

Architectural Models

Robert B. France

Colorado State University

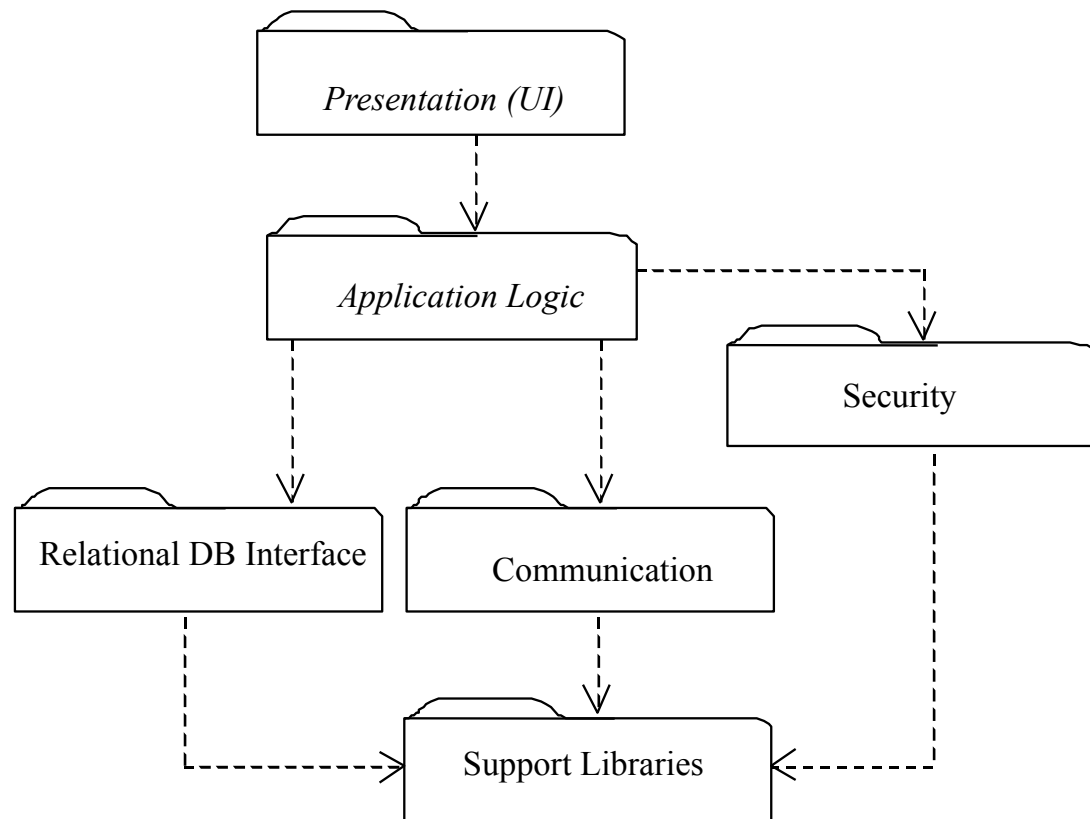
What is an architectural design?

- Concerned with identifying subsystems and their relationships.

Model shown is for a single system

Italicized subsystems represent new systems; normal style represent existing subsystems.

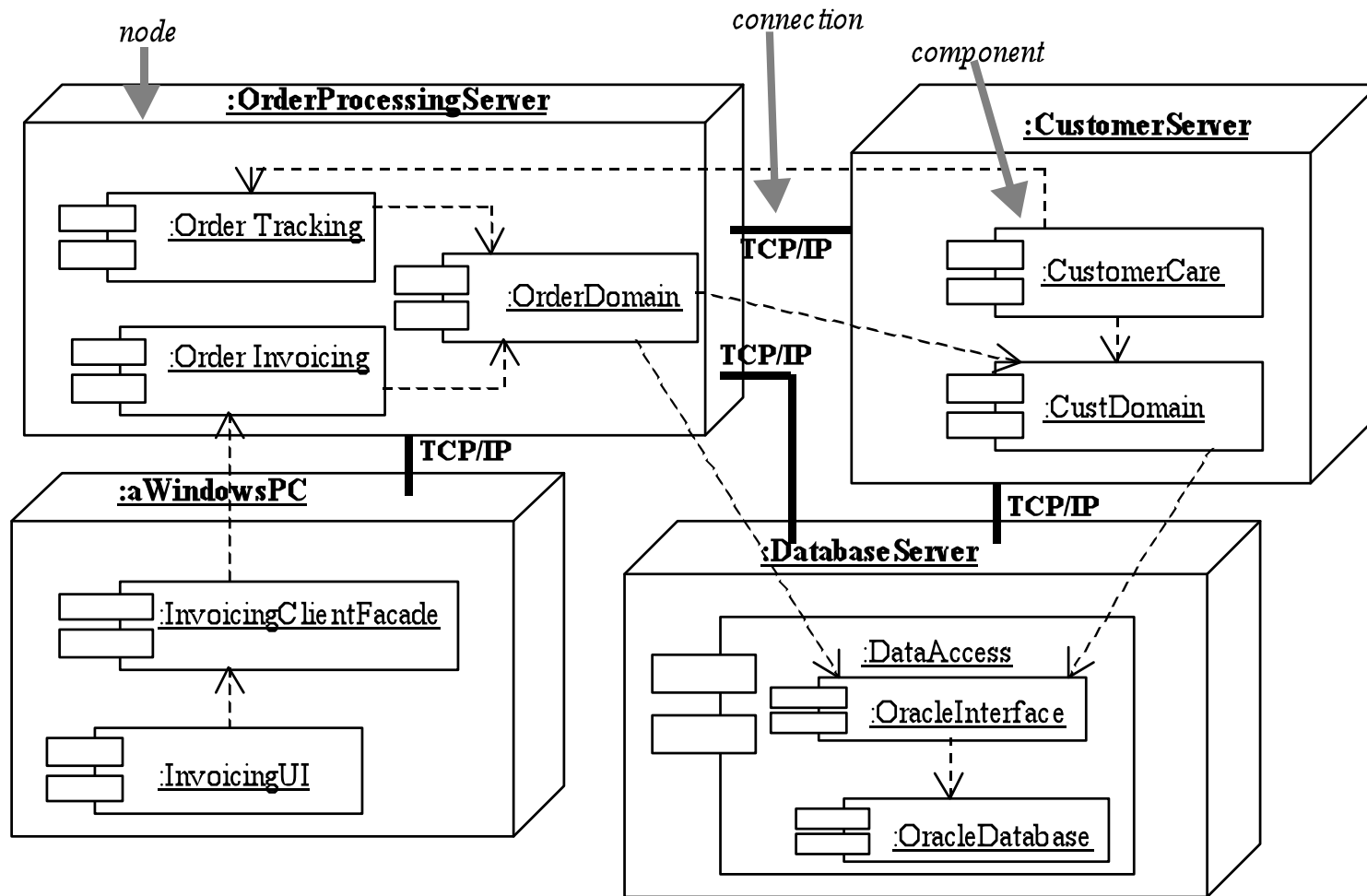
Dashed arrows represent dependencies; the precise nature of the dependency, including protocols, can be stated as annotations.



Architecture Types

- Logical Architecture
 - Organization of a system into logical units (e.g., layers, subsystems)
 - Logical units do not necessarily result in units of implementation (e.g., Java packages, components)
- Implementation (Deployment) Architecture
 - Organization of a system into physical units (e.g., source files, Java packages, components)

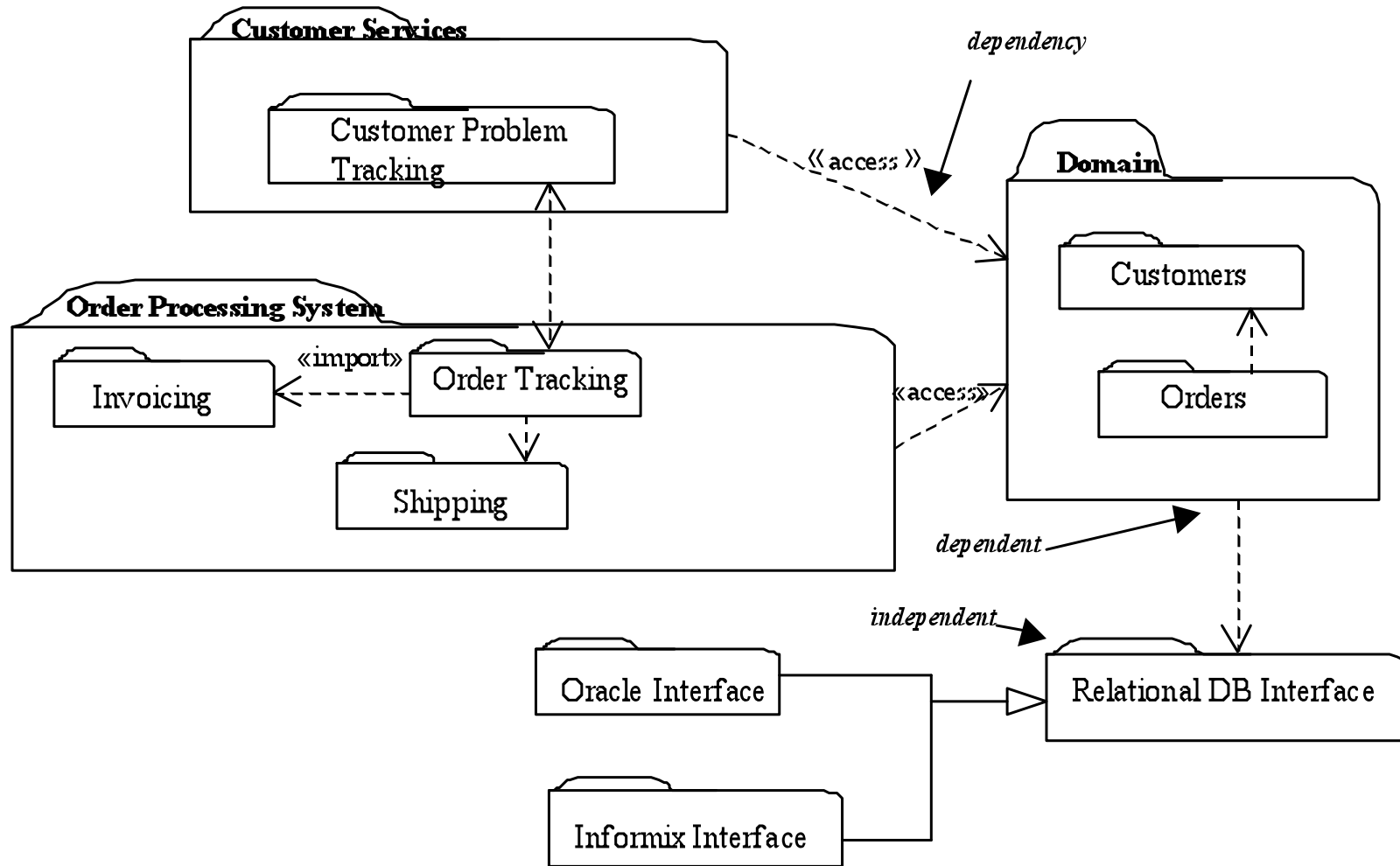
Example Deployment Diagram



Architectural Modeling

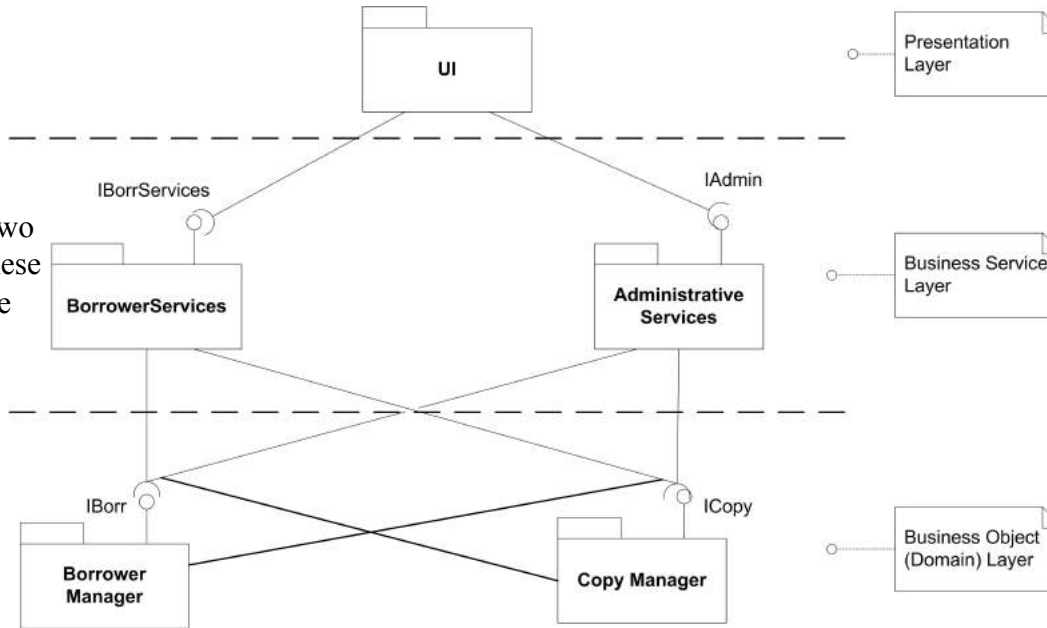
- Two forms covered in CS414
 - Basic
 - Subsystems are connected by access, import relationships
 - Component
 - Subsystems connected via provided and required interfaces

Basic Architecture

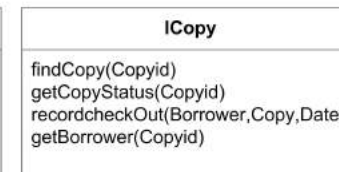
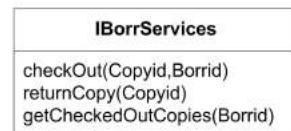


Component-based Architecture

Use cases grouped into borrower services and administrative services; two subsystems for each of these groupings is created – one responsible for borrower services, the other for administrative services



Both administrative and borrower services need copy and borrower information. This observation results in a decision to include two subsystems for managing copies and borrowers.

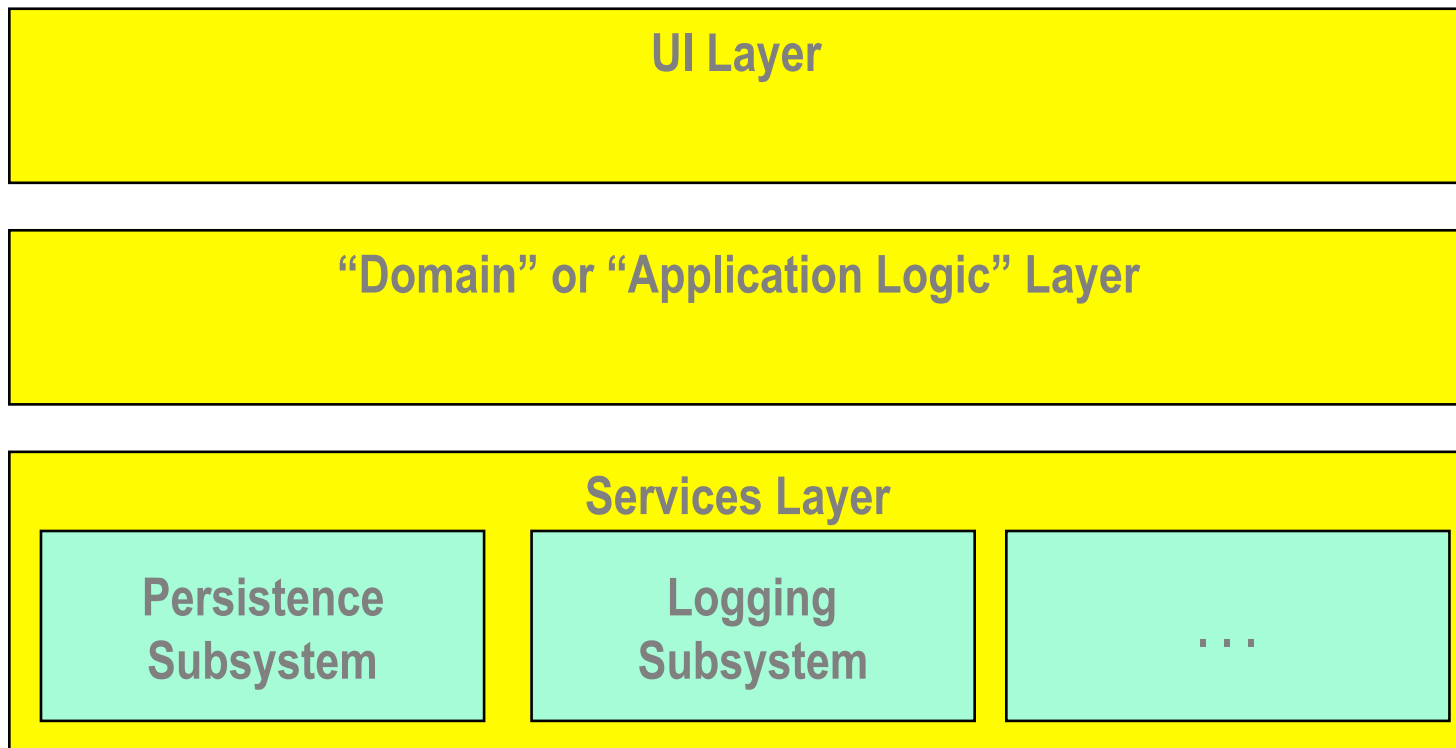


Layered Logical Architectures

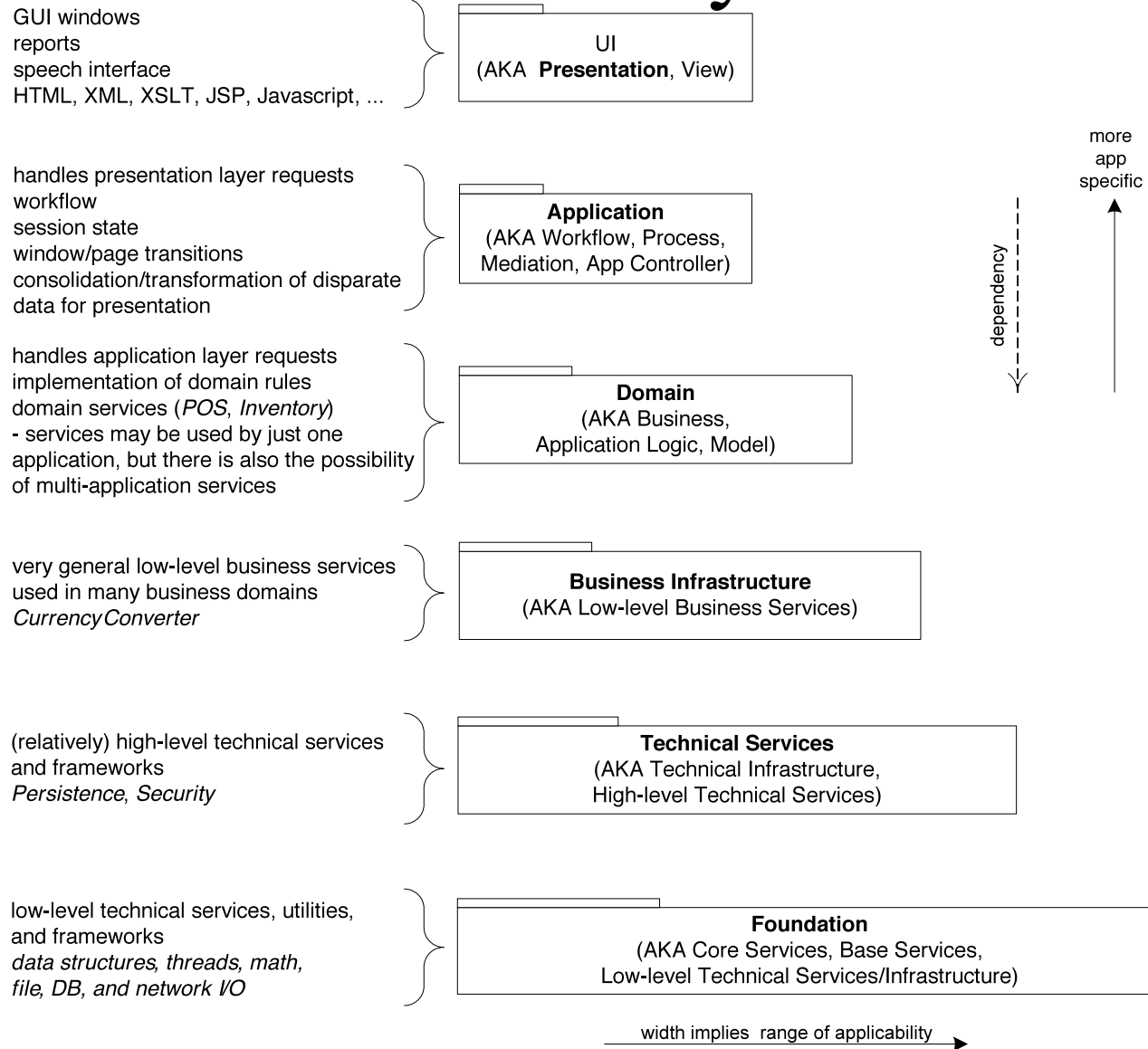
- Layered
 - System organized into layers of functionality
 - Presentation/Application/Persistence (Data)
 - Each layer uses the services of a lower layer
 - Supports separation of concerns

3-Tiered Layered Architectures

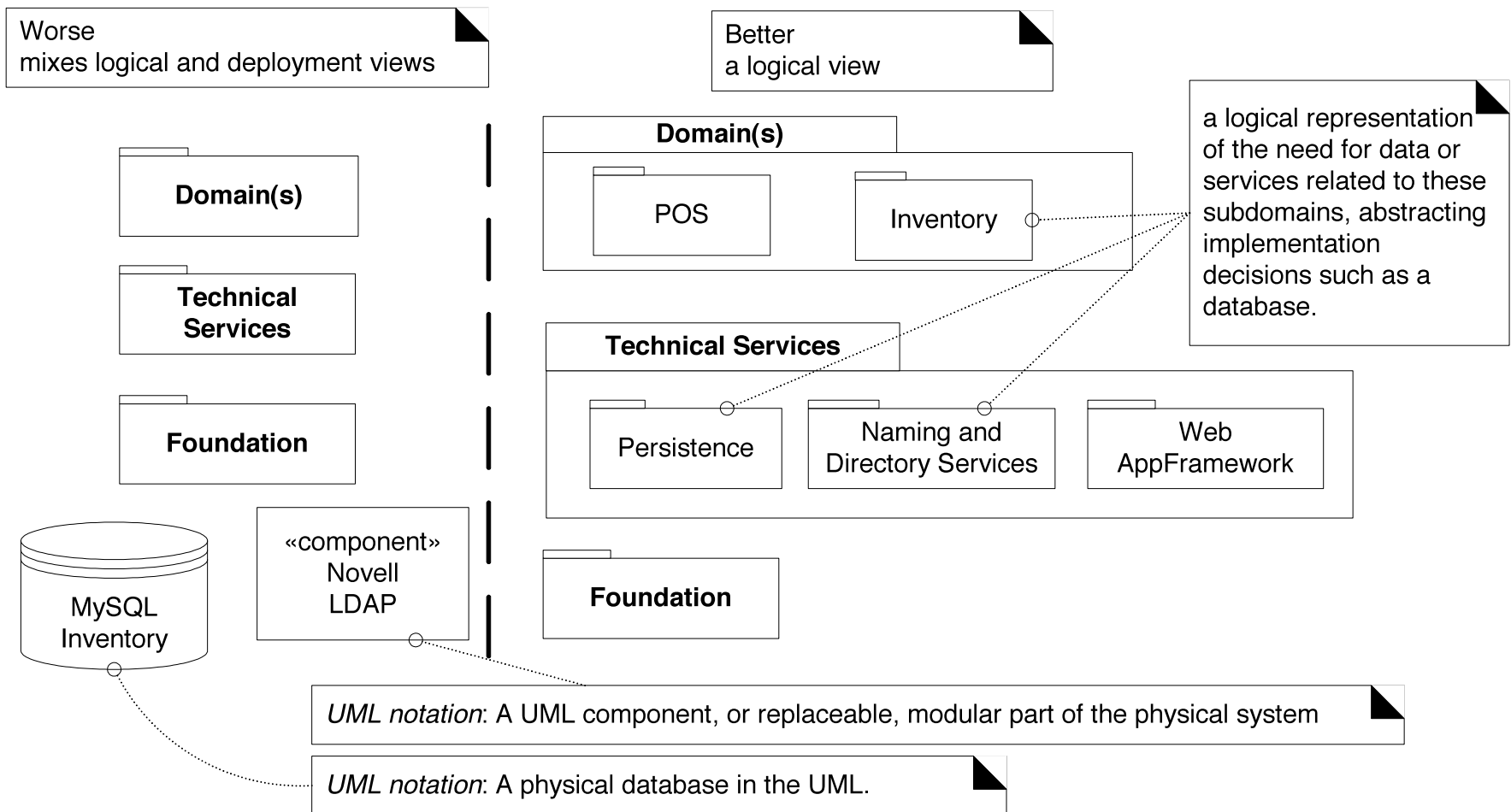
- Separate presentation and application logic, and other areas of concern.



N-Tiered Layers



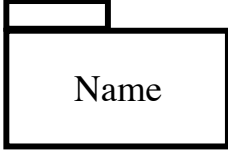
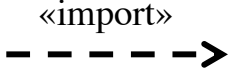
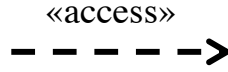
Architecture Guidelines



Packages

- A *package* is a collection of model elements
- In logical architectures a package is a subsystem
- A package is rendered as a tabbed folder

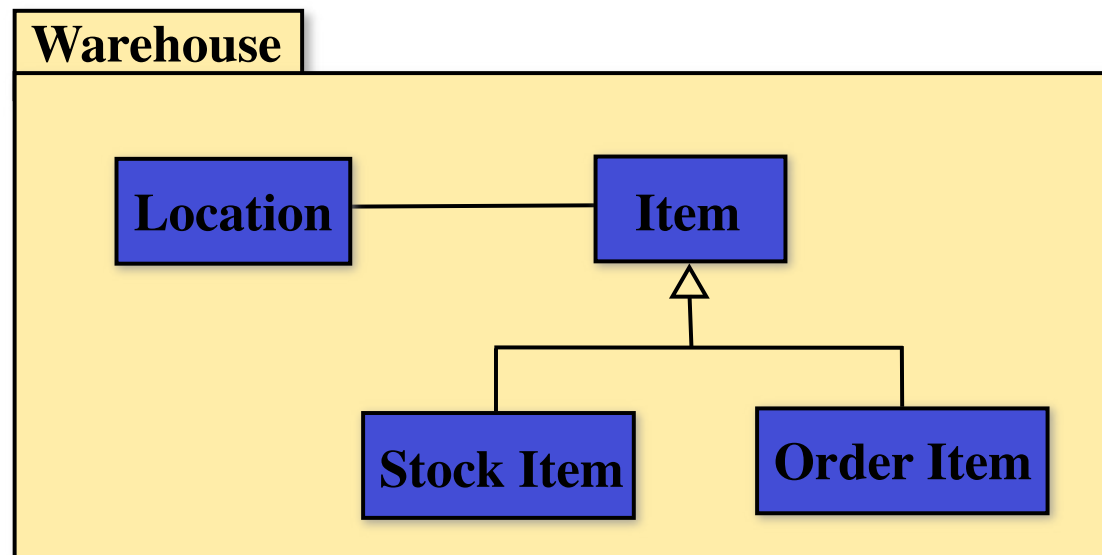
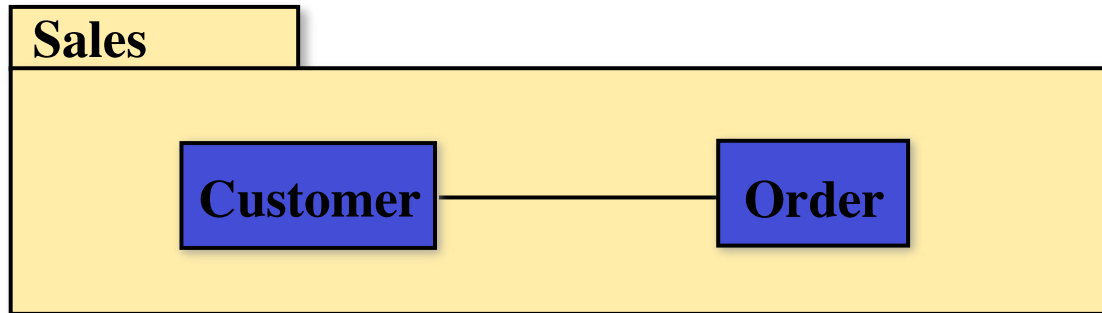
Core Concepts

Construct	Description	Syntax
Package	A grouping of model elements.	
Import	A dependency indicating that the public contents of the target package are added to the namespace of the source package.	
Access	A dependency indicating that the public contents of the target package are available in the namespace of the source package.	

Packages as a Namespace

- A package defines a namespace
 - A model element belongs to at most one package
 - Model elements can be referenced outside of the packages they are defined in
 - package1::class1
 - package3::class1
 - A package can contain other packages
 - package1::package2::class2

Package Examples

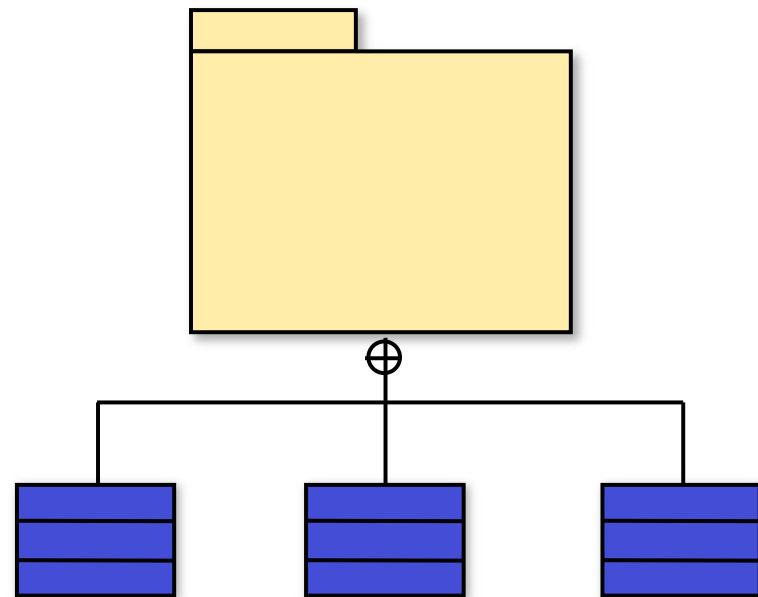
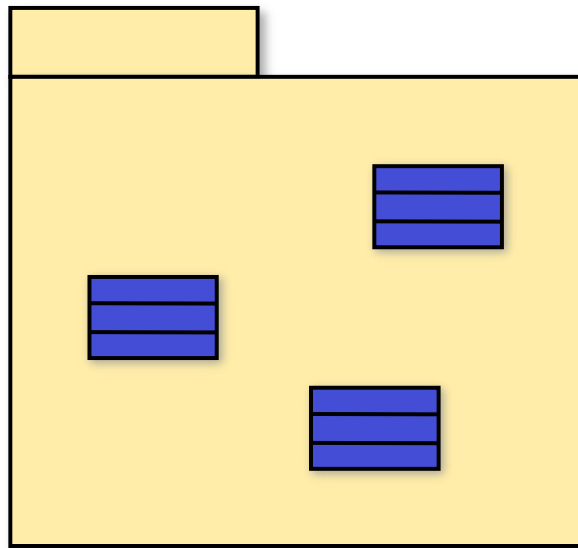


Packages

- A package owns its model elements
 - destruction of the package results in destruction of the model elements
 - relationship between a model element and its package is a composition

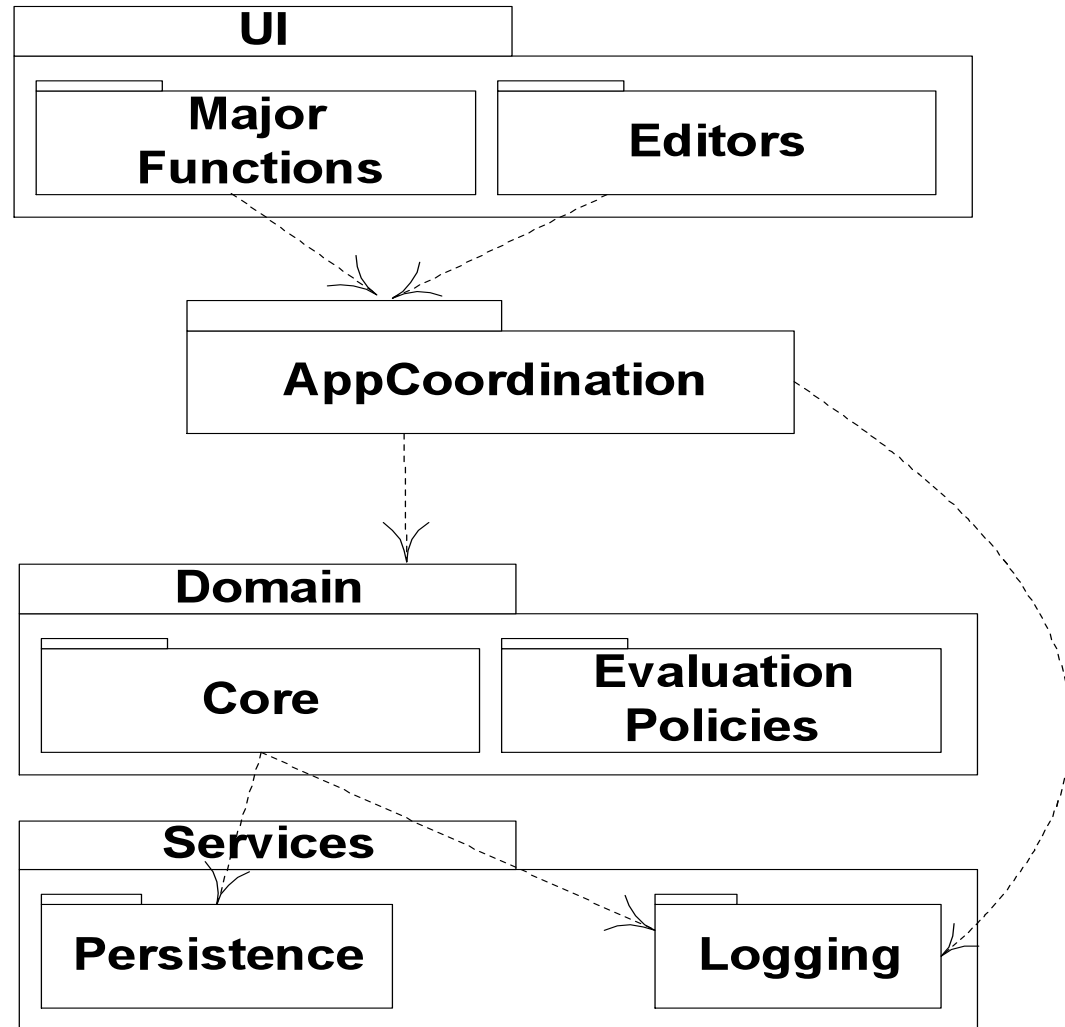
Package Containment

- Packages are shown in static diagrams
- Two equivalent ways to show containment:



Package Dependencies

- It is useful to show the coupling with UML dependency lines.
- A CASE tool can reverse engineer these diagrams.



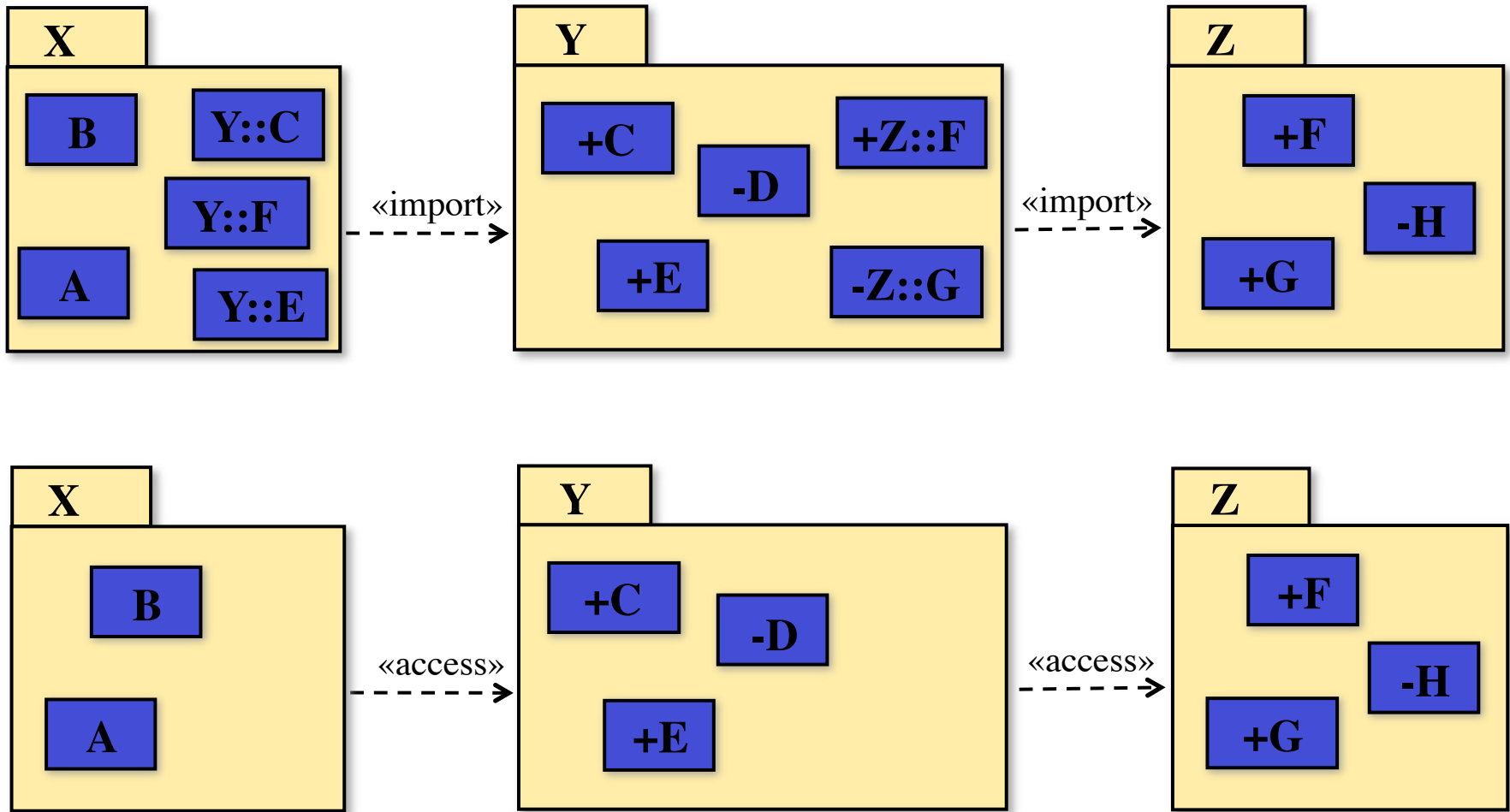
Accessing Package Elements

- To control access to elements in packages they are accessed or imported.
- *Import/Access*: A package that imports/ accesses another has access to the public elements of the imported/accessed package
 - public elements are said to be *exported*

Import versus Access

- Package1 contains public Class1; Package2 contains public Class2.
- Package1 access Package2
 - elements in Package1 must reference Class2 as follows: Package2::Class2
- Package1 import Package2
 - elements in Package1 can reference Class2 as just Class2

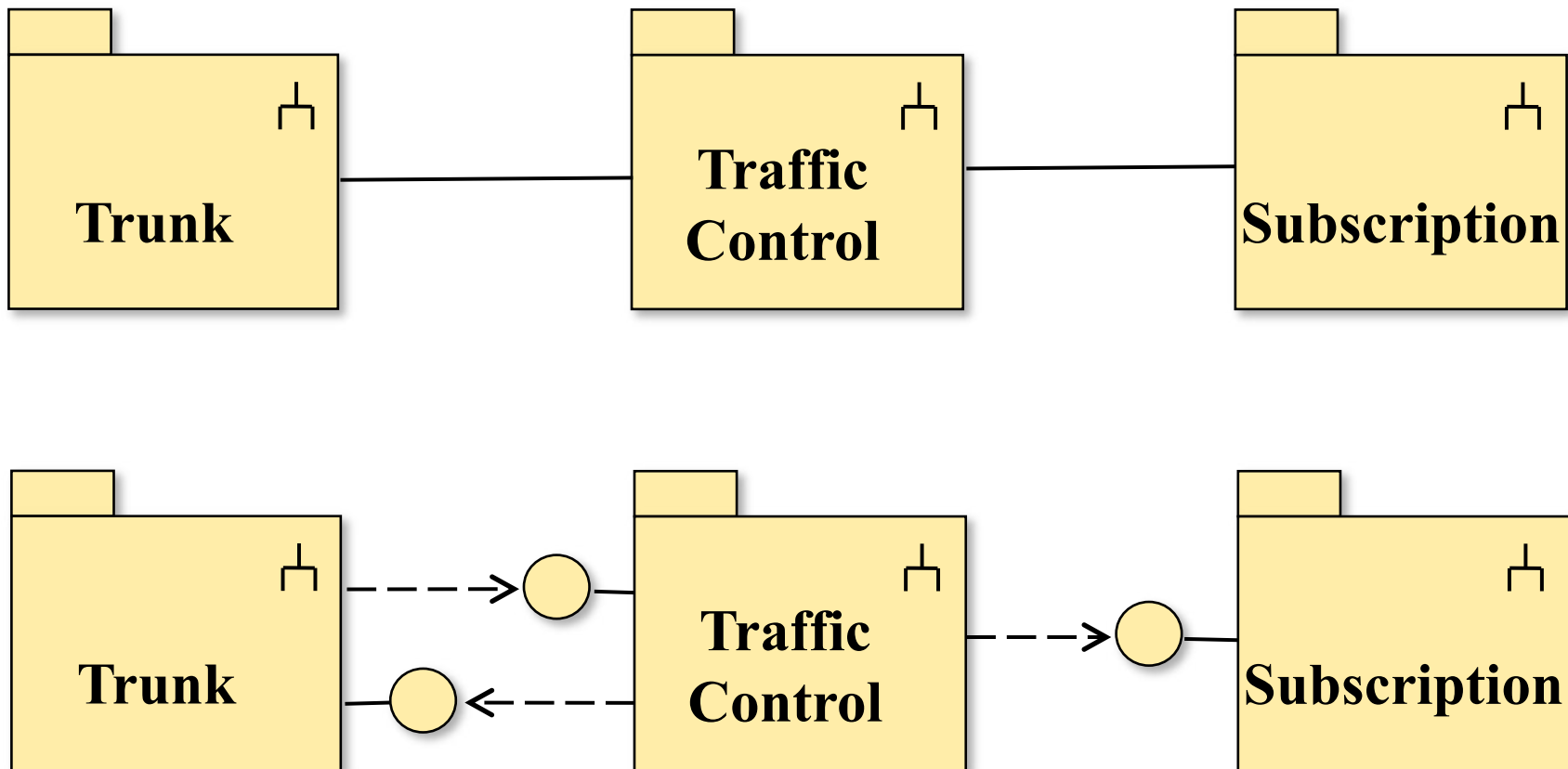
Import vs. Access



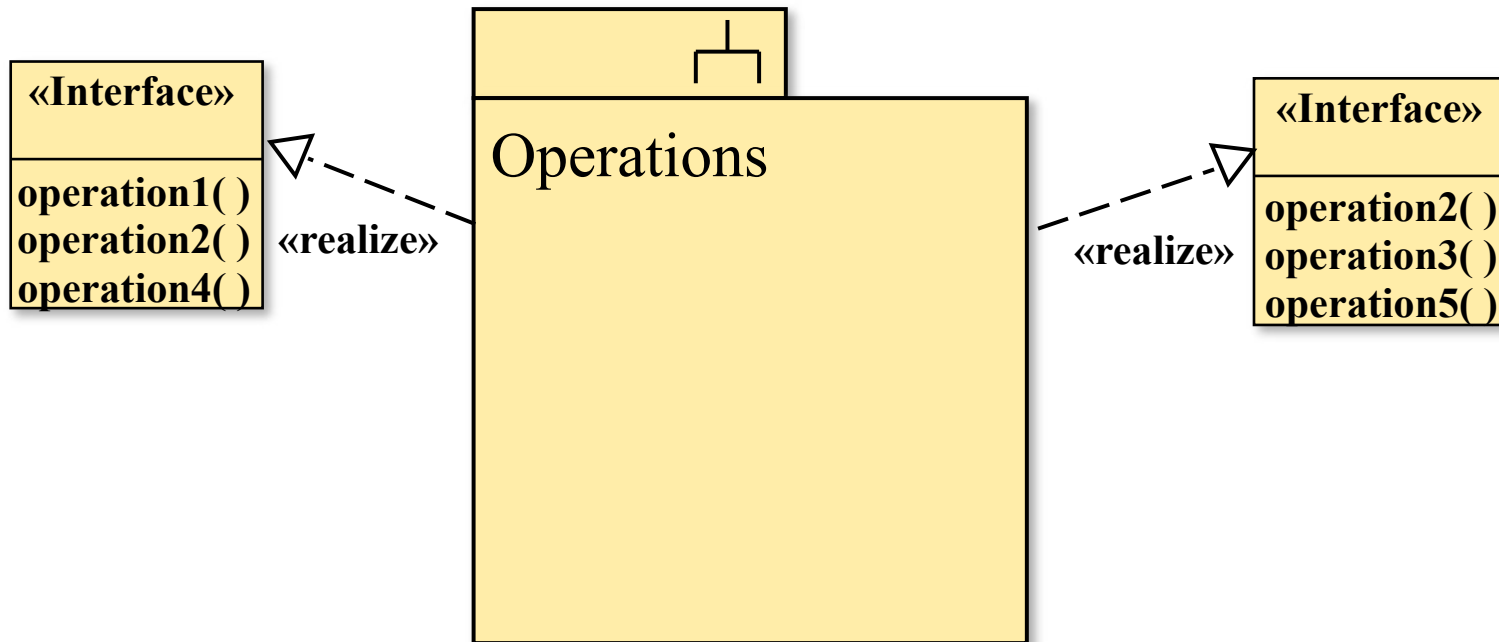
Robert B. France

Arch-21

Subsystem Interfaces – Component-based architecture



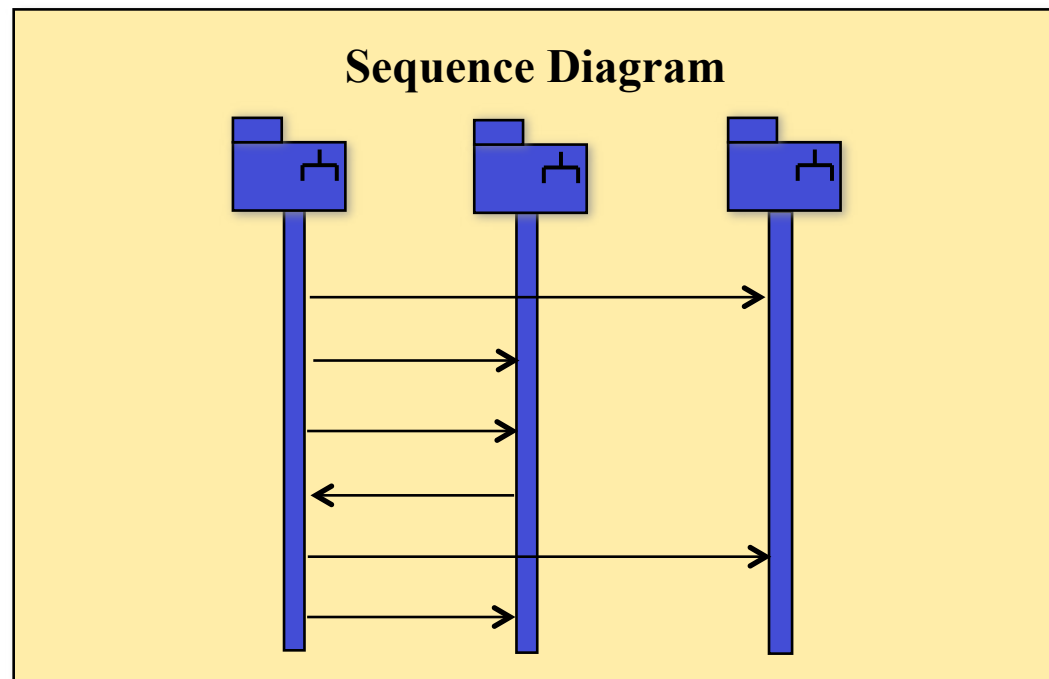
Operations and Interfaces



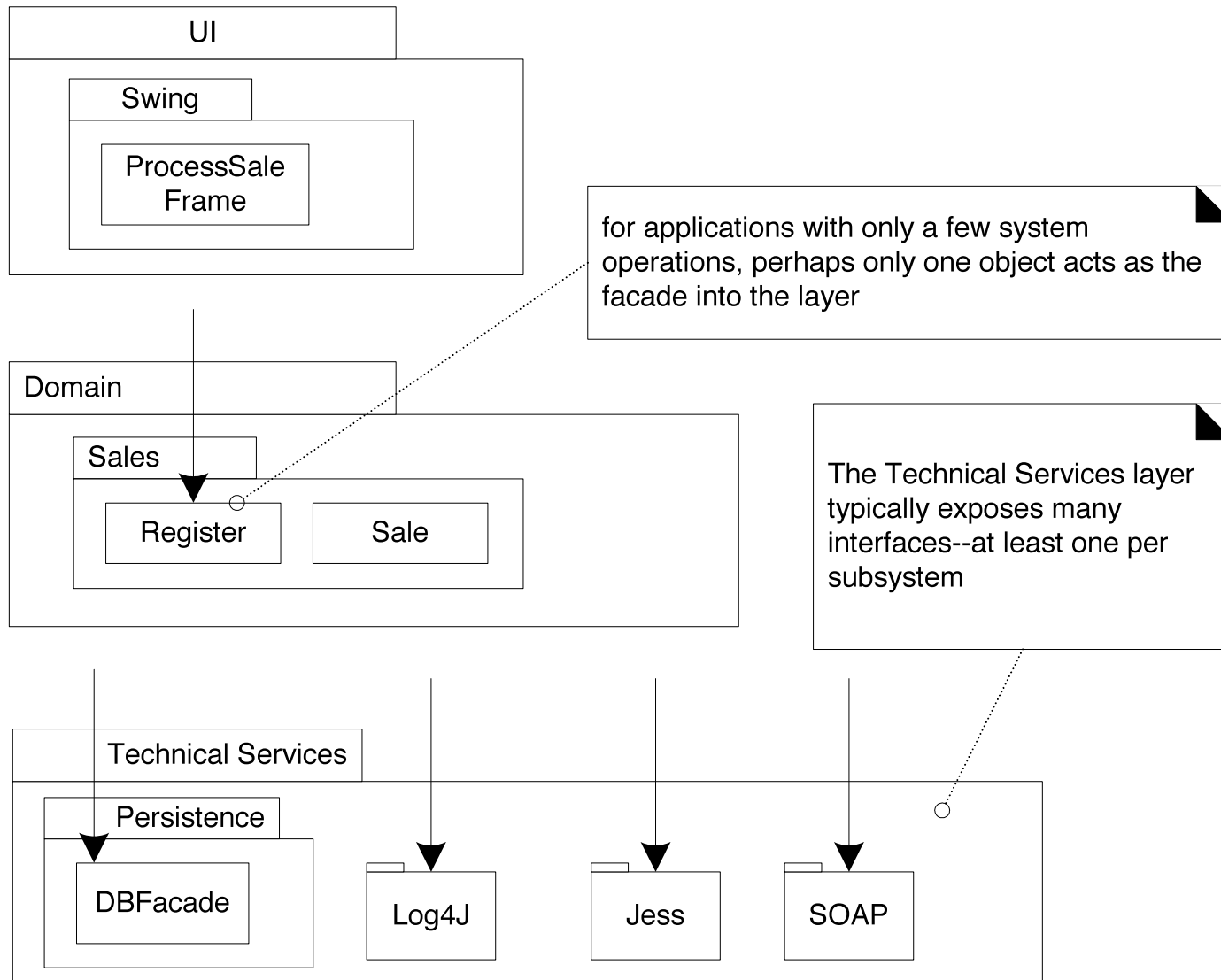
The subsystem must support all operations in the offered interfaces

Subsystem Interactions

- Subsystems can be shown in interaction diagrams
 - sequence diagrams

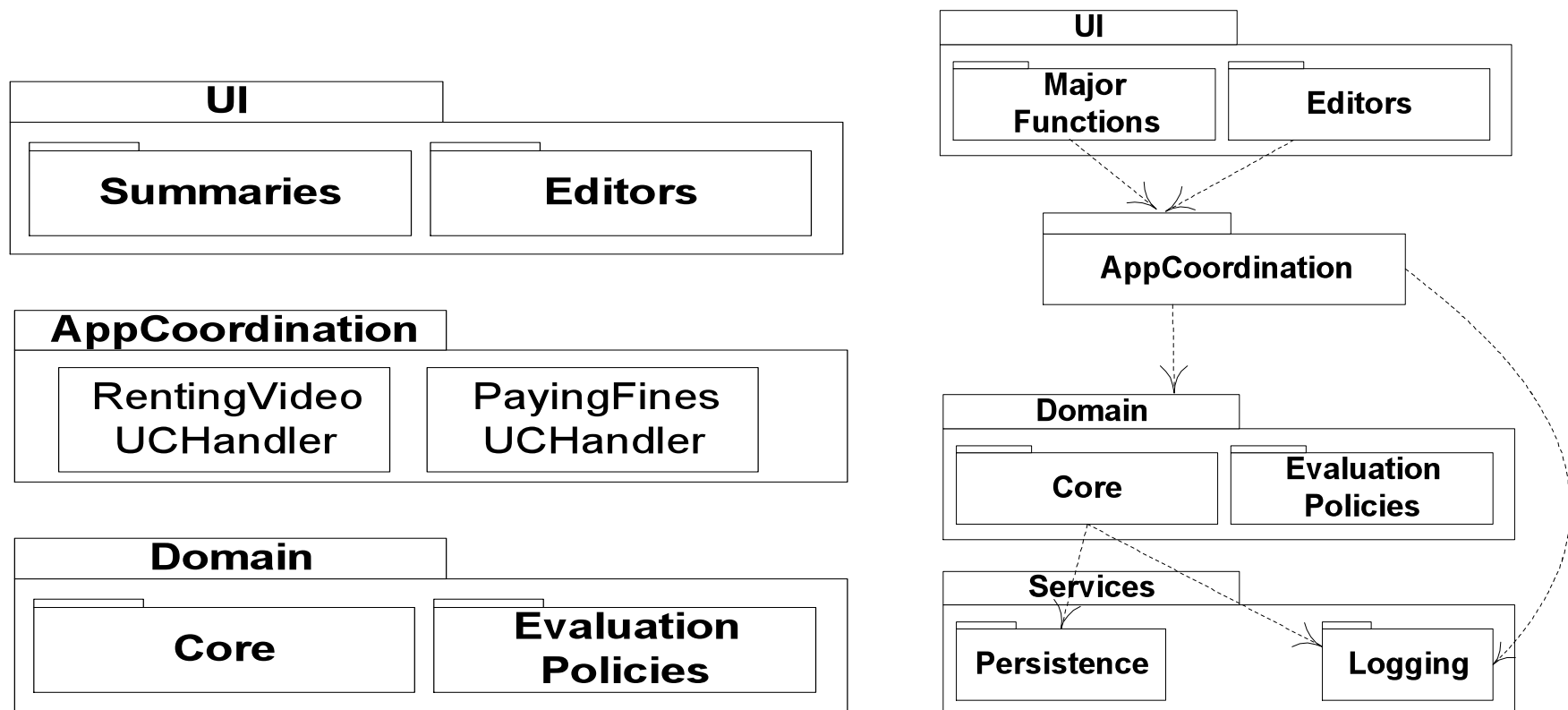


Facades

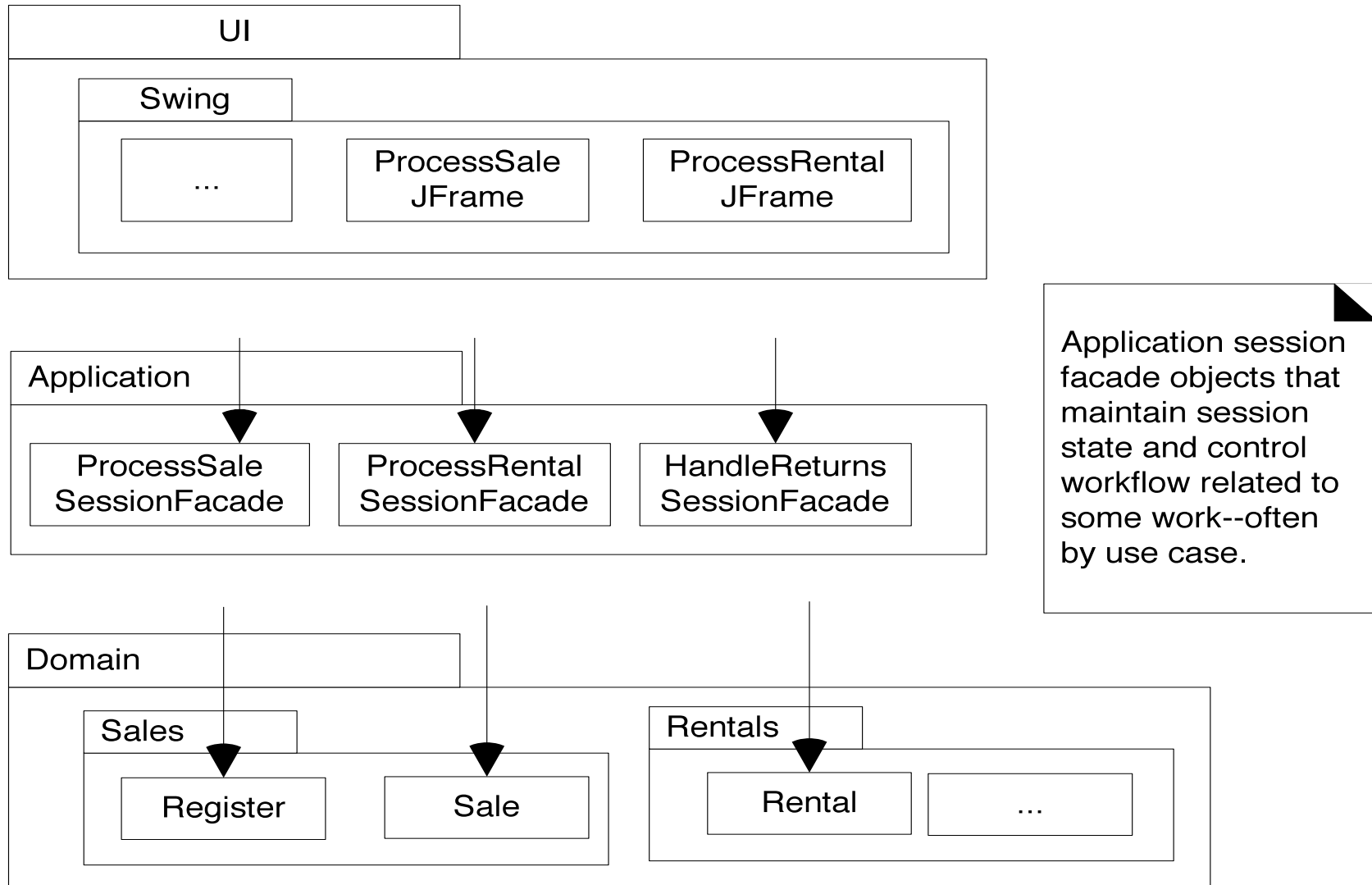


Application Coordination Layer

- When you have many system operations consider an “application coordination layer” whose objects are use case controllers



Another Example: Session Facades



Software Architectural Styles

Patterns for structuring architectures

Elements of Architecture Style

- Vocabulary of component and connector types
- Rules for combining architectural elements and/or constraints on valid combinations
- Interpretation rules
- Adaptation rules/guidelines, tradeoffs, rationale

Common Architecture Styles

- Pipes and filters
- Layered systems
- Event-based, implicit invocation
- Repositories

Pipes and Filters - Characteristics

- Components: filters
 - A filter consumes inputs and produces outputs.
- Connectors: pipes
 - inputs and outputs are transferred via pipes.

Pipes and Filters - Properties

- A filter's behavior is independent of behavior of other filters.
- Filters do not have knowledge of where their inputs originate from or to where their outputs are directed.
- Correctness of system should not depend on order in which filters are executed.

Pipes and Filters - Specializations

- Pipelines: topology restricted to linear sequences of filter
- Bounded pipes: amount of data on pipes bounded
- Typed pipes: type of data transmitted on pipes restricted
- Batch sequential systems: filters consume all inputs as a single entity

Pipes and Filters - Examples

- Unix piping mechanisms
- Compilers

Pipes and Filters - Benefits

- Supports reuse
- Easy to maintain and enhance
- System can be understood as a simple composition of behaviors
- Specialized analysis of throughput and deadlock
- Concurrency

Pipes and Filters - Drawbacks

- Interactive use not supported
- Data representation may not be optimal for processing in all filters, resulting in some parsing and unparsing (this could have a negative impact on performance)
- Could be required to maintain an invariant property over two separate but related streams

Layered Structures - Characteristics

- Components: subsystems (called layers) that provide services
- Links: protocols that define how the modules will interact

Layered Architectures - Properties

- In some systems layers are hidden from all except their adjacent outer layers.
 - Such layers are called virtual machines
- Some systems limit interactions only to the adjacent layers.

Layered Architectures - Benefits

- Support designs based on increasing levels of abstraction.
- Limited interaction between layers facilitates maintenance.
- Layers can be replaced by modules with identical interfaces (basis for defining layering standards).

Layered Systems - Drawbacks

- Layered architecture may have performance costs.
 - Performance considerations may require closer coupling of layers
- May be difficult to find right layer of abstraction for services.

Layered Structure - Examples

- Layered communication protocols (e.g., OSI)
- Database systems: end-user layer -> application layer -> DBMS -> physical database
- Operating systems: user layer -> utilities layer -> kernel

Event Based Systems - Characteristics

- Components: subsystem with a service interface
- Links: Explicit and implicit service (procedure) calls
 - Implicit call: when a subsystem (module) broadcasts an event all services registered with the event are invoked

Implicit Calls

- Components register an interest in an event by associating a service (procedure) with the event.
- When event is broadcast, the system invokes all services registered with the event.
- Examples: model analysis/simulation systems.

Implicit Calls - Properties

- Event broadcasters do not know which services will be invoked.
 - Such modules cannot make assumptions about the order in which services will be executed, or which services will be executed.
 - Explicit calls are used if control over services is needed.

Implicit Calls - Benefits

- Eases maintenance and extensions
 - A new behavior can be added by registering a new service with an event.
 - A component can be replaced by another component with an identical interface.

Implicit Calls - Drawbacks

- Components lose control of actions taken as a result of generating events.
- If generation of an event requires access to a global data repository by a number of services, concurrency control is needed. This could result in degradation of services.
- Verification of properties problematic.

Implicit Calls - Examples

- Programming environments
- Database systems - to ensure consistency constraints
- User interfaces - to separate representation from data management
- Syntax-directed editors - to support incremental semantic checking

Control Models

- Concerned with how modules are controlled/coordinated
- Two general approaches:
 - Centralized control
 - Event-based control

Centralized Control

- Subsystem (module) coordinates system execution; this subsystem is called the system controller.
- Forms:
 - Call - return model
 - Manager model

Event - Driven Systems

- Coordination is distributed among modules.
- Two types:
 - Broadcast models: Events are broadcast to all modules; any module that can react will react.
 - Interrupt - driven models: Interrupts are detected by a handler and appropriate action is taken.

Broadcast Model

- Efficient variant
- Modules register interest in event with handler
- When handler detects an event it checks the register and passes the event to registered modules

Interrupt - Driven Model

- Allows very fast responses to events
- Complex to program
- Difficult to validate

Repositories - Characteristics

- Components: a central data structure and components that interact with the data structure.
- Links: varies ...

Repositories - Variants

- 2 major classes of repositories
 - Databases: input streams determine the services to be executed
 - Blackboards: the state of the repository determines the services to be executed

Heterogeneous Architectures

- Most systems involve a combination of architectural styles.
- Hierarchical heterogeneity: a component's (or link's) internal structure may have different style.
- Link heterogeneity: a component may use a variety of links to interact with others.
- Level heterogeneity: A level may be elaborated using a different style.