Advanced Software Engineering
Part 02 – Architectural Styles

Dr. Amjad AbuHassan

Outline

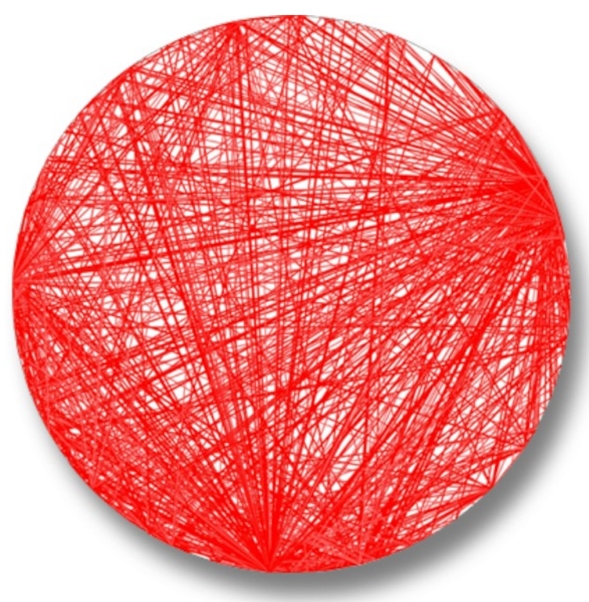
- Introduction
- Static structuring
- Dynamic structuring

Big Ball of Mud

- A big ball of mud might describe a simple scripting application with event handlers wired directly to database calls, with no real internal structure.
- Many trivial applications start like this then become impractical as they continue to grow.
- In general, architects want to avoid this type of architecture at all costs.

Big Ball of Mud cont.

- A Big Ball of Mud architecture visualized from a real code base



Architecture Styles

- Architecture style is the structure of how the user interface and backend source code are organized and how that source code interacts with a datastore.
- Two types of structure in general:
  - Static structure
  - Dynamic structure

Static Structures

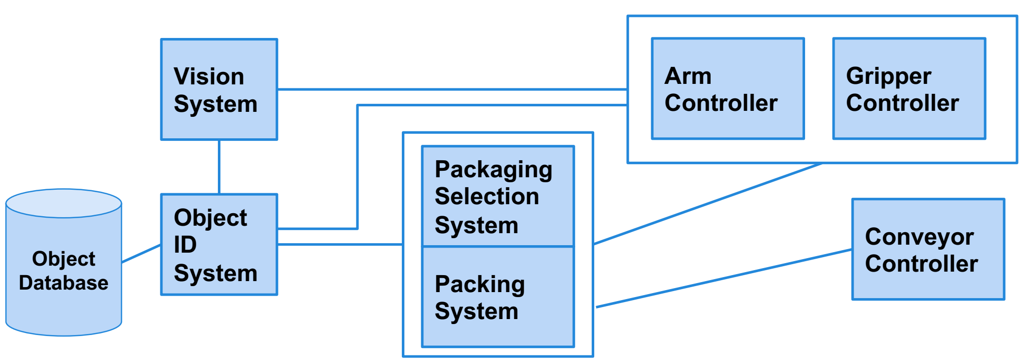
- Defines the internal design elements and their arrangement.
  - Software elements: modules, classes, packages.
  - Data elements: Database entries/tables, data files.
  - Hardware elements: Servers, CPUs, disks, networking environment
- The static arrangement of elements defines associations, relationships, or connectivity between these elements.

Dynamic Structures

- Defines the runtime elements and their interactions.
- May describe flow of information between elements
  - A sends messages to B
- May describe flow of control in a particular scenario.
  - `A.action() invokes B.action()`
- May describe effect an action has on data.
  - Entry E is created, updated, and destroyed.

Static Structuring

- Can be visualized as block diagrams presenting an overview of the system structure.

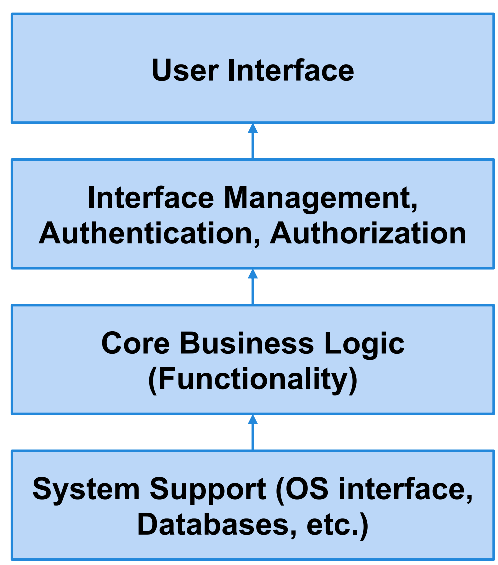


Basic Architectural Styles

- Four common styles: layered, shared repository, client/server, pipe & filter
- The style used affects the performance, robustness, availability, maintainability, etc. of the system.
- Complex systems might not follow a single model - mix and match.

Layered Model

- System functionality organized into layers, with each layer only dependent on the previous layer.
- Allows elements to change independently.
- Supports incremental development.



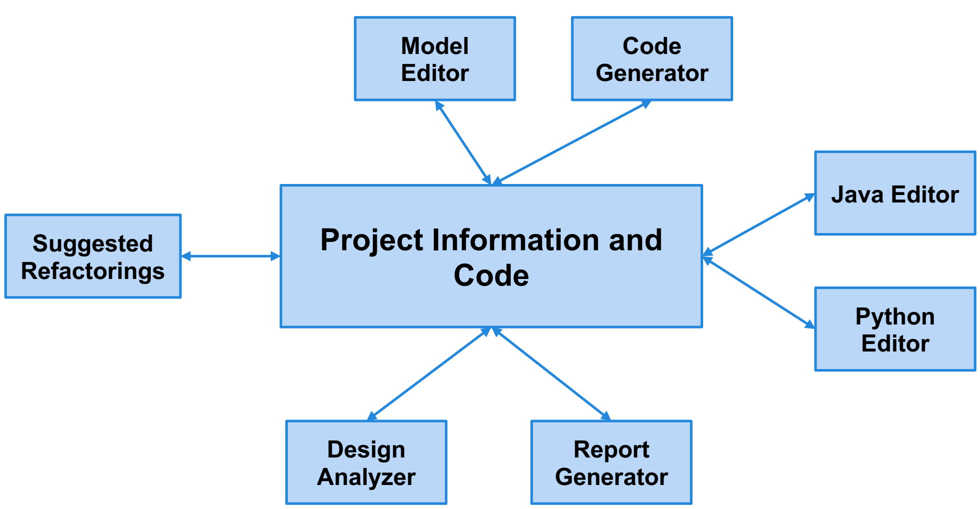
Will be discussed in detail in the next Part

The Repository Model

Subsystems often exchange and work with the same data. This can be done in two ways:

- Each subsystem maintains its own database and passes data explicitly to other subsystems.
- **Shared data is held in a central repository and may be accessed by all subsystems.**
- Repository model is structured around the latter.

IDE Example



Repository Model Characteristics

Advantages

- Efficient way to share large amounts of data.
- Consistency
- Components can be independent (May be more secure).

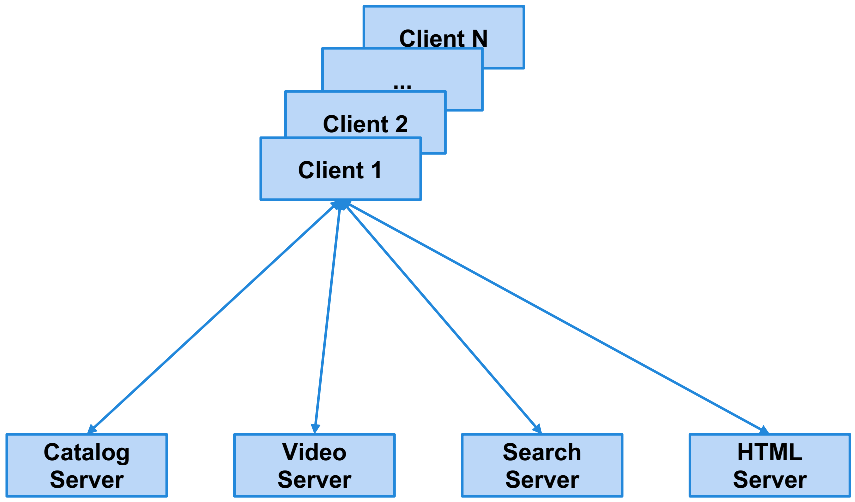
Disadvantages

- Single point of failure.
- Subsystems must agree on a data model.
- Data evolution is difficult and expensive.

Client-Server Model

- Functionality organized into services, distributed across a range of components:
- A set of servers that offer services.
  - Print server, file server, code compilation server, etc..
- Set of clients that call on these services.
  - Through locally-installed front-end.
  - Distributed systems connected across the internet.

Film Library Example



Client-Server Model Characteristics

Advantages

- Distributed architecture (Failure in one server does not impact others).
- Effective use of networked systems and their CPUs (cheaper hardware).
- Easy to add new servers or upgrade existing servers.

Disadvantages

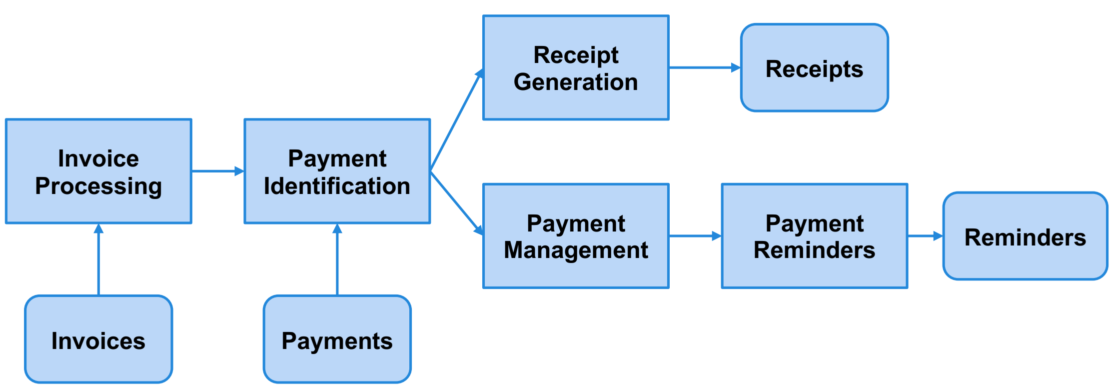
- Each service is a point of failure.
- Data exchange may be inefficient (server -> client -> server).
- Management problems if servers owned by others.

Pipe and Filter Model

Input is taken in by one component, processed, and the output serves as input to the next component.

- Each processing step transforms data.
- Transformations may execute sequentially or in parallel.
- Data can be processed as items or batches.

Customer Invoicing Example



Pipe and Filter Characteristics

Advantages

- Easy to understand communication between components.
- Supports subsystem reuse.
- Can add features by adding new subsystems to the sequence.

Disadvantages

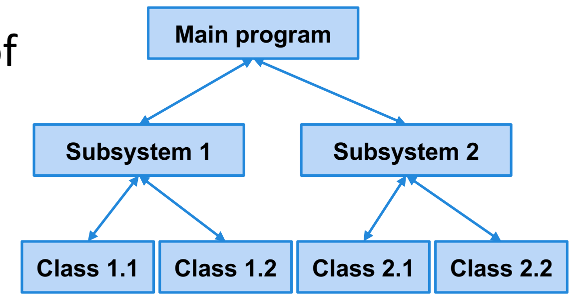
- Format for data communication must be agreed on. Each transformation needs to accept and output the right format.
- Increases system overhead.

Dynamic Structure - Control Models

- During execution, how do the subsystems work together to respond to requests?
  - Centralized Control:
    - One subsystem has overall responsibility for control and stops/starts other subsystems.
  - Event-Based Control:
    - Each subsystem can respond to events generated by other subsystems or the environment.

Centralized Control: Call-Return

- A central piece of code (Main) takes responsibility for managing the execution of other subsystems.



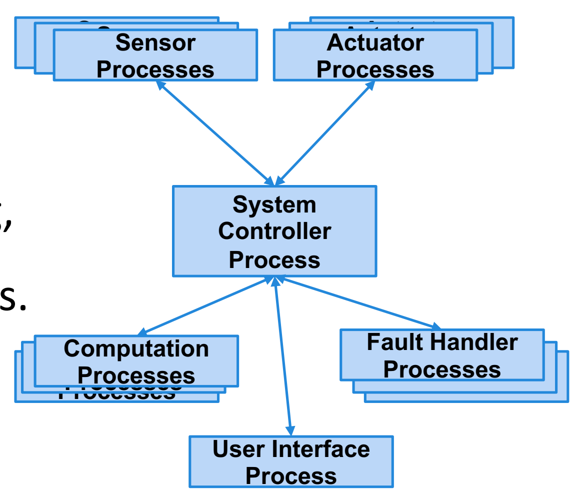
Call-Return Model

- Applicable to sequential systems.
- Top-down model where control starts at the top of a subroutine and moves downwards.

Centralized Control: Manager Model

Manager Model

- Applicable to concurrent systems.
- One process controls the stopping, starting, and coordination of other system processes.



Decentralized Control: Event-Driven Systems

Control is driven by externally-generated events where the timing of the event is out of control of subsystems that process the event.

- Broadcast Model
  - An event is broadcast to all subsystems.
  - Any subsystem that needs to respond to the event does do.
- Interrupt-Driven Model
  - Events processed by interrupt handler and passed to proper component for processing.

Broadcast Model

An event is broadcast to all subsystems, and any that can handle it respond.

- Subsystems can register interest in specific events. When these occur, control is transferred to the registered subsystems.
- Effective for distributed systems. When one component fails, others can potentially respond.
  - However, subsystems don’t know when or if an event will be handled.

Interrupt-Driven Model

Events processed by interrupt handler and passed to proper component for processing.

- For each type of interrupt, define a handler that listens for the event and coordinates response.
- Each interrupt type associated with a memory location. Handlers watch that address.
- Used to ensure fast response to an event.
    - However, complex to program and hard to validate.

Nuclear Plant Interrupt Example