

CSIT314 Software Development Methodologies



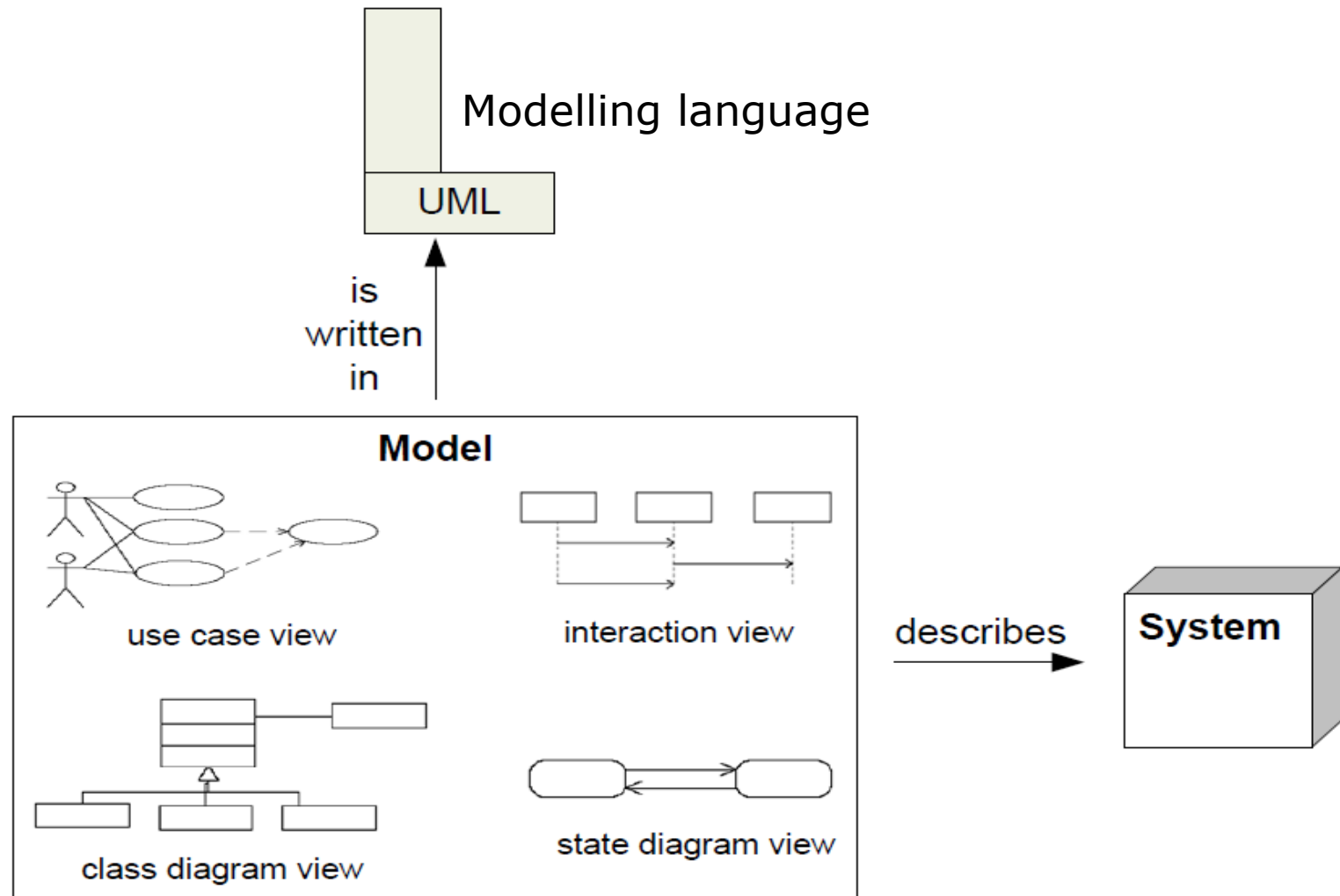
Advanced Unified Modelling Language

Modelling Language

- ❑ Two important components of a software development methodology:
 - Process (discussed in the previous lecture)
 - Modelling language

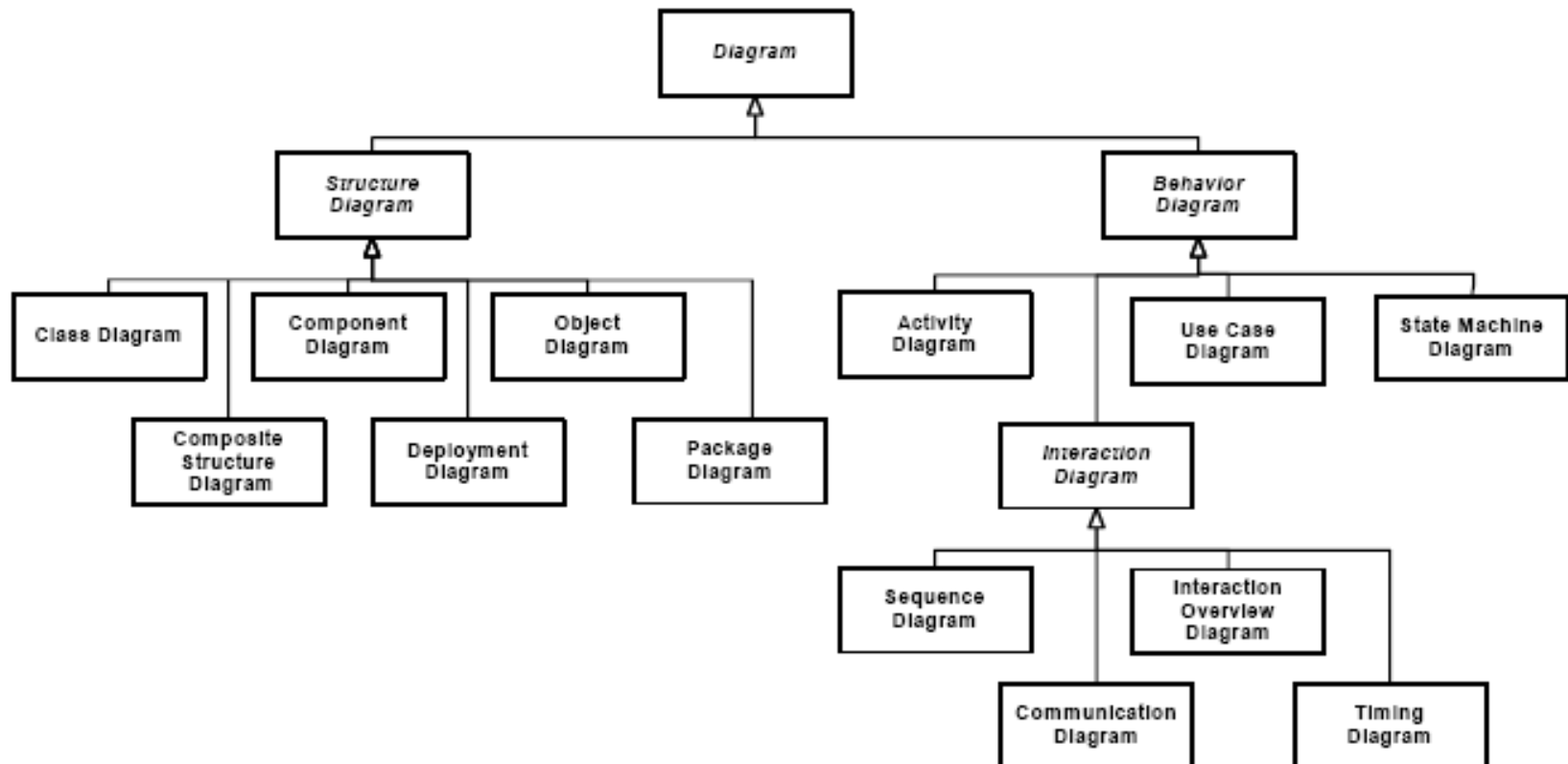
- ❑ What is a model?
 - A model describes a system using a **well defined (modelling) language**, which has clear syntax and semantics suitable for both human and computer interpretation.

Modelling Language (cont.)

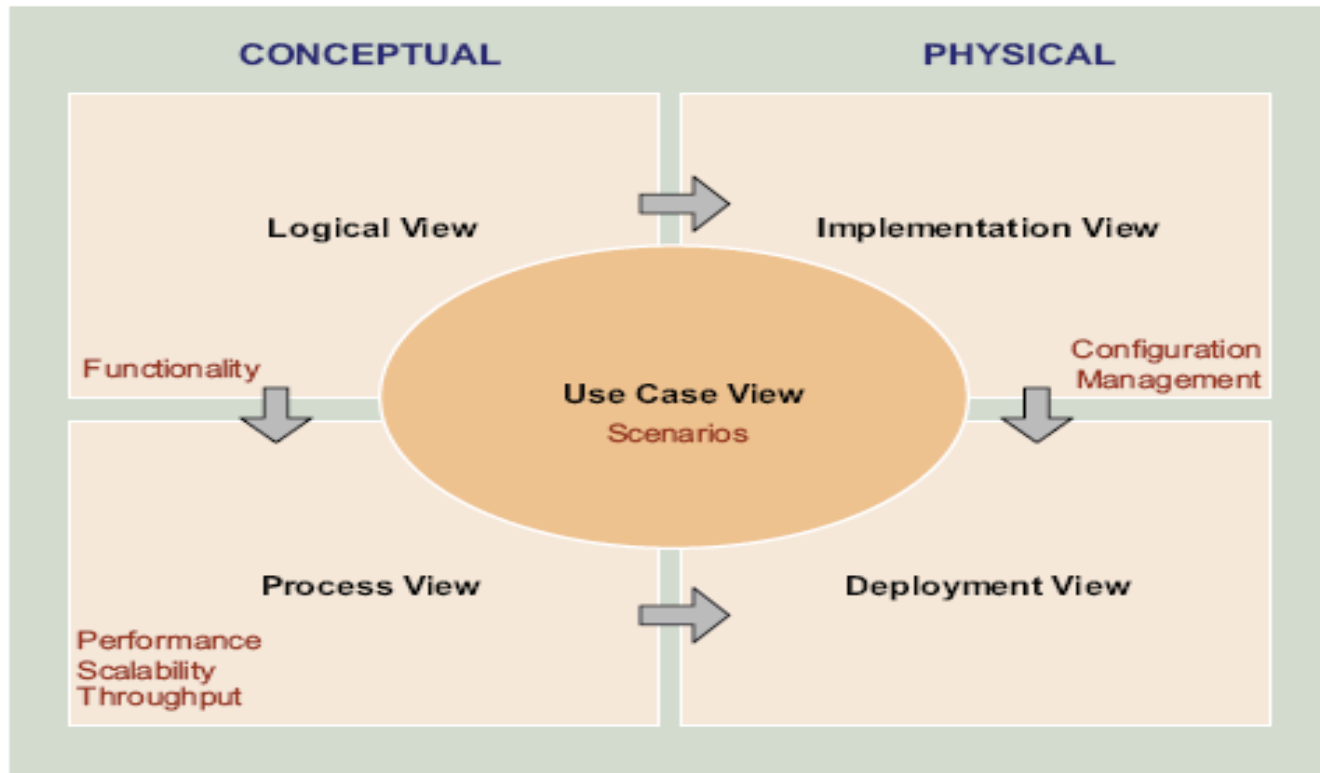


UML-2 diagrams

❑ OMG's classification of UML-2 diagrams:



“4+1” View



This version of “4+1” view from First Software Services

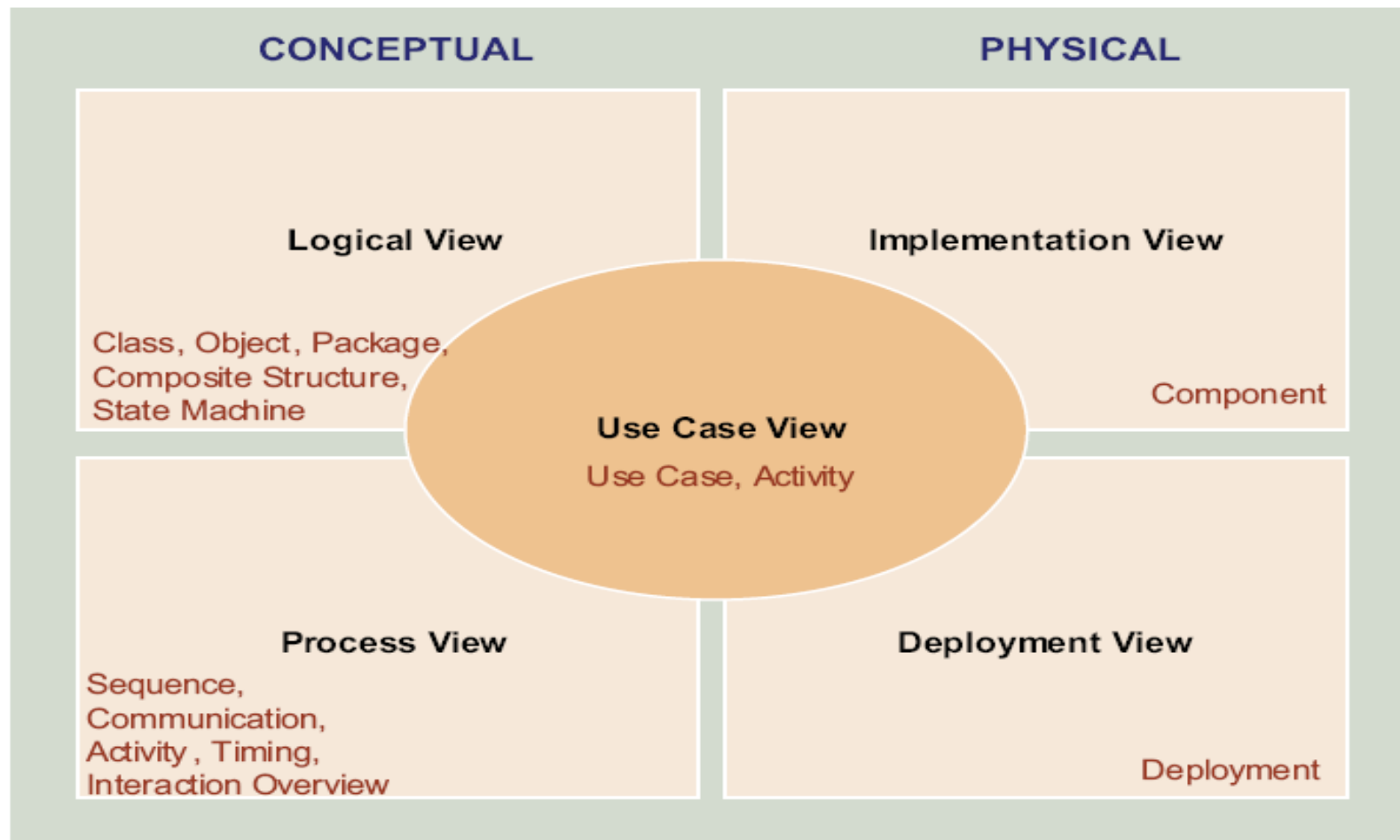
More information on the “4+1” view is available at:

<http://www.win.tue.nl/~mchaudro/sa2004/Kruchten4+1.pdf>

Conceptual and physical

- ❑ The views are used to describe the system from the viewpoint of **different stakeholders**, such as end-users, developers and project managers.
- ❑ Logical View and Process View
 - conceptual level
 - used from analysis to design.
 - The logical view is concerned with the **functionality** that the system provides to end-users
 - The process view deals with the **dynamic aspects** of the system, explains the system processes and how they communicate, and focuses on the **runtime** behavior of the system
- ❑ Implementation View and Deployment View
 - physical level
 - represent the application components as built and deployed.

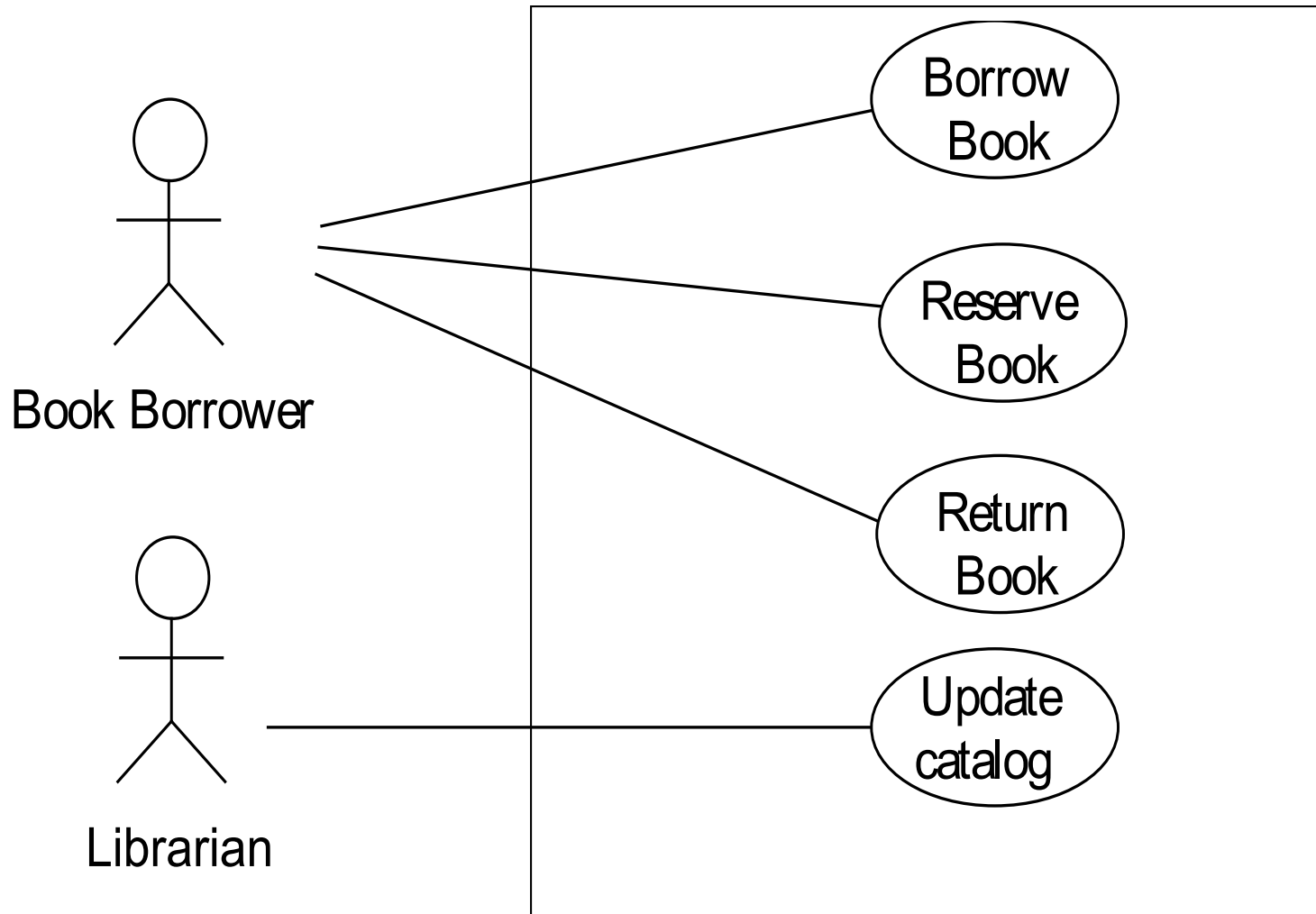
Diagrams and views



Review: Use case view

- Use Cases model the *functional requirements* of a system as one or more *Actors* interacting with one or more *Use Cases*
 - An Actor is a **user** or **external system**
 - A Use Case is a typical interaction between an actor and the system
- We have functional requirements, so why do we need Use Cases?
 - Use Cases specify the behavior from *user's point of view*, not the system's point of view

Review: Use cases (cont.)

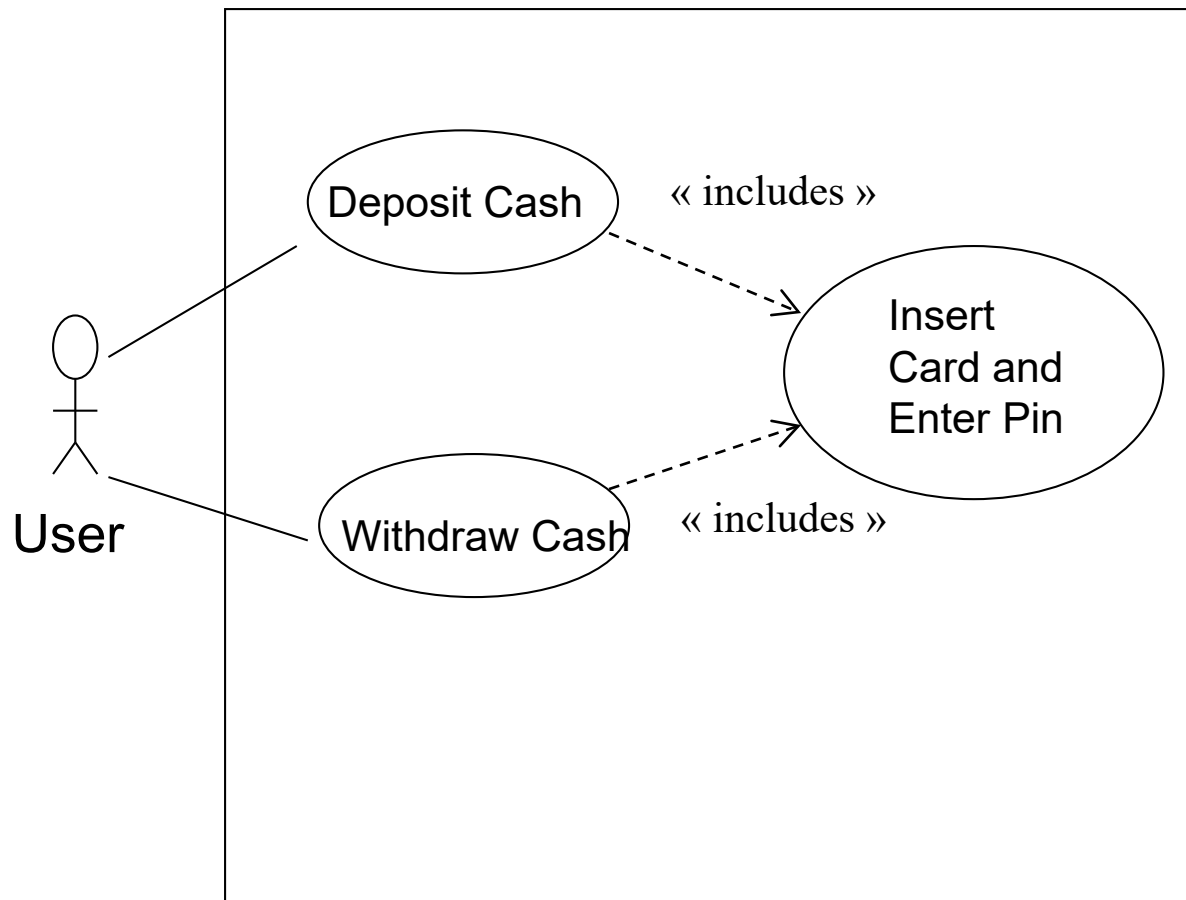


A boundary box indicating the scope of the system

Review: Use Case Relationship

Includes Example

▣ This can be drawn as:

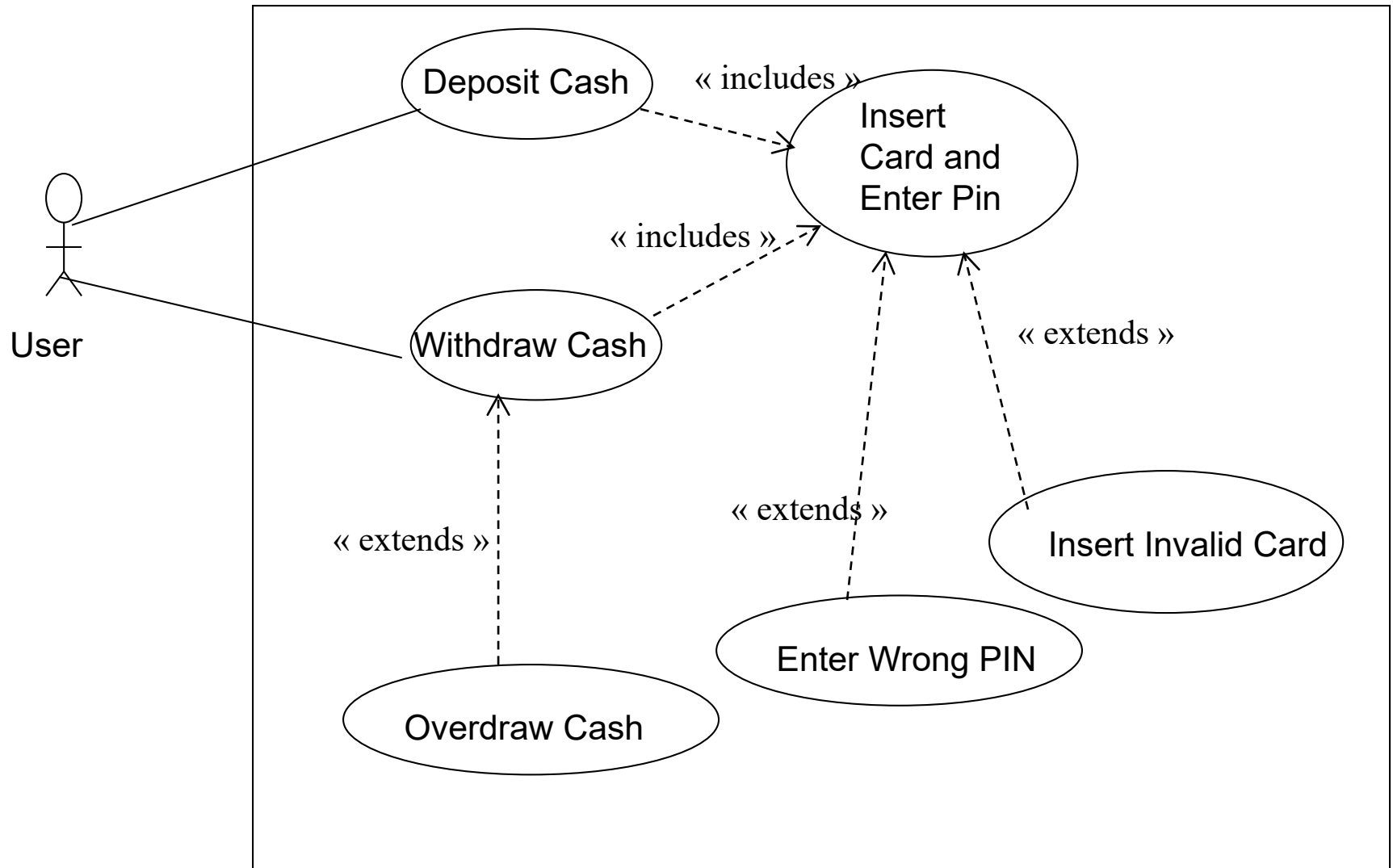


Review: Use Case Relationship

Extends Association

- *Extends* is good for modelling
 - Optional parts of Use Cases
 - Complex/Alternative courses of the same interaction
 - Parts of the interaction that only occur under certain conditions

Extends Example (2)

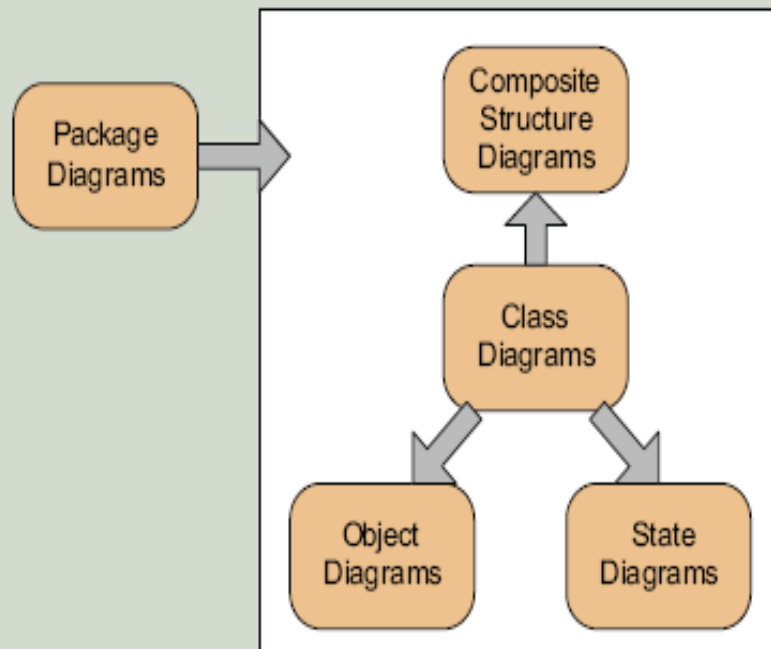


Review: use case description

Name: Borrow Book	ID: 2
Stakeholders and goals: Student – wants to borrow book from university library	
Description: A borrower wants to borrow a book and must present his/her ID card to the system.	
Actors: Borrower	
Trigger: The borrower in the library brings a book(s) to the librarian's desk	
Normal flow: <ol style="list-style-type: none">1. The borrower swipes his/her card through the card reading machine2. The system checks the ID of the card in the database3. The borrower passes the book through the bar-code scanner4. The system displays the date for returning the book5. Repeat steps 3-4 for each book to borrow6. End	
Sub-flows: None	
Alternative/Exceptional flows: 2a Invalid card: The <i>Invalid Card</i> use case is performed 2b Unpaid fine: The <i>Pay Fine</i> use case is performed 2c Book overdue: 2c1 – The system displays a message displaying the book overdue, the days overdue, and whether a fine must be paid	

Logical view

MODELING LOGICAL VIEW WITH UML2



1. Start with class diagrams to model the system
2. Use package diagrams to logically group diagrams

Optional use

3. Object diagrams when relationships between classes need to be explained through instances
4. State Charts when internal states of a specific class are to be explained
5. Composite Structures when parts of a class and relationships between parts are to be modeled

Logical View

Package

□ Easy –

- It really just models Java packages (or their equivalents in C#)
- It simply identifies a group of classes with a namespace
- Typically shows just the public classes

javax.servlet



HttpServlet



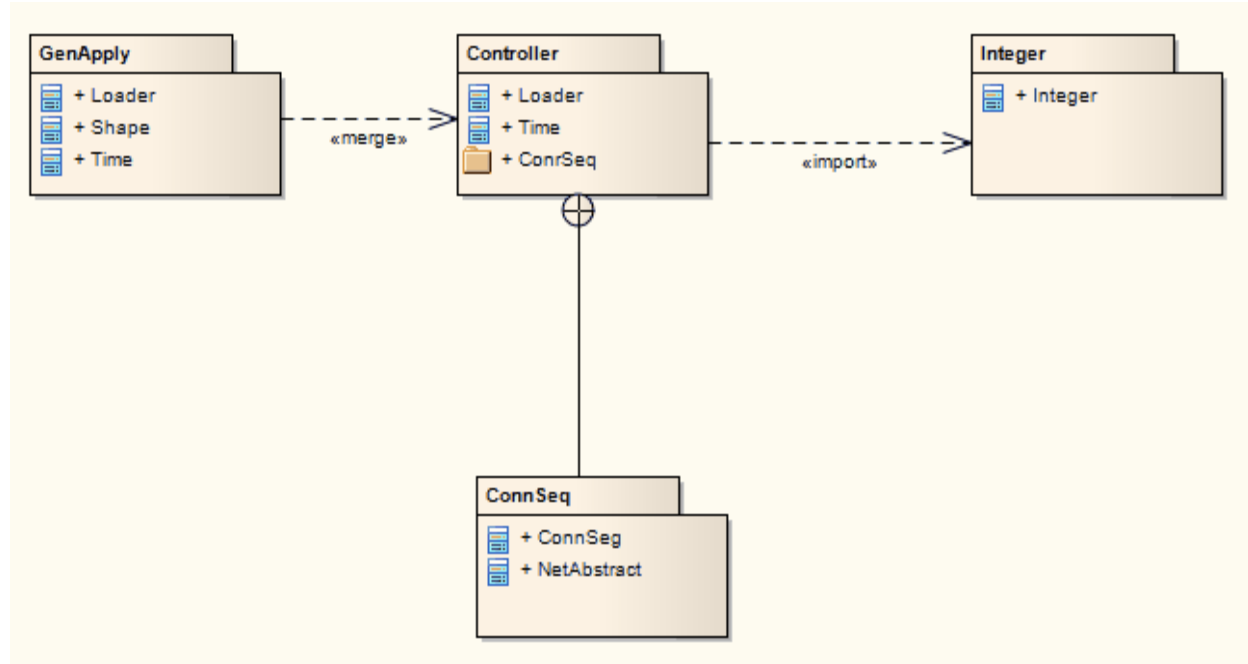
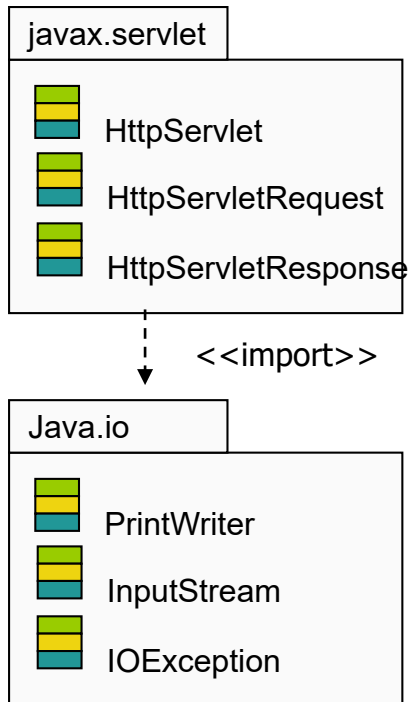
HttpServletRequest



HttpServletResponse

Package

- You can express dependencies among packages
- e.g. javax.servlet does depend on java.io

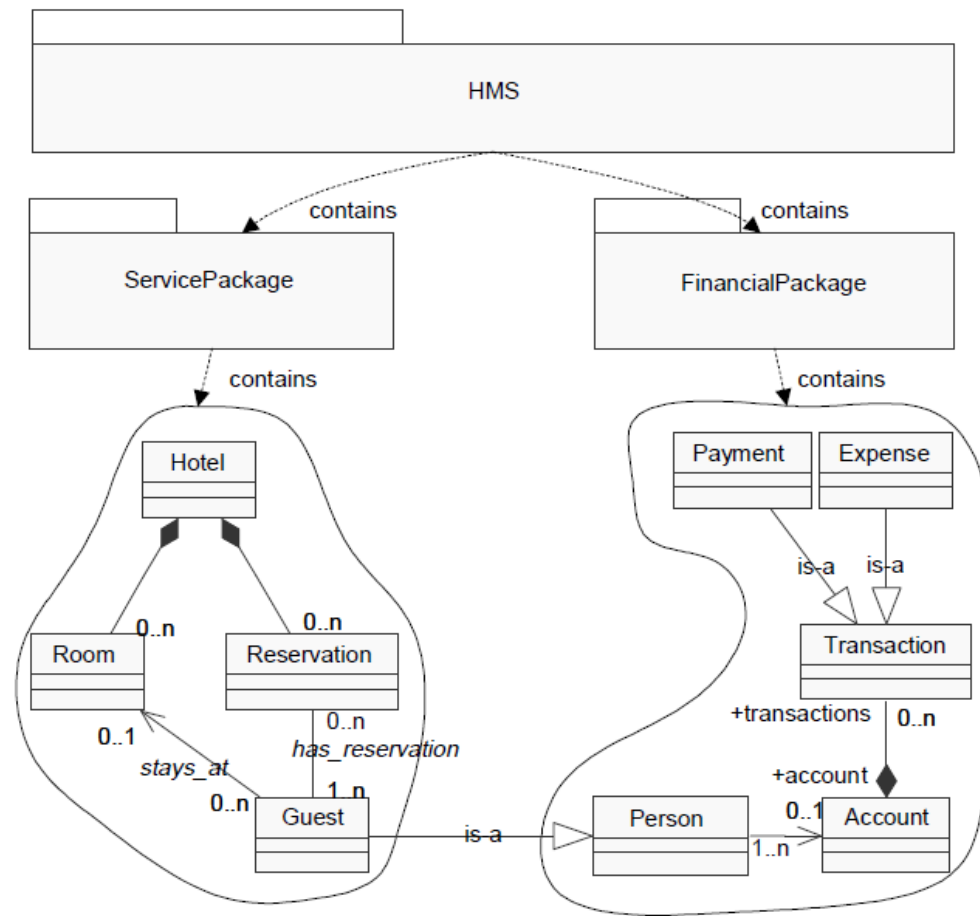


Why do you want package diagrams?

- ❑ Most often they will be used to represent division of work among teams in a larger project.
- ❑ Each team will be developing code for a specific part of the large project
 - Team-1 : DataCommunications package
 - Team-2 : Persistency package
 - Team-3 : InterfaceBuilder package
 - ...
- ❑ Each package has many classes – a subset of which will be used from code in other packages.
- ❑ Package diagrams may then be useful in high-level overview models for the system

Why do you want package diagrams?

(cont.)



Source: from Egyed "Automated Abstraction of Class Diagrams, TSE 2002

Logical View: class diagram

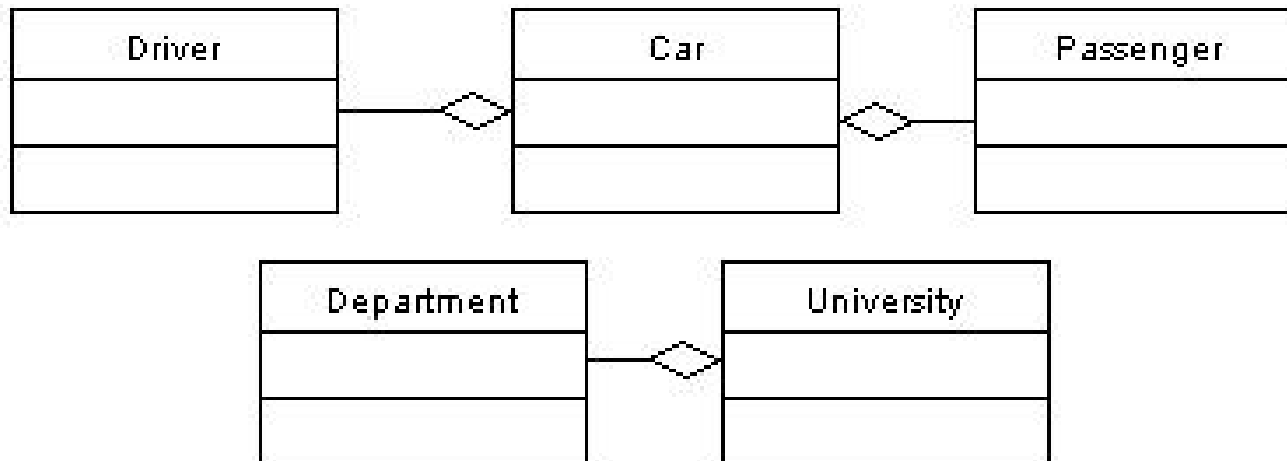
Review: class relationships

- There are three kinds of relationships between objects:
 - Aggregation: one object is a part of another object.
 - Composition
 - Inheritance: one object is a more specialized version of another object. This is also called specialization.
 - Association: The objects need to communicate (basically everything else - the default option)

Aggregation

Aggregation - a special form of association

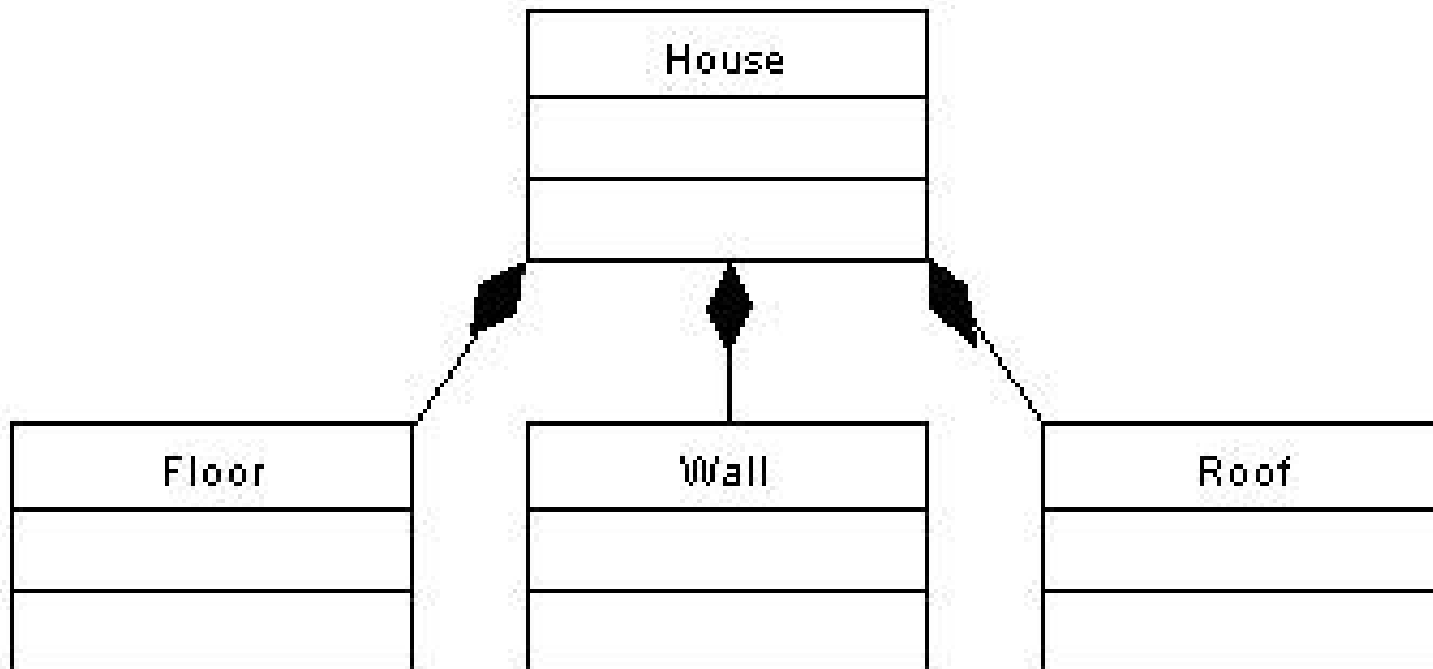
- an 'assembly-component' (i.e. an "is-a-part-of") relationship.
- assembly object is a single object made up of component objects.
 - e.g. a car HAS A driver and HAS passengers
 - e.g. a department IS PART OF a university
- **These things can also exist *independently***



Composition

Composition (____ is composed of ____)

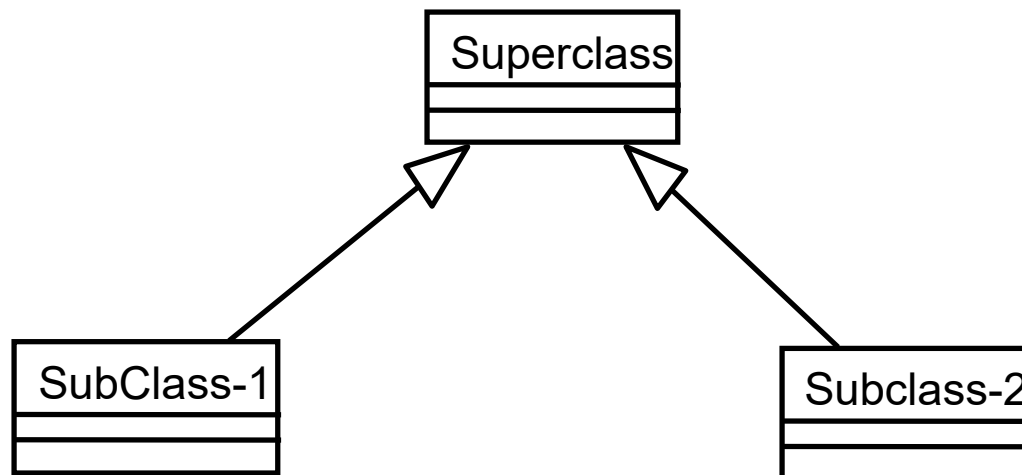
E.g. a house is composed of a floor, roof and 4 walls – if the house is destroyed, then the other things are also destroyed



Inheritance

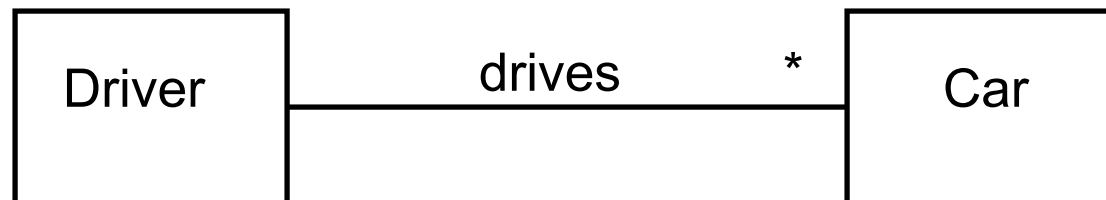
Generalization/specialization - is another special form of association

- subclasses describe properties that are specializations of a superclass
- an "is-a" relationship
- all properties of superclass are inherited by subclass (attributes, operations and associations)
- generalization hierarchy represents a tree of classes.



Association example

- ❑ Associations are the most generic – relationships.
- ❑ Any relationships which do not fit into the other two categories are association.
- ❑ These relationships often are of the type '***has-a***', '***uses***', '***communicates with***' or '***makes requests of***'



Review: Paper and Pen exercise

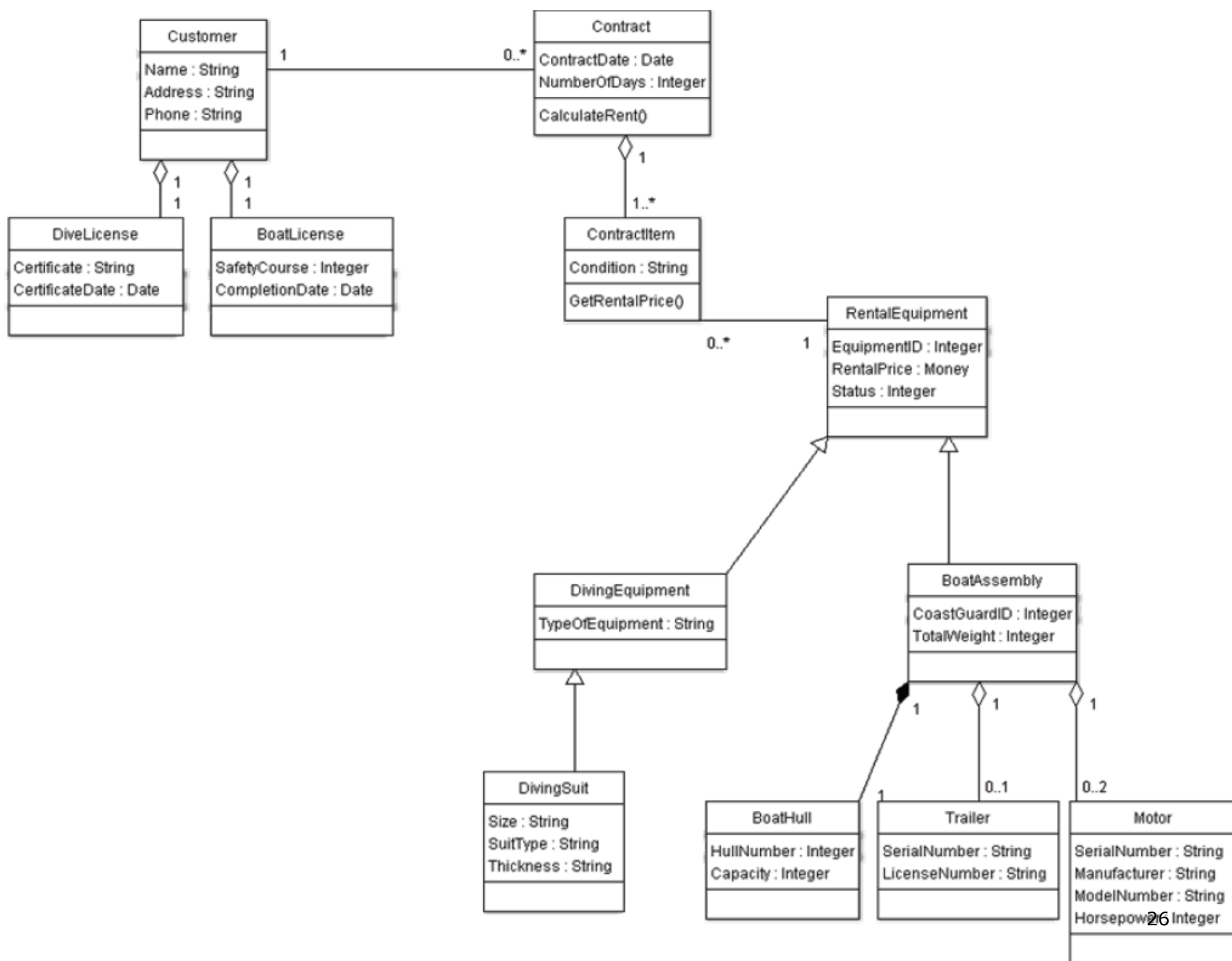
- ❑ DDT is a small business that rents diving equipment and boats.
- ❑ Due to tremendous growth of DDT, the owner, Dick, wants to build a new system for rental.

Source: "The object-oriented approach: concepts, system development, and modeling with UML" by John W. Satzinger, Tore U. Ørvik

Paper and Pen exercise (cont)

DDT requirements

- ❑ Customers can rent either diving equipment or boats from DDT. Each customer has different license (boat license or dive license).
- ❑ When the customer has made a decision about what to rent, a rental agreement, or contract, is produced and signed.
- ❑ DDT rents all of the usual diving equipment such as tanks, regulators, weight belts, diving suits, and depth gauges. Especially, divingsuits come in various sizes, thicknesses, and types (dry and wet).
- ❑ Some boats are rented with a trailer, some without a motor, and some with one or two motors. Trailers and motors always stay with the sameboat; they are never rented by themselves or taken away from the boat they belong to (except to be serviced or replaced).
- ❑ A customer can rent one piece of equipment or many pieces of equipment; for example diving customer can rent only regulator or a collection of items for diving. A customer who already rented a boat can rent a diving equipment. Thus, A contract contains many items of equipment. (In other words, each piece of rental equipment is rented many times, and over time it is associated with many contracts).



Logical View

State Machines

- Sequence diagrams allow you to model the dynamic aspects of a system by considering the Use Cases.
- A state machine is a **behavioral** model which specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.
- A State Machine diagram is drawn for every object class that demonstrates behaviour, i.e. it has to respond to external stimuli, human or machine.
- The set of state machine diagrams for all classes that exhibit externally observable behaviour = the **dynamic model** of the system

State Diagrams

❑ What is a State?

- It is a set of values (of attributes) that describe an object at a specific point in an object's life
- Student's name, student's ID, student's program, student's semester, etc. determine a state of the Student object.

❑ Does an object's state change?

- Some of the objects in the system are quite dynamic in that they pass through a variety of states over their existence.
- E.g. A student can change from "new" to "current" to "former"

State Diagrams

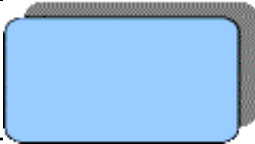



- What is a State diagram?
 - It is a dynamic model that shows the different states that a single object can pass during its lifetime in response to events.

State Diagram

Event

- What is an Event?
 - An event is something that takes place at a certain point in time and changes a value(s) that describe an object (the state of an object)
 - E.g. the student graduates from the university is an event (that changes the Student's state from "current" to "former")
 - Sometimes we even have *concurrent events*. These are two events that are related only by coincident and have no effect on each other.
 - e.g. 2 planes take off at the same time

State Diagram

Term and Definition	Symbol
A State Is shown as a rectangle with rounded corners.	
An Initial State Is shown with a small filled in circle	
A Final State Is shown with as a circle surrounding a small solid filled-in circle. This represents the completion of activity.	
An Event Is an occurrence that triggers a change in state. It is used to label a transition.	Event Name
A Transition Indicates that an object in the first state will enter the second state. Is triggered by the occurrence of the event labeling the transition. Is shown as a solid line.	

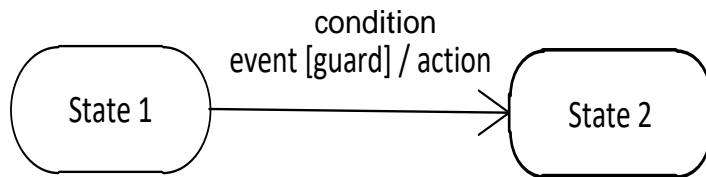
Transitions

- ▣ A transition is a relationship between 2 states indicating that an object in the first state will perform certain actions and enter the 2nd state when a specific event occurs.

5 Parts to a Transition

- ❑ **Source State** - The state affected by the transition, - when this object receives the event, then it may “fire”
- ❑ **Event Trigger** - The event whose reception by the object makes it eligible to “fire”
- ❑ **Guard Condition** - An optional expression that is evaluated to see if the transition should “fire”. A guarded transition fires when its event occurs, but only if the condition is true.
 - E.g. "when you go out in the morning (event), if the temperature is below freezing (condition), then put on your gloves (actions)" and move to next state.
- ❑ **Action** - An executable atomic computation that may act directly on the object that owns the state machine and indirectly on other objects that are visible to the object.
- ❑ **Target State** - The state that is active after the completion of the transition.

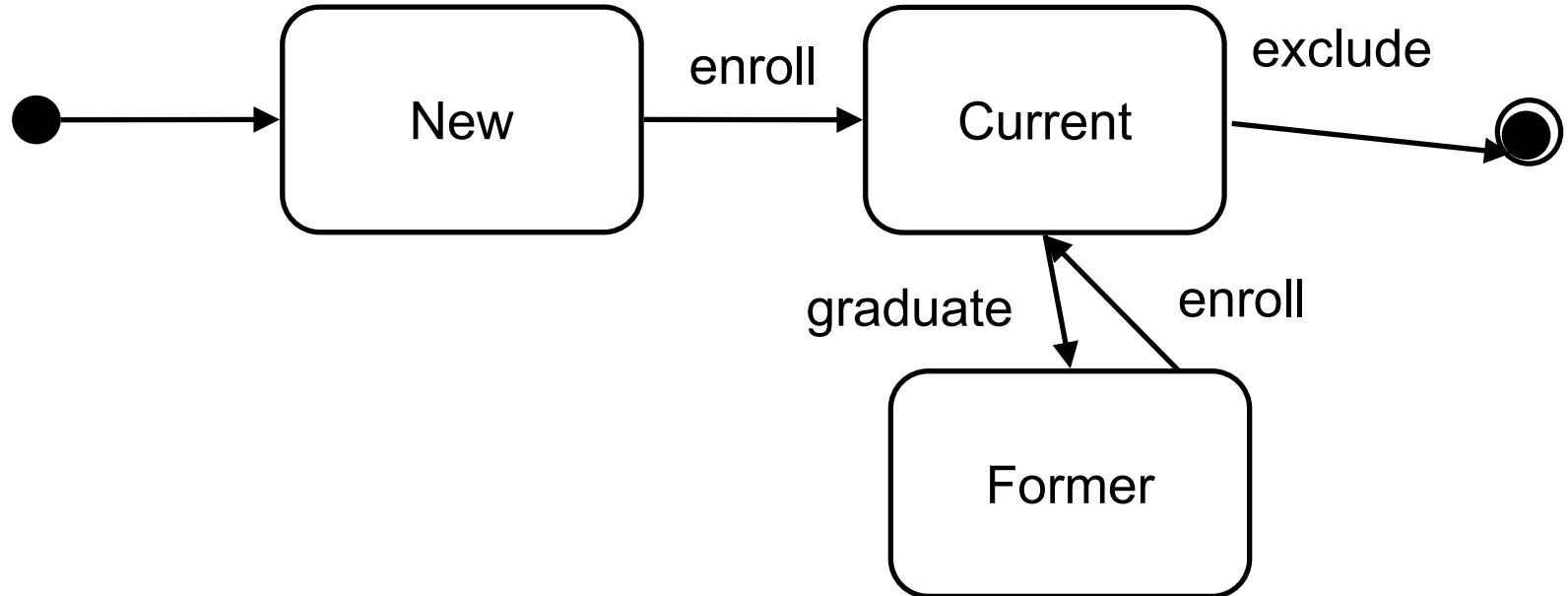
State Machine Notations



Note: guard and action are optional

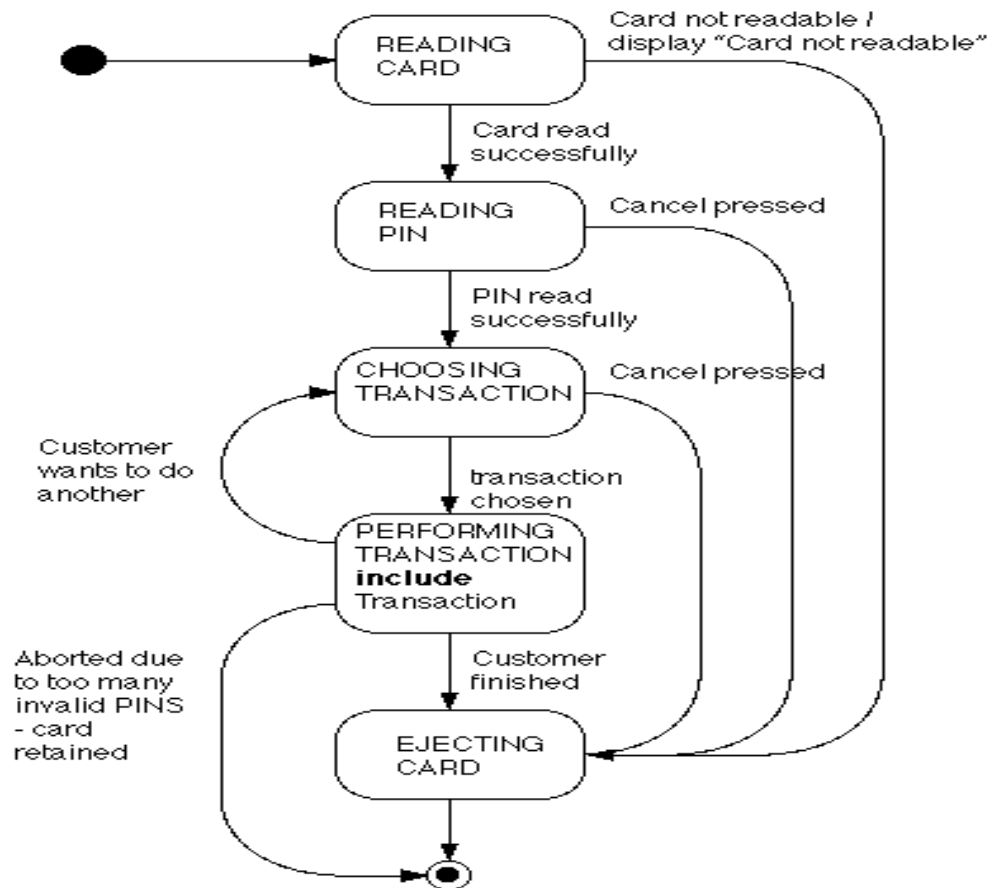
Simple State Diagram

- Different states of a student



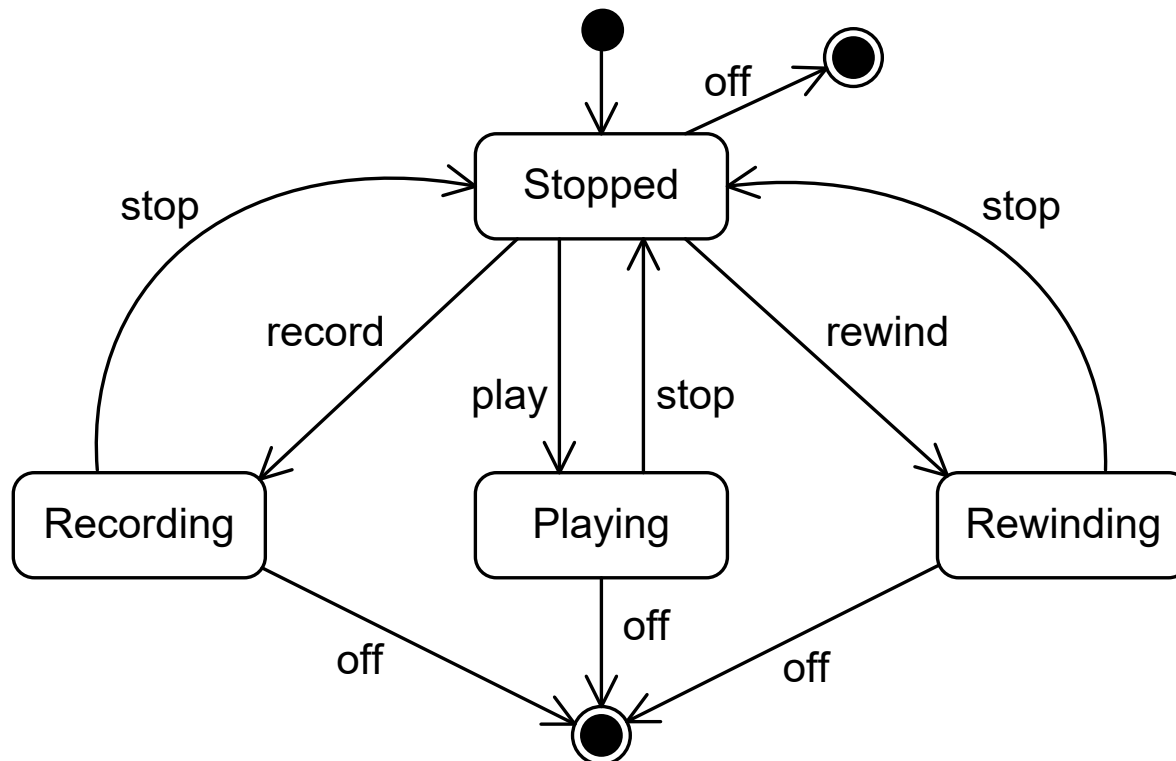
Example: ATM machine

State-Chart for One Session

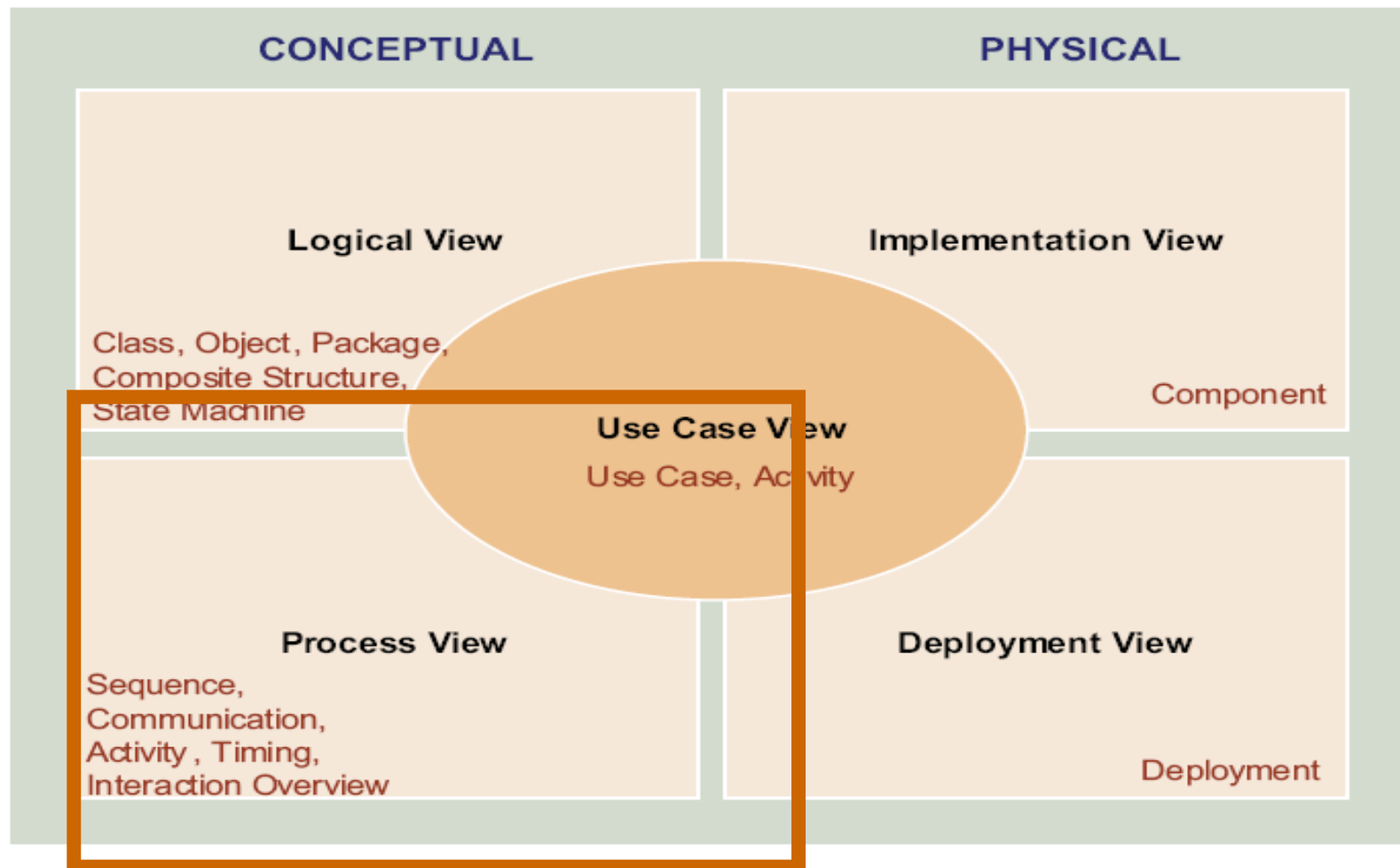


Pen and Paper exercise

Draw a state diagram for a tape recorder

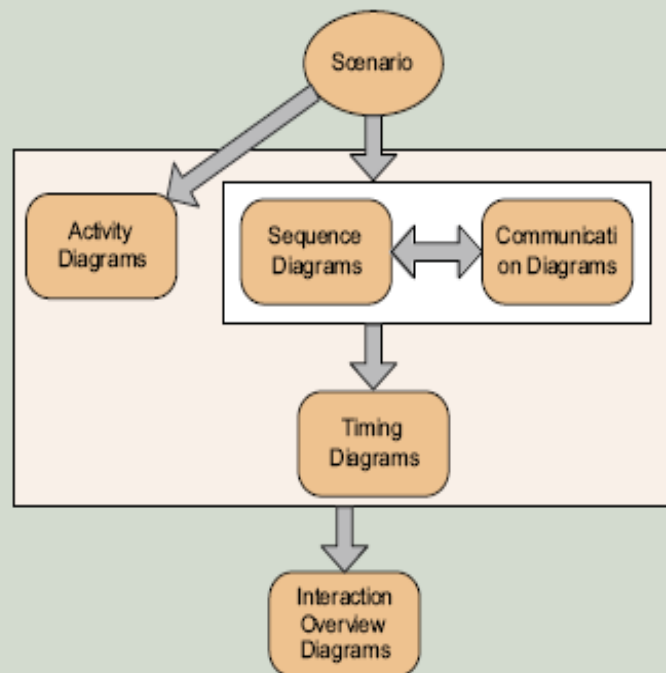


Diagrams and views



Process View

MODELING PROCESS VIEW WITH UML2



1. Use either Sequence or Communication Diagrams for modeling simple interactions in use case realizations

Optional use

2. Add Activity diagrams to realize scenarios where business logic is a sequence of actions and involves branching and parallel processing
3. Add timing diagrams when modeling for performance
4. For complex scenarios, that can be composed of other scenarios, use Interaction overview diagrams

Process view

What are Sequence Diagrams?

- ❑ Sequence Diagrams show how certain objects (or groups of objects) *collaborate* in some sequential behaviour.
- ❑ Generally a Sequence Diagram describes one Use Case

Sequence diagram

□ Message exchange

- Communication between objects is achieved by exchanging messages
- The objects send messages to each other
- These messages lead to the **operations**, which means that an object understands precisely those messages for which it has operations

Sequence Diagram

- ❑ Used to model how objects interact together
- ❑ Graphically, a sequence diagram is a table that:
 - shows objects arranged along the X axis and
 - messages, ordered in increasing time, along the Y axis. These messages indicate the activation of an operation.
- ❑ The object that initiates the action is placed on the left and other increasingly subordinate objects to the right.

Sequence Diagram (cont.)

Objects

- Shows instance name and/or class name
(instance_name:class_name)

Hung : Student

|

Hung :

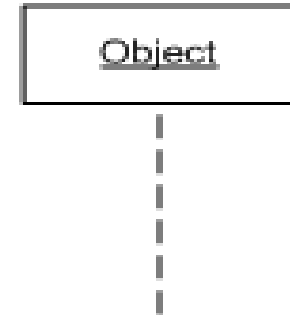
|

: Student

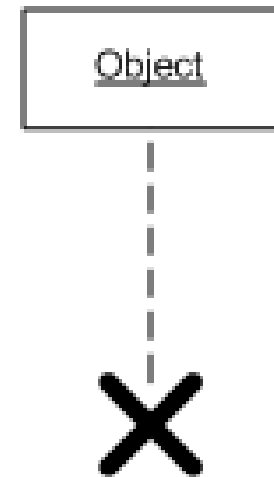
|

Sequence Diagram (cont.)

A lifeline shows the life of an object



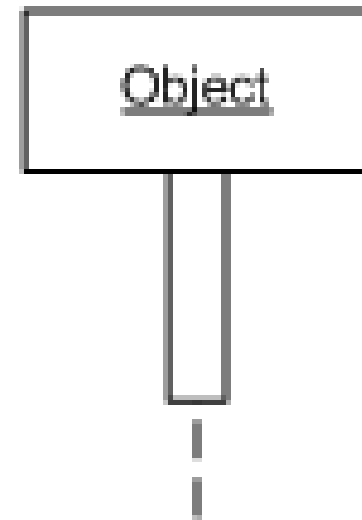
The X shows the point at which the object is deleted



Sequence Diagram (cont.)

Focus of control

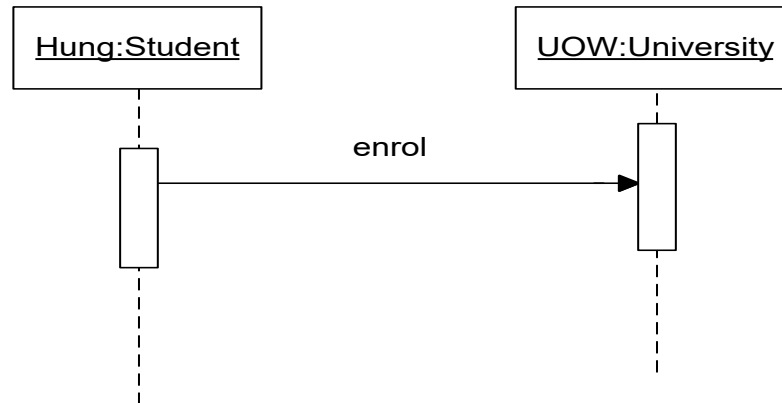
- Shown by a long, hollow, narrow rectangle placed over a lifeline
- This shows which object has control (sending or receiving messages)



Sequence Diagram (cont.)

Simple message

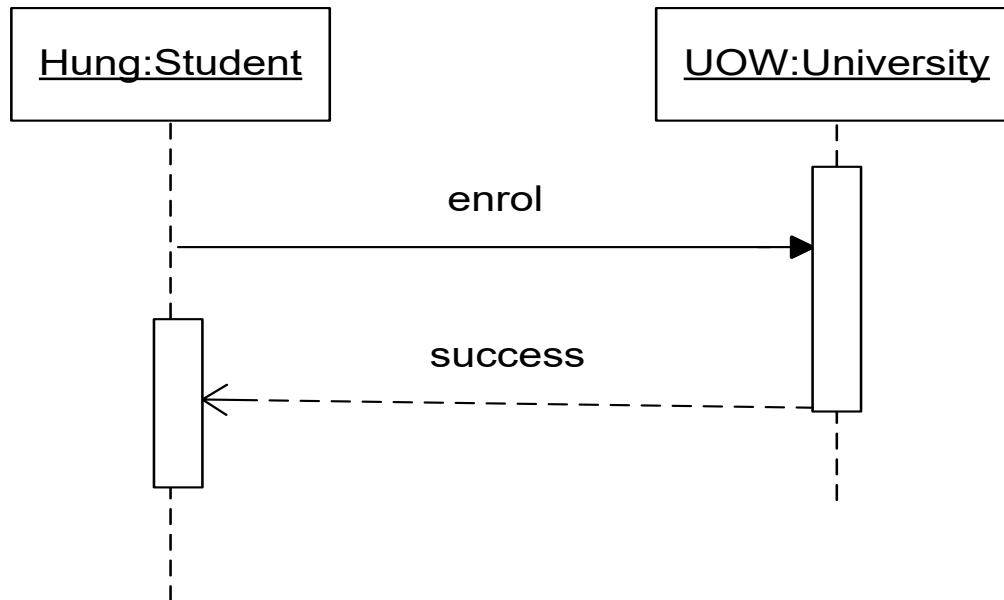
- Shown by a line with a filled in arrow head



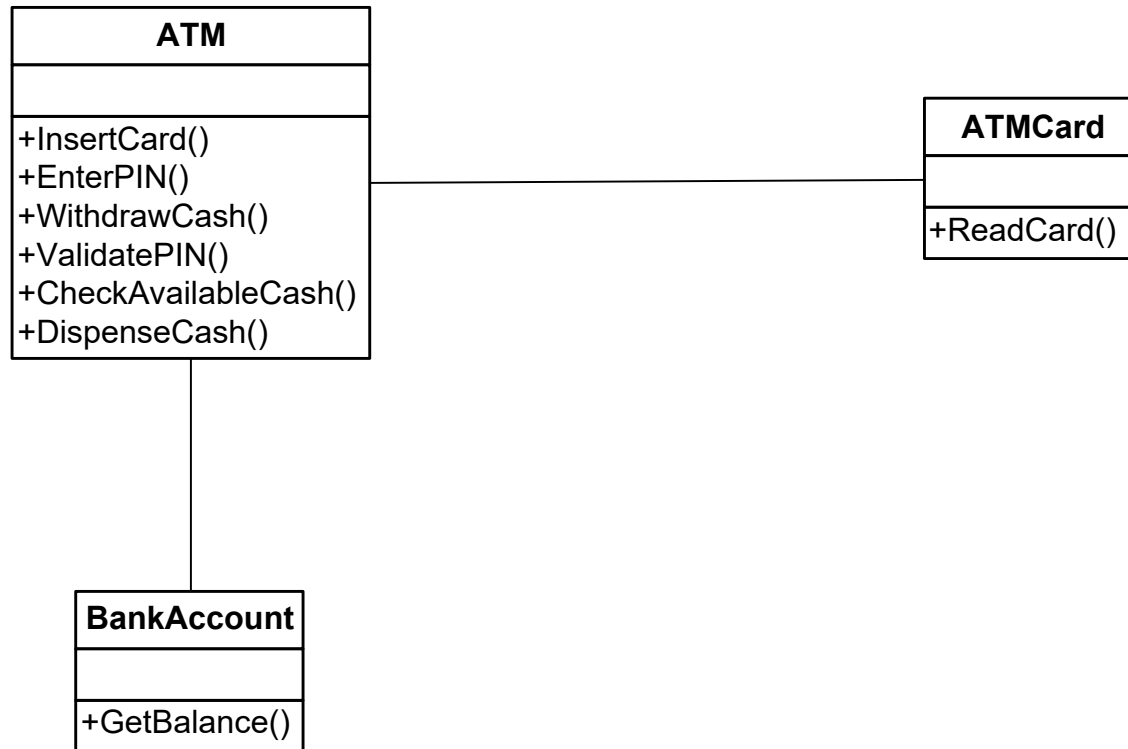
Sequence Diagram (cont.)

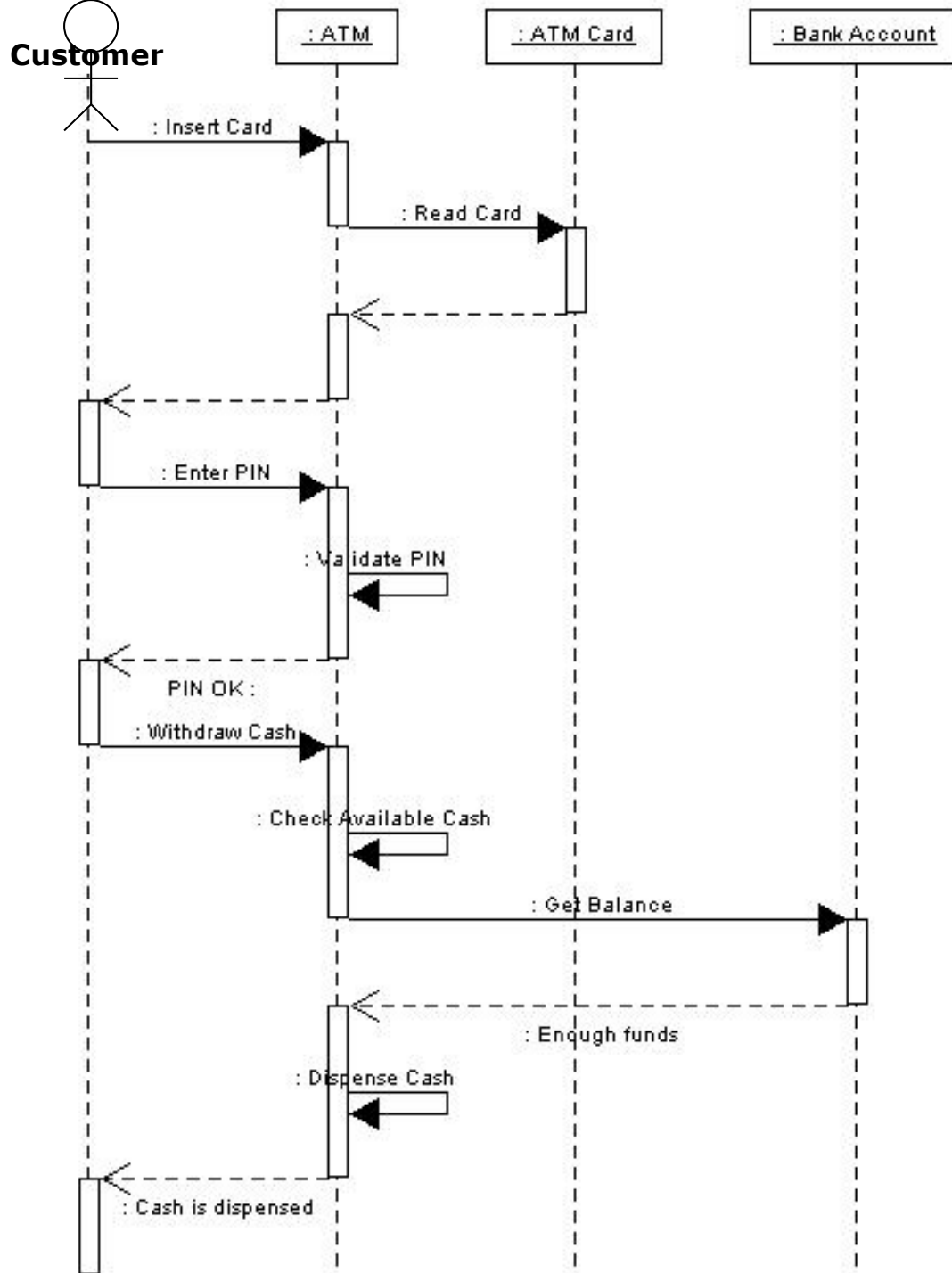
Message return (or return from function)

- Shown by dotted line with empty arrow head
- Note: Not required, and the diagram quickly becomes busy if show all returns



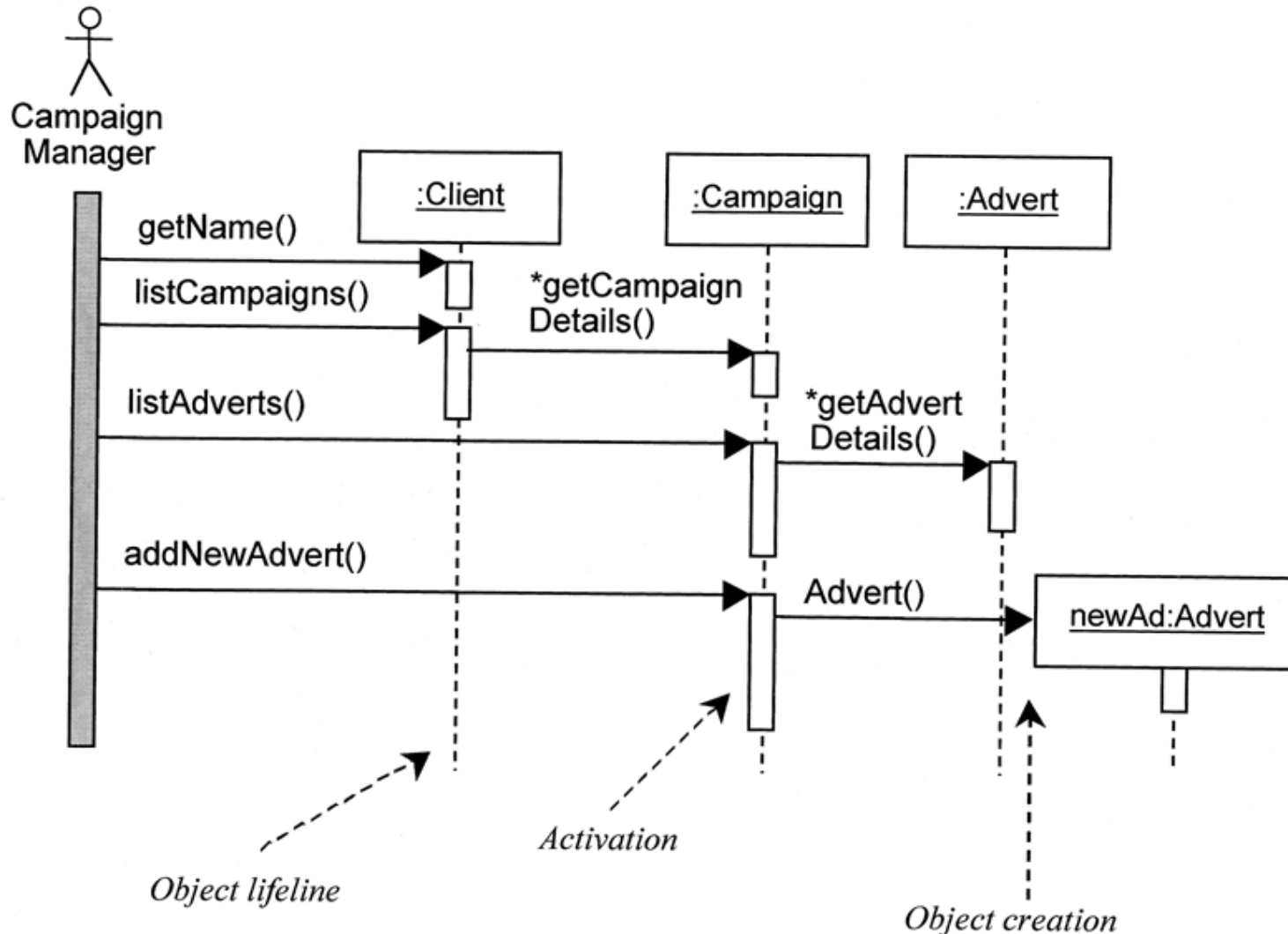
ATM Example





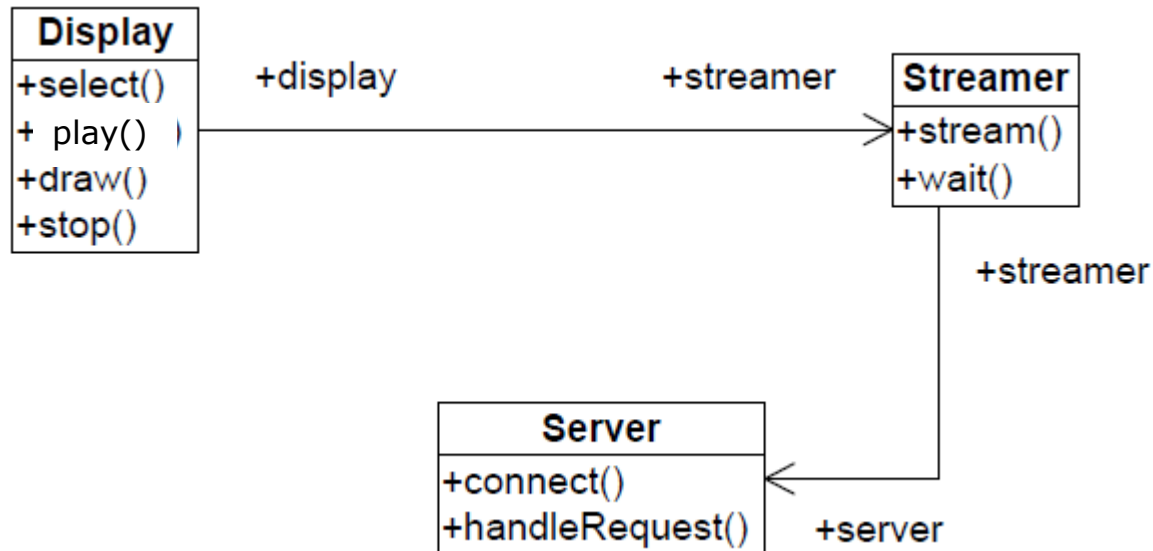
Sequence
Diagram showing
Withdraw Cash
use case

Another Example



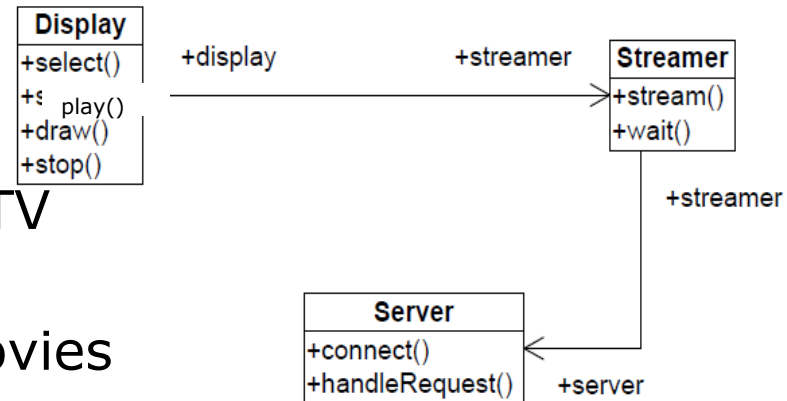
Pen and Paper Exercise

- A simplified Video-on-demand (VOD) system



Pen and Paper Exercise (cont.)

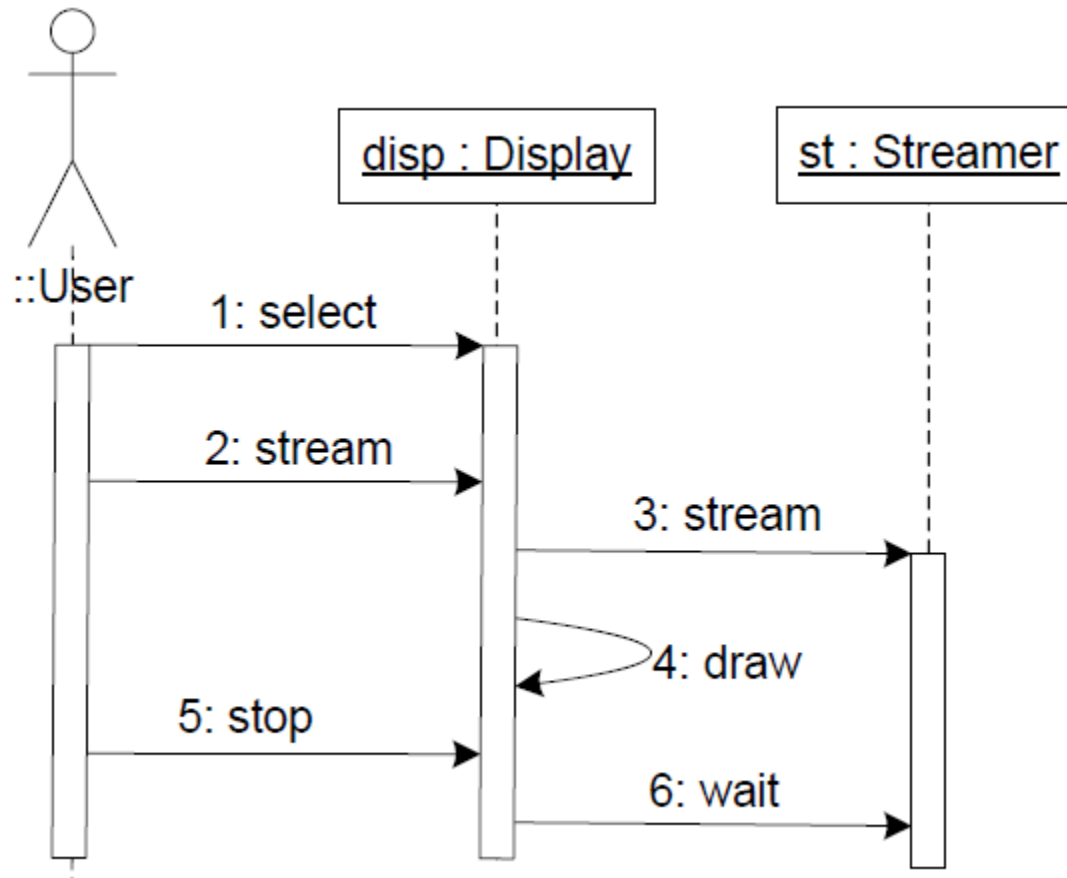
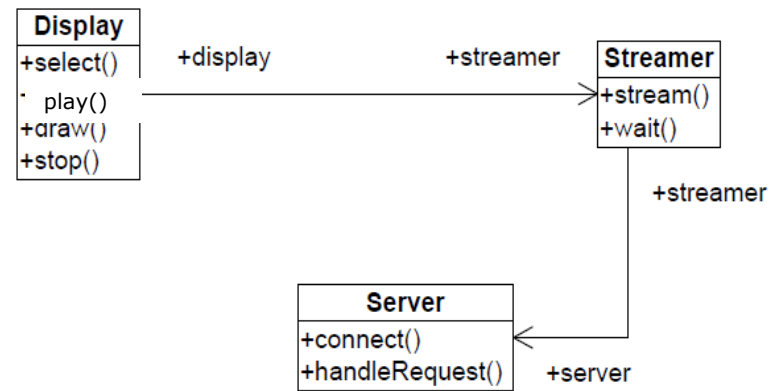
- ❑ Use Case: Watch a movie
- ❑ Participating Actors: User
- ❑ Triggering event: User turning on TV
- ❑ Normal Flow of events
 - The system displays a list of movies
 - The user selects a movie that she wants to see
 - She then starts playing the selected movie
 - The system retrieves the movie stream and renders the movie
 - The user stops viewing the movie.
 - The system stops retrieving the movie stream



Pen and Paper Exercise (cont.)

- ▣ Based on the class diagram, draw a sequence diagram for use case “Watch a movie”.

Pen and Paper Exercise



B-C-E design

- Consider applying the B-C-E model
- Entity-layer
 - Holds the data and is responsible for its persistence.
- Control layer
 - Contains the rules about how to combine info and how to deal with interaction
- Boundary layer
 - Responsible for representing info to user and to receive user interactions.

B-C-E design(cont.)

□ <<entity>> classes

- Represent business domain concepts – you will usually have identified many in the domain class diagram
- Entity classes used to store and manage information in the system – they map directly to database tables
- Some entity classes may be transient – i.e. they do not get stored in DB but are only temporary

B-C-E design (cont.)

□ <<boundary>> classes

- Used to model interaction between the environment and the system
- Boundary class typically represents a line between an actor and a use case in the use case diagram.
- Actors interact only with boundary classes (actors might be user or external system)
 - In some cases, two actors can interface to a use case via the same boundary class.
- *A boundary class does not provide the actual behavior of the use case - it typically represents the GUI - e.g. dialog, menu, screen.*

B-C-E design (cont.)

□ <<control>> classes

- Also known as a *use case controller* class - coordinates all the messaging between the entity classes.
 - The use case controller implements the sequence diagram messaging.
- They are the “glue” between boundary and entity classes – i.e. they connect the user with the business data
- They contain the application-specific business rules
 - E.g. a password should have 8 characters; a student should enroll in 4 subjects per semester; a customer cannot withdraw more than \$800 per day.
- Control classes are used to model control behavior specific to one or more use cases - Usually have one control class per use case (some use cases may not need one)
 - They are created when the system performs the use case and usually die when the use case is performed

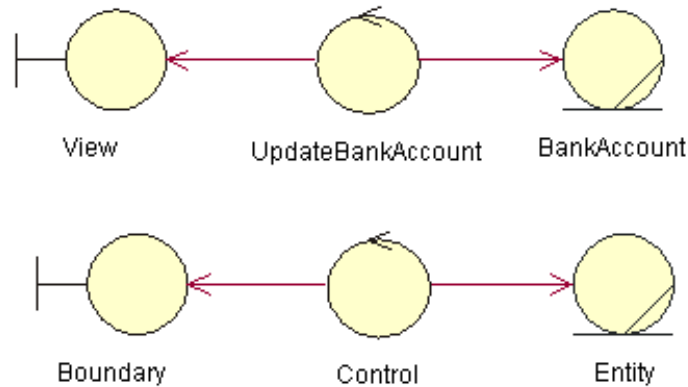
B-C-E design (cont.)

□ **Basic rules of b-c-e**

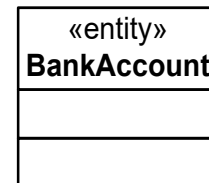
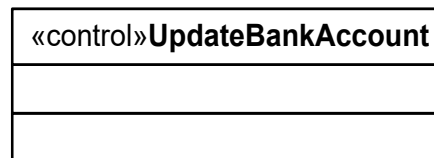
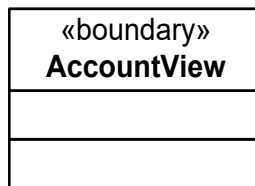
- actors can only talk to boundary classes
- boundary classes can only talk to controllers and actors
- entity classes can only talk to controllers
- controllers can talk to boundary and entity classes and to other controllers, but not to actors

B-C-E design(cont.)

Option 1:



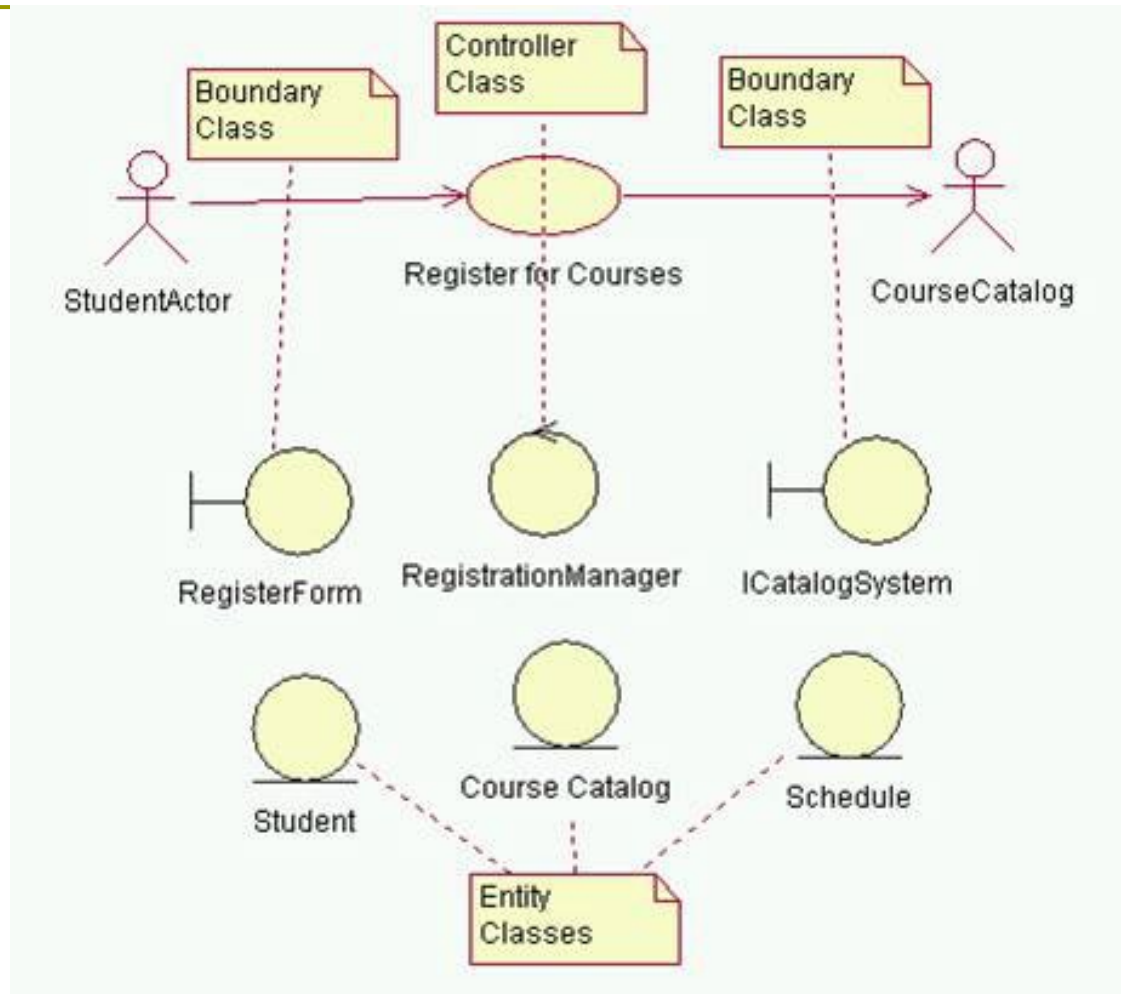
Option 2:



UML Representation

B-C-E design(cont.)

The boundary object collects the information from the actor and translates it into an interface neutral form that can be used by the control and entity objects.

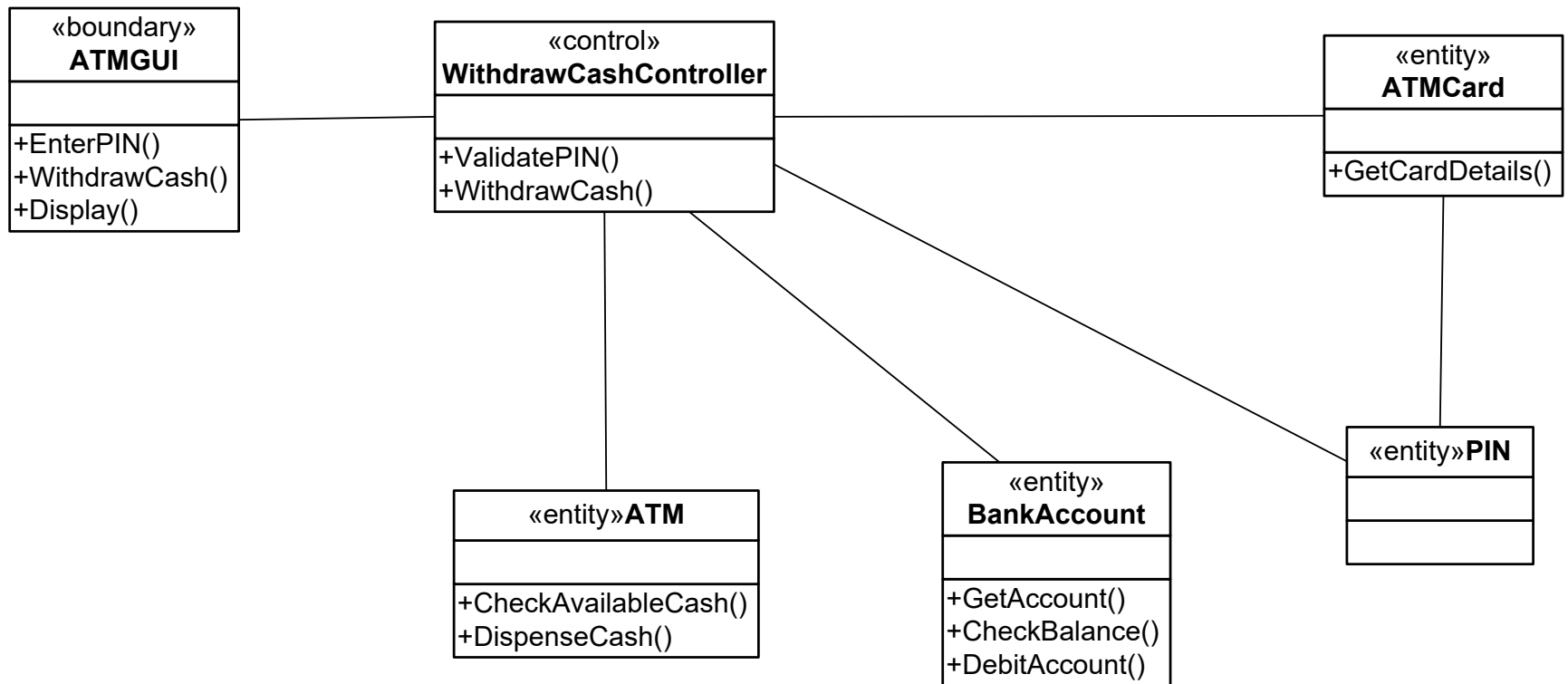


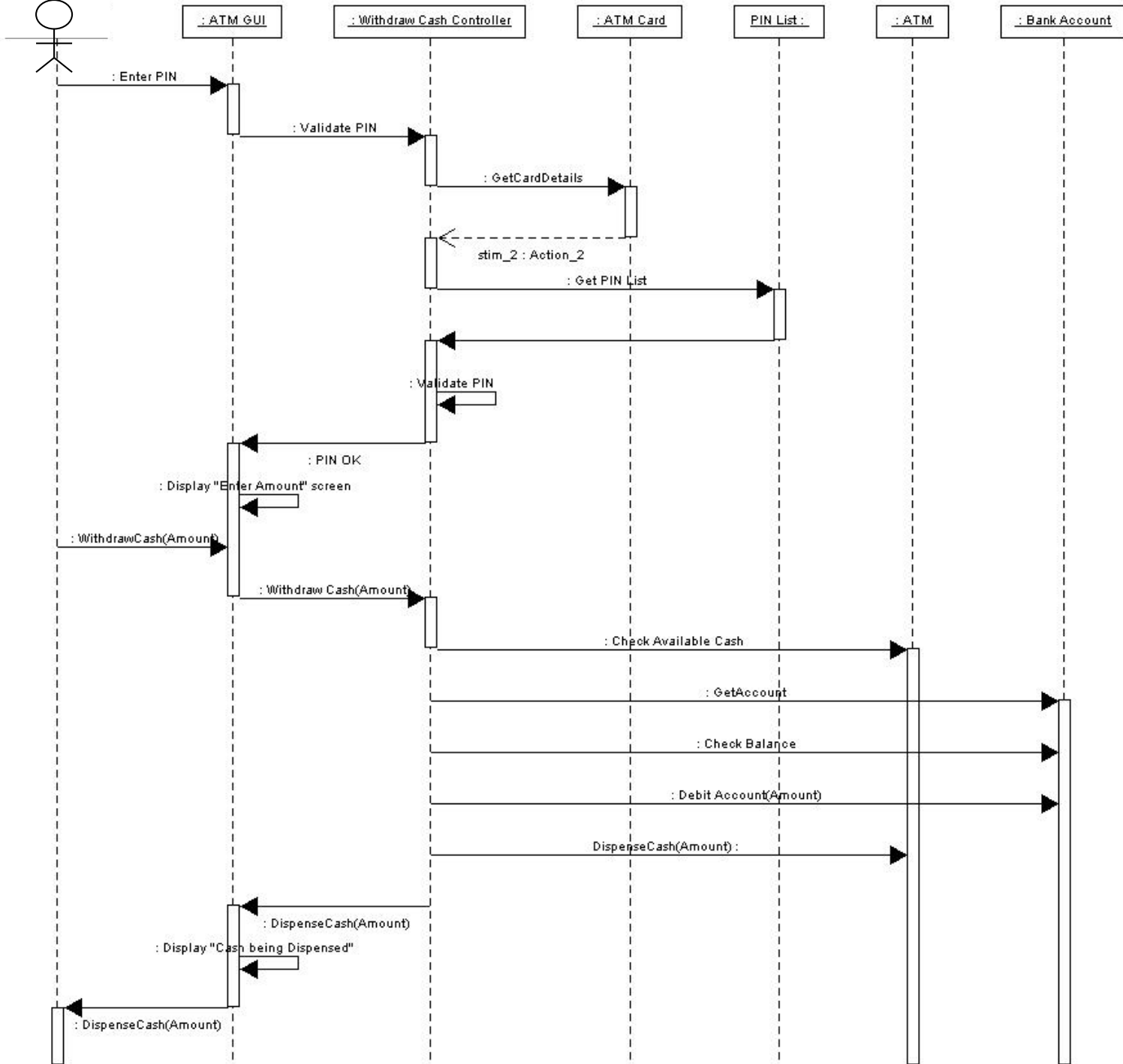
B-C-E design (cont.)

□ **Applying the b-c-e framework**

- Take an ATM system. A use case has been identified called Withdraw Cash.
- A class diagram models this system. The diagram is not very refined.

B-C-E design (cont.)



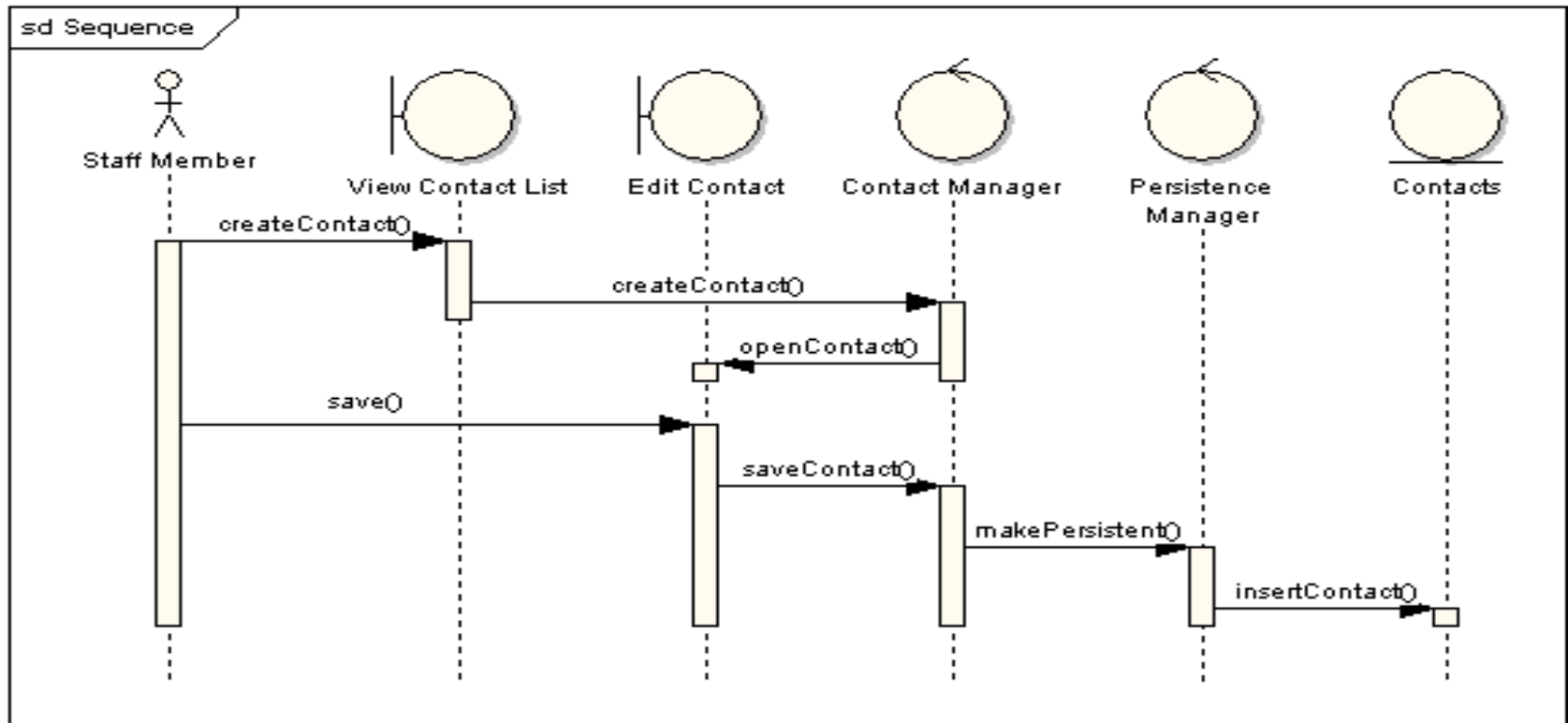


Process View

Communications diagrams

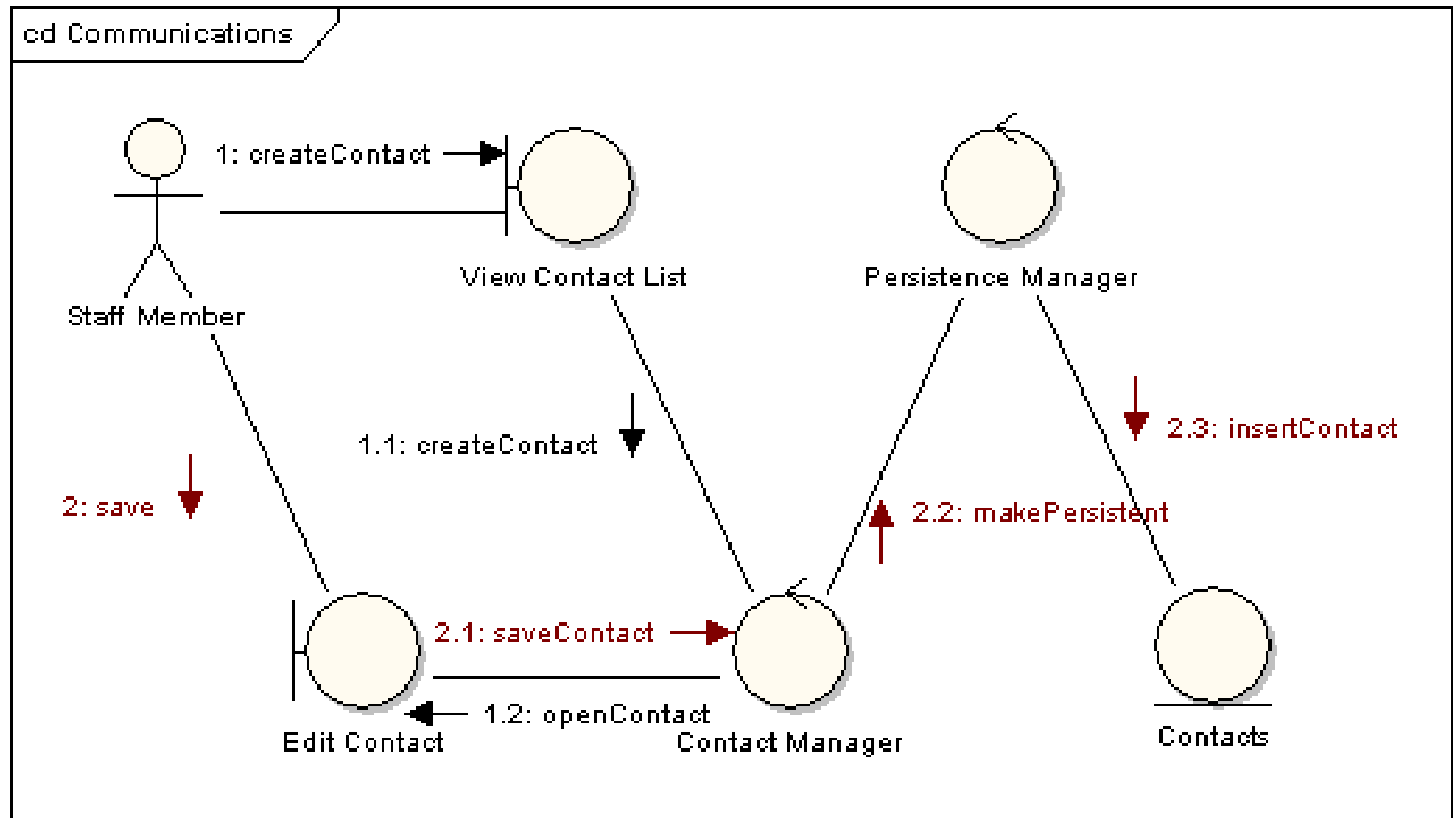
- ❑ (In UML-1, called “collaboration diagrams”)
- ❑ Convey the same information as a sequence diagram – a difference in style.
 - Sequence diagrams focus on messages occurring over a timeline
 - Communication diagrams focus more on links between participating objects
 - ❑ Sequence of messages is still shown, but sequence is less clear and relationships are more clear

Sequence diagram vs. Communications diagrams



An example of sequence diagram

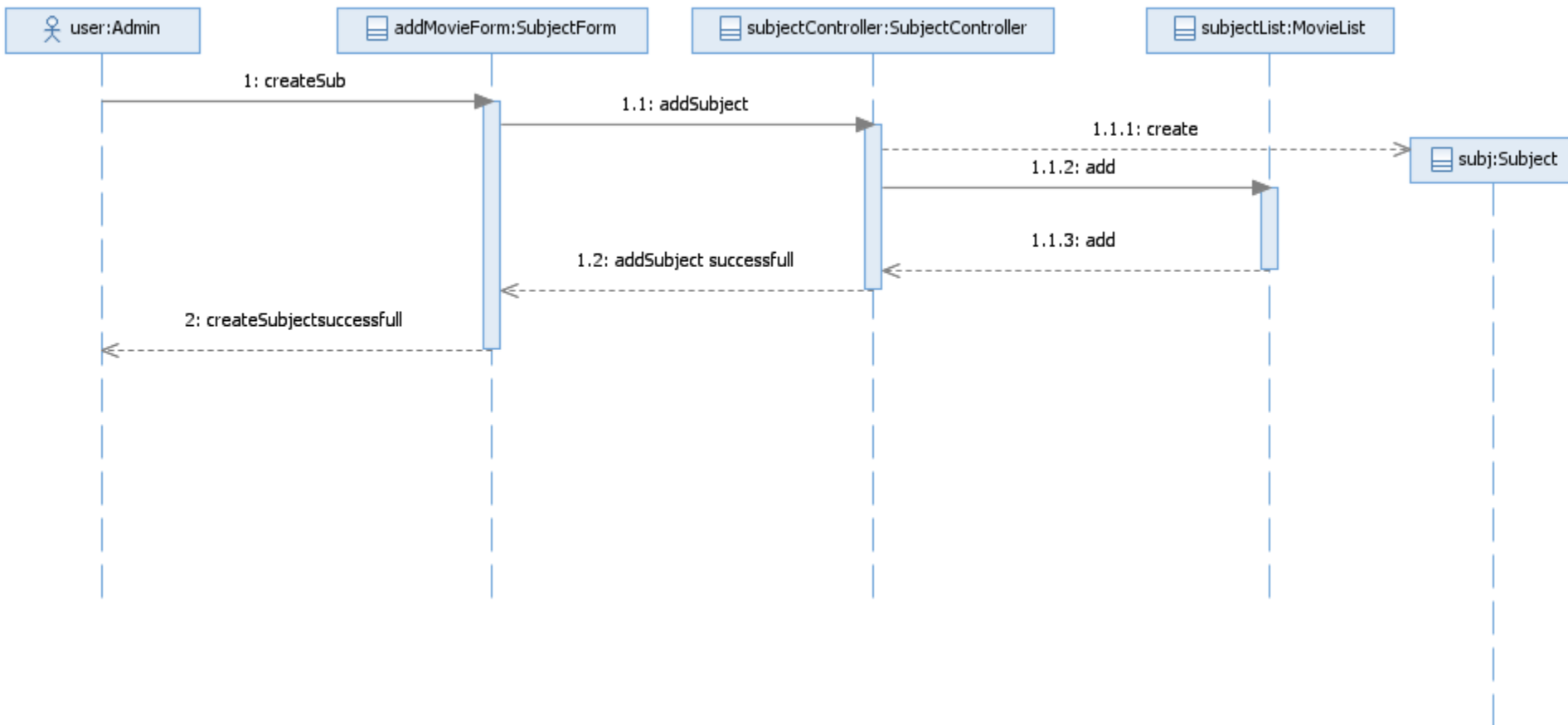
Sequence diagram vs. Communications diagrams



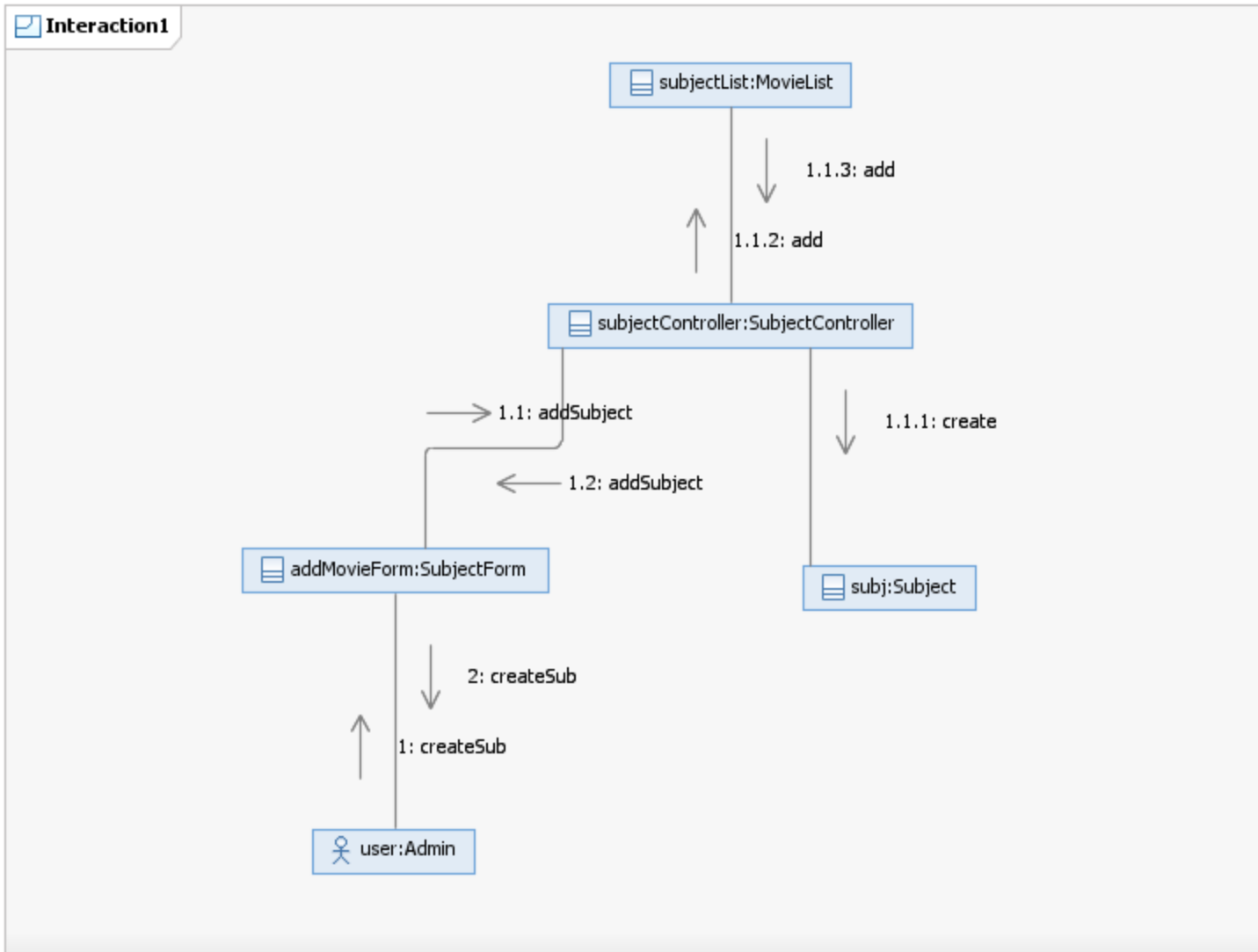
An example of communications diagram

Pen and Paper exercise

- Draw a communication diagram corresponding to the following sequence diagram



Pen and Paper exercise

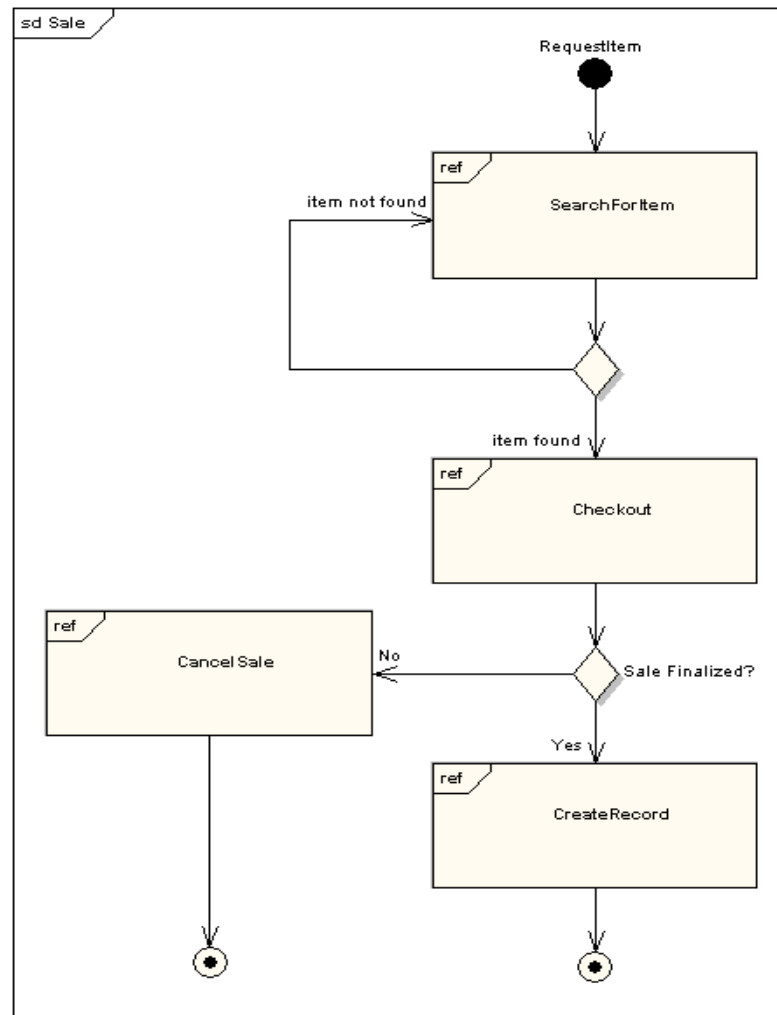


Process View

Interaction Overview Diagrams

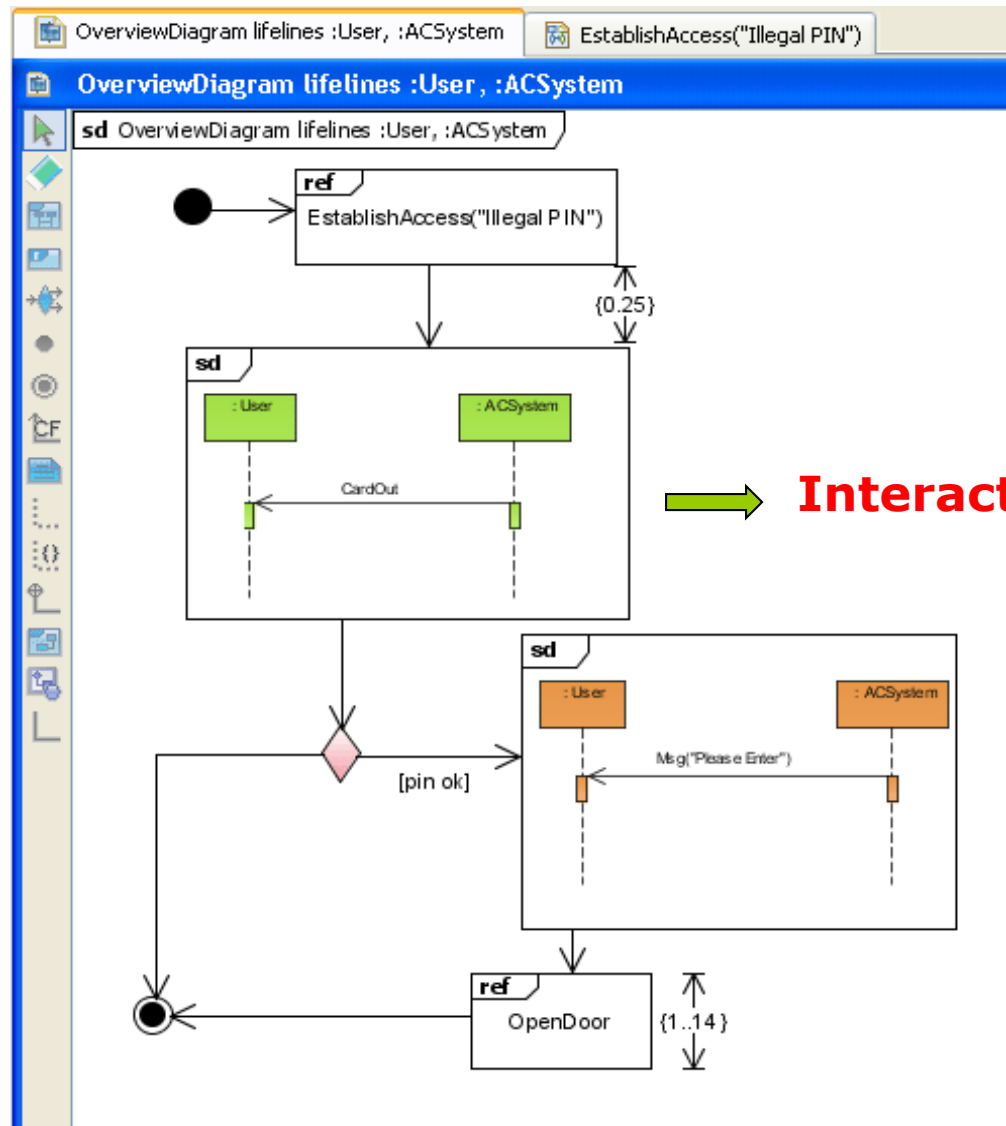
- Interaction overview diagram
 - a form of **activity diagram** in which the nodes represent “interaction diagrams”.
 - “Interaction diagrams”
 - sequence,
 - communication,
 - And interaction overview diagram (recursive definition!).
- New elements:
 - interaction occurrences
 - interaction elements.

Interaction Overview Diagram

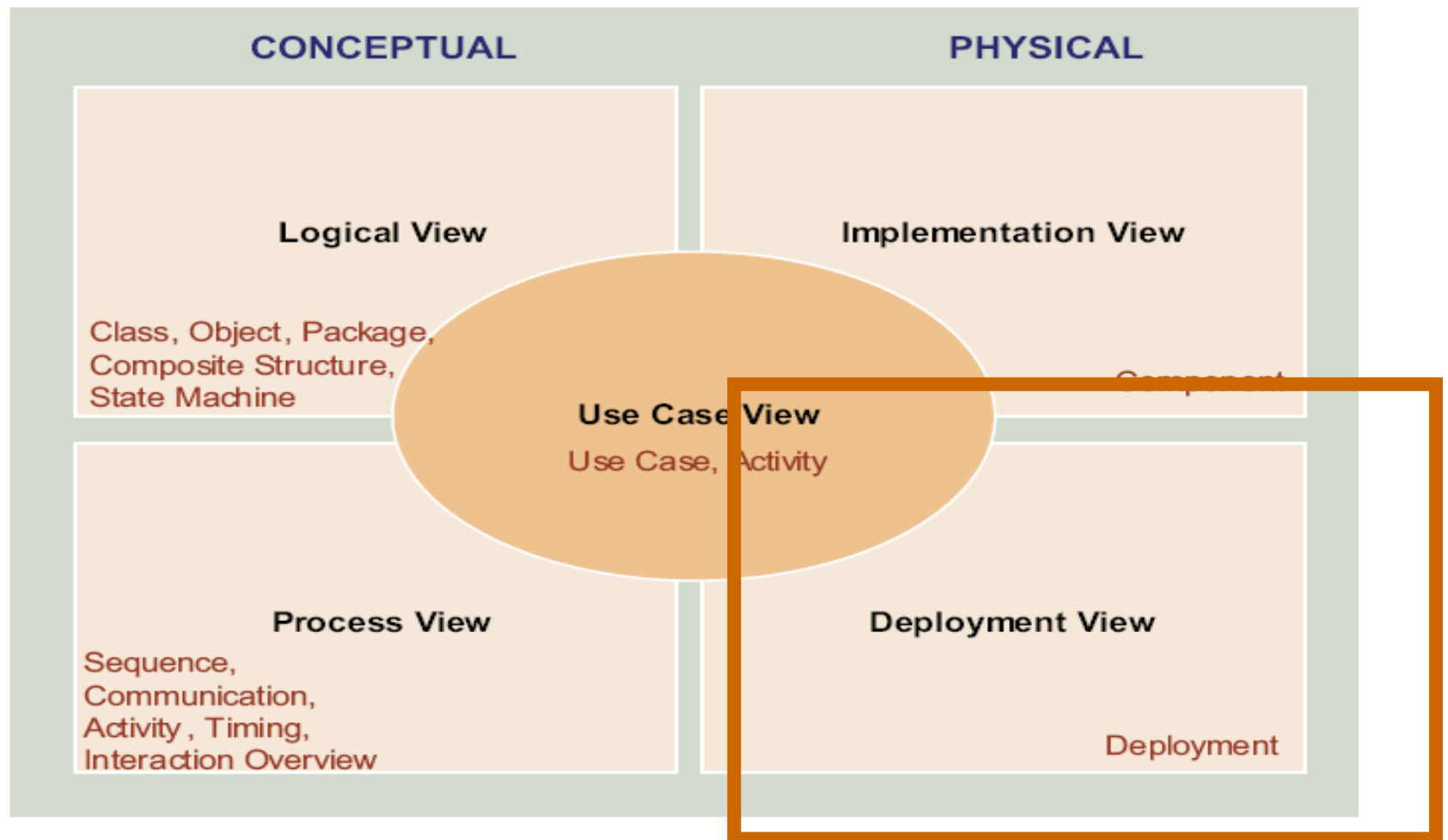


→ **Interaction occurrence**

Interaction Overview Diagram

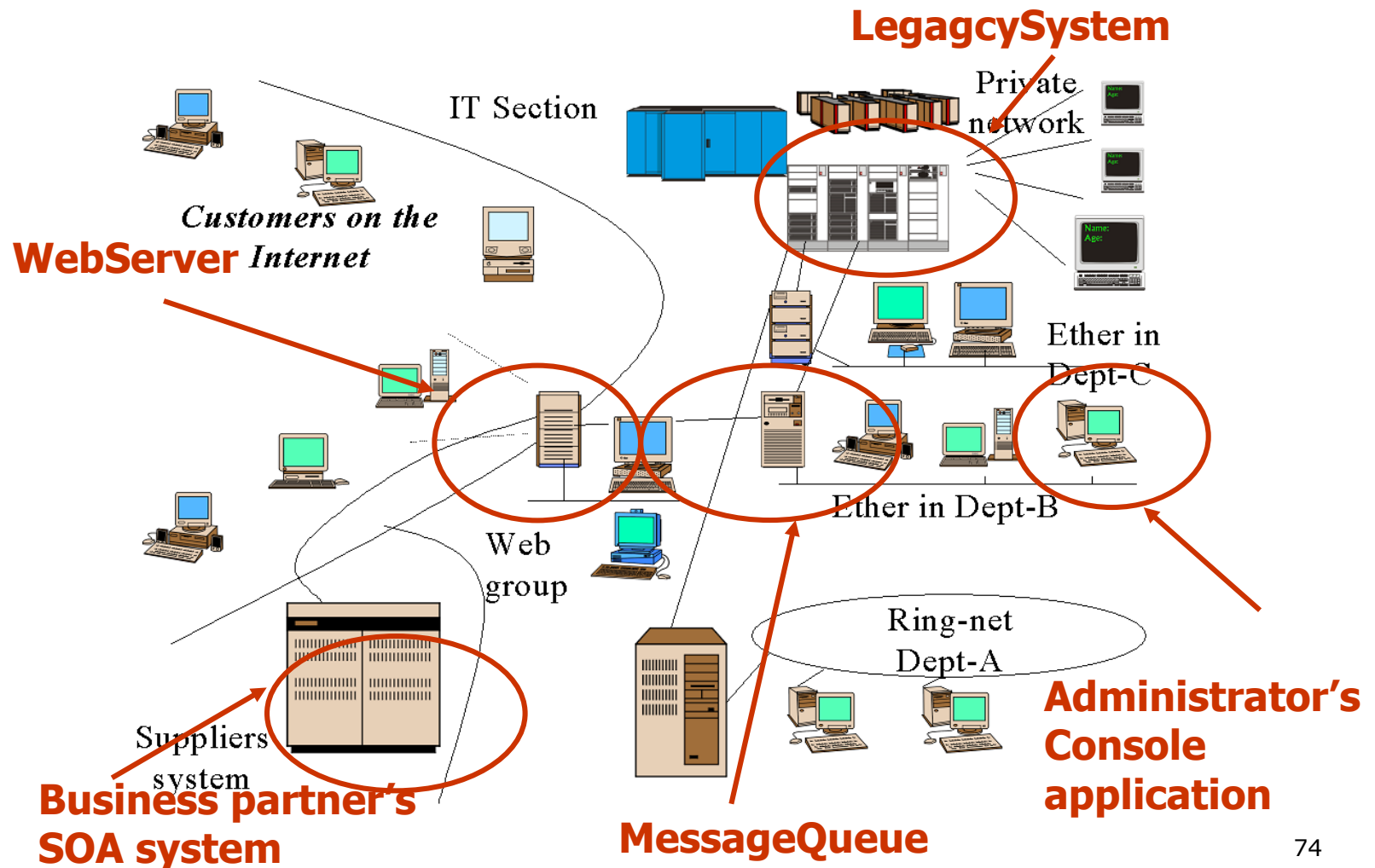


Diagrams and views



Deployment view

Where do things run?



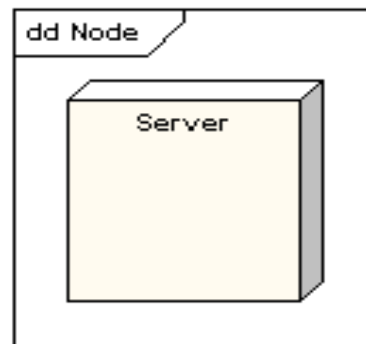
Deployment Diagrams

□ Deployment Diagrams

A deployment diagram models the run-time architecture of a system. It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.

□ Node

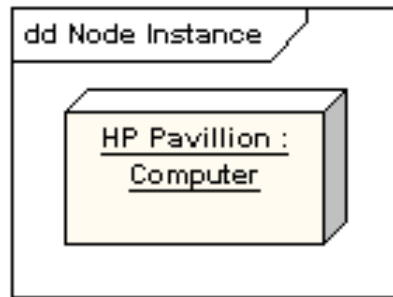
A Node is a hardware (or software) element, shown as a three-dimensional box shape.



Deployment diagram

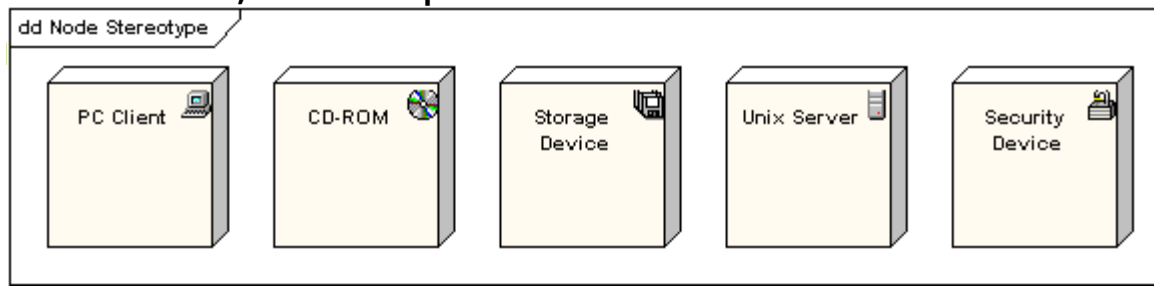
□ Node Instance

- Name and base node type



□ Node Stereotypes

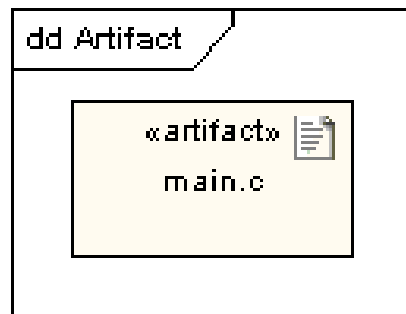
- standard stereotypes are provided for nodes,
 - «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc».



Deployment diagram

□ Artifact

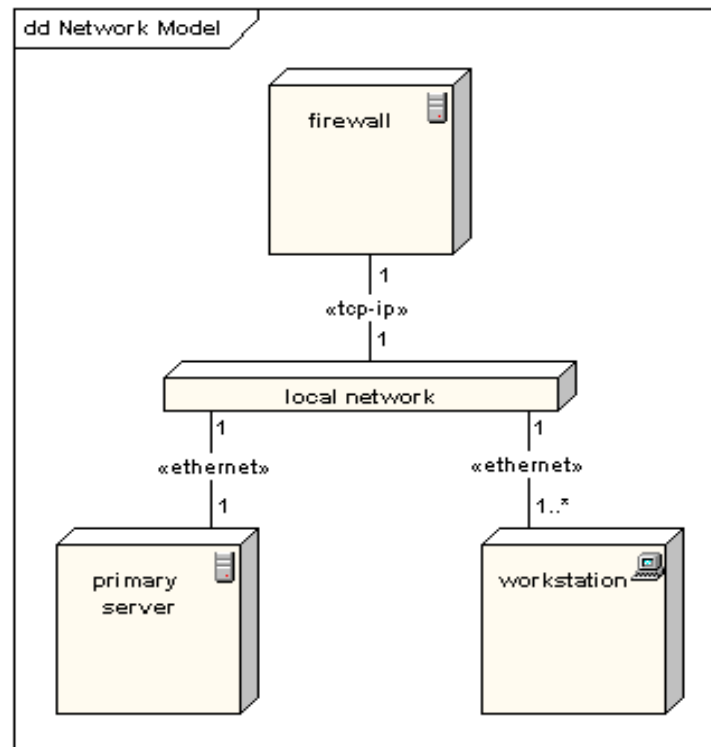
- Any product of the software development process: source files, executables, design documents, test reports, prototypes, user manuals, etc.
- An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon.
- Appear in Deployment diagrams as contents of the nodes on which they are deployed



Deployment diagram

□ Association

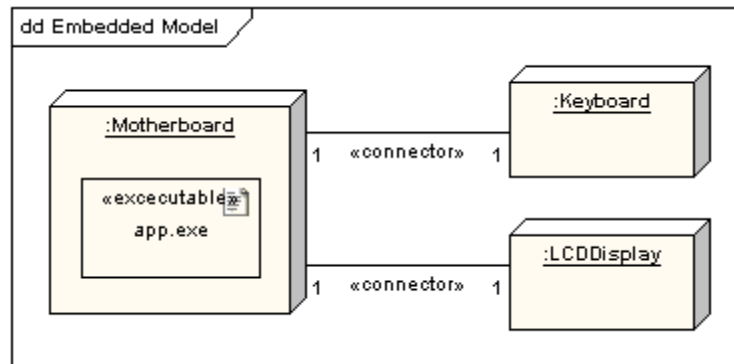
- An association represents a communication path between nodes



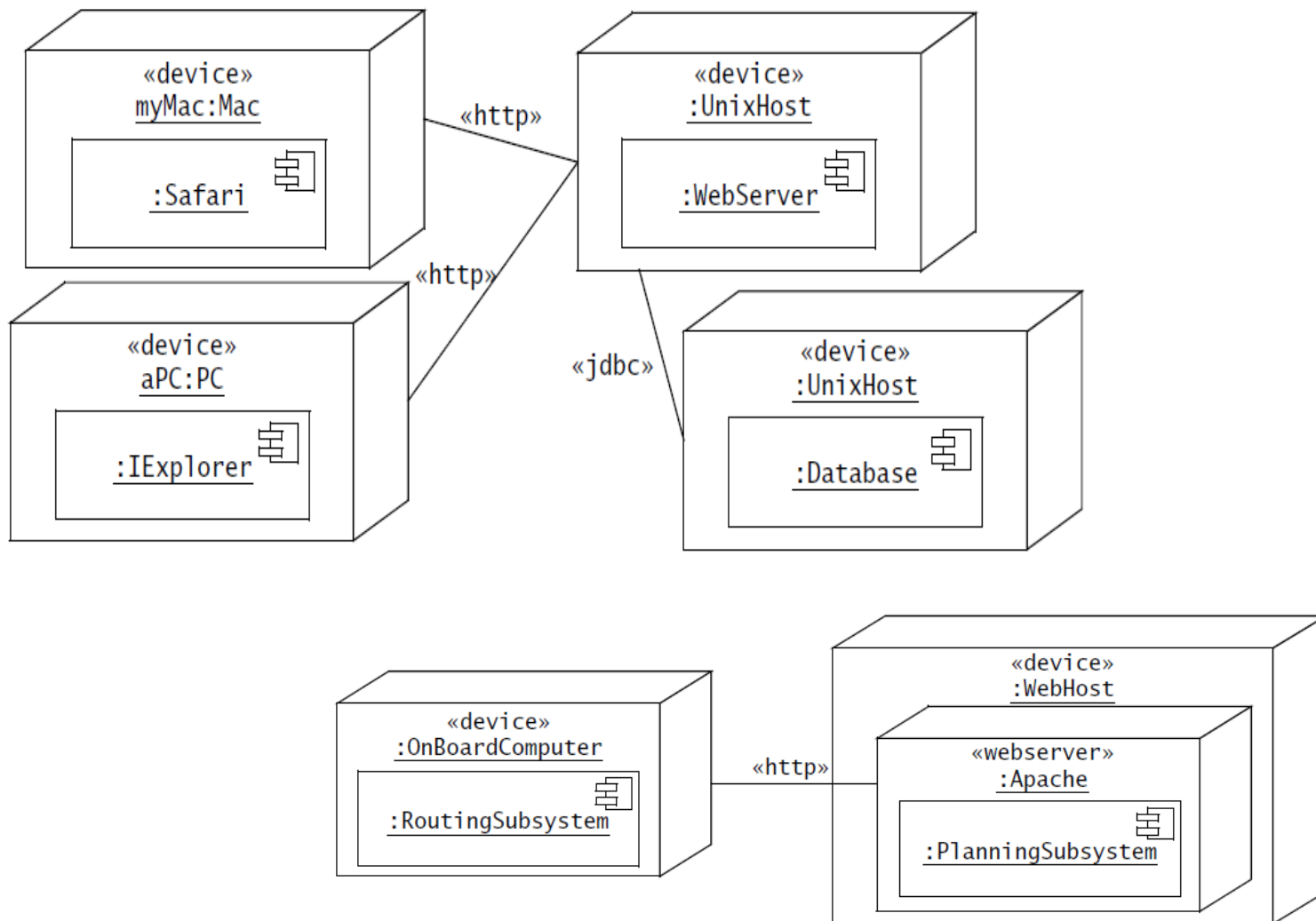
Deployment diagram

□ Node as Container

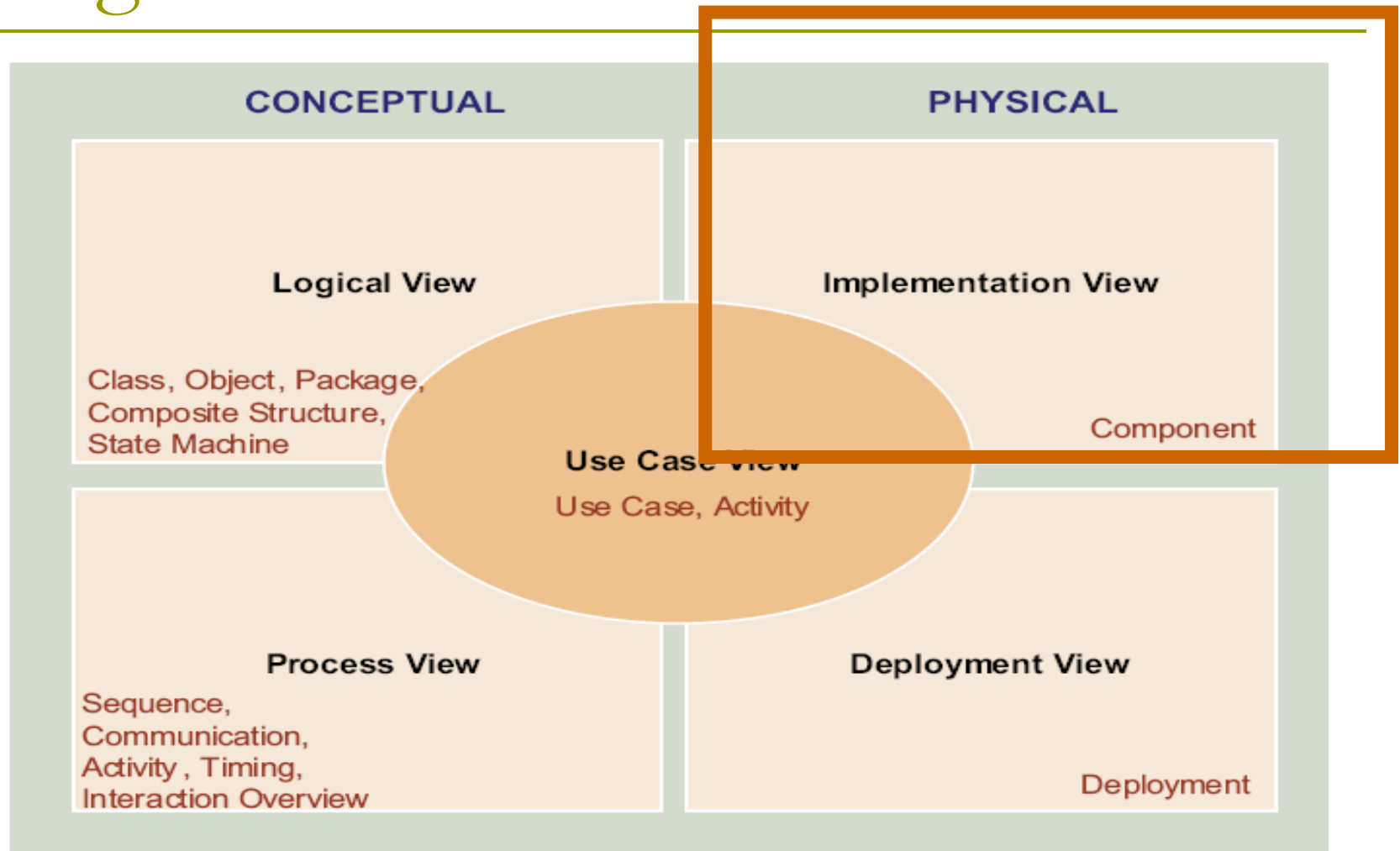
- A node can contain other elements, such as components or artifacts.



Deployment diagram



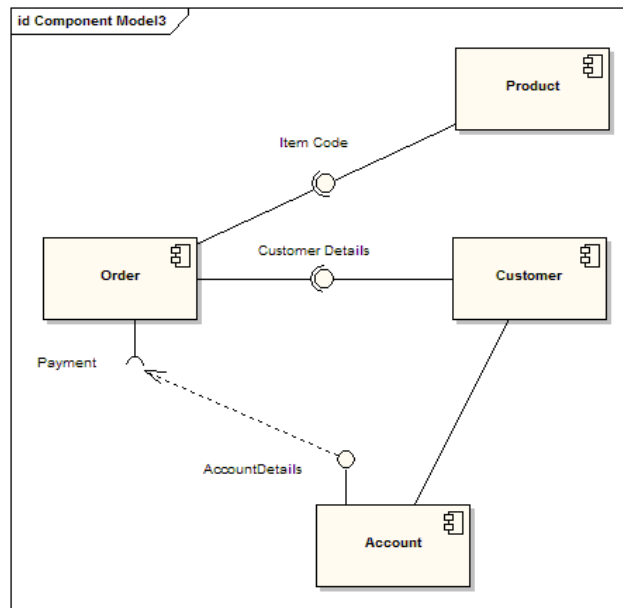
Diagrams and views



Implementation View

Component diagrams

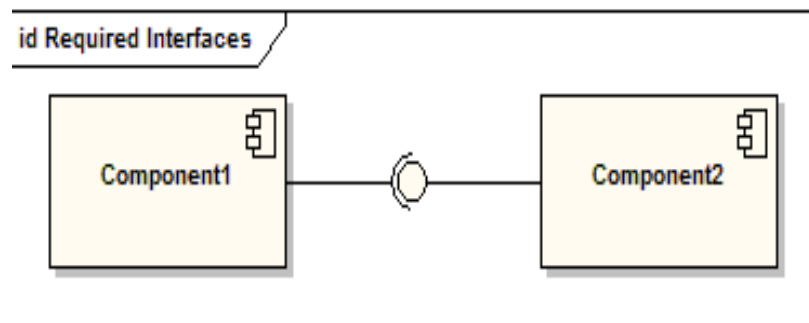
- ❑ Component diagrams illustrate the pieces of **software, embedded controllers**, etc., that will make up a system.
- ❑ A component is usually implemented by one or more classes (or objects) at runtime.



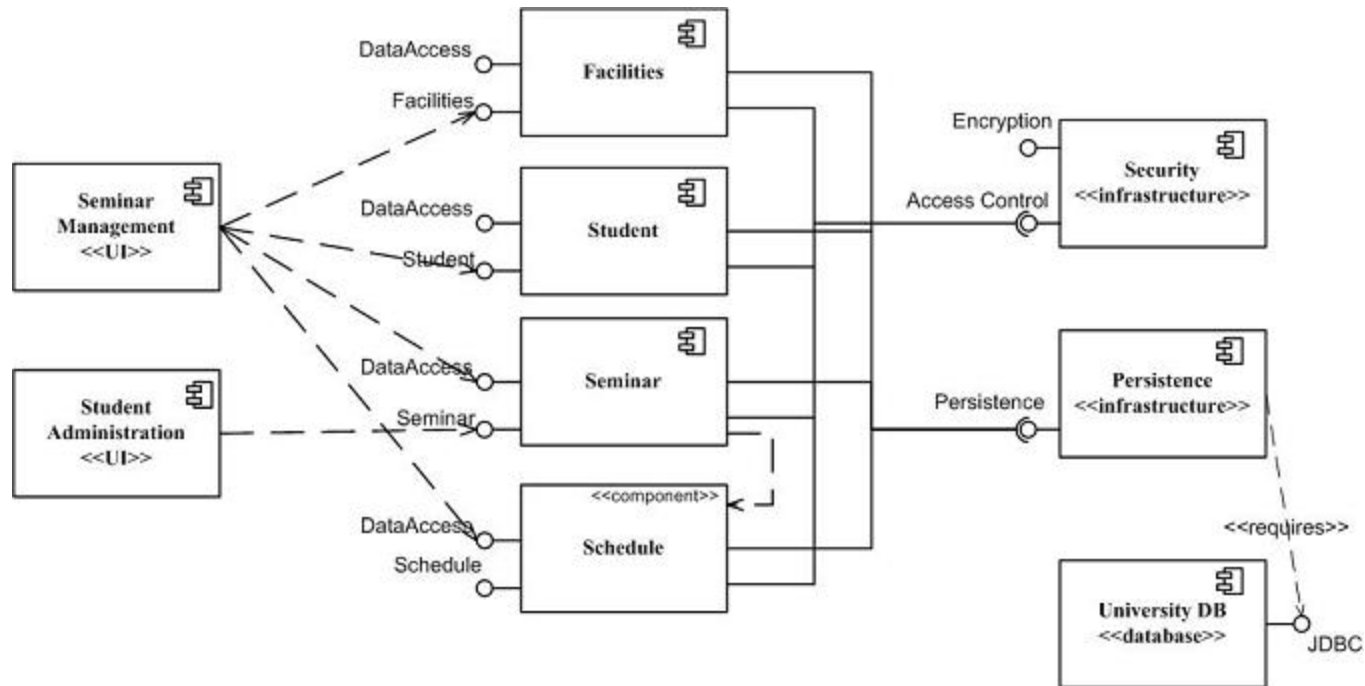
Component diagrams (cont.)

□ Assembly Connector

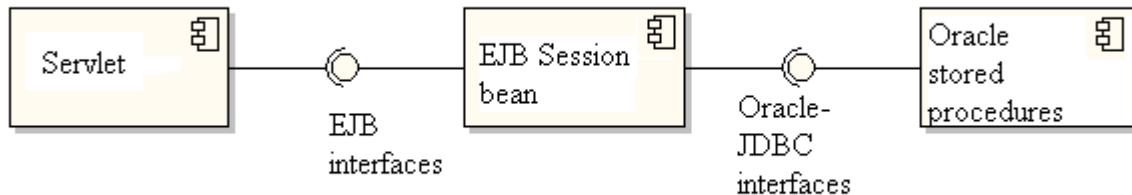
- The assembly connector bridges a component's **required interface** (Component1) with the **provided interface** of another component (Component2); this allows one component to provide the services that another component requires.



Component diagrams (cont.)



Component diagrams (cont.)



Components can be refined to include information about the interfaces they provide and the classes they contain

