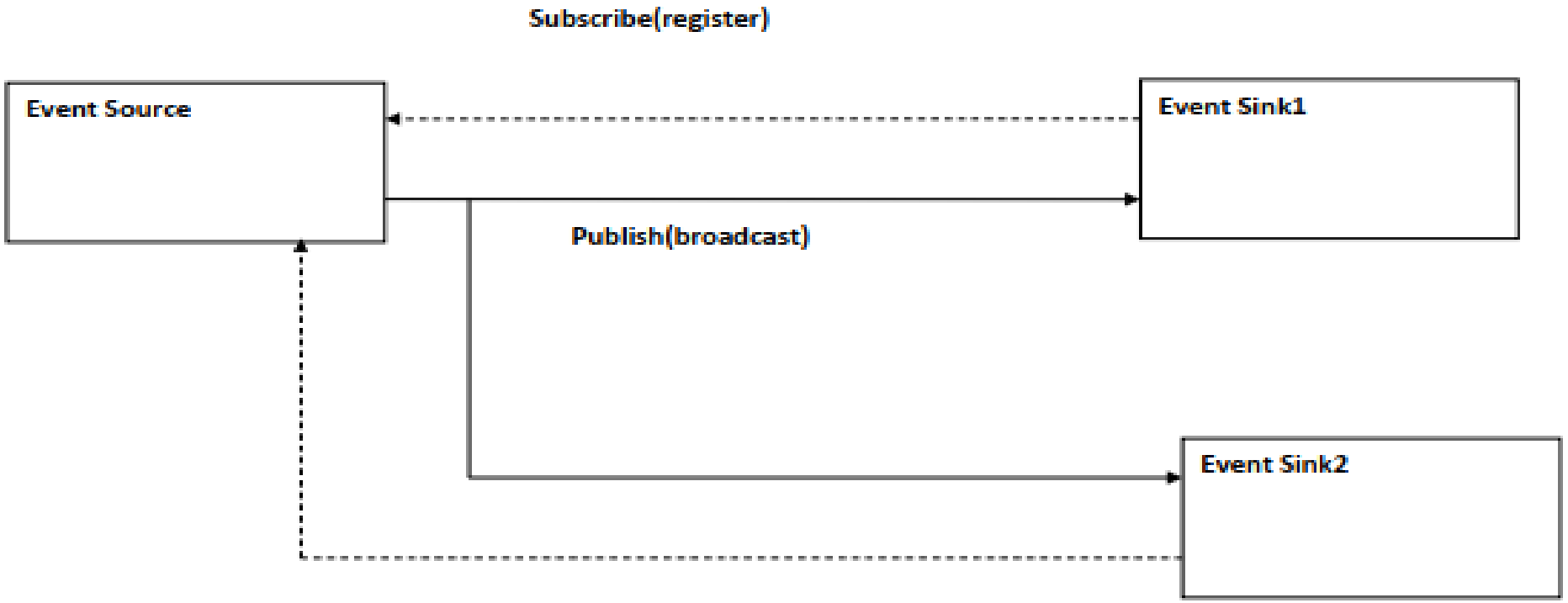
- Each component acts as its own process and acts as both a client and a server to other peer components.
- Any component can initiate a request to any other peer component.
- **Characteristics**
  - Scale up well
  - Increased system capabilities
  - Peers are distributed in a network, can be heterogeneous, and mutually independent.
  - Robust in face of independent failures. Highly scalable
  - This differs from client/server architectures, in which some computers are dedicated to serving the others.
  - Components do not offer the same performance under heavy loads.

Event-Based Architecture

- Components interact by broadcasting and reacting to events
  - Component expresses interest in an event by subscribing to it
  - When another component announces (publishes) that event has taken place, subscribing components are notified
  - Implicit invocation is a common form of publish-subscribe architecture
  - Registering: subscribing component associates one of its procedures with each event of interest (called the procedure)
- Characteristics
  - Strong support for evolution and customization
  - Easy to reuse components in other event-driven systems
  - Difficult to test

Cont'd

- In **broadcast models** an event is broadcast to all sub-systems. Any sub-system which can handle the event may do so.

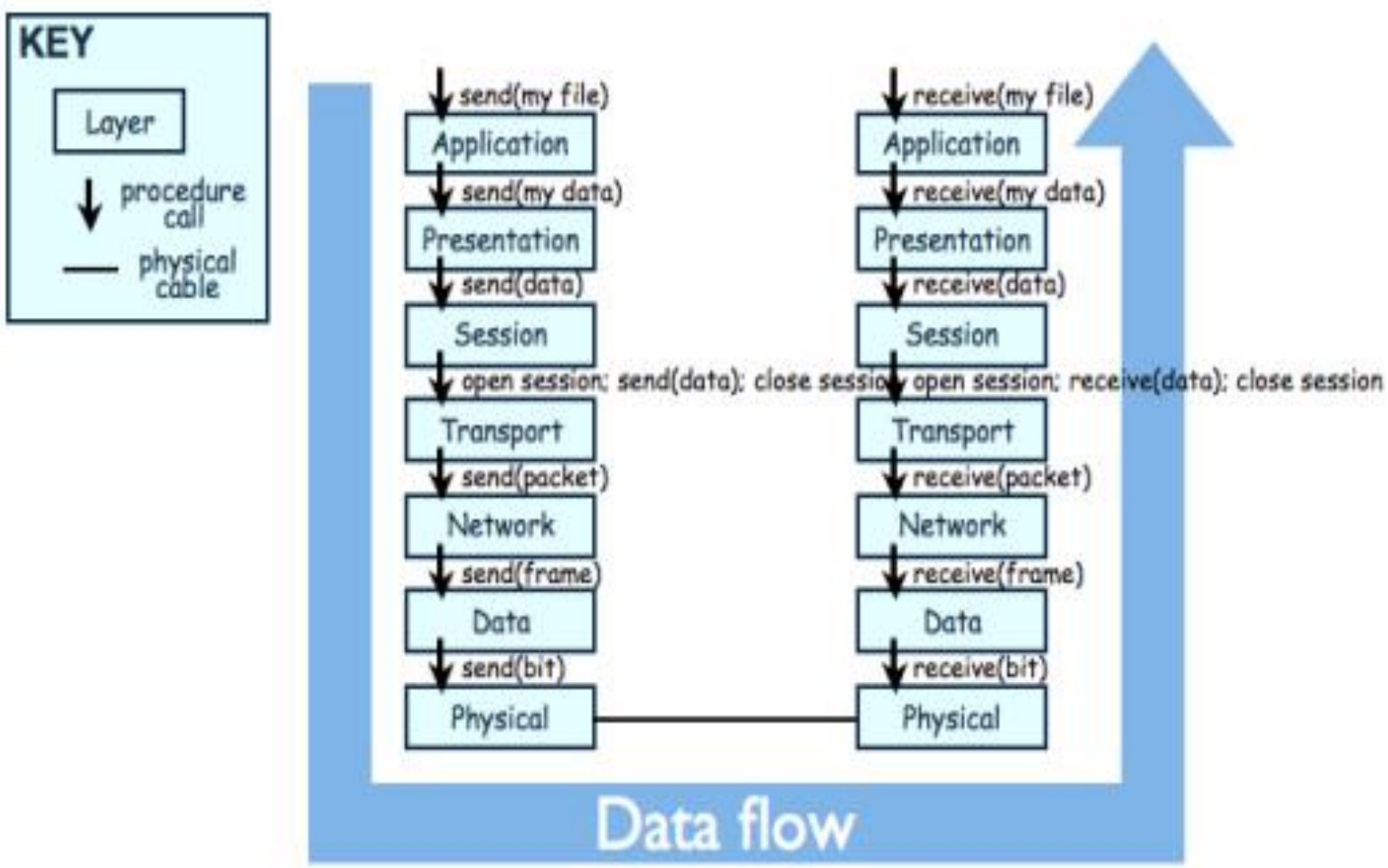


Layering

- Layers are hierarchical
  - ✓Each layer provides service to the one outside it and acts as a client to the layer inside it
  - ✓Layer bridging: allowing a layer to access the services of layers below its lower neighbor
  - ✓Each layer has two interfaces.
  - ✓Upper interface provides services and Lower interface requires services.
- The design includes **protocols**
  - ✓Explain how each pair of layers will interact
- **Advantages**
  - ✓ High levels of abstraction
  - ✓Relatively easy to add and modify a layer
- **Disadvantages**
  - ✓ Not always easy to structure system layers
  - ✓System performance may suffer from the extra coordination among layers

cont'd

- The **OSI** model



MVC (Model-View-Controller)

- In the MVC paradigm, the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by **three types of objects**, each specialized for its task.
  - The **view** manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application.
  - The **controller** interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.
  - The **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

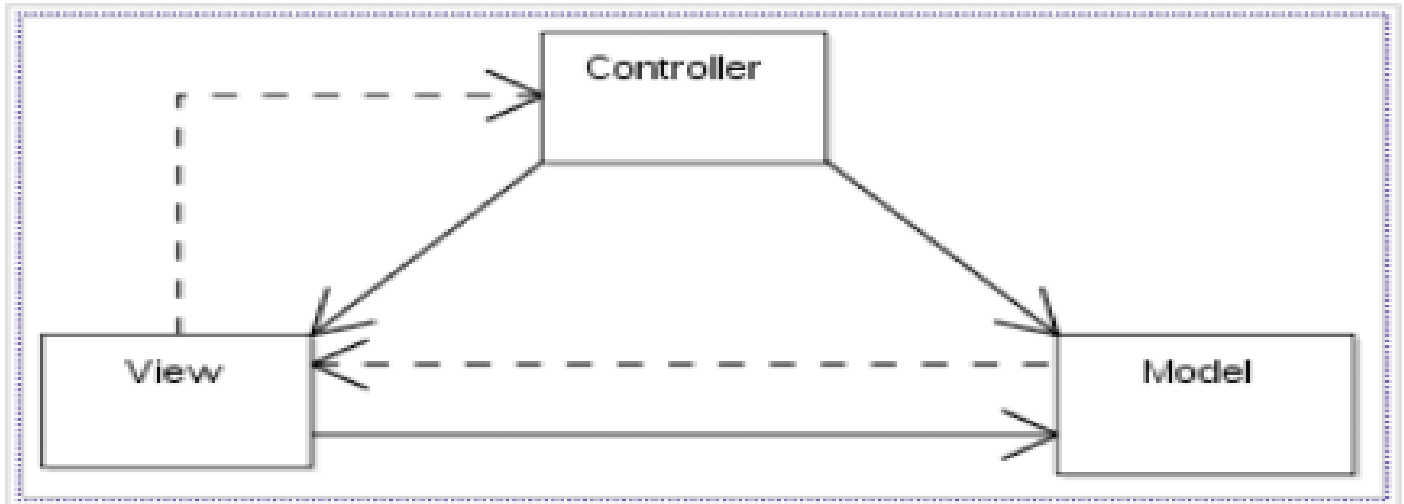
MVC ….

- The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.
- MVC was originally developed to map the traditional input, processing, output roles into the GUI system:

Input → Processing → Output
Controller → Model → View

Cont'd

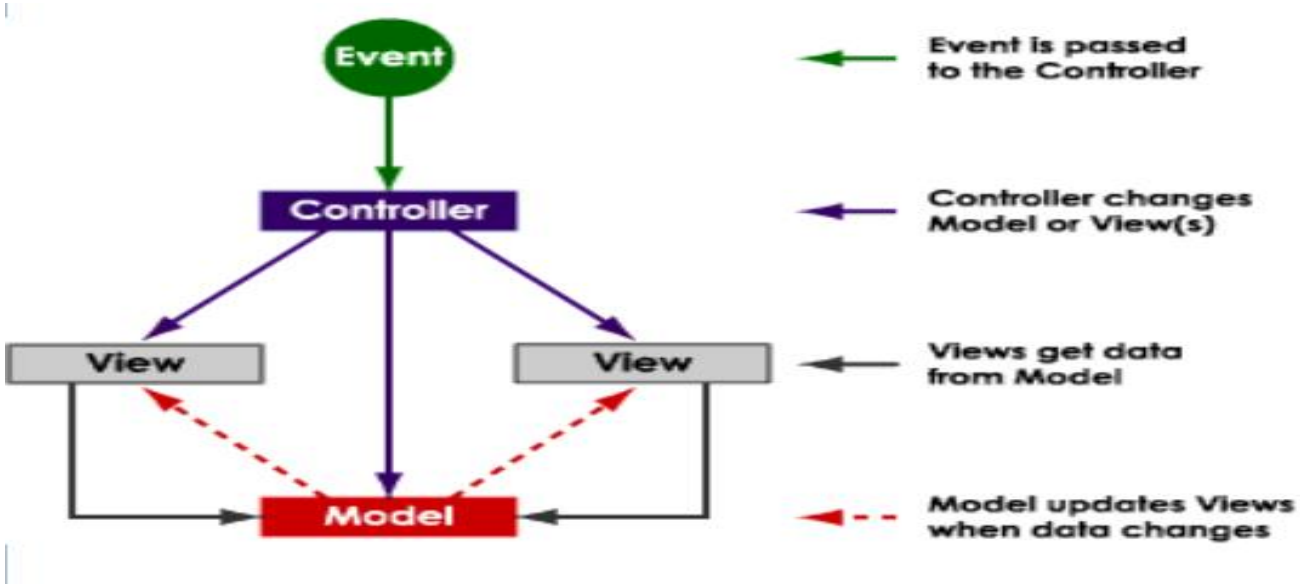
- The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.



Model-View-Controller concept. Note: The solid line represents a direct association, the dashed an indirect association via an observer (for example).

MVC benefits

- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one don't affect the others
  - can develop in parallel once you have the interfaces



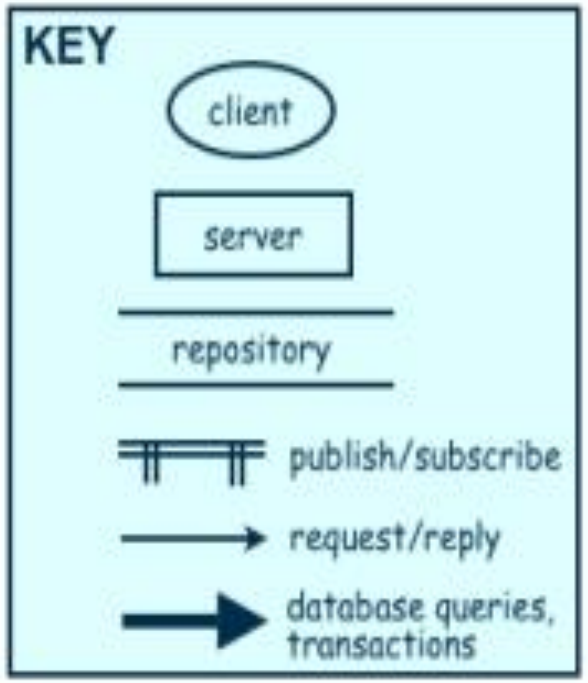
Summary (MVC)

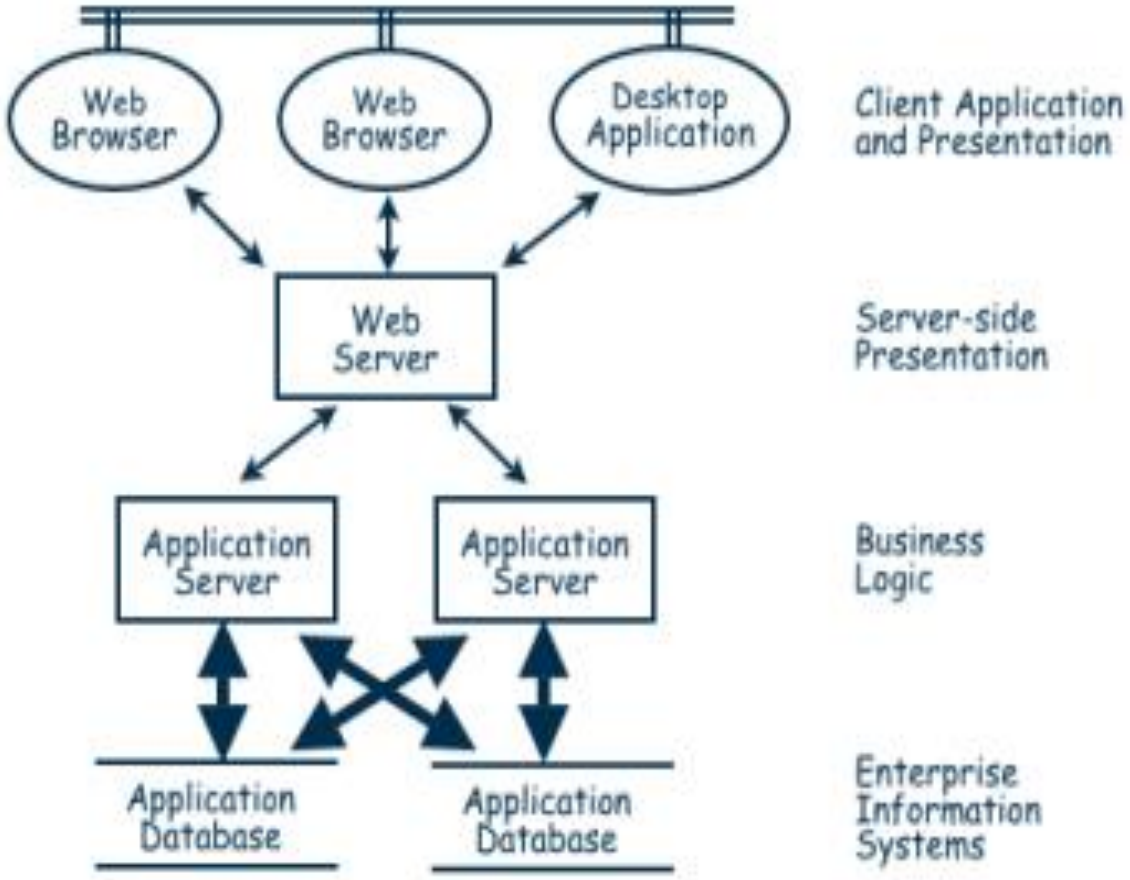
- The intent of MVC is to keep neatly separate objects into one of tree categories
- **Model**
  - The data, the business logic, rules, strategies, and so on
- **View**
  - Displays the model and usually has components that allows user to edit change the model
- **Controller**
  - Allows data to flow between the view and the model
  - The controller mediates between the view and model

Combining Architectural Styles

- Actual software architectures rarely based on purely one style
- Architectural styles can be combined in several ways
  - Use different styles at different layers (e.g., overall client-server architecture with server component decomposed into layers)
  - Use mixture of styles to model different components or types of interaction (e.g., client components interact with one another using publish-subscribe communications
- If architecture is expressed as collection of models, documentation must be created to show relation between models

Combination of Event-based, Client-Server, and Repository Architecture Styles

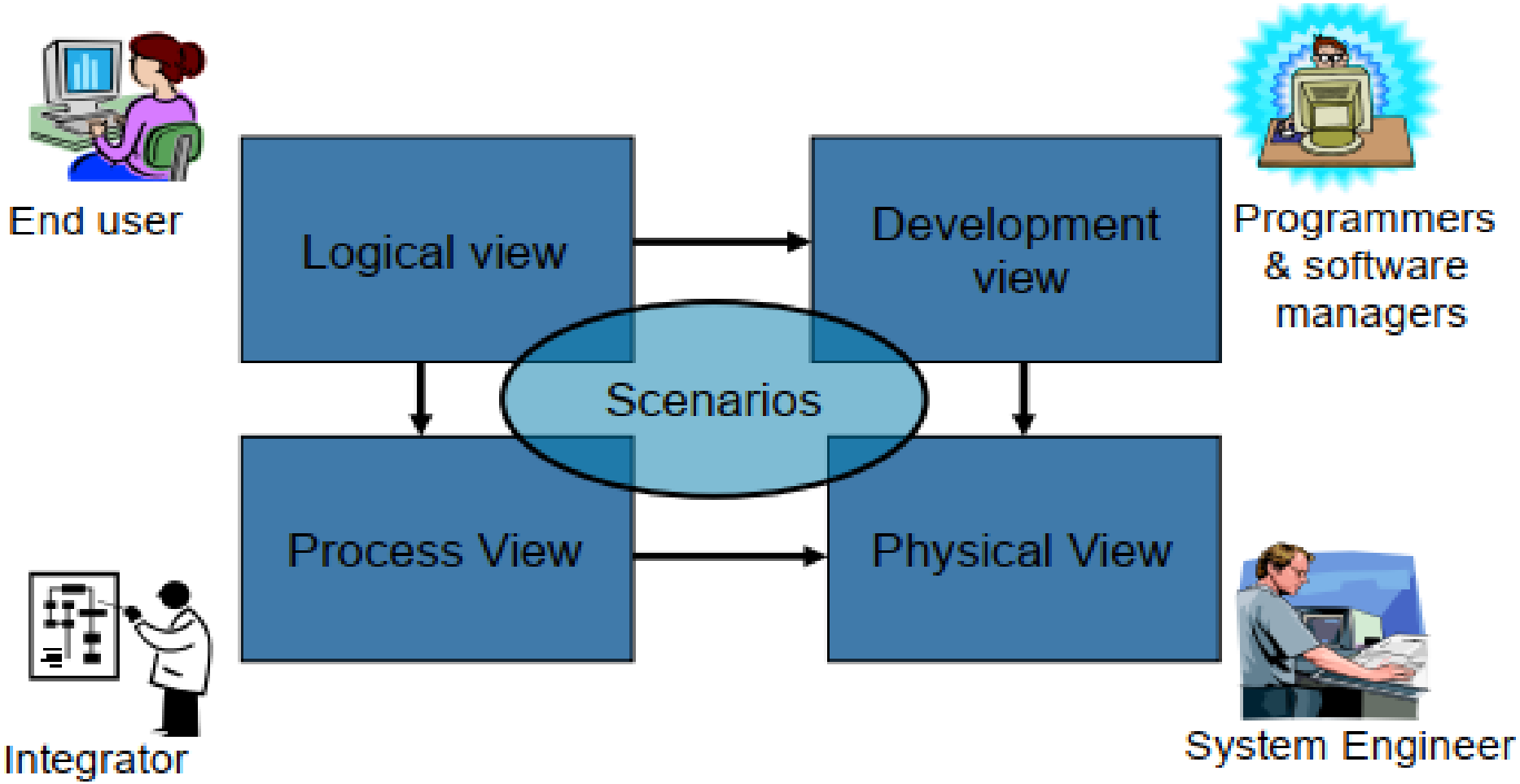




Architectural view models

- Architecture documents do not address the concerns of all stakeholders .
- Different Stakeholders : end-user, system engineers, developers, clients, architects and project managers.
- Architecture documents contained complex diagrams, some times they are hard to be represented on the documentation.
- Using different notations for several **Views,** each one addressing specific set for concerns. Using “4+1” view model

4+1 View Model of Architecture



Thank You !