Introduction

Overview

Enterprise Applications

- Persistent data - for years
- LOTS of data
- Many concurrent users
- Lots of user interface screens
- Integrates with other systems
- Kinds of Enterprise Systems
  - B2C
  - B2B
  - Single business information systems
  - Are enterprise applications limited to "business" systems?

Examples?

Come up with as many examples of enterprise systems as you can

Life Within a Business

- Conceptual dissonance
  - between business processes and the software
  - between business employees' expectations and developers' goals
- Business processes change quickly
- Business processes may be inconsistent across the company
- Business logic is often not logical and out of our control

Design Philosophy

In SWE200, you learned a bunch of design adages like "prefer composition over inheritance." What adage applies to business logic?

Considerations

- Sometimes performance matters
- Almost impossible to design it in
- Upgrading anything can change performance (up or down)
- Old optimizations can become degradations

Performance Definitions

Response time	the time it takes the system to process a request
Responsiveness	the time it takes the system to respond to a request
Latency	the minimum time required to get a response even if no work is required
Throughput	the amount of work that can be done in a given amount of time
Load	how much stress the system is under
Load sensitivity	how response time varies with load
Efficiency	performance divided by resources
Capacity	maximum effective throughput or load

Two truths and a lie

The user can live with lower response time as long as responsiveness is high. Load sensitivity is a constant for a given system. Latency is affected by the connectivity of our network.

Scalability

- Definition: how adding resources affects performance
- Vertical Scalability - adding power to existing servers
- Horizontal Scalability - adding more servers to our system

Discuss

Does the design of the software affect horizontal or vertical scalability?

SWE200

- What is the definition of a design pattern?
- How is this class different?
  - Patterns are more complex
  - Selection of patterns depends on physical architecture of the system
  - Selection of a pattern in one portion of the system affects the choice of patterns elsewhere
  - One pattern often includes other patterns

Everything's Connected!

- Website to help
  - web.cs.ship.edu/~merlin/lsa
  - username: lsa
  - password merlin
- Click on Overview in Mapping to Relational Databases
- Each of his narratives has a page (click on any square)
- For each section we will study, diagram of related patterns
  - red: know it WELL
  - blue: know its intent
- Search for a pattern to see all of the places it is referenced

Study Ideas

Discuss strategies for how you can manage this complexity - individually and collectively

Layering?

- Organization of the largest parts of the system
- Like a layered cake
- Each layer rests on a lower layer and uses services provided by that lower layer
- Lower layers are unaware of the layers above them
- Each layer can only see the layer immediately below it

Benefits of Layering

- You can understand each layer in isolation
- You can make substitutions for layers without changing anything else in the system
- You minimize dependencies between layers
- Layers make good places for standardization
  - Which makes people able to develop competing solutions
- One lower layer can support a wide variety of higher-level solutions

Downsides to Layering

- While layers encapsulate somethings well, some changes will ripple down the system
- Extra layers can affect performance

Layering in Code

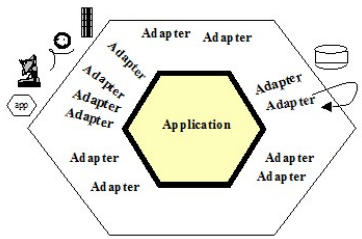
- Very simple: one procedure - but keep the three layers as separate subroutines
- More complex: each layer is a class
- Even more complex: each layer is a package (or two or three)

Translate

Describe exactly what the code for the very simple description would look likd

Alternatives to Layering

- There are many architectural patterns other than layering
- Hexagonal Architecture
  - Everything outside of the core of the system is an interface



- nice symmetry- all things outside the system are equivalent
- However, the user interface that you provide seems different than an interface to another system (a service you provide to something else)
- This asymmetry is even more clear when you think about the data source - it is essentially a system that is providing a service to you, so you would be one of the things requiring an adapter in its hexagonal architecture.

[1] Alistar Cockburn, Web page titled, "The Pattern: Ports and Adapters ("Object Structural")", http://alistair.cockburn.us/Hexagonal+architecture, accessed 7/15/2014.

Classic layers

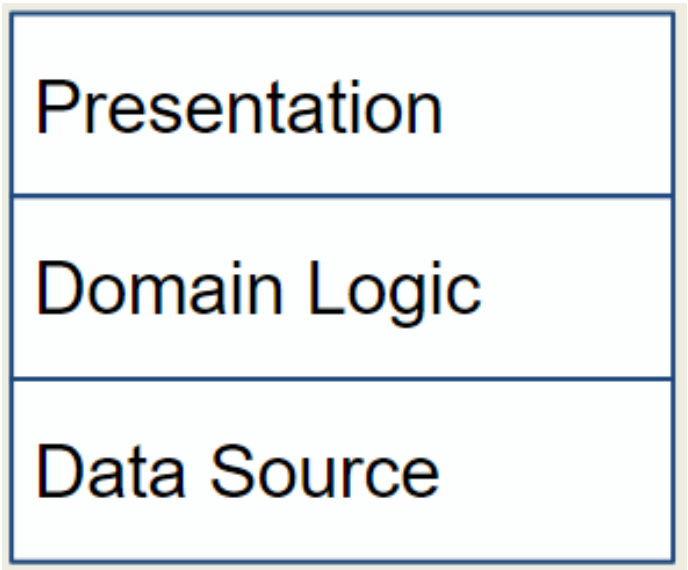


Figure: Classic Layers

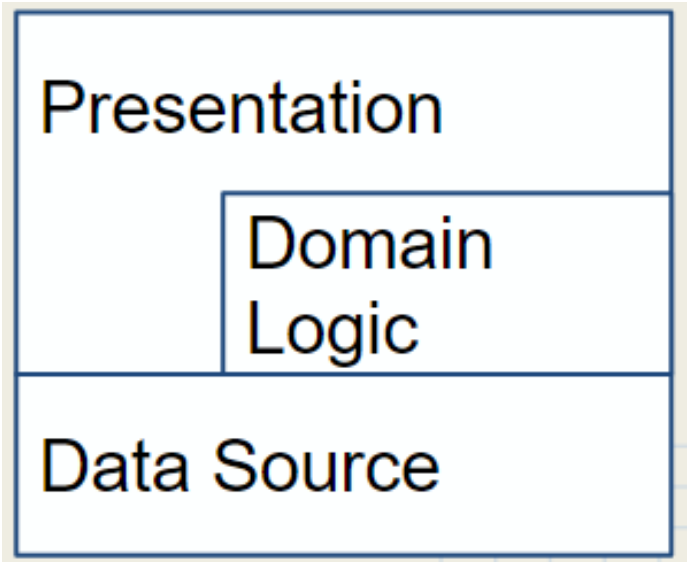


Figure: Sometimes the Presentation layer is given read only access to the data source

Discuss

Why would we limit presentation's interactions with the data source to read only access?

Presentation Layer

- Handles interactions between the user and the software
- aka: User Interface
- Primary Responsibilities
  - Display information to the user
  - Convert request from the user into actions upon the Domain

Kinds of Presentation

Rich Client GUI not in a web browser

Web interface UI in web browser

Command line controller Text commands at a prompt

Programmatic controller No human user at all

Domain Logic Layer

- aka: Business Logic
- Calculations
- Validation of data from presentation
- System behavior

Data Source

- Communication with other systems that carry out tasks for the application
  - Transaction Monitors
  - Other applications
  - Persistent data
  - Messaging systems

Choices

- Server
- Desktop
- Thin client or browser
- Phone

What's the difference between a Tier and a Layer?

Things to Weigh

- Responsiveness
- Server Roundtrips
- Disconnected Operation
- Sharing/Synchronizing of Information

Placement by Layer

- Data Source
  - Almost always on a server (lots of data)
  - When disconnected, can substitute a temporary local data source
- Presentation
  - Driven by the type of UI you want
  - Rich client on the client machine
  - Web interface on the server
- Domain Logic
  - Either on the client or the server
  - Splitting it between the two is complicated

Complexity Boosters - AVOID!

- distribution
- explicit multi-threading
- paradigm chasms (OO vs. RDMS vs. scripting)
- multiplatform development
- extreme performance requirements

Each comes at a high cost to complexity and therefore development and maintenance times