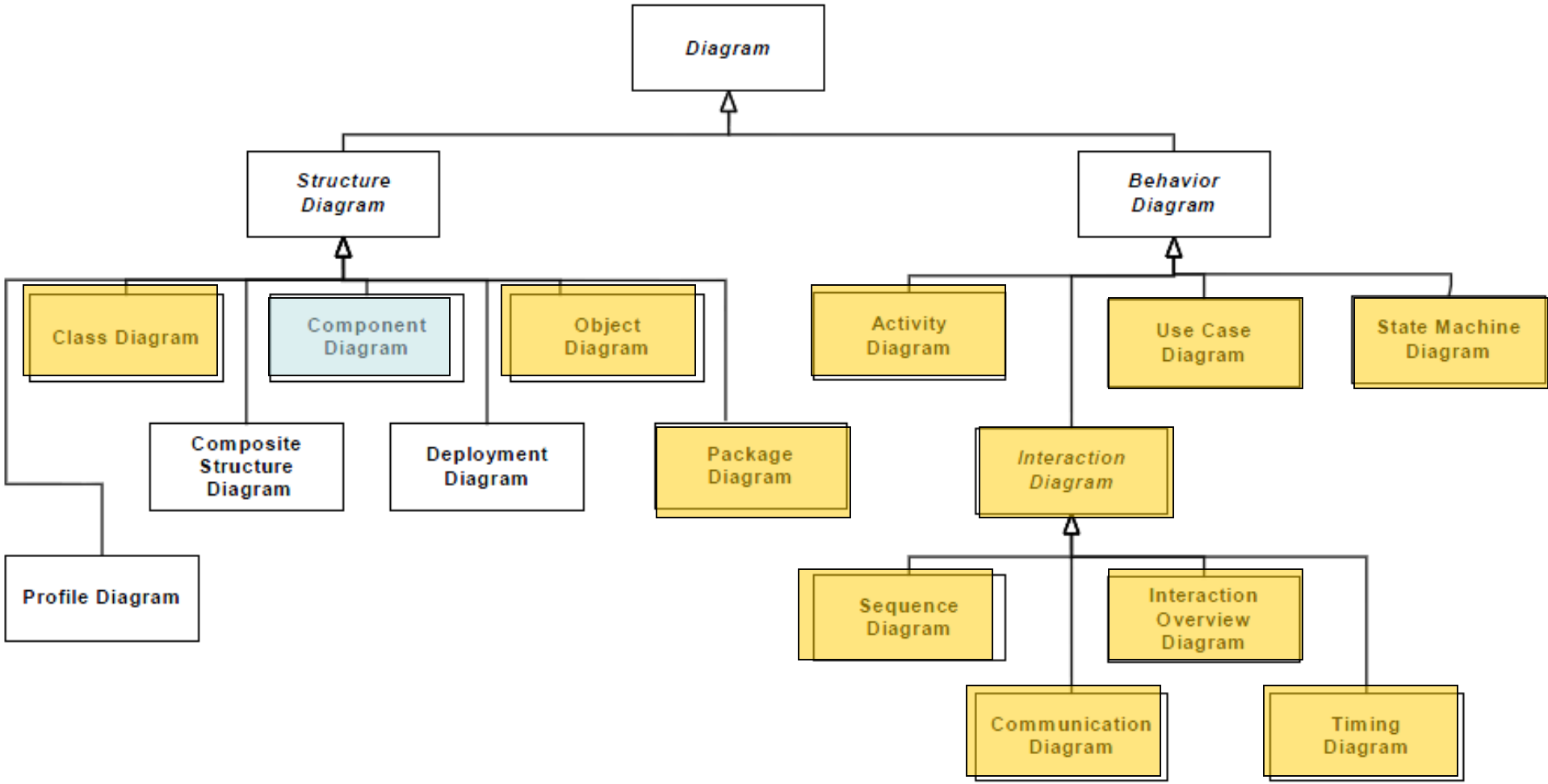
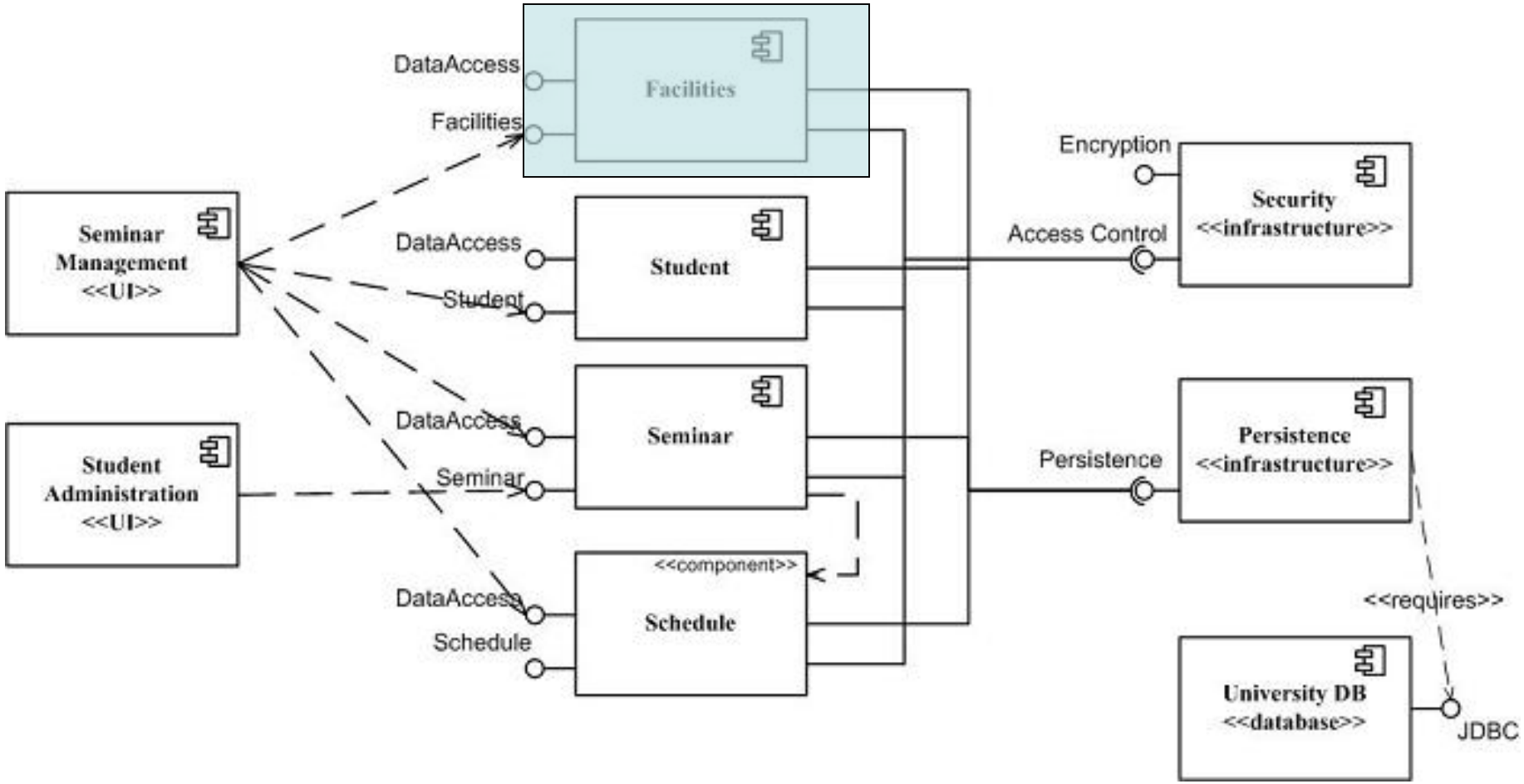
Object-Oriented Technology and UML

Component Diagram

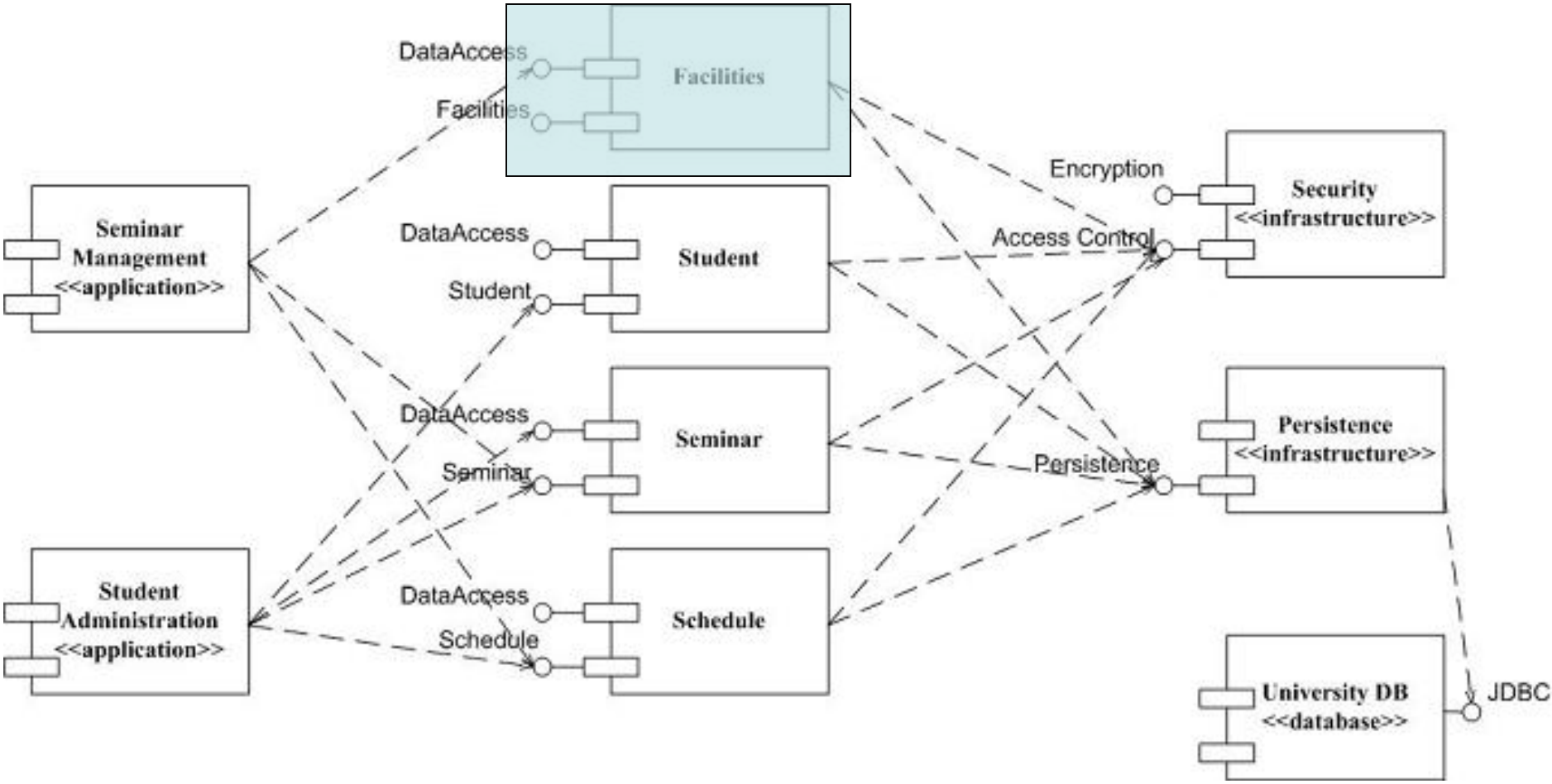
Component Diagram



Example of Component Diagram



Example of Component Diagram



Component Diagram

- A diagram that shows the organization of and dependencies among a set of components
- Component diagrams address the static implementation view of a system

Component

- A component is a replaceable part of a system that conforms to and provides the realization of a set of interfaces

Component

- Component diagram has a higher level of abstraction than a Class Diagram
- Usually a component is implemented by one or more classes (or objects) at runtime
  - They are building blocks so a component can eventually encompass a large portion of a system

Components and Interfaces

- An interface is a collection of operations that specify a service that is provided by or requested from a class or component
  - An interface that a component realizes is called a provided interface, meaning an interface that the component provides as a service to other components. A component may declare many provided interfaces
  - The interface that a component uses is called a required interface, meaning an interface that the component conforms to when requesting services from other components. A component may conform to many required interfaces
  - Also, a component may both provide and require interfaces

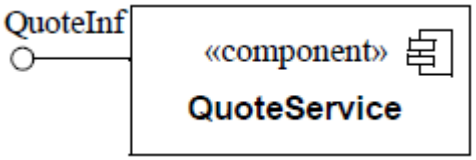
Categories of components

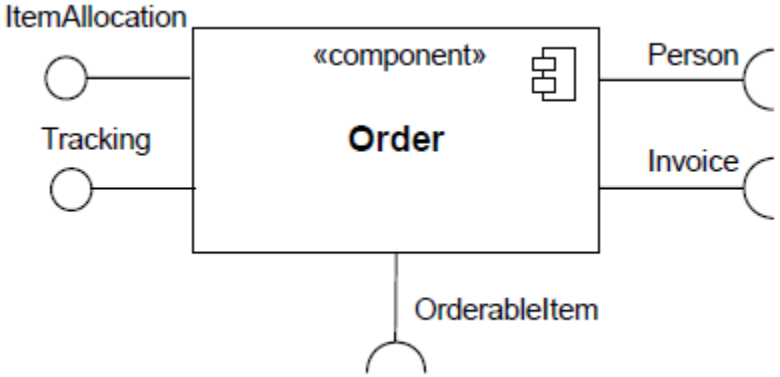
- Deployment
- Work Product
- Execution Component

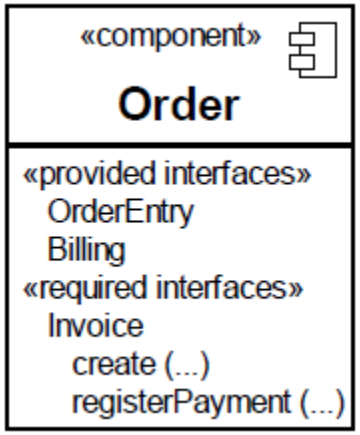
Representation of component

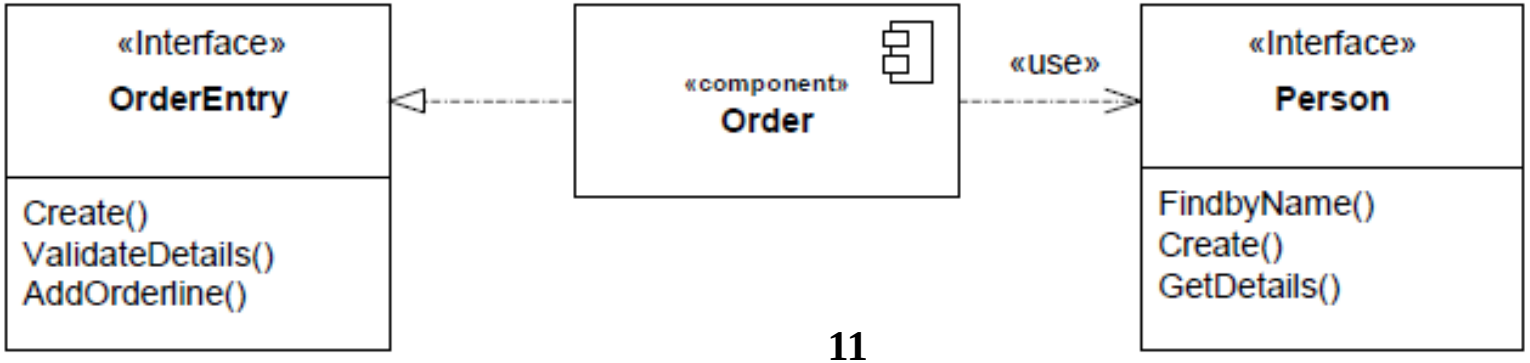
- A component is shown as a rectangle with a small two-pronged icon in its upper right corner
- The name of the component appears in the rectangle
- A component can have attributes and operations, but these are often elided in diagrams

Representation of component

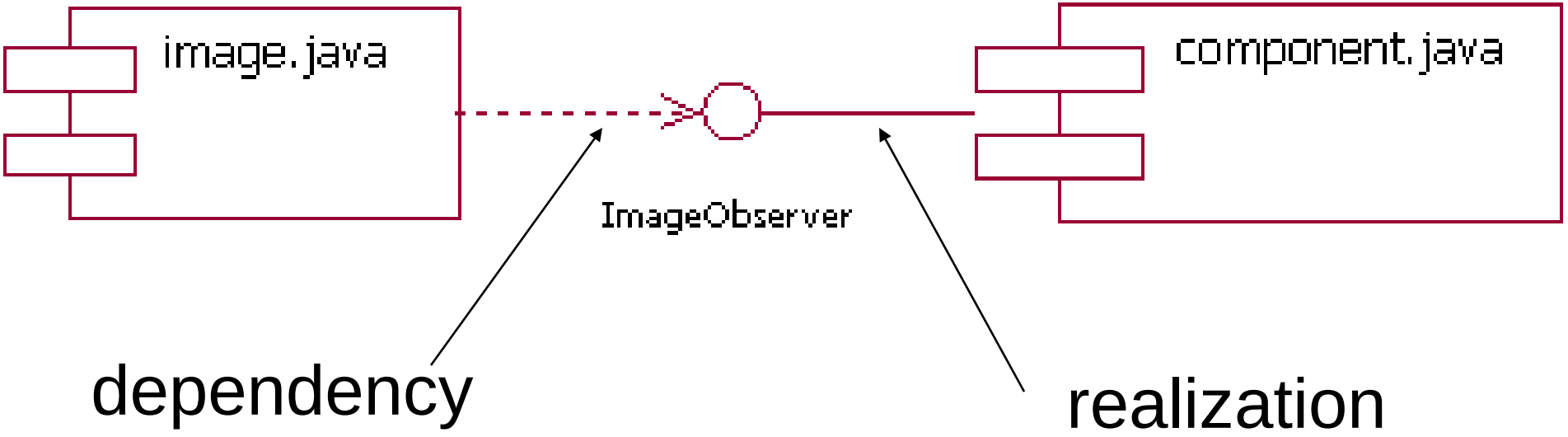




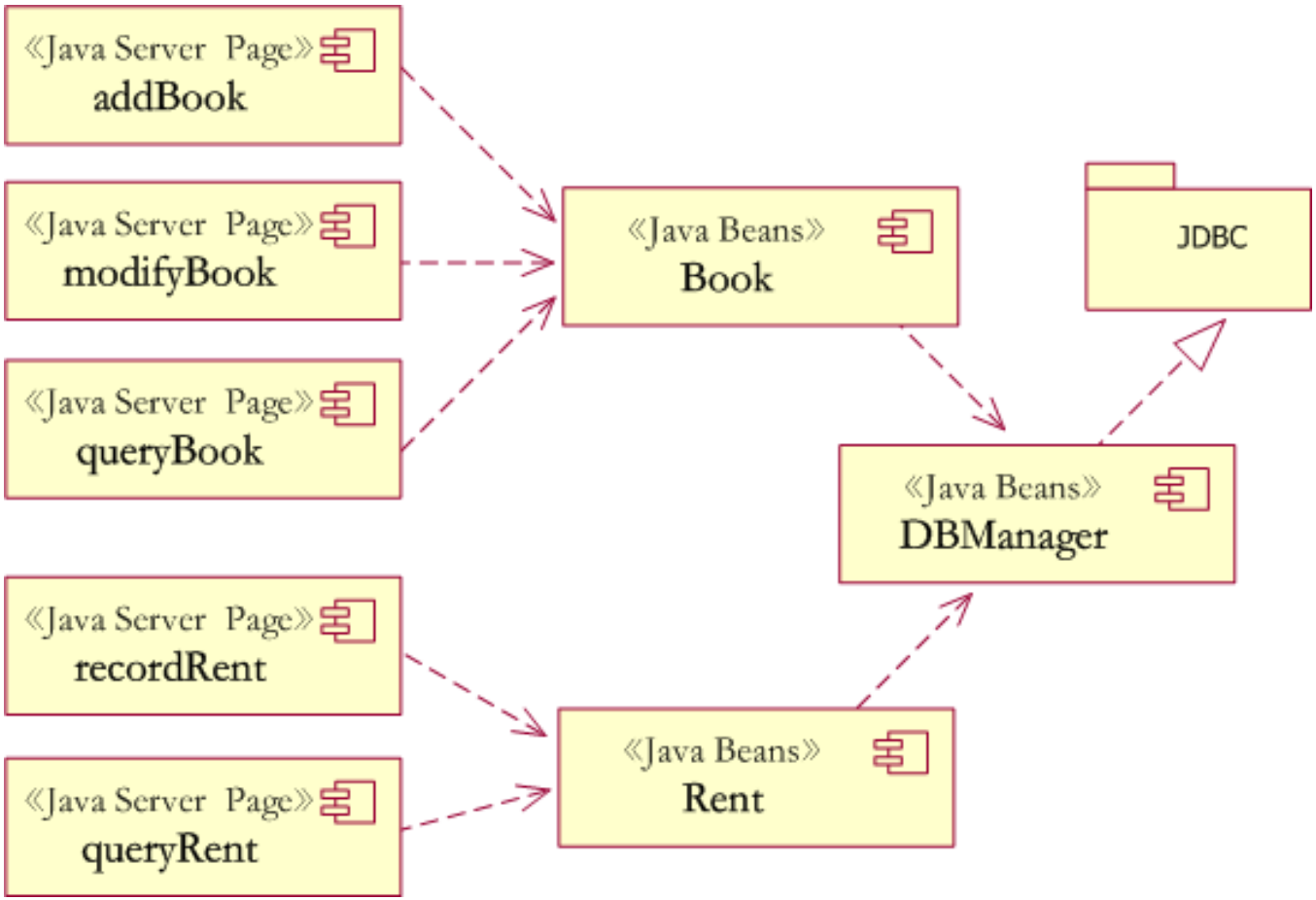




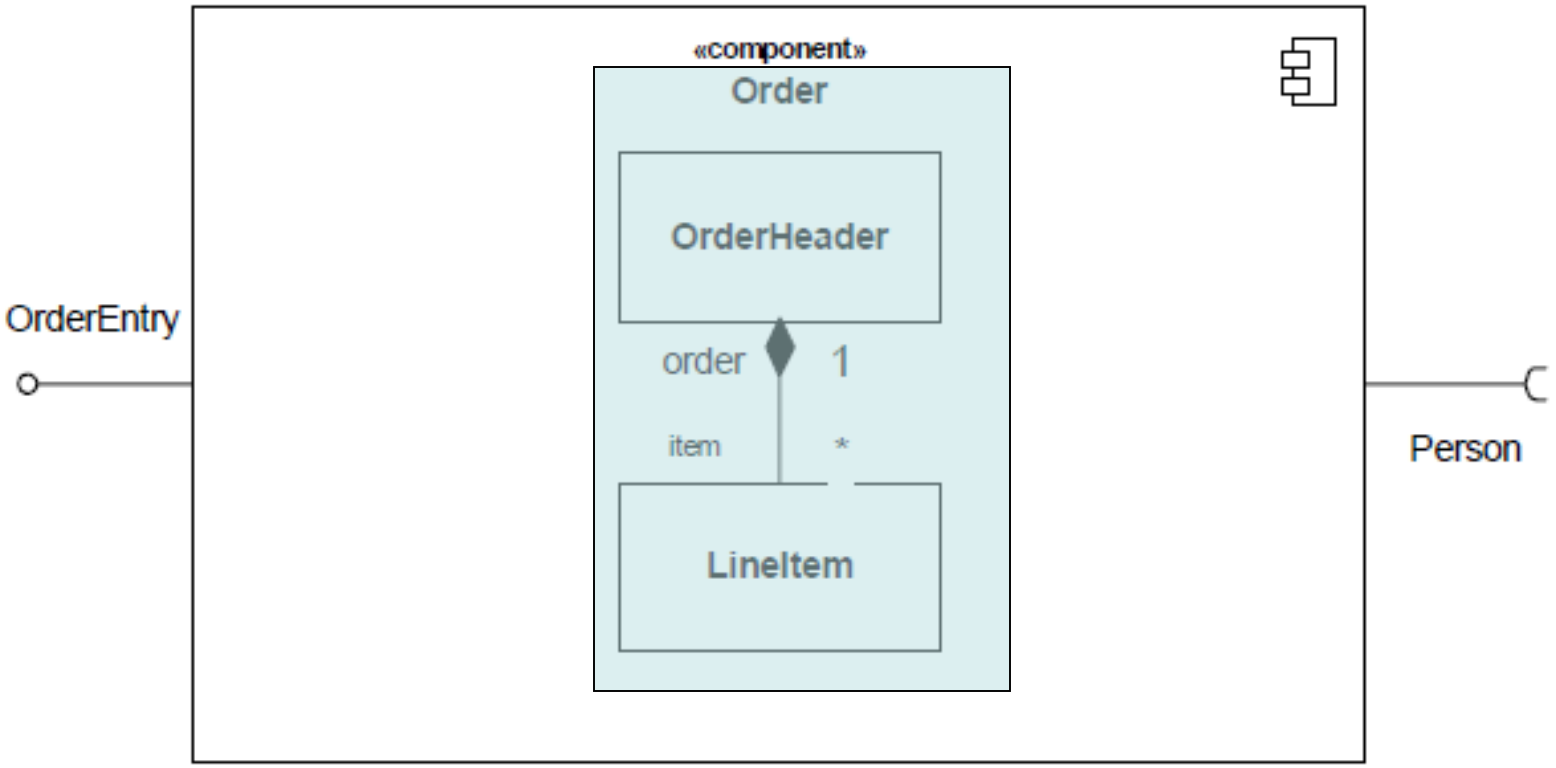
Relationship between a component and its interface



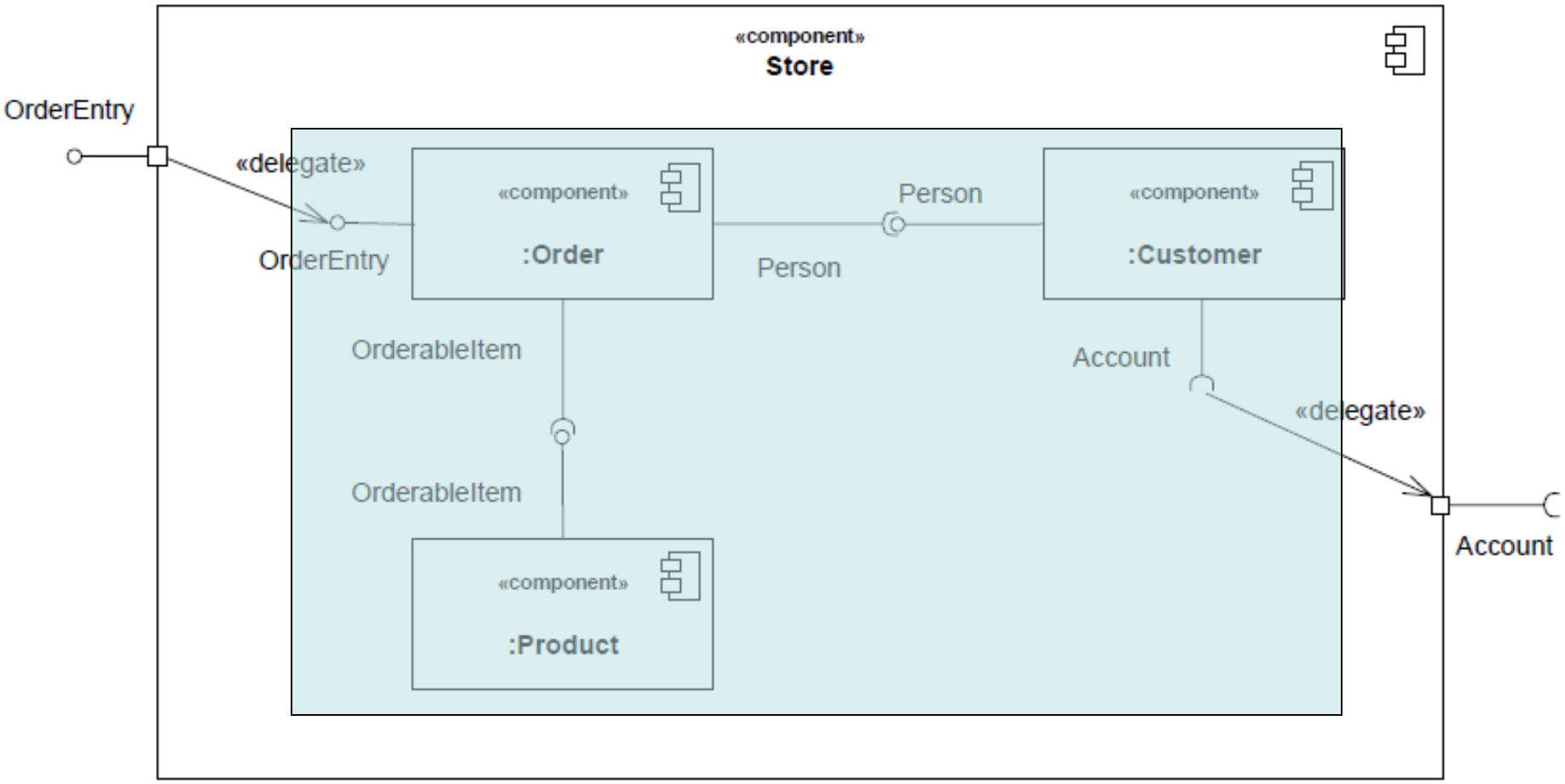
Basic component diagram



Nested component diagram



Nested component diagram



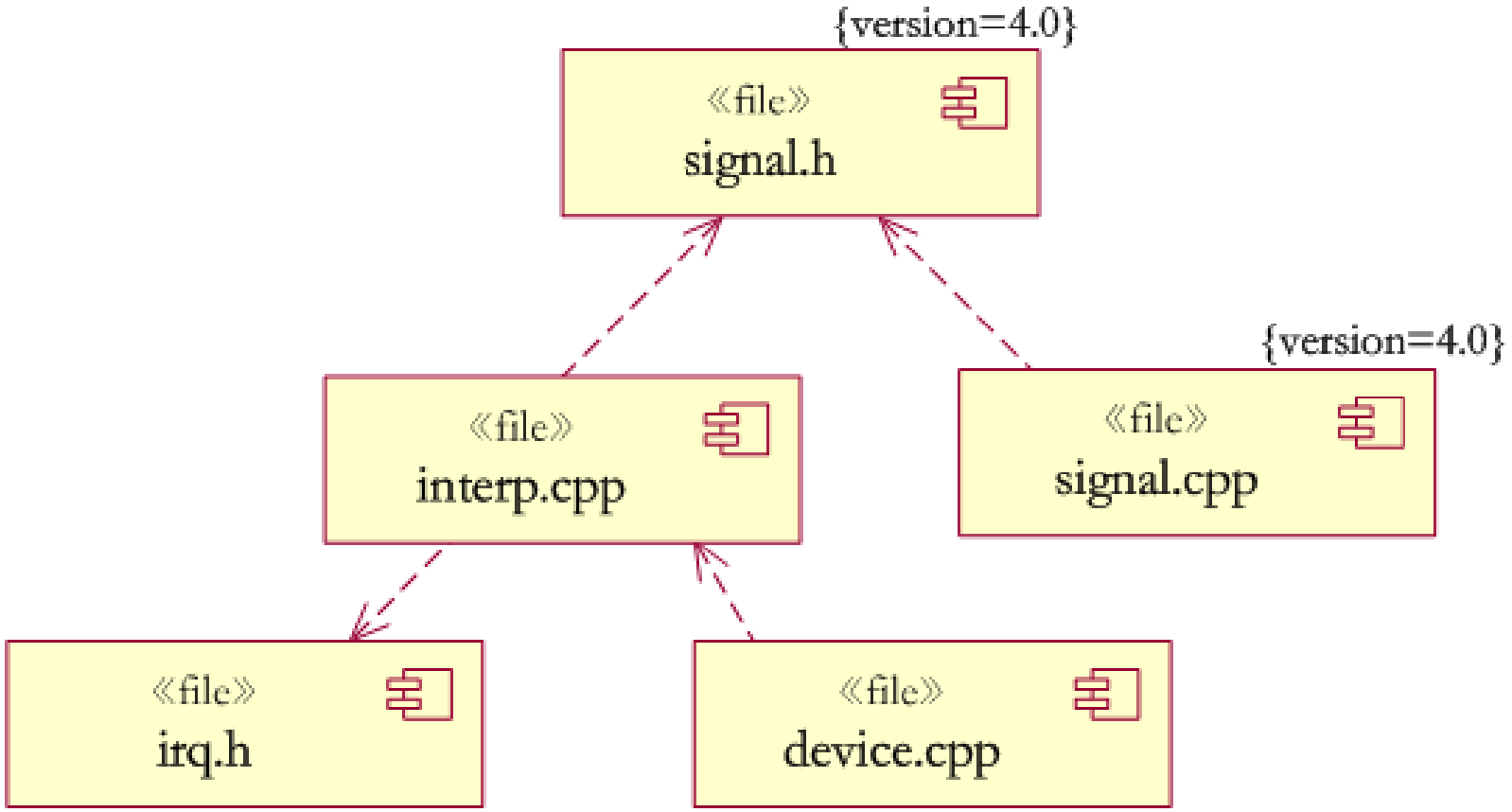
Review of Component Diagram

- The Component Diagram helps to model the physical aspect of an Object-Oriented software system
- It illustrates the architectures of the software components and the dependencies between them
- Those software components including run-time components, executable components also the source code components.

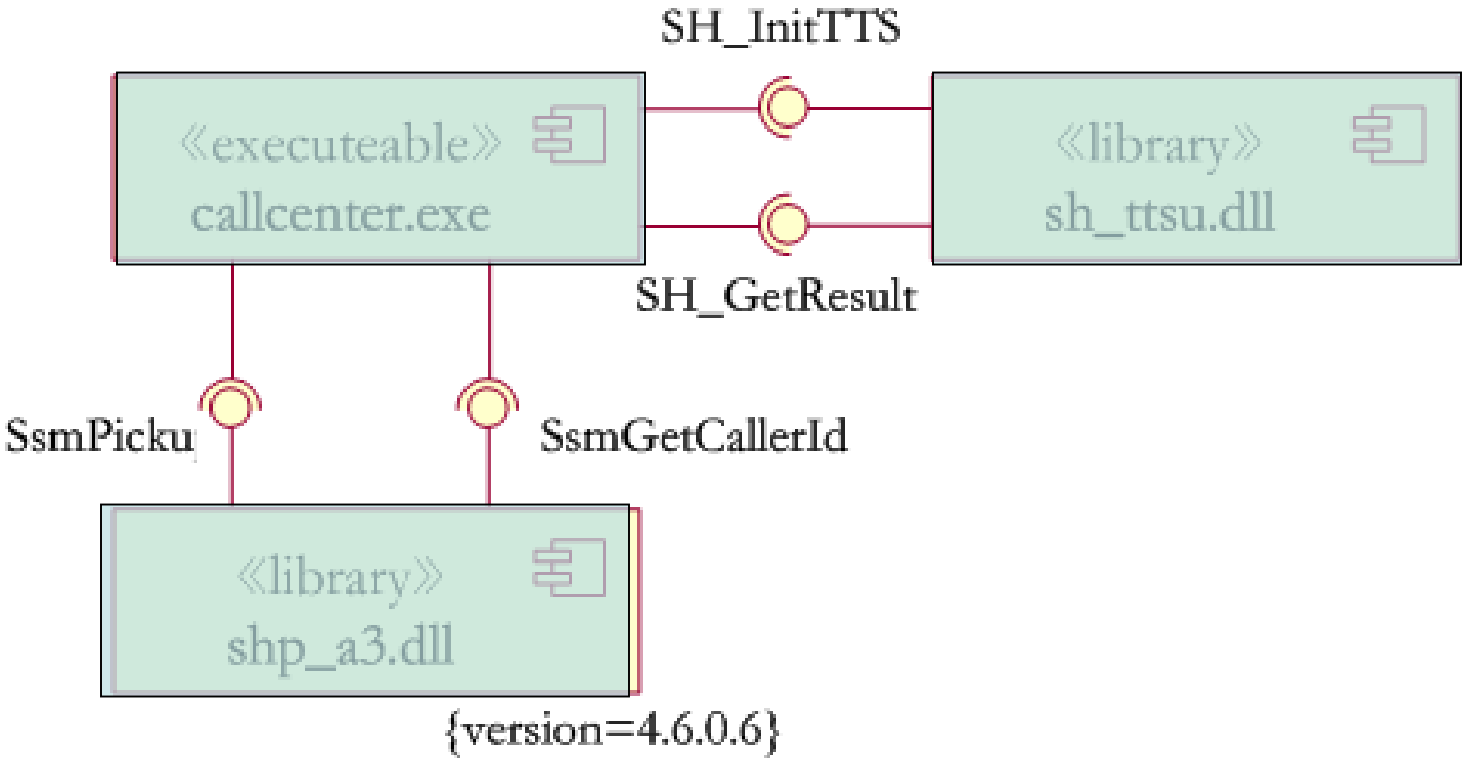
Usage of component diagrams

- Model the components of a system
- Model database schema
- Model executables of an application
- Model system’s source code

Example: source code



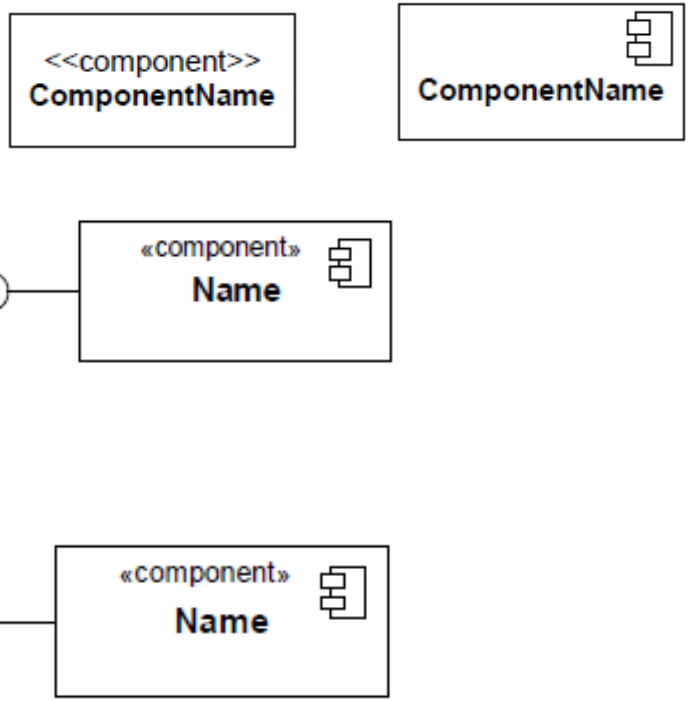
Example: executables of an application



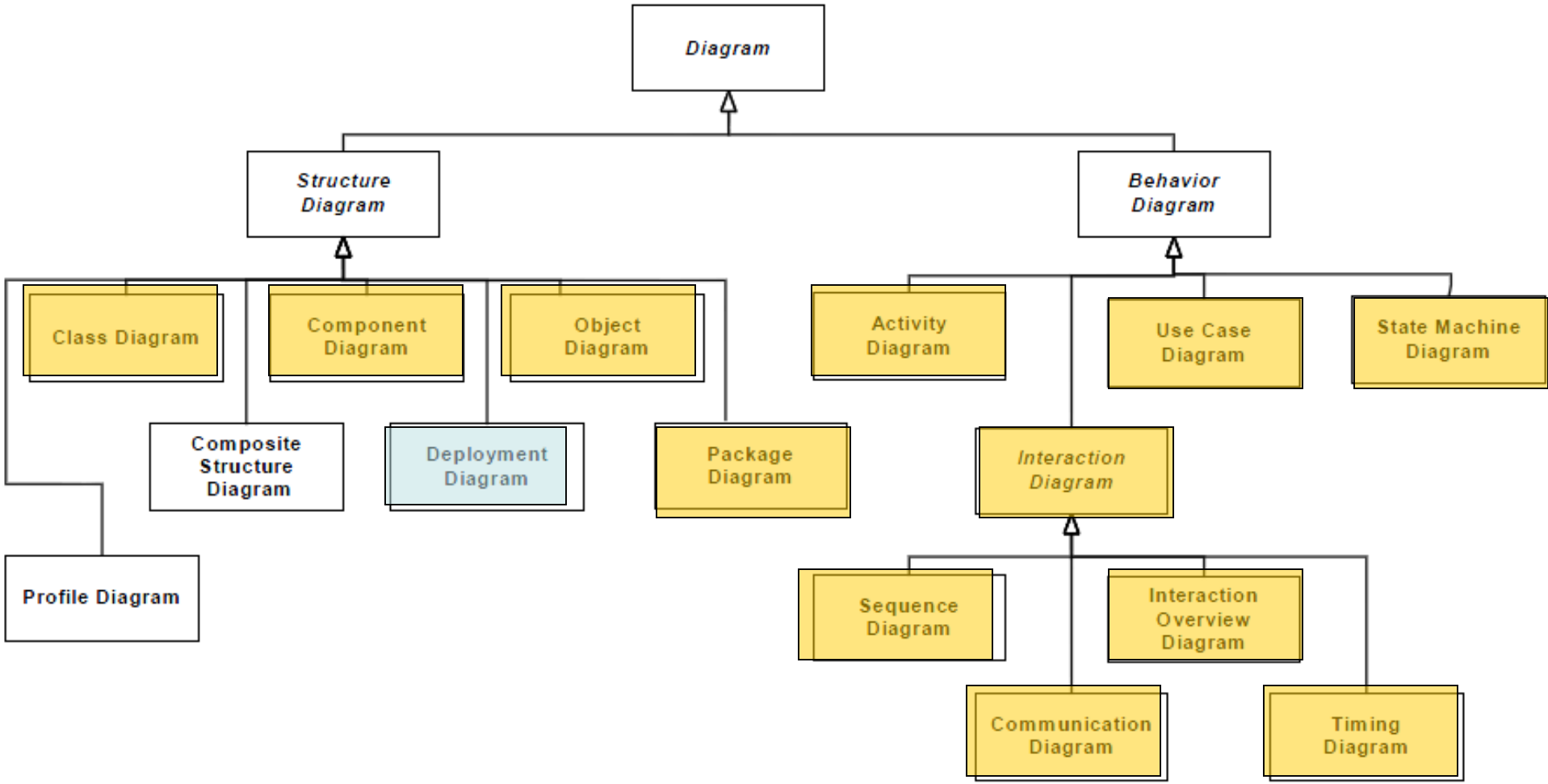
Style of UML Component Diagram

- Apply One Component Stereotype Consistently
  - Choose either the <<component>> or the bandage-box symbol and apply it consistently
- Show Only Relevant Interfaces
  - Depict only the interfaces that are applicable to the goals of your diagram
- Make Components Dependent Only on Interfaces
  - By making components dependent on the interfaces of other components, instead of on the other components themselves

Q & A


<<component>>
ComponentName
ComponentName
«component»
Name
«component»
Name

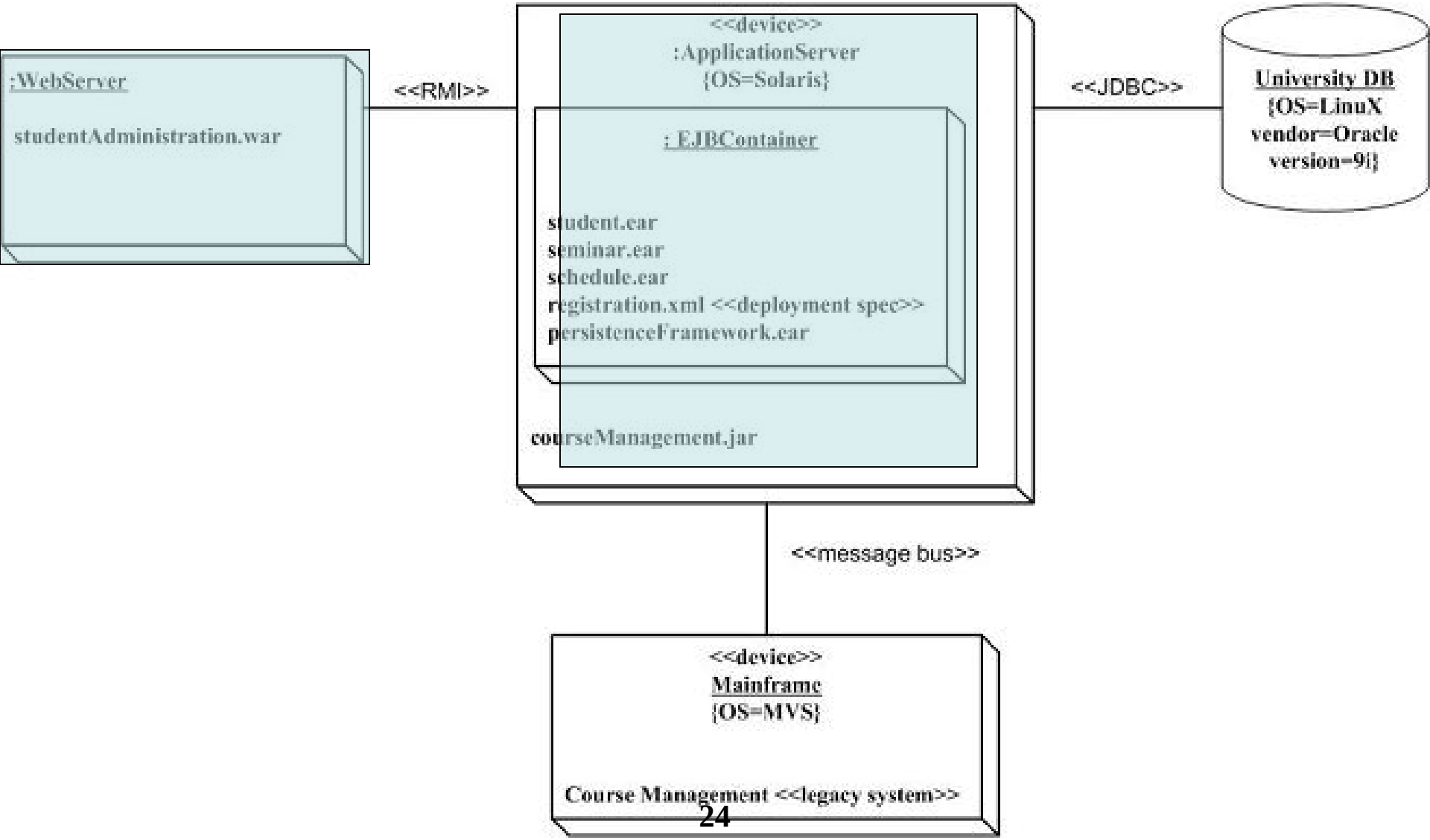
Deployment Diagram



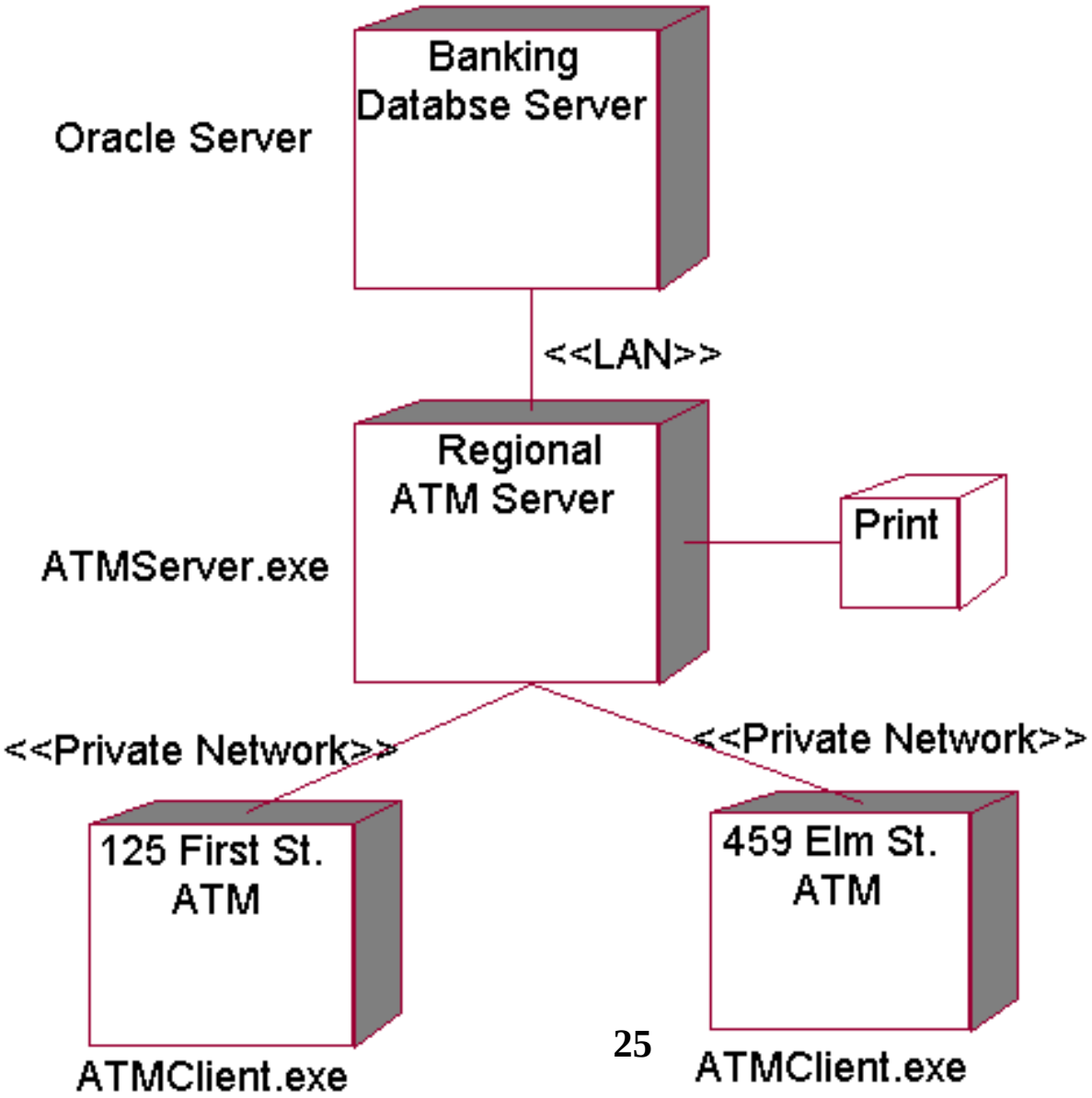
Deployment Diagram

- Deployment diagrams are one of the two kinds of diagrams used in modeling the physical aspects of an object-oriented system
- A deployment diagram shows the configuration of run time processing nodes and the artifacts that live on them.

Example of Deployment Diagram



Example of Deployment Diagram

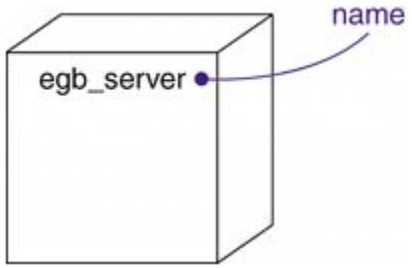


Deployment Diagram

- With the UML, you use deployment diagrams to visualize the static aspect of these physical nodes and their relationships and to specify their details for construction
- Deployment diagrams commonly contain
  - Nodes
  - Dependency and association relationships

Node

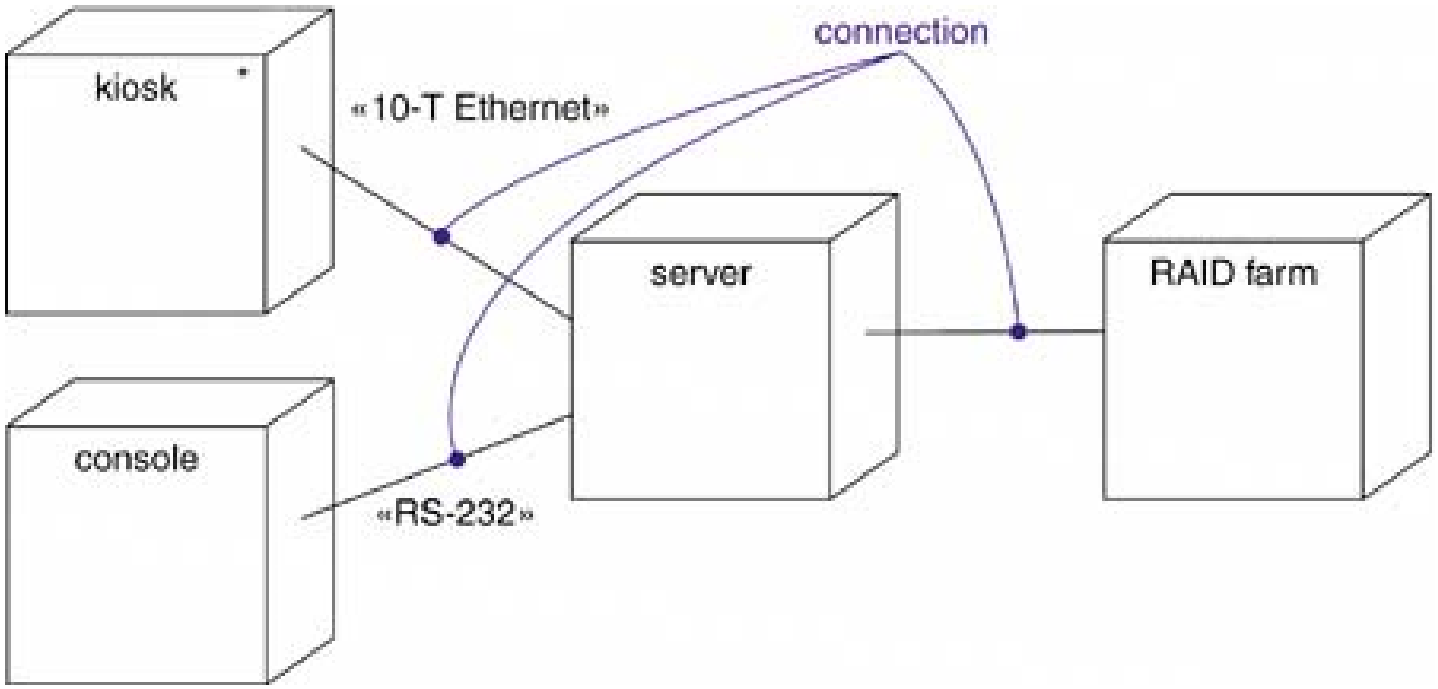
- A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability
- Graphically, a node is rendered as a cube



Connections

- The most common kind of relationship you'll use among nodes is an association
  - In this context, an association represents a physical connection among nodes, such as an Ethernet connection, a serial line, or a shared bus
- You can even use associations to model indirect connections, such as a satellite link between distant processors

Connections

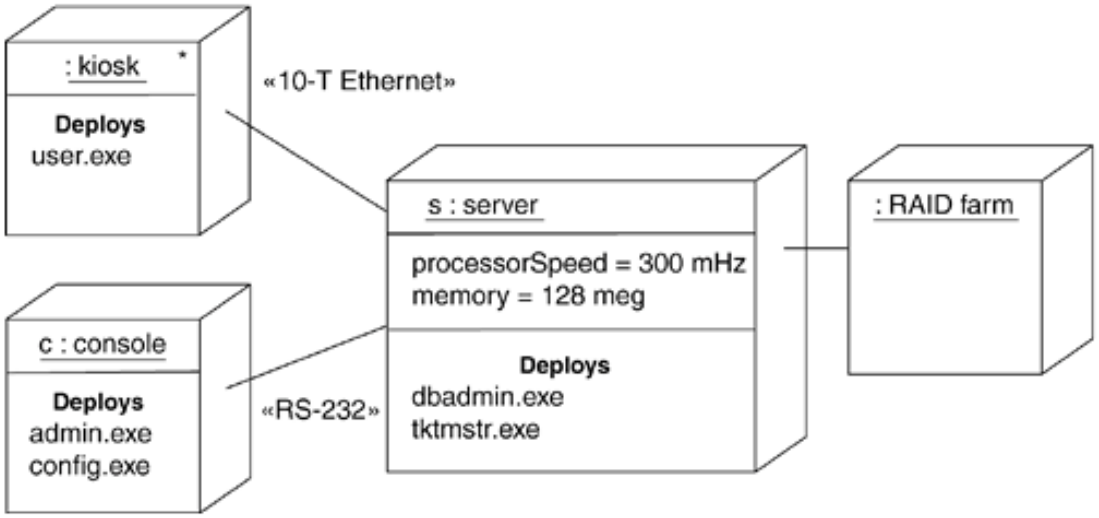


Node

- Modeling Processors and Devices
  - Processors
    - A processor is a node that has processing capability, meaning that it can execute a artifact
  - Devices
    - A device is a node that has no processing capability (at least, none that are modeled at this level of abstraction) and, in general, represents something that interfaces to the real world

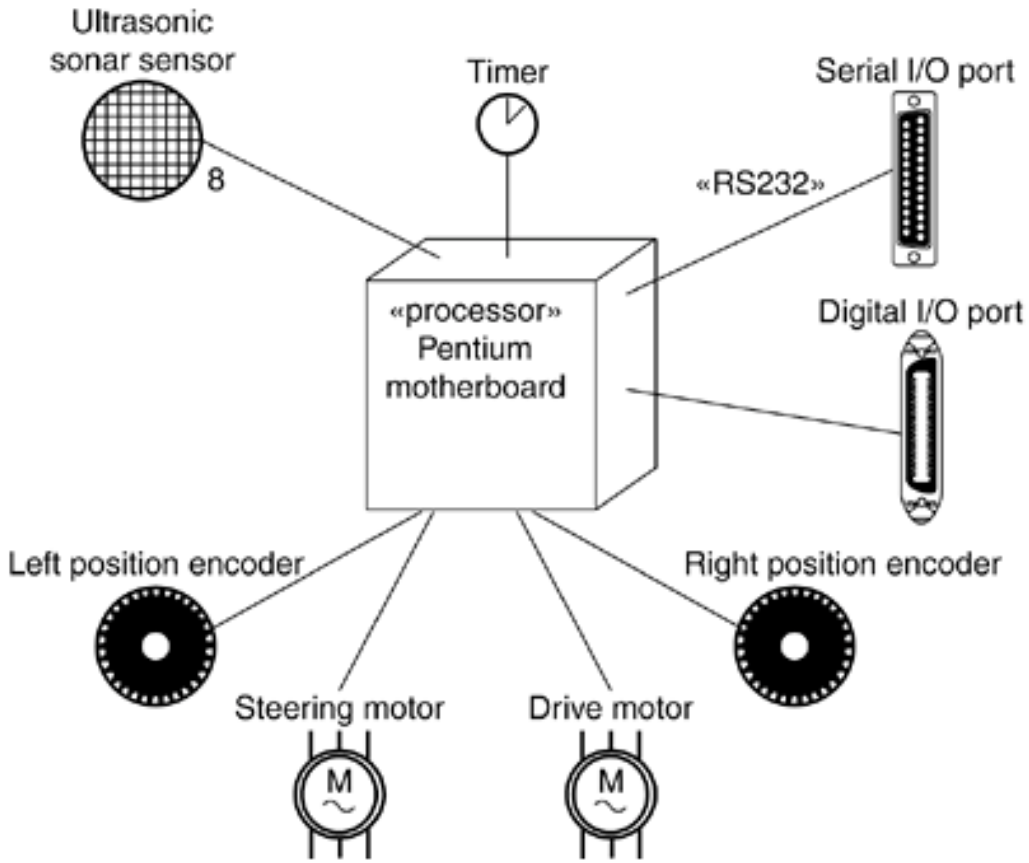
Node

- Modeling the Distribution of Artifacts
  - When you model the topology of a system, it's often useful to visualize or specify the physical distribution of its artifacts across the processors and devices that make up the system.



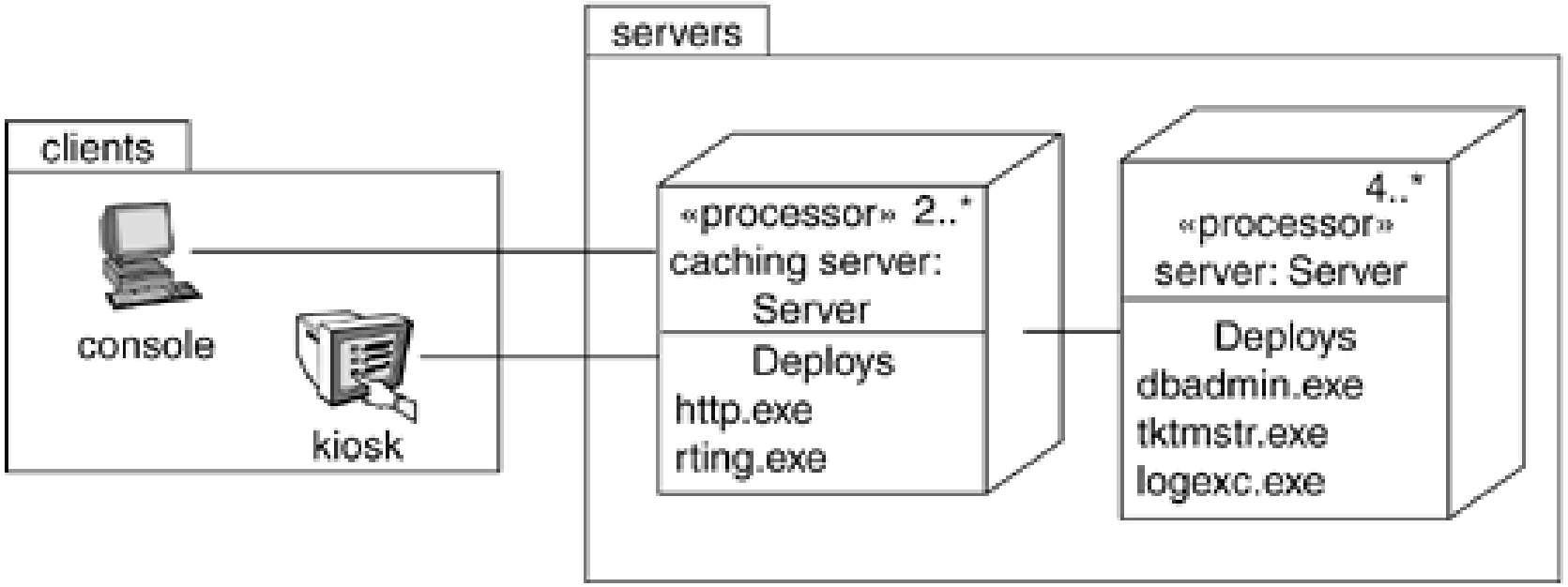
Usage of Deployment Diagram

- Modeling an Embedded System



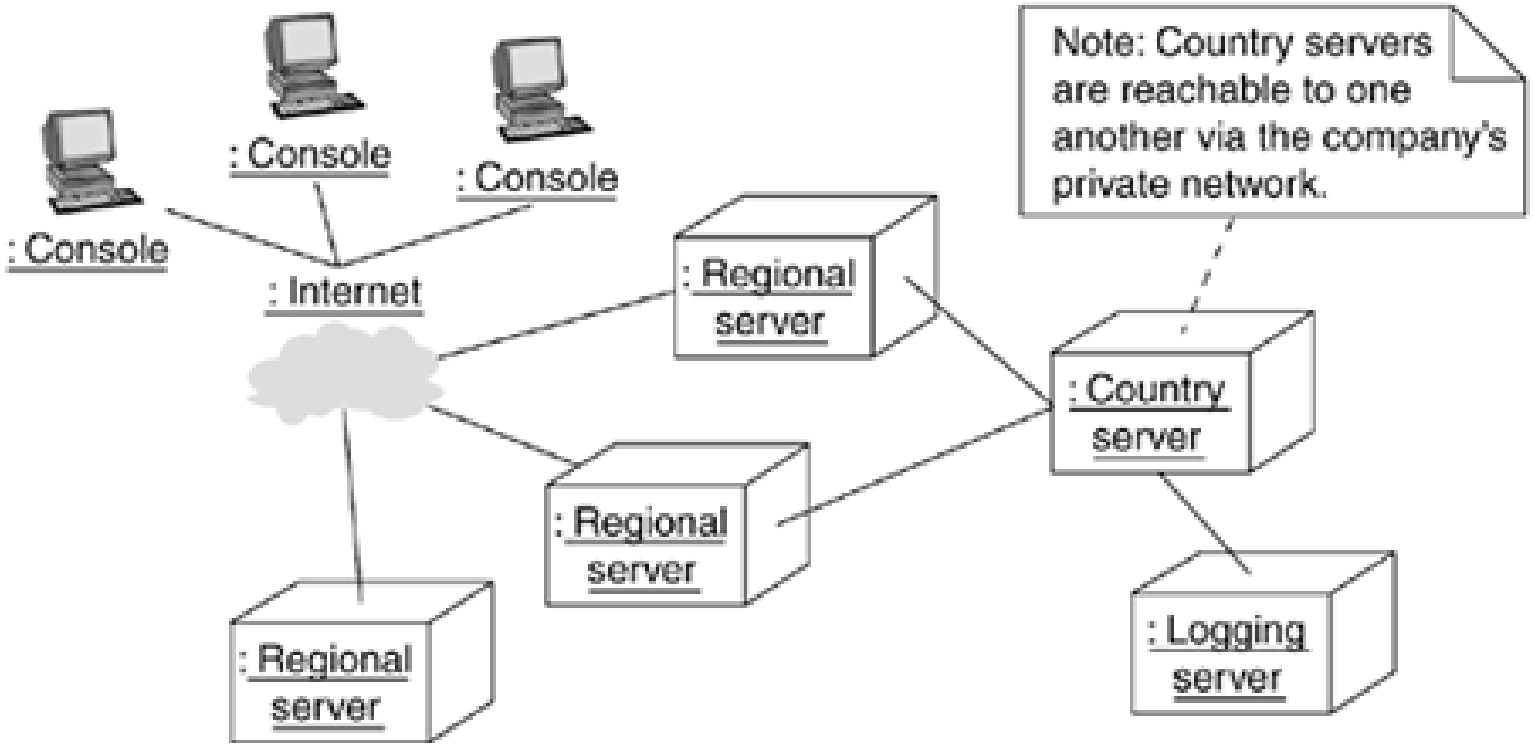
Usage of Deployment Diagram

- Modeling a Client/Server System



Usage of Deployment Diagram

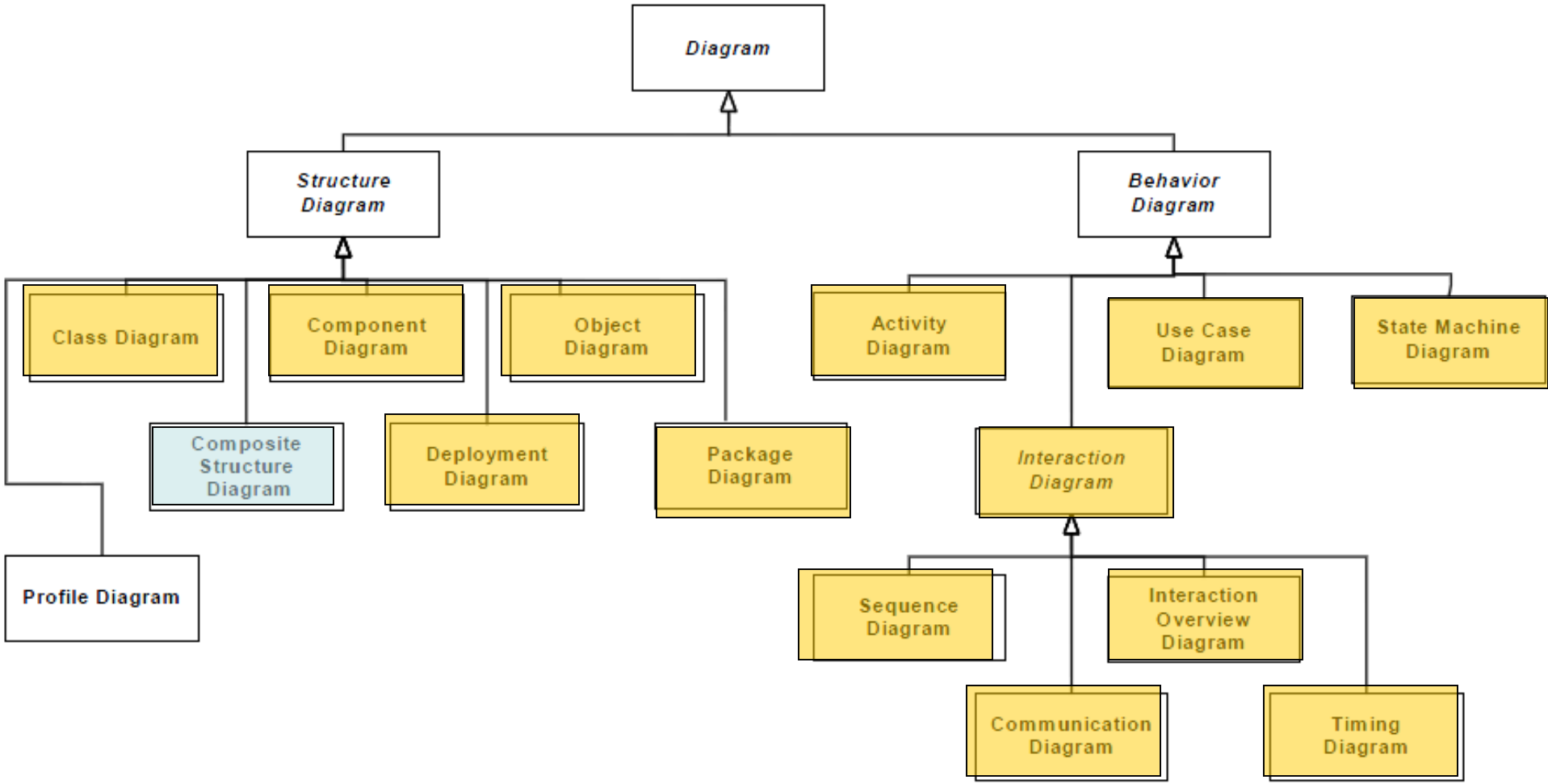
- Modeling a Fully Distributed System



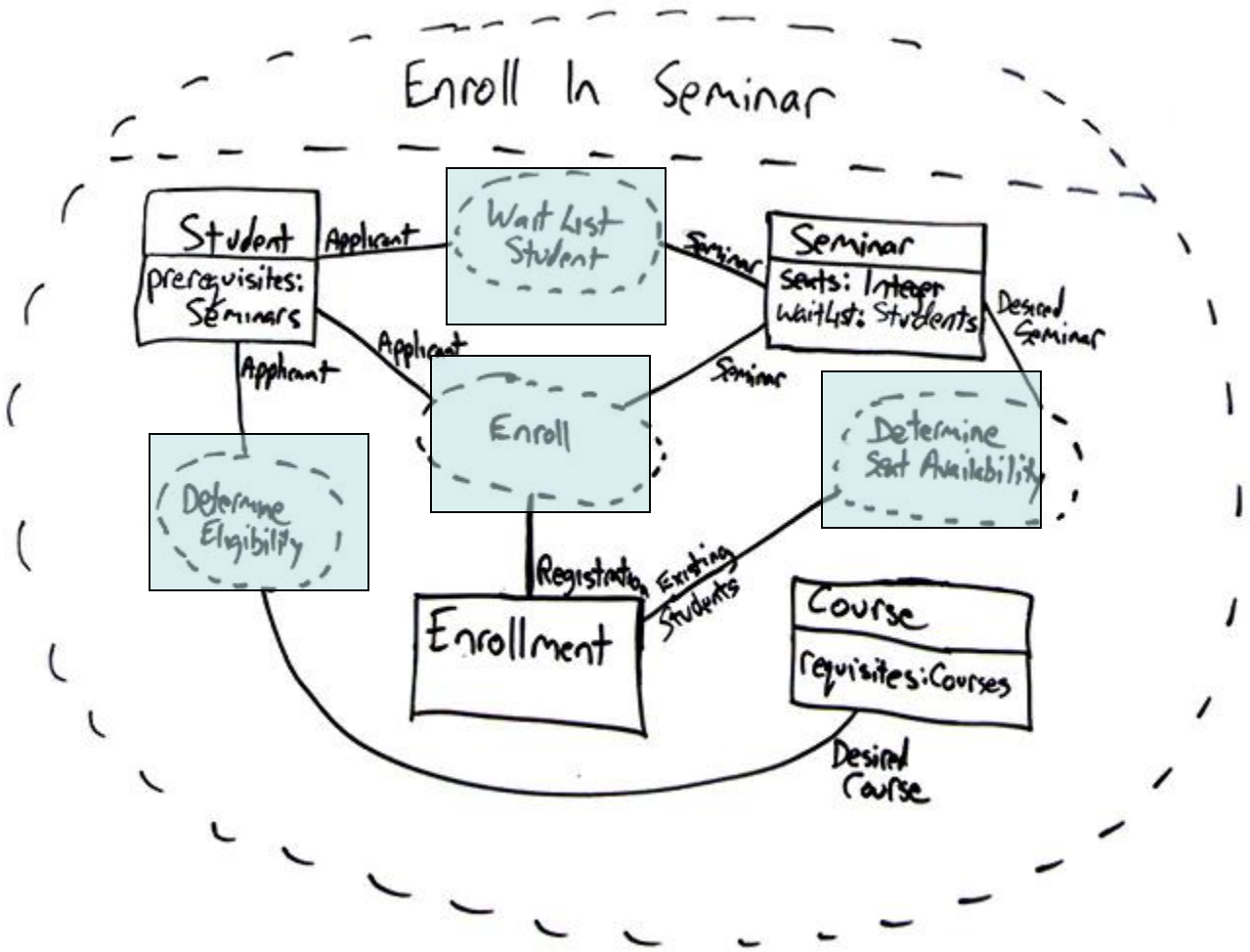
Style of UML Deployment Diagram

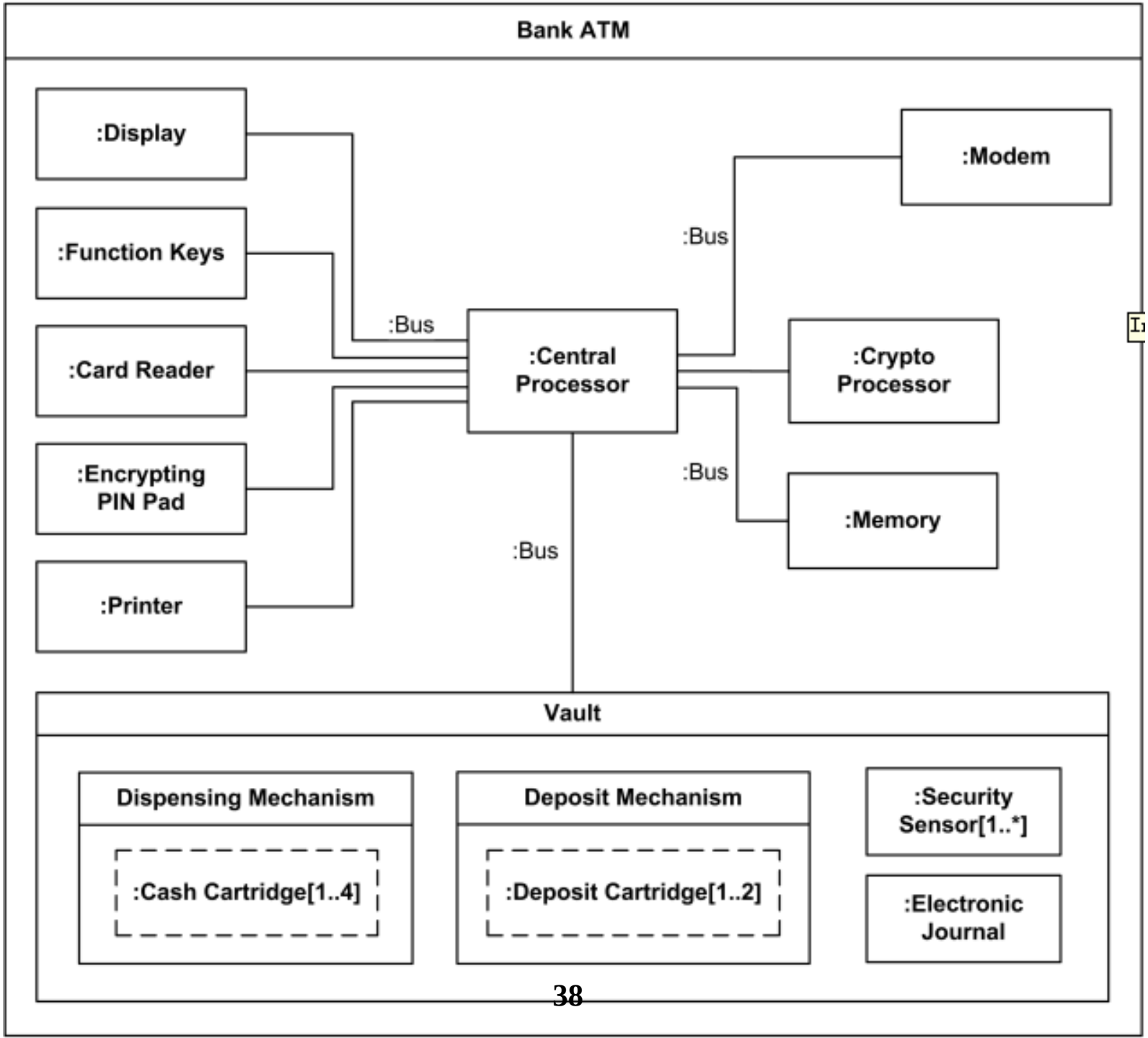
- Use Descriptive Terms to Name Nodes
  - Client, Application Server, Database Server, Mainframe
- Model Only Vital Software Component
- Apply Visual Stereotypes to Nodes
- Indicate Communication Protocols via Stereotypes

Composite Structure Diagram

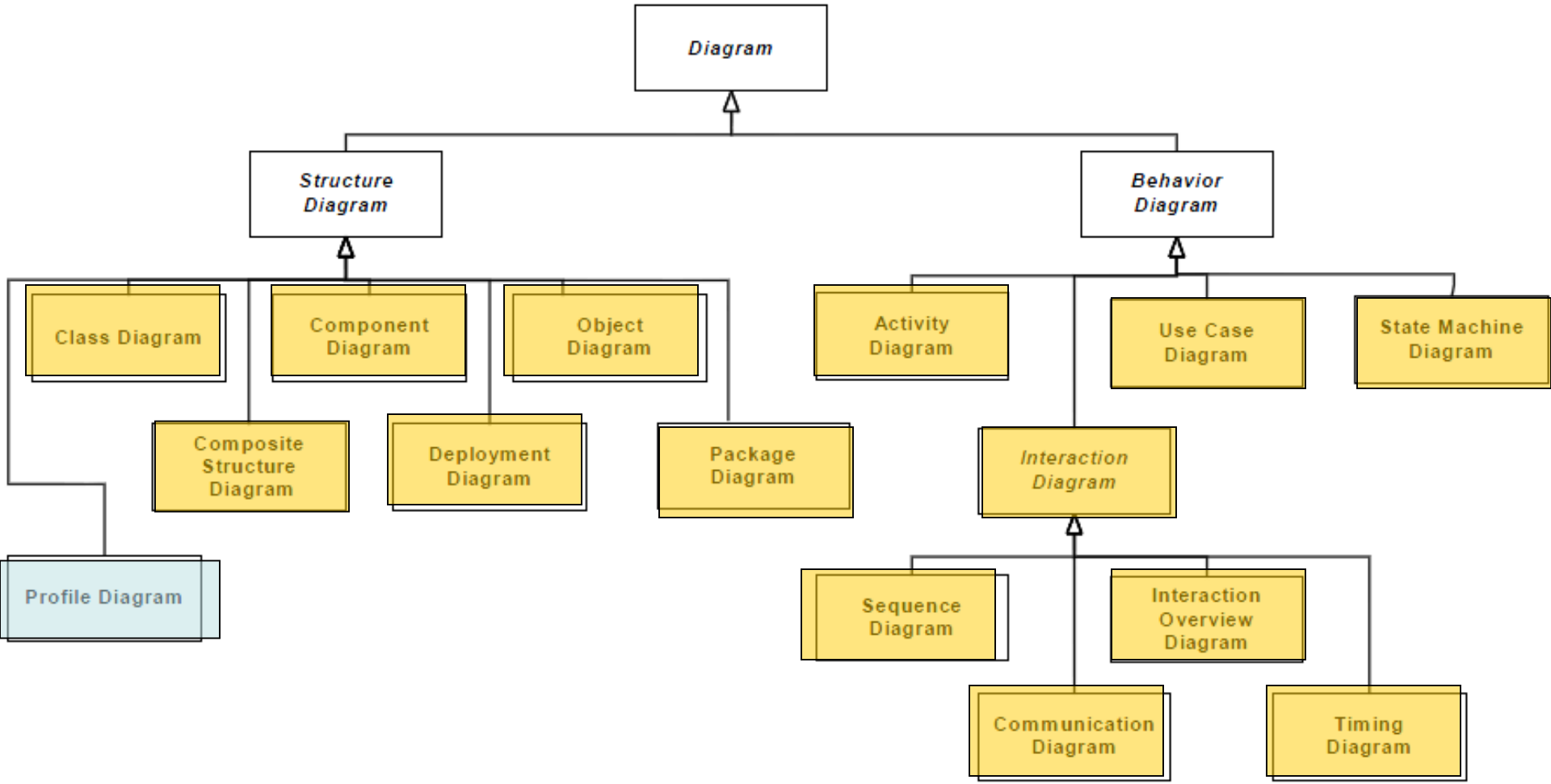


Example of Composite Structure Diagram




Bank ATM
:Display
:Function Keys
:Card Reader
:Encrypting PIN Pad
:Printer
:Bus
:Central Processor
:Bus
:Modem
:Bus
:Crypto Processor
:Bus
:Memory
:Bus
Vault
Dispensing Mechanism
:Cash Cartridge[1..4]
Deposit Mechanism
:Deposit Cartridge[1..2]
:Security Sensor[1..*]
:Electronic Journal

Profile Diagram



Forward and Reverse Engineering

- Forward Engineering
  - Given the UML model, generate the corresponding code
- Reverse Engineering
  - Given the code for some classes, generate the corresponding model