Basic Concepts

Software Architecture
Lecture 3

Learning Objectives

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

What is Software Architecture?

- **Definition:**
  - A software system's architecture is the set of *principal design decisions* about the system
- Software architecture is the blueprint for a software system's construction and evolution
- Design decisions encompass every facet of the system under development
  - Structure
  - Behavior
  - Interaction
  - Non-functional properties

Examples of Design Decisions

- System Structure (e.g., central component)
-
- Functional behaviour (e.g., sequence of opeations)
- Interactions (e.g., event notifications)
- Non-functional properties (e.g., no single point of failure)
- System's Implementation (e.g., Using Java Swing toolkit)

What is "Principal"?

- "Principal" implies a degree of importance that grants a design decision "architectural status"
  - It implies that not all design decisions are architectural
  - That is, they do not necessarily impact a system's architecture
- How one defines "principal" will depend on what the stakeholders define as the system goals

Temporal Aspect

- Design decisions are and unmade over a system's lifetime → Architecture has a temporal aspect
- At any given point in time the system has only one architecture
- A system's architecture will change over time
  - Architectures can be forked, converge etc.
  - Typically many related architectures are in play

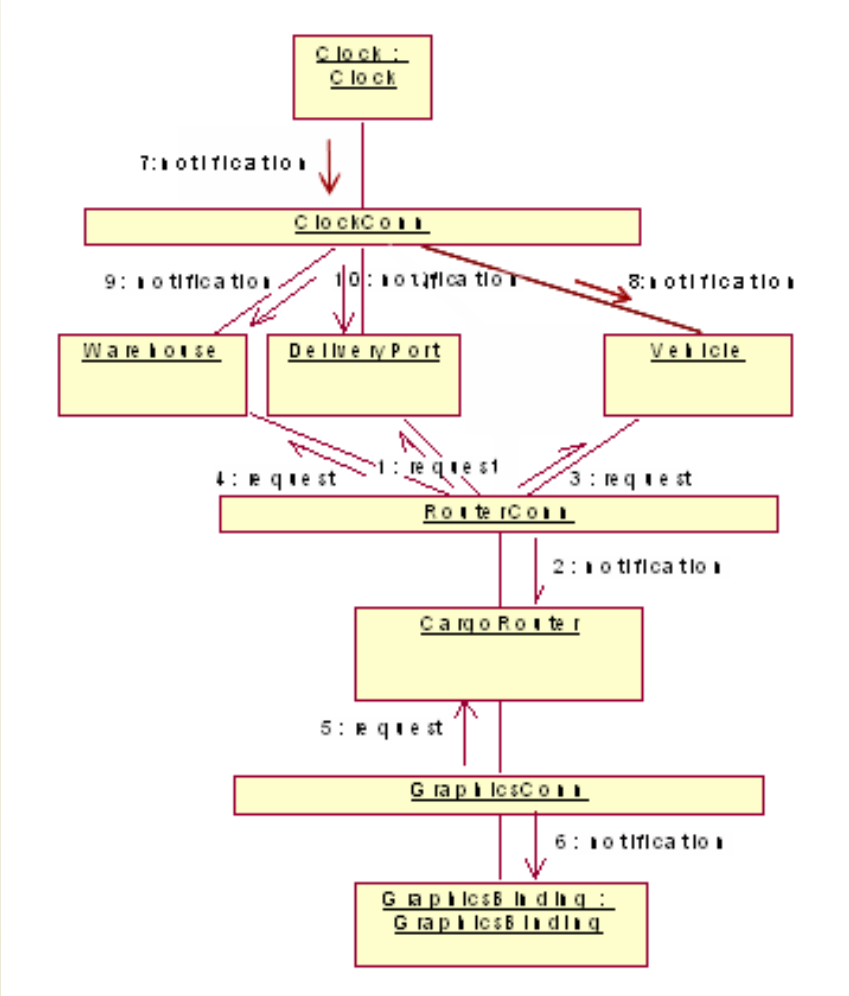
Learning Objectives

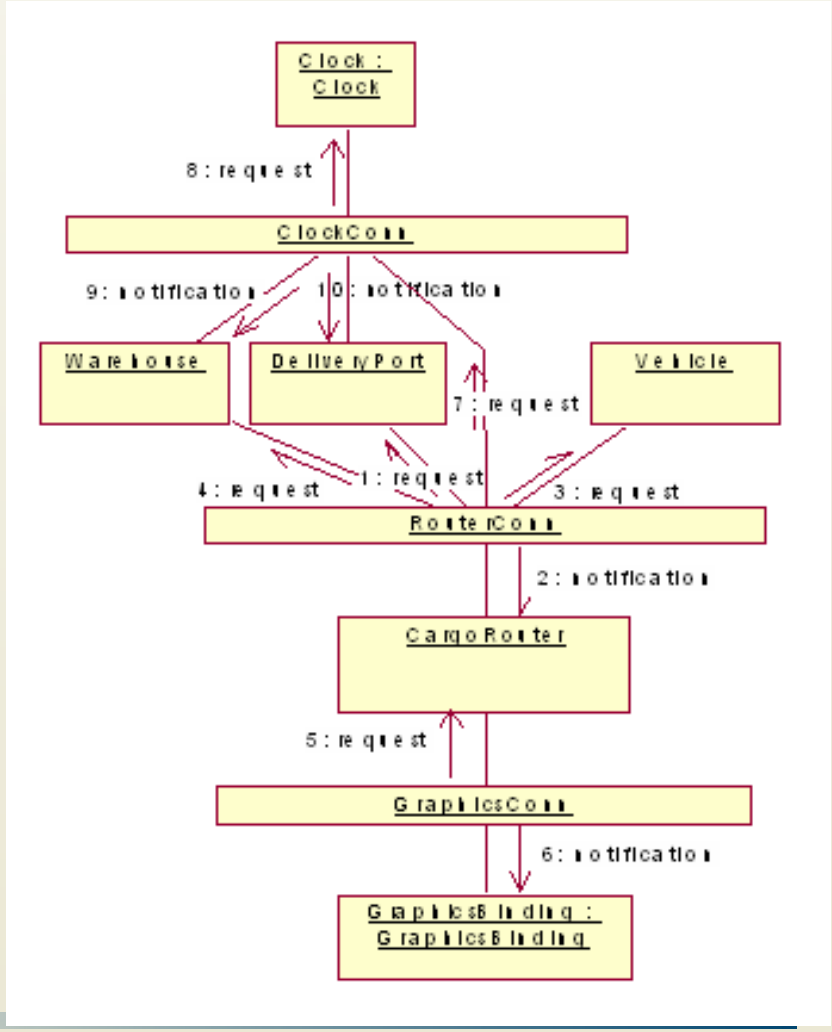
- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

Prescriptive vs. Descriptive Architecture

- A system's *prescriptive architecture* captures the design decisions made prior to the system's construction
  - It is the *as-conceived* or *as-intended* architecture
- A system's *descriptive architecture* describes how the system has been built
  - It is the *as-implemented* or *as-realized* architecture

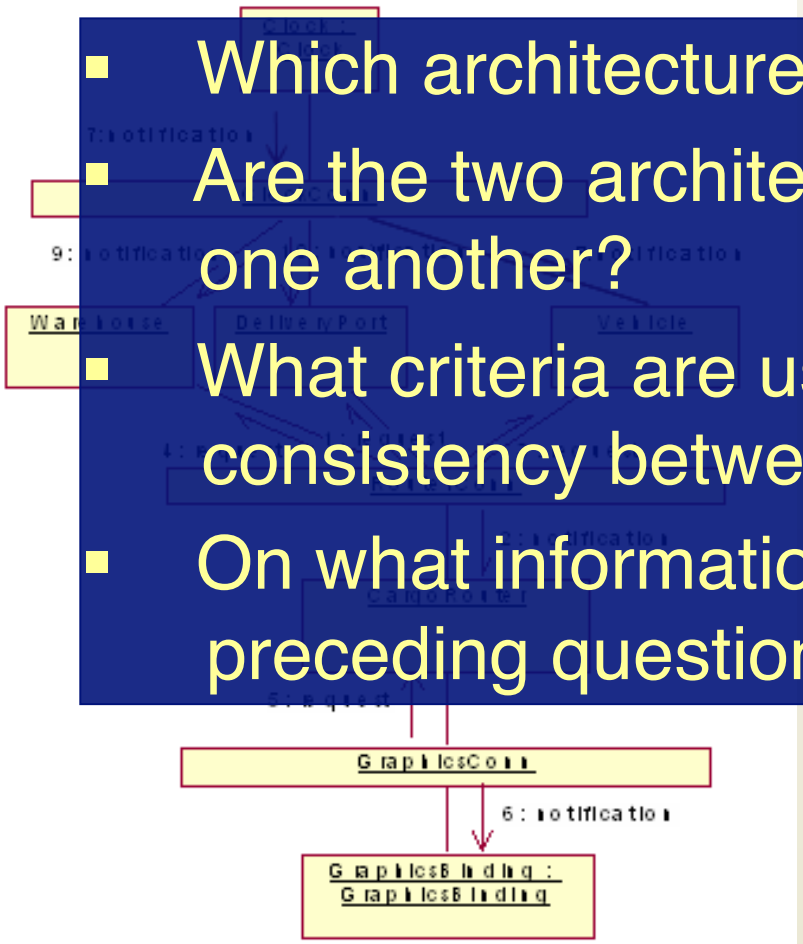
Prescriptive vs. Descriptive

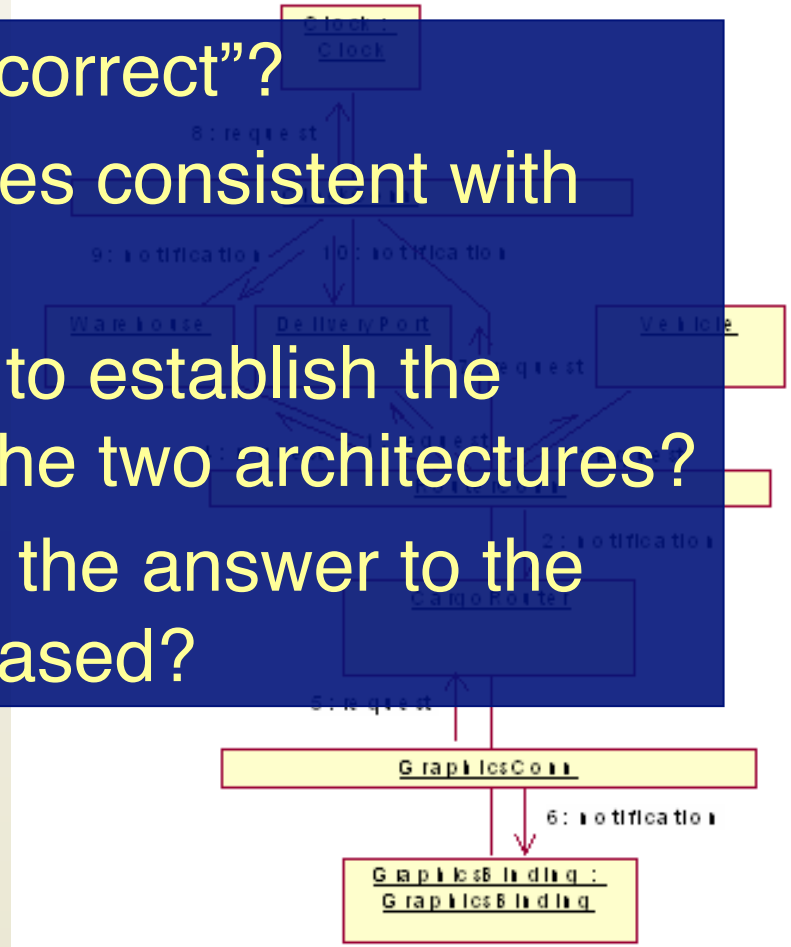




Prescriptive vs. Descriptive

- Which architecture is “correct”?
- Are the two architectures consistent with one another?
- What criteria are used to establish the consistency between the two architectures?
- On what information is the answer to the preceding questions based?





Learning Objectives

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

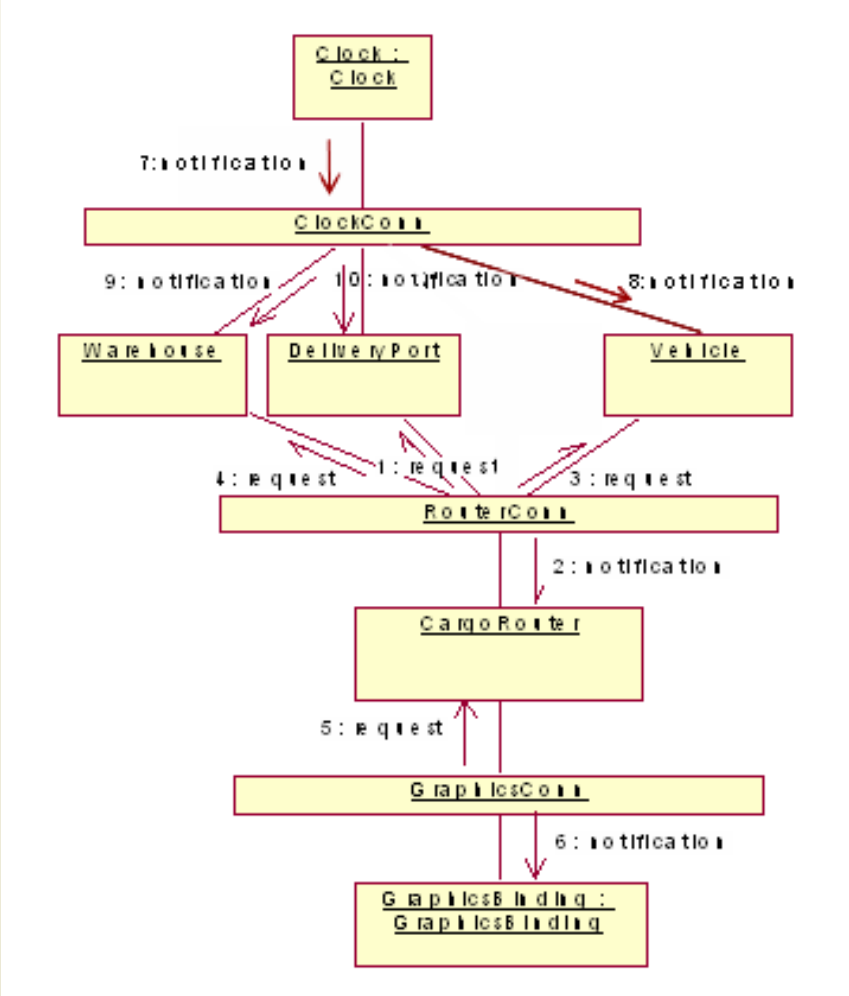
Architectural Evolution

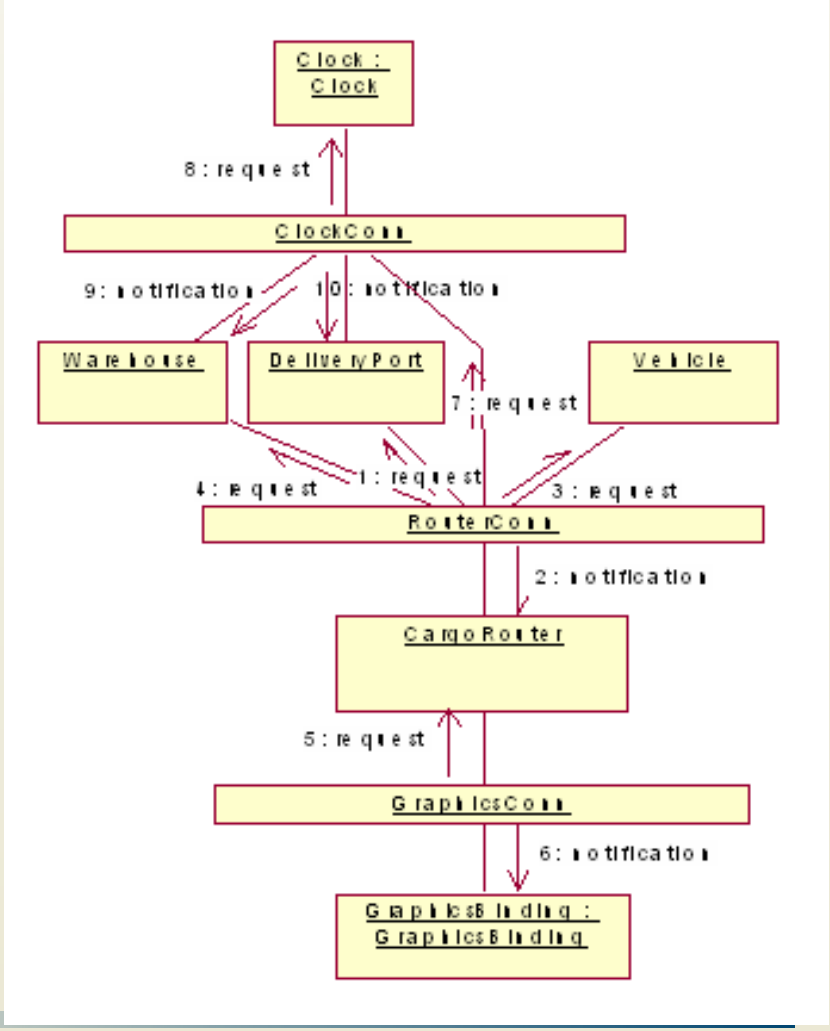
- When a system evolves, ideally its prescriptive architecture is modified first
- In practice, the system – and thus its descriptive architecture – is often directly modified
- This happens because of
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

Architectural Degradation

- Two related concepts
  - Architectural drift
  - Architectural erosion
- *Architectural drift* is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which do not violate any of the prescriptive architecture's design decisions
- *Architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture

Architectural Drift or Erosion ?

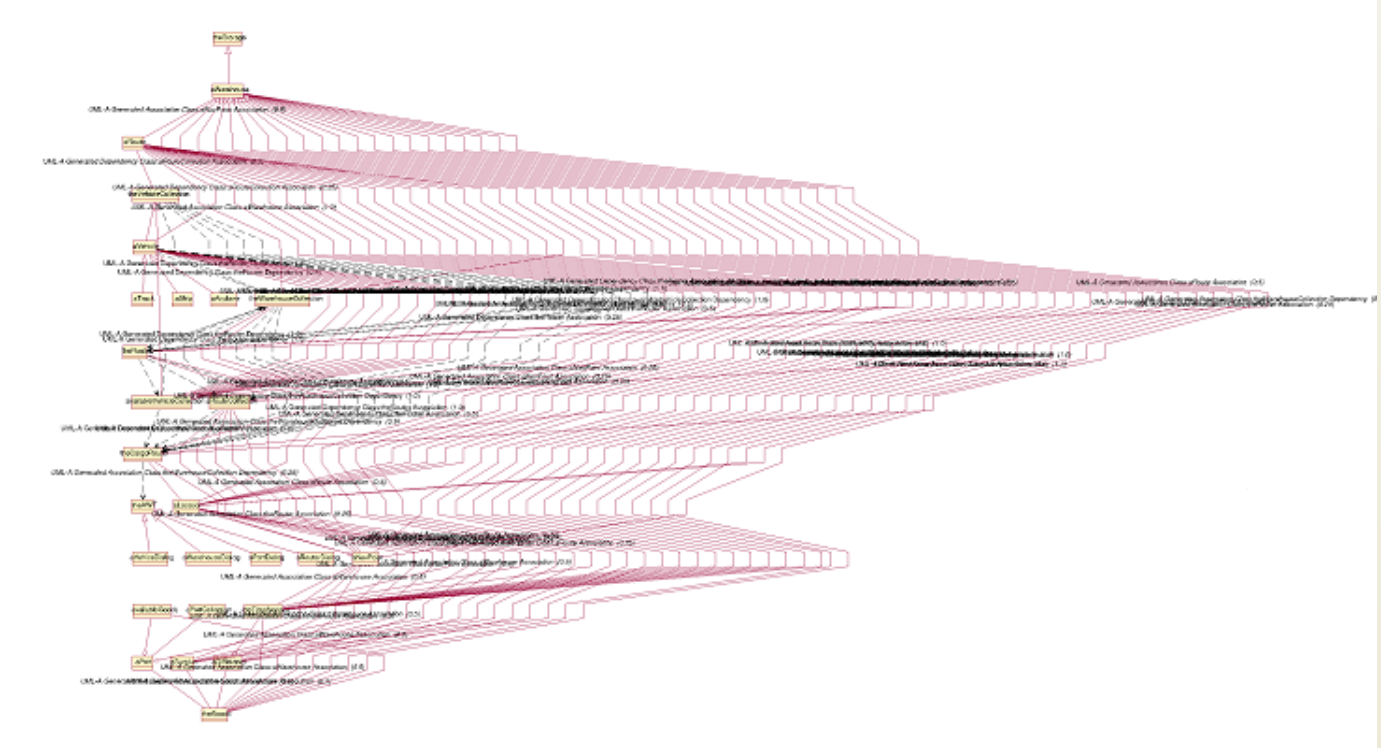




Architectural Recovery

- If architectural degradation is allowed to occur, one will be forced to *recover* the system's architecture sooner or later
- *Architectural recovery* is the process of determining a software system's architecture from its implementation-level artifacts
- Implementation-level artifacts can be
  - Source code
  - Executable files
  - Java .class files

Can you recover this architecture ?



Learning Objectives

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

Software Architecture's Elements

- A software system's architecture typically is not (and should not be) a uniform monolith
- A software system's architecture should be a composition and interplay of different elements
  - Processing
  - Data, also referred as information or state
  - Interaction

Components

- Elements that encapsulate processing and data in a system's architecture are referred to as *software components*
- **Definition**
  - A *software component* is an architectural entity that
    - encapsulates a subset of the system's functionality and/or data
    - restricts access to that subset via an explicitly defined interface
    - has explicitly defined dependencies on its required execution context
- Components typically provide application-specific services

Examples of Components

- Application-specific components
  - Examples: Cargo, warehouse, vehicle
- Limited reuse components
  - Examples: Web servers, clocks, connections
- Reusable components
  - Examples: GUI components, class and math libraries

Connectors

- In complex systems *interaction* may become more important and challenging than the functionality of the individual components
- **Definition**
  - A *software connector* is an architectural building block tasked with effecting and regulating interactions among components
- In many software systems connectors are usually simple procedure calls or shared data accesses
- Connectors typically provide application-independent interaction facilities
  - Can be described independent of the components

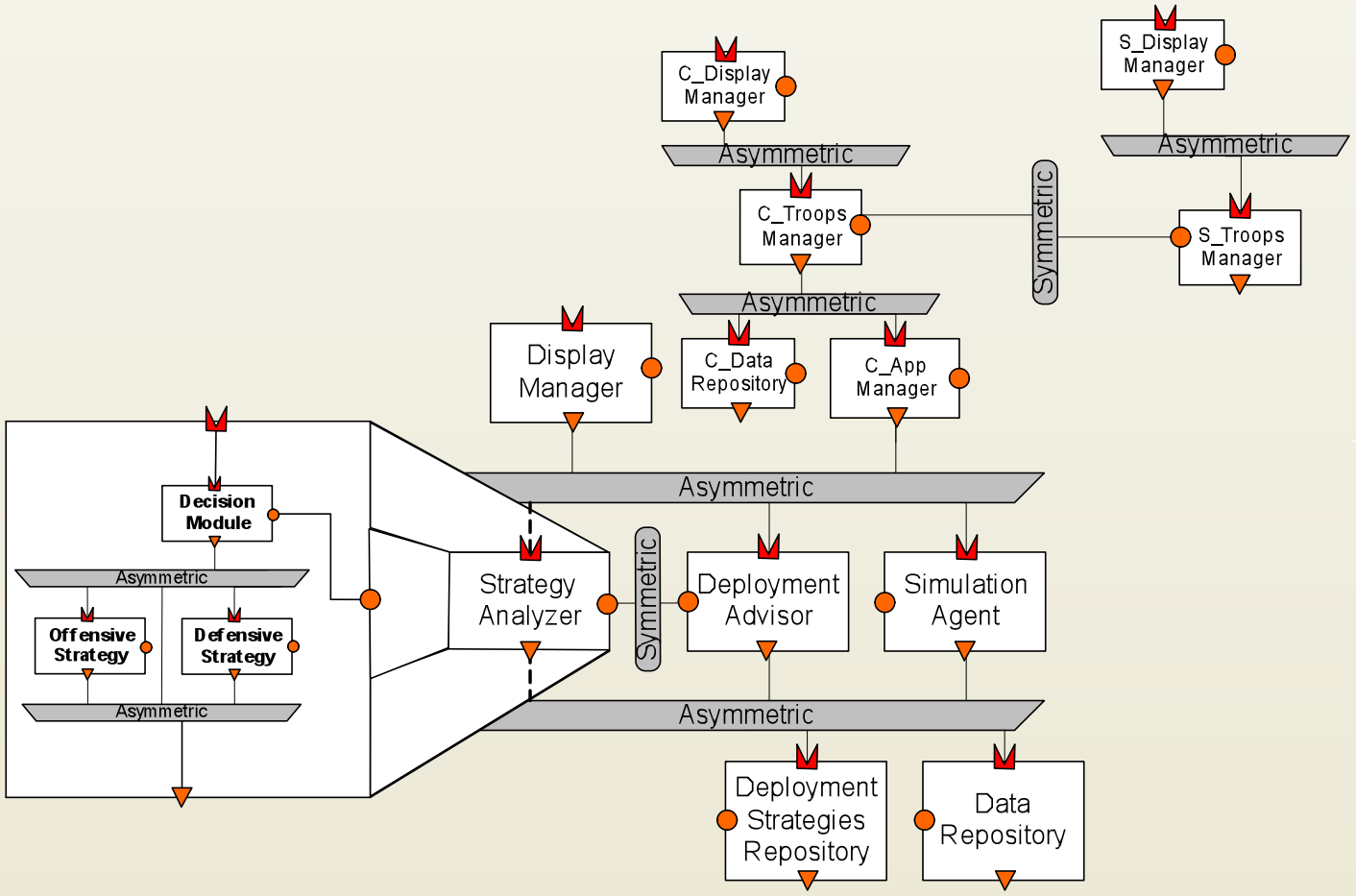
Examples of Connectors

- Procedure call connectors
- Shared memory connectors
- Message passing connectors
- Streaming connectors
- Distribution connectors
- Wrapper/adaptor connectors

Configurations

- Components and connectors are composed in a specific way in a given system's architecture to accomplish that system's objective
- **Definition**
  - An *architectural configuration*, or topology, is a set of specific associations between the components and connectors of a software system's architecture

An Example Configuration



Learning Objectives

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

Architectural Styles

- Certain design choices regularly result in solutions with superior properties
  - Compared to other possible alternatives, solutions such as this are more elegant, effective, efficient, dependable, evolvable, scalable, and so on
- **Definition**
  - An *architectural style* is a named collection of architectural design decisions that
    - are applicable in a given development context
    - constrain architectural design decisions that are specific to a particular system within that context
    - elicit beneficial qualities in each resulting system

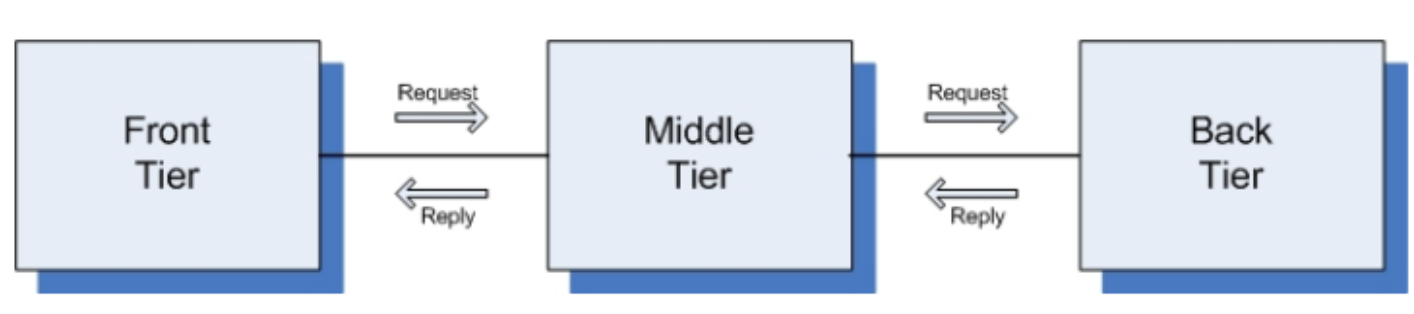
Architectural Style: Example

- REST style (Representational State Transfer) – HTTP
  - Uniform Interface between clients and servers
  - Stateless: No client context stored on server between requests. All state is carried in the request URL.
  - Clients should be able to cache responses to requests
  - Layered architecture: Clients cannot tell if they are connected directly to the server or thro' a proxy
  - Code on demand (optional): Server should be able to extend the client's functionality thro' client-side scripts

Architectural Patterns

- **Definition**
  - An *architectural pattern* is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears
- A widely used pattern in modern distributed systems is the *three-tiered system* pattern
  - Science
  - Banking
  - E-commerce
  - Reservation systems

Three-Tiered Pattern



- Front Tier
  - Contains the user interface functionality to access the system's services
- Middle Tier
  - Contains the application's major functionality
- Back Tier
  - Contains the application's data access and storage functionality

Differences between Style and Pattern

- **Style**
  - Provides a set of guiding principles in adopting solutions
  - Requires considerable effort to apply. Architect needs to justify the design choices based on the architectural style.
- **Pattern**
  - Provides concrete solutions, although parameterized to the specific problem.
  - Requires very little manual effort or justification to apply.
  - Usually applies to specific systems (e.g., GUI-based systems)

Learning Objectives

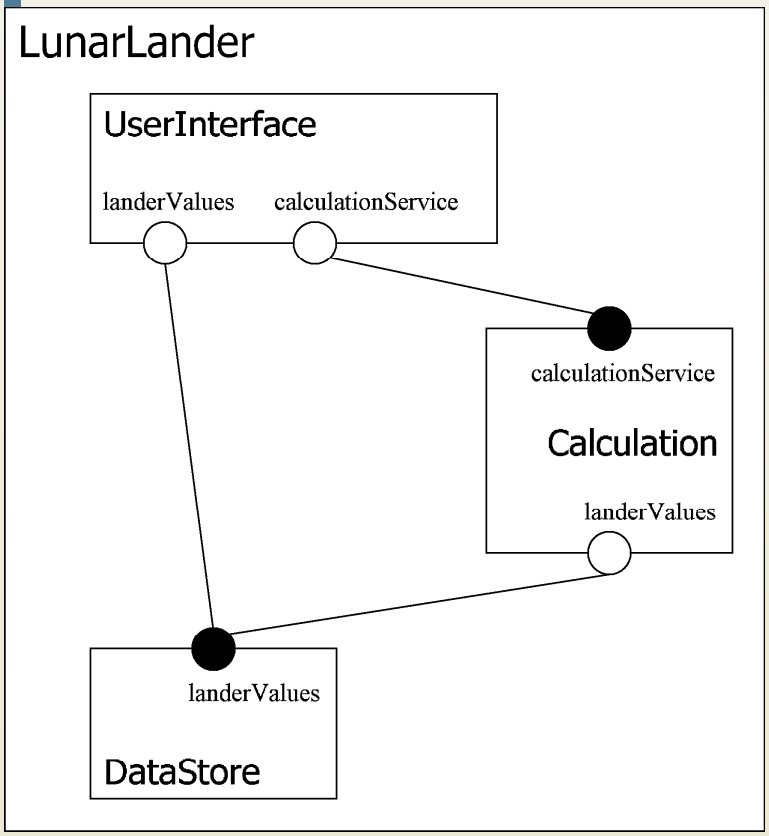
- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

Architectural Models, Views, and Visualizations

- **Architecture Model**
  - An artifact documenting some or all of the architectural design decisions about a system
- **Architecture Visualization**
  - A way of depicting some or all of the architectural design decisions about a system to a stakeholder
- **Architecture View/Perspective**
  - A subset of related architectural design decisions
  - Typically pertain to a cross-cutting functionality

Architectural Visualization: Example

Graphical Diagram

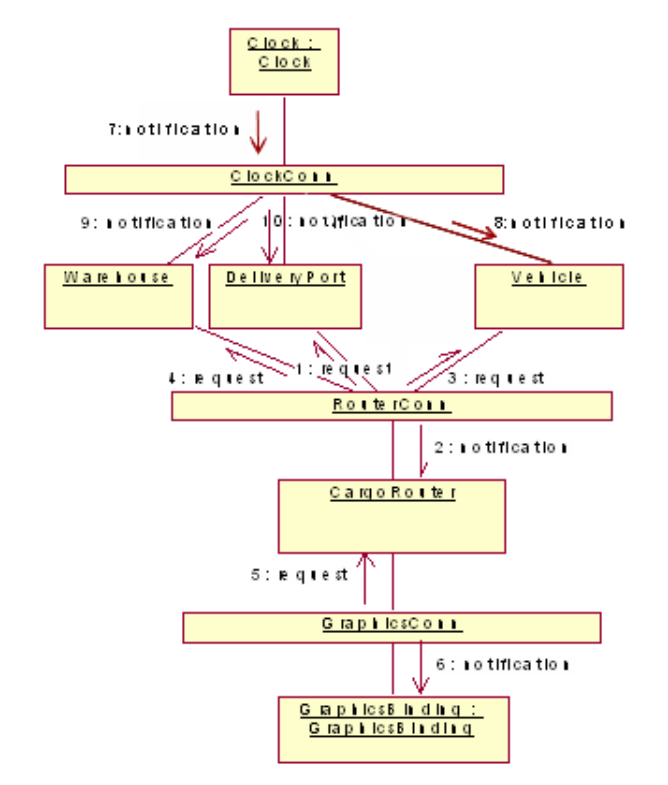


Textual descriptions

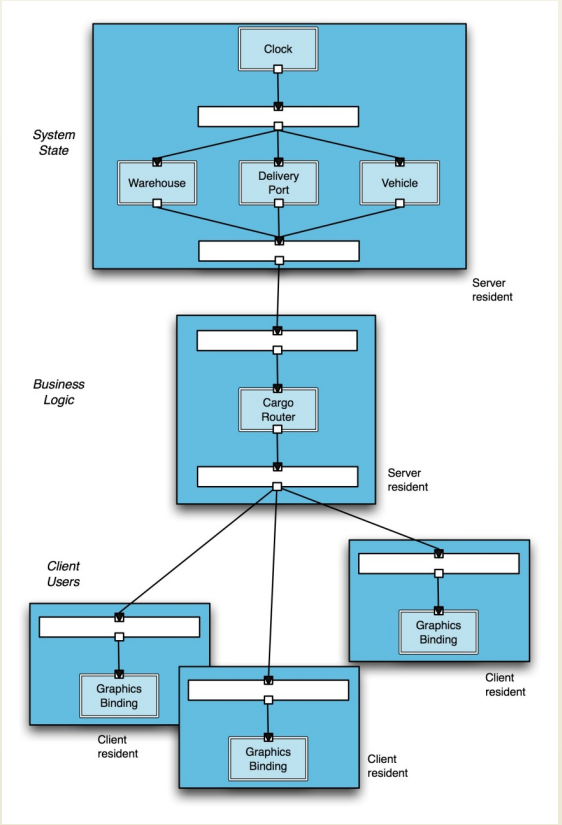
```
component DataStore{
    provide landerValues;
}

component Calculation{
    require landerValues;
    provide calculationService;
}

component UserInterface{
    require calculationService;
    require landerValues;
}

component LunarLander{
inst
    U: UserInterface;
    C: Calculation;
    D: DataStore;
bind
    C.landerValues -- D.landerValues;
    U.landerValues -- D.landerValues;
    U.calculationService --
C.calculationService;
}
```

Architectural Views: Example

Structural View



Deployment View



Learning Objectives

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture