CS 4500
Software Development

Software Architectures

Ferdinand Vesely

September 27, 2019

Software Architectures

Software Architecture

- Describes the overall structure of the system
- Manner in which data and procedural components collaborate
- Essential tool for complexity management

Software Architecture

Two levels of abstraction:

1. Architecture in the small
   - How an individual program is decomposed into components
2. Architecture in the large
   - Organization of large / distributed systems
   - Composed of several programs, other systems

Architectural Patterns

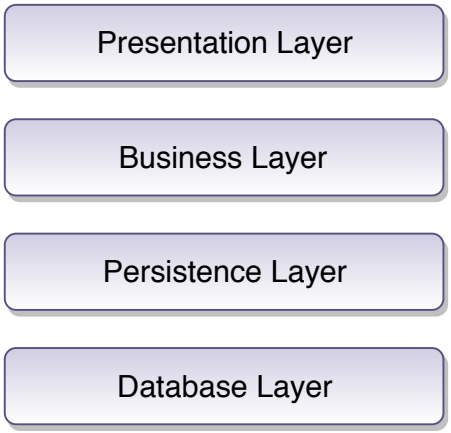
- Stylized, abstract description of good practice
- Tried and tested in different systems and environments
- System organization successful in previous systems

Layered Architecture

- Also: *multitier architecture*
- Layers with related functionality
- Layer provides services to the layer above it
- Lowest-level layers represent core services
  - likely to be used throughout the system.

Layered Architecture

Generic Example



Layered Architecture

- Separation and independence – fundamental
- Allow changes to be localized
- Supports incremental development of systems
- Portability: replace layers – as long as interface stable
- Interface change – only adjacent layer affected

Layered Architecture

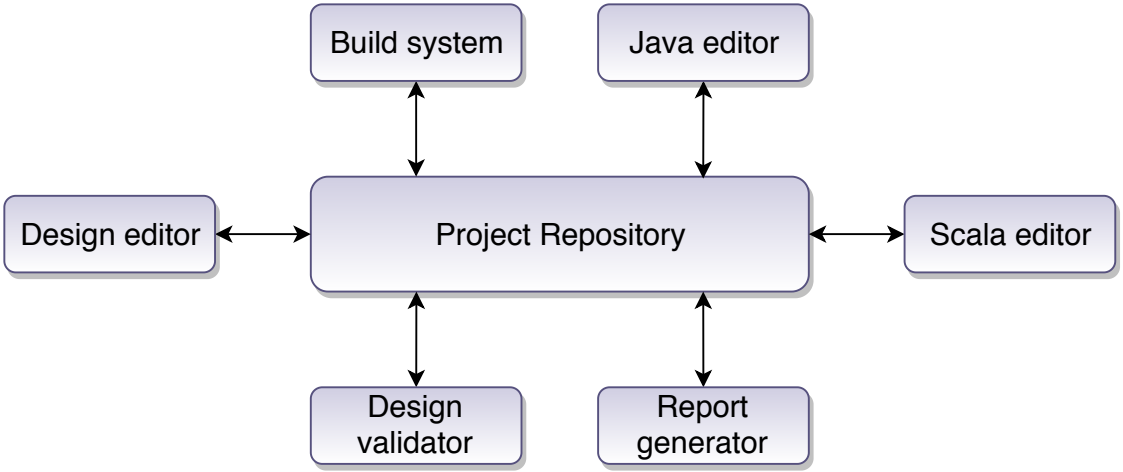
- Clean separation between layers – often difficult
- Performance can be a problem
  - multiple levels of interpretation of a service request

Repository Architecture

- All data: managed in a central repository
- Accessible to all system components
- No direct interaction between components
- Efficient for sharing large amounts of data among components
- Actions can be triggered by components
- Data-driven systems – action triggered by data update

Repository Architecture

Example: IDE



Repository Architecture

Pros:

- Components can be independent
  - don't need to know about other components
- Changes made by one component – propagated to all components
- Data can be managed consistently – centralization

Cons:

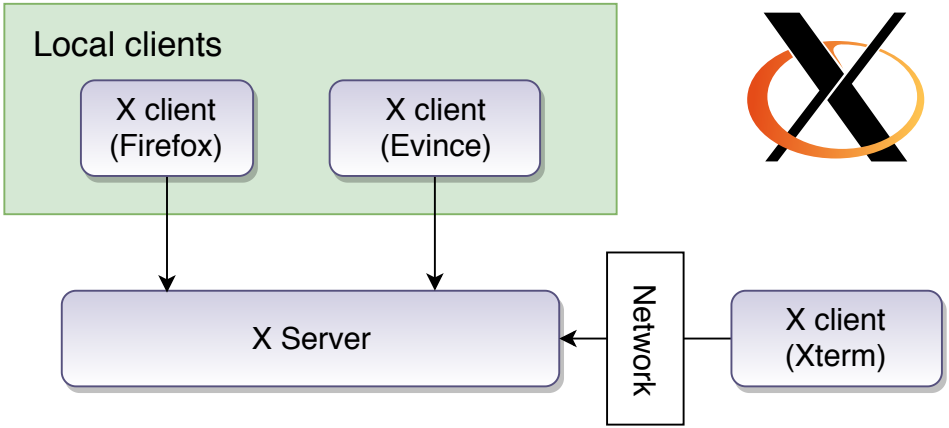
- Repository = single point of failure
- Problems in repository affect the whole system
- Possible inefficiencies in organizing *all* communication through repository
- Distributing the repository – problematic

Client-Server Architecture

Major components:

1. Server(s)
   - Offer services to other components.
   - Examples: print server, file server, compile server
2. Client(s)
   - Connect to server to use service
   - Normally: several instances of a client program on different computers
3. Network
   - Allows clients to access services
   - Most client–server systems: implemented as distributed systems

Client-Server Architecture



Can be distributed but also running on a single machine

Client-Server Architecture

Pros:

- Servers can be distributed across a network
- General functionality (e.g., printing) – available to all clients from a single server

Cons:

- Each service: single point of failure
  - DoS or server failure
- Performance may be unpredictable
  - depends on the network AND the system
- Possible management problems –

Pipe and Filter

Data processing in a system:

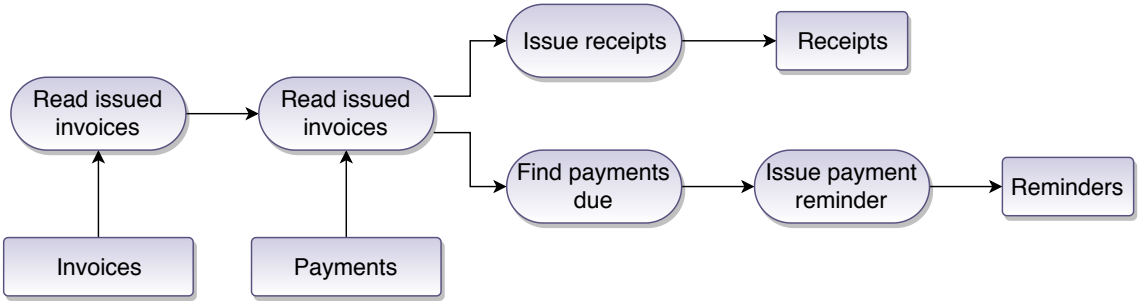
- Each processing component (filter): discrete and carries out one type of data transformation
- Data flows (as in a pipe) from one component to another for processing
- Commonly used for batch and transaction-based data processing applications
- Inputs processed in separate stages to generate related outputs
- Can be sequential, concurrent, coroutines…

Pipe and Filter

Example: Compiler

Pipe and Filter

Example: Business Batch Processing



Pipe and Filter

Pros:

- Easy to understand
- Supports transformation reuse
- Workflow style matches structure of business processes
- Evolution by adding transformations straightforward
- Can be implemented as sequential or concurrent system

Cons:

- Buffering: overflows
- Deadlocks
- Pipes allowing only one data type – filters need to do parsing – slowdowns

Summary

- Software architecture ≈ description of how a software system is organized
- Architectural patterns – means of reusing knowledge about generic system architectures
- Layered Architecture, Repository, Client-server, Pipe and Filter – common patterns