System Analysis and Design

Software Architecture Design

Salahaddin University
College of Engineering
Software Engineering Department
2011-2012

Amanj Sherwany
http://www.amanj.me/wiki/doku.php?id=teaching:su:system_analysis_and_design

The Mission

"Build a user friendly web-front for our 250 mainframe systems"

The Project

- A large Iraqi financial institution was opening a new customer phone support department.
- They wanted more user friendly tools than the 3270-based they normally used.
- The front-end should reflect the functions of the back-end, i.e., be a façade for the mainframe systems. *(functional requirement)*

The IBM 3270

The Basic Use Case

- A customer calls the phone support service and authenticates using a Customer-ID and PIN code.
- A new call case is created in the CRM system.
- The phone support clerk picks up the call.
- The Customer-ID is transferred to the xFront application and the total customer engagement is fetched from the back-ends while the clerk greets the customer.

The Basic Use Case, *Cont'd*

- The customer and the clerks handle the customers problem and every action done by clerk on the customers behalf is logged in the CRM case.
- The customer hangs up.
- The clerk writes a call summery in the CRM case and closes the case.

Forces

- Worker Ergonomics *(non-functional requirements)*
  - The phone service personnel spends 40 hours per week in front of xFront in a tightly time passed environment.
- Performance *(non-functional requirements)*
  - More than three seconds waiting time for use data to appear is experienced as an unnecessary delay.
- Scalability *(non-functional requirements)*
  - The back-end computers use old and esoteric protocols (SNA LU 6.2 and LU/2, Univac uts, …) and are resource constrained.

Forces, *Cont'd*

- Maintainability *(non-functional requirements)*
  - The data models in the mainframe systems does not reflect the current business model and their current views.
- Flexibility *(non-functional requirements)*
  - The mainframes are due for a major overhaul, the entire retail banking system are being renewed.
- Concurrency Problems *(non-functional requirements)*
  - Data about a customer can be updated, simultaneously, both from the xFront and from the old 3270 applications.

Non-Functional Requirements

- Performance
- Scalability
- Maintainability
- Flexibility
- Reliability
- Security

Ilities (from Wikipedia)

Within systems engineering, **-ilities** are aspects or non-functional requirements. They are so-named because most of them end in "-ility."

A subset of them (Reliability, Availability, Serviceability, Usability, and Installability) are together referred to as RASUI.

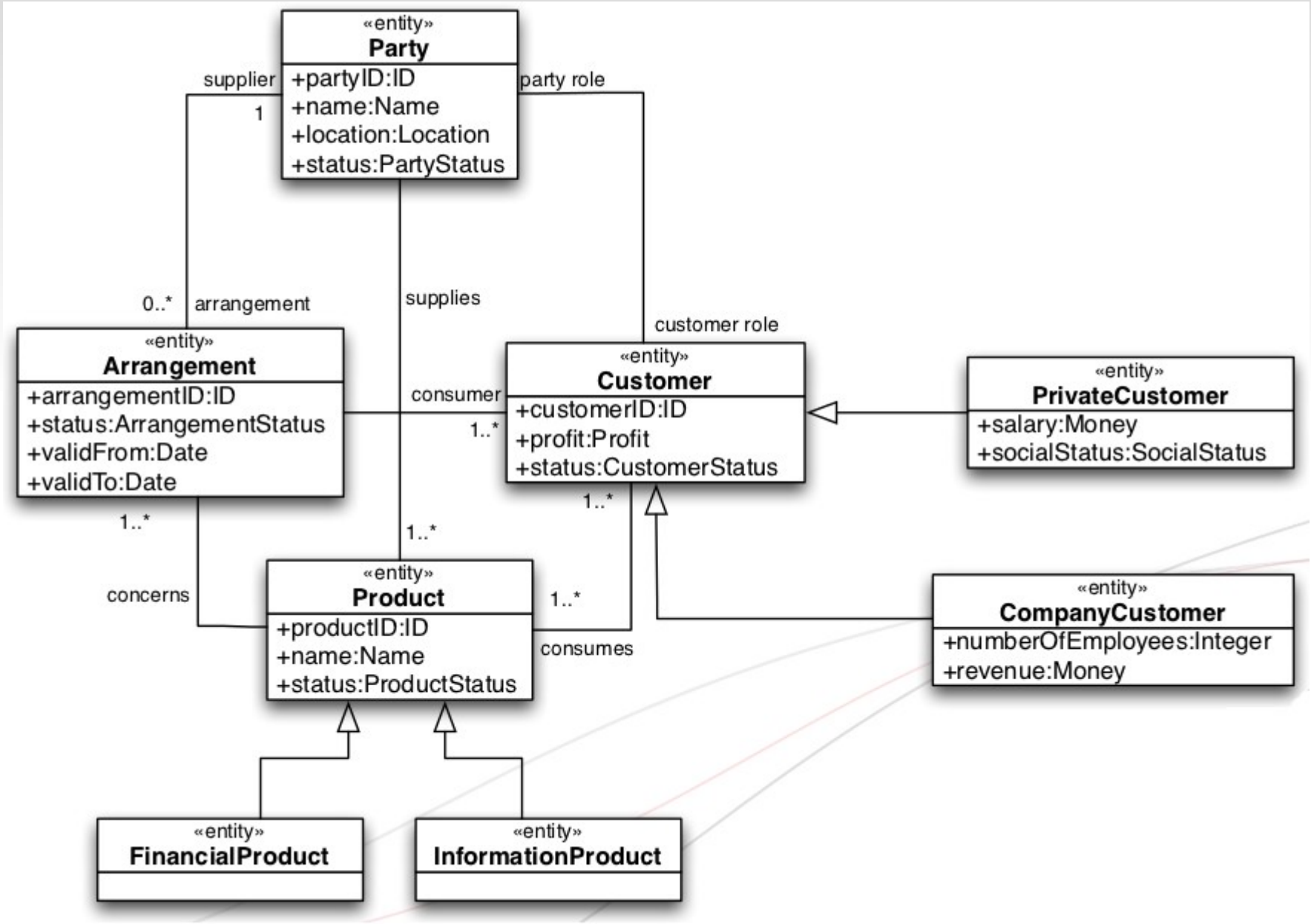
For databases RASR is an important concept (Reliability, Availability, Scalability, and Recoverability). The "-ilities" often include:

- accessibility
- accountability
- adaptability
- administrability
- affordability
- agility
- availability
- composability
- configurability
- customizability
- degradability
- demonstrability
- dependability
- deployability
- distributability
- durability
- evolvability
- extensibility
- flexibility
- installability
- interoperability
- maintainability
- manageability
- mobility
- nomadicity
- operability
- portability
- predictability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- reusability
- scalability
- seamlessness
- serviceability (a.k.a. supportability)
- securability
- simplicity
- stability
- survivability
- tailorability
- testability
- timeliness
- understandability
- usability
- haniability

Decisions

- Domain Model
  - The system should use a common domain model based on Financial Services Object Model (FSOM)
  - The domain model should be implemented in Plain Old Java Objects (POJO) so they can function as the model objects in the client MVC model.
  - The same model object implementations should be used in the client and the server

Domain Model



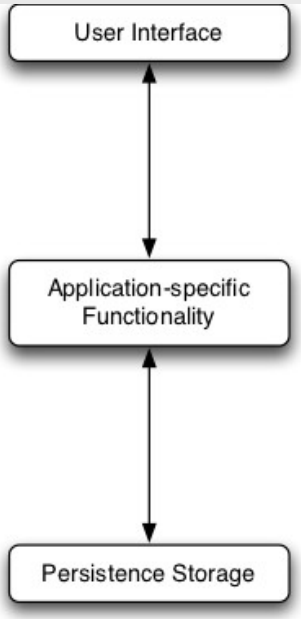
Decisions, *Cont'd*

- Architecture
  - The basic architecture should be a three-tiered in order to minimize the traffic and connections to the mainframes.
- Client
  - The client should be a Java/Swing program in order to enable a rich and ergonomic user interaction.

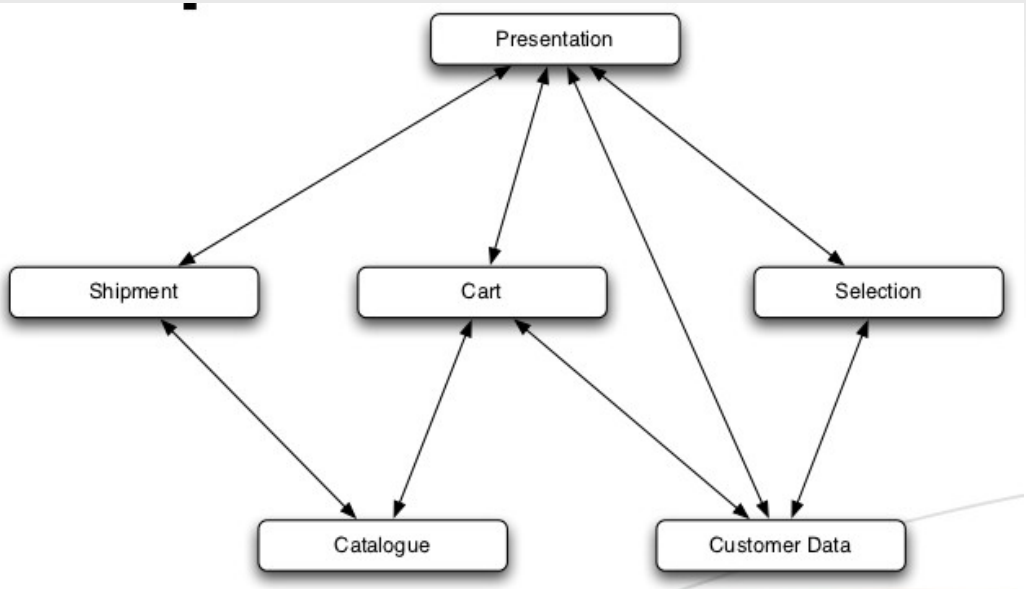
What is Software Architecture?

- "The **decomposition** of a system into a set of **modules** and interconnecting these modules"
- "The description of **elements** from which system are built, **interactions** among those elements, **patterns** that guide the composition, and **constraints** on these patterns"
- "...deals with the design and implementation of the **high-level structure** of a software"

Example



Architecture pattern (3-tier)



Reference model for on-line shopping

The Basic Truths

- Proper design of architecture is crucial to fulfilling the non-functional requirements
  - A job well-done makes this possible,
  - But does not guarantee anything
- All possible non-functional requirements cannot be addressed by the software architecture

Four Popular Heuristics

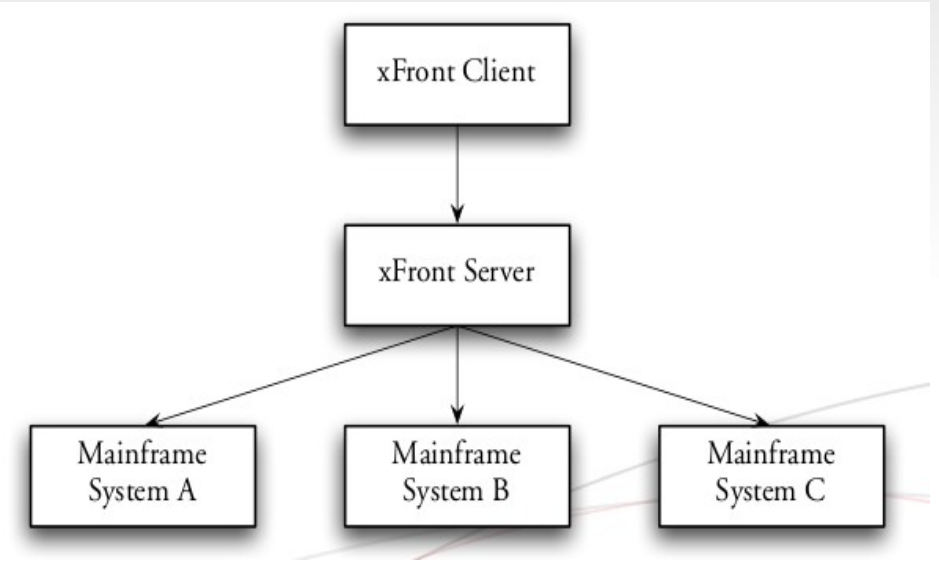
- Don't assume that the original statement of the problem is necessarily the best, or even the right one
- In partitioning, choose the elements so that they are as independent as possible (=low external and high internal complexity)
- Simplify, simplify, simplify
- Build in and maintain option as long as possible in the design and implementations of complex systems –you will need them

Decisions, *Cont'd*

- Server
  - The server should abstract the differences between the back-end systems in order to be able to replace them with minimal system impact.
  - The server should use lazy evaluation of the domain model object in order to conserve back-end bandwidth.
  - The server should use standard connection techniques for different kinds of back-ends in order to be comprehensible and/or natural for subject matter experts.

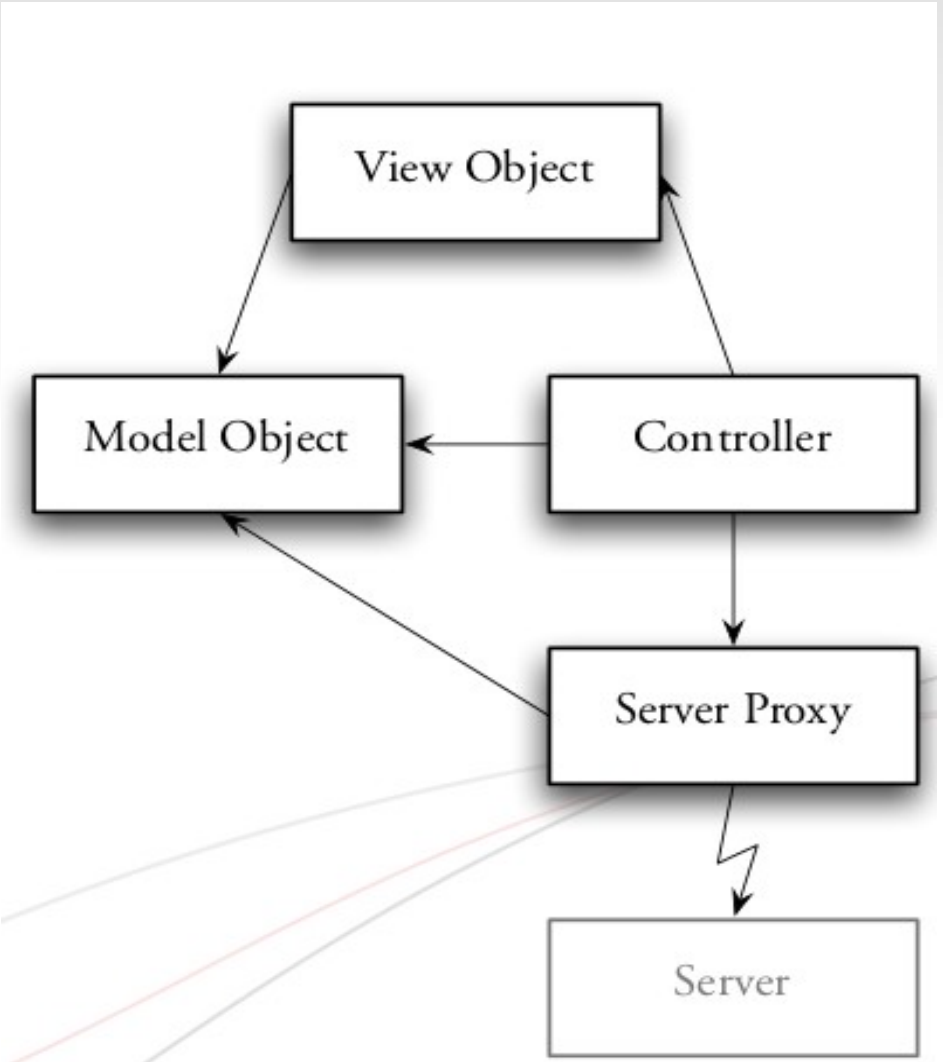
Basic Architecture

- Three tier in order to offload the resource constrained back-end systems.
- Java/Swing client in order to ensure ergonomics quality.



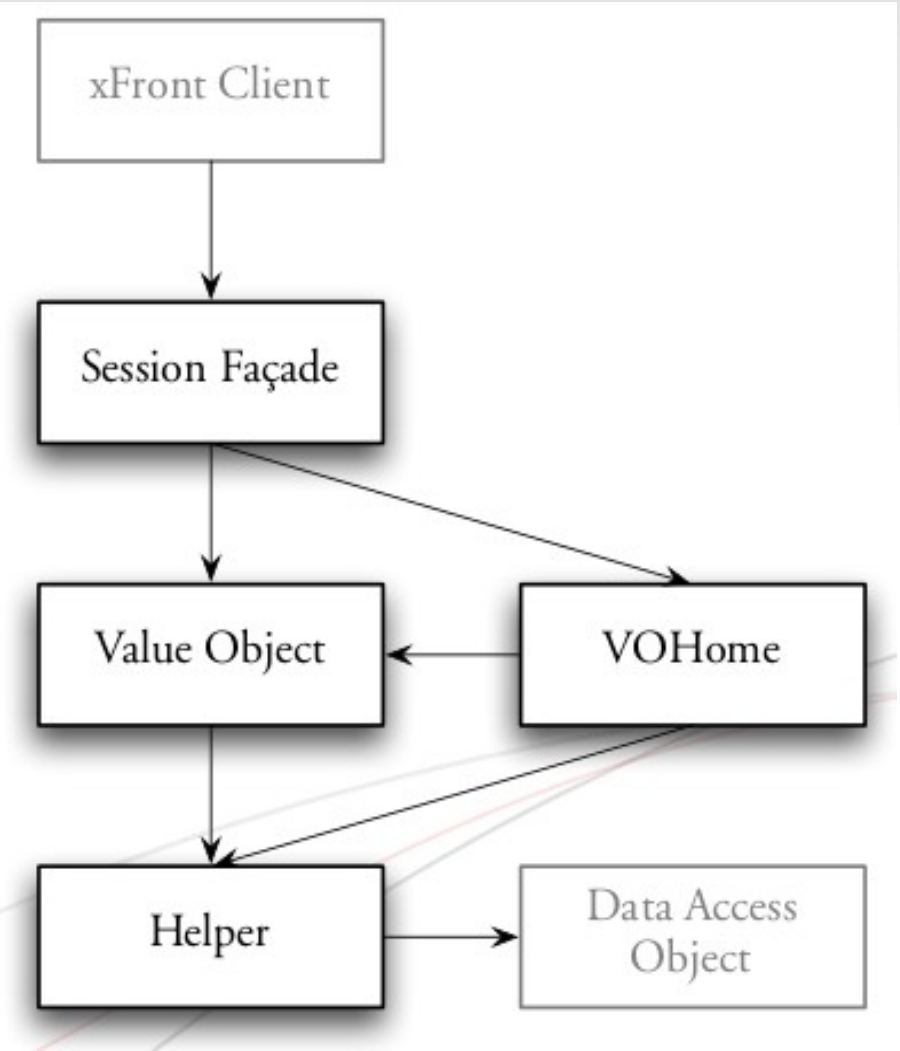
Client Architecture

- Model/View/Controller architecture since Swing is MVC.
- A server proxy abstracts the network communication and exposes one method per server interface function.
- POJO acts as model in the MVC.



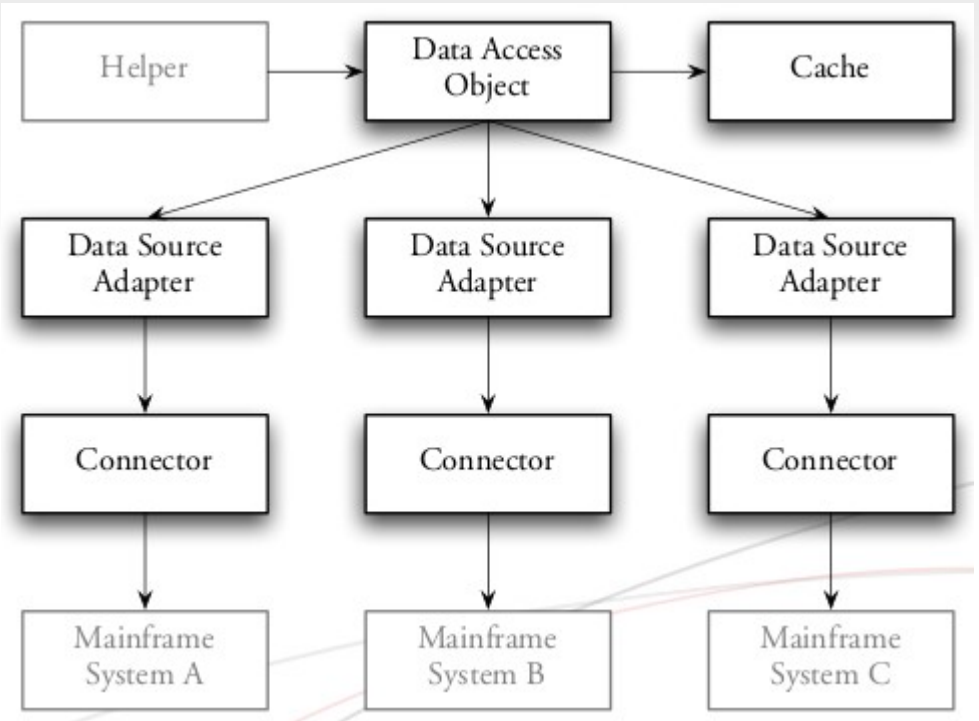
Server Architecture, Business Layer

- Session Façade to abstract the business layer in terms of use cases.
- Java Value Object as Domain Model implementation.
- Value Object Home as object factory.
- Value Object Helpers to do lazy evaluation.



Server Architecture, Integration Layer

- Data Access Objects to abstract away data source.
- Cache to minimize back-end bandwidth requirements.
- Data Source Adapters to encapsulate back-end knowledge.
- Standard Connectors for least surprise to subject matters experts.



References

- This presentation is almost a direct copy/paste from a keynote given by Thorbiörn Fritzon, Systems Architect at Oracle (then Sun) at Uppsala University.