UNIVERSITÀ DEGLI STUDI DI BERGAMO
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

UML as Architecture Description Language

PROGETTAZIONE, ALGORITMI E COMPUTABILITÀ (38090-MOD1)

Corso di laurea Magistrale in Ingegneria Informatica

RELATORE
Prof.ssa Patrizia Scandurra

SEDE
DIGIP

DATA
04-10-2021

REFERENCES

- UML 2.0 Superstructure Specification

  www.omg.org

- The Diagrams of UML 2.0

  by Scott W. Ambler, 2003-2004

  www.agilemodeling.com/essays/umlDiagrams.htm

- UML basics: The component diagram
  - By Donald Bell (bellds@us.ibm.com), IT Architect, IBM Corporation
    http://www.ibm.com/developerworks/rational/library/dec04/bell/

Outline

- Some UML diagrams for SA design
  - UML component diagram
  - UML package diagram
  - UML deployment diagrams
- Architecture views in UML
  - The 4+1 view model in UML
  - Example (a typical three-tier application)

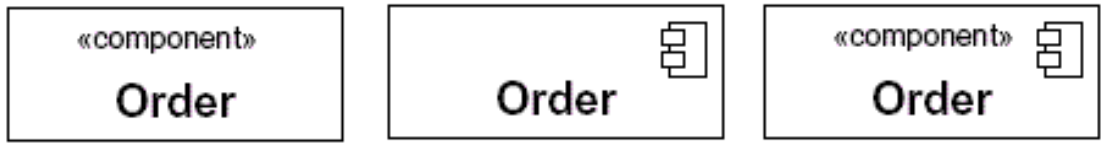
Describing an architecture using UML

- All UML diagrams can be useful to describe several aspects (*views*) of the architectural model

- UML diagrams particularly suitable for architecture modelling in AMDD:
  - **Component diagrams**
  - **Deployment diagrams**
  - **Package and class diagrams**

UML component diagram

- Component notation (alternative ways)



- **Components can be labelled with a stereotype**
  - There are a number of **standard UML stereotypes**, e.g. <<subsystem>>
  - Stereotypes for denoting a particular **"component model"**: EJB components, .NET components, etc.

UML component diagram elements

- A component can have
  - **Interfaces (or provided interfaces)**

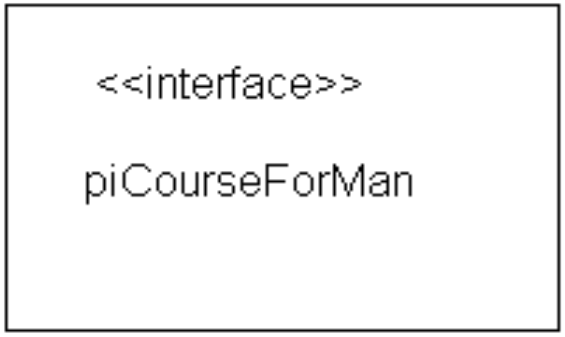
    An interface represents a declaration of a set of operations and obligations
  - **Usage dependencies (required interfaces)**

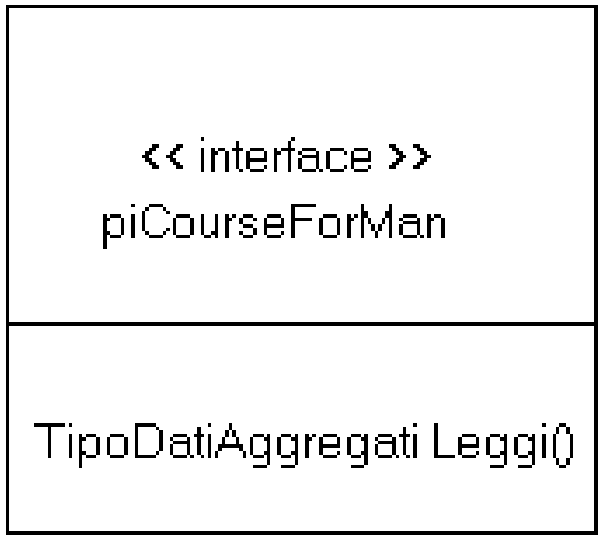
    A usage dependency is relationship which one element requires another element for its full implementation
  - **Connectors**
    - Connect two components (*assembly connector*)
    - Connect the external contract of a component to the internal structure (*delegation connector*)
  - **Ports** (from UML 2)

    Port represents an interaction point between a component and its environment

INTERFACE definition

- A component defines its behaviour in terms of provided and required interfaces
- An interface
  - Is the definition of a collection of one or more operations
  - Provides only the operations but not the implementation
  - Implementation is normally provided by a class/component
  - In complex systems, the physical implementation is provided by a group of classes rather than a single class
- Interface definition

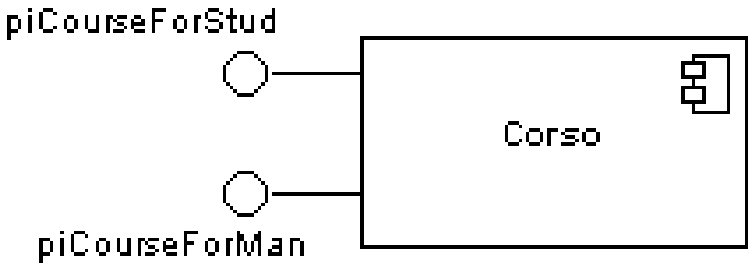




INTERFACE use

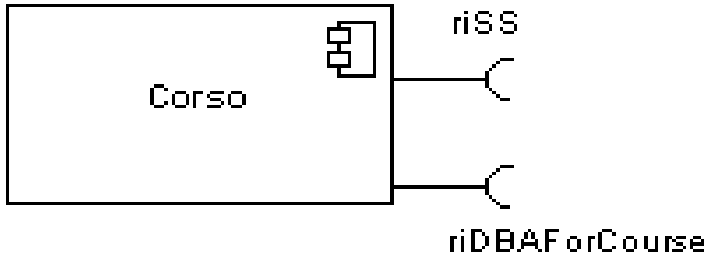
A **provided interface**

- Characterize services that the component offers to its environment
- Is modeled using a ball, labelled with the name, attached by a solid line to the component



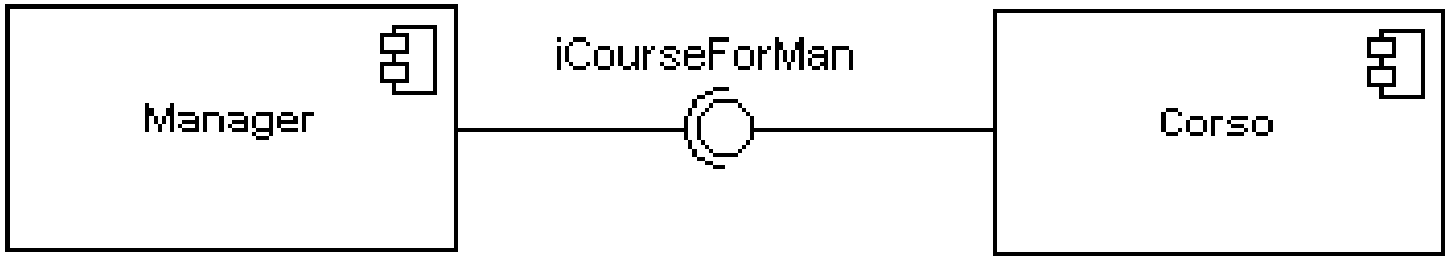
A **required interface**

- Characterize services that the component expects from its environment
- Is modeled using a socket, labelled with the name, attached by a solid line to the component

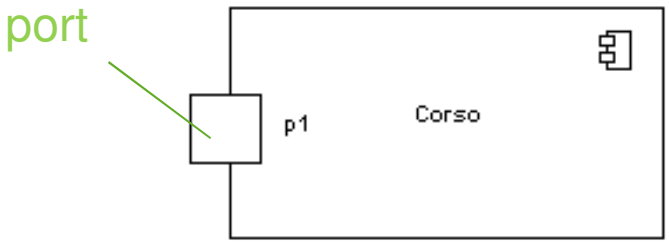


INTERFACE (the ball-and-socket notation)

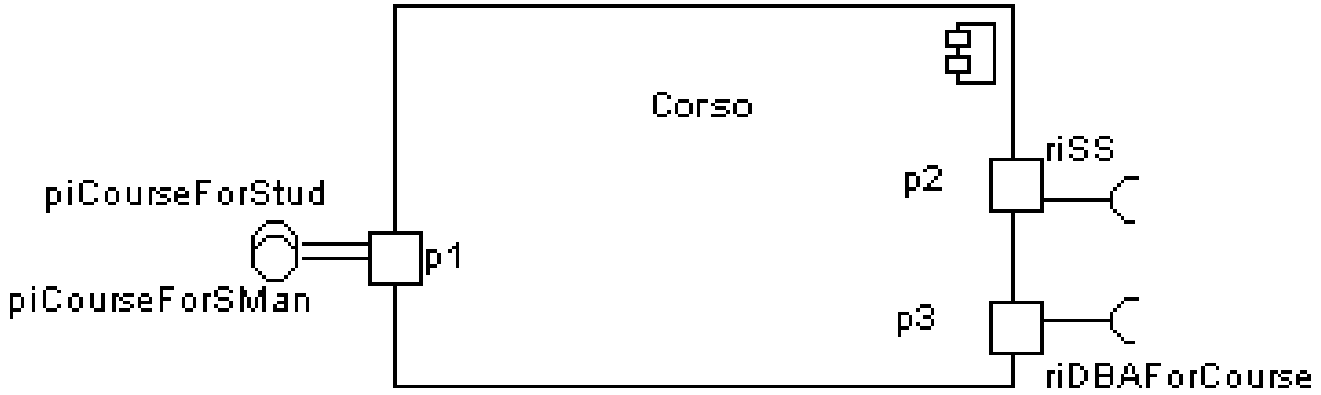
- Where two components respectively provides and requires the same interface, their two notations may be combined (***ball-and-socket***)
- The interfaces in question serves to mediate interactions between the two components (***assembly connector***)



PORT (from UML 2)



- A small square symbol, can be named
- Specifies a distinct **interaction point**
  - Between that component and its environment
  - Between that component and its internal parts
- It is associated with the interfaces that specify the nature of the interactions that may occur over a port

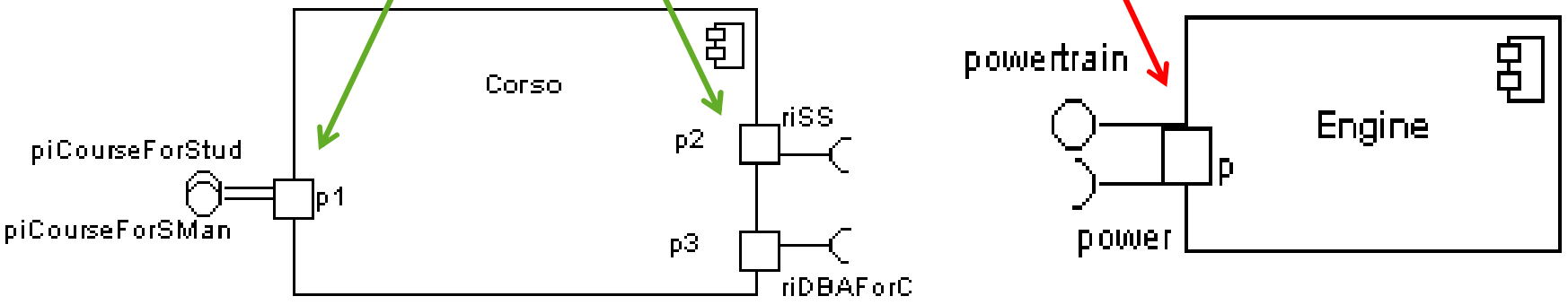


Port and communication

Ports can support **unidirectional communication**

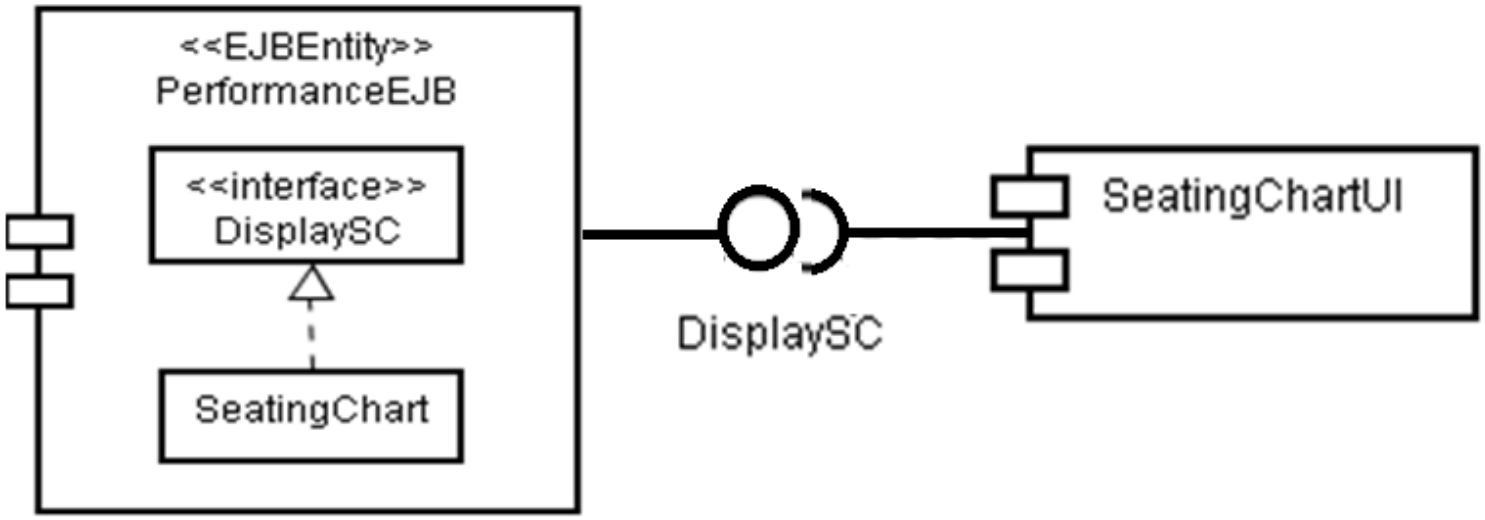
or **bi-directional communication**

(*for call-back asynchr. interactions*)



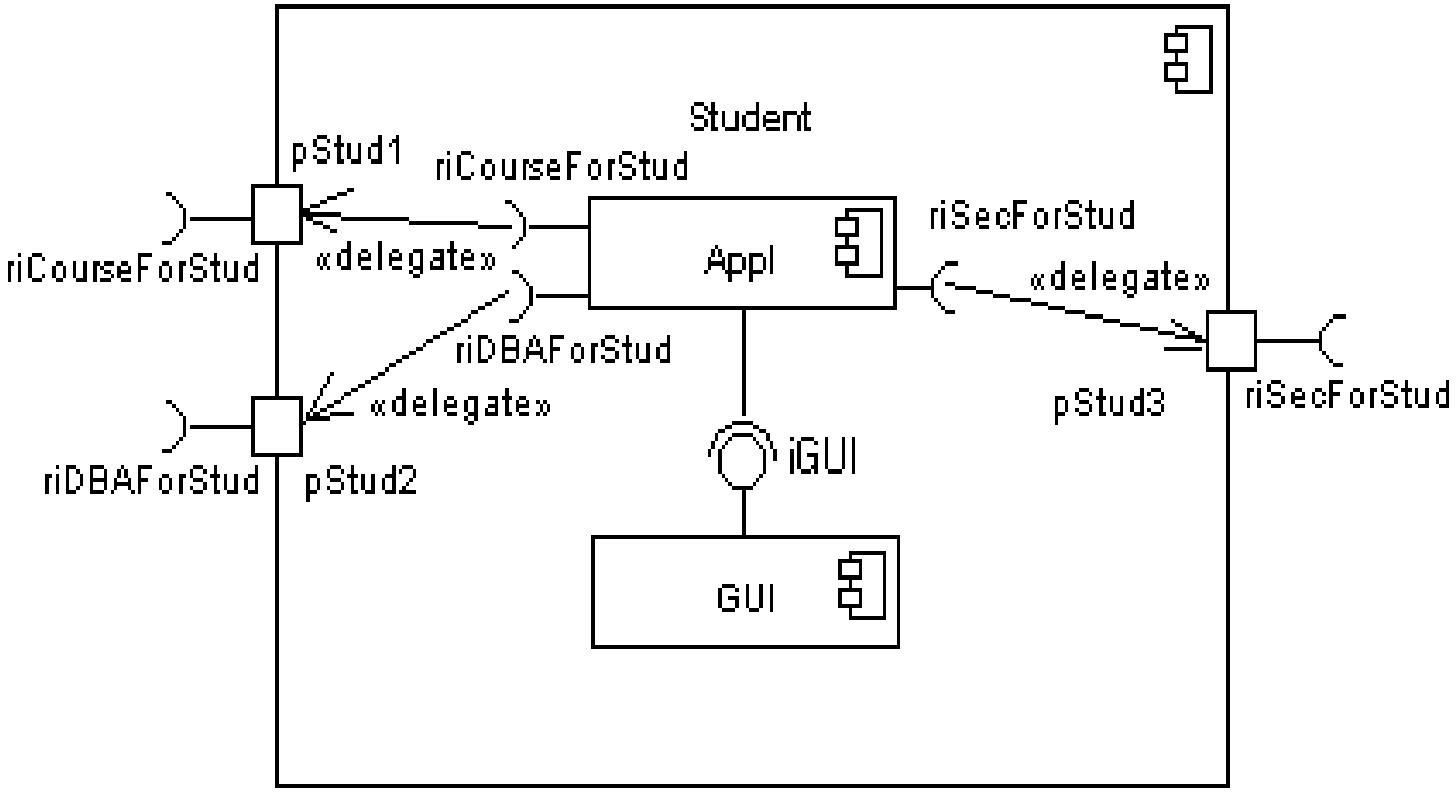
Components and realizing classes

- An internal (white box) view of a component is where the realizing classes are nested within the component
- The **interfaces of the classes realizing the component define the interface of the component**
  - *Semantics*: when an interface is accessed on the component, the component actually forwards the invocation to the corresponding interface on one of the resident classes



Composite components

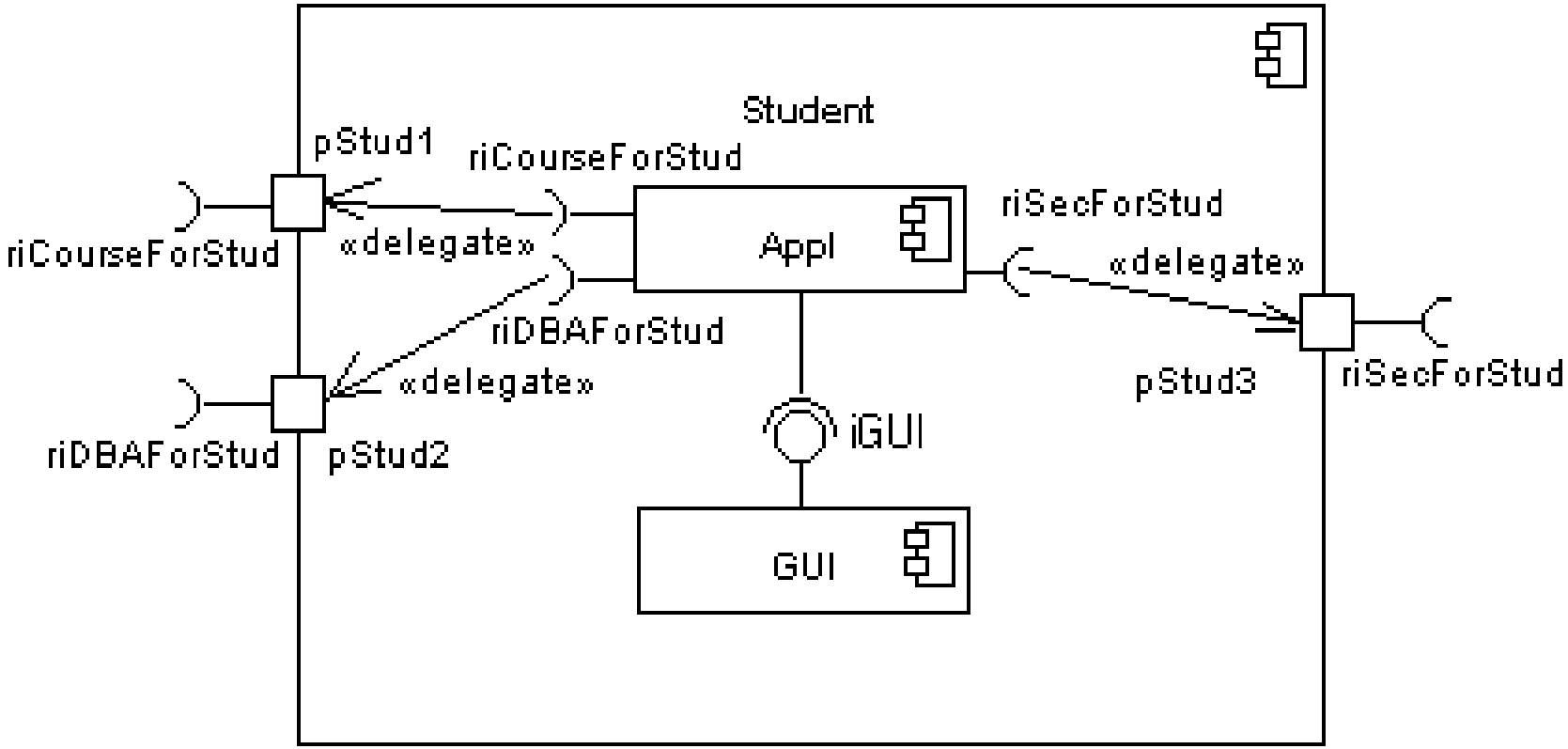
- Components can be composite (*hierarchical design*)
- **Realizing components** are nested within the component
- Delegation connectors forwards the invocation to the correspondings interface on one of the resident components and viceversa



Connectors

Components can be composed (assembled) together by:

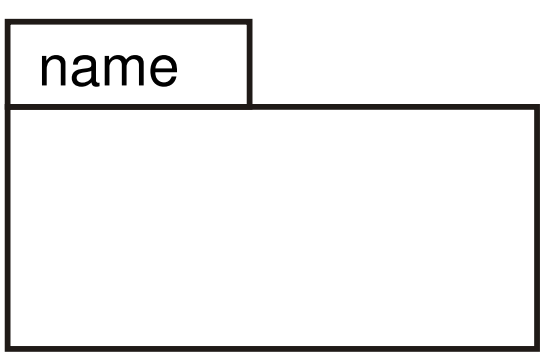
- **assembly connectors** *(ball-and-socket connections)*
- **delegation connectors**, representing the forwarding of signals between composite components and its sub-components (interfaces of the same kind: *required-to-required*, *provided-to-provide*d)

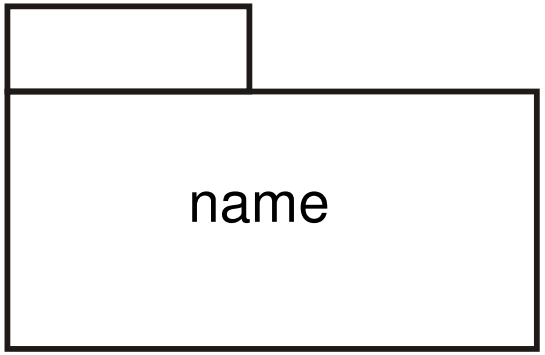


UML Package diagrams

A **package** is a collection of logically related modeling elements

- to organize your elements by grouping them and placing them into a container
- *namespace* functionality as a folder in Windows or a package in Java
  - The name of an element in a package must be unique
  - You may create an element of the same name in a different package



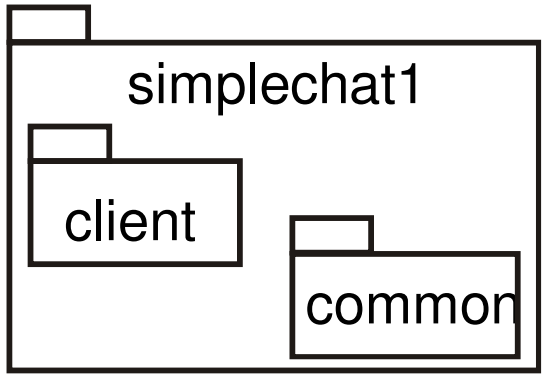


What a package contains

UML modeling elements: other packages (nesting), classes definitions, use cases definitions, UML diagrams of any kind, etc.

In a package you can put three types of elements:

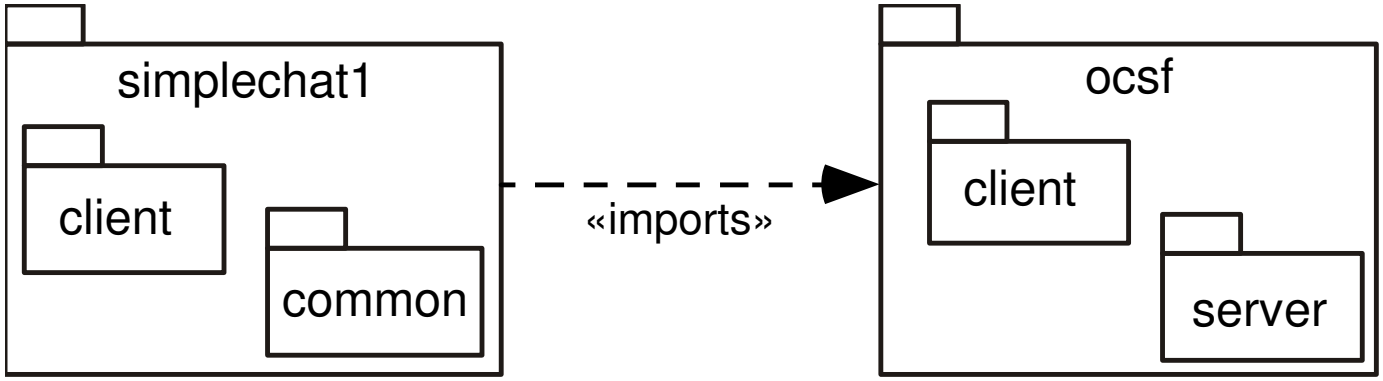
- elements **owned** by the package itself
- elements **imported** from other packages
- elements **accessed** from another package



Package diagrams: package dependency

<<import>> = you bring a copy from another package into the package you are working in

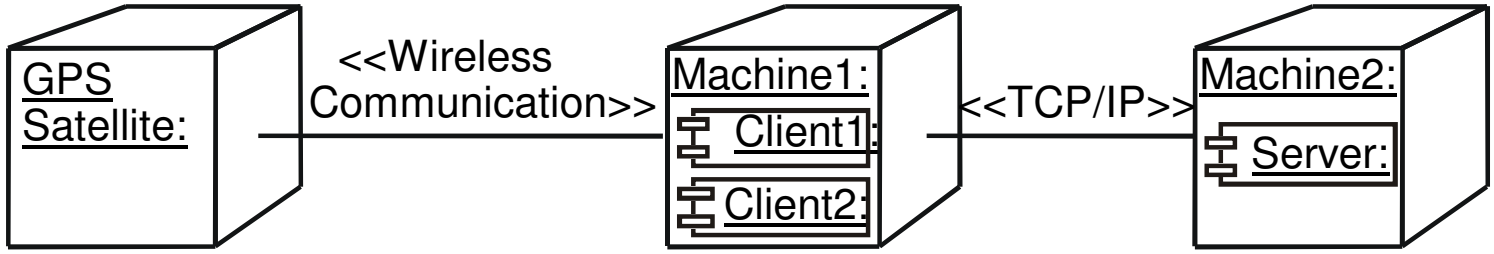
<<access>> = you access in another package; the accessed elements stays in its package



Apply **principles of cohesion/coupling** among packages, in order to increase cohesion and decrease dependencies as much as possible

UML deployment diagrams

- Show the hardware where various *instances of components* reside at run-time: *the execution infrastructure*
- It consists of *nodes* and *associations* between nodes
- A node reppresents a **computational unit** such as a computer, a sensor or a device
- Associations between nodes show how communication takes place



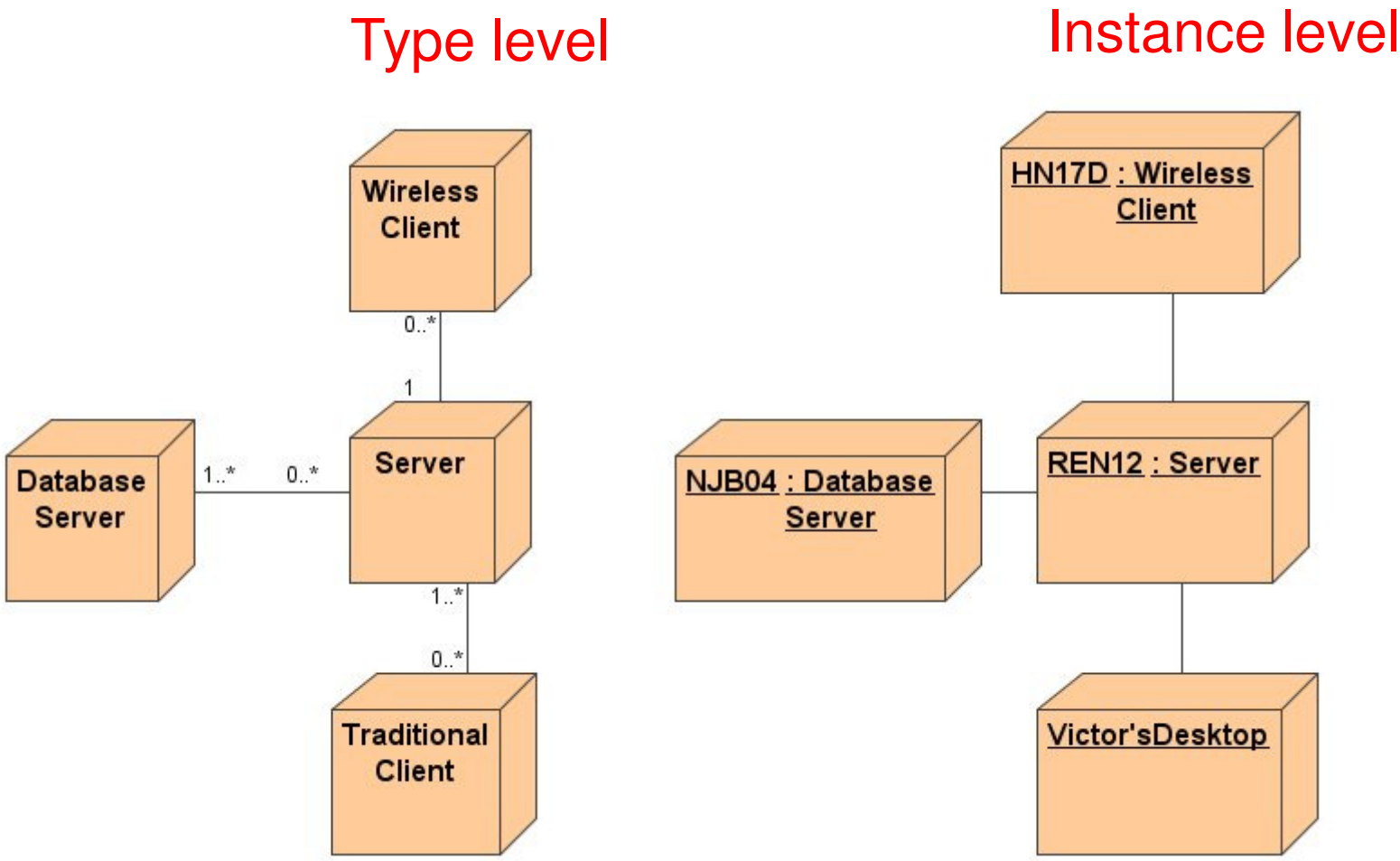
UML deployment diagrams

You can distinguish **two levels of representation**:

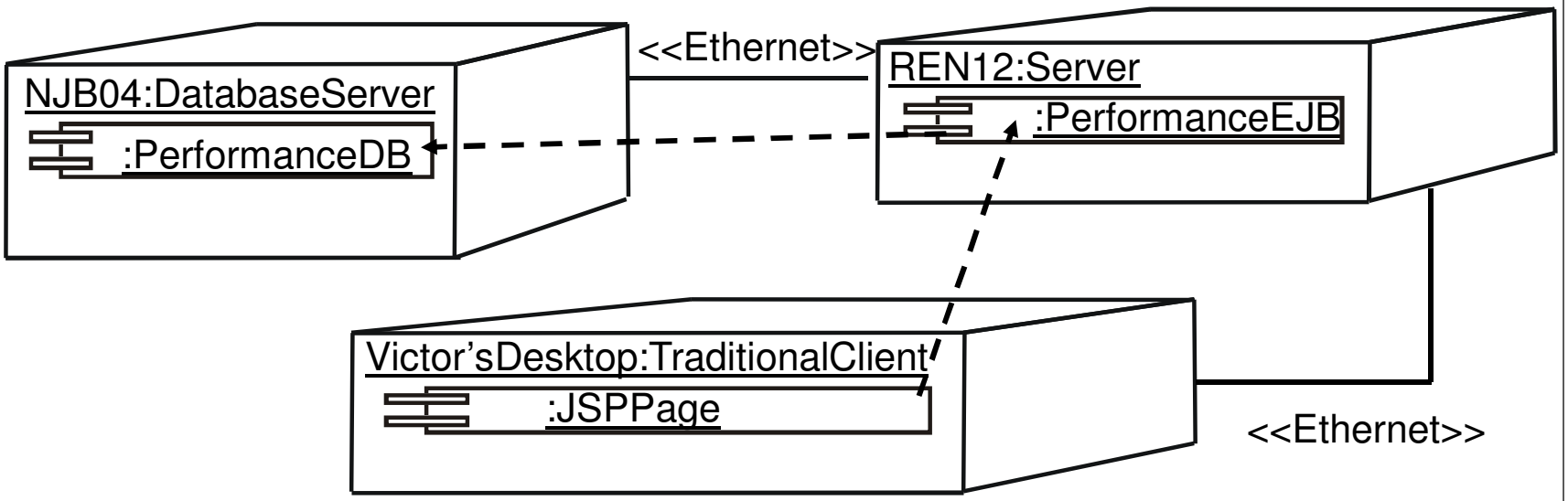
1. **Type level**: A deployment diagram **can define *node types* and *associations types* for node communication** (much like a class diagram that define object types)

2. **Instance level**. A deployment diagram can be instantiated (like object diagrams) with **node instances** and **links** (or node association instances)
   - It **may include *component instances* within *node instances*** to illustrate the runtime environment

UML deployment diagrams: node types and node instances



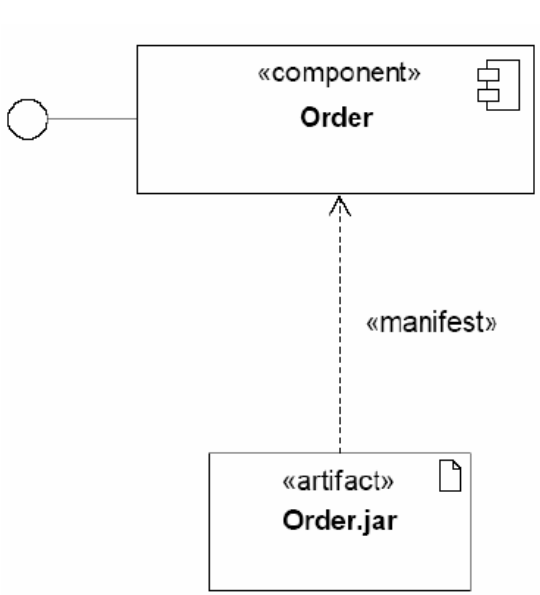
UML deployment diagrams with components: (Instance level)

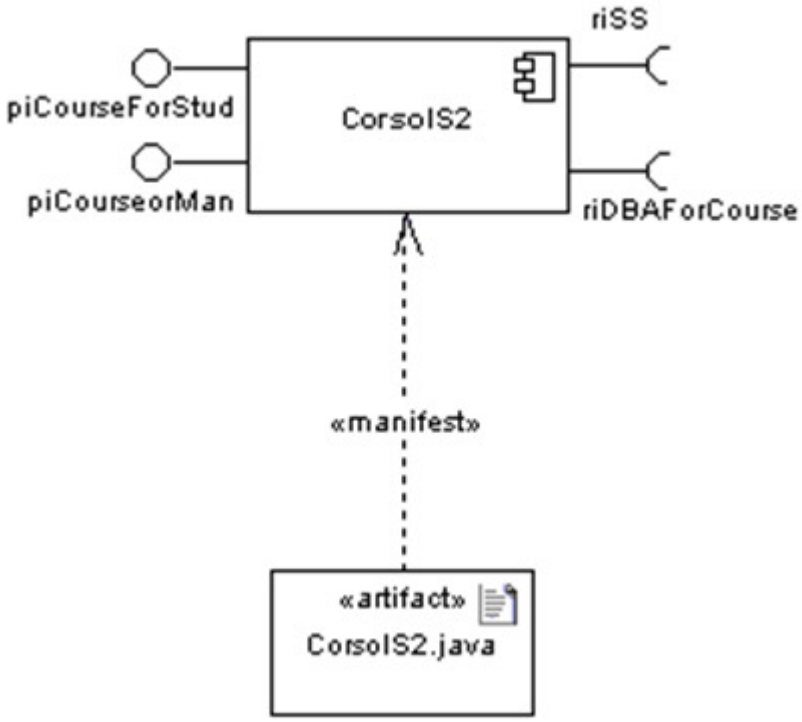


Note that the components drawn in a deployment diagram are always **component instances**

Deployment diagrams and artifacts

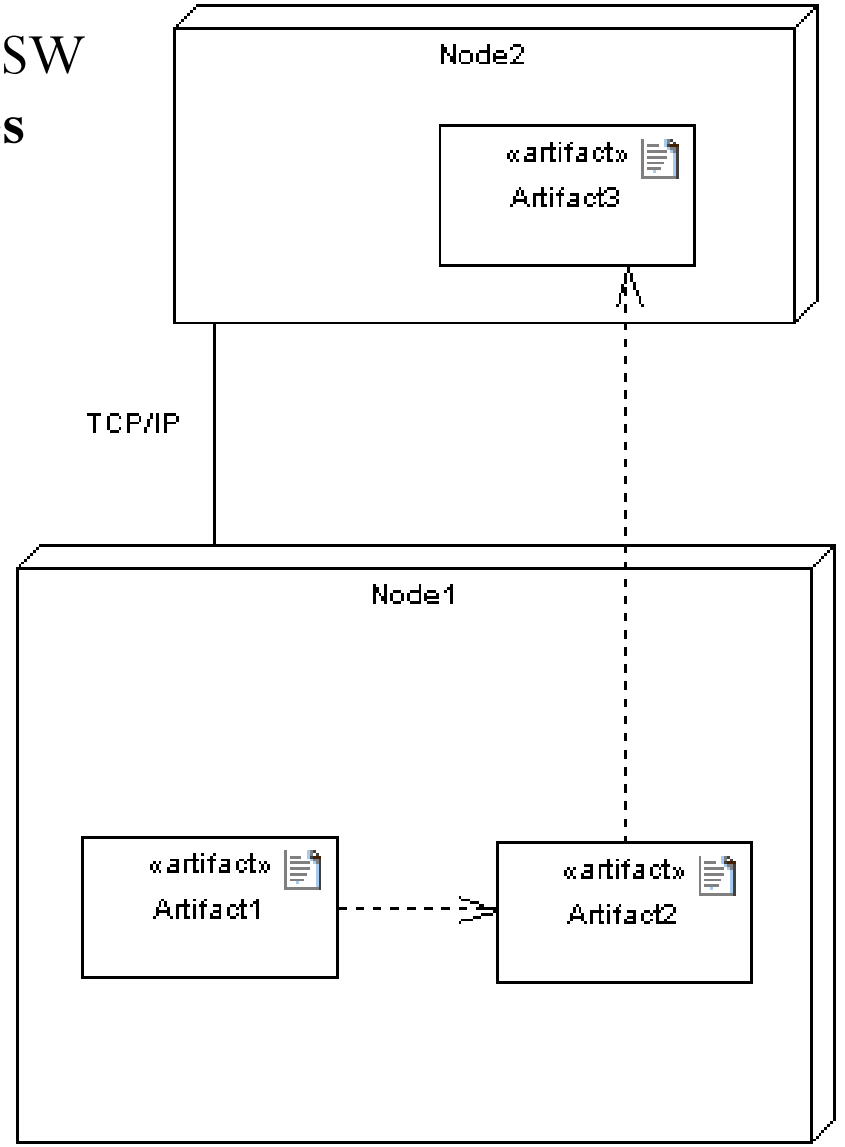
- An artifact manifest (by an association **<<manifest>>**) one or more components
  - To denote the actual implementation when the component is executed (source code, JAR file, DLL, etc.)





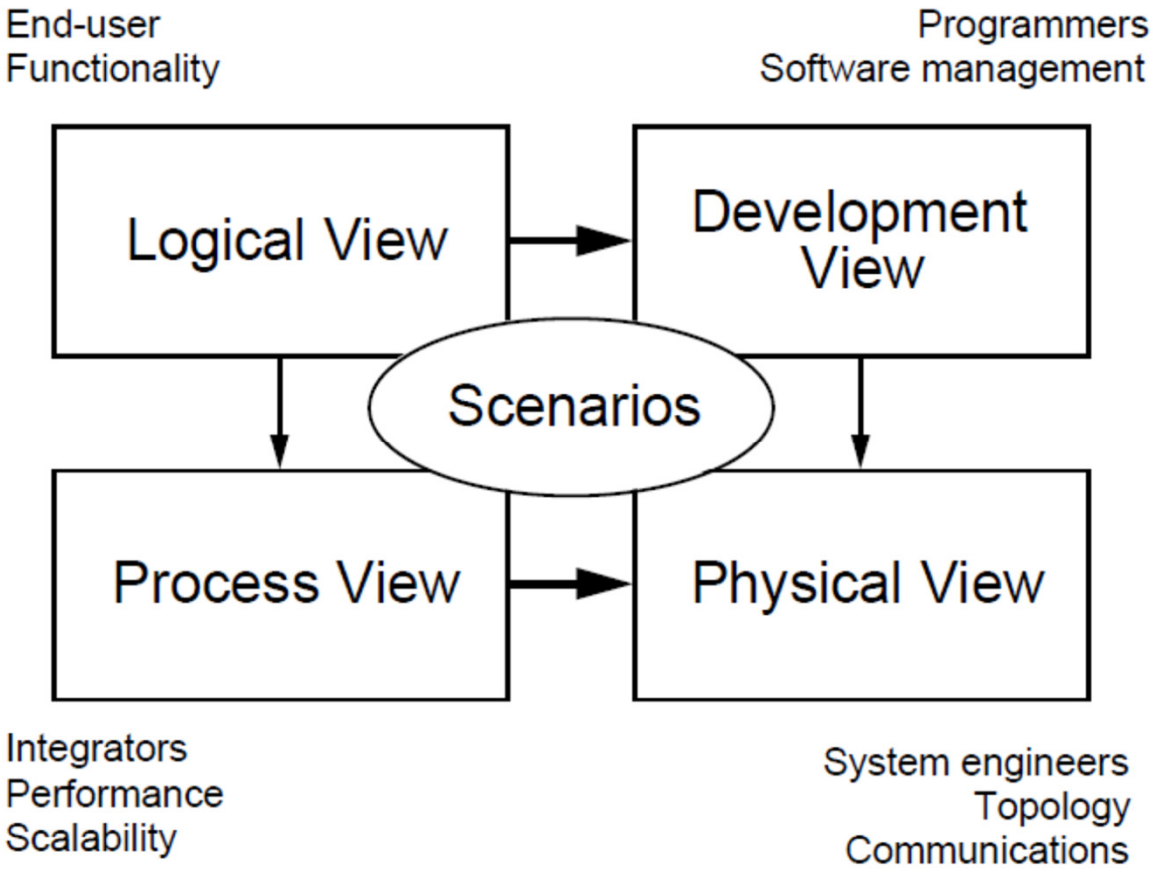
UML deployment diagrams and SW artifacts

- Deployment diagrams may contain SW **artifacts and their dependecies**
- An artifact
  - A concrete element of information in the physical world
  - Ex: source files, binary executable files, XML document, table in a database system,….



The “4+1” View Model of SA

- In order to address large and challenging architectures, the model may be made up of five main views



How to use UML as ADL to describe architecture views

Functional (or Logical) view:

- Component diagrams, activity diagrams

Runtime (or Process) view:

- State machine diagrams, object diagrams, component diagrams, sequence diagrams, activity diagrams

Development (or *Implementation or Module*) *view:*

- Class diagrams, package diagrams, sequence diagrams, state machine diagrams

Deployment (or Physical) view:

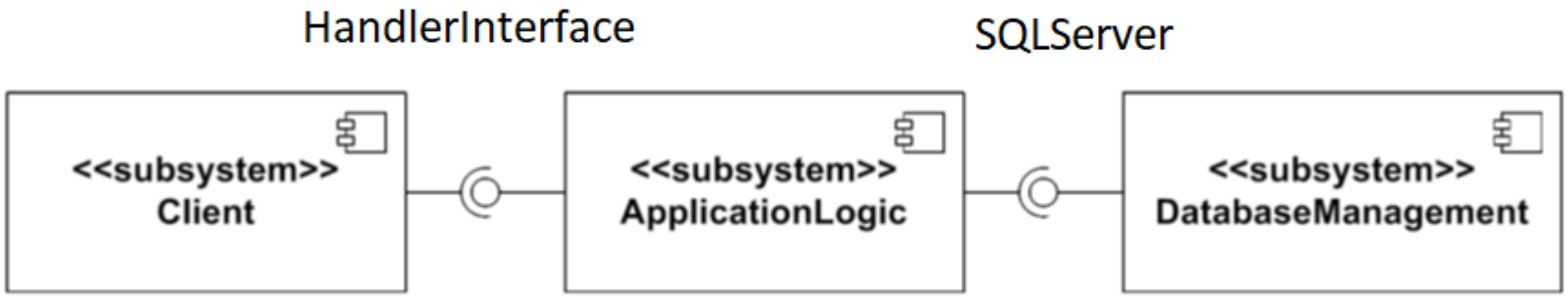
- Deployment diagram

Scenario view:

- Sequence diagrams, activity diagrams

Architectural views: E-commerce case study

- **Functional view** of a typical *three-tier architecture*

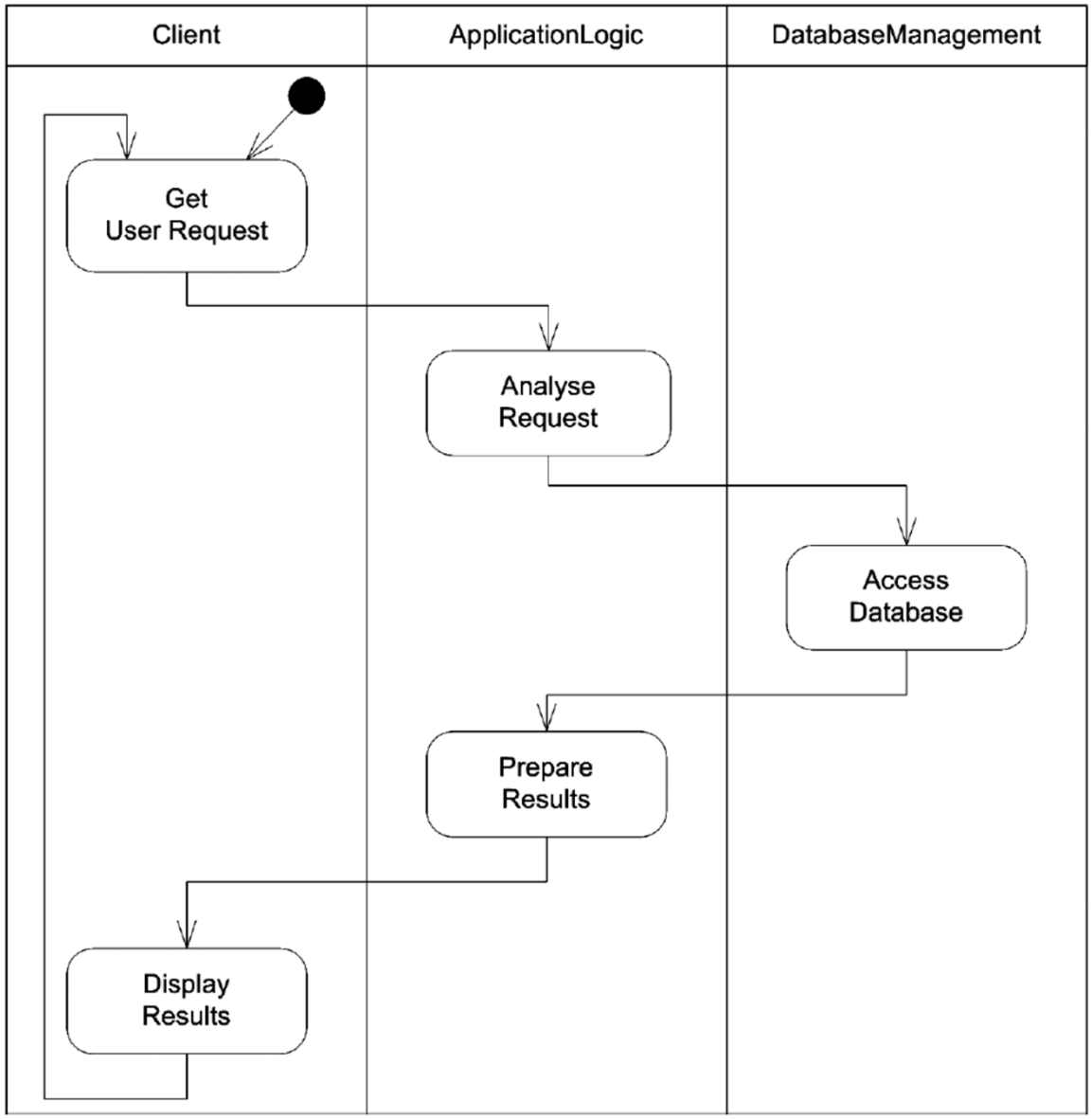


UML component diagram

Architectural views: E-commerce case study

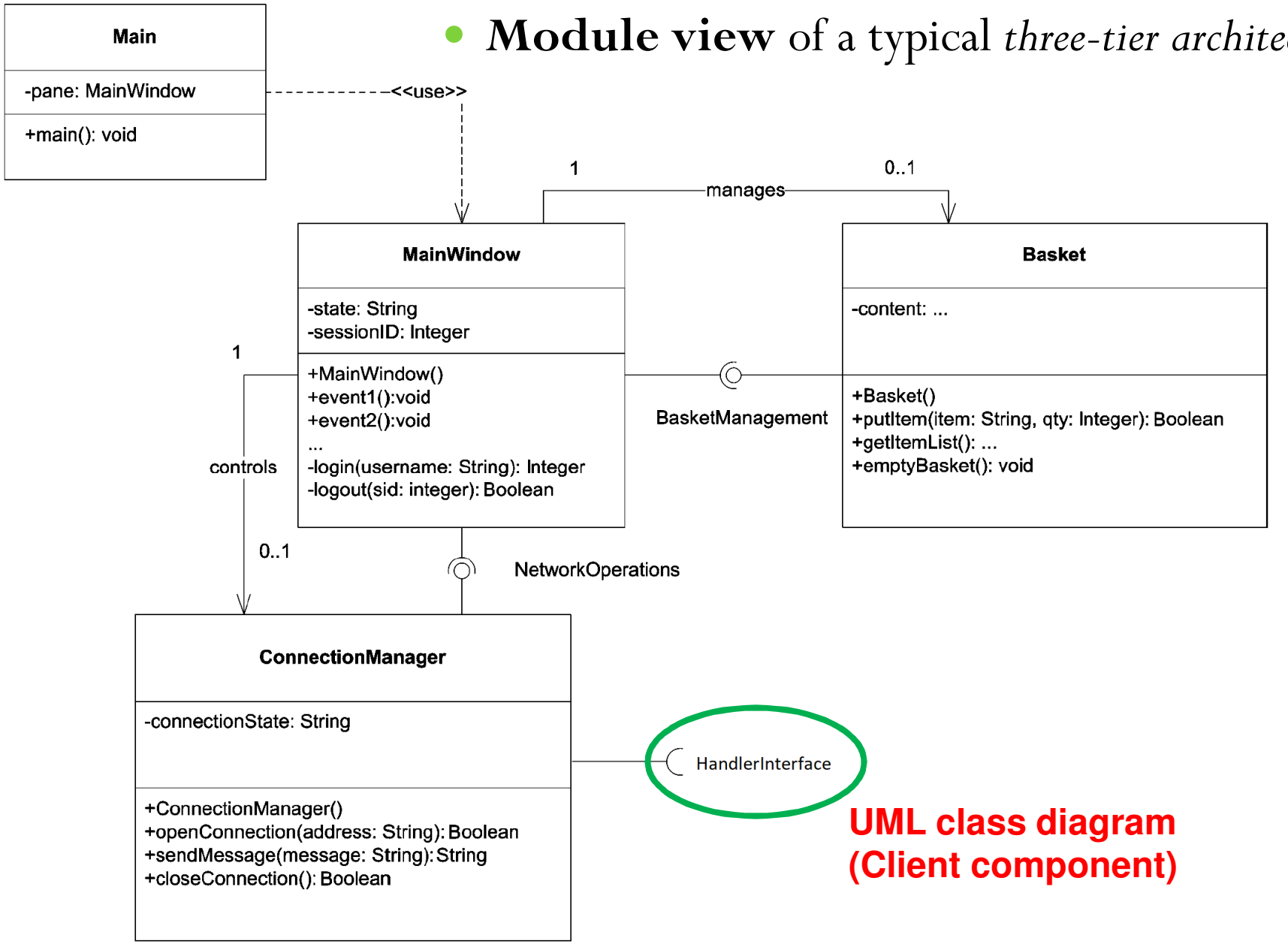
- **Functional view** of a typical *three-tier architecture*

UML activity diagram (shows the flow of execution among components)



Architectural views: E-commerce case study

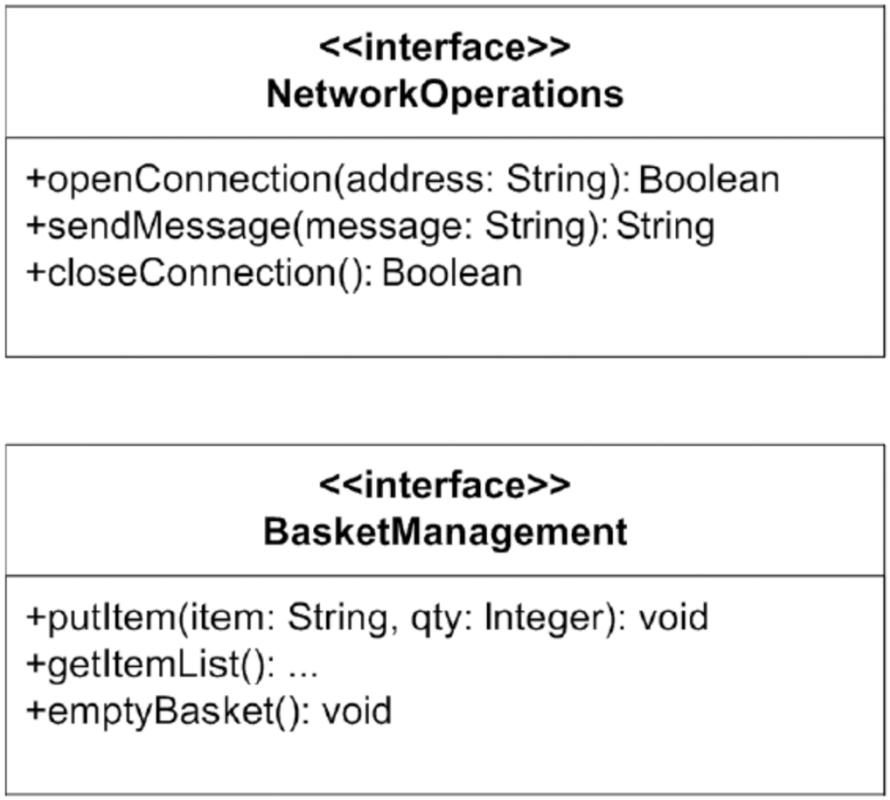
- **Module view** of a typical *three-tier architecture*



UML class diagram (Client component)

Architectural views: E-commerce case study

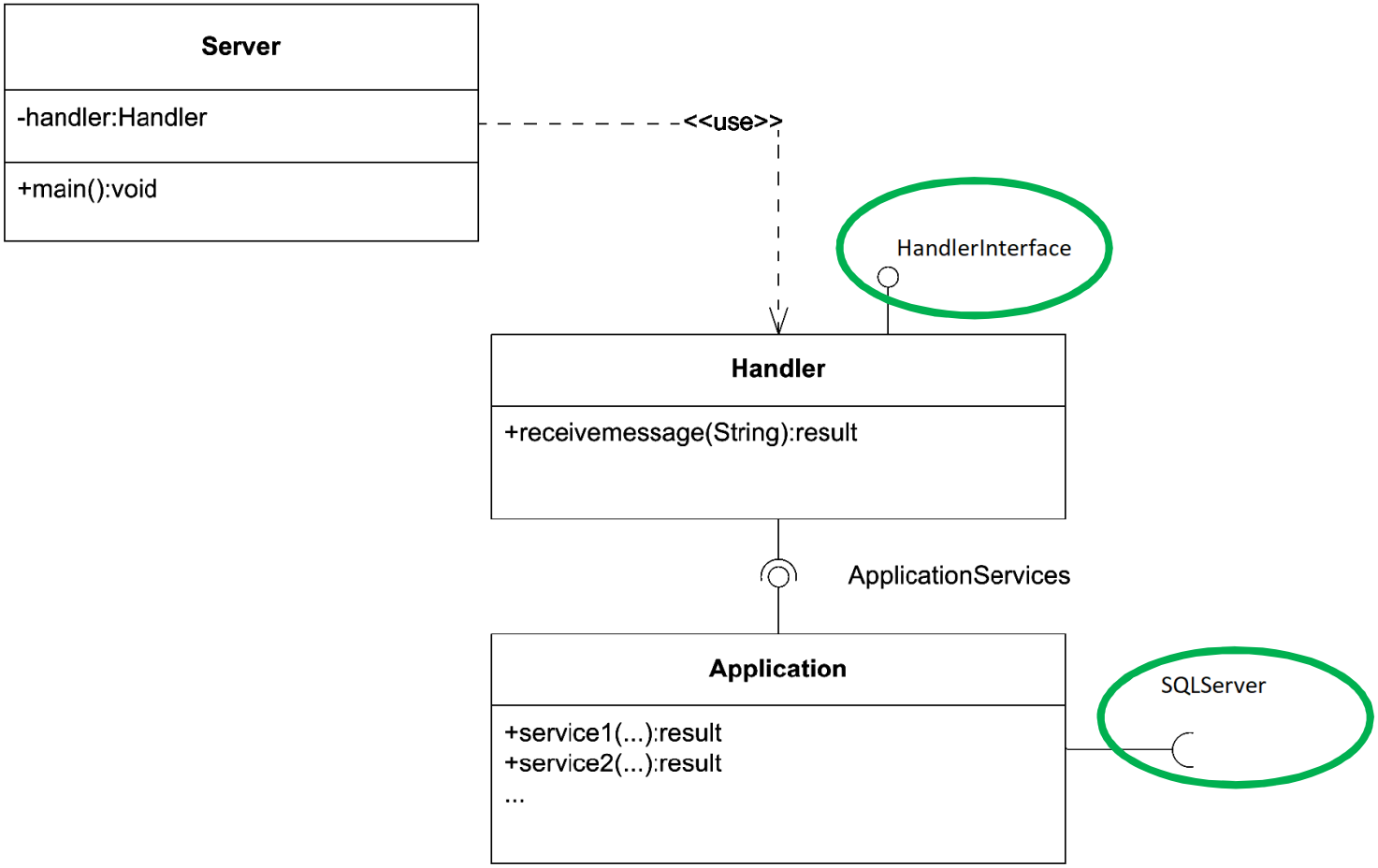
- **Module view** of a typical *three-tier architecture*



UML class diagram (Client interfaces)

Architectural views: E-commerce case study

- **Module view** of a typical *three-tier architecture*

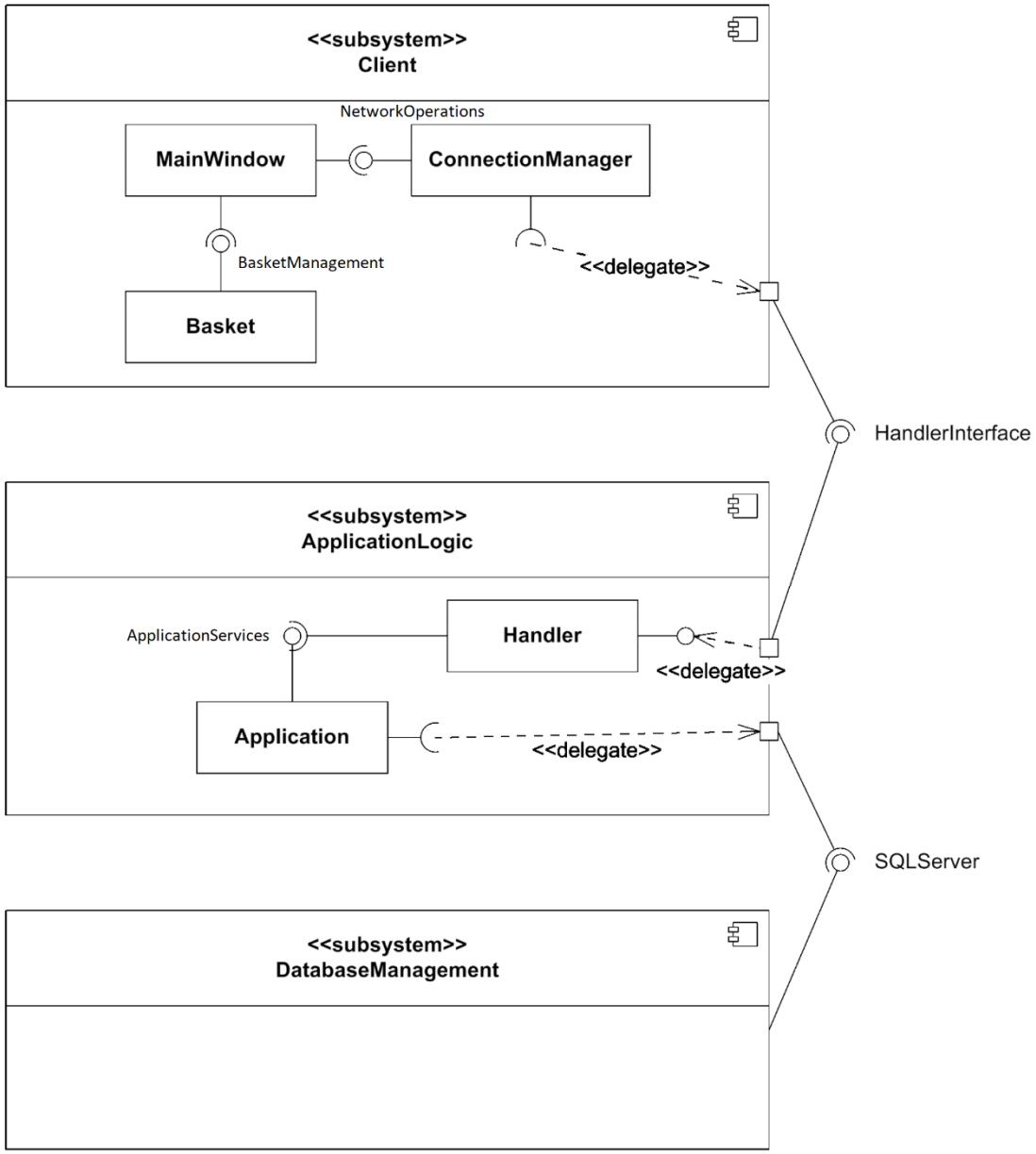


UML class diagram (ApplicationLogic component)

Architectural views: E-commerce case study

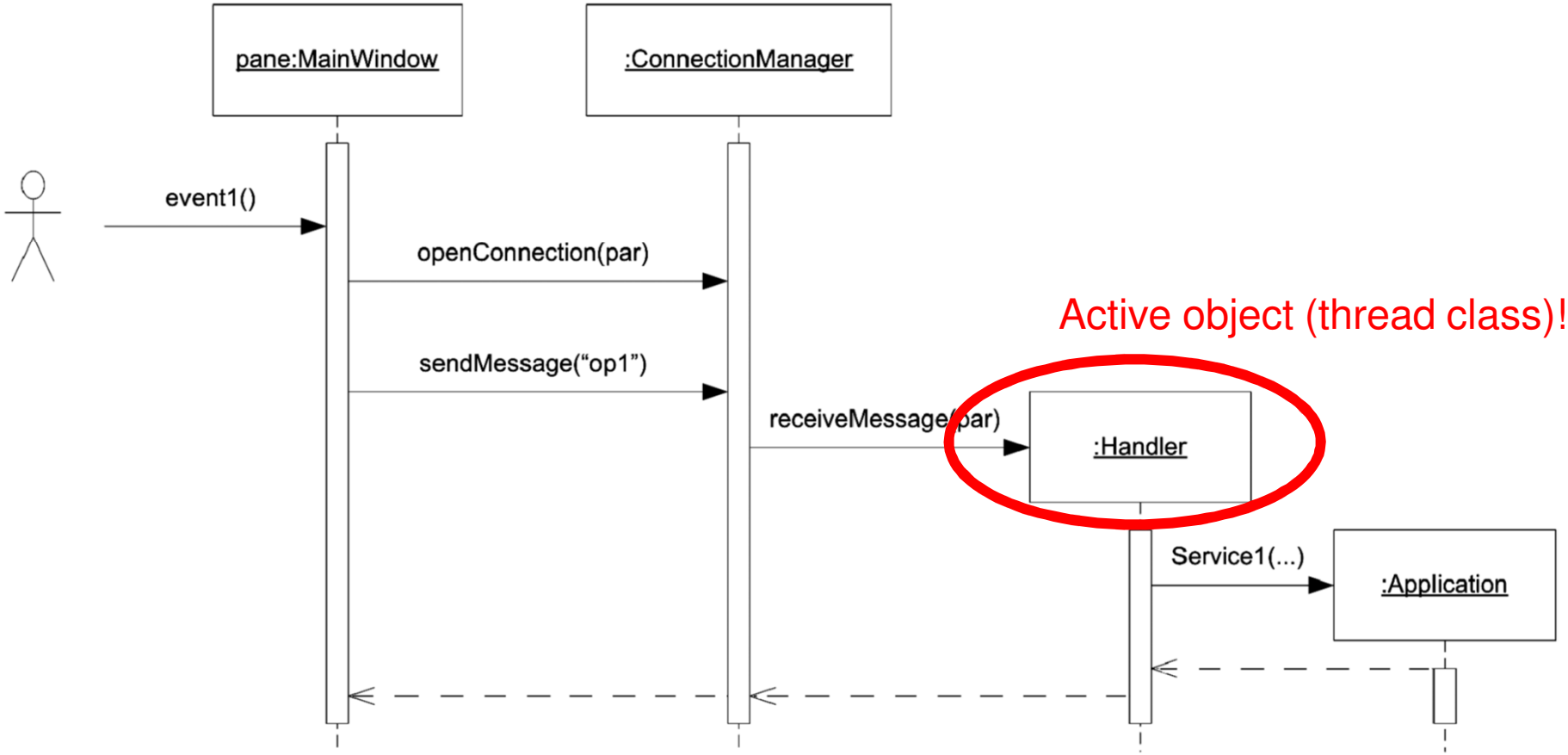
- **Functional view** of a typical *three-tier architecture* (refined!)

UML component diagram (Top level)



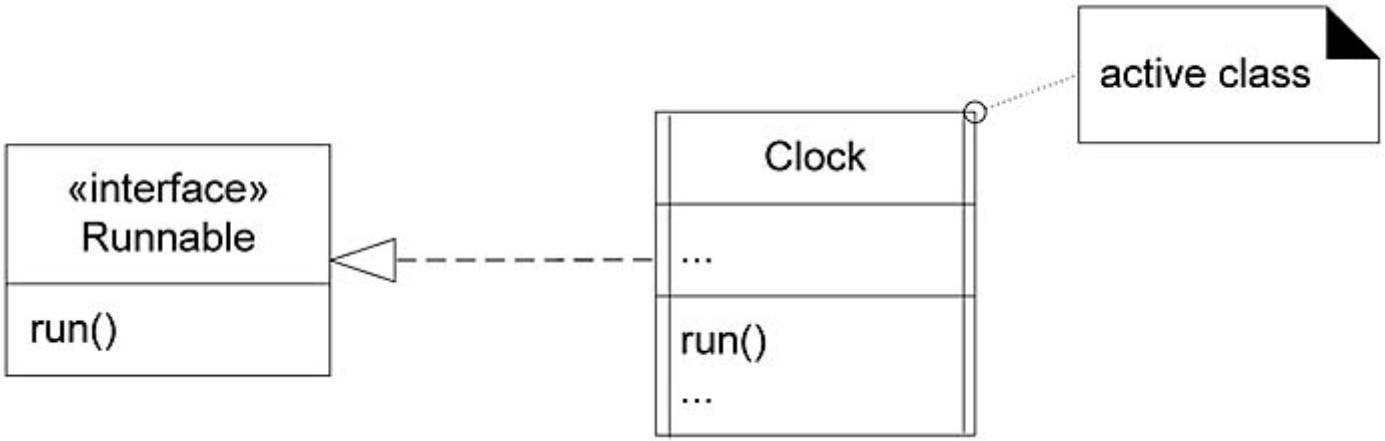
Architectural views: E-commerce case study

- **Scenario view** of a typical *three-tier architecture*



UML sequence diagram (Top level)

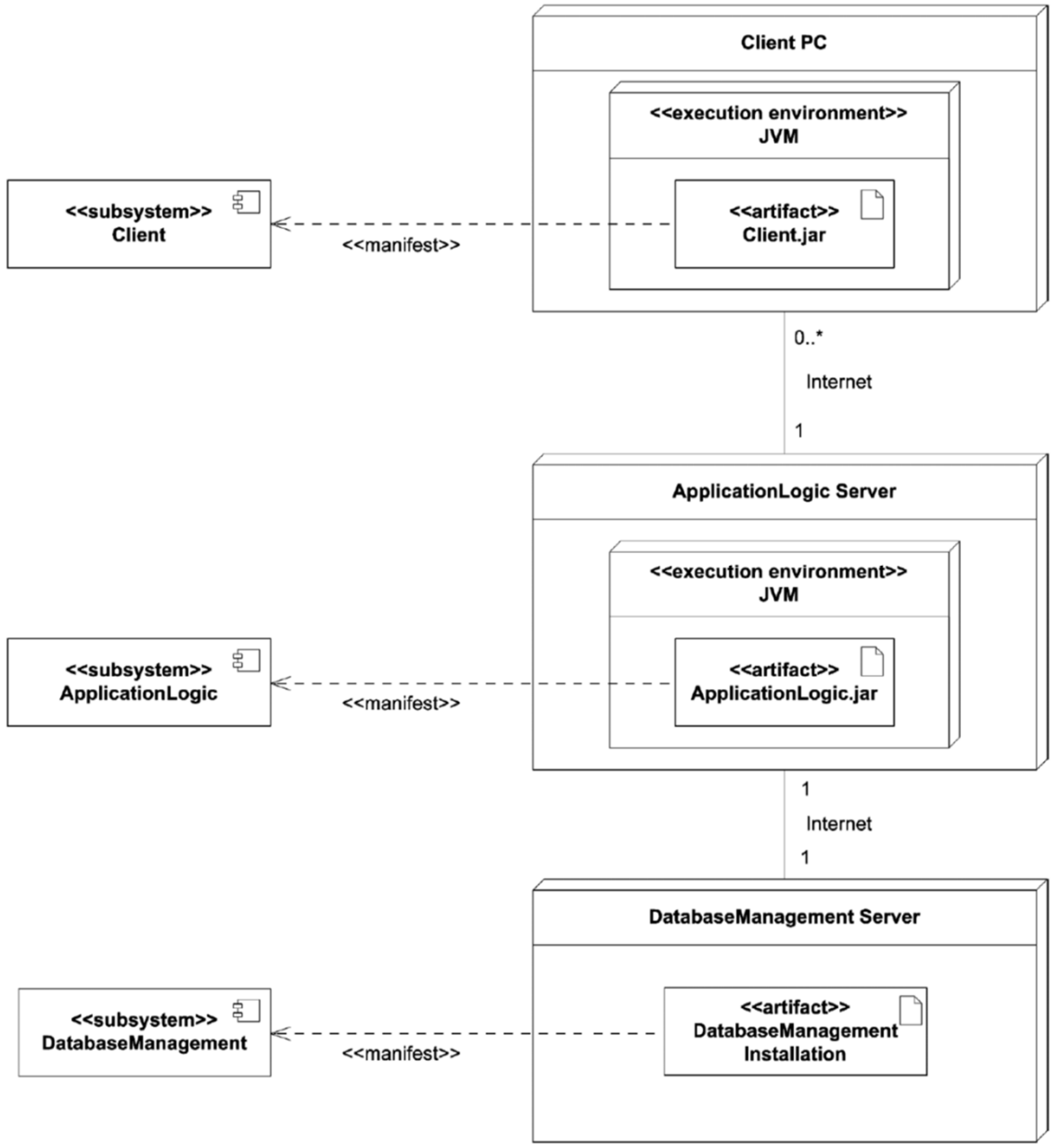
Active classes in UML



Architectural views: E-commerce case study

- **Deployment view** of a typical *three-tier architecture*

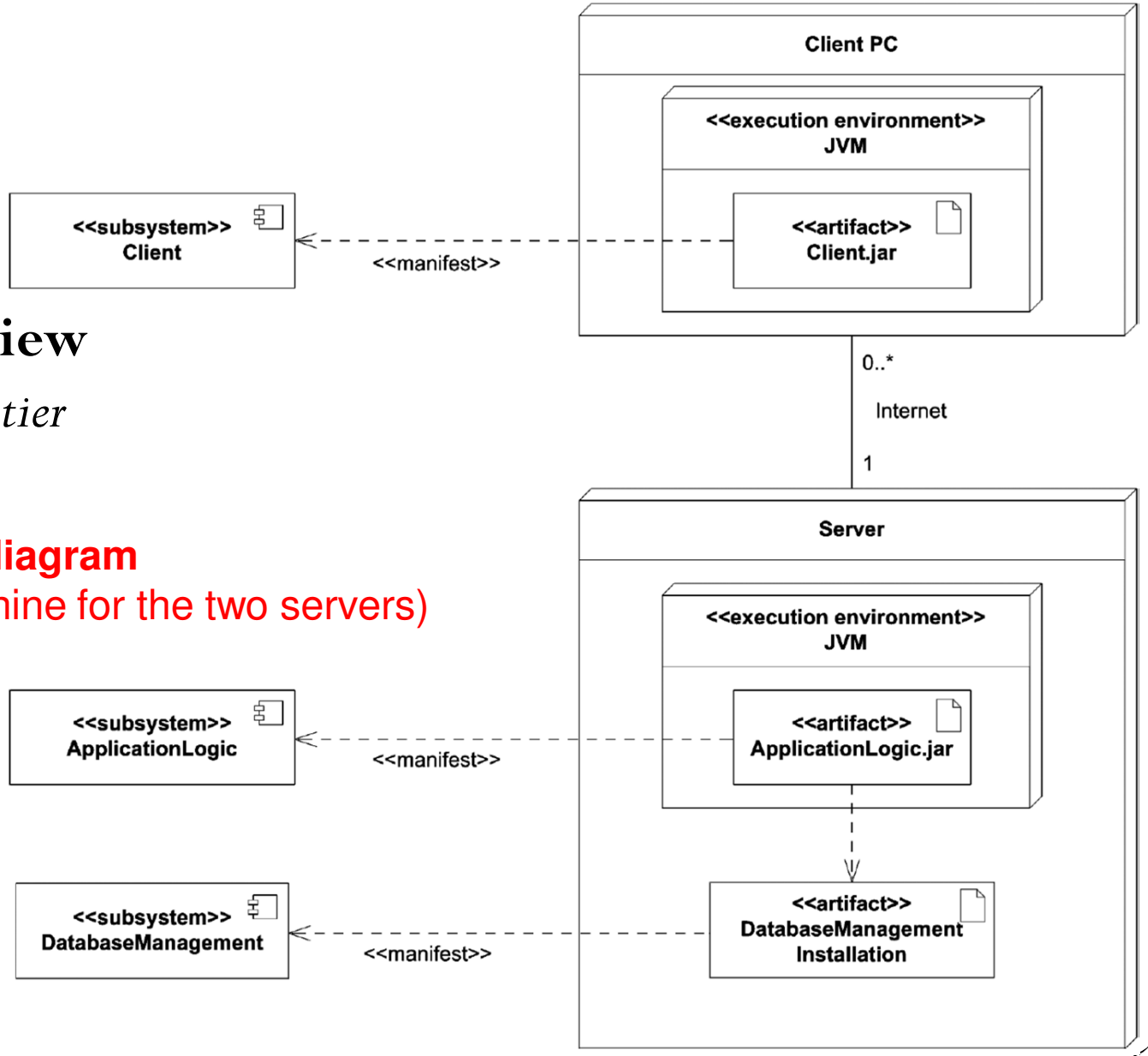
UML deployment diagram (variant 1: two separated servers)



Architectural views: E-commerce case study

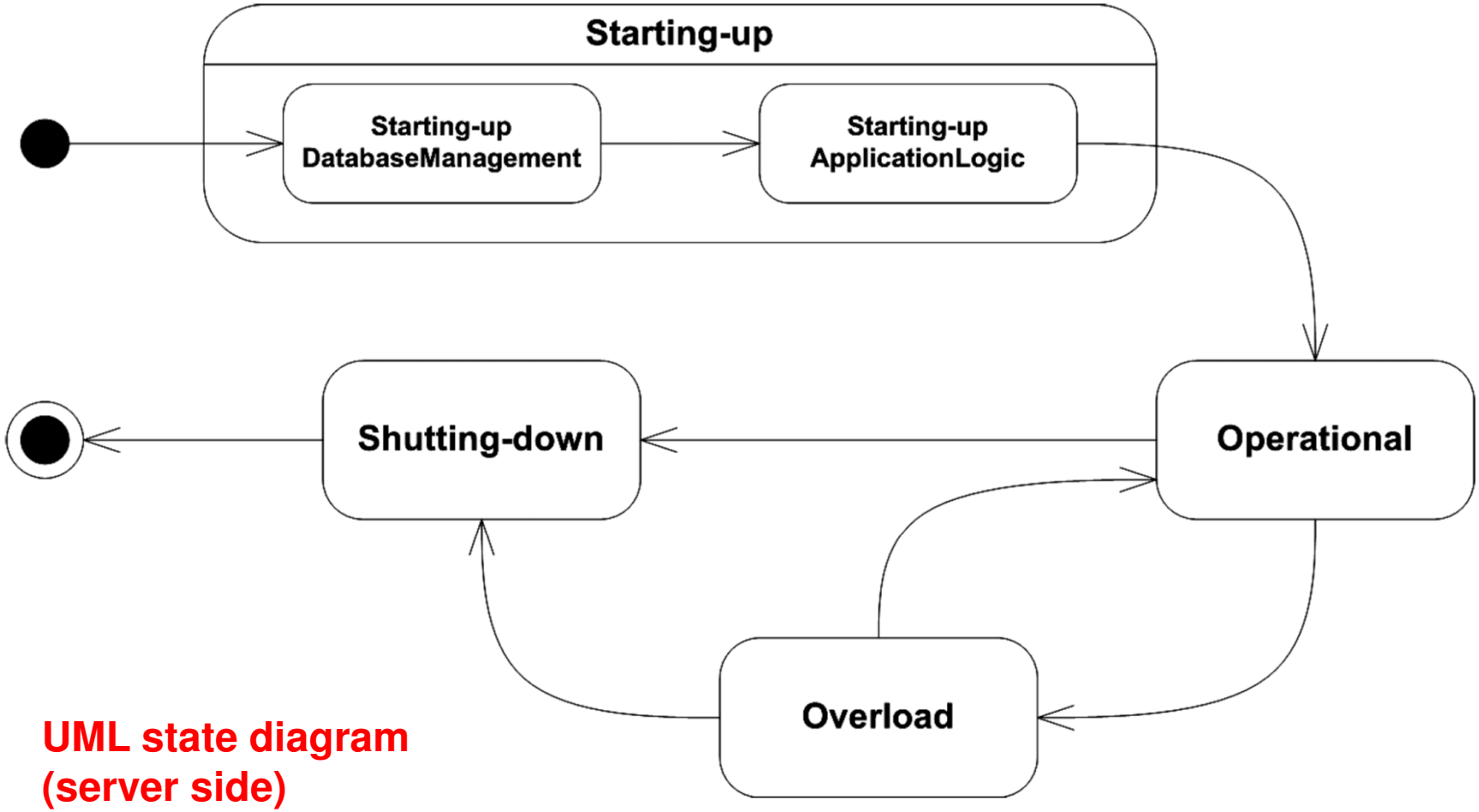
- **Deployment view**
  of a typical *three-tier architecture*

UML deployment diagram
(variant 2: one machine for the two servers)



Architectural views: E-commerce case study

- **Runtime view** of a typical *three-tier architecture*



UML state diagram (server side)