Overview of Architecture (based on Larman)

- Have looked over Larman's approach to Software Design Patterns, based on underlying ideas/concepts
- Now follow these ideas for the architecture, and some basic architectural patterns

High-level Architecture

- Transition from
  - Analysis (black-box view)

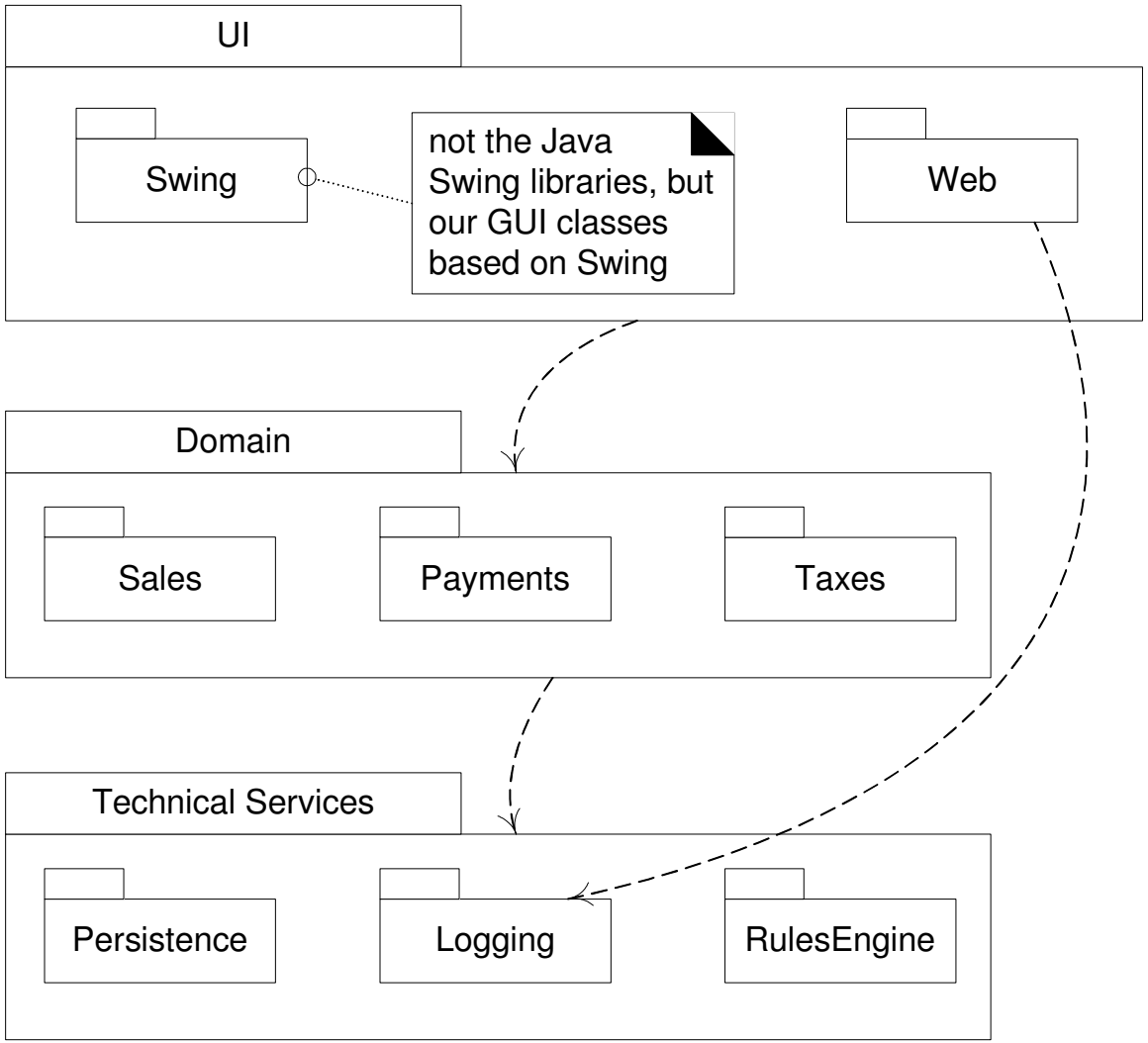
  to
  - Design
- Look at large scale design
- High level Logical Architecture
- Describe using UML Package Diagram
- Examples:
  - Layers Pattern, Model-View-Controller

Software Architecture

A definition from the UML User Guide**:**

An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition.

Example Illustrated using Package Diagram



UP Architecture (Larman)

The UP considers 6 architectural views.

One could consider these as just different views of the model, and the Logical Architecture being what is usually considered Software Architecture, and Deployment and Implementation the more concrete implementation views of same.

Logical

Process

Deployment

Data

Use Case

Implementation

UP Architecture (Larman)

1. Logical

o Conceptual organization of the software in terms of the most important layers, subsystems, packages, frameworks, classes, and interfaces. Also summarizes the functionality of the major software elements, such as each subsystem.

o Shows outstanding use-case realization scenarios (as interaction diagrams) that illustrate key aspects of the system.

o A view onto the UP Design Model, visualized with UML package, class, and interaction diagrams.

2. Process

o Processes and threads. Their responsibilities, collaborations, and the allocation of logical elements (layers, subsystems, classes, ...) to them.

o A view onto the UP Design Model, visualized with UML class and interaction diagrams, using the UML process and thread notation.

3. Deployment

o Physical deployment of processes and components to processing nodes, and the physical network configuration between nodes.

o A view onto the UP Deployment Model, visualized with UML deployment diagrams.

Normally, the "view" is simply the entire model rather than a subset, as all of it is noteworthy.

UP Architecture (Larman)

4. Data

o Overview of the persistent data schema, the schema mapping from objects to persistent data (usually in a relational database), the mechanism of mapping from objects to a database, database stored procedures and triggers.

o A view onto the UP Data Model, visualized with UML class diagrams used to describe a data model.

5. Use case

o Summary of the most architecturally significant use cases and their non-functional requirements. That is, those use cases that, by their implementation, illustrate significant architectural coverage or that exercise many architectural elements. For example, the *Process Sale use case, when fully* implemented, has these qualities.

o A view onto the UP Use-Case Model, expressed in text and visualized with UML use case diagrams.

6. Implementation

o First, a definition of the Implementation Model: In contrast to the other UP models, which are text and diagrams, this "model" *is the actual source code, executables,* and so forth. It has two parts: 1) deliverables, and 2) things that create deliverables (such as source code and graphics). The Implementation Model is all of this stuff, including web pages, DLLs, executables, source code, and so forth, and their organization—such as source code in Java packages, and bytecode organized into JAR files.

o The implementation view is a summary description of the noteworthy organization of deliverables and the things that create deliverables (such as the source code).

UP Architecture

Important views

- Logical Architecture
- Deployment Architecture

Logical Architecture

- Logical architecture
  - More abstract than realization in software
- Architectural patterns
  - Higher-level than software design patterns
- Layers
  - A grouping of software elements (e.g. classes, packages, subsystems) that has cohesive responsibility for a major aspect of the system

Logical Architecture

- Example: Layers pattern
  - Layered architecture
  - Organise into discrete layers with distinct related responsibilities
  - Communication and coupling from higher to lower layers, avoid lower to higher coupling

Over 100 design patterns that are variations of the layers pattern listed by Linda Rising

Layers

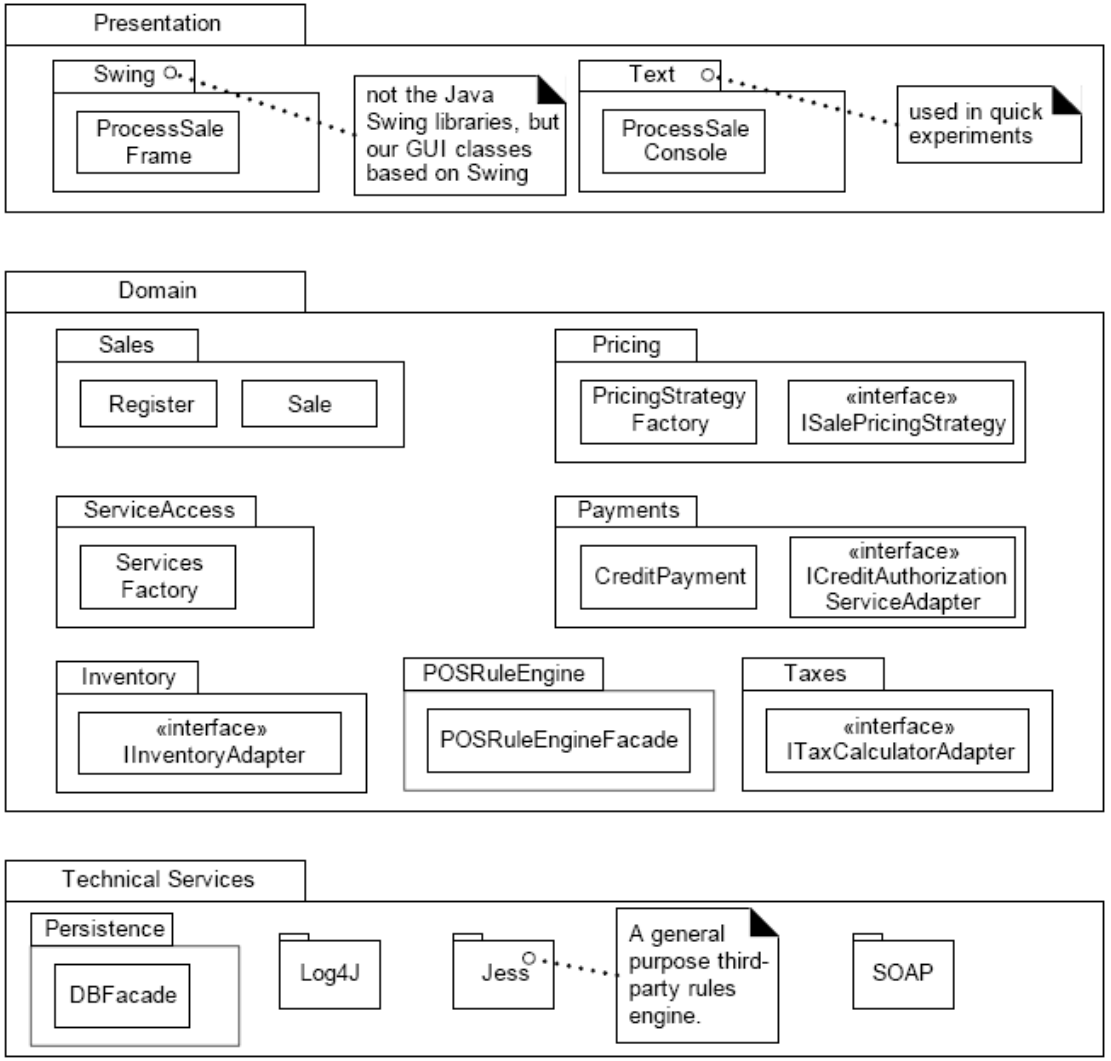
Typical layers in object-oriented system:

- User Interface
- Application Logic and Domain Objects
  - Application specific software
- Technical Services
  - Usually application independent and reusable

Application/Domain Layer

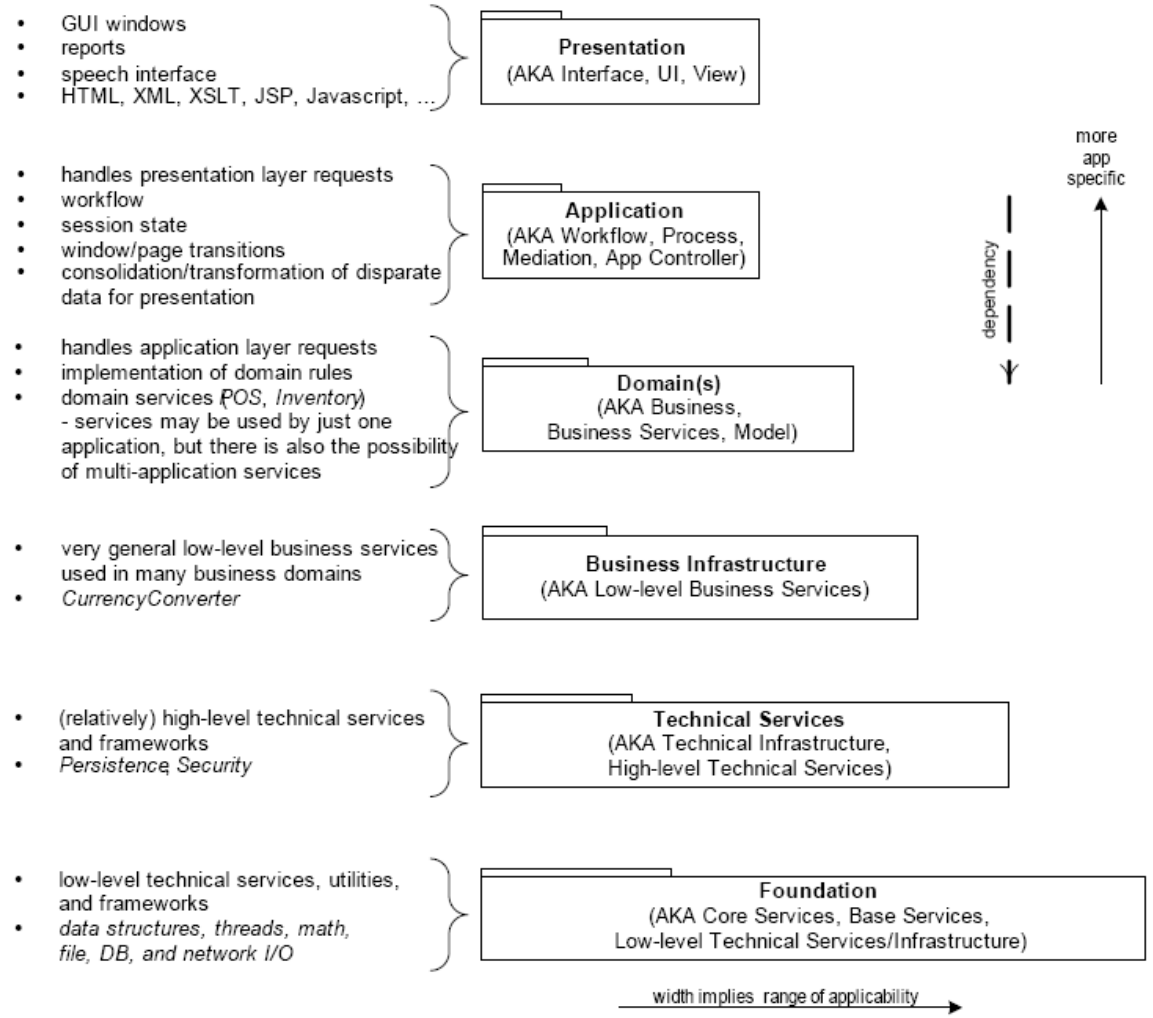
- If the **application layer** contains mostly domain classes, i.e. software classes that correspond to real-word domain, then Larman suggests this be called the **domain layer**
- Note the Domain Model describes *real-world concepts* whereas the domain layer describes *software classes* in the Design Model

Main Layers Example



Layers Example

Typical Layers of an information system (Larman):



Expanded view

- GUI windows
- reports
- speech interface
- HTML, XML, XSLT, JSP, Javascript, ...

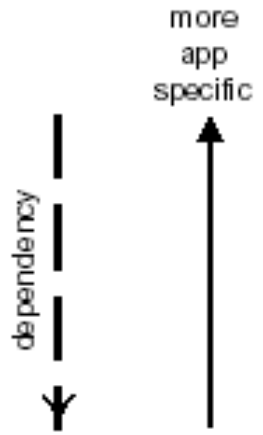
Presentation
(AKA Interface, UI, View)

- handles presentation layer requests
- workflow
- session state
- window/page transitions
- consolidation/transformation of disparate data for presentation

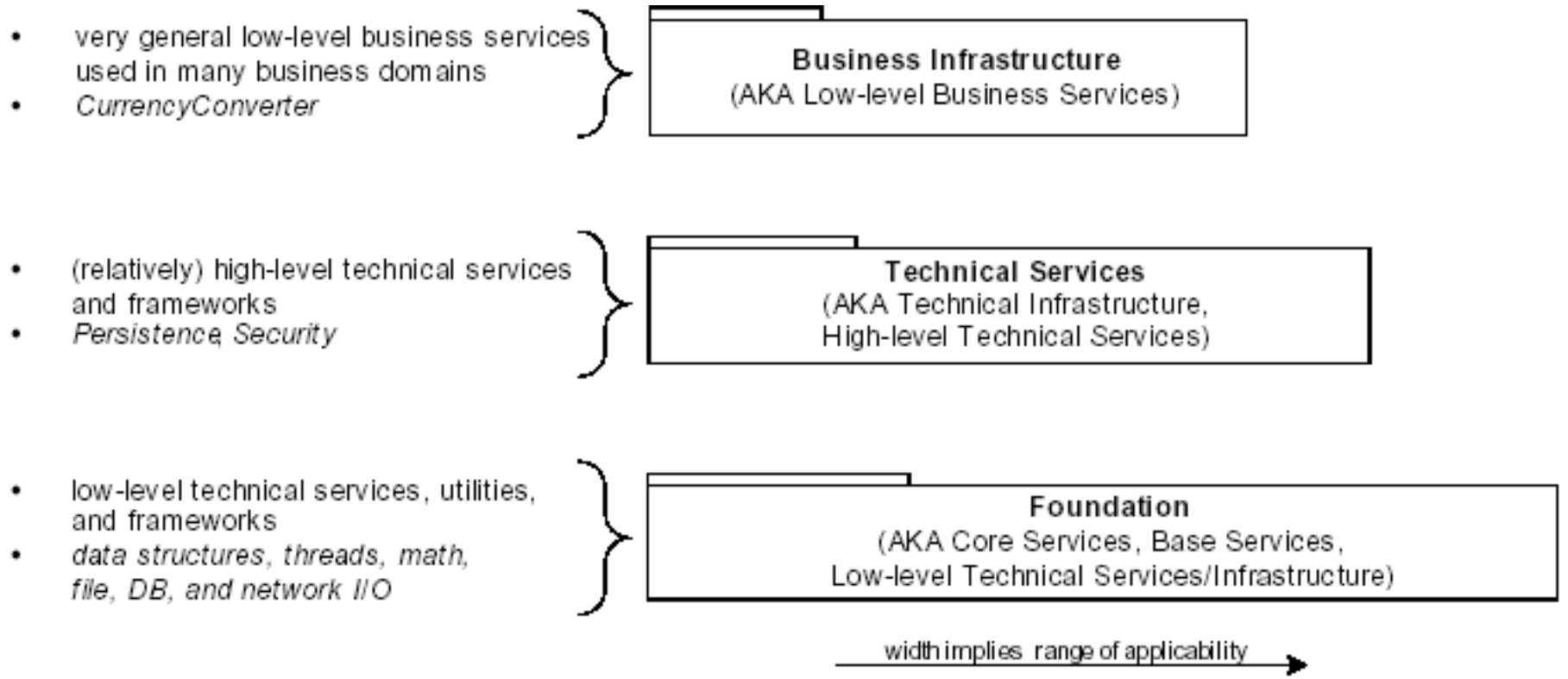
Application
(AKA Workflow, Process, Mediation, App Controller)

- handles application layer requests
- implementation of domain rules
- domain services (*POS*, *Inventory*)
  - services may be used by just one application, but there is also the possibility of multi-application services

Domain(s)
(AKA Business, Business Services, Model)



Expanded View

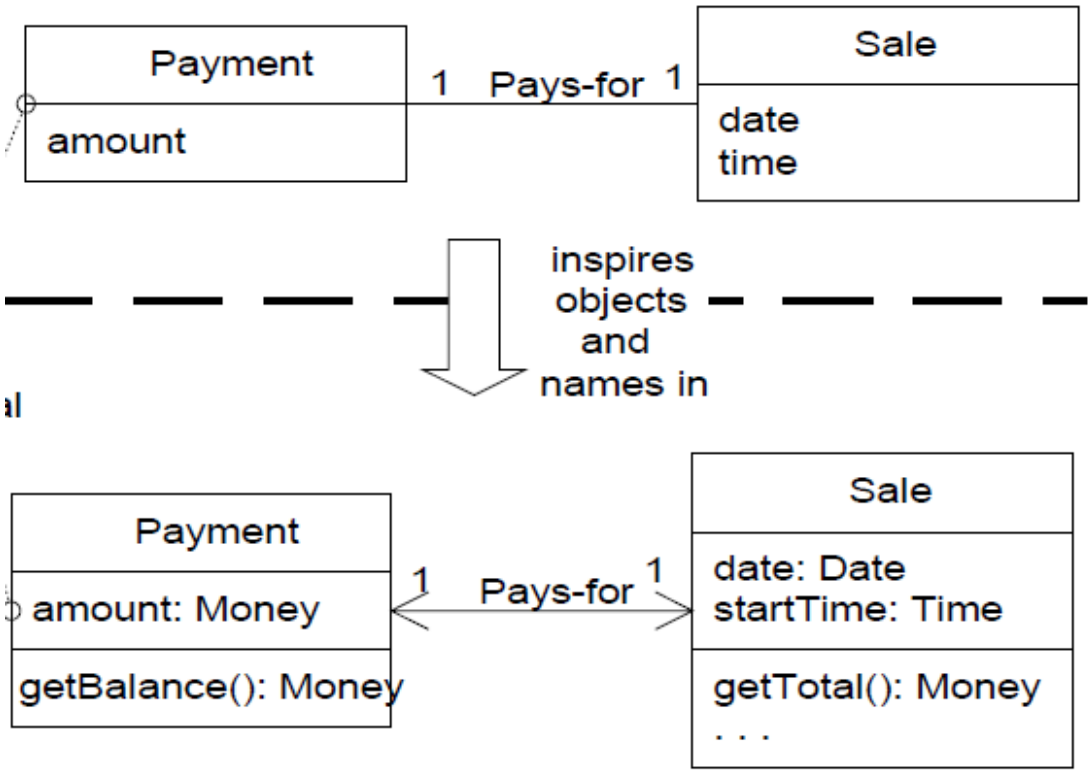


Layers Pattern Benefits

- Separation of concerns: from high to low level services; application specific to general
- Related functionality and complexity encapsulated
- Layers can be replaced (e.g. new UI layer)
- Lower layers (or parts) can be reusable

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.



UP Design Model

Model-View Separation Principle

- Important layer separation principle
- Based on original Model-View-Controller architectural pattern as used in Smalltalk-80 by Alan Kay
- Term “model” here refers to domain layer and would originally correspond more or less to the data objects in the system (i.e. the data model of the information system)

Model-view separation

Principle

- The model(domain) objects should not have direct knowledge of view(presentation) objects.
- the domain classes should encapsulate the information and behavior related to application logic.

Guideline: Model-View Separation

- Do not connect or couple non-UI objects directly to UI objects
  - decoupling
  - reuseability
- Do not put application logic in the UI object methods
  - cohesion
  - should delegate application requests to non-UI objects

Model-View Separation

- supports cohesive model definitions that focus on the domain process, rather than on interfaces.
- allows separate development of the model and user interface layers.
- minimizes the impact of requirements changes in the interface upon the domain layer.
- allows new views to be easily connected to an existing domain layer, without affecting the domain layer.
- allows multiple simultaneous views on the same model object.
- allows execution of the model layer independent of the user interface layer
- allows easy porting of the model layer to another user interface framework.

Architectural Design

Summary:

- Overview of Architectural Design
- General Layers Pattern
- Model-View Separation

Next: closer look at architectures, distributed architecture models