SWEN30006
Software Modelling and Design

Logical Architecture and UML Package Diagrams

Larman Chapter 13

0x2B | ~0x2B

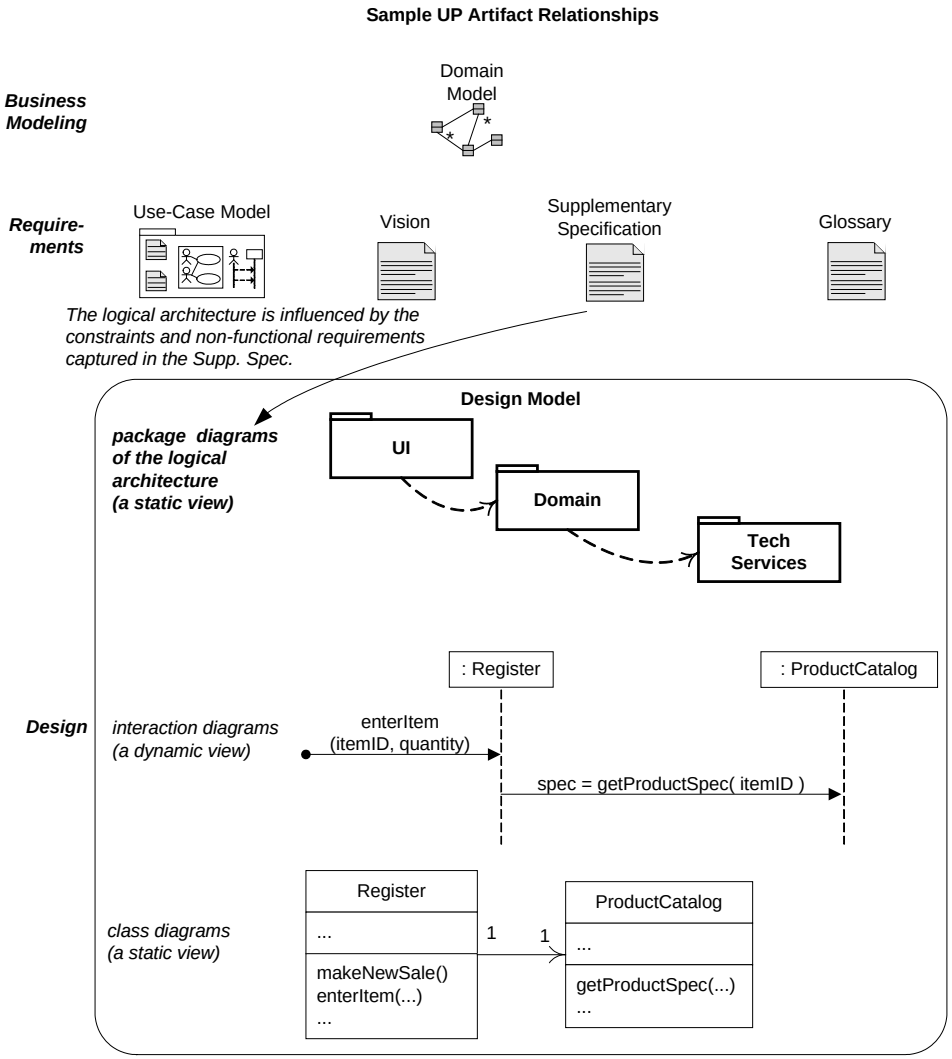
—Hamlet

Objectives

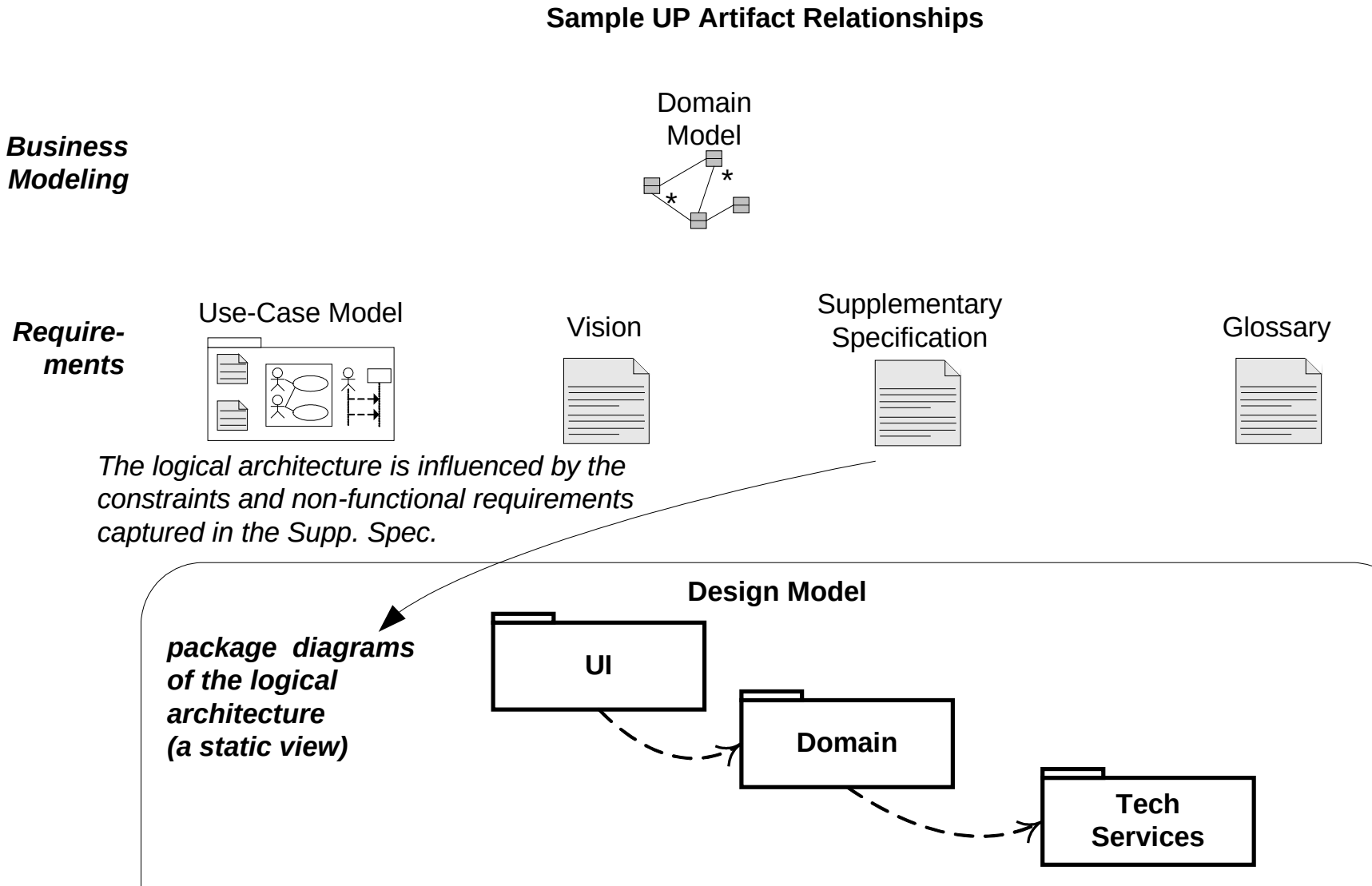
On completion of this topic you should be able to:

- Define a logical architecture using layers.
- Illustrate the logical architecture using UML package diagrams.

Sample UP Artefact Influence



Sample UP Artefact Influence



Definition: Software Architecture

- the set of significant decisions about the organisation of a software system
- the selection of the structural elements and the interfaces by which the system is composed
- their behaviour as specified in the collaborations among those elements
- the composition of these structural and behavioural elements into progressively larger subsystems
- the architectural style that guides this organisation

Logical Architecture and Layers

Logical Architecture:

- The large-scale organisation of the software classes into packages, subsystems and layers.
- *Logical:* not concerned with networking, physical computers, or operating system processes

  (cf. *deployment architecture*)

Layer:

- Coarse-grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system.

Layered Architecture

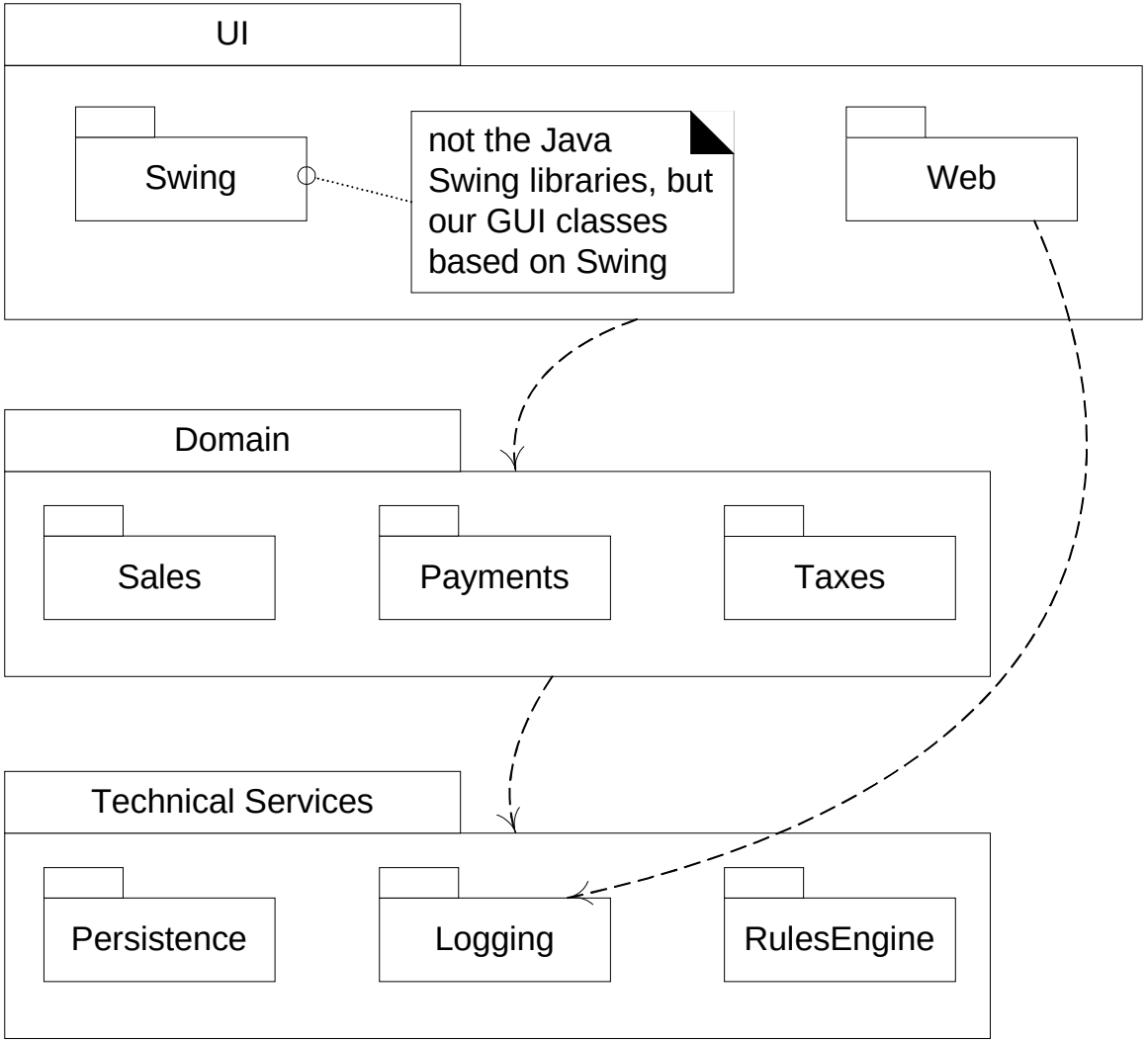
Strict:

- Layer only calls upon the services of the layer directly below it
  - e.g. a network protocol stack

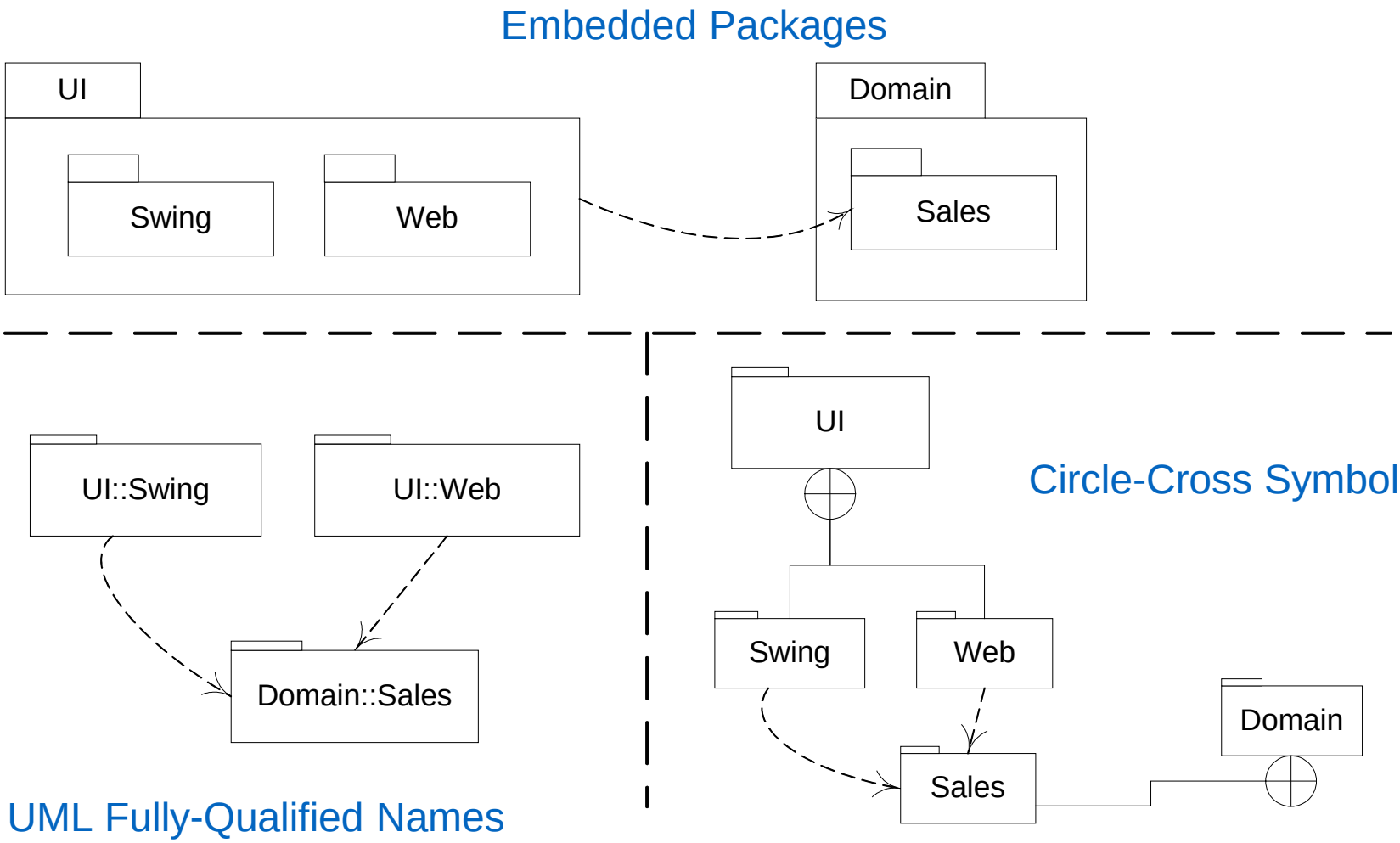
Relaxed:

- A layer calls upon the services in several lower layers
  - e.g. information systems

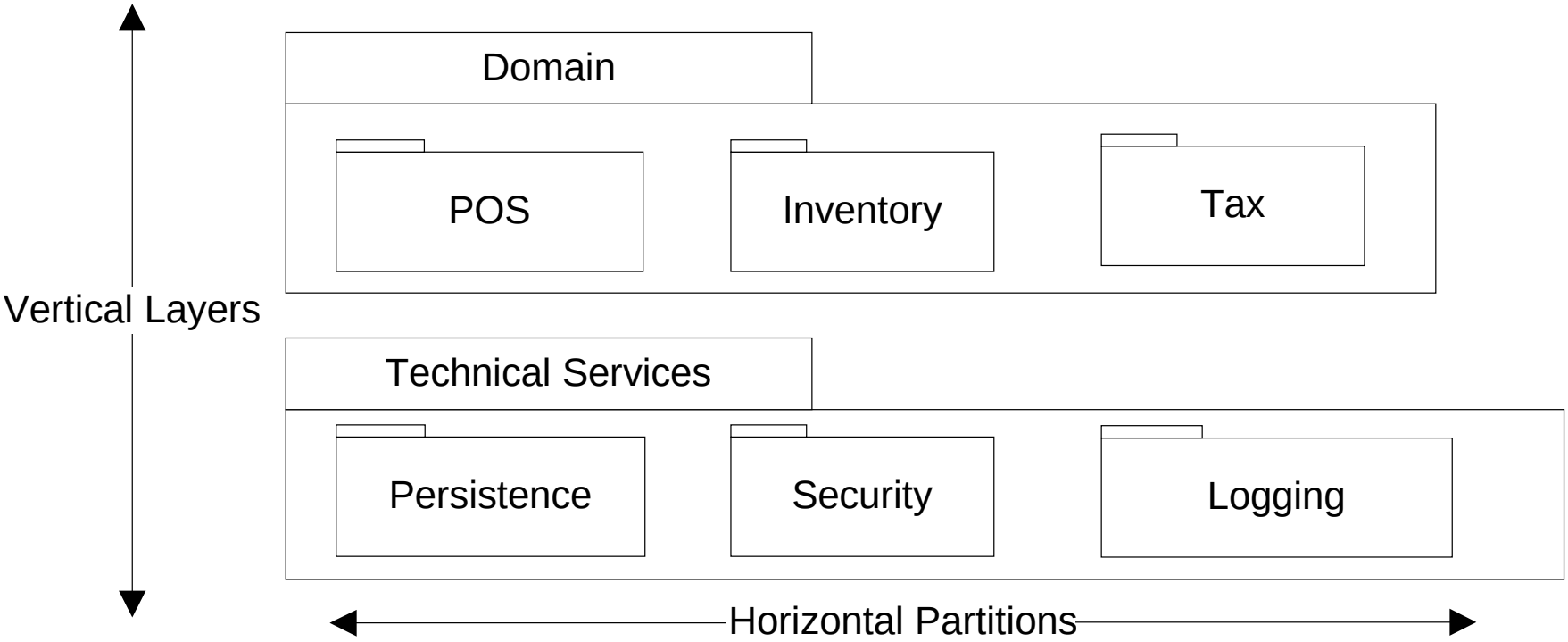
UML Package Diagram Notation: Layers



Package Nesting: Alternatives



Layers and Partitions



Guideline: Design with Layers

- Organise large-scale logical structure of system into distinct cohesive layers from high application specific to low general services
  - Maintain separation of concerns

    e.g. No application logic in UI objects
- Collaboration and coupling from higher layers to lower layers
  - Lower to higher layer coupling is avoided.

Common Layers: IS Logical Architecture

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

UI
(AKA **Presentation**, 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
(AKA Business, Application Logic, Model)

- very general low-level business services used in many business domains
- *CurrencyConverter*

Business Infrastructure
(AKA Low-level Business Services)

- (relatively) high-level technical services and frameworks
- *Persistence*, *Security*

Technical Services
(AKA Technical Infrastructure, High-level Technical Services)

- low-level technical services, utilities, and frameworks
- *data structures, threads, math, file, DB, and network I/O*

Foundation
(AKA Core Services, Base Services, Low-level Technical Services/Infrastructure)

more app specific

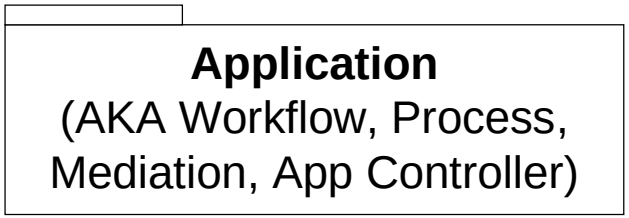
dependency

width implies range of applicability

Common Layers: IS Logical Architecture

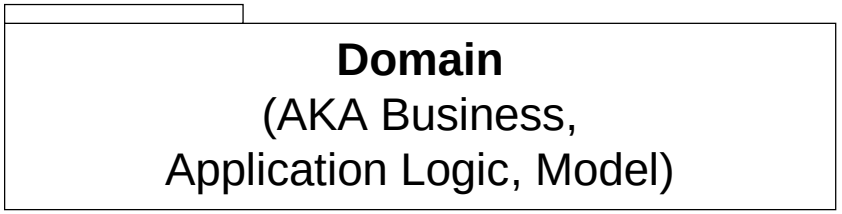
- GUI windows
- reports
- speech interface
- HTML, XML, XSLT, JSP, Javascript, ...

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



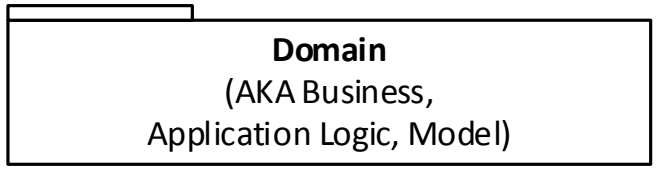
- handles application layer requests
- implementation of domain rules
- domain services (*POS*, *Inventory*)

- services may be used by just one application, but there is also the possibility of multi-application services



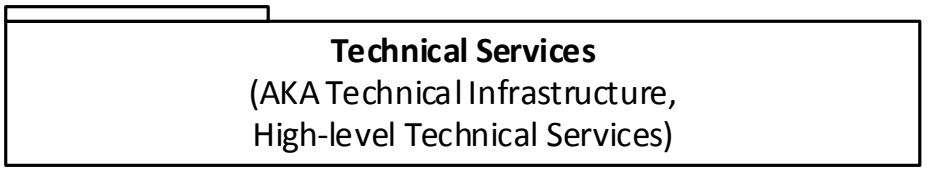
Common Layers: IS Logical Architecture

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



very general low-level business services used in many business domains
CurrencyConverter

(relatively) high-level technical services and frameworks
Persistence, *Security*



low-level technical services, utilities, and frameworks
data structures, threads, math, file, DB, and network I/O

Mapping UML Packages to Code

// --- UI Layer

com.mycompany.nextgen.ui.swing
com.mycompany.nextgen.ui.web

// --- DOMAIN Layer

// packages specific to the NextGen project
com.mycompany.nextgen.domain.sales
com.mycompany.nextgen.domain.payments

// --- TECHNICAL SERVICES Layer

// our home-grown persistence (database) access layer
com.mycompany.service.persistence

// third party
org.apache.log4j
org.apache.soap.rpc

// --- FOUNDATION Layer

// foundation packages that our team creates
com.mycompany.util

Using Layers Helps Address Problems

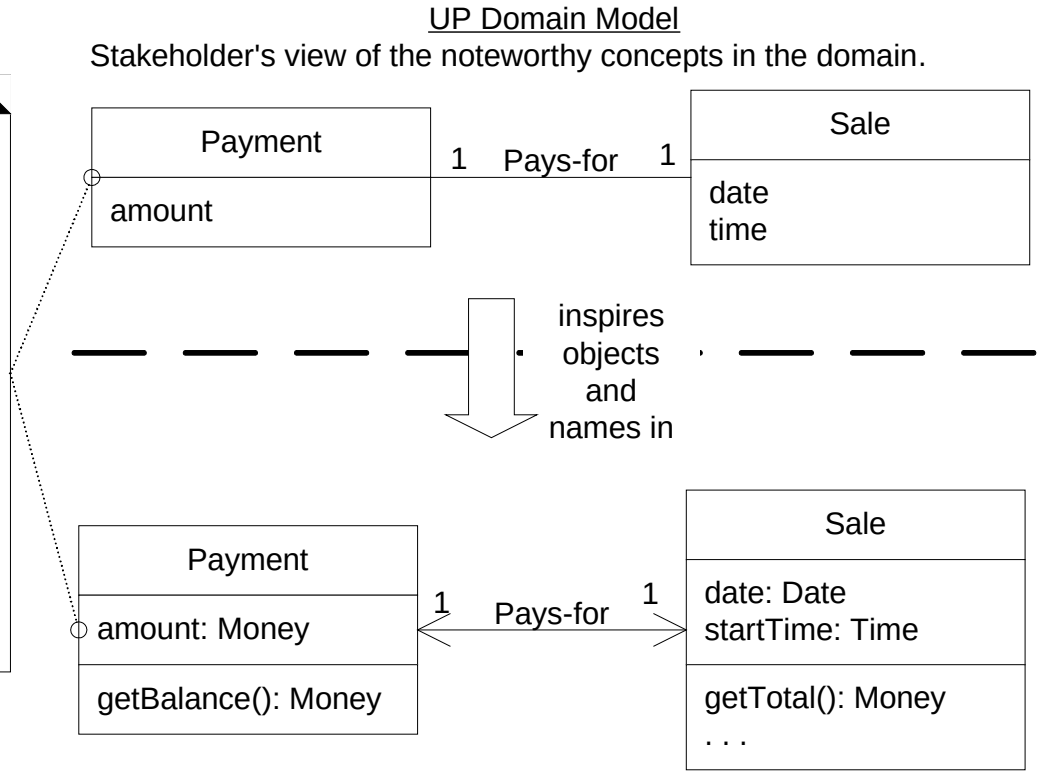
- Changes rippling through system due to coupling
- Intertwining of application logic and UI, reducing reuse and restricting distribution options
- Intertwining of general tech. services or business logic with application specific logic, reducing reuse, restricting distribution, and complicating replacement
- High coupling across areas of concern, impacting division of development work

Domain Layer/Model Relationship

A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

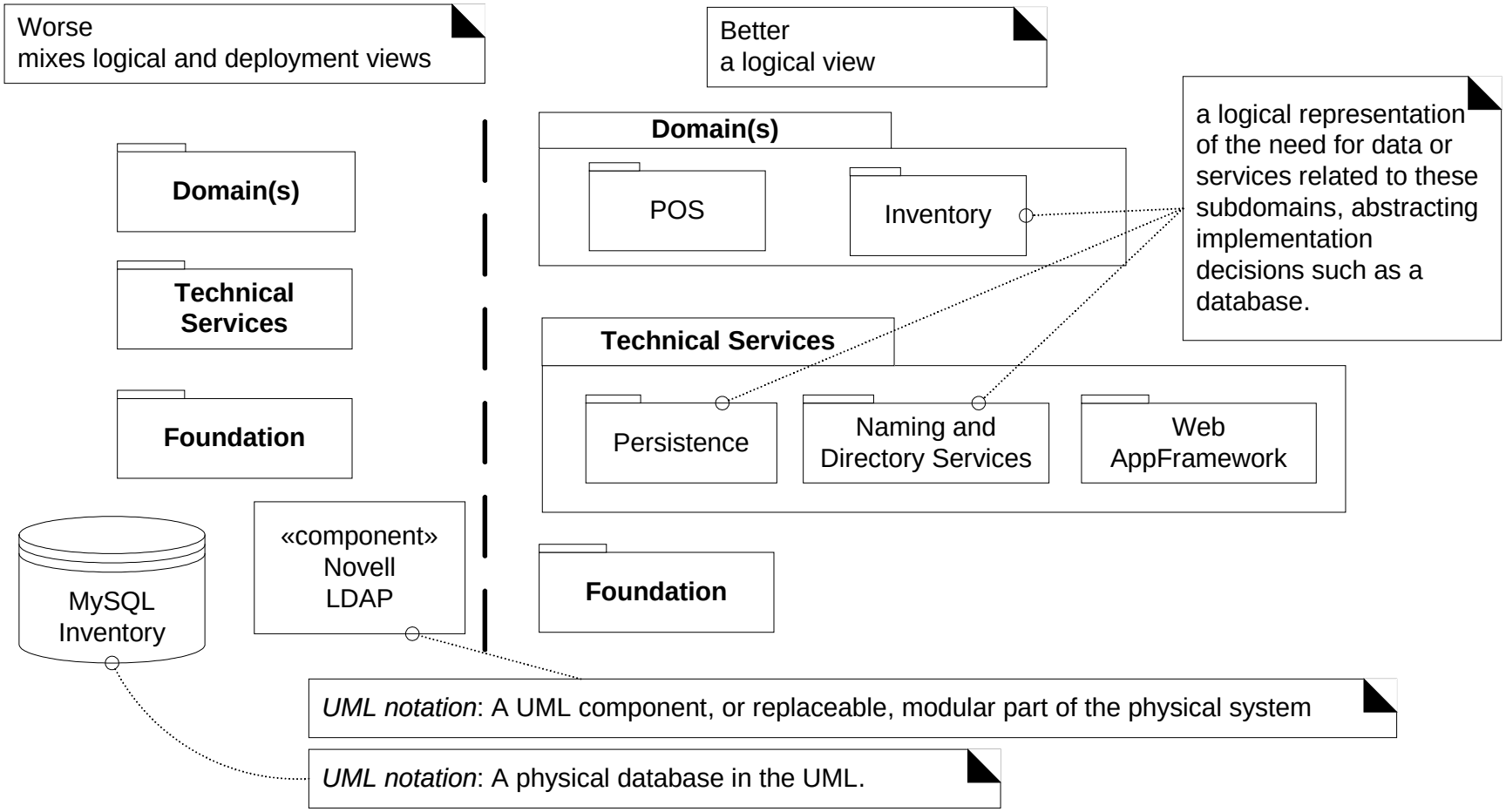


Domain layer of the architecture in the UP Design Model

The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

Mixing Views of the Architecture

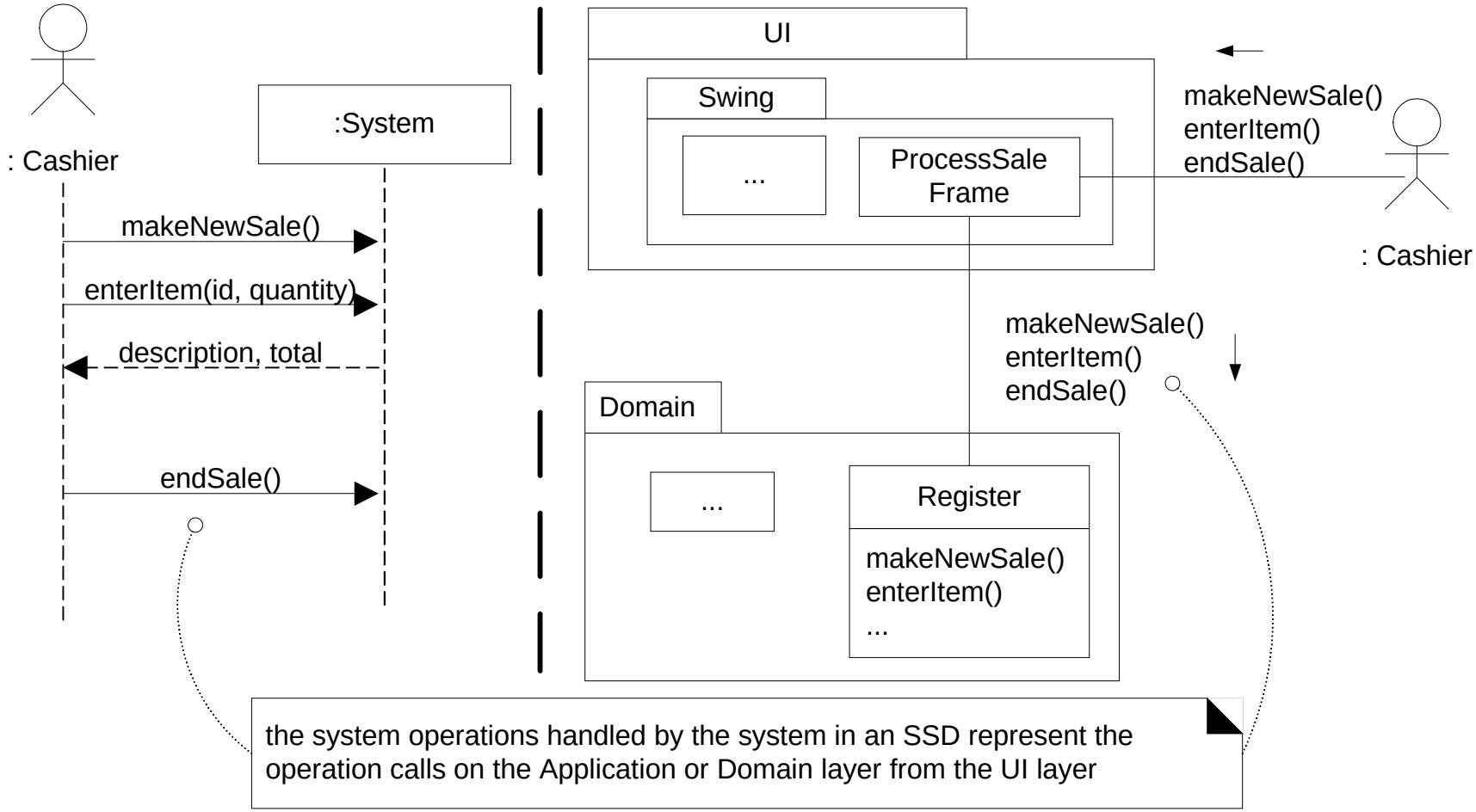


Model-View Separation Principle

What kind of visibility should other packages have to the UI layer? How should non-window classes communicate with windows?

1. Don’t couple non-UI objects directly to UI objects.
   - Non-UI objects should be reusable, and attachable to a new UI.
2. Don’t put application logic in UI object methods.
   - UI objects should only initialise UI elements, receive UI events, and delegate application logic requests to non-UI objects.

System Operations: SSDs and Layers



UML Component Diagrams

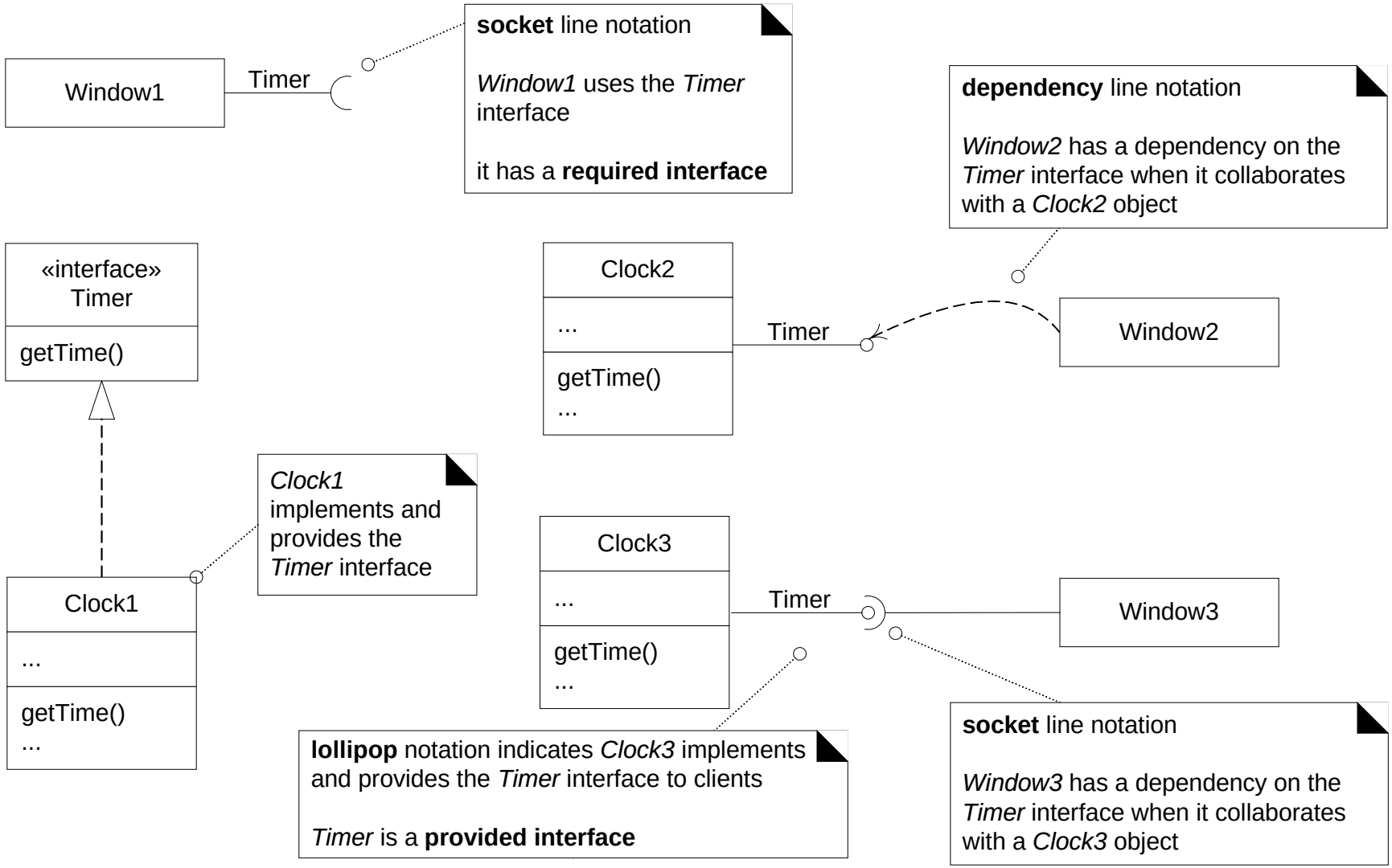
Component:

- a modular part of a system
- encapsulates its contents
- replaceable within its environment
- defines its behaviour in terms of provided and required interfaces.

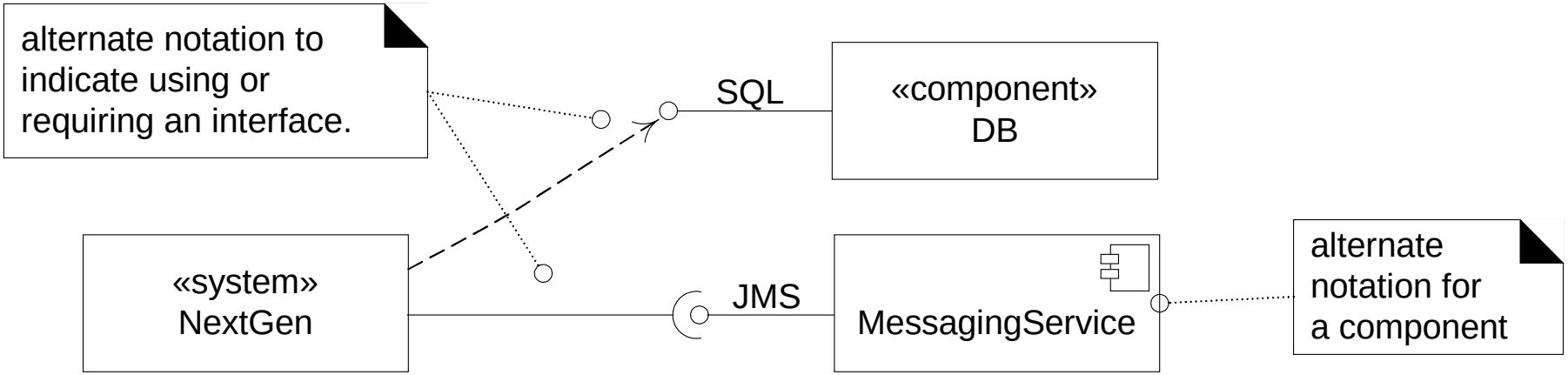
How is this different from a class?

- It is a class!
- One that provides a design-level perspective

Interfaces: Required and Realized



UML Components



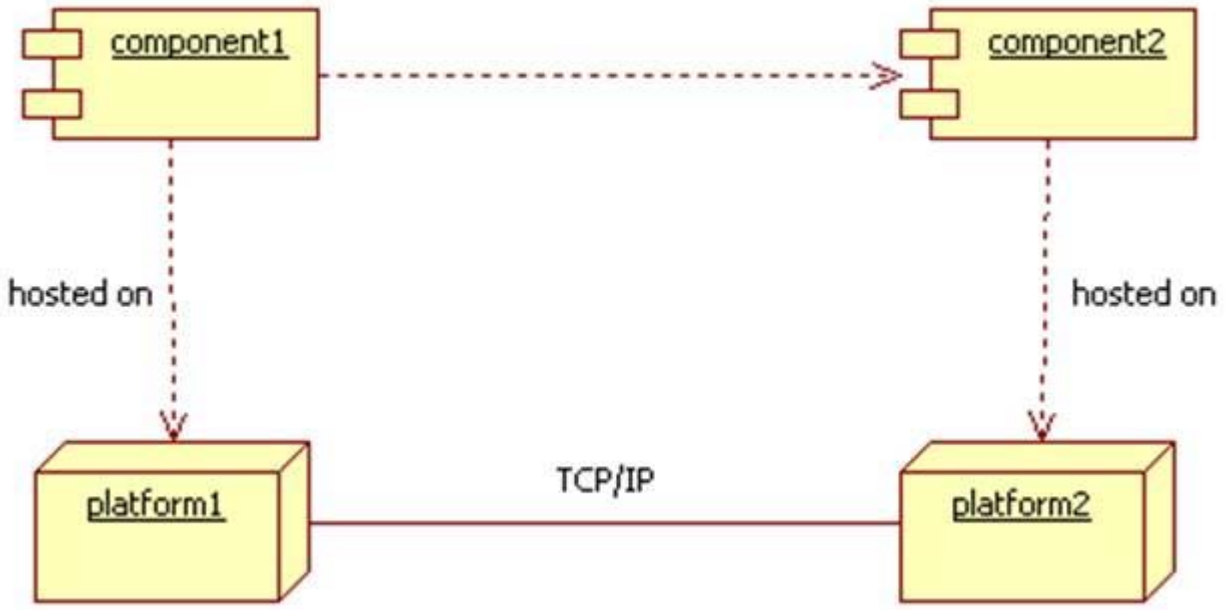
Interface Standards:

- SQL (Structured Query Language)
- JMS (Java Message Service)

Distributed Architectures

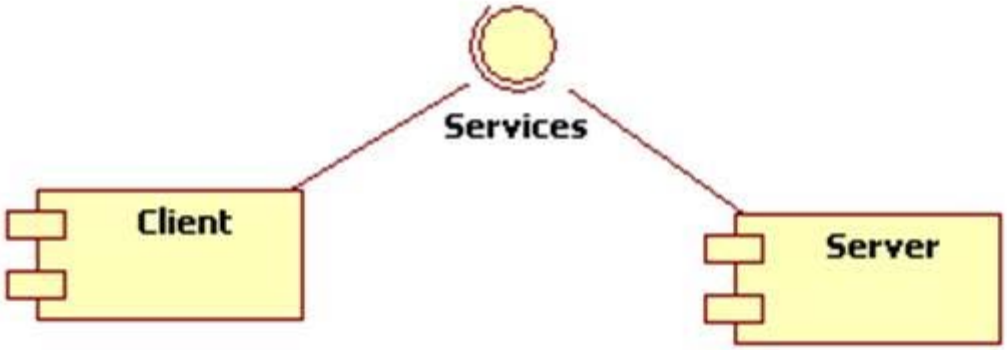
Components are typically:

- Hosted on different platforms
- Communicate through a network



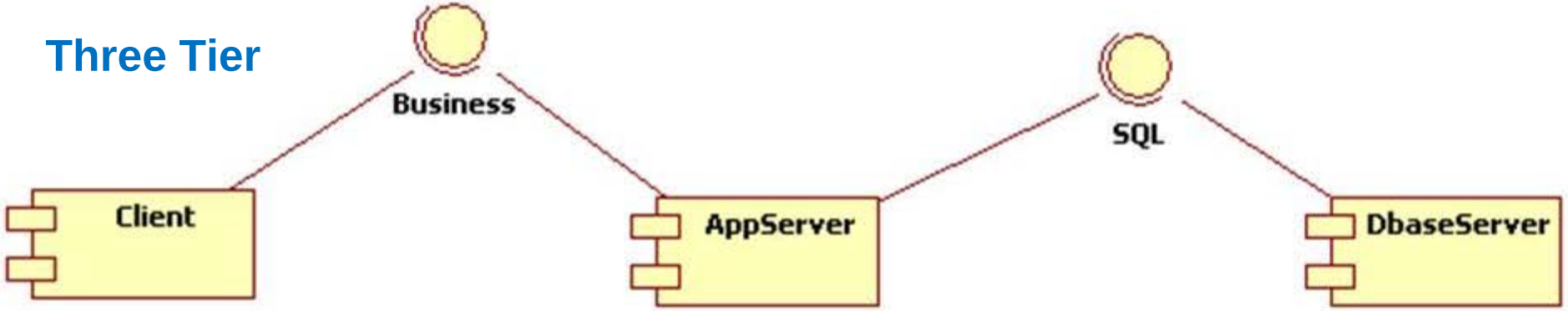
Client-Server Architecture

- Two component types: *Clients* and *Servers*
- Server: perpetually listens for Client requests
  - When request is received, Server processes request, then sends response back to Client
  - Servers may be *Stateless*, or *Stateful* which allows for transactional interaction (*Session*)



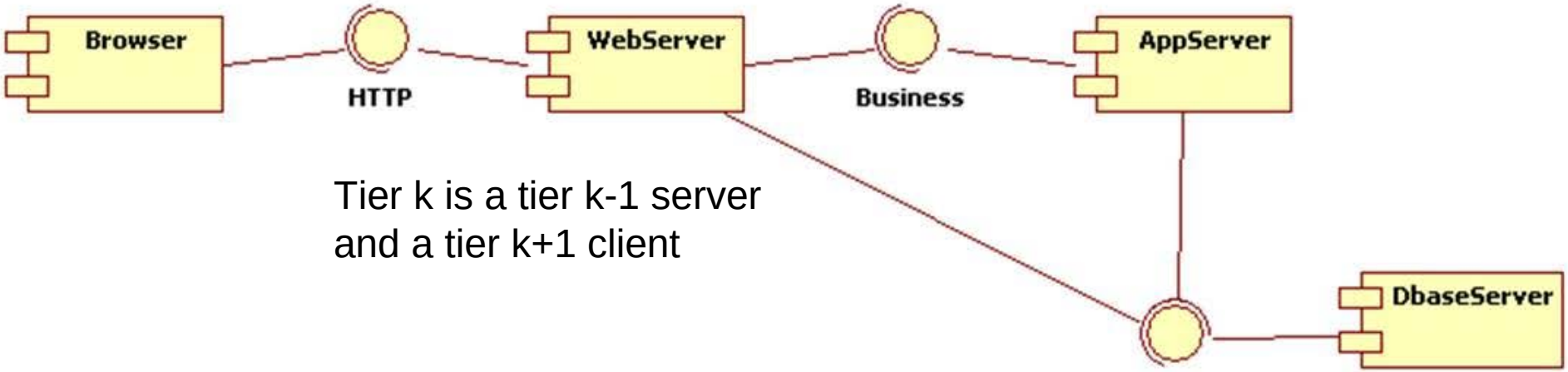
More Client-Server Architectures

Three Tier



Tier 2 is a tier 1 server
and a tier 3 client

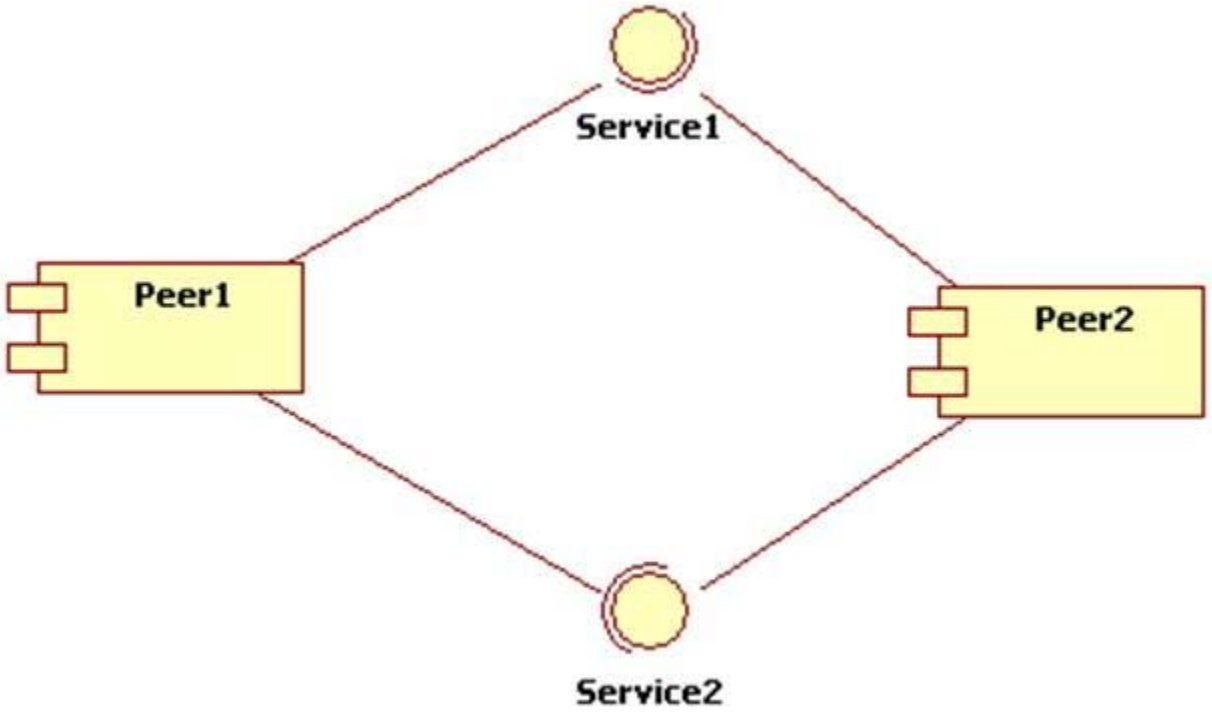
Four Tier



Tier k is a tier k-1 server
and a tier k+1 client

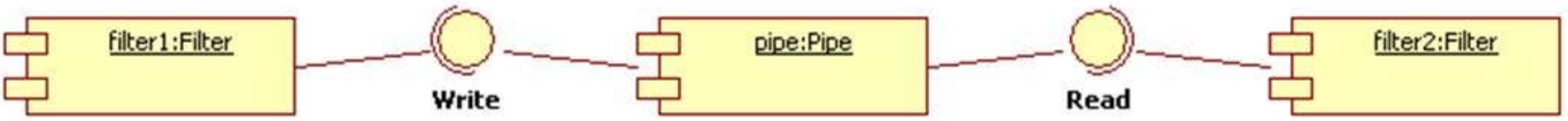
Peer-to-peer (P2P) Architecture

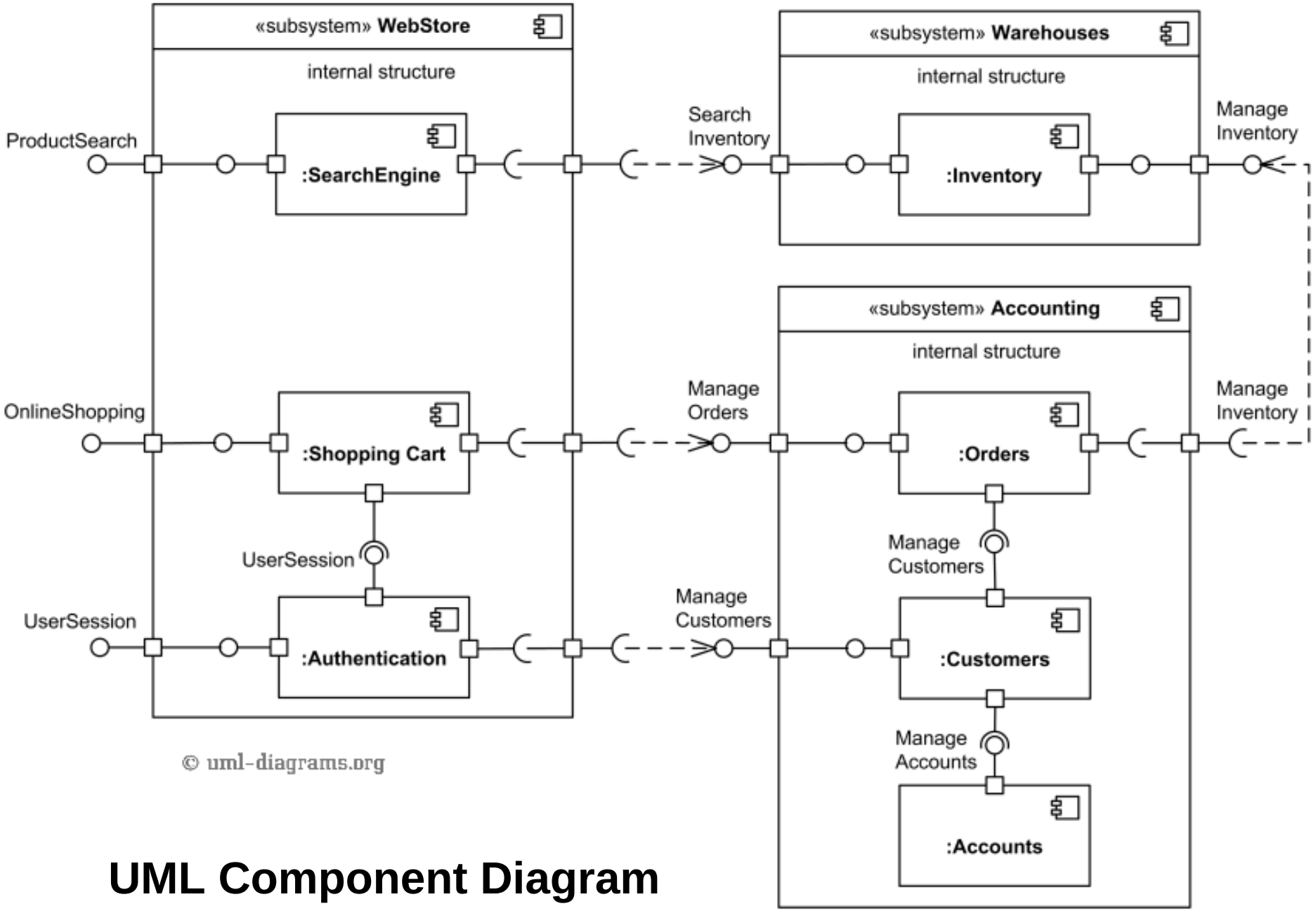
- Roles of client and server switch back and forth between components



Pipeline Architecture

- One of the oldest distributed architectures
- *Filter* perpetually reads data from an input *Pipe*, processes it, then writes the result to an output Pipe
- Can be static and linear, or can be dynamic and complex





UML Component Diagram