Logical Architecture and UML Package Diagrams

Software Architecture

- A set of significant decisions about:
  - Organization of a software system
  - Selection of structural elements
  - Interfaces between elements
  - Behavioral collaboration between elements
  - Composition of elements into subsystems
  - Architectural “style” of organization

Logical Architecture and UML Package Diagrams

- High level, large scale Architecture Model.
- At this level, the design of a typical OO system is based on several architectural layers, such as a UI layer, an application logic (or “domain”) layer, and so forth.
- Goal is to design a logical architecture with layers and partitions using UML package diagrams

Logical Architecture And Layers

- Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.
- Logical - because there’s no decision about how these elements are deployed across different operating system processes or across physical computers in a network (deployment architecture).

Layering Pattern

- A layer is a very coarse-grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system.
- Also, layers are organized such that “higher” layers (such as the UI layer) call upon services of “lower” layers, but not normally vice versa.
- Strict layered architecture vs. Relaxed Layered Architecture
- A logical architecture doesn’t have to be organized in layers. But it’s very common, and hence, introduced at this time.

Why use Layers

- Source code changes are rippling throughout the system many parts of the systems are highly coupled.
- Application logic intertwined with UI, neither reusable
- Potentially general technical services or business logic is intertwined with more application-specific logic, so it cannot be reused, distributed to another node, or easily replaced with a different implementation.
- High coupling across different areas of concern. It is thus difficult to divide the work along clear boundaries for different developers.
- The purpose and number of layers varies across applications and application domains (information systems, operating systems, and so forth).
Typically layers in an OO system

- User Interface.
- Application Logic and Domain Objects software objects representing domain concepts (for example, a software class Sale) that fulfill application requirements, such as calculating a sale total.
- Technical Services general purpose objects and subsystems that provide supporting technical services, such as interfacing with a database or error logging. These services are usually application-independent and reusable across several systems.

Common Layers

Package Coupling

NextGen Example

- No classes, just packages and package dependencies
- Most common form
Architectural Interaction Diagram

[Larman 2002]