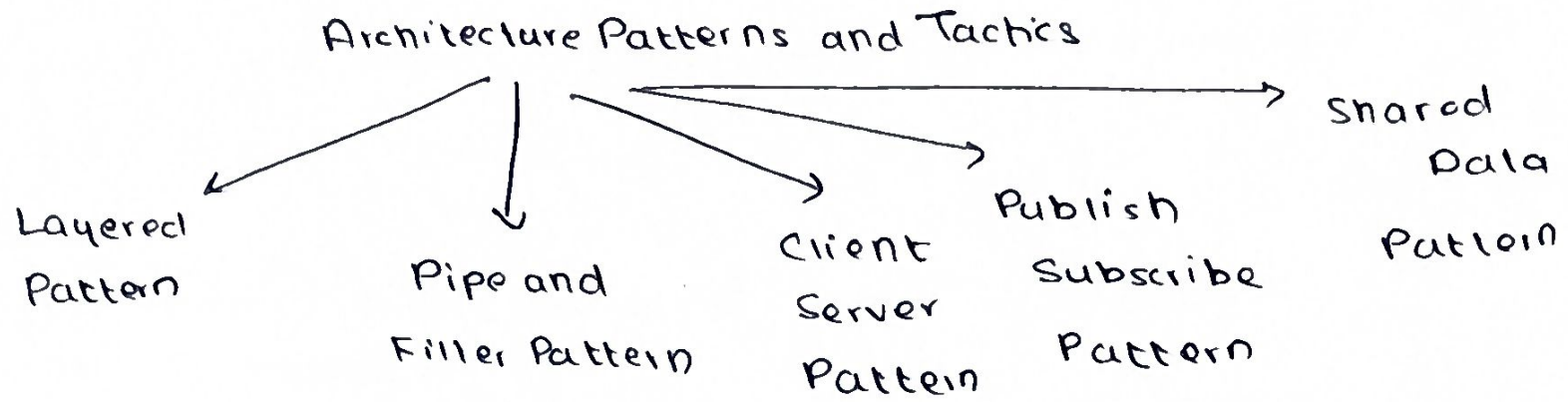
Software Architecture

Unit 4

Achieving Architectural Qualities

* Architecture Patterns and Tactics



1. Layered Pattern

Overview : divides a system into layers, with each layer offering a specific set of services.

→ Layers are stacked one on top of the other, and a higher layer is allowed to use the services of the layers directly below it

→ This enforces a strict structure & flow of interactions within the system

Elements : Key element = layer → represents a group of modules that provide cohesive services

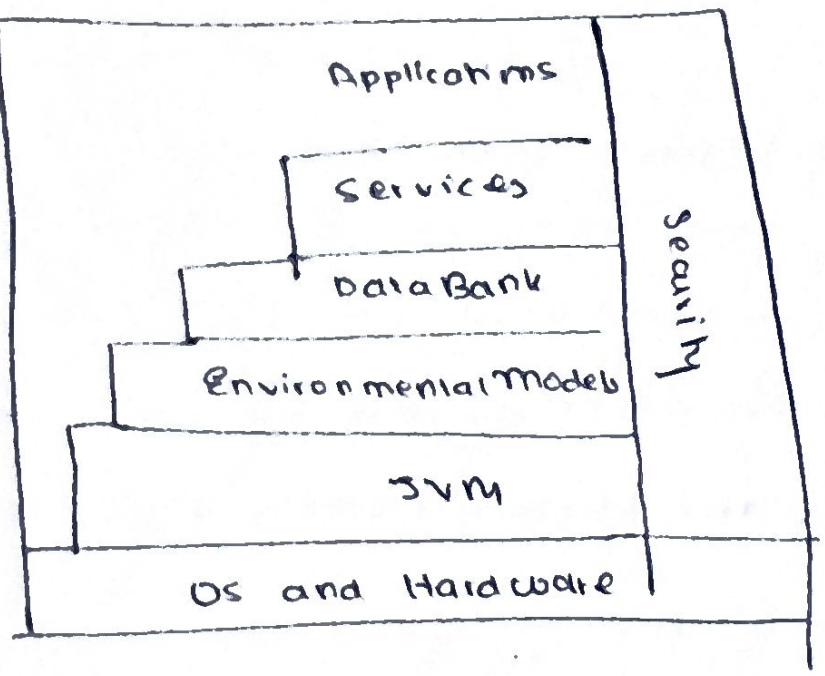
→ Each layer should define what modules it contains and what services it provides.

Relations - Layers have an allowed-to relationships, meaning a higher layer can use services from the layer immediately below it, but cannot skip layers

Constraints - each module must be assigned to a single layer
- The system must have at least two layers
- Allowed-to relations cannot be circular

Weaknesses
- Adding more layers introduces cost & complexity
- Having too many layers = performance penalty

Eg



8. Pipe and Filter Pattern

Overview - data is passed through a series of filters where each filter processes the data and then sends the transformed output to the next filter through pipes

- focus is on each filter performing a specific transformation task

Elements

(i) Filter - a component that reads in data, transforms it and sends it to the next stage.

(ii) Pipe - Filters work independently, allowing them to run in parallel

- Each filter focuses on a specific operation

(ii) Pipe - connects one filter to the other

- ensures data flow from one stage to another in a sequence

Relations - output from one filter becomes the input for the next filter through a pipe

Constraints - Filters must agree on the data format they are passing between them

- system should avoid circular dependence - Filter o/p should not feed back into itself or a higher filter

Weaknesses - not well-suited for interactive systems where user input may influence parts of the system

- can be computationally expensive when there are many filters.

3. Client - Server Pattern

Overview : has 2 main components - clients & servers

Client - initiate interactions with server, request services or data

Servers - fulfill those requests by processing them & return results to clients

— allows for separation of responsibilities between requestors & servers, making it scalable

Elements

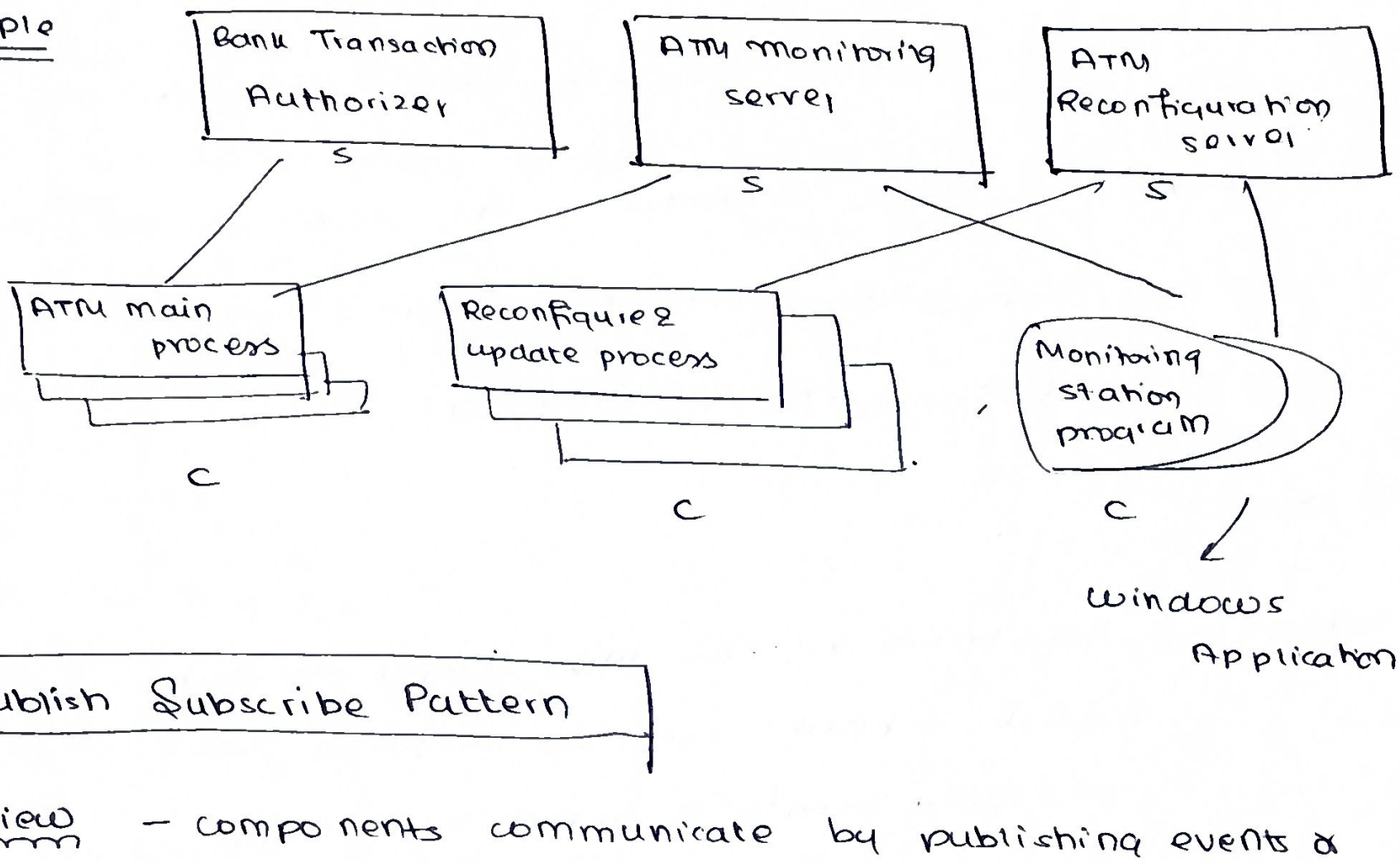
1. Client — component that invokes services of a server
   - clients have ports that describe the services that require
2. Serva - a componentthat provides services to clients
   - servers have ports that describe the services the provide
   - important characteristics include info. about the nature of the server ports (how many clients), performance characteristics (max. rates of service invocation)

Relationship - attachment relationship associates clients & servers

Constraints - no. of attachments to a given port
- allowed relations among servers

Weaknesses - server can be a performance bottleneck
- server can be a single point of failure
- Decisions about where to locate functionality (in the client or server) are often complex and costly to change after a system has been built.

Example



4. Publish Subscribe Pattern

Overview - components communicate by publishing events & subscribing to those events

- When an event occurs (published by a component), the system broadcasts the event to all components that have subscribed to it.

- It decouples the producers (publishers) from the consumers (subscribers), as publishers do not need to know who the subscribers are.

Elements

- Components - any component that has at least one publish or subscribe port
  - These ports define what events a components can publish or subscribe to
  - A component can both publish and subscribe

- Publish-Subscribe Connector - This serves as the communication mechanism, distinguishing between announce (publish) & listen (subscribe) roles for each component

Relations:

Relations — attachment relation links components to the publish-subscribe connector, defining which components announce events & which ones listen

Constraints → all components must be connected to an event distributor that routes
→ may be restrictions on what events a components can listen to or publish
- components cannot listen to their own events
- limits on how many publish-subscribe connections exist

Weaknesses - can increase latency and decrease scalability

- With many publishing events, the system can become slow

- Event delivery timing may not predictable, and the order in which subscribers receive events is not guaranteed.

Examples

Examples

① GUI - user's low level input actions are treated as events that are routed to appropriate input handlers

② MVC - view components are notified when the state of a model object changes

③ ERP systems - integrate many components, each of which are interested only in a subset of system events

④ Mailing lists

⑤ Social Networks

5. Shared Data Pattern

Overview - communication between data accessors is mediated by a shared-data store.

- control may be initiated by the data accessors or the data store
- data is made persistent by the data store.

Elements - shared data source - central data component - store data and manages aspects like data type, distribution and access permissions

Data accessor components - components that interact with the shared data store by reading or writing data

Data Reading and Writing Connector - defines how the data accessors interact with the shared data store

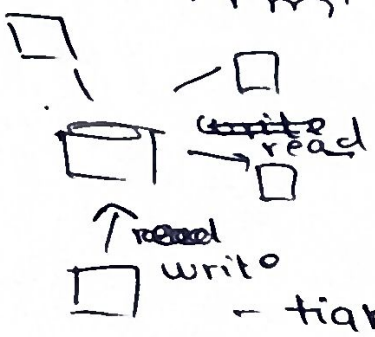
- A key consideration is whether the connector supports transactional operations (i.e all parts of a transaction succeed or fail together)

Relations - attachment relation associates data accessors w/ the data store

Constraints - data accessors are tightly connected to the data store - must use defined protocols to interact w/ it

- performance of the entire system can be limited by how the shared data store handles multiple concurrent requests

Weaknesses - shared data store can become a performance bottleneck



data store can be a single point of failure

- tightly coupling the producers and consumers to a single data store can make the system rigid & harder to scale or modify.