

CSIT314 Software Development Methodologies



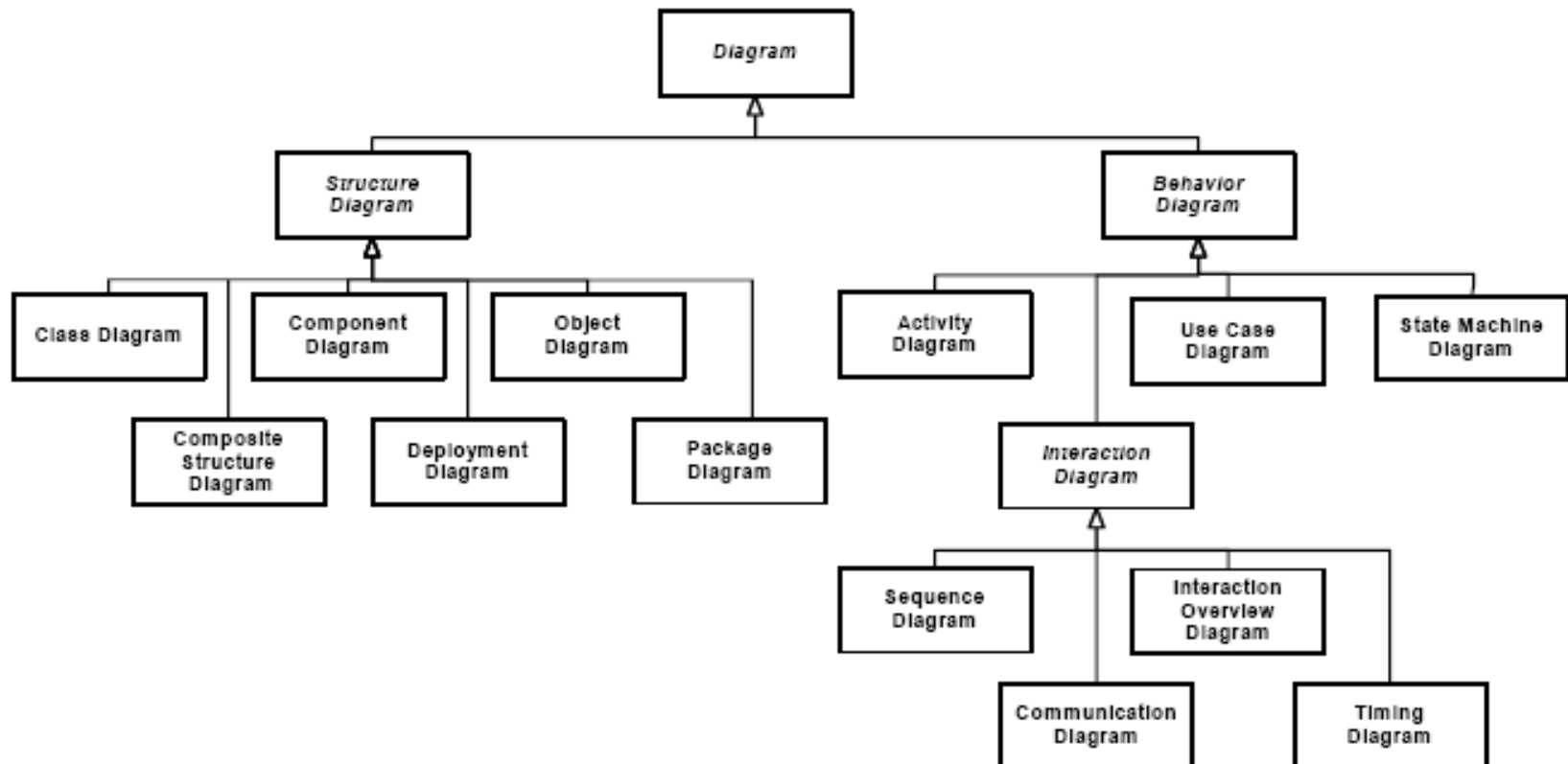
UML modelling – Part 2

Overview of Lecture

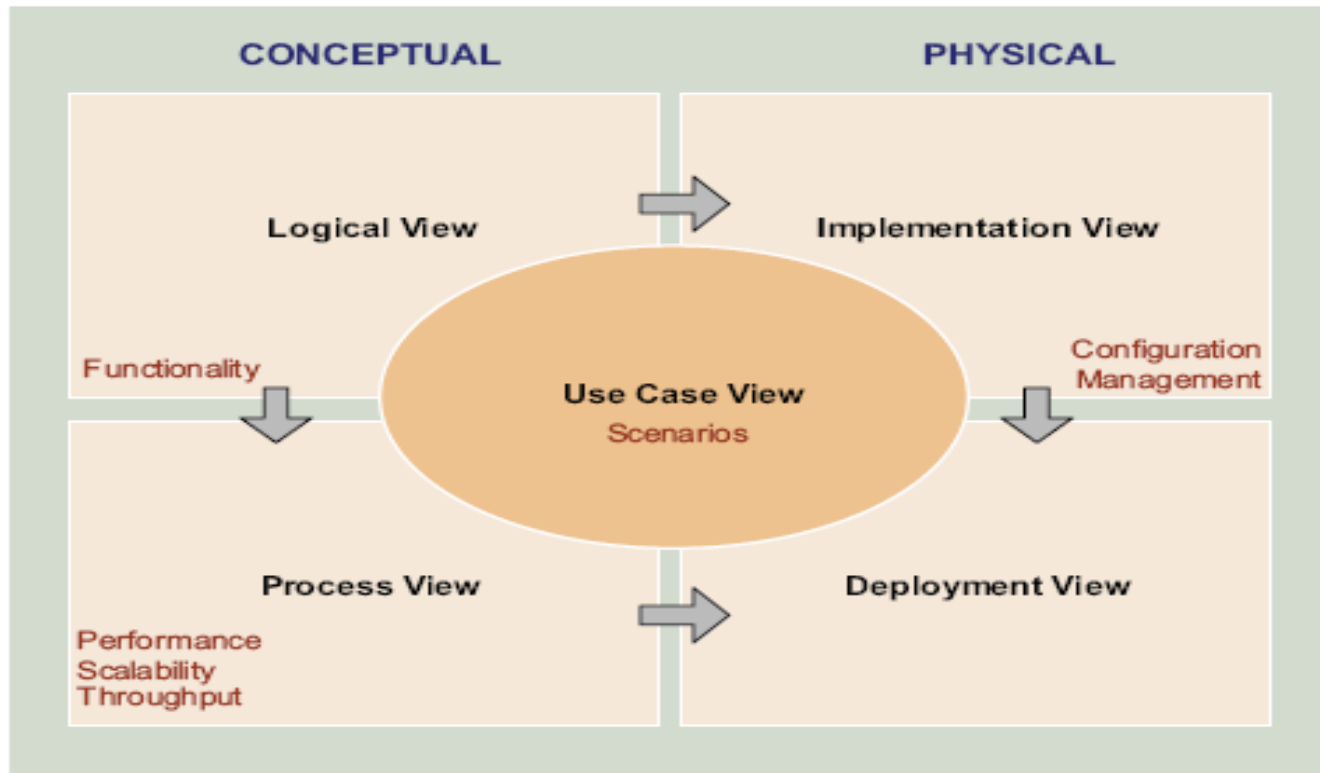
- ❑ UML reminder
- ❑ The "4 + 1" view
- ❑ Logical view
 - Packages
 - State diagrams
- ❑ Process view
 - Communication diagrams
 - Activity diagram
 - Interaction Overview Diagrams
- ❑ Deployment view
 - Deployment diagrams
- ❑ Implementation view
 - Component diagrams
- ❑ UML diagrams and RUP

UML-2 diagrams

- ❑ OMG's classification of UML-2 diagrams:



“4+1” View



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

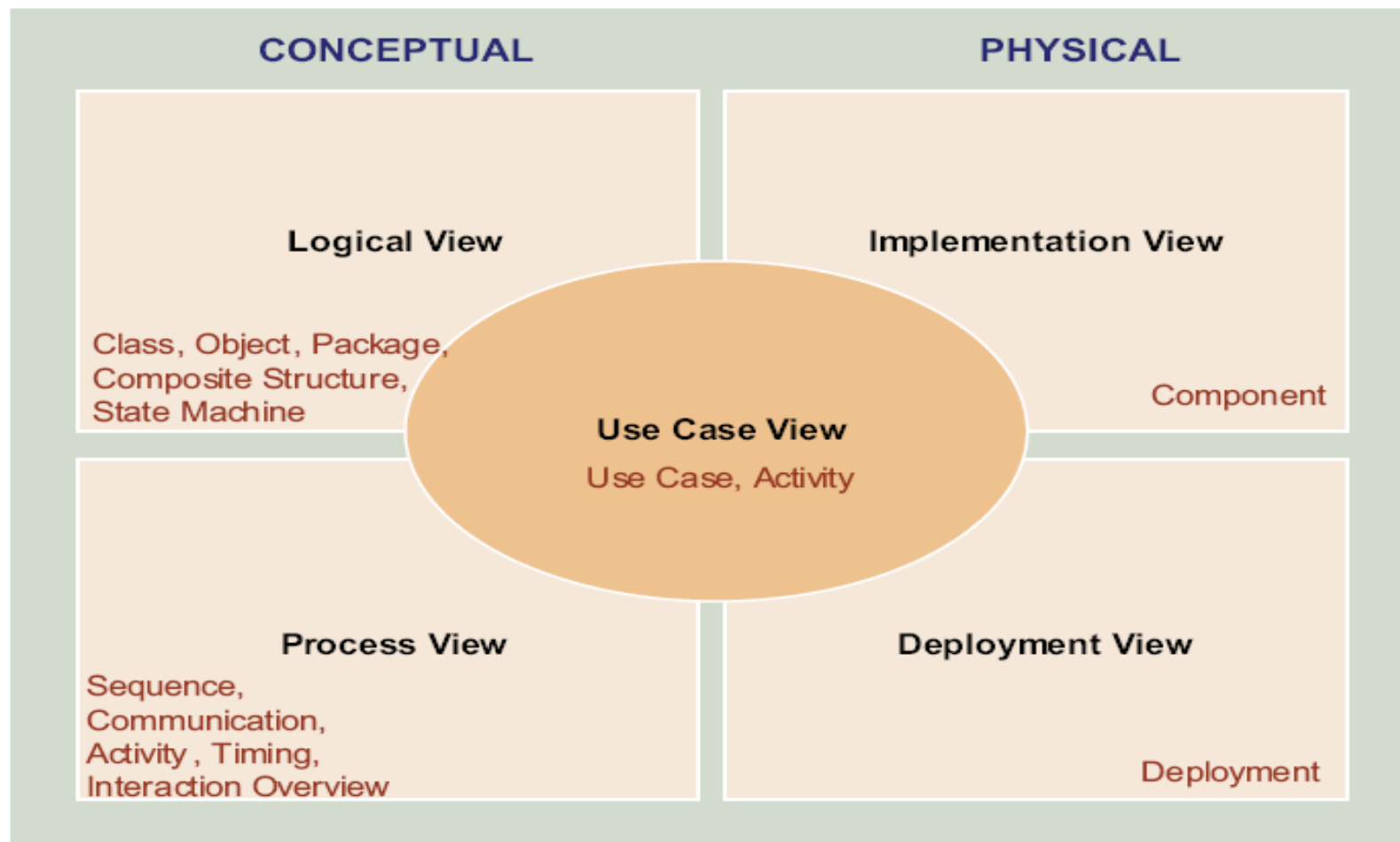
More information on the “4+1” view is available at:

<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

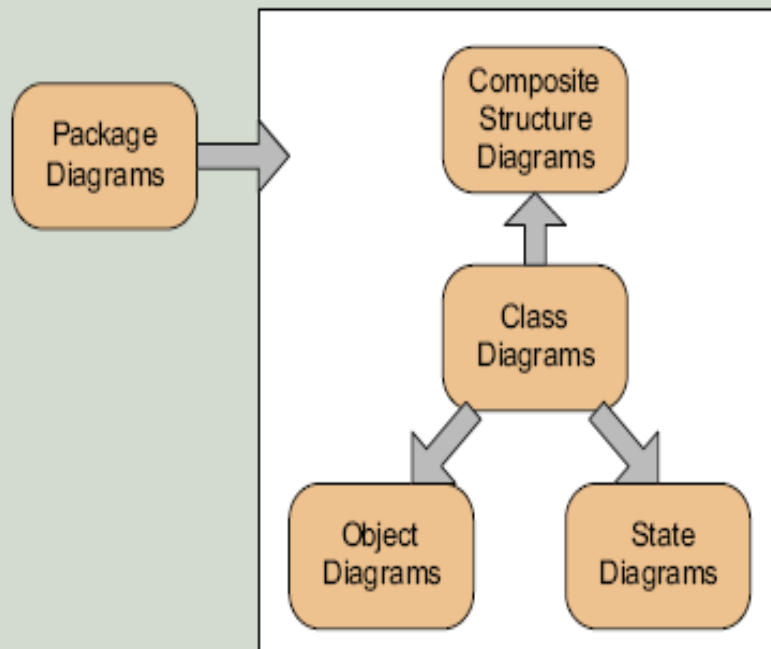


What did we cover?

- “Use case” from the “Use case view”
- “Class” and “object” from the “Logical view”
- “Sequence” from the “Process view”
- Now we continue exploring a little more!

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

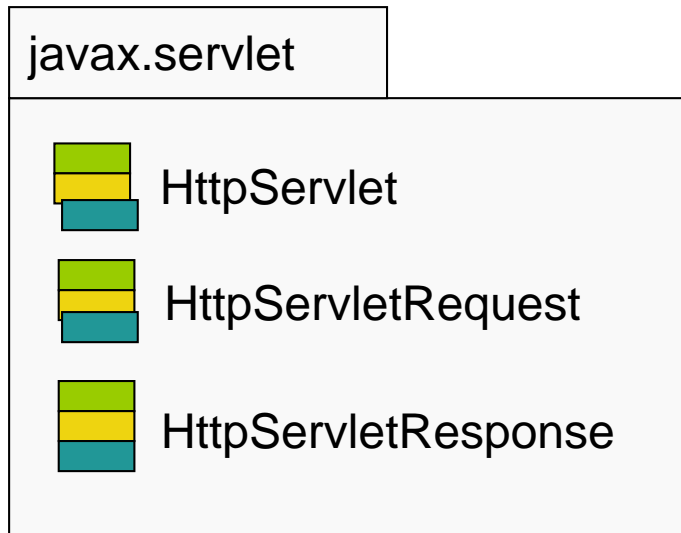
3. Object diagrams when relationships between classes need to be explained through instances
4. State Charts when internal states of a specific class are to be explained
5. Composite Structures when parts of a class and relationships between parts are to be modeled

Logical View

Package

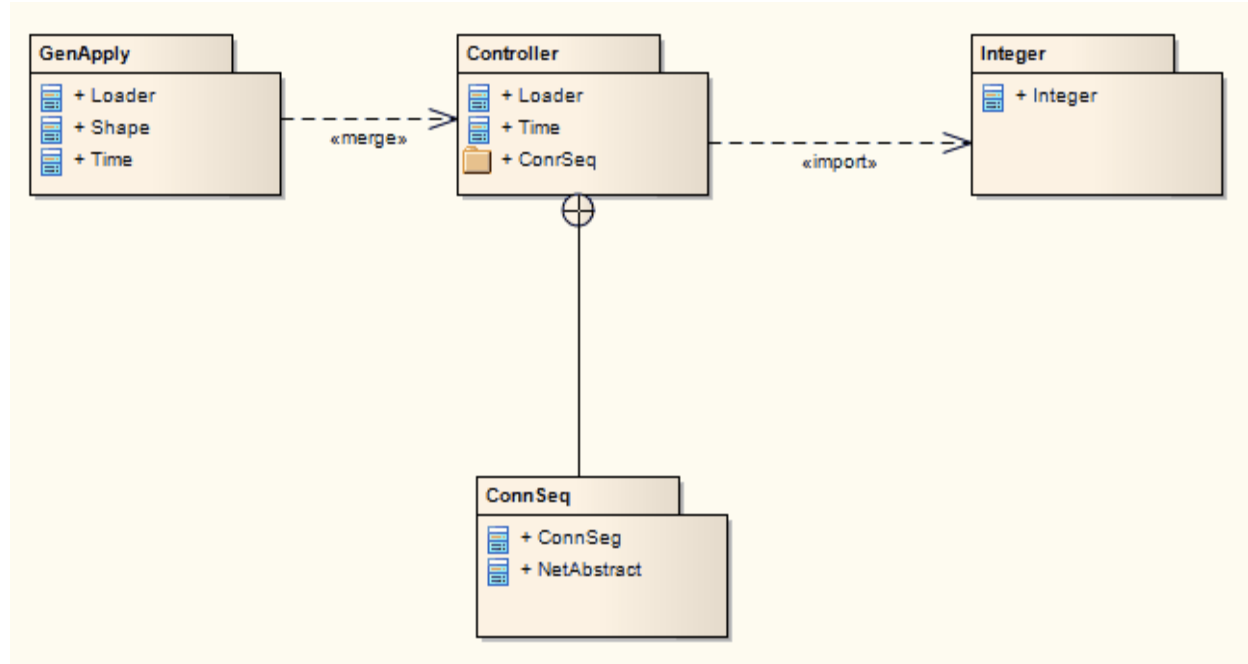
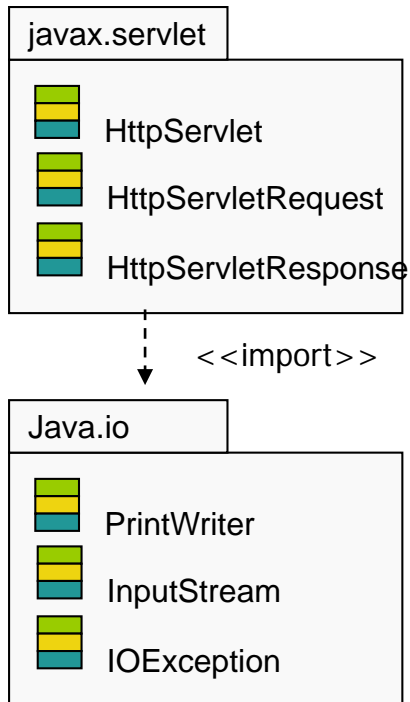
□ Easy –

- 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



Package

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

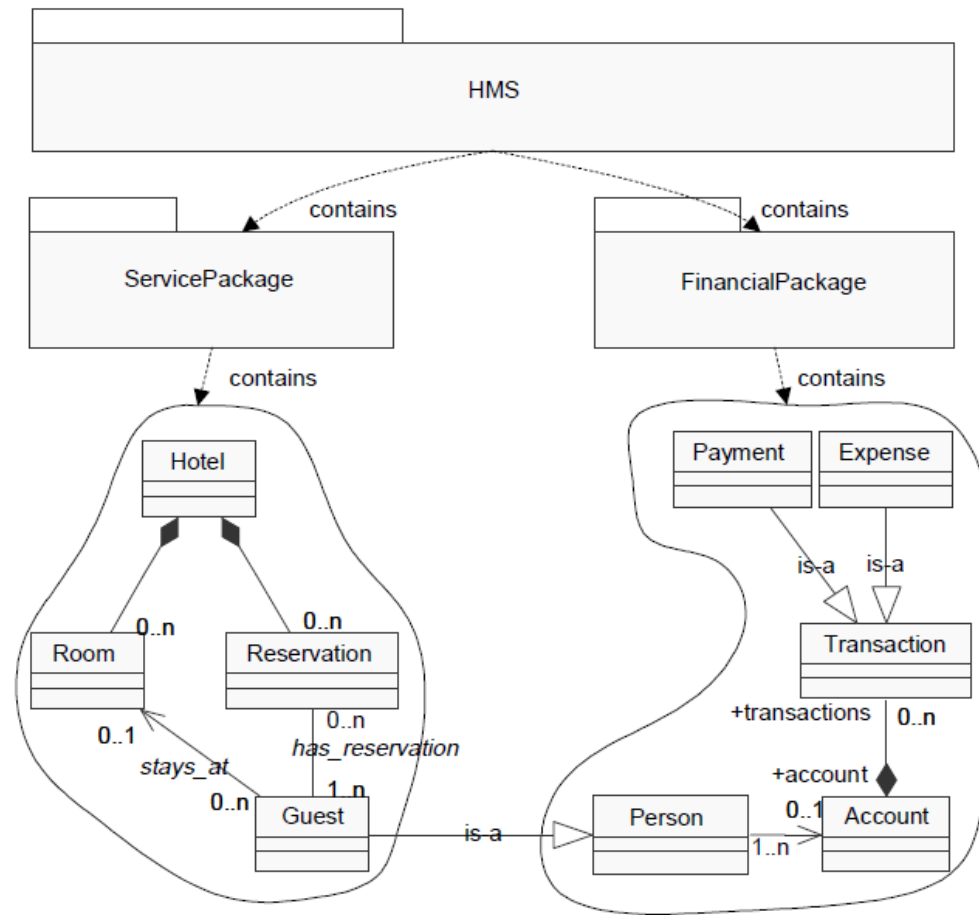


Why do you want package diagrams?

- ❑ 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

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

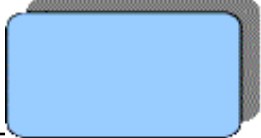



- ❑ What is a State diagram?
 - It is a dynamic model that shows the different states that a single object can pass during its lifetime in response to events.

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.	

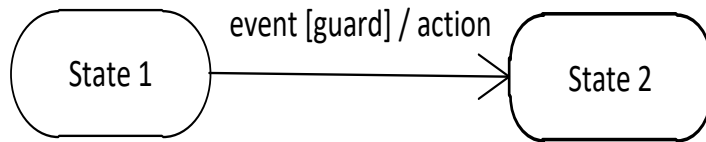
Transitions

- ▣ 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.

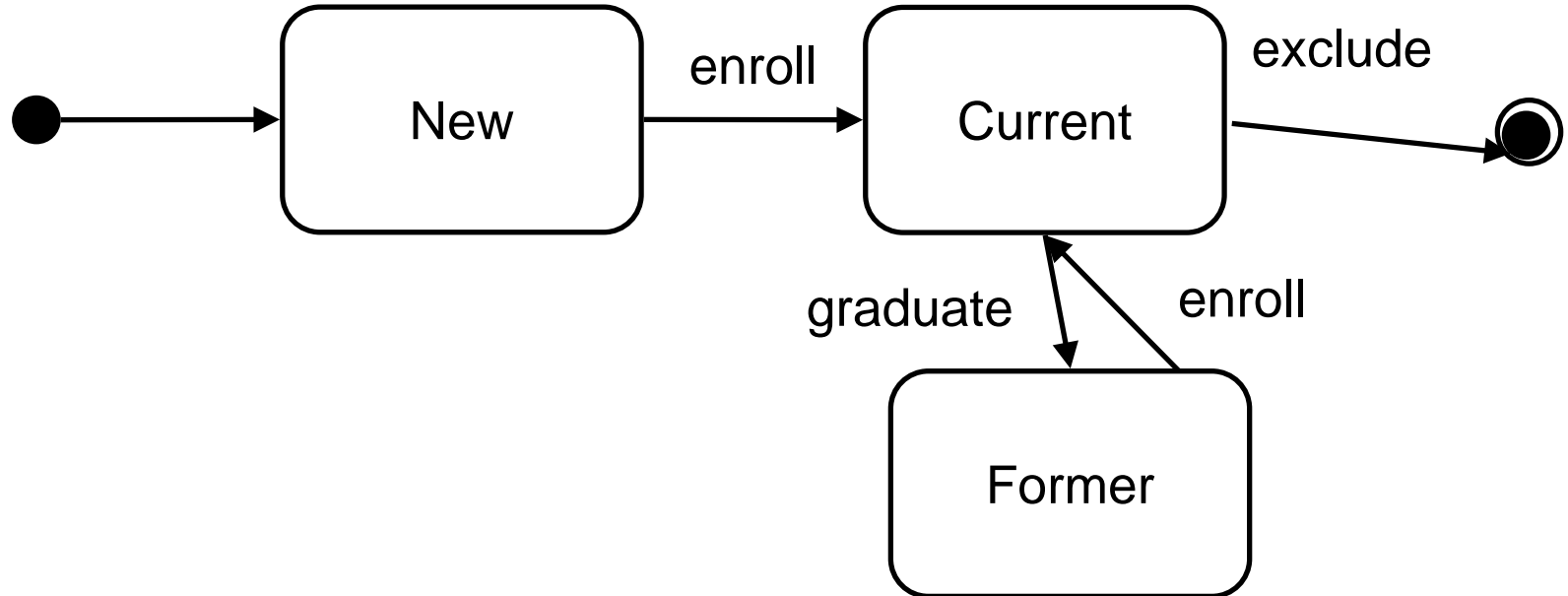
State Machine Notations



Note: guard and action are optional

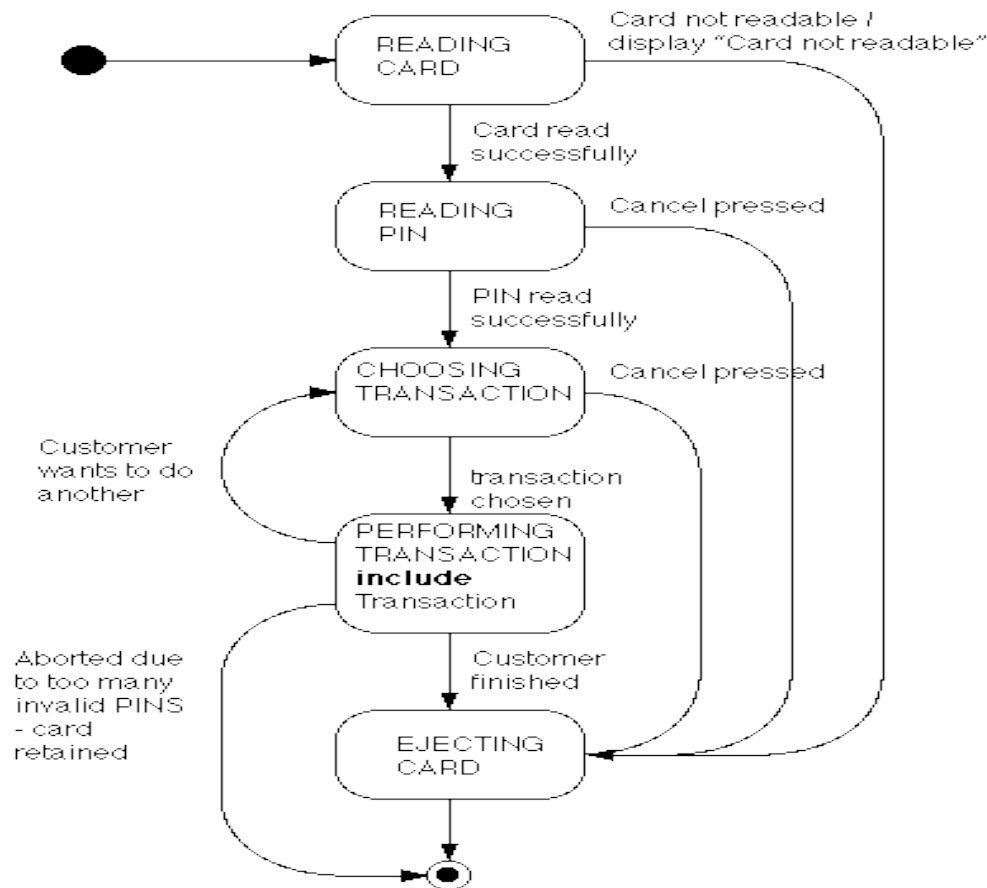
Simple State Diagram

- Different states of a student



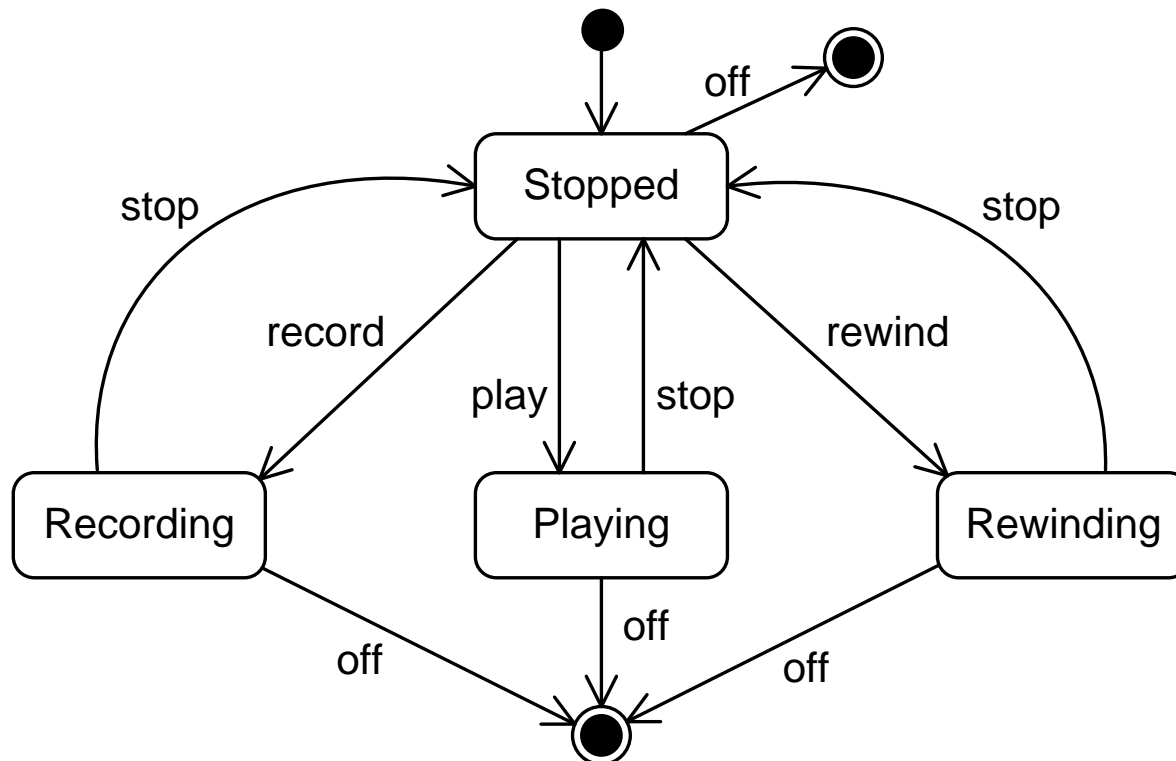
Example: ATM machine

State-Chart for One Session

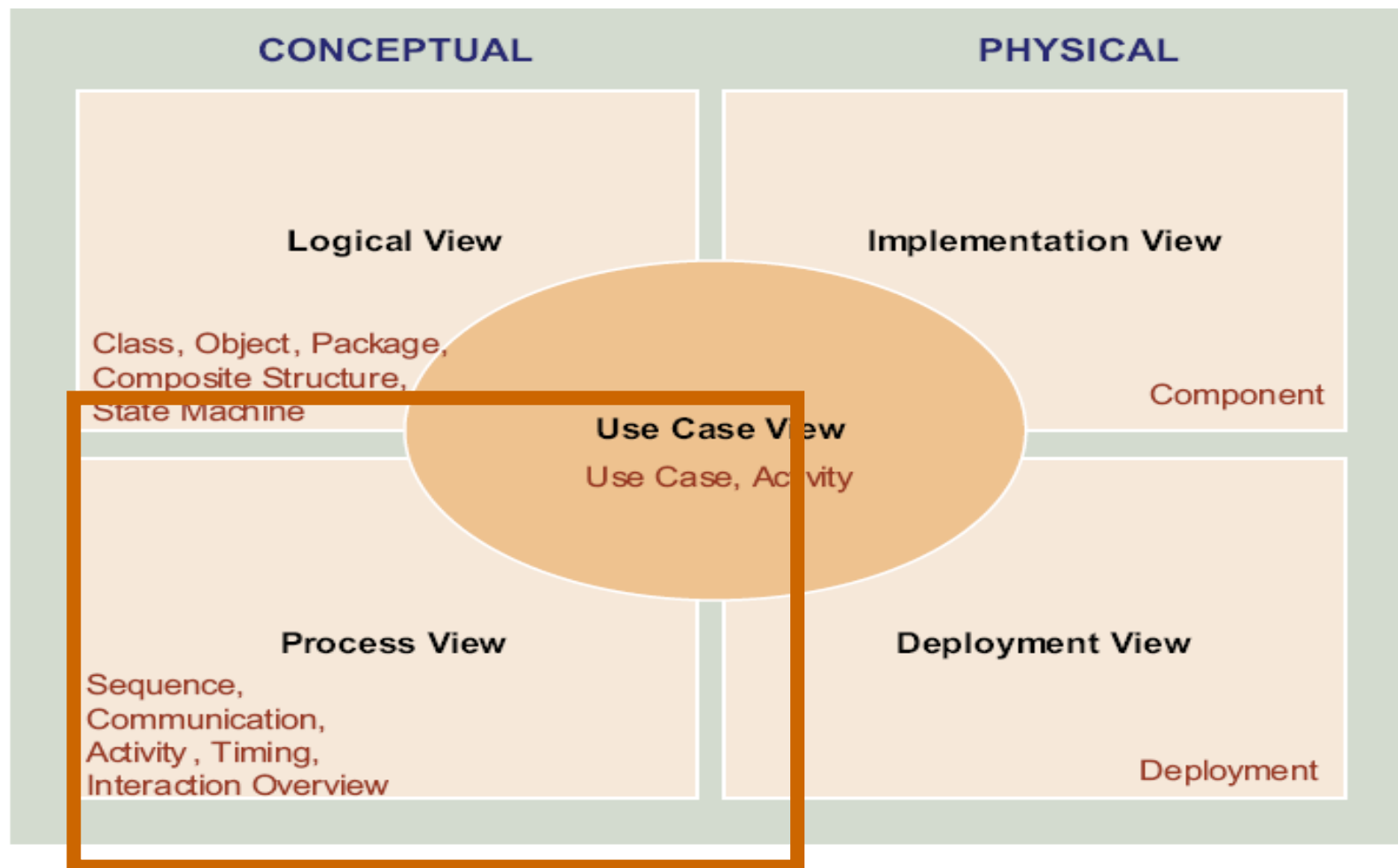


Pen and Paper exercise

Draw a state diagram for a tape recorder

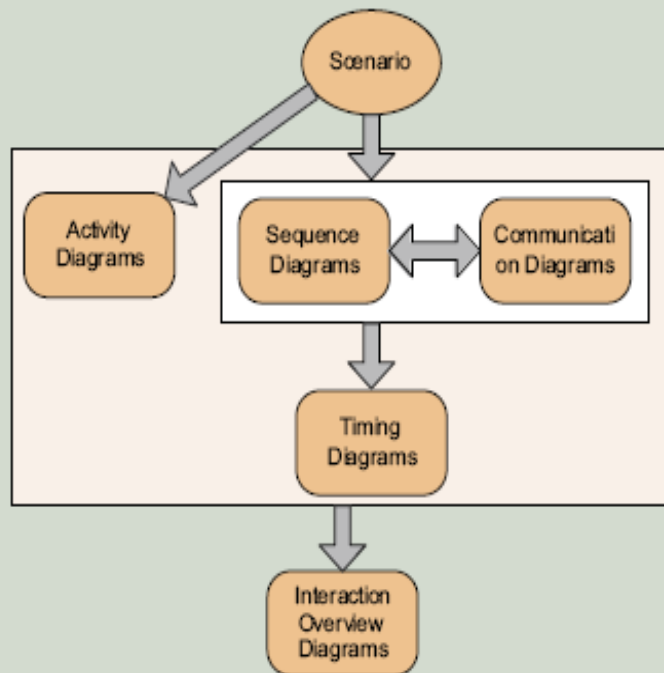


Diagrams and views



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

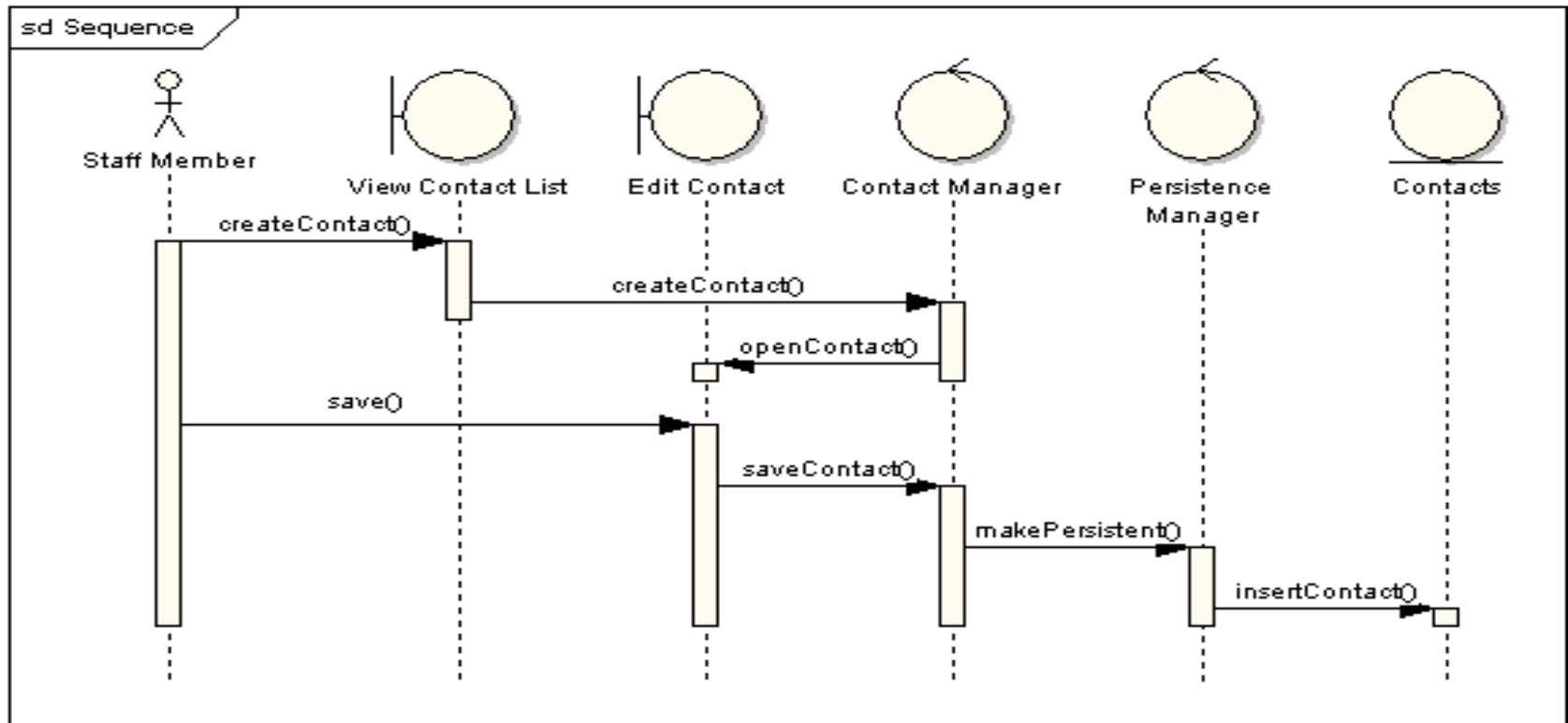
- ❑ We covered sequence diagrams
 - Those wonderful diagrams showing objects, their lifelines, and the sequence of method invocations
 - Also the more preliminary planning versions of sequence diagrams where you work with crude categories like boundary, processing, and entity elements
- ❑ So we have lots more to cover.

Process View

Communications diagrams

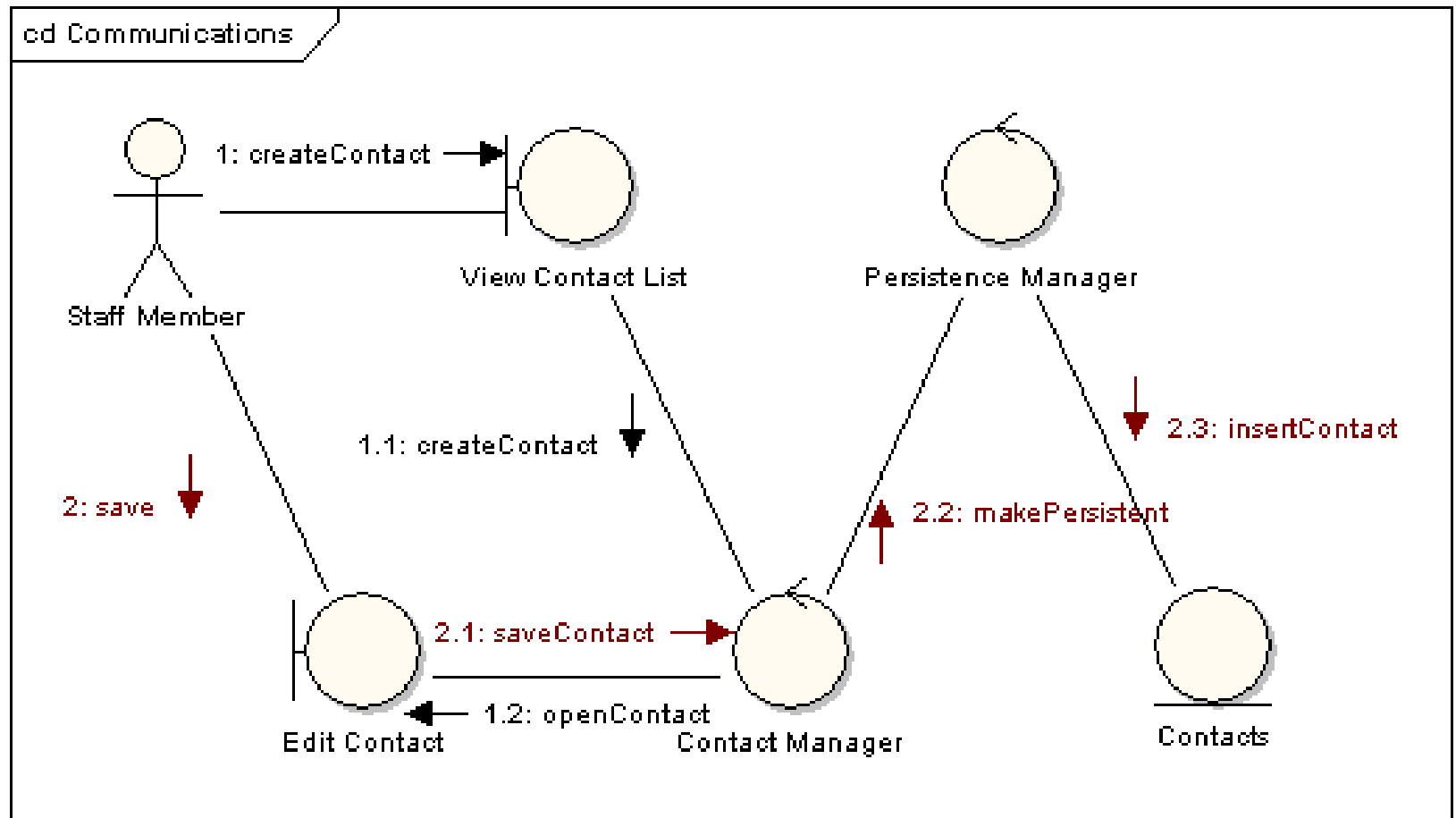
- ❑ (In UML-1, called “collaboration diagrams”)
- ❑ Convey the same information as a sequence diagram – a difference in style.
 - Sequence diagrams focus on messages occurring over a timeline
 - Communication diagrams focus more on links between participating objects
 - ❑ Sequence of messages is still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

Sequence diagram vs. Communications diagrams



An example of sequence diagram

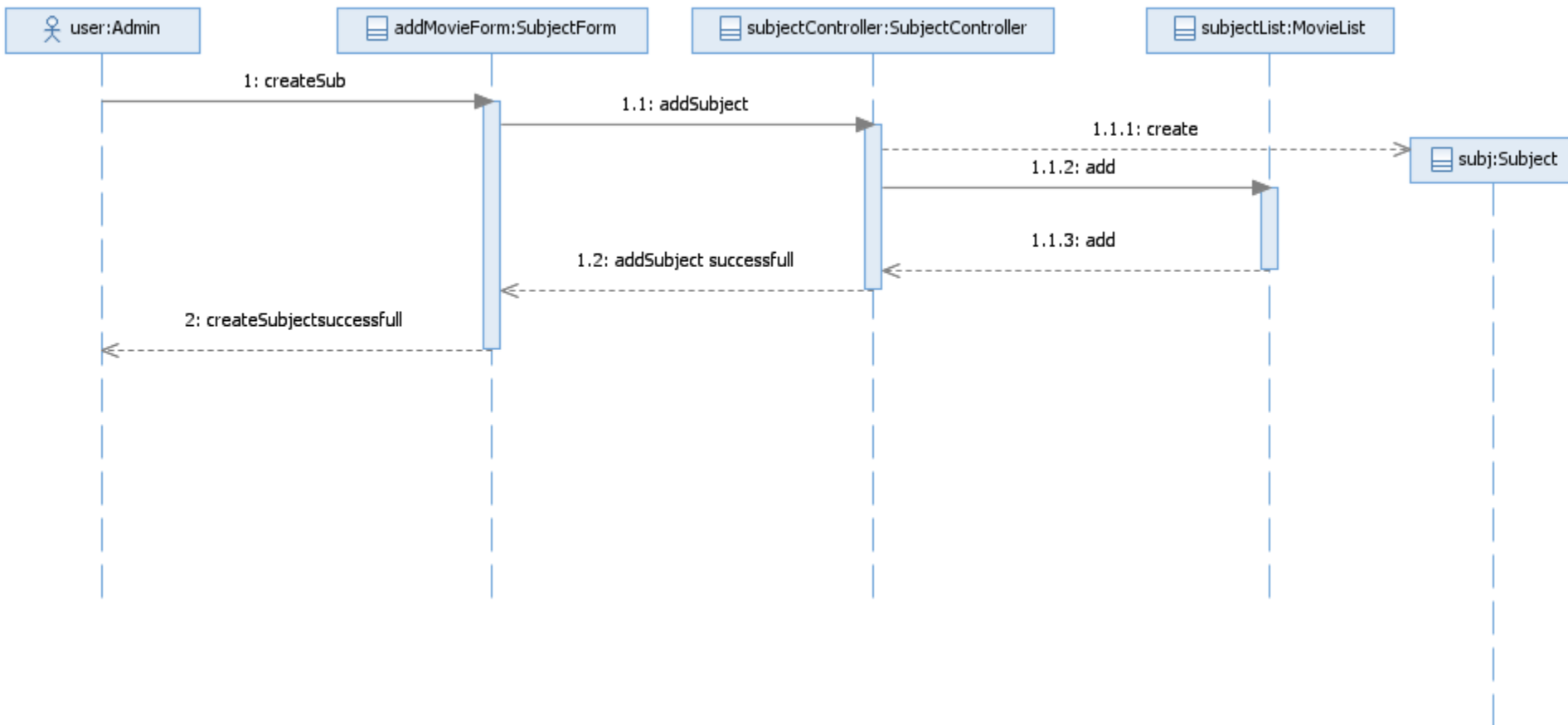
Sequence diagram vs. Communications diagrams



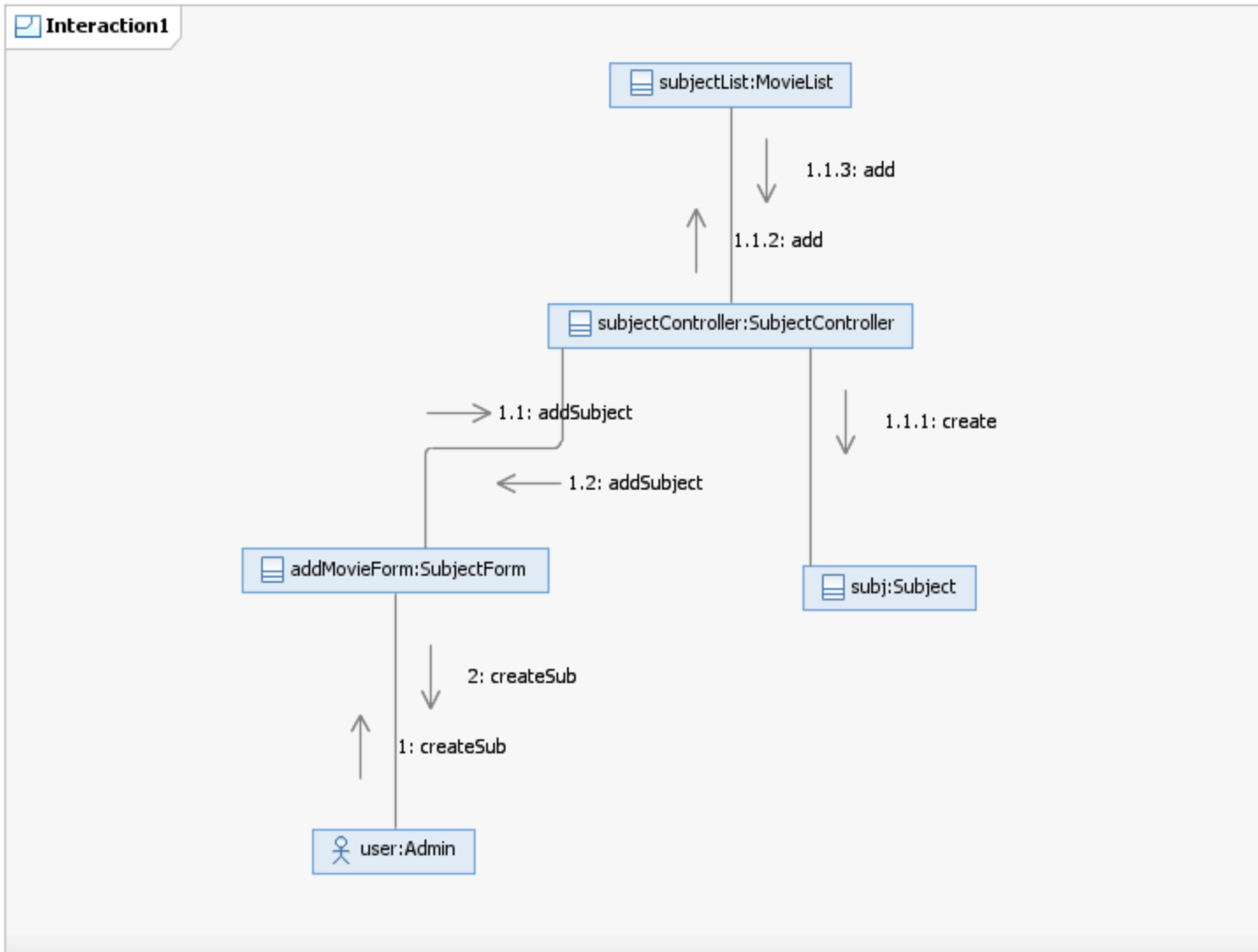
An example of communications diagram

Pen and Paper exercise

- Draw a communication diagram corresponding to the following sequence diagram



Pen and Paper exercise



Process View

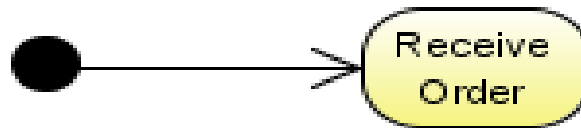
Activity diagrams

- ❑ Activity diagrams are used to model the workflow behind the system.
- ❑ Activity are also useful for:
 - Analyzing a use case by describing what actions need to take place and when they should occur
 - Describing a complicated sequential algorithm; and modelling applications with parallel processes

Elements in activity diagrams

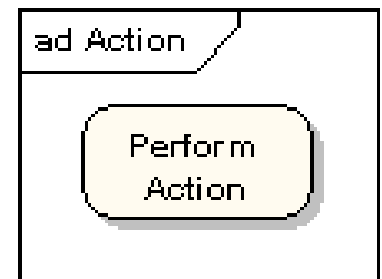
❑ Initial node

- (Complex activity diagrams can have multiple initial nodes)



❑ Action

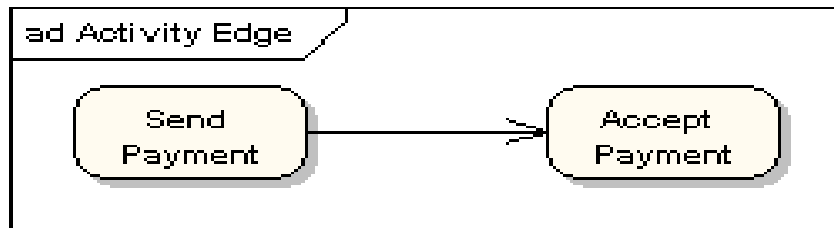
- Incoming activity edges
 - ❑ control flow and data flow from other nodes.
 - ❑ Action starts when all of its input conditions are satisfied.
- Outgoing activity edges
 - ❑ When action completes it sends its outputs via outgoing edges to successor nodes



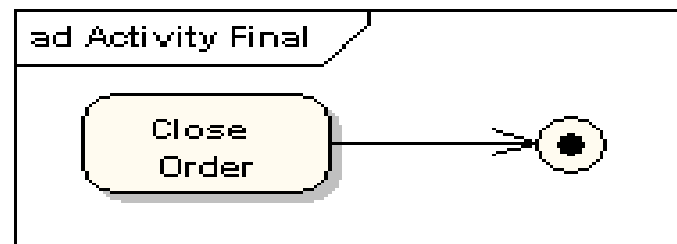
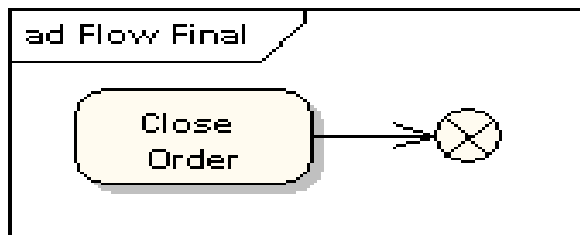
Many of the elements are the same as those used in "State" diagrams.

Elements in activity diagrams -2

□ Control flow



- Final nodes – two kinds “flow final” “activity final”
(flow final terminates one parallel flow, activity final implicitly terminate all flows)



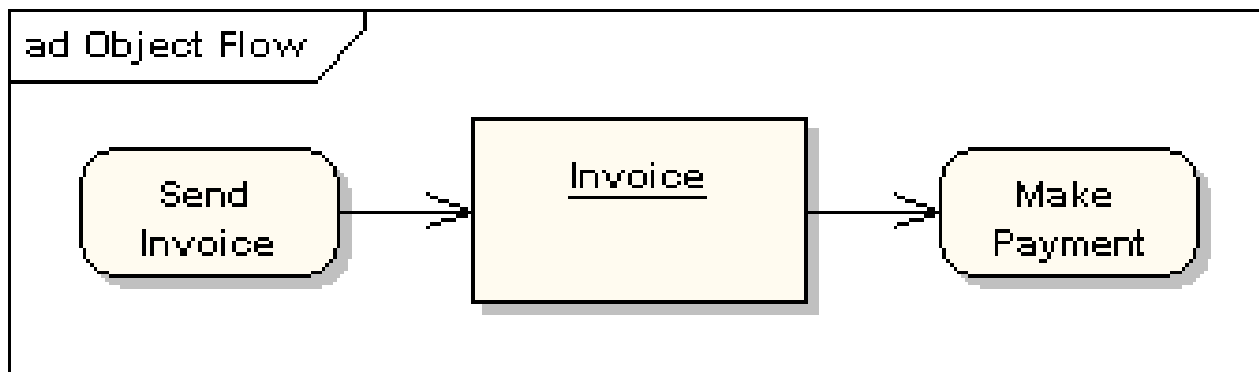
Elements in activity diagrams -3

❑ ObjectNode

- Rectangle named with object type
- Indicates such an object becomes available at some point in activity

❑ ObjectFlow

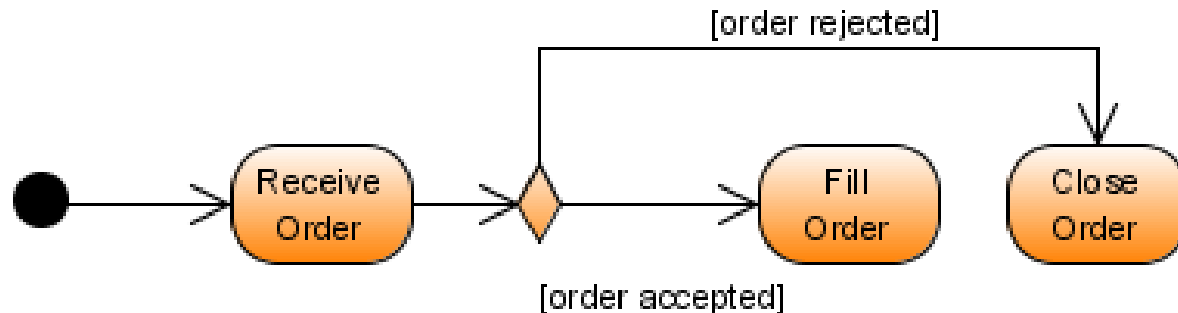
- An object flow is a path along which objects or data can pass.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.



Elements in activity diagrams-4

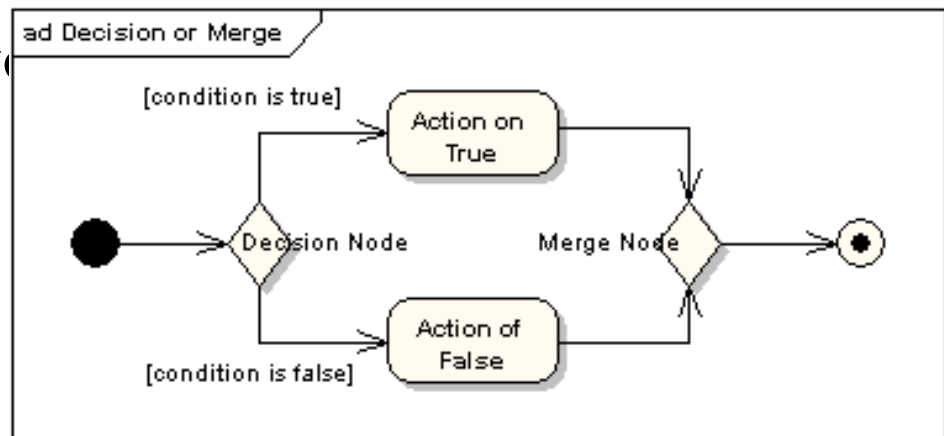
DecisionNode

- A decision node is a control node that chooses between outgoing flows.



MergeNode

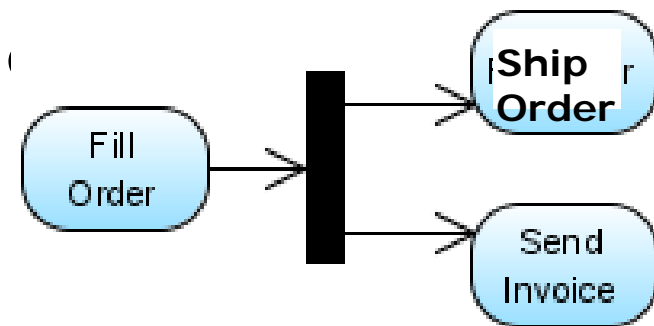
- Rejoin the alternative decision node



Elements in activity diagrams-5

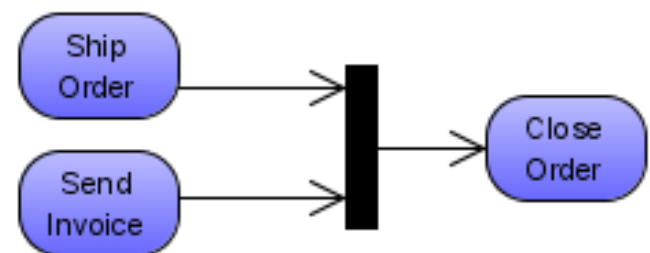
□ ForkNode

- A fork node has one incoming multiple outgoing edges.
- Implies parallel execution follows



□ JoinNode

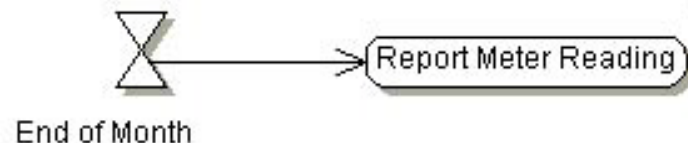
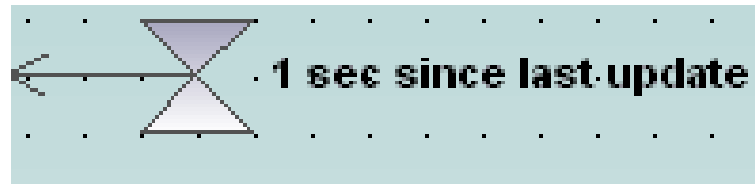
- A join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received.



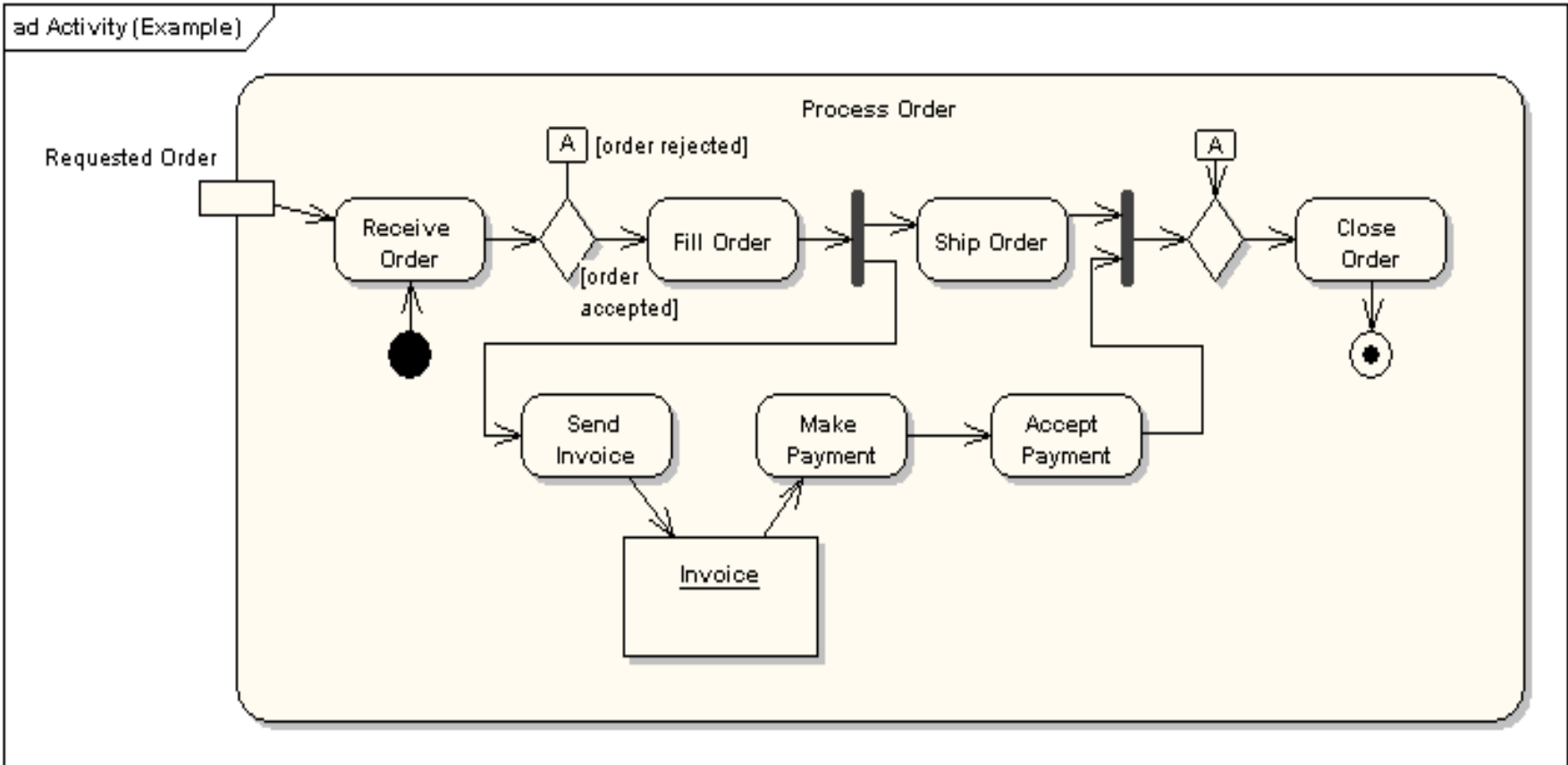
Elements in activity diagrams

□ Timed triggers

- These can appear as sources of input arcs to actions or to “fork” structures in the overall activity



Activity diagram – example



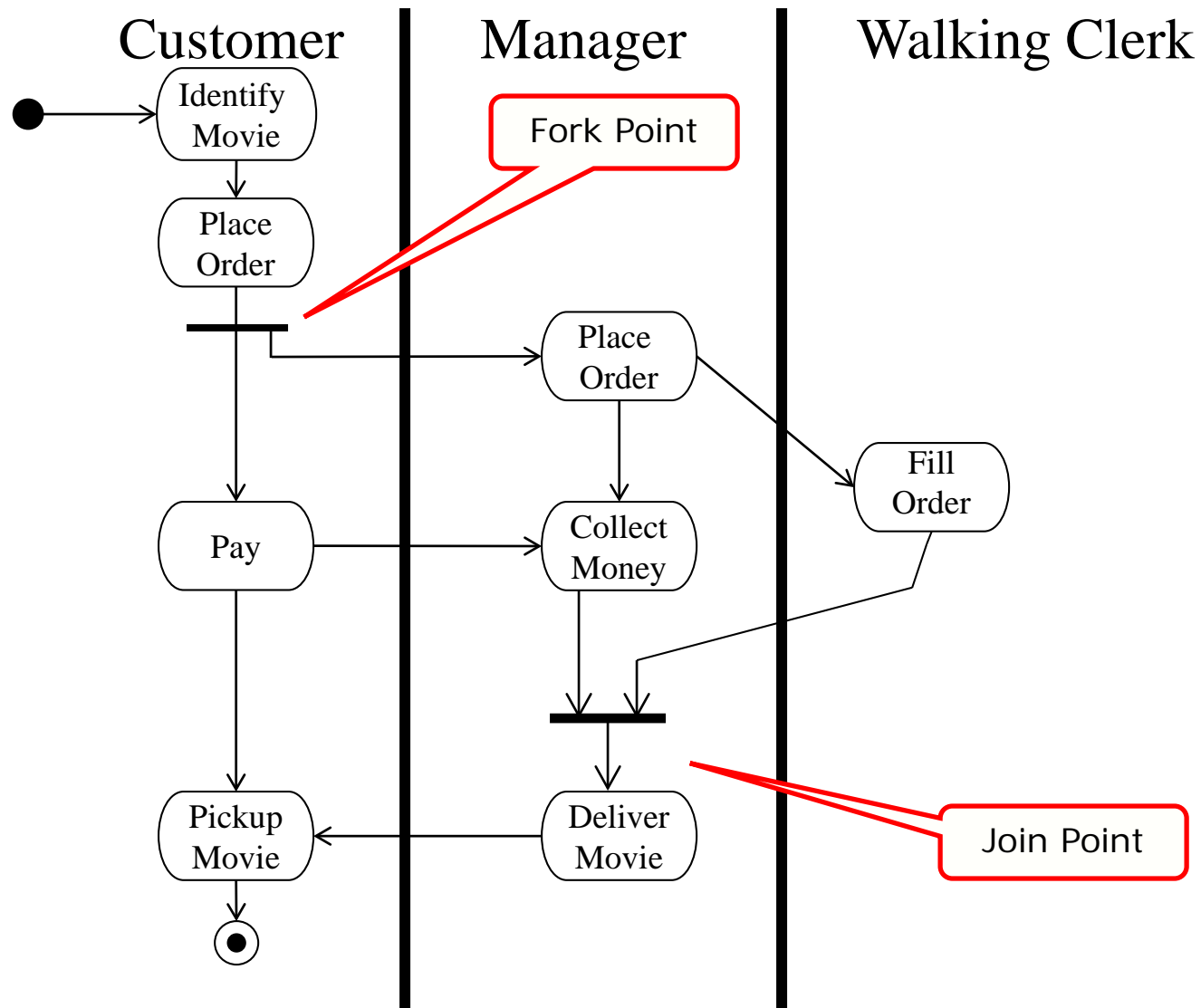
Swimlanes

Source: adapted from Easterbrook, 2006

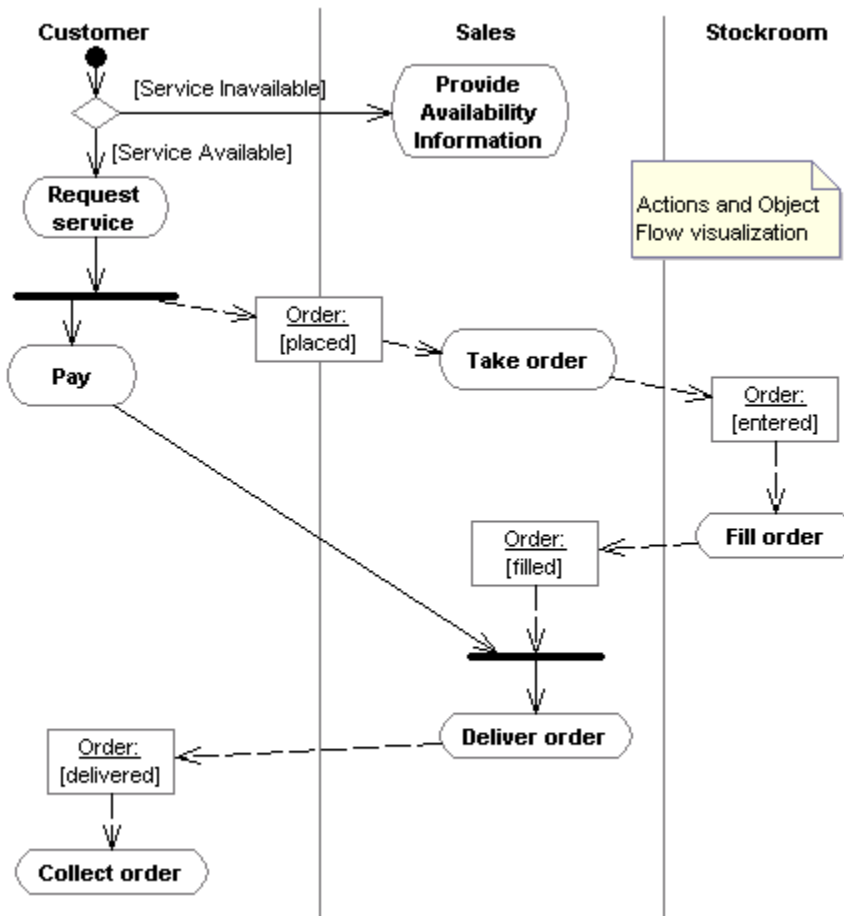
- ❑ Activity diagrams tell you what happens, but they do not tell you who does what
 - From the implementation perspective, this means that the diagram does not convey which class is responsible for which activity
 - From the domain view, this means that the diagram does not show which people or departments are responsible for which activity
- ❑ Swimlanes are a way around this
 - Swimlanes are indicated by vertical dashed lines which separate the diagram into zones
 - Each zone represents a particular class, person or department etc

Swimlanes and Fork/Join Points

Source: adapted from Easterbrook, 2006



Swimlanes – another example



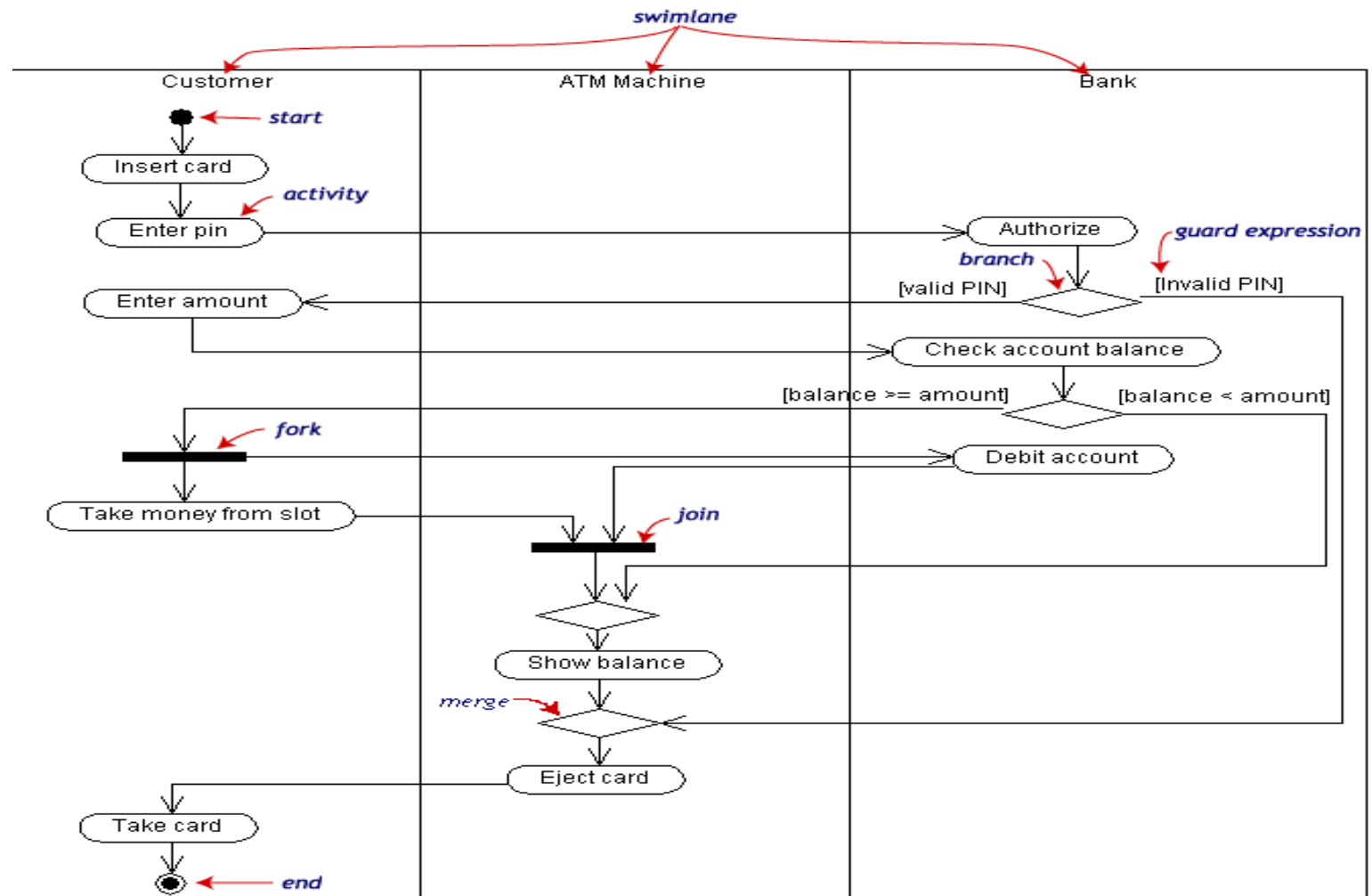
Typically used to show how several "processors" (people or cpus!) may be working concurrently

Pen and paper exercise

- Draw an activity diagram for the process of withdraw money from a bank account through an ATM.
 - This process would involve three entities: customer, ATM and the bank system.

Withdraw money activity

One potential solution

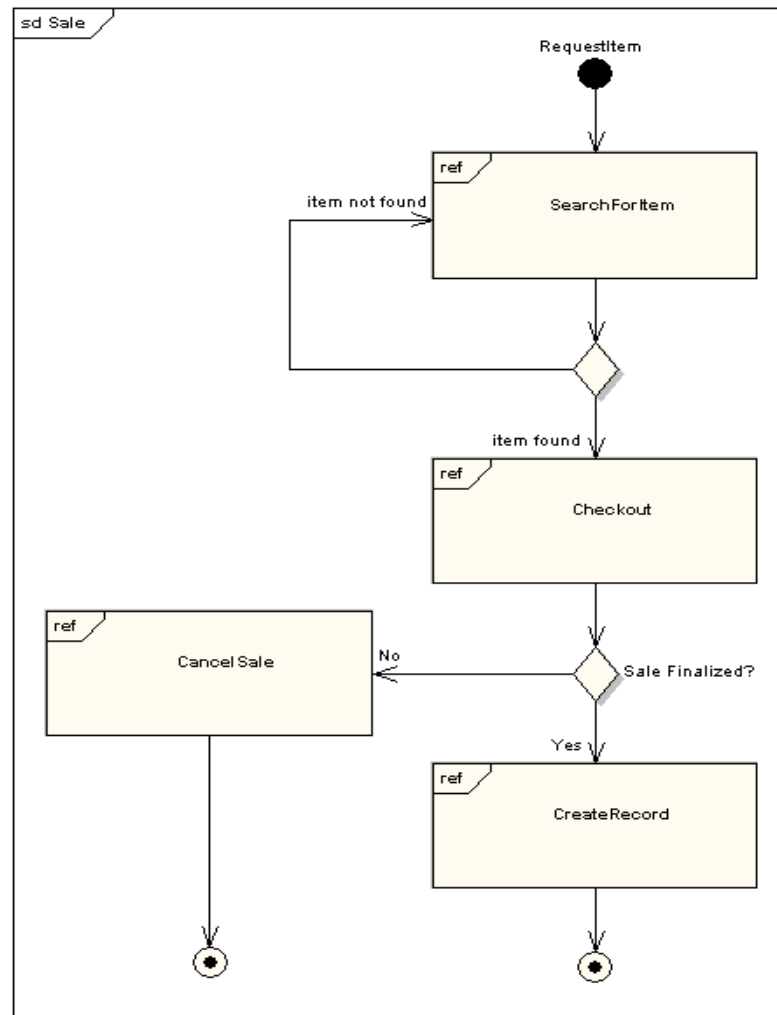


Process View

Interaction Overview Diagrams

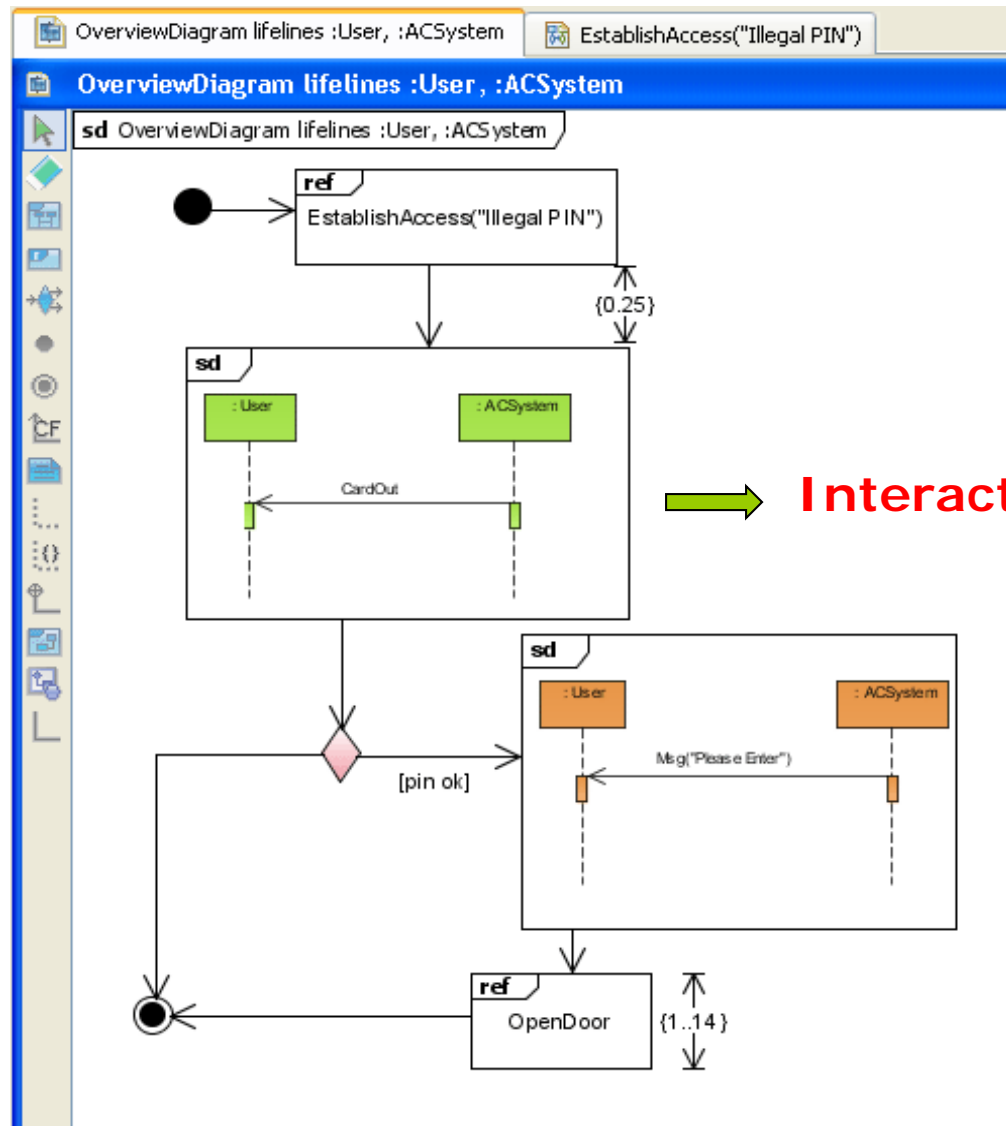
- Interaction overview diagram
 - a form of **activity diagram** in which the nodes represent “interaction diagrams”.
 - “Interaction diagrams”
 - sequence,
 - communication,
 - And interaction overview diagram (recursive definition!).
- New elements:
 - interaction occurrences
 - interaction elements.

Interaction Overview Diagram



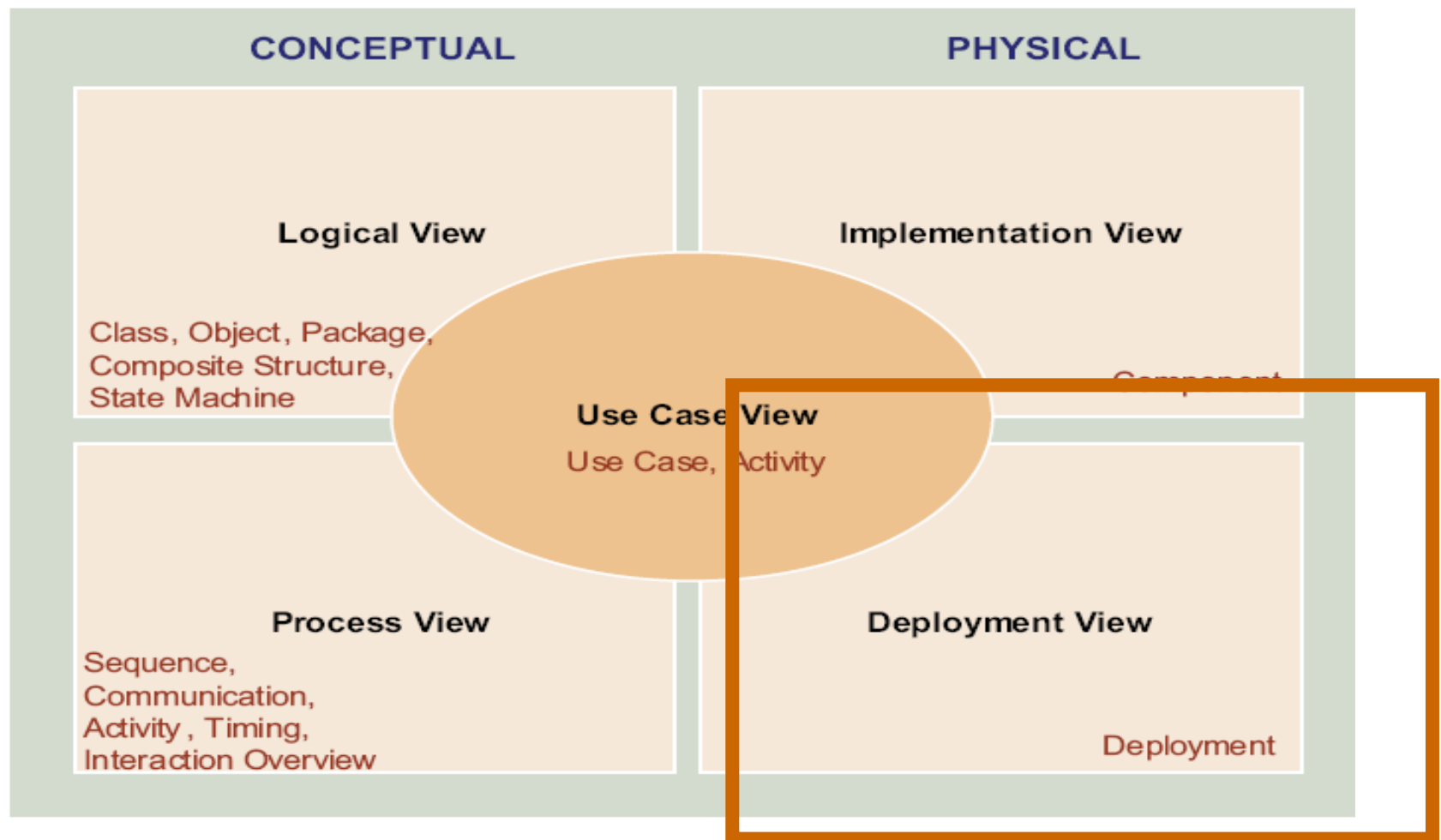
→ Interaction occurrence

Interaction Overview Diagram



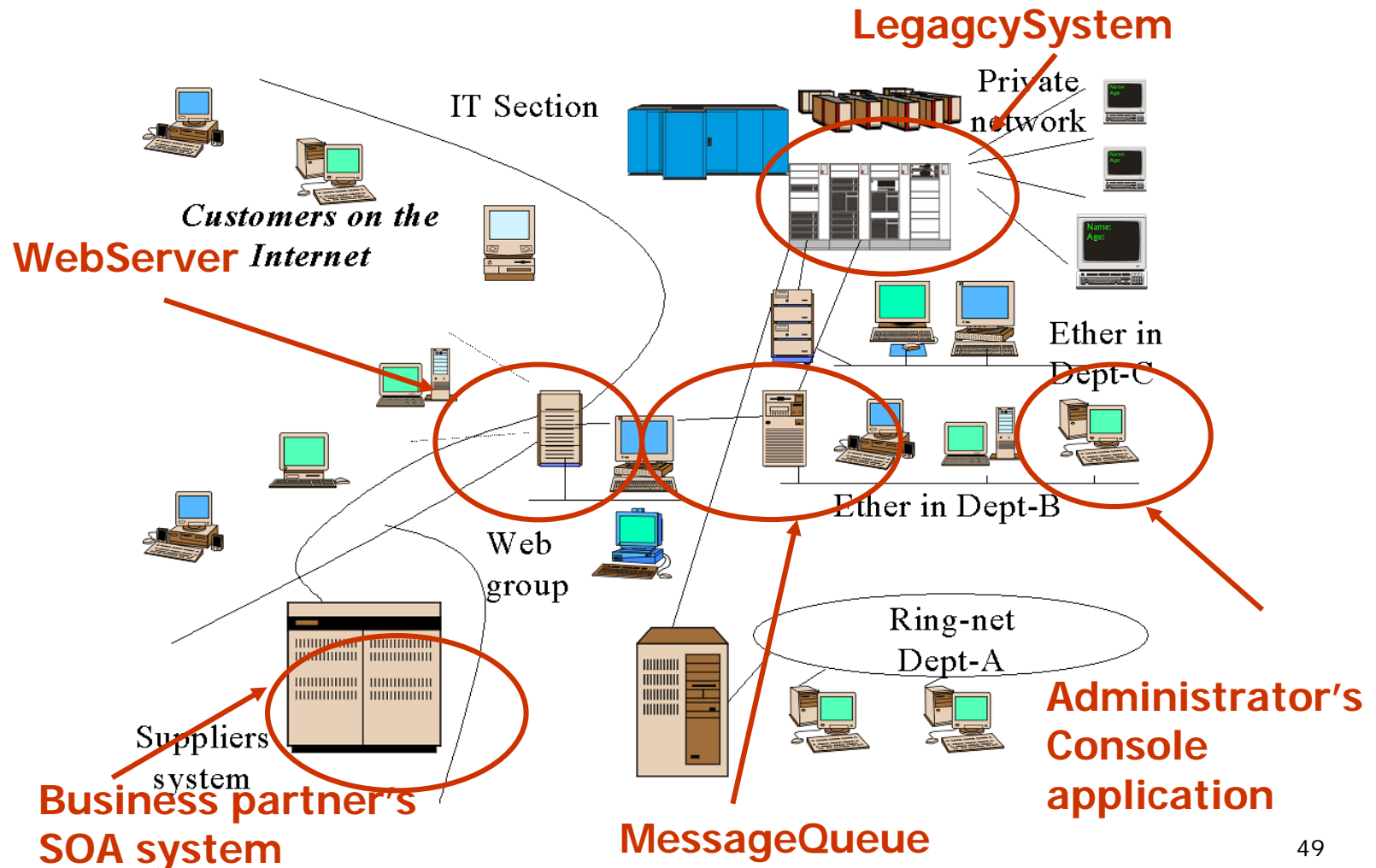
➡ Interaction element

Diagrams and views



Deployment view

Where do things run?



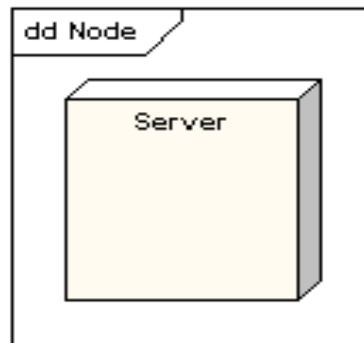
Deployment Diagrams

■ 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

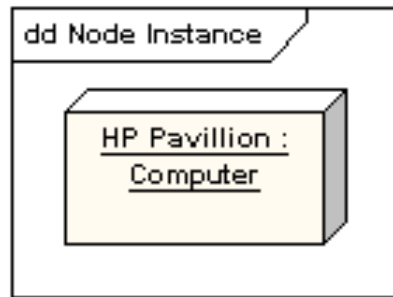
A Node is a hardware (or software) element, shown as a three-dimensional box shape.



Deployment diagram

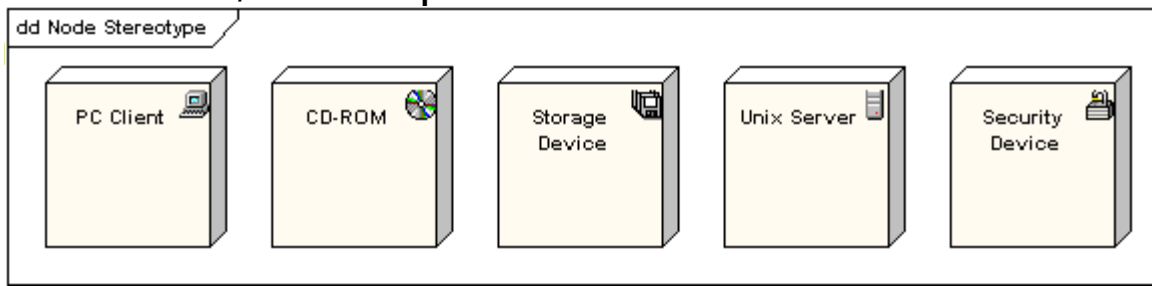
□ Node Instance

- Name and base node type



□ Node Stereotypes

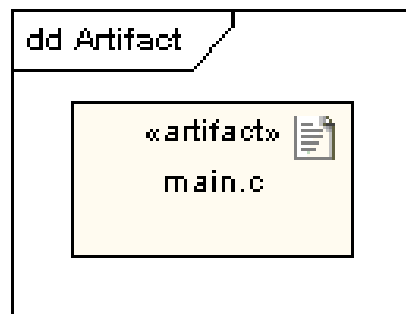
- standard stereotypes are provided for nodes,
 - «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc».



Deployment diagram

□ Artifact

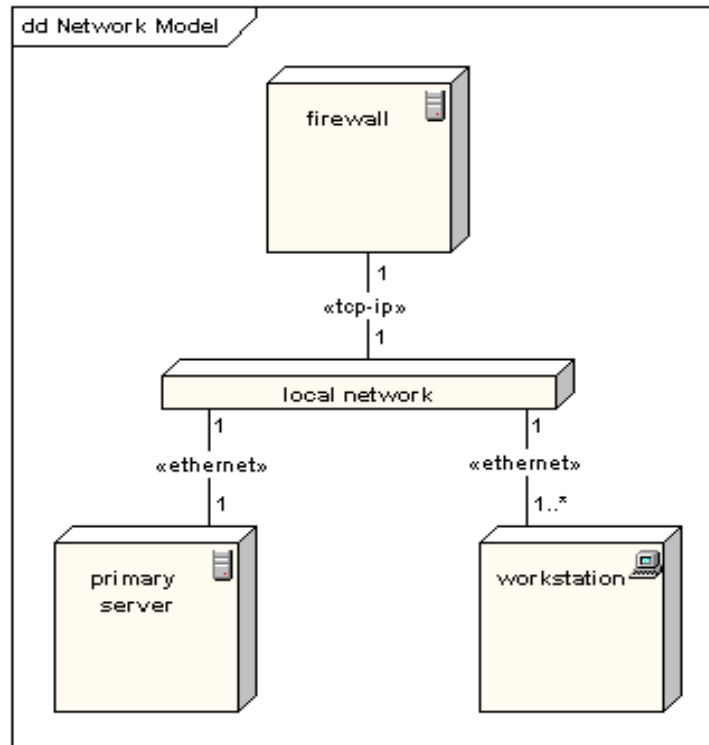
- Any product of the software development process: source files, executables, design documents, test reports, prototypes, user manuals, etc.
- An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon.
- Appear in Deployment diagrams as contents of the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



Deployment diagram

□ Association

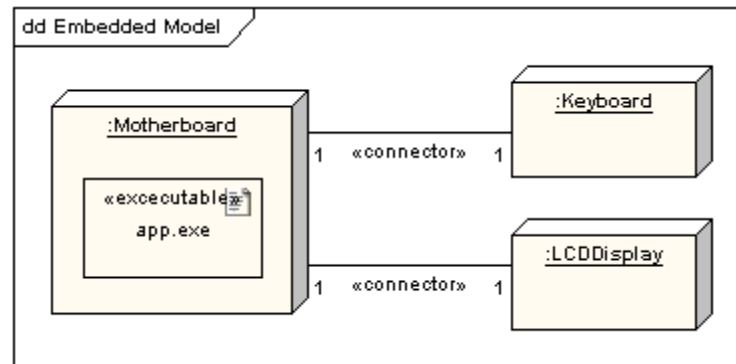
- An association represents a communication path between nodes



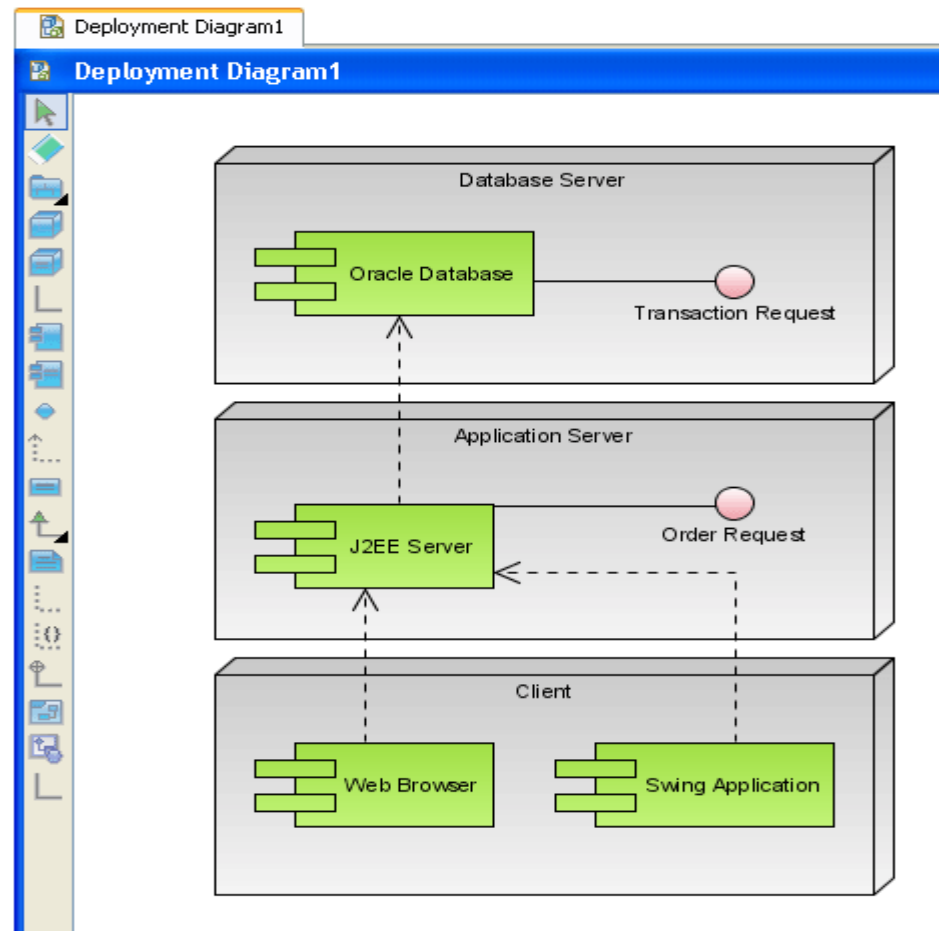
Deployment diagram

□ Node as Container

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



Deployment diagram

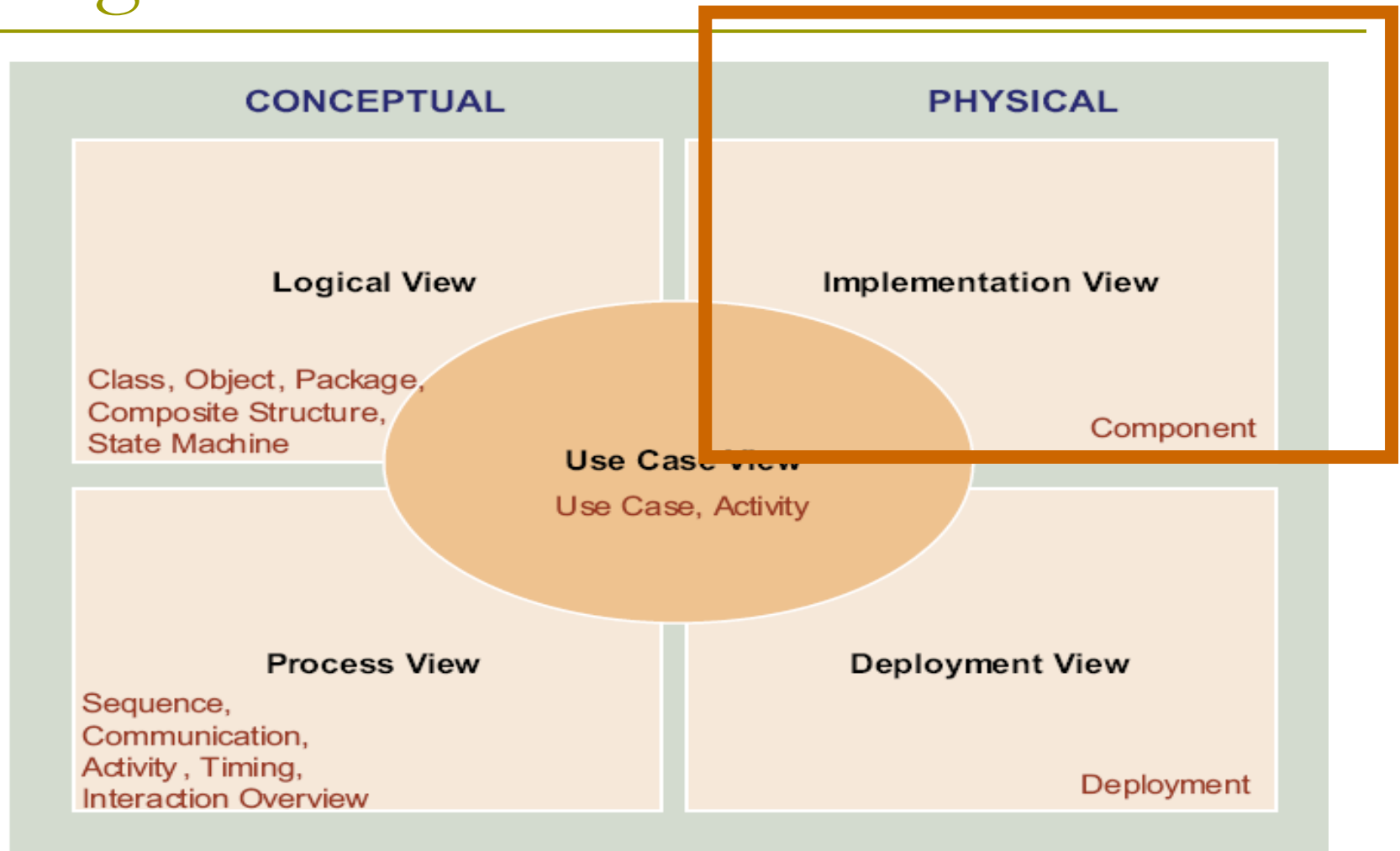


Note: contents of Nodes are shown as components in older UML-1 component notation

Flashback quiz

- ▣ Which of the following is not a RUP discipline/workflow (i.e. group of activities):
 - A. Requirements
 - B. Analysis & Design
 - C. Implementation & Test
 - D. Deployment
 - E. Project Management
 - F. Product Support

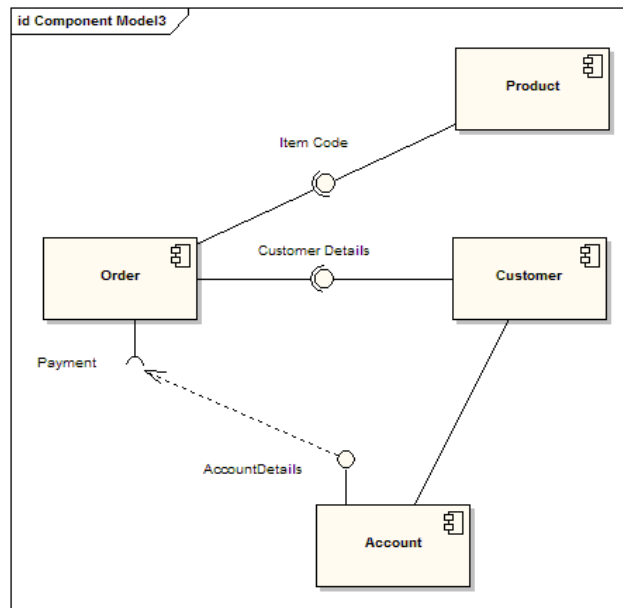
Diagrams and views



Implementation View

Component diagrams

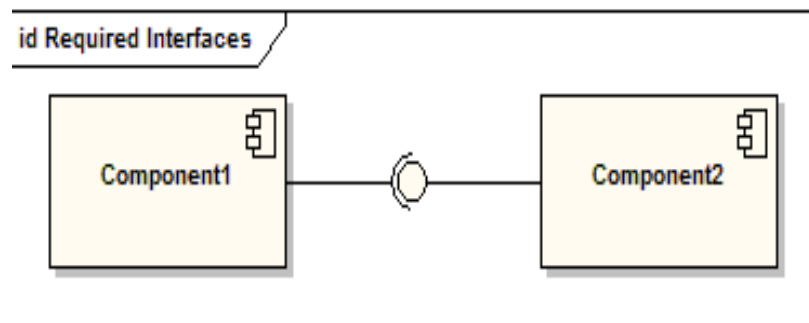
- ❑ Component diagrams illustrate the pieces of **software, embedded controllers**, etc., that will make up a system.
- ❑ A component is usually implemented by one or more classes (or objects) at runtime.



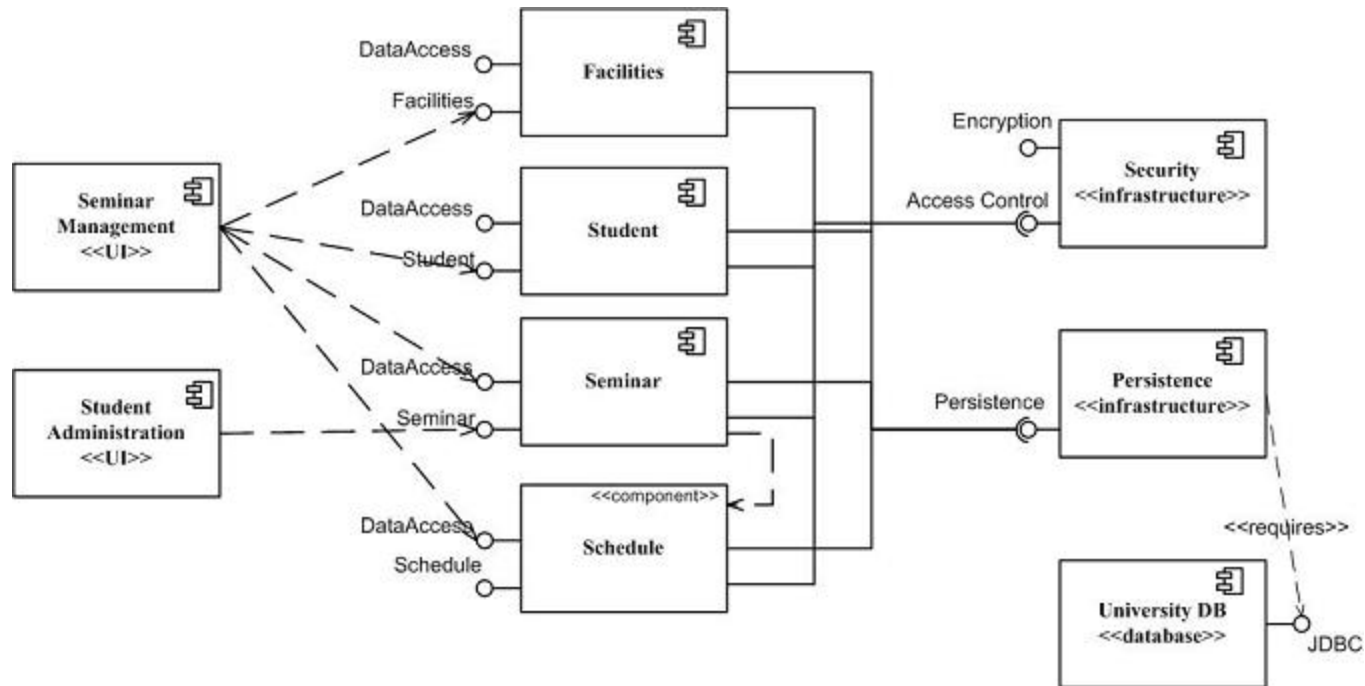
Component diagrams (cont.)

□ 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.)

