Logical Architecture + Physical Architecture (Deployment Diagrams)

Bennett p. 348-356

Larman 37.1

System Architecture

- A major part of system design is defining the ***system architecture***
- The architecture of an information system is a ***high-level view*** (the big picture) of the system.
- The architect must consider the non-functional requirements
- The architect groups classes together into ***subsystems*** and model the system as a set of interacting components.
- It is called the logical architecture because there is no decision about how these elements are deployed across different operating system processes or across physical computers in a network

Subsystems

- A subsystem typically groups together elements of the system that share some **common properties**

Examples:

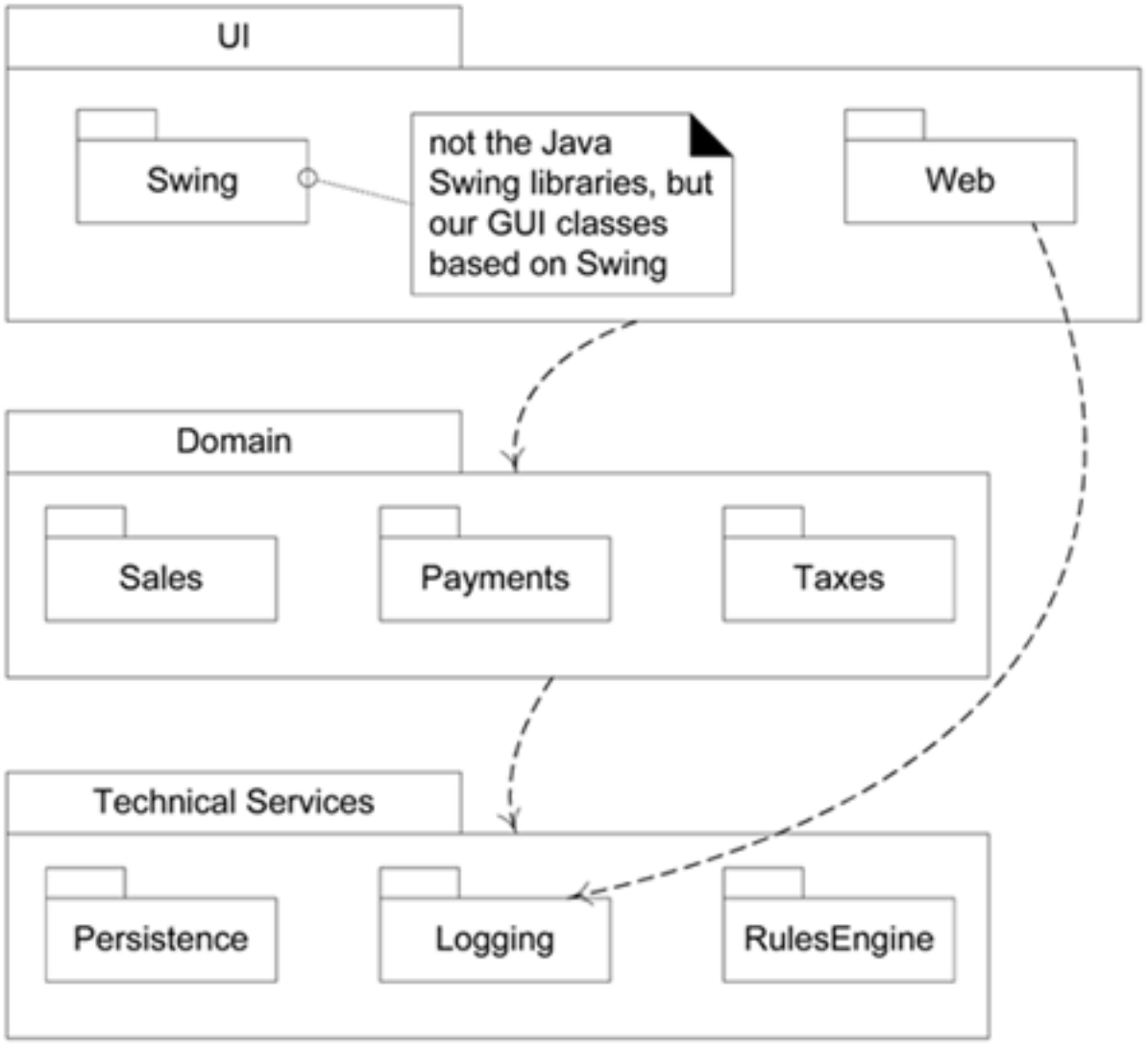
- User interface subsystem
- Database subsystem

The sub-division of an information system into sub-systems has the following advantages:

- **It produces smaller units of development**
- **It helps to maximize reuse**
- **It helps the developers to cope with complexity**
- **It improves maintainability**

Example

A logical architecture using a UML package diagram.



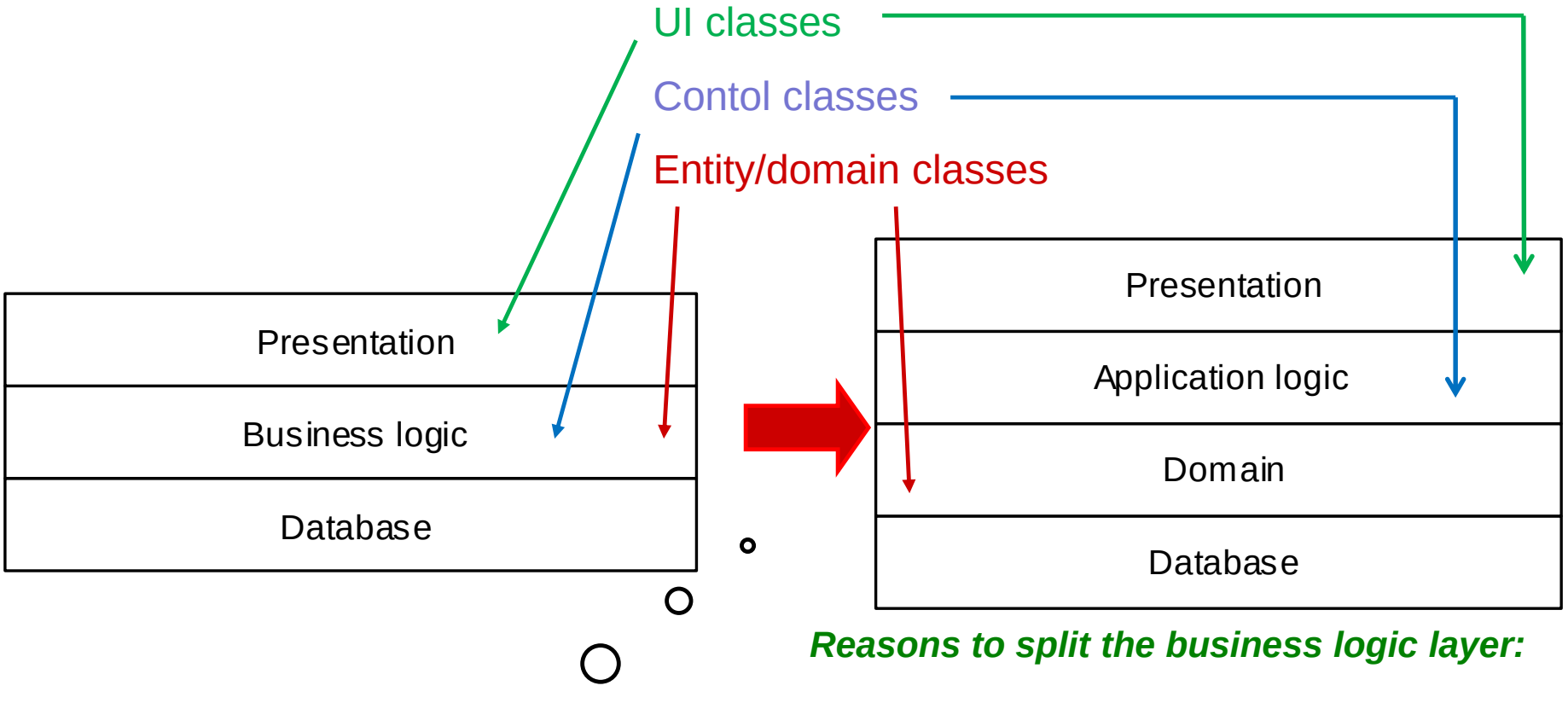
Layering and Partitioning

Two general approaches to the division of a software system into subsystems:

- **Layering (open and closed)**
- **Partitioning**

Both approaches are often used together on one system

Three & Four Layer Architectures



Business logic layer

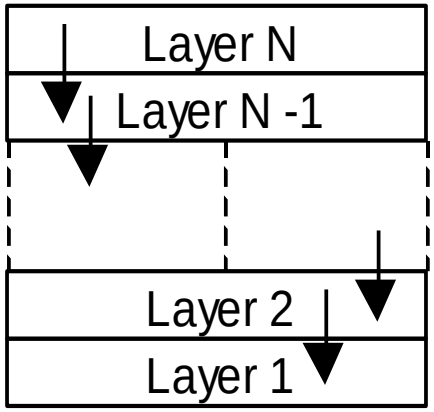
can be split into two layers

Reasons to split the business logic layer:

- to avoid that the entity/domain classes gets to much responsibility
- More subsystems can share the same entity classes

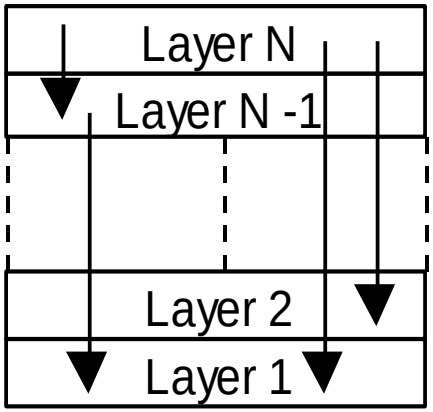
Layered Architecture

Larmans strict architecture



Closed architecture—messages may only be sent to the adjacent lower layer.

Larmans relaxed architecture

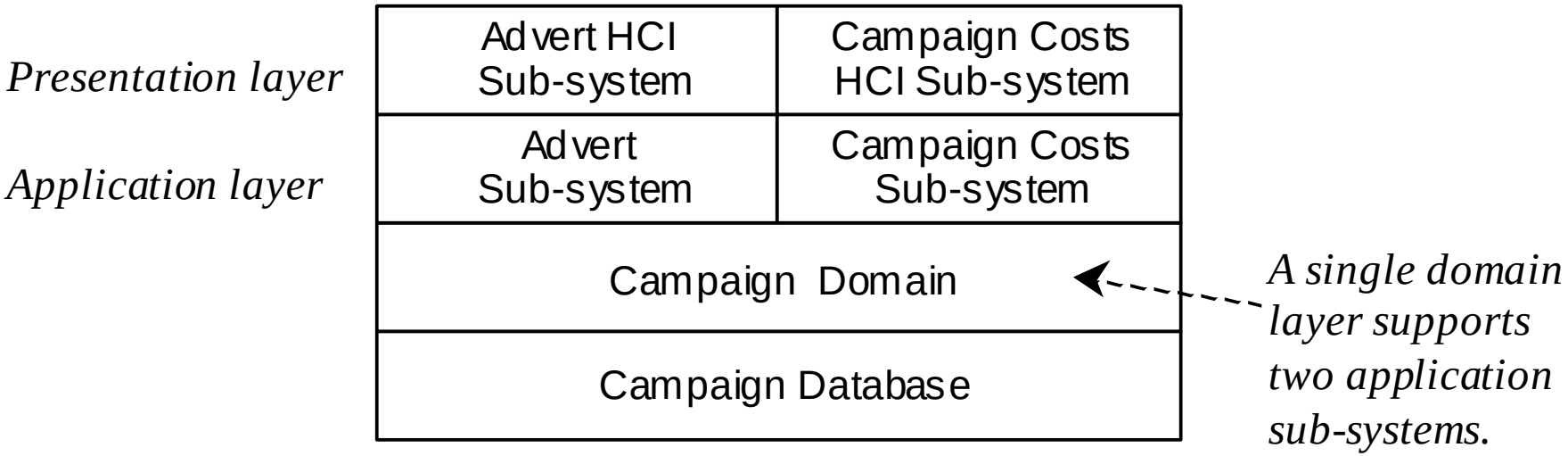


Open architecture—messages can be sent to any lower layer.

Minimizes the interdependency between layers.
Leads to more complicated program structures but ensures low coupling

More effective programs, but produce high coupling and more compact code

Partitioned Subsystems

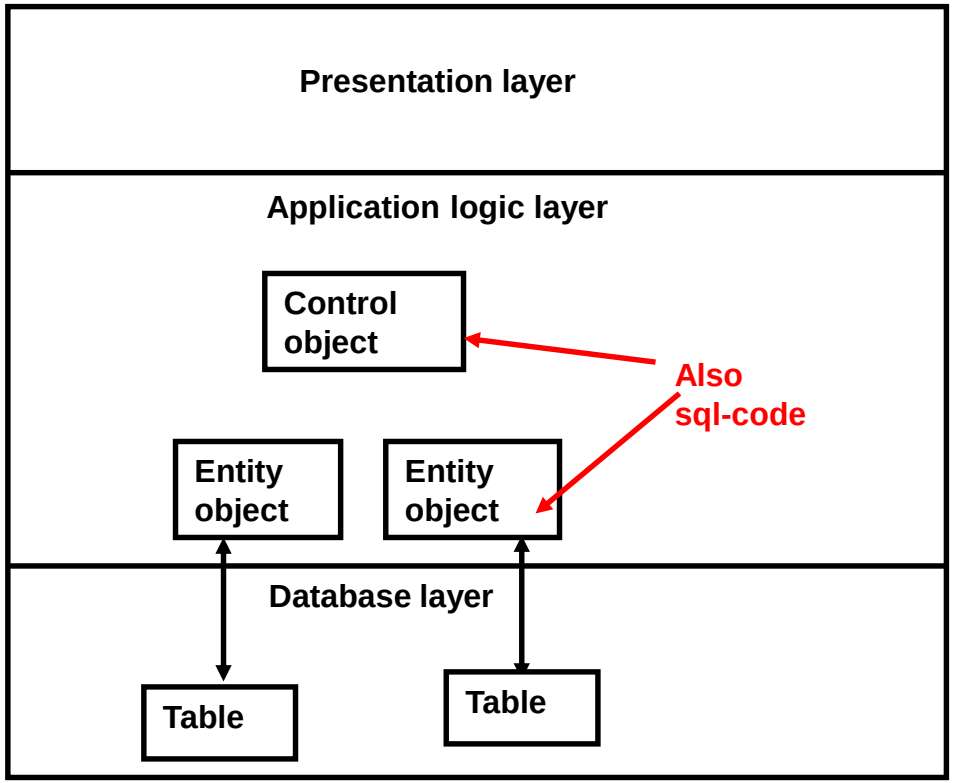


- **Each sub-application has its own presentation layer**
- **The Application layer is split into 2 subsystems serving its own sub-application**
- **The Campaign Domain-layer updates and read data from the Campaign database and provide services to the layers above.**
- **Note that the sub-applications share the domain-layer.**

Direct mapping

If entity/domain classes or control classes contain code, who read or store data in a database then it is called direct mapping

This is the case if the methods of an object contain SQL-statements.



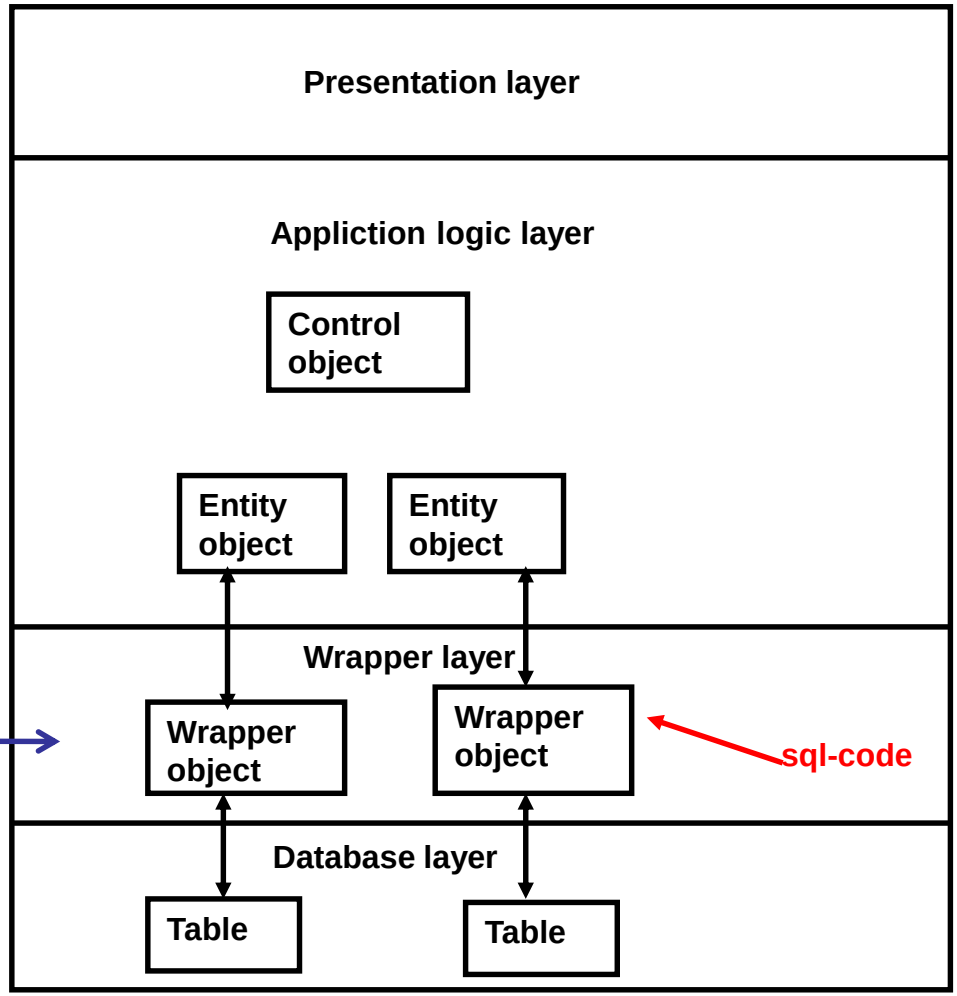
Benfits:
Easy to implement

Draw-backs:
Produces high coupling and low cohesion since the entity classes are responsible for classes in other layers

Indirect mapping

Every entity class has its own wrapper class, which contains calls to the database or files.

- This is the case if an object calls its wrapper class, who contains SQL-calls to at table in the database.



Pure fabrication and Indirection pattern

Benefits:
Produces low coupling and high cohesion since only the wrapper layer is responsible for creating access to the persistent

Draw-backs:
An extra layer add to the complexity of the architecture

Benefits of a layered architecture

- Separation of concerns: a separation of high from low-level services, and of application-specific from general services. This **reduces coupling** and dependencies, **improves cohesion**, increases reuse potential, and increases clarity.
- Related complexity is encapsulated
- Some layers can be replaced with new implementations.
- Lower layers contain reusable functions.
- Development by teams is aided because of the logical segmentation.

Guidelines

- **The responsibilities of the objects in a layer should be strongly related to each other and should not be mixed with responsibilities of other layers.**
  - For example, objects in the UI layer should focus on UI work, such as creating windows, capturing mouse and keyboard events etc.
  - Objects in the application logic or "domain" layer should focus on application logic, such as calculating a sales total or taxes, or moving a piece on a game board.

- **UI objects should not do application logic!**
  - For example, a Java Swing JFrame (window) object should not contain logic to calculate taxes or move a game piece.
  - Application logic classes should not trap UI mouse or keyboard events.

Deployment Diagram

A deployment diagram represents the **physical architecture**

It shows the assignment of concrete software artifacts (such as files) to computational nodes.

Computational nodes:

1.Device node

- A **physical** computing resource (computer, mobil phone…)

2.Excecution environment node

- A **software** computing resource that runs within an outer node (operating system, database engine, web browser….)

Deployment Diagram