COMPONENT DIAGRAM

INTRODUCTION

- UML component diagrams describe software components and their dependencies to each others
  - A component is an autonomous unit within a system
  - The components can be used to define software systems of arbitrary size and complexity
  - UML component diagrams enable to model the high-level software components, and the interfaces to those components

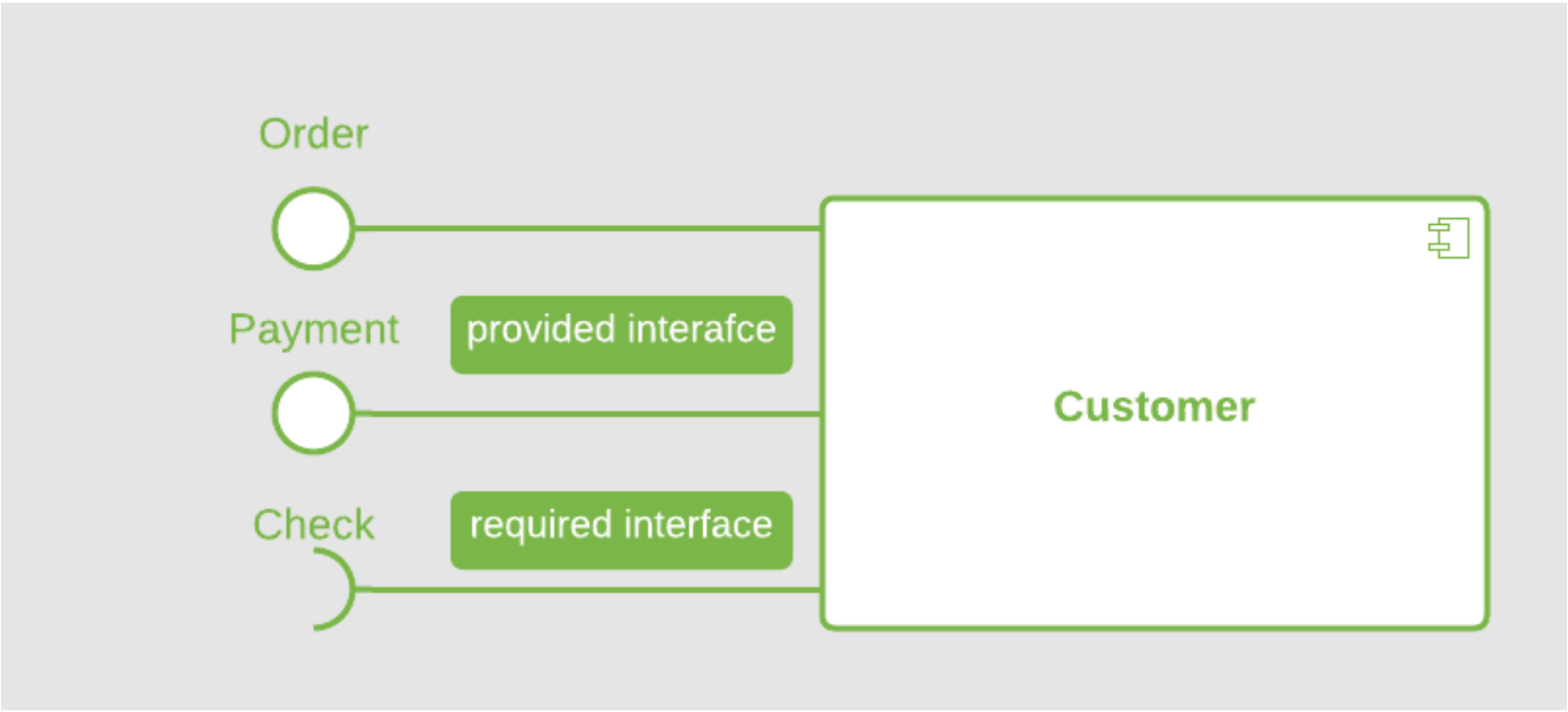
INTRODUCTION

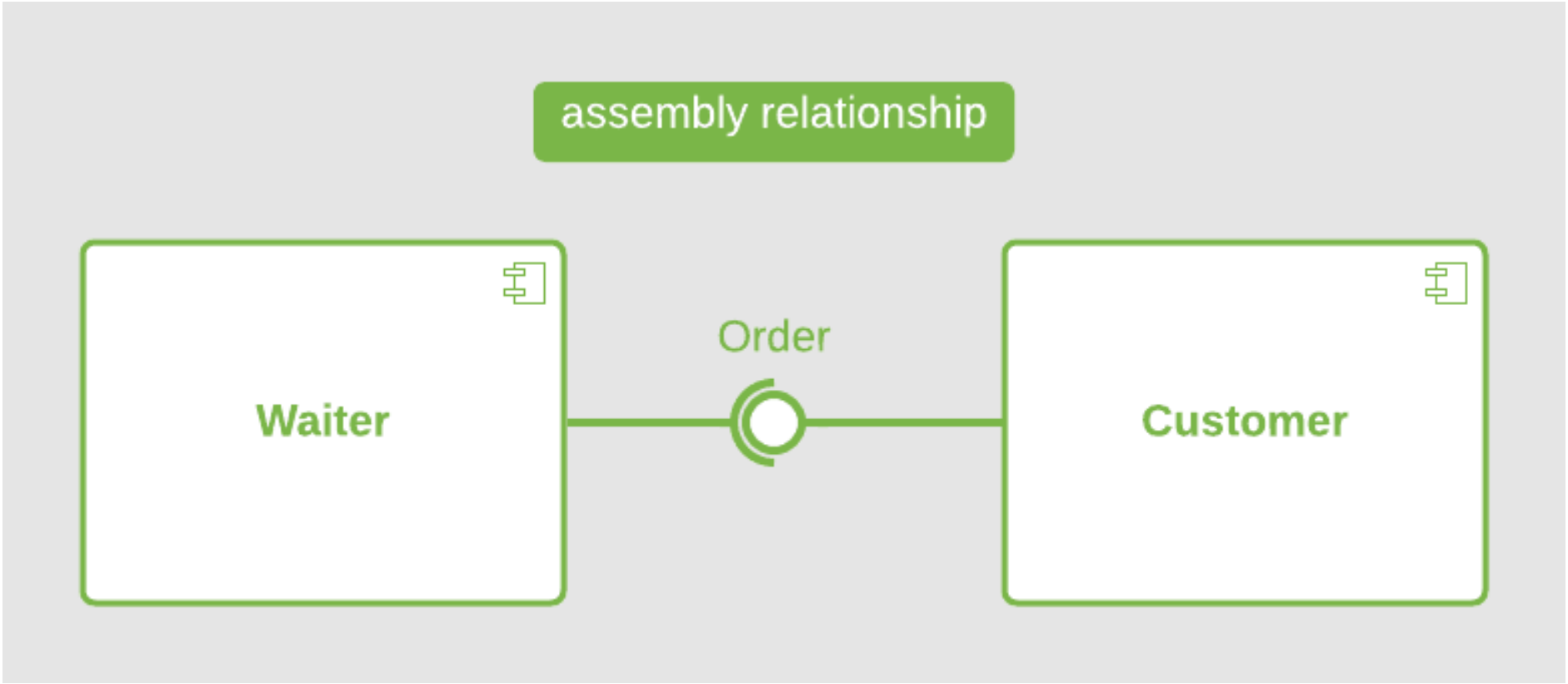
- UML component diagrams describe software components and their dependencies to each others
  - Important for component-based development (CBD)
  - Component and subsystems can be flexibly REUSED and REPLACED
  - A dependency exists between two elements if changes to the definition of one element may cause changes to the other
  - Component Diagrams are often referred to as “wiring diagrams”

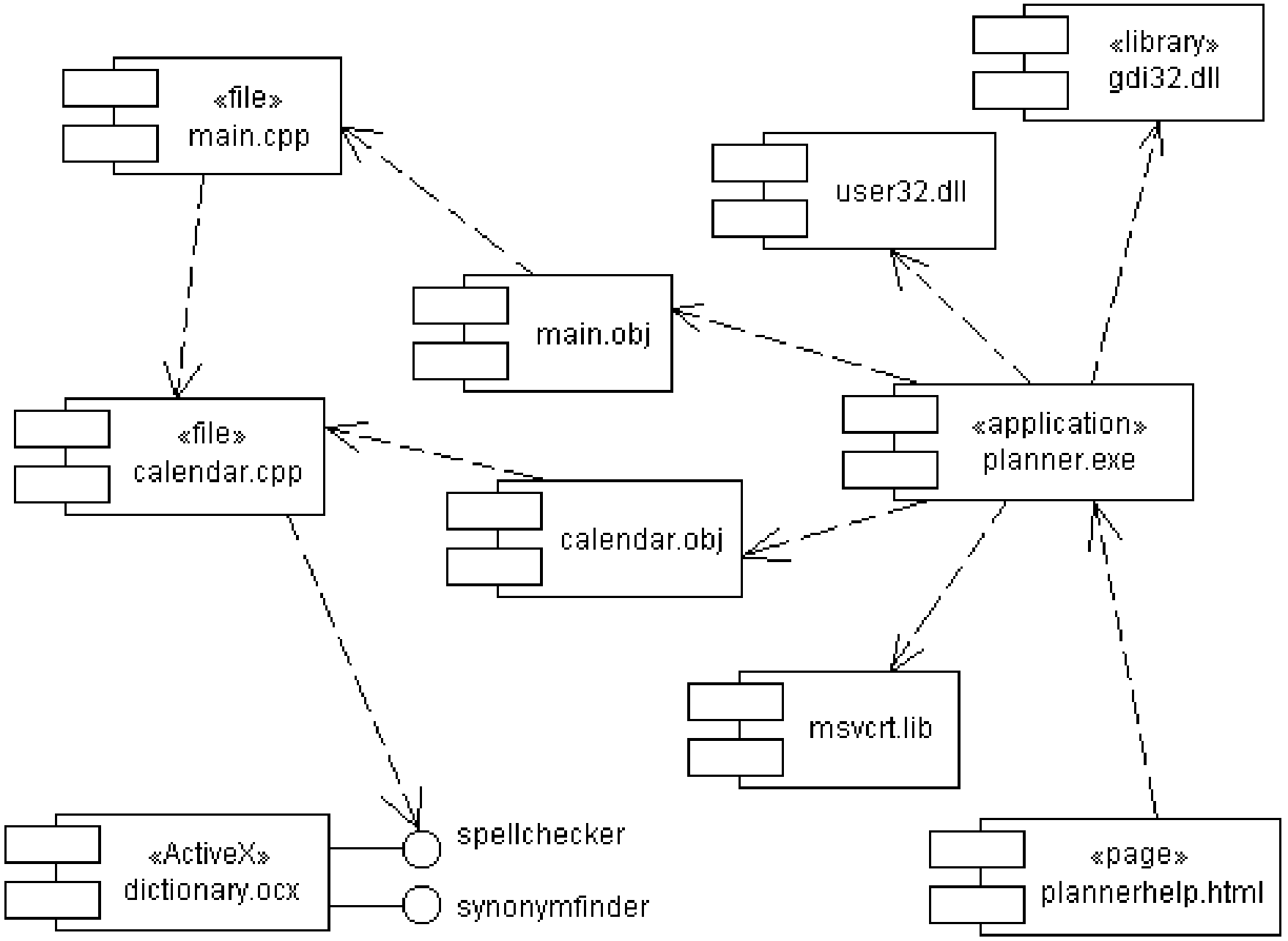
COMPONENT in UML 2.0

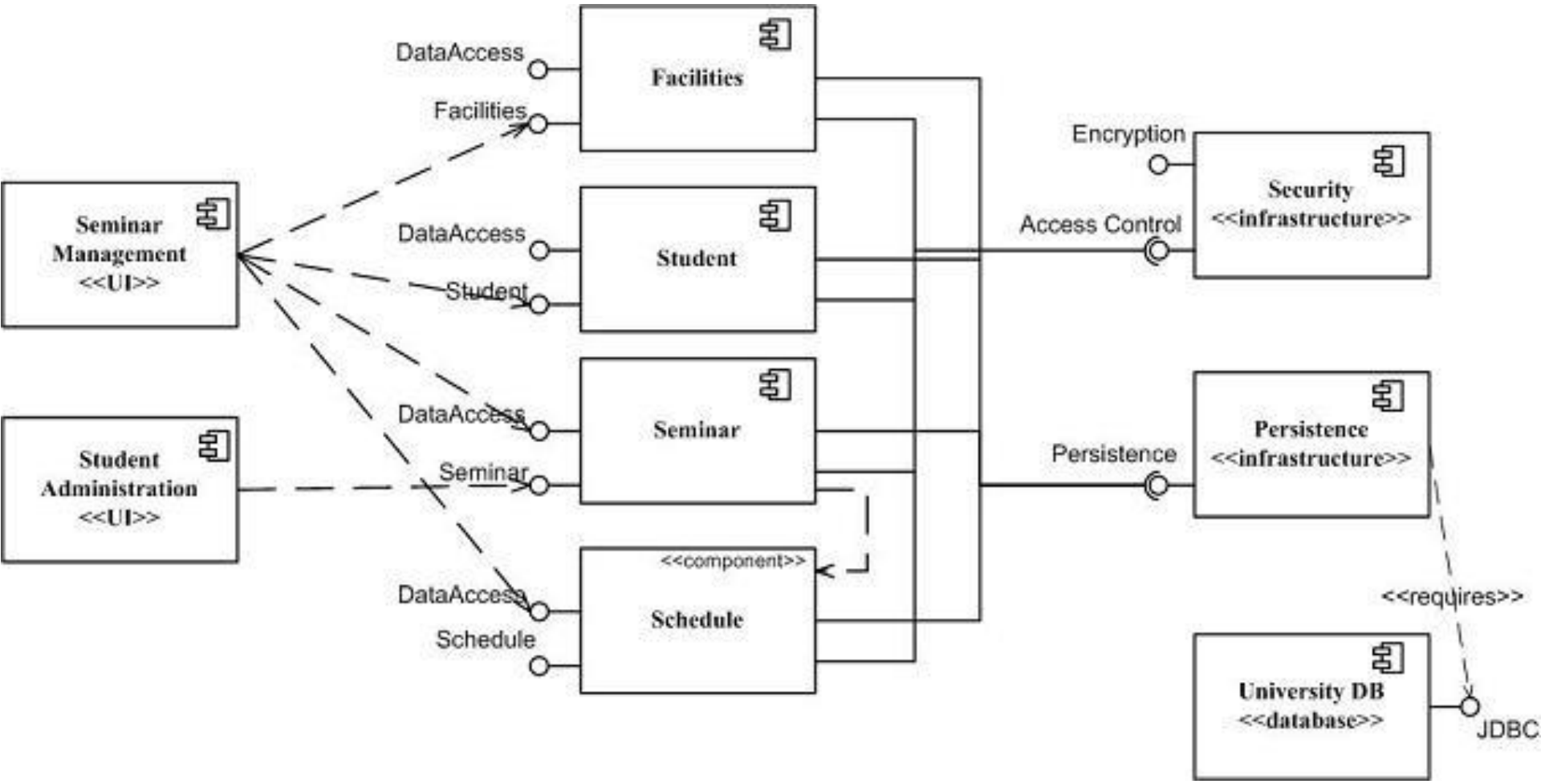
- Modular unit with well-defined interfaces that is replaceable within its environment

- **Autonomous** unit within a system
  - Has one or more provided and required interfaces
  - Its internals are hidden and inaccessible
  - A component is encapsulated
  - Its dependencies are designed such that it can be treated as independently as possible


Order
Payment
provided interafce
Check
required interface
Customer

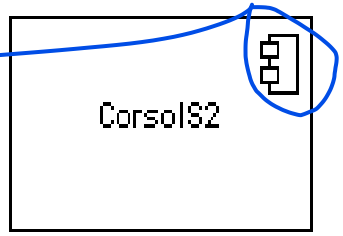

assembly relationship
Waiter
Order
Customer


«file»
main.cpp
«library»
gdi32.dll
user32.dll
main.obj
«file»
calendar.cpp
«application»
planner.exe
calendar.obj
msvcrt.lib
«ActiveX»
dictionary.ocx
spellchecker
synonymfinder
«page»
plannerhelp.html


Seminar Management <<UI>>
Student Administration <<UI>>
DataAccess
Facilities
Facilities
DataAccess
Student
Student
DataAccess
Seminar
Seminar
DataAccess
Schedule
Schedule
<<component>>
Facilities
Student
Seminar
Schedule
Encryption
Access Control
Security <<infrastructure>>
Persistence
Persistence <<infrastructure>>
<<requires>>
University DB <<database>>
JDBC

COMPONENT NOTATION

- A component is shown as a rectangle with
  - A keyword <<component>>
  - Optionally, in the right hand corner a component icon can be displayed
    - A component icon is a rectangle with two smaller rectangles jutting out from the left-hand side
    - This symbol is a visual stereotype
  - The component name
- Components can be labelled with a stereotype.
- There are a number of standard stereotypes eg: <<entity>>, <<subsystem>>



Component ELEMENTS

- A component can have
  - Interfaces

    An interface represents a declaration of a set of operations and obligations
  - Usage dependencies

    A usage dependency is relationship which one element requires another element for its full implementation
  - Ports

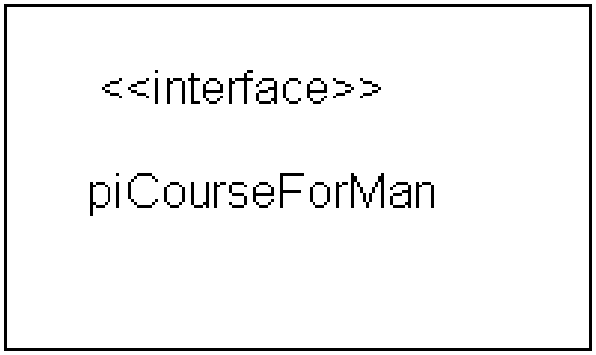
    Port represents an interaction point between a component and its environment
  - Connectors
    - Connect two components
    - Connect the external contract of a component to the internal structure

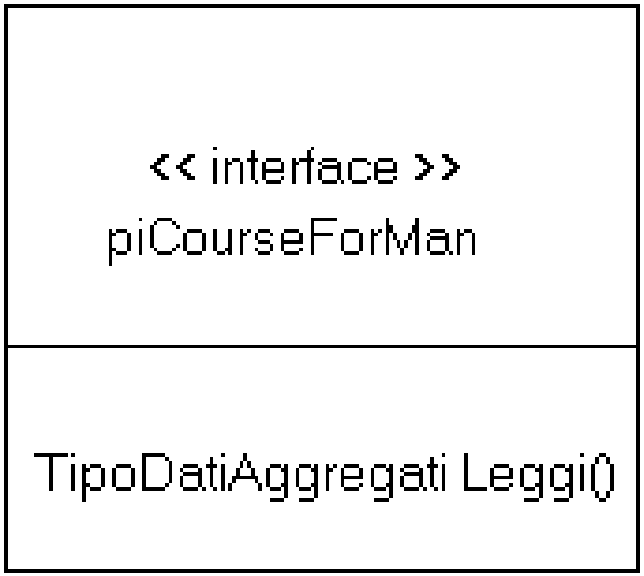
INTERFACE

- 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

- May be shown using a rectangle symbol with a keyword «interface» preceding the name.
- For displaying the full signature, the interface rectangle can be expanded to show details.
- Types
  - Provided
  - Required





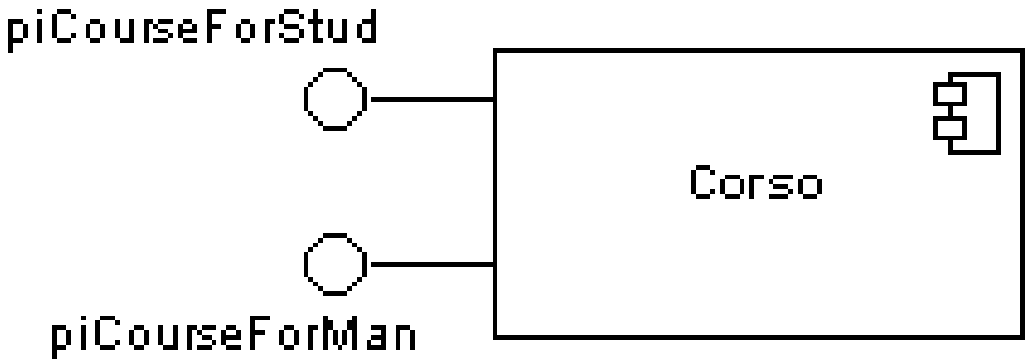
INTERFACE

- 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

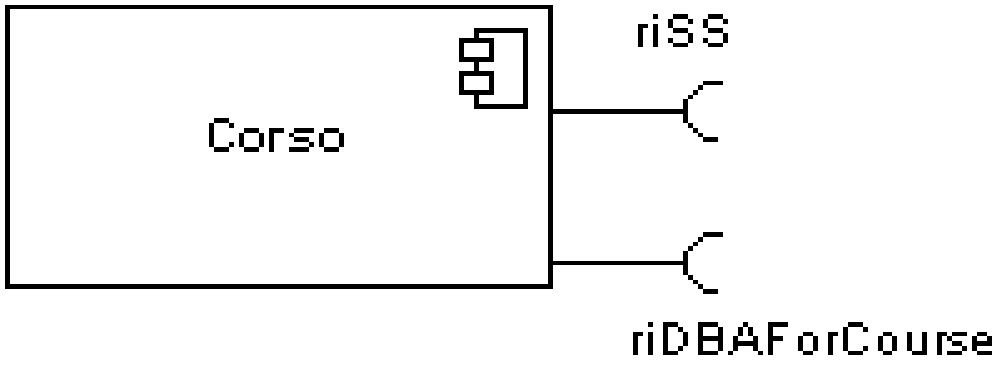
- A required interface
  - Characterize services that the component expects from its environment
  - Is modeled using a socket, labelled with the name, attached by a solid line to the component
  - In UML 1.x were modeled using a dashed arrow

INTERFACE

- A provided interface

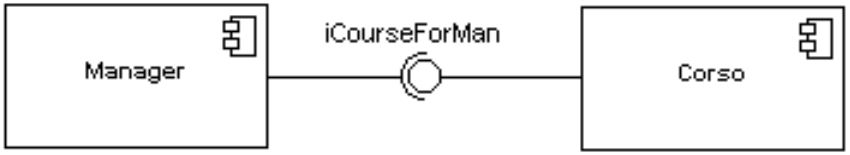


- A required interface



INTERFACE

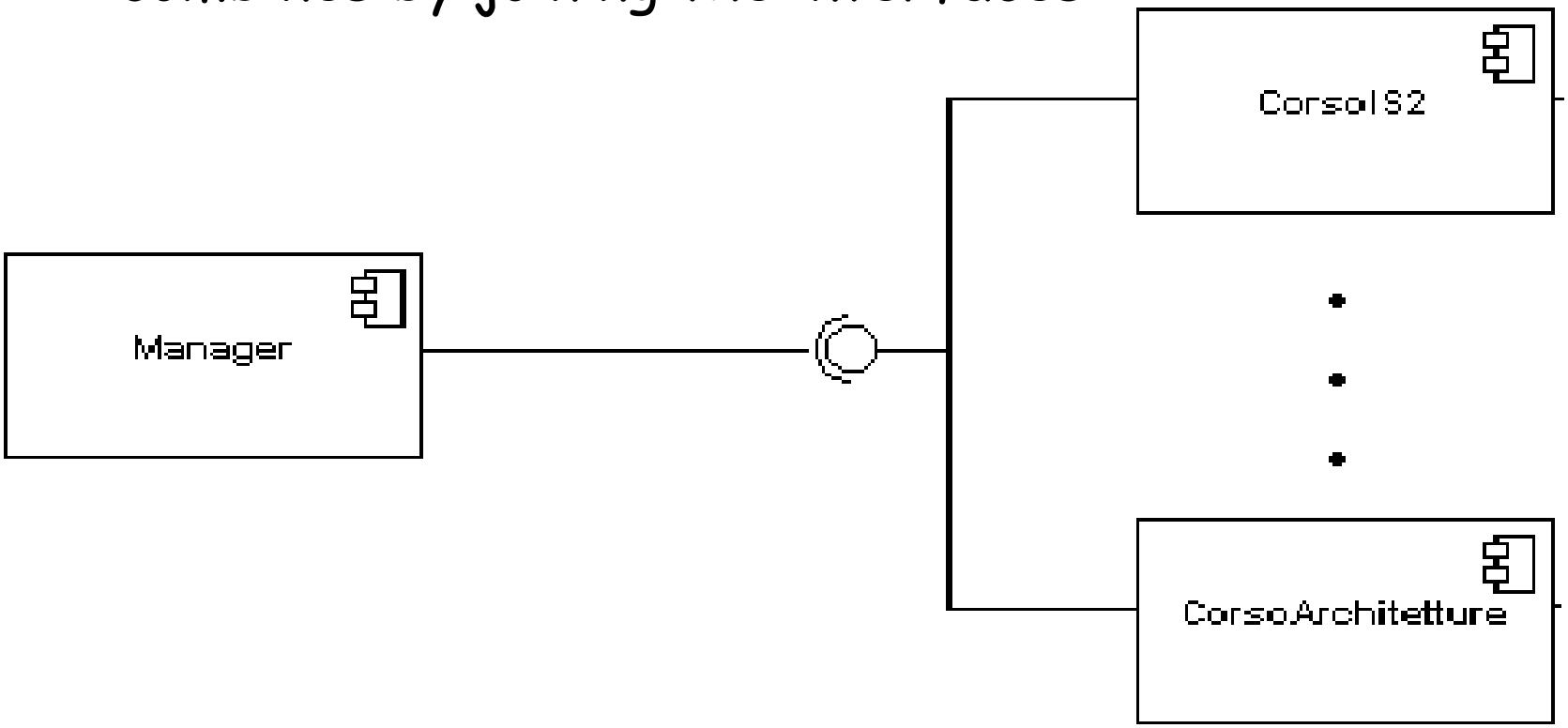
- Where two components/classes provide and require the same interface, these two notations may be combined



  - The ball-and-socket notation hint at that interface in question serves to mediate interactions between the two components
  - If an interface is shown using the rectangle symbol, we can use an alternative notation, using dependency arrows

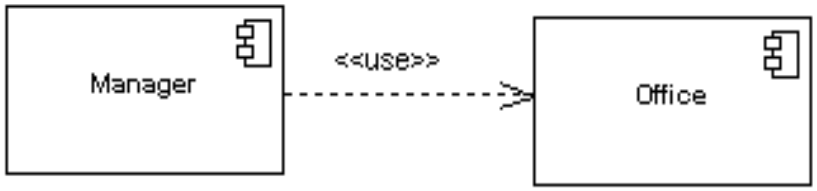
INTERFACE

- In a system context where there are multiple components that require or provide a particular interface, a notation abstraction can be used that combines by joining the interfaces

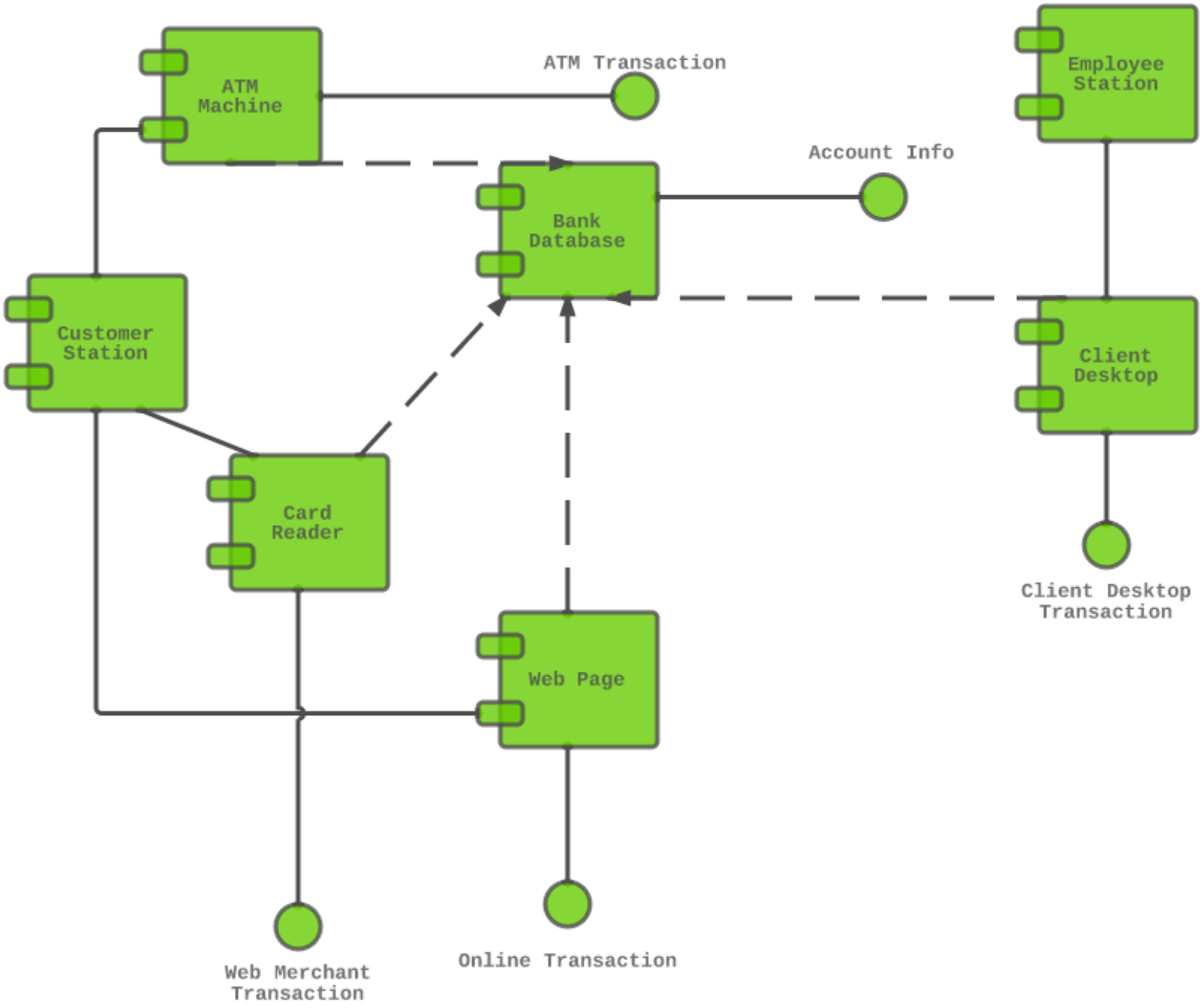


USAGE DEPENDENCIES

- Components can be connected by usage dependencies.

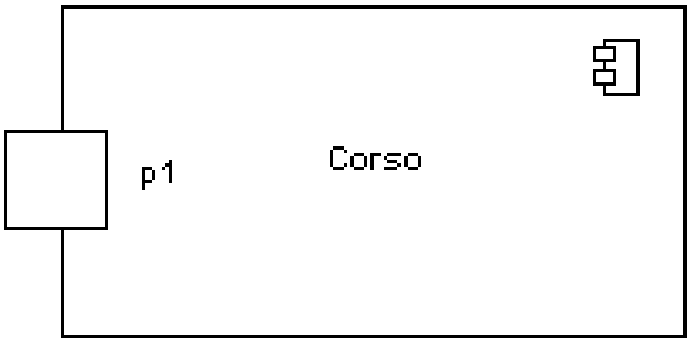


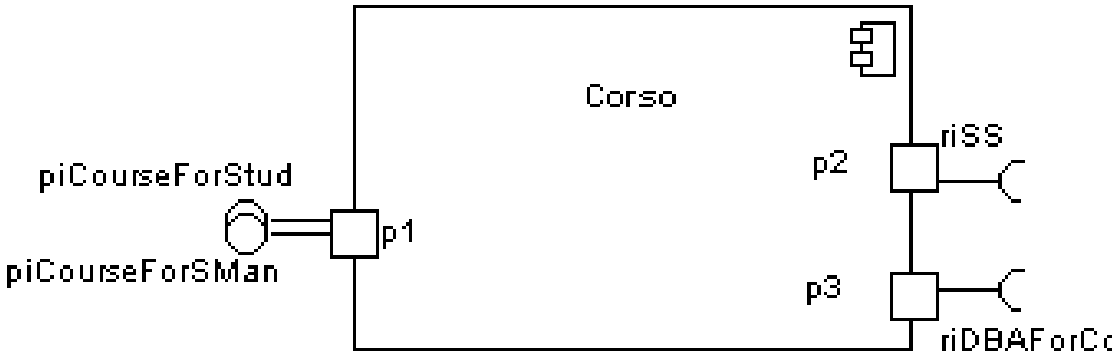
- Usage Dependency
  - A usage dependency is relationship which one element requires another element for its full implementation
  - Is a dependency in which the client requires the presence of the supplier
  - Is shown as dashed arrow with a <<use>> keyword.
  - The arrowhead point from the dependent component to the one of which it is dependent.


ATM Machine
ATM Transaction
Employee Station
Account Info
Bank Database
Customer Station
Client Desktop
Card Reader
Client Desktop Transaction
Web Page
Web Merchant Transaction
Online Transaction

PORT

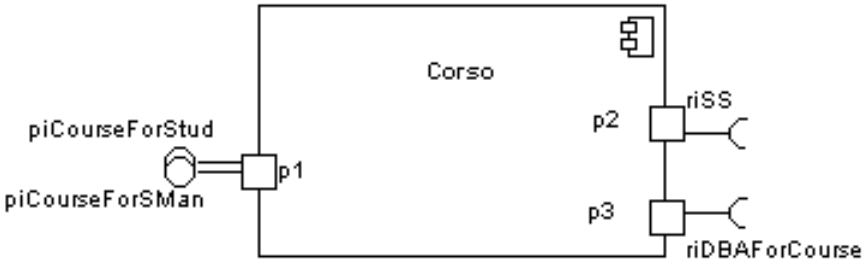
- Specifies a distinct interaction point
  - Between that component and its environment
  - Between that component and its internal parts
- Is shown as a small square symbol
- Ports can be named, and the name is placed near the square symbol
- Is associated with the interfaces that specify the nature of the interactions that may occur over a port

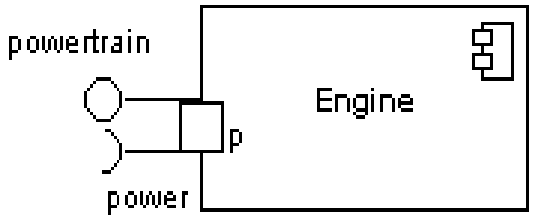




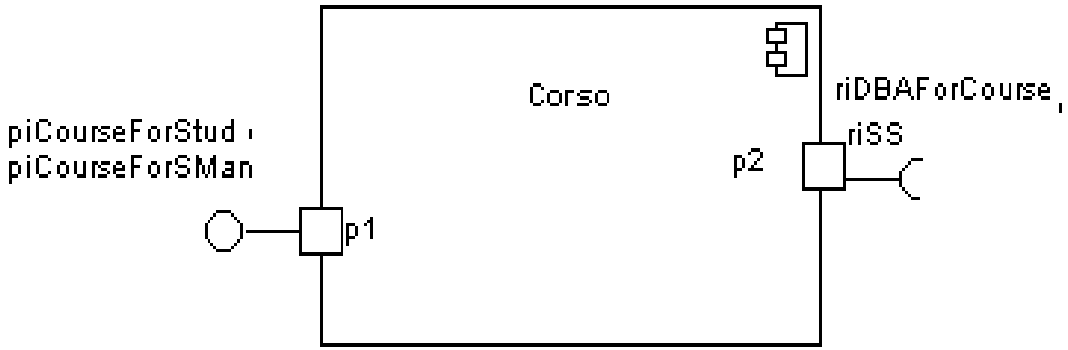
PORT

- Ports can support unidirectional communication or bi-directional communication



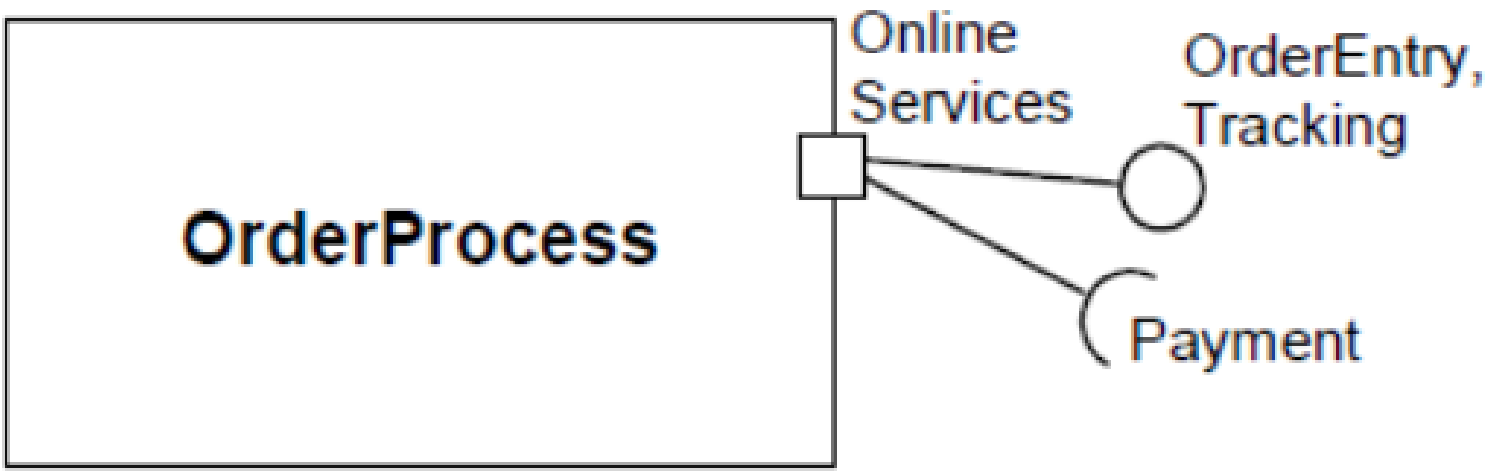


- If there are multiple interfaces associated with a port, these interfaces may be listed with the interface icon, separated by a commas



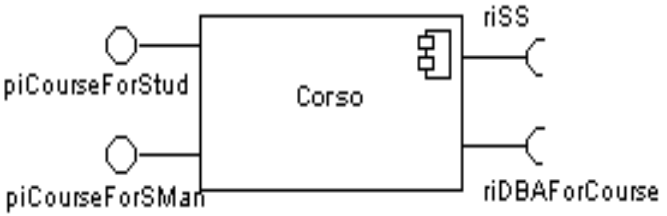
PORT

- All interactions of a component with its environment are achieved through a port
- The internals are fully isolated from the environment
- This allows such a component to be used in any context that satisfies the constraints specified by its ports
- Ports are not defined in UML 1.x

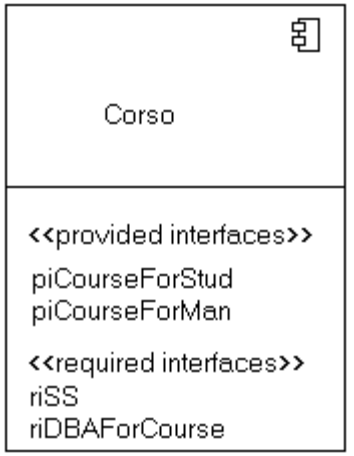

OrderProcess
Online
Services
OrderEntry,
Tracking
Payment

EXTERNAL VIEW

- A component have an external view and an internal view.
- An external view (or black box view) shows publicly visible properties and operations

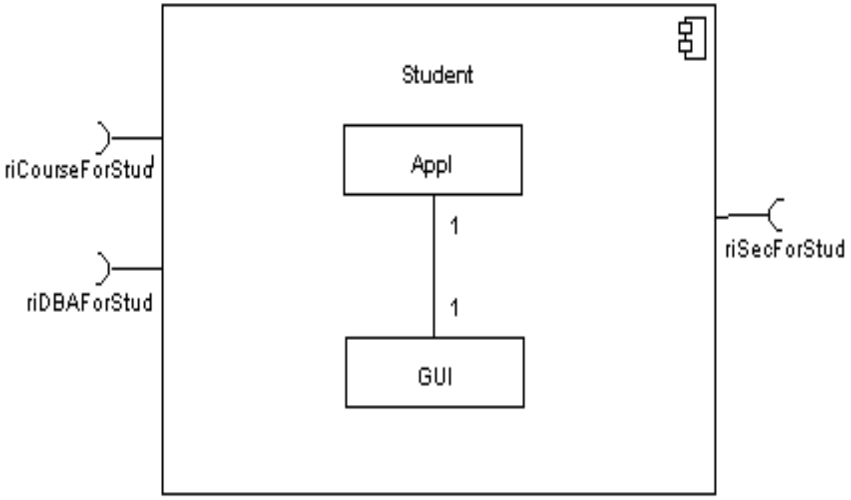


- An external view of a component is by means of interface symbols sticking out of the component box
- The interface can be listed in the compartment of a component box



INTERNAL VIEW

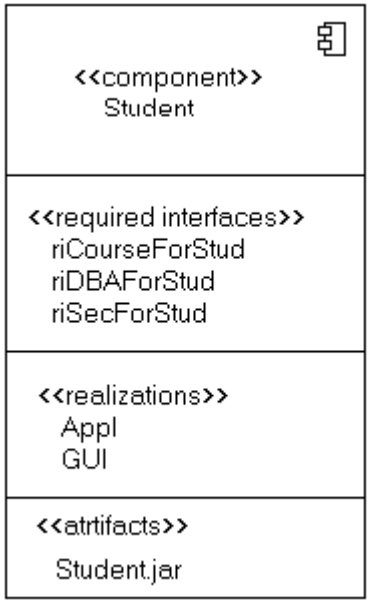
- An internal, or white box view of a component.
- The realizing classes /components are nested within the component shape



- Realization is a relationship between two set of model elements
  - One represents a specification
  - The other represent an implementation of the latter

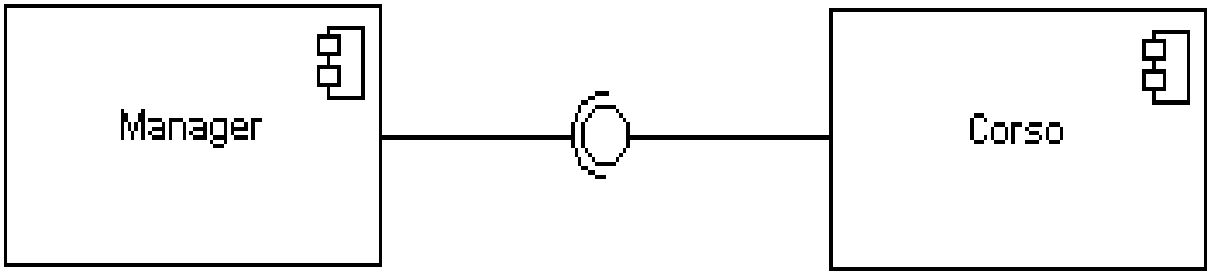
INTERNAL VIEW

- The internal class that realize the behavior of a component may be displayed in an additional compartment
- Compartments can also be used to display parts, connectors or implementation artifacts
- An artifact is the specification of a phisycal piece of information



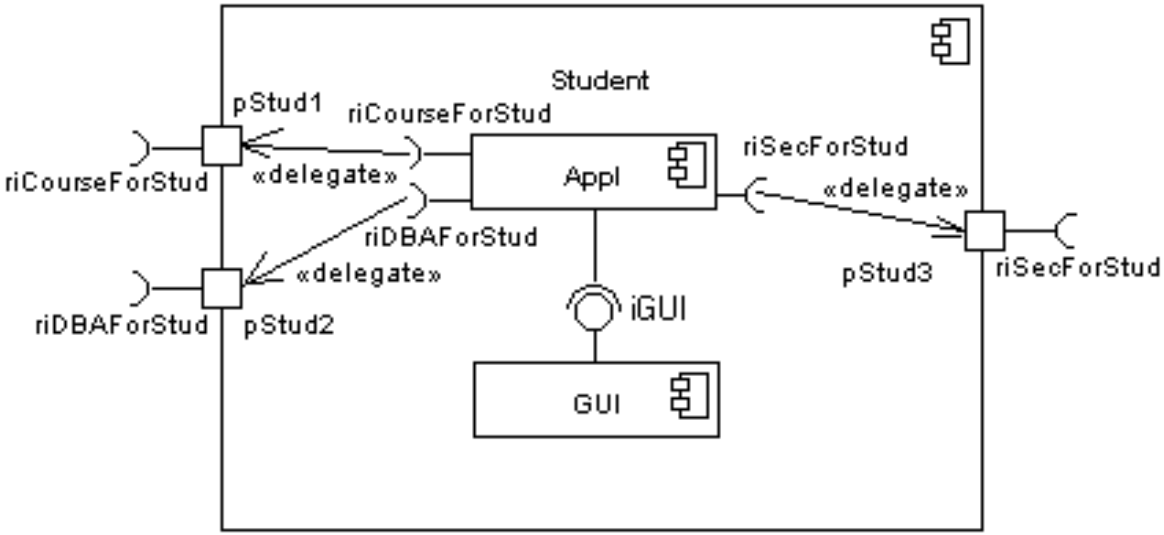
CONNECTORS

- Two kinds of connectors:
  - Delegation
  - Assembly
- ASSEMBLY CONNECTOR
  - A connector between 2 components defines that one component provides the services that another component requires
  - Must only be defined from a required interface to a provided interface
  - An assembly connector is notated by a "ball-and-socket" connection



CONNECTORS

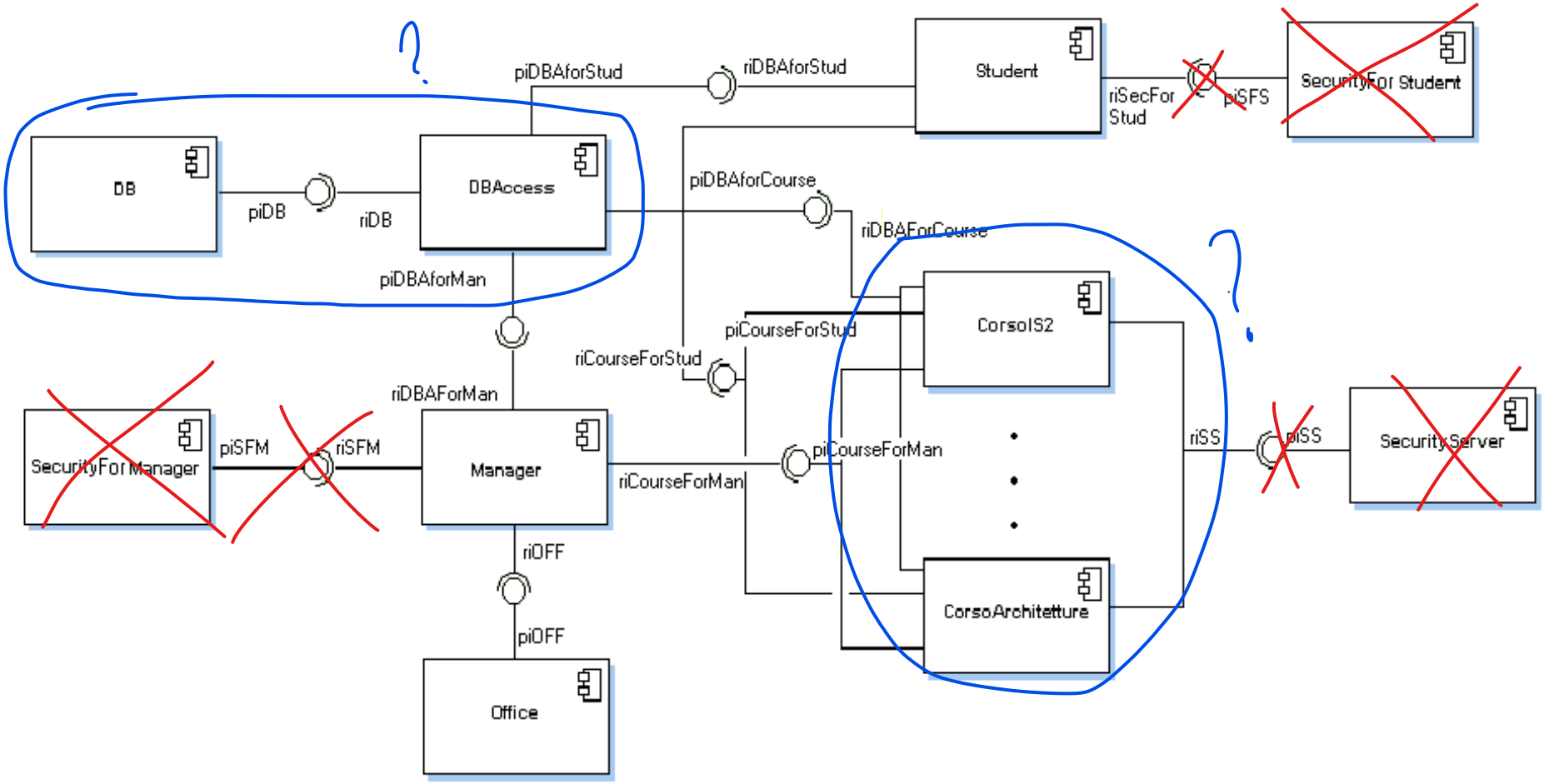
- DELEGATION CONNECTOR
  - Links the external contract of a component to the internal realization
  - Represents the forwarding of signals
  - He must only be defined between used interfaces or ports of the same kind



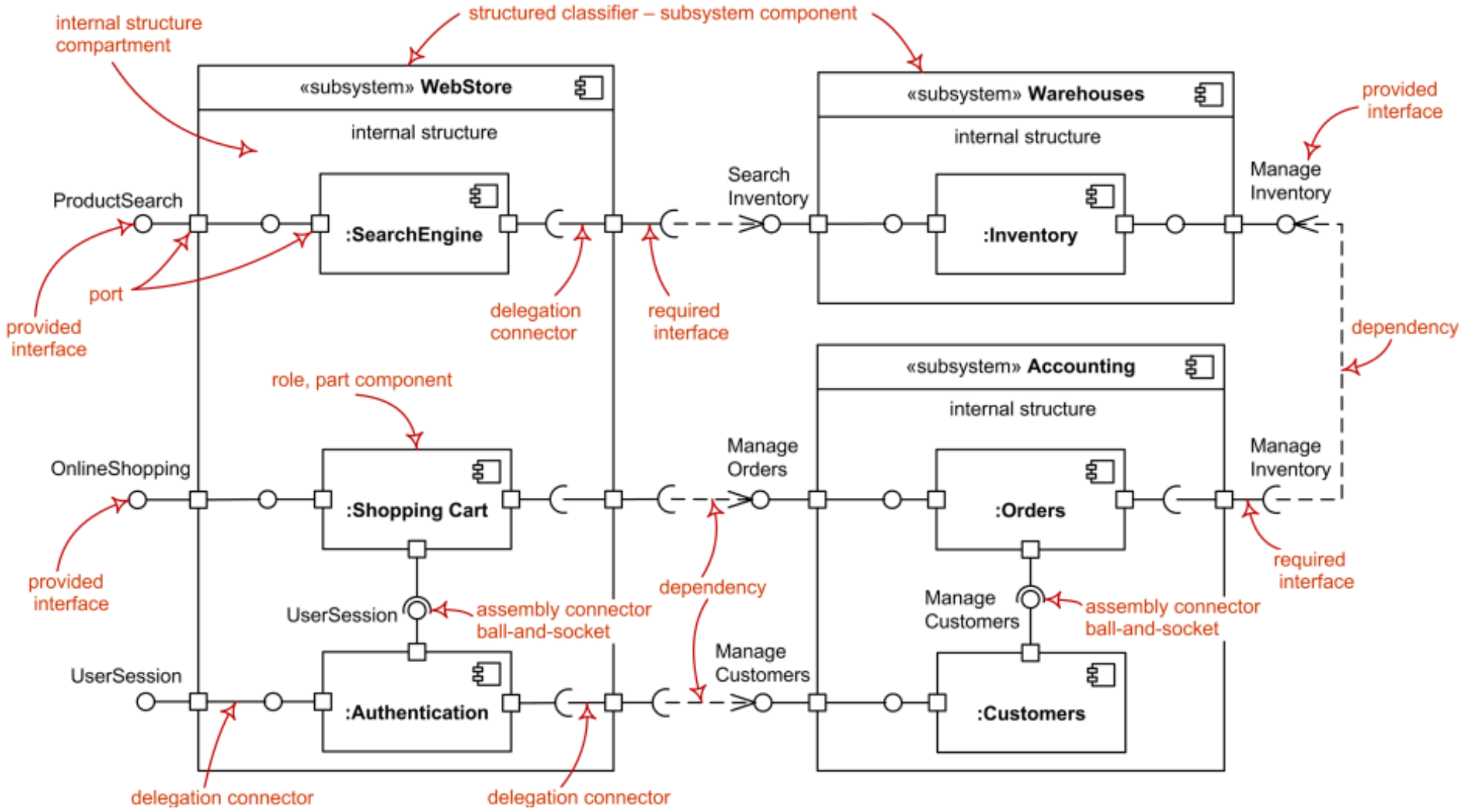
CASE STUDY

- Development of an application collecting students' opinions about courses
- A student can
  - Read
  - Insert
  - Update
  - Make data permanent about the courses in his schedule
- A professor can only see statistic elaboration of the data
- The student application must be installed in pc client (sw1, sw2)
- The manager application must be installed in pc client (in the manager's office)
- There is one or more servers with DataBase and components for courses management

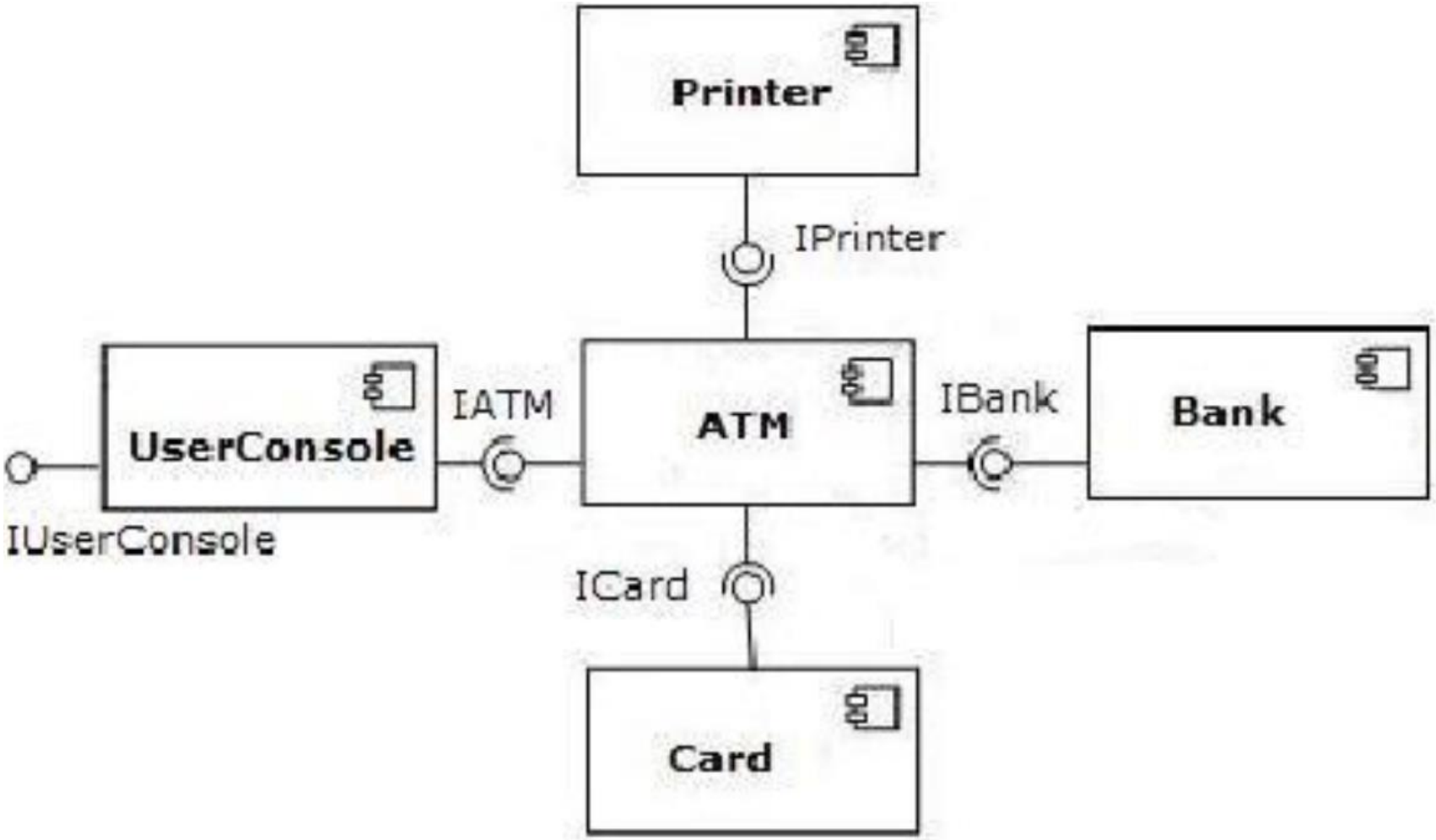
CASE STUDY


piDBAforStud
riDBAforStud
Student
riSecFor Stud
piSFS
SecurityFor Student
DB
piDB
riDB
DBAccess
piDBAforCourse
riDBAForCourse
piDBAforMan
piCourseForStud
riCourseForStud
CorsoIS2
riDBAForMan
SecurityFor Manager
piSFM
riSFM
Manager
piCourseForMan
riCourseForMan
riSS
piSS
SecurityServer
riOFF
piOFF
CorsoArchitetture
Office

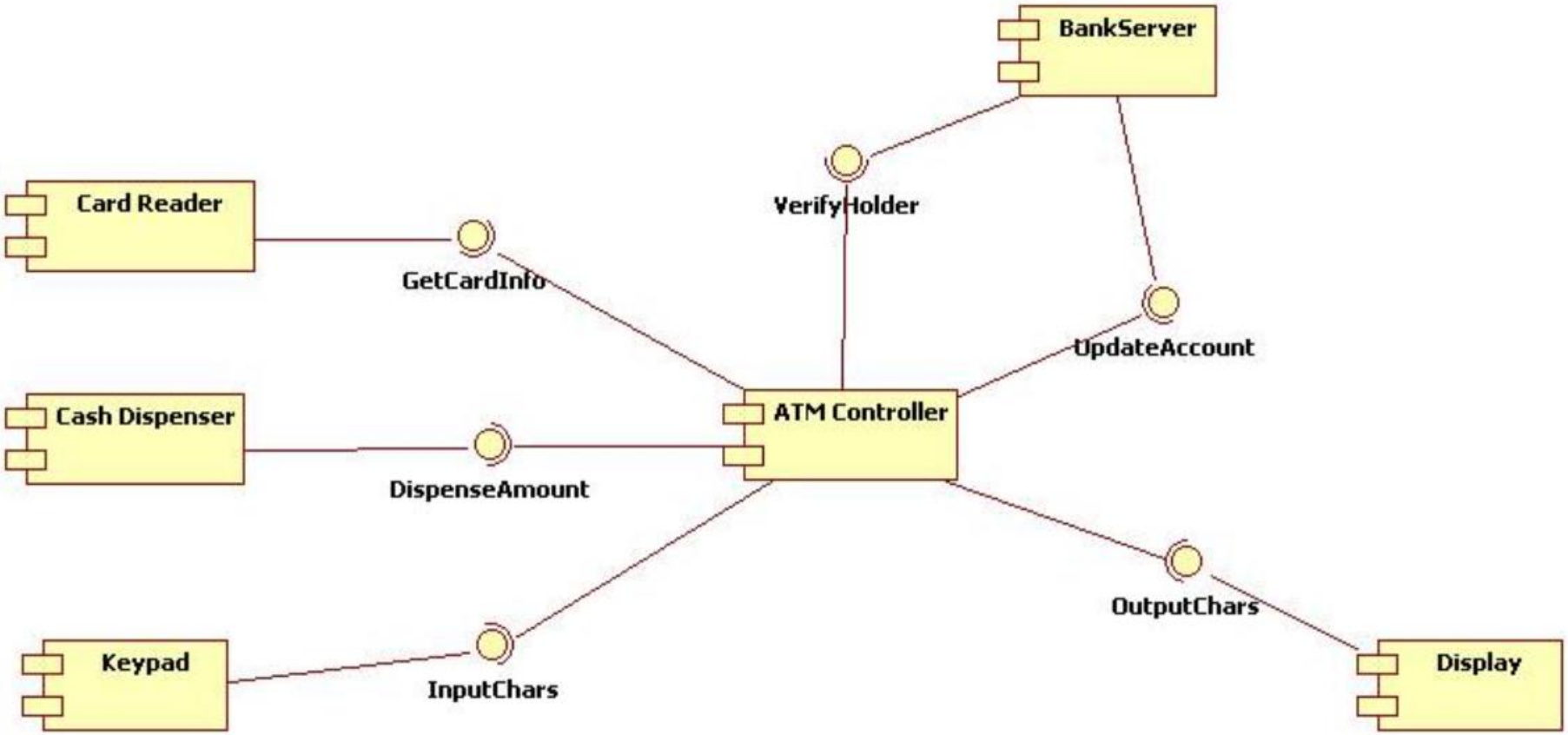
ONLINE SHOPPING

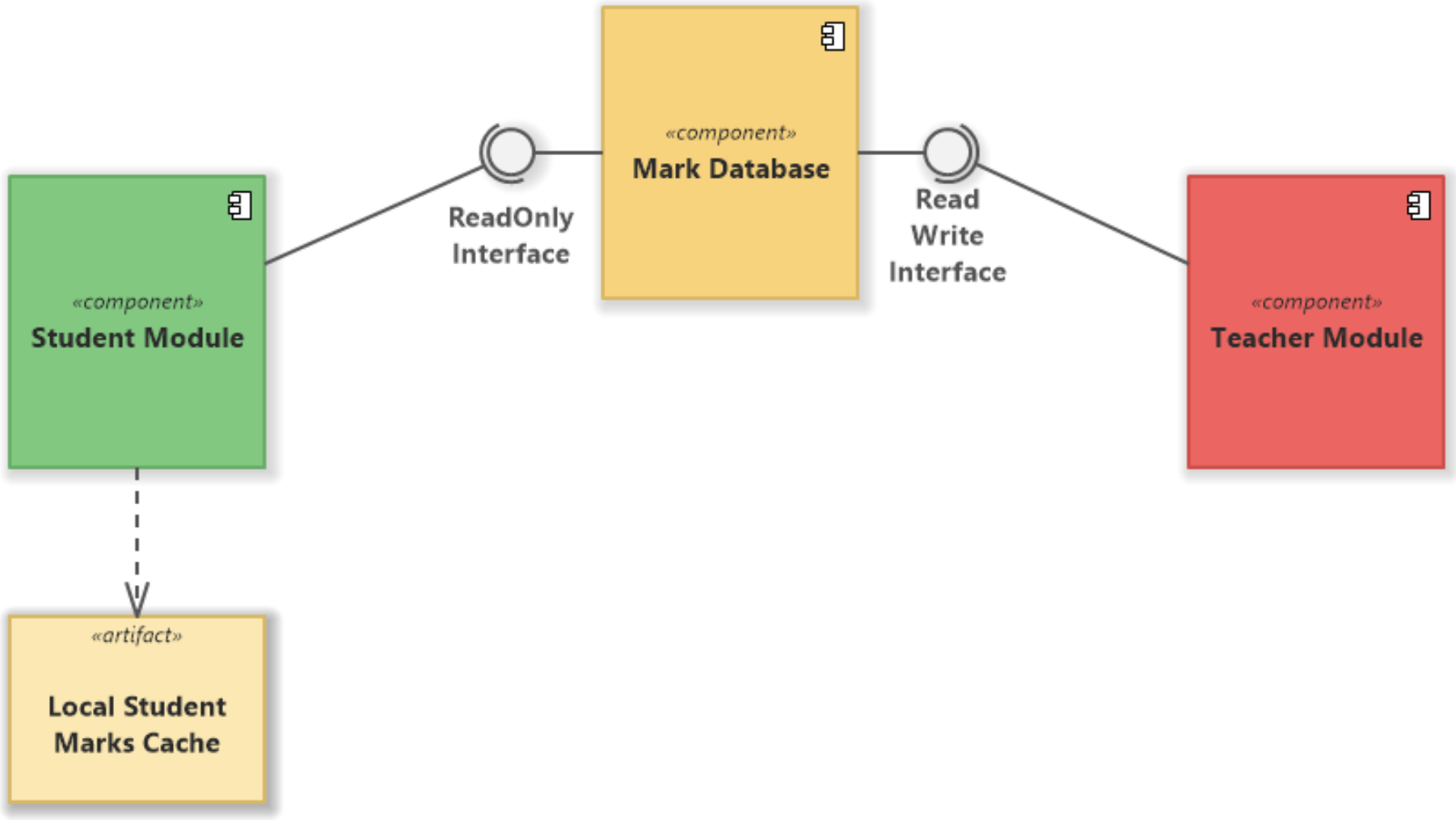


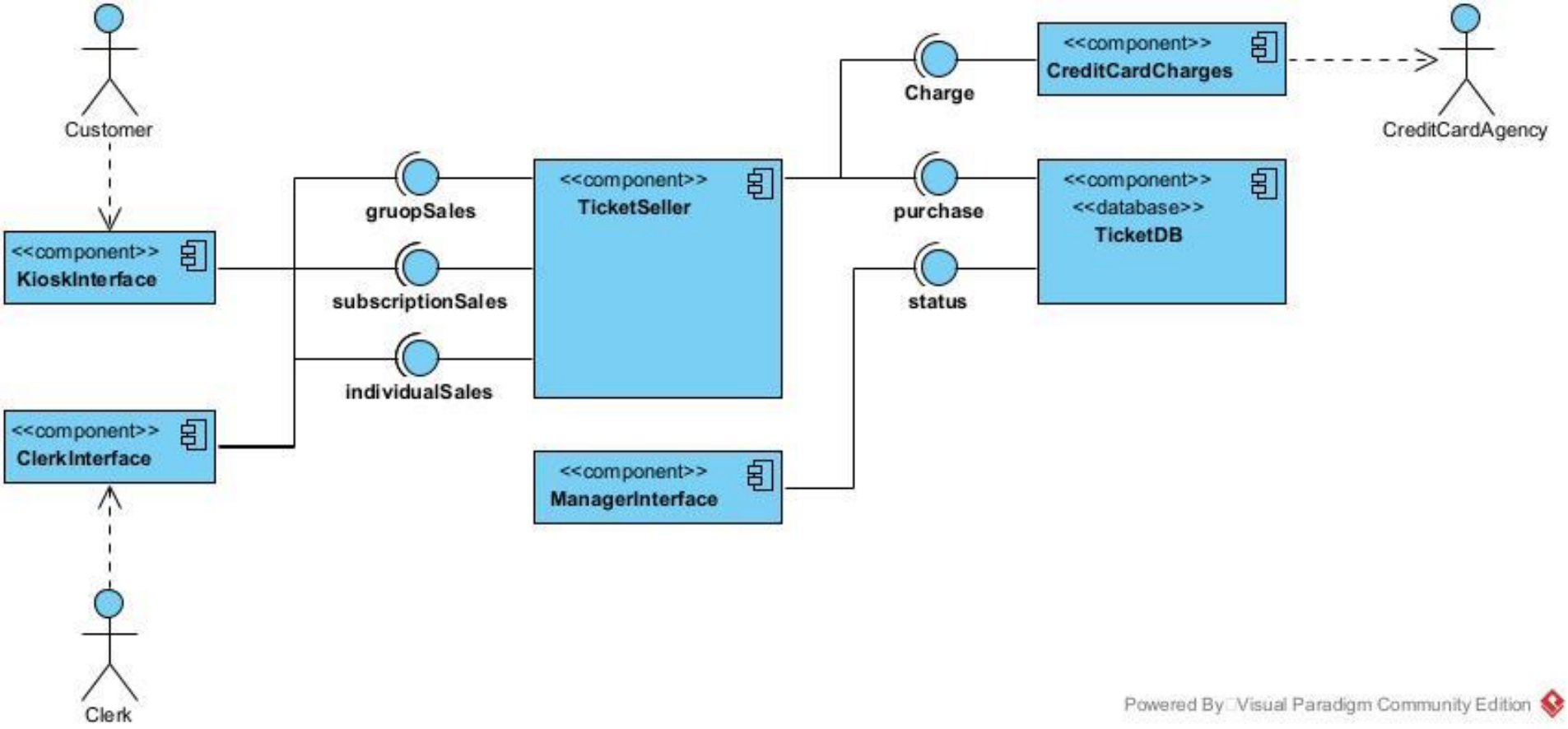
ATM Component Diagram

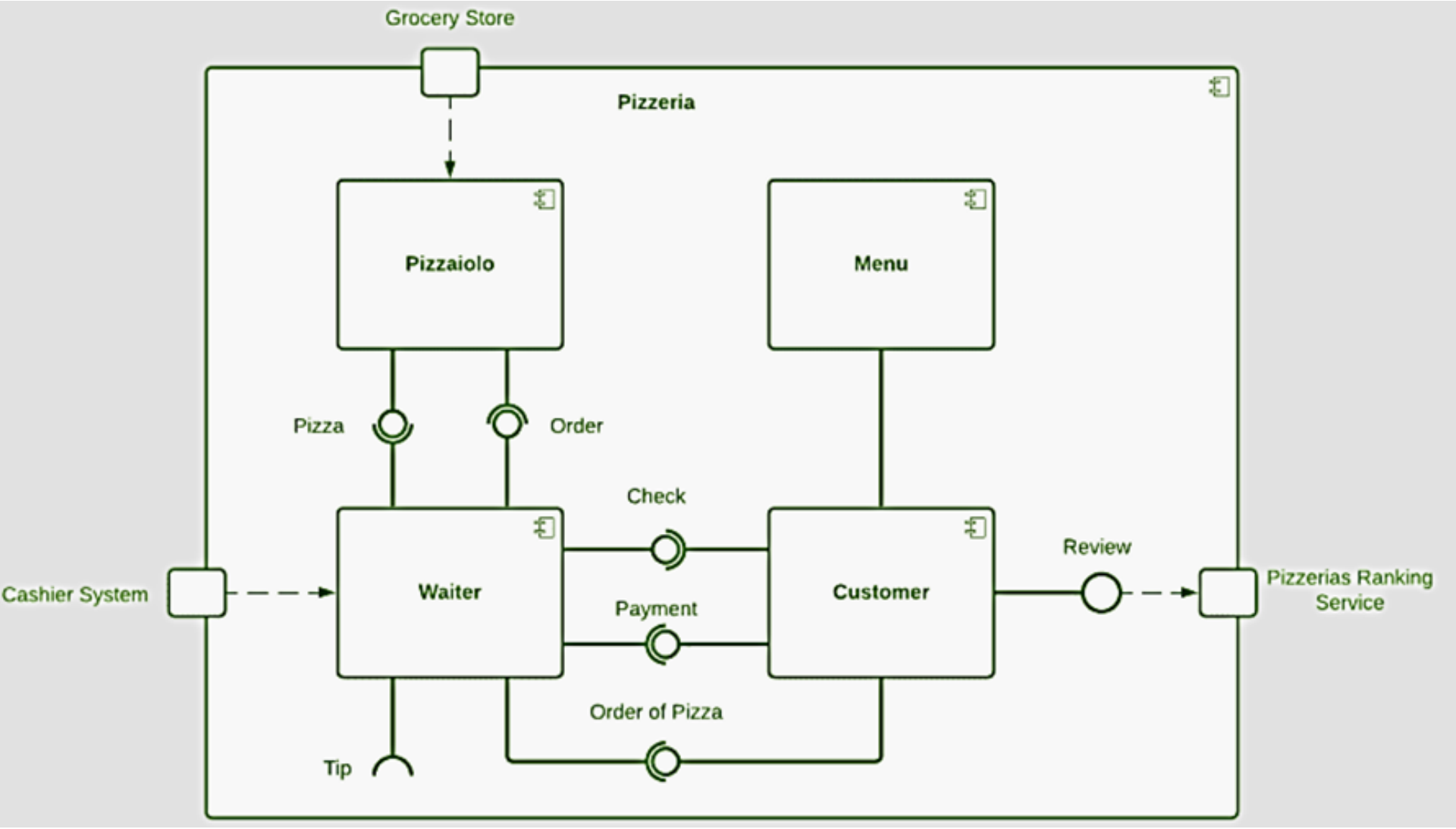


ATM Component Diagram



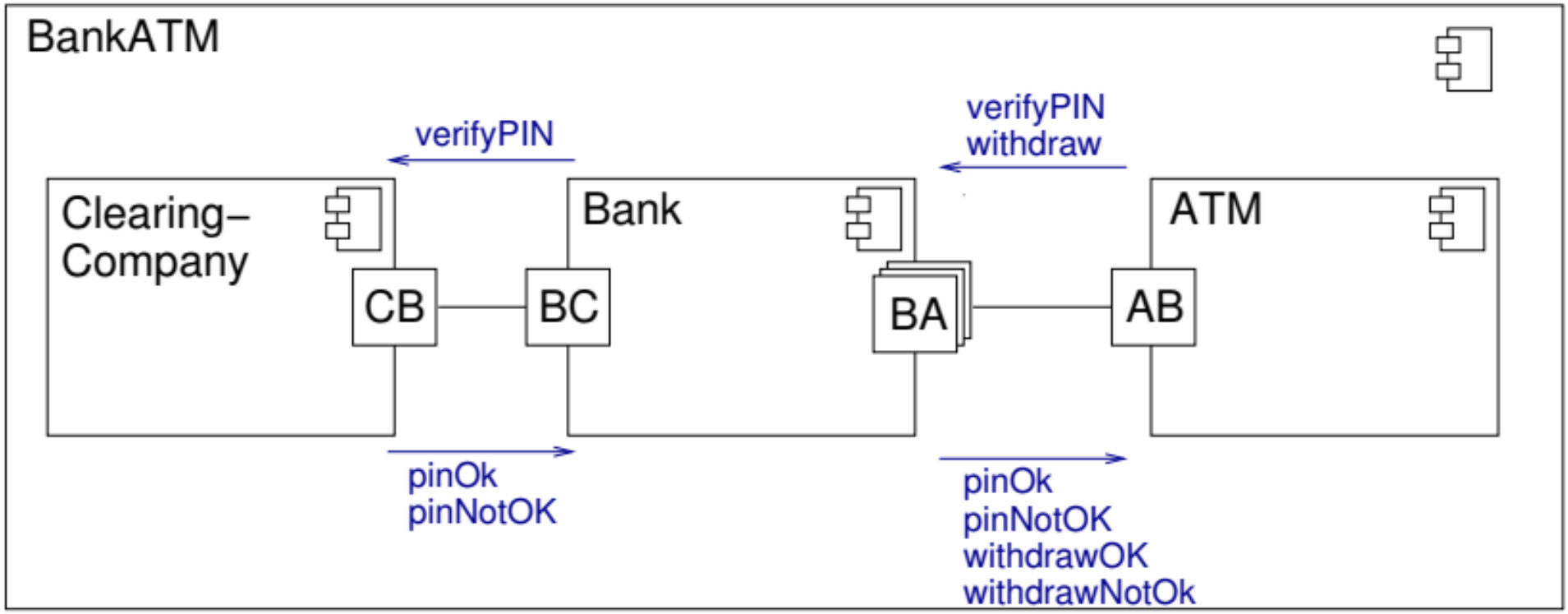

«component»
Mark Database
ReadOnly
Interface
Read
Write
Interface
«component»
Student Module
«component»
Teacher Module
«artifact»
Local Student
Marks Cache


Customer
<<component>>
KioskInterface
<<component>>
ClerkInterface
Clerk
gruopSales
subscriptionSales
individualSales
<<component>>
TicketSeller
<<component>>
ManagerInterface
Charge
purchase
status
<<component>>
CreditCardCharges
<<component>>
<<database>>
TicketDB
CreditCardAgency
Powered By Visual Paradigm Community Edition

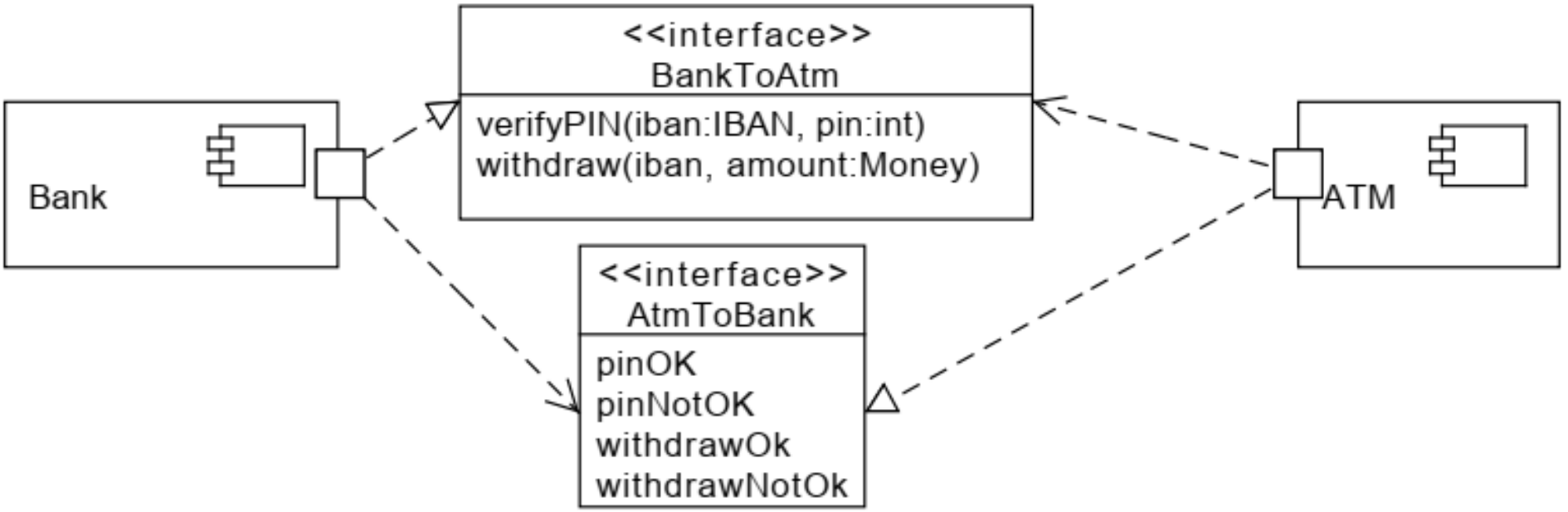


https://medium.com/swlh/uml-diagrams-the-pizzeria-59a30af5fd6d

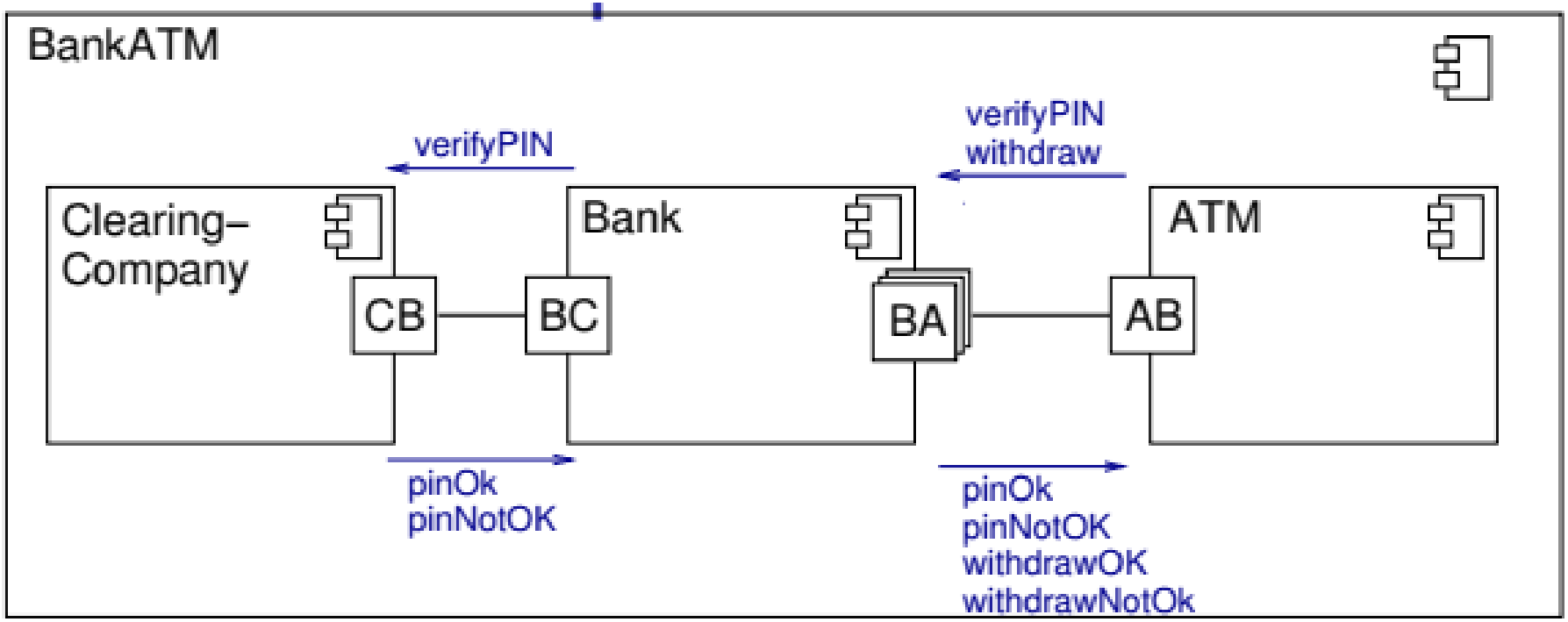
Example Bank–ATM Component Diagram



Port BA


<<interface>>
BankToAtm
verifyPIN(iban:IBAN, pin:int)
withdraw(iban, amount:Money)
Bank
ATM
<<interface>>
AtmToBank
pinOK
pinNotOK
withdrawOk
withdrawNotOk

Asynchronous Calls implementation



```
public class Atm implements AtmToBank {
    private BankToAtm bank;
    private int amountToWithdraw;
    // From the user interface
    public void enterPinAndAmount(String pin,
int amount) {
        amountToWithdraw = amount;
        new Thread(() ->
{bank.verifyPin(pin)}).start();
    }
```

```
    // From the bank
    public void pinOk() {
        new Thread(() ->
            {bank.withdraw(amount)}).start();
    }
    public void pinNotOk() { throw new
                        PinNotOkException(); }
    public void withdrawOk() {
        self changed();
        self notfiyObservers("withdraw ok");
    }
    public void withdrawNotOk() {..}
}
```

Lambda expressions in Java

```
public class Test {
    public static void main(String[] args)
    {
  // Creating Lambda expression for run()
  // method in functional interface "Runnable"
        Runnable myThread = () ->
        {
// Used to set custom name to the current thread
            Thread.currentThread().setName("my
Thread");
```

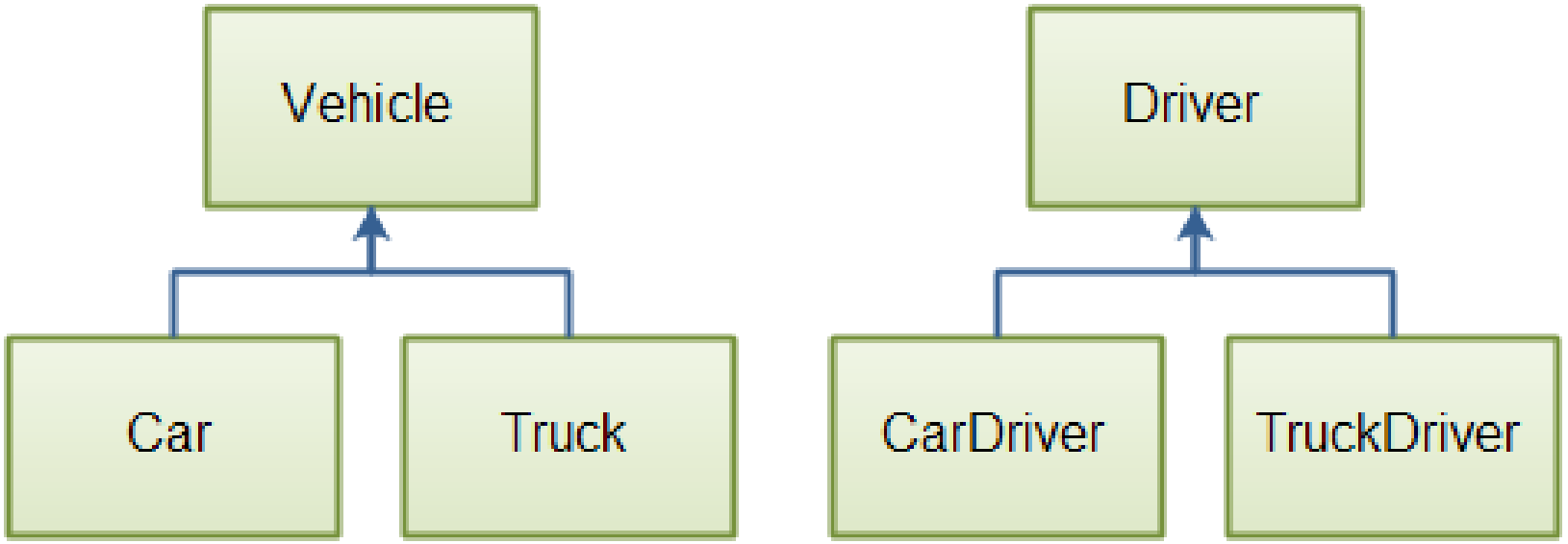
Lambda expressions in Java

```
        System.out.println(
                Thread.currentThread().getName()
                        + " is running");
                };

// Instantiating Thread class by passing
// Runnable reference to Thread constructor
                Thread t1 = new Thread(myThread);
                 // Starting the thread
                t1.start();
        }
}
```

Interface – default methods

```
public interface Storable {
    public void store();
}
```

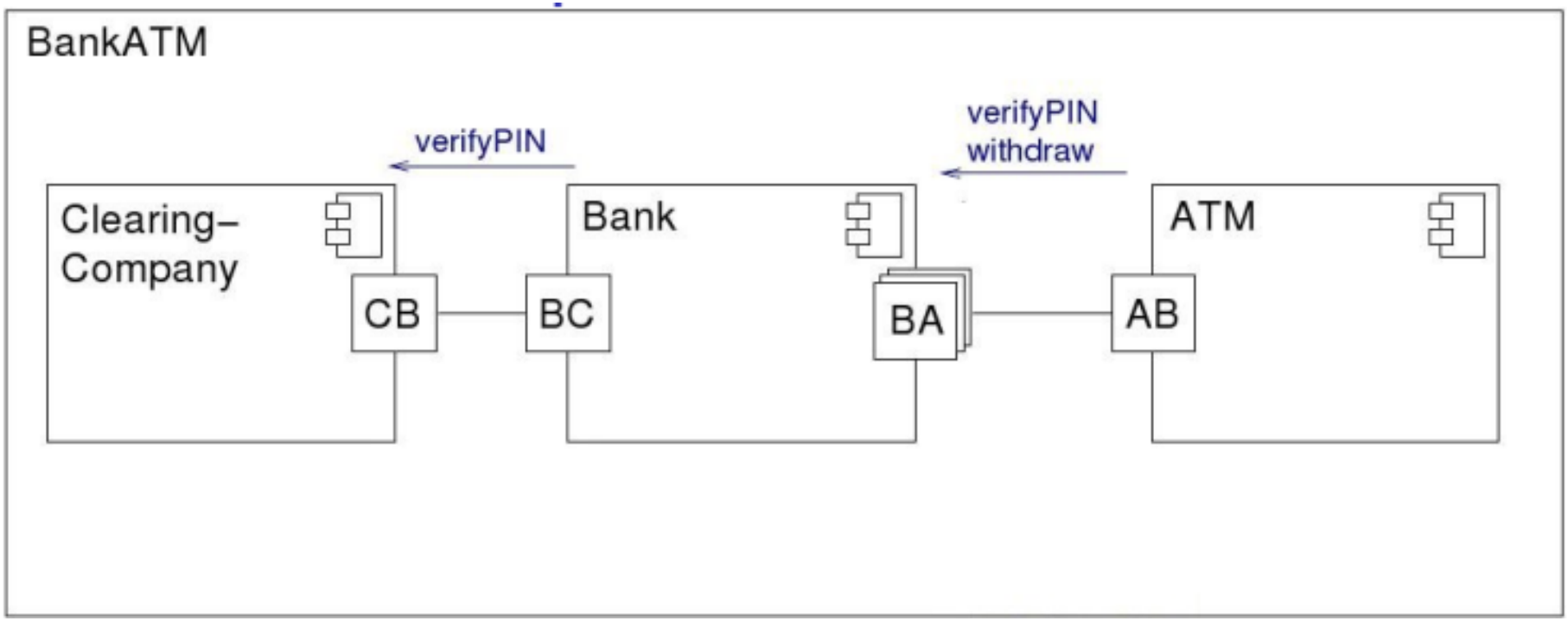


Car implements Storable

```
Car car = new Car();
Storable storable = (Storable) car;
storable.store();
```

- Pin is passed from class ATM to the function verifyPin(pin) of interface
- verifyPin() may be a default method of interface which can be overridden

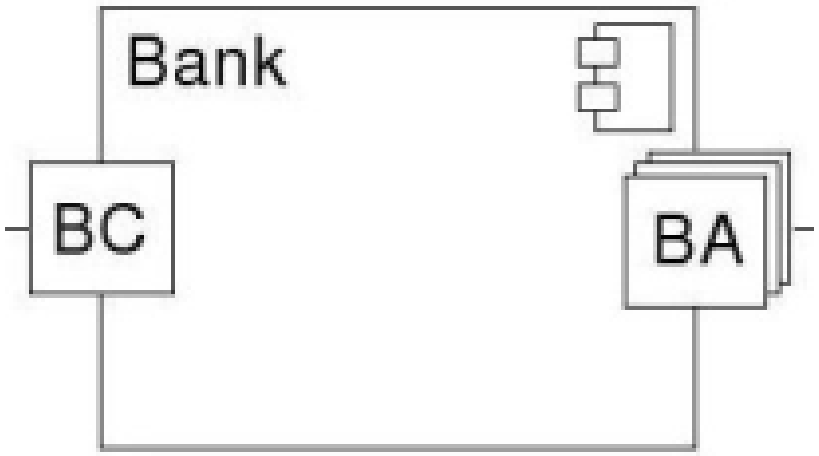
Synchronous Call implementation



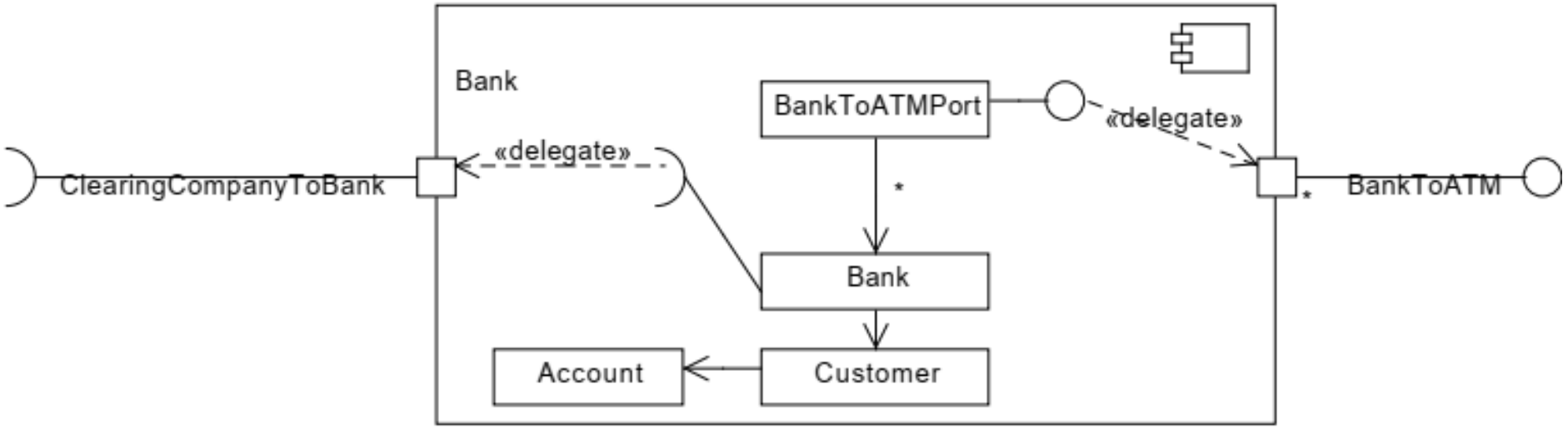
```
public class Atm {
private BankToAtmSync bank;
// From the user interface
public void enterPin(String pin, int amount)
{
      boolean pinOk = bank.verifyPin(pin);
      boolean withdrawOk = false;
      if (pinOk) {
         withdrawOk = bank.withdraw(amount);
      }
```

```
        else {
            throw new PinNotOkException("...");
        }
        if (withdrawOk) {
            self changed();
            self notifyObservers("withdraw ok");
        }
    }
}
```

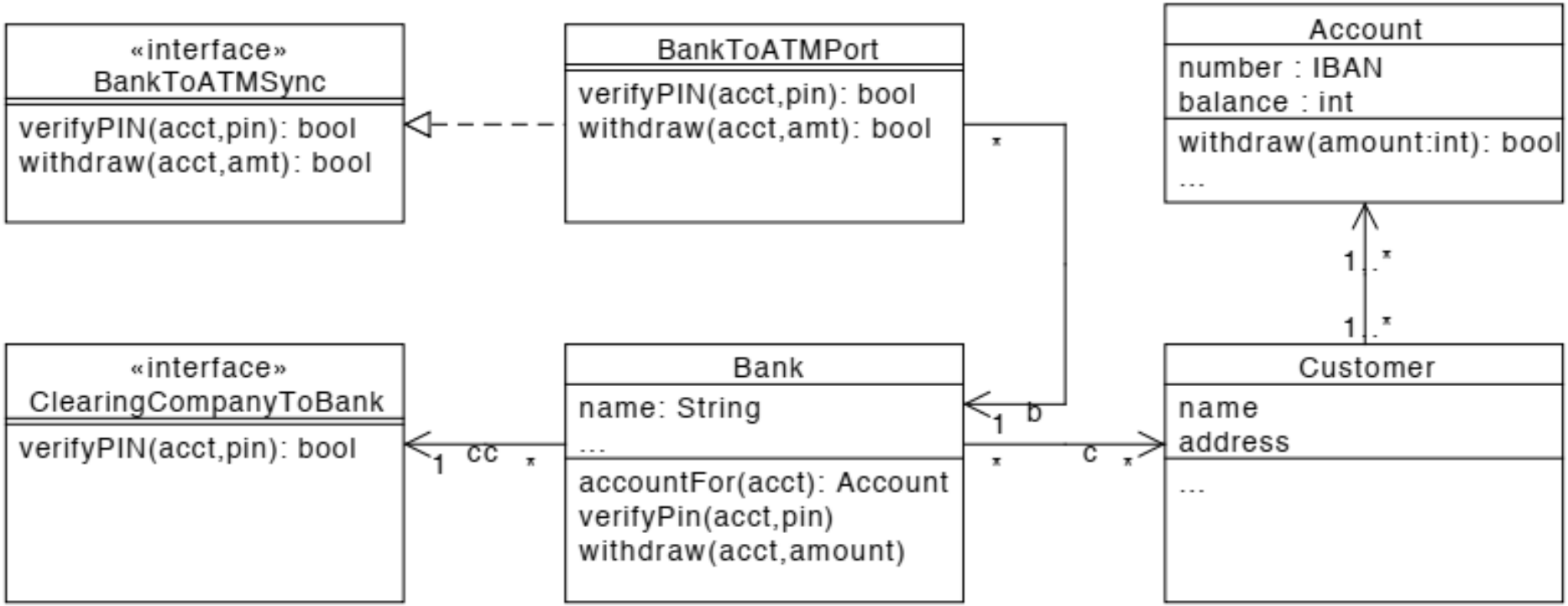
Bank component seen from outside

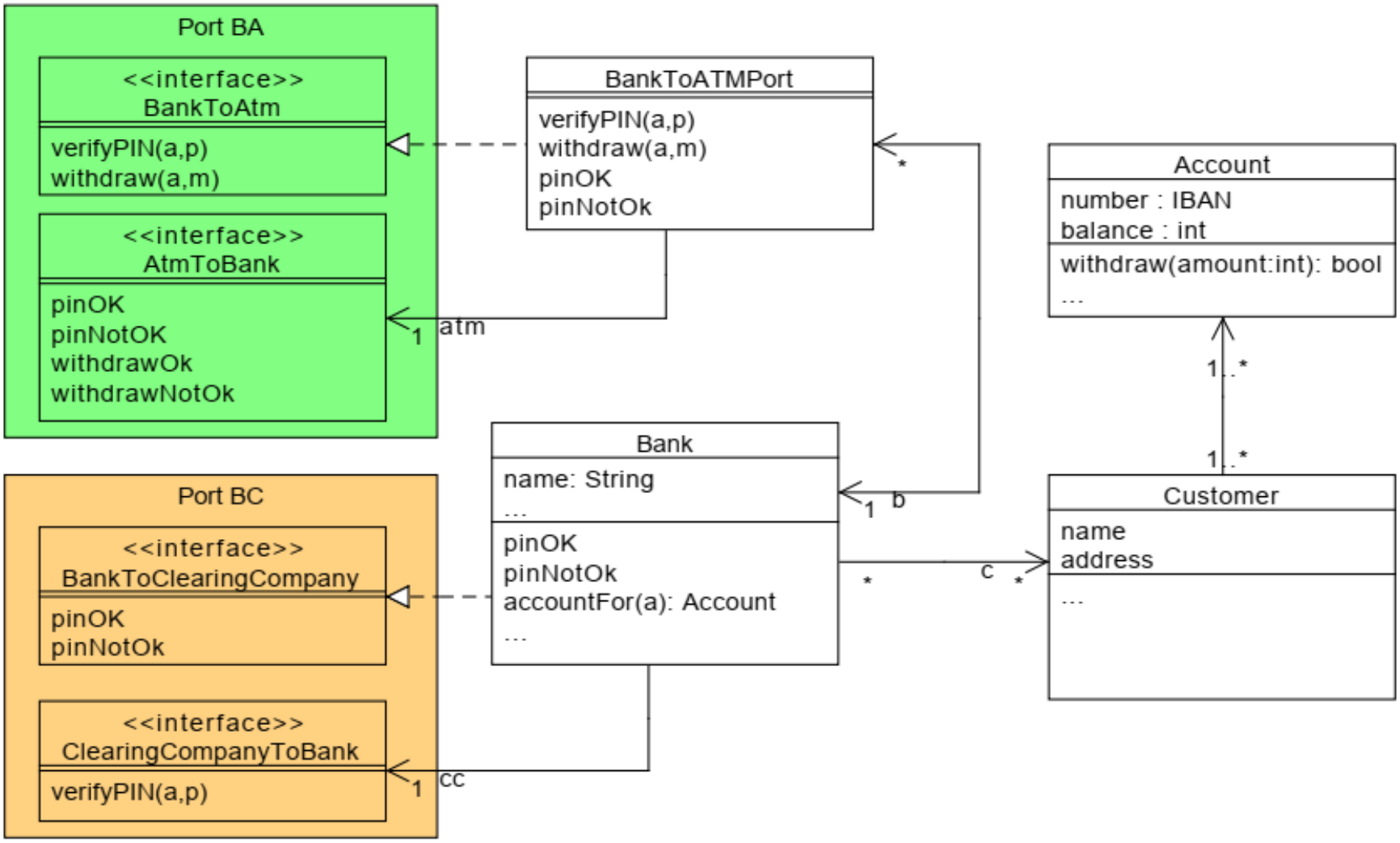


Bank component seen from inside



Detailed Class Diagram for the Bank Component




Port BA
<<interface>>
BankToAtm
verifyPIN(a,p)
withdraw(a,m)
<<interface>>
AtmToBank
pinOK
pinNotOK
withdrawOk
withdrawNotOk
BankToATMPort
verifyPIN(a,p)
withdraw(a,m)
pinOK
pinNotOk
atm
1
*
Account
number : IBAN
balance : int
withdraw(amount:int): bool
...
1..*
1..*
Bank
name: String
...
pinOK
pinNotOk
accountFor(a): Account
...
1
b
*
c
*
Customer
name
address
...
Port BC
<<interface>>
BankToClearingCompany
pinOK
pinNotOk
<<interface>>
ClearingCompanyToBank
verifyPIN(a,p)
1
cc

Rules for implementing components

- Provided interfaces must to be implemented by some class
- Required interfaces must used by one or several classes
- No access to and from classes of other components
- Use packages to indicate classes belonging to a component

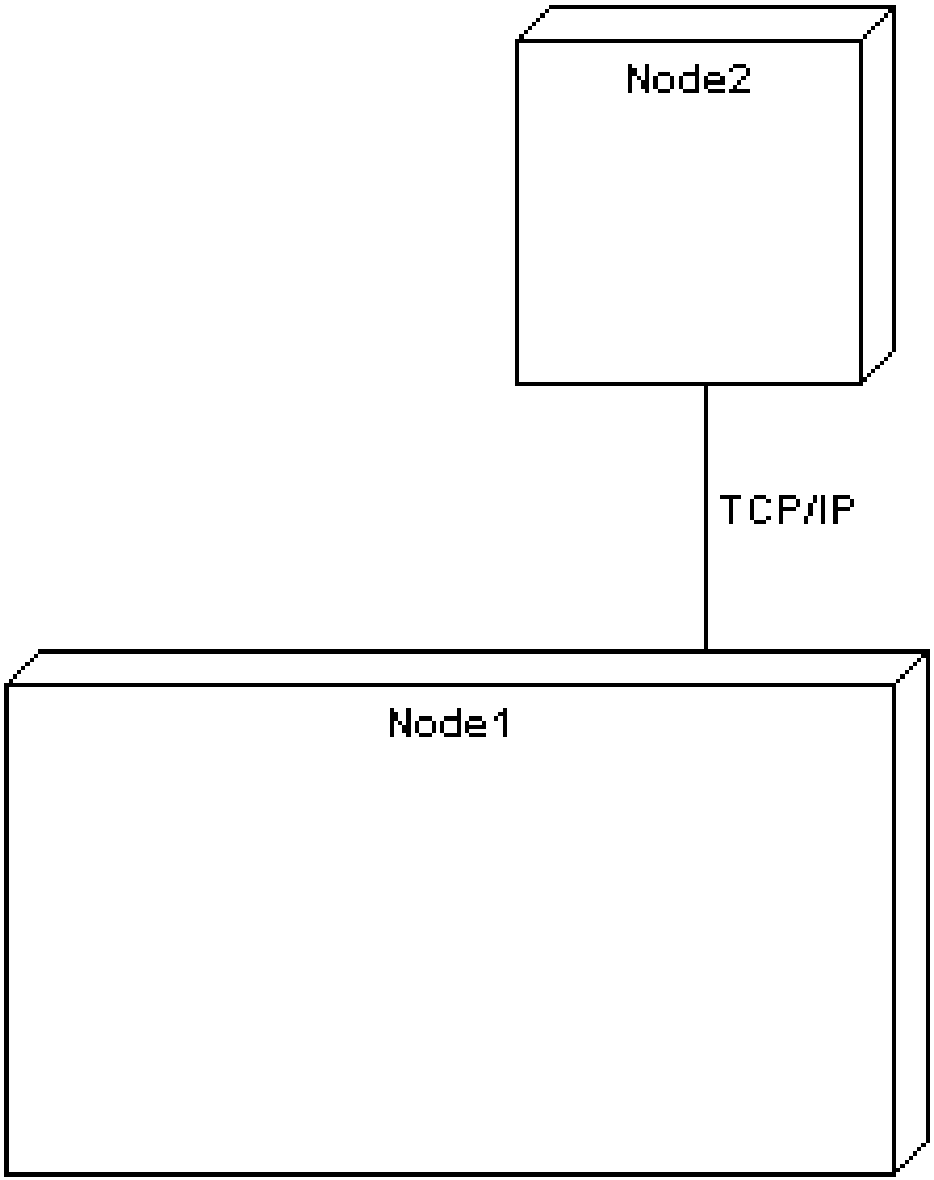
DEPLOYMENT DIAGRAMS

- There is a strong link between components diagrams and deployment diagrams

- Deployment diagrams
  - Show the physical relationship between hardware and software in a system
  - Hardware elements:
    - Computers (clients, servers)
    - Embedded processors
    - Devices (sensors, peripherals)
  - Are used to show the nodes where software components reside in the run-time system

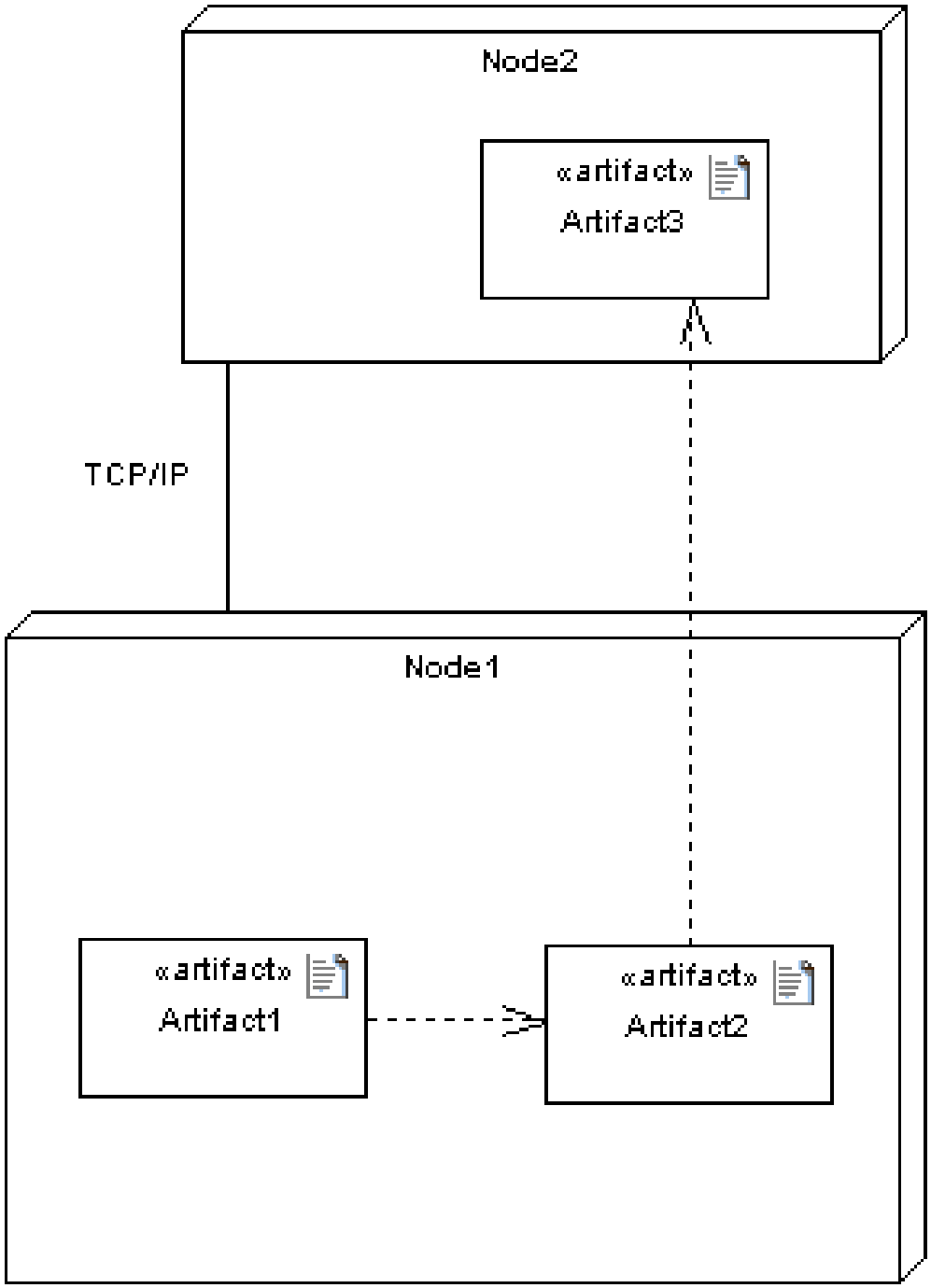
DEPLOYMENT DIAGRAMS

- Deployment diagram
  - Contains nodes and connections
  - A node usually represent a piece of hardware in the system
  - A connection depicts the communication path used by the hardware to communicate
  - Usually indicates the method such as TCP/IP


Node2
TCP/IP
Node1

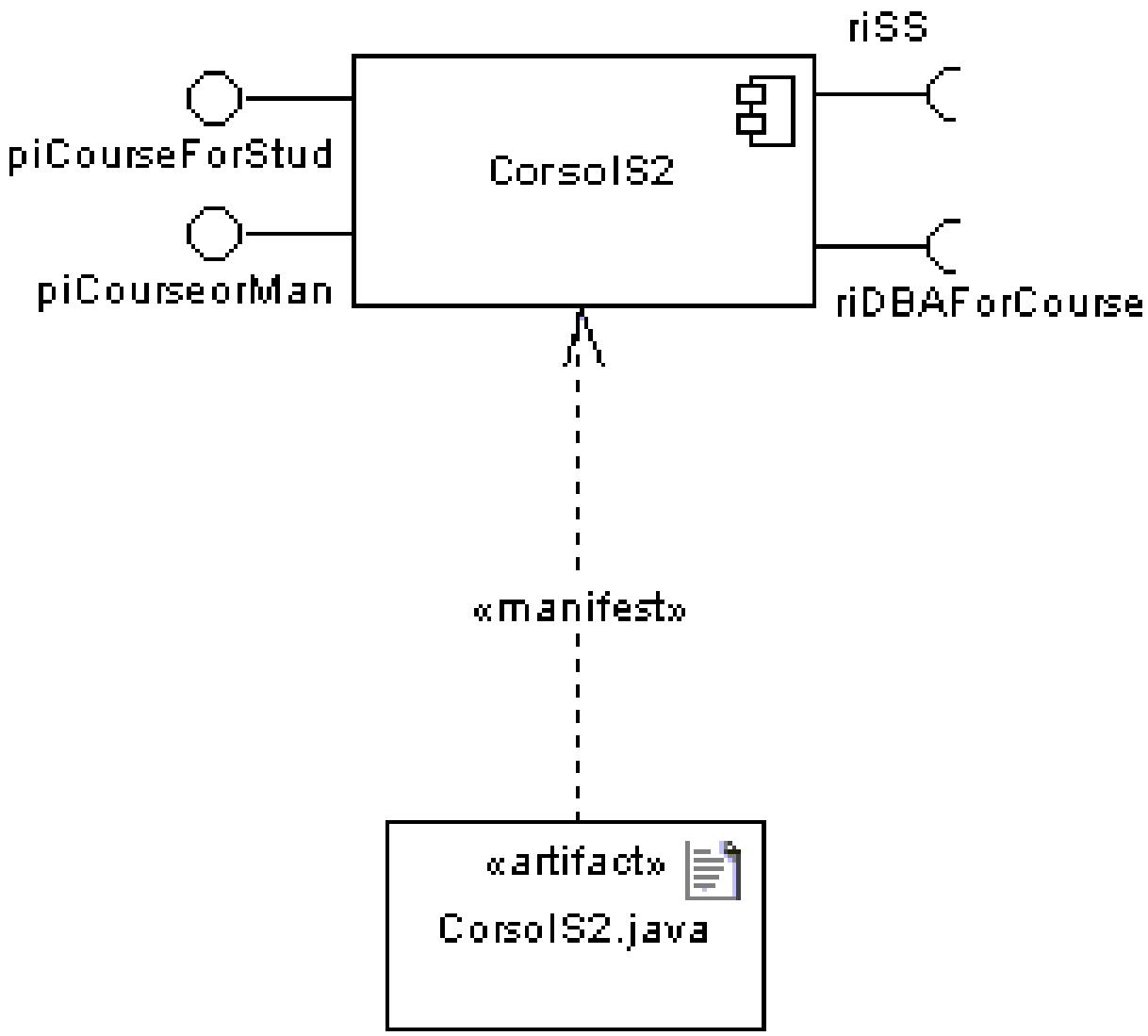
DEPLOYMENT DIAGRAMS

- Deployment diagrams contain artifact
- An artifact
  - Is the specification of a physical piece of information
  - Ex: source files, binary executable files, table in a database system,….
  - An artifact defined by the user represents a concrete element in the physical world

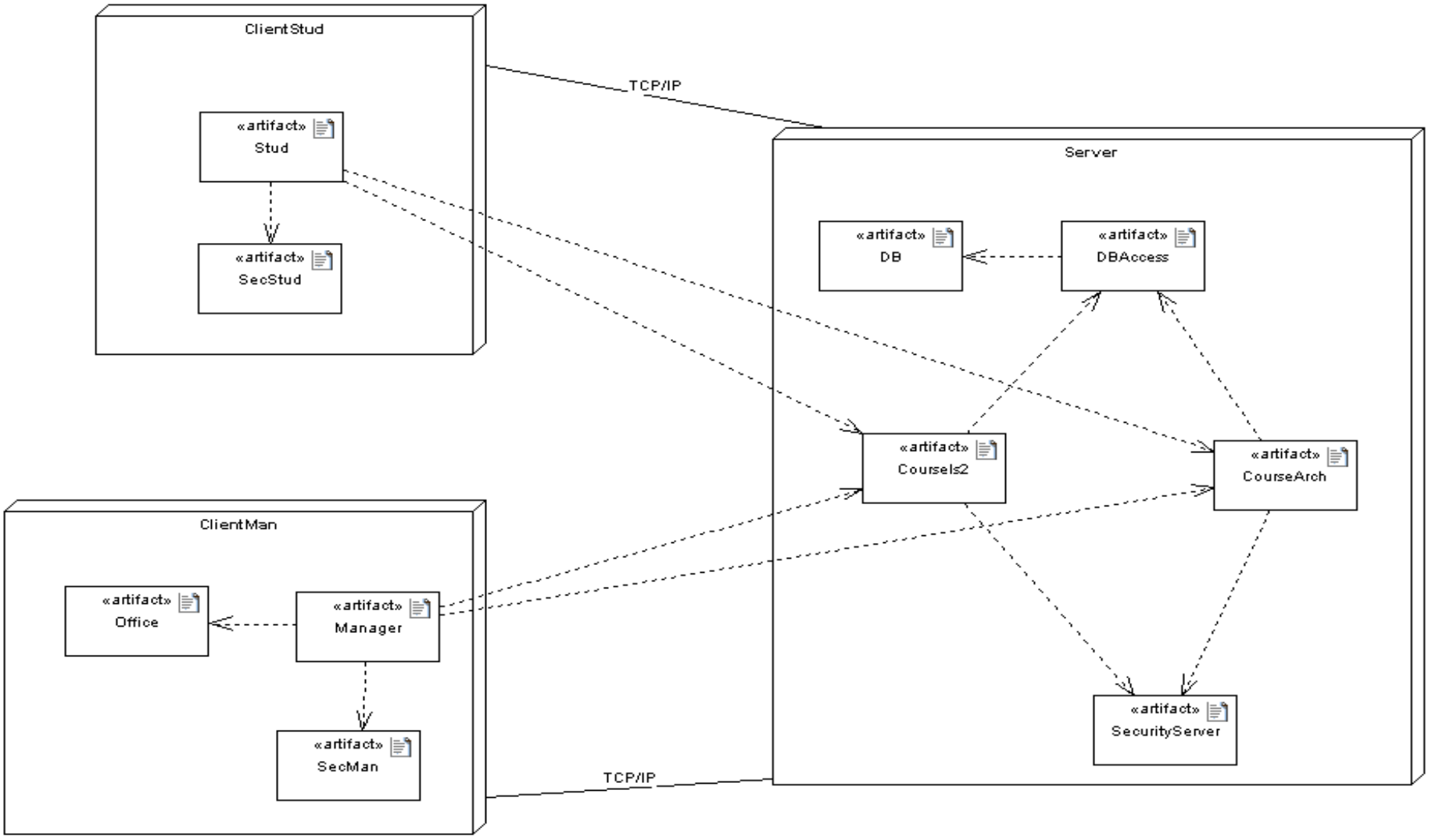

Node2
«artifact»
Artifact3
TCP/IP
Node1
«artifact»
Artifact1
«artifact»
Artifact2

DEPLOYMENT DIAGRAMS

- An artifact manifest one or more model elements
- A <<manifestation>> is the concrete physical of one or more model elements by an artifact
- This model element often is a component
- A manifestation is notated as a dashed line with an open arrow-head labeled with the keyword <<manifest>>


riSS
piCourseForStud
CorsoIS2
piCourseorMan
riDBAForCourse
«manifest»
«artifact»
CorsoIS2.java

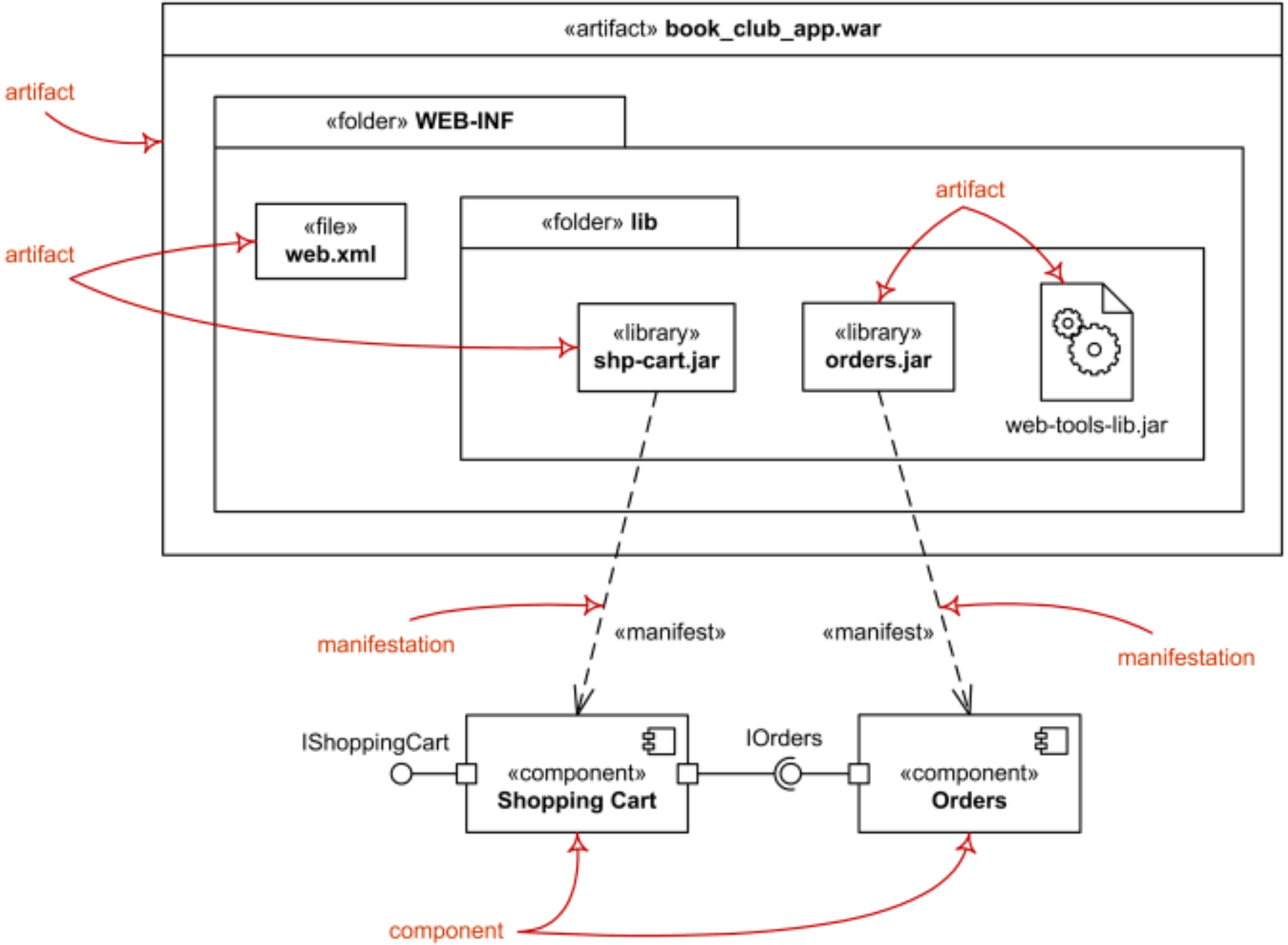
DEPLOYMENT DIAGRAMS



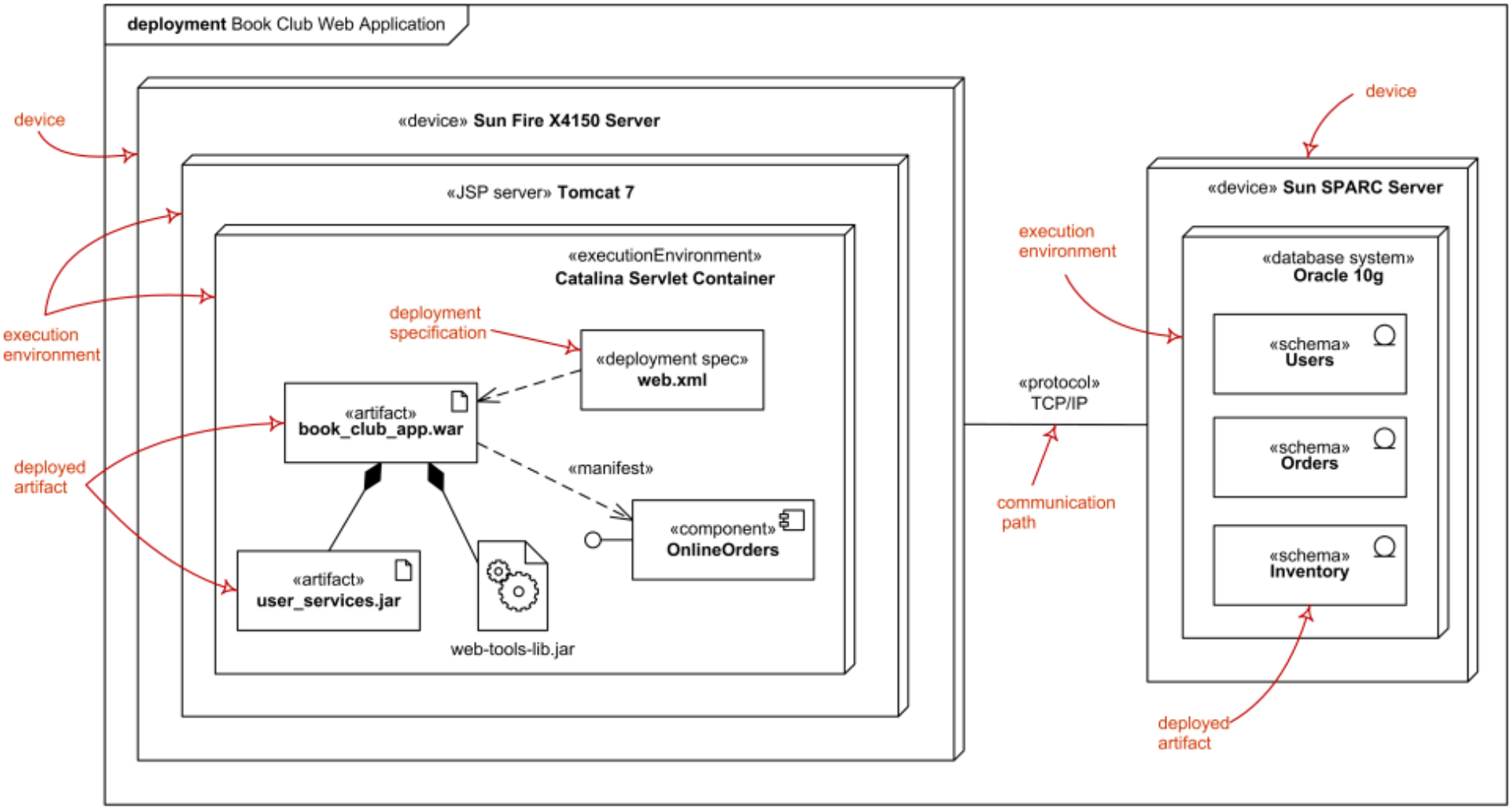
DEPLOYMENT DIAGRAMS

- Some common types of deployment diagrams are:
  - Implementation (manifestation) of components by artifacts,
  - Specification level deployment diagram,
  - Instance level deployment diagram,
  - Network architecture of the system.

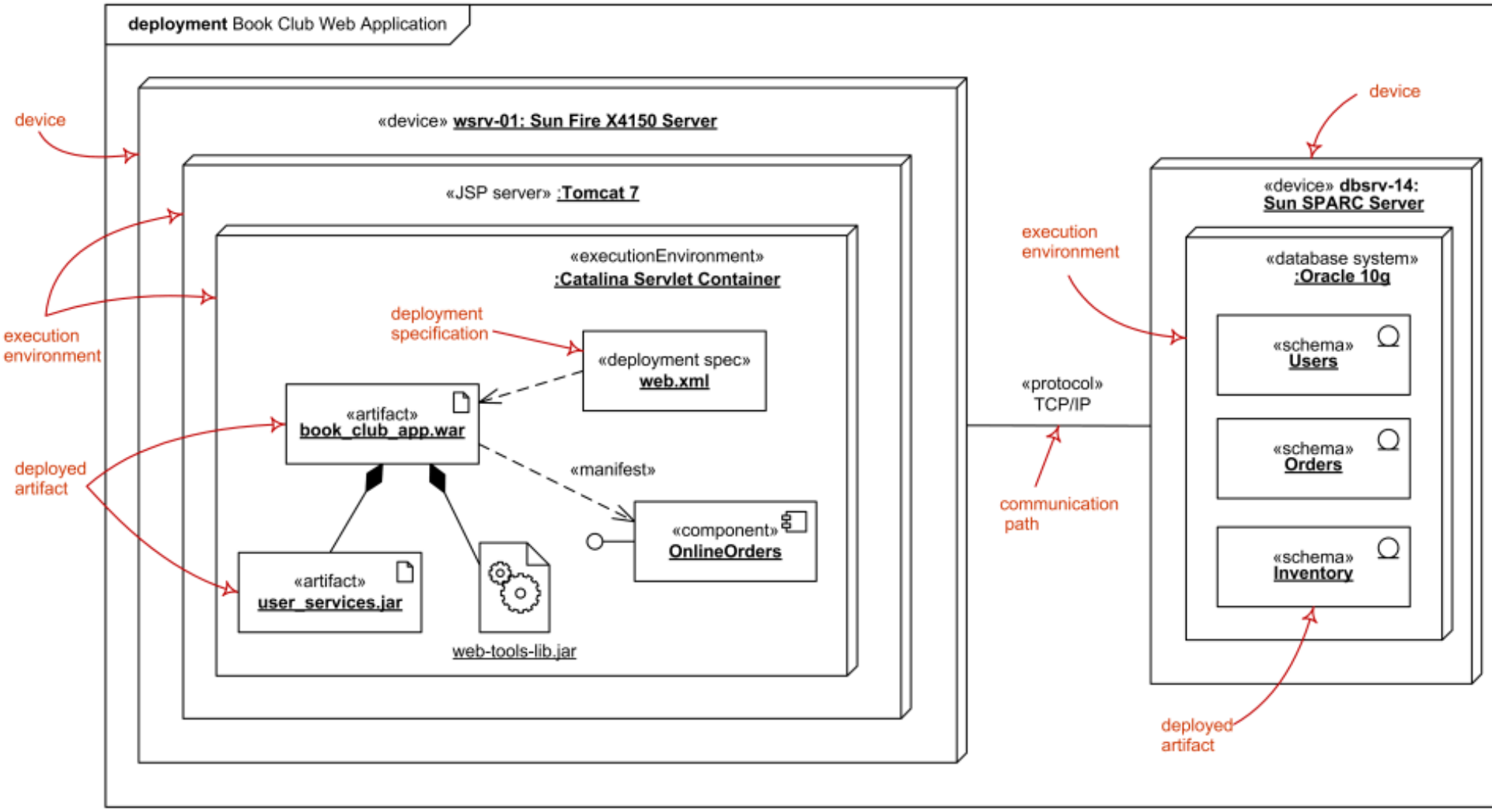
Manifestation of Components by Artifacts



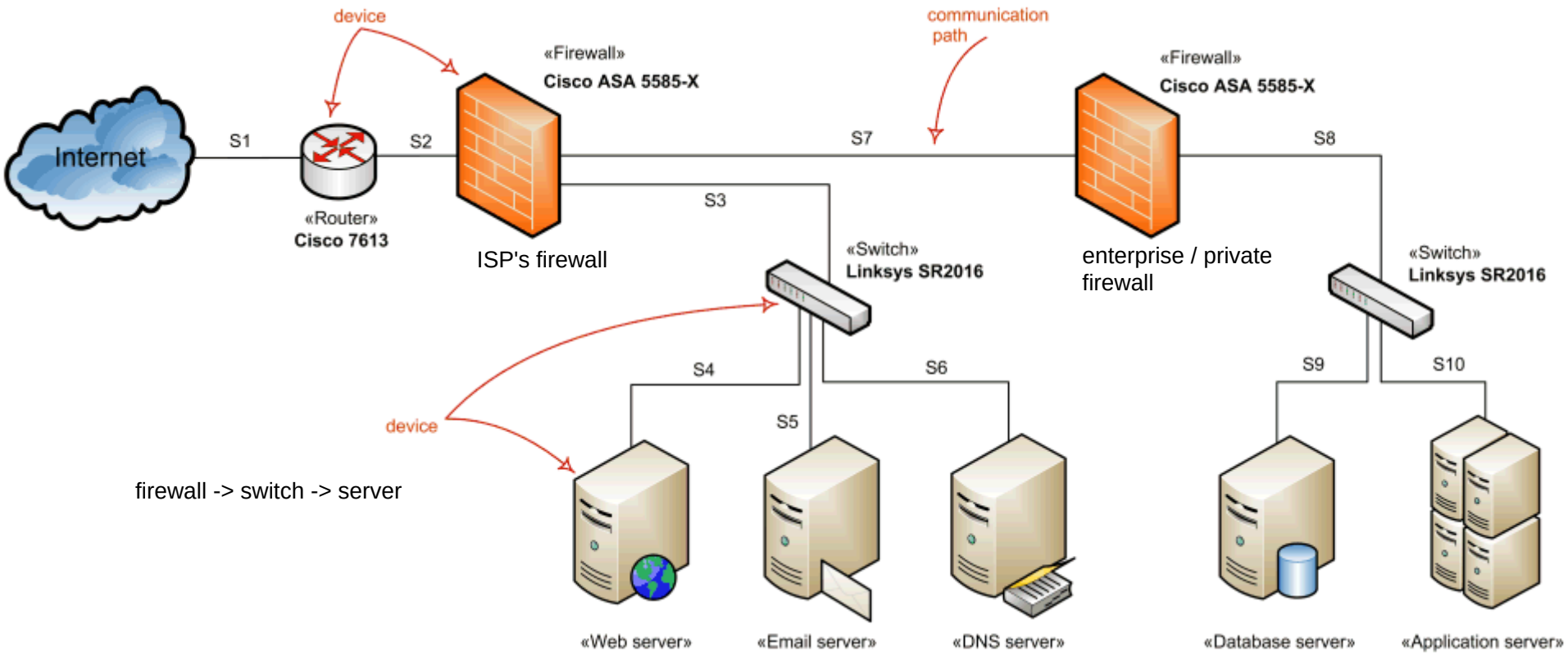
Specification Level Deployment Diagram



Instance Level Deployment Diagram



Specification Level Network Architecture



firewall -> switch -> server