

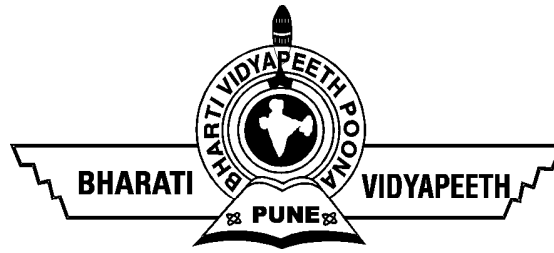
OBJECT ANALYSIS AND DESIGN

UNIT IV



Learning Objectives

- Modeling with UML
- Basic Building Blocks of UML
- A Conceptual Model of UML
- Basic
- Structural Modeling
- UML Diagrams
- Case Studies



OOAD

Overview of the UML



Learning Objectives

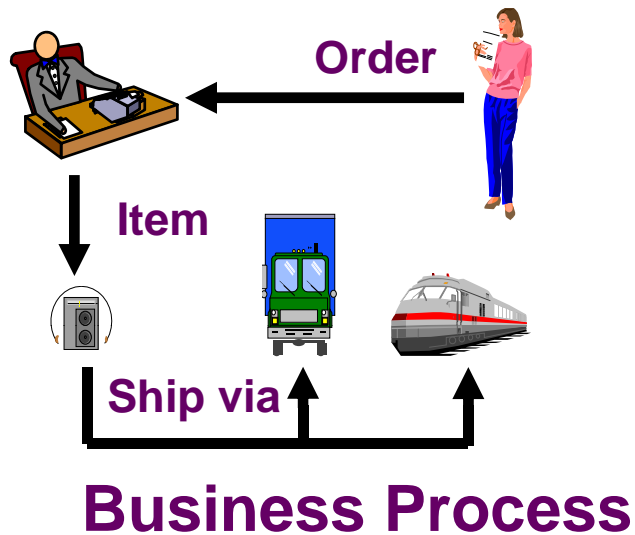
- What is Visual Modeling?
- History of the UML
- Unified Modeling Language (UML) Overview
- Architecture & Views
- Basic Building Blocks of UML
 - Structural Things
 - Behavioral Things
 - Grouping Things
 - Annotational Things
 - Relationship
 - UML Diagrams



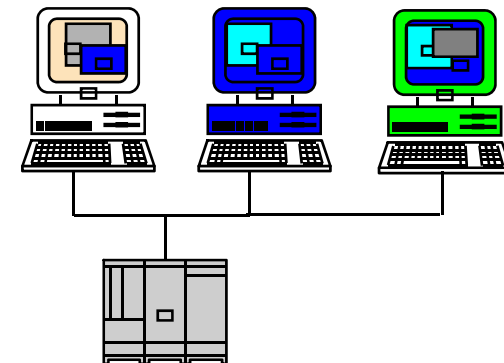
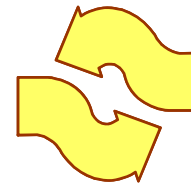
Learning Objectives Cont..

- Extensibility of UML
- Using UML Concepts in a Nutshell
- Rules of UML
- Process for Using UML

What is Visual Modeling?



“Modeling captures essential parts of the system.”
Dr. James Rumbaugh

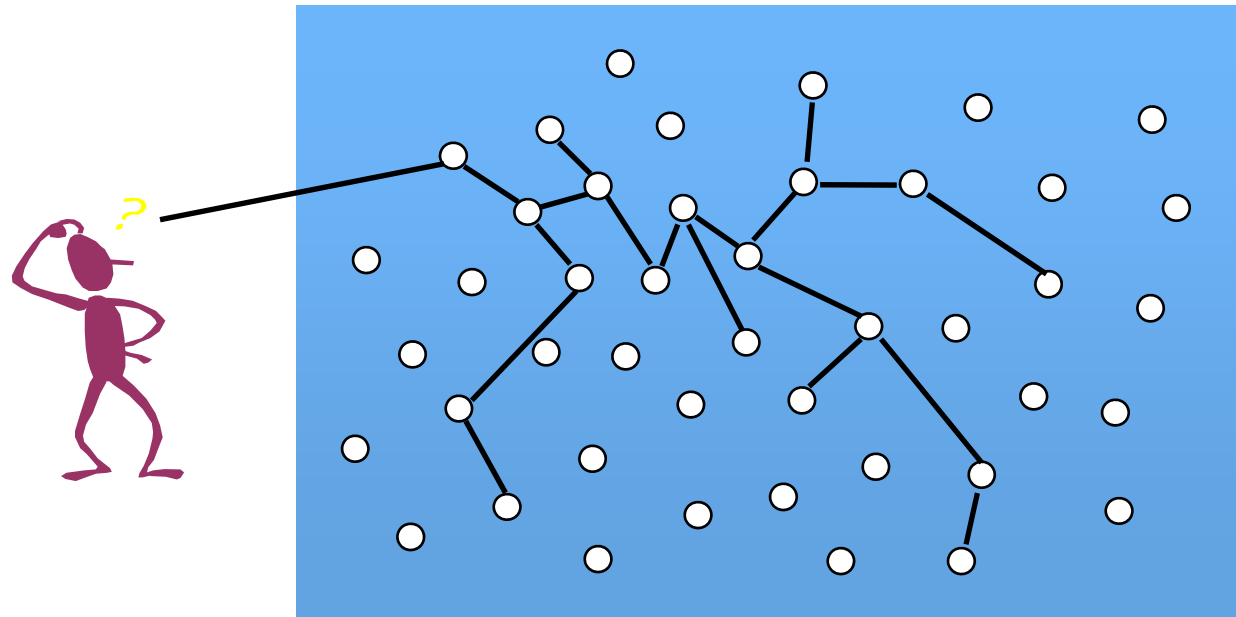


Computer System

Visual Modeling is modeling using standard graphical notations

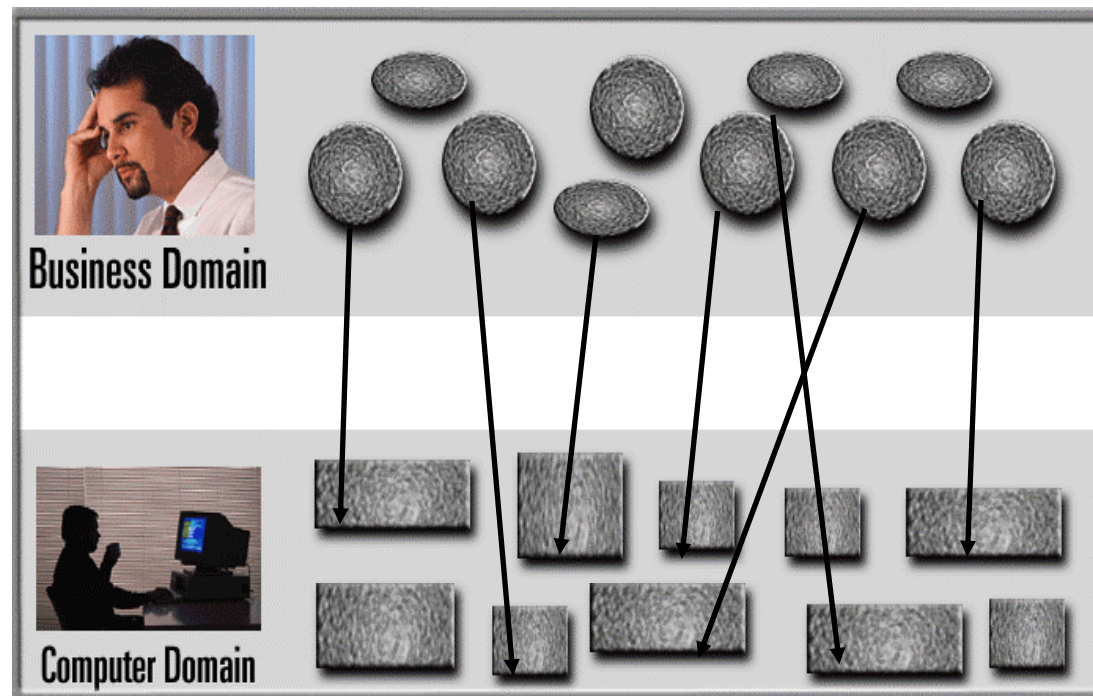
Visual Modeling Captures Business Process

Use Case Analysis is a technique to capture business process from user's perspective



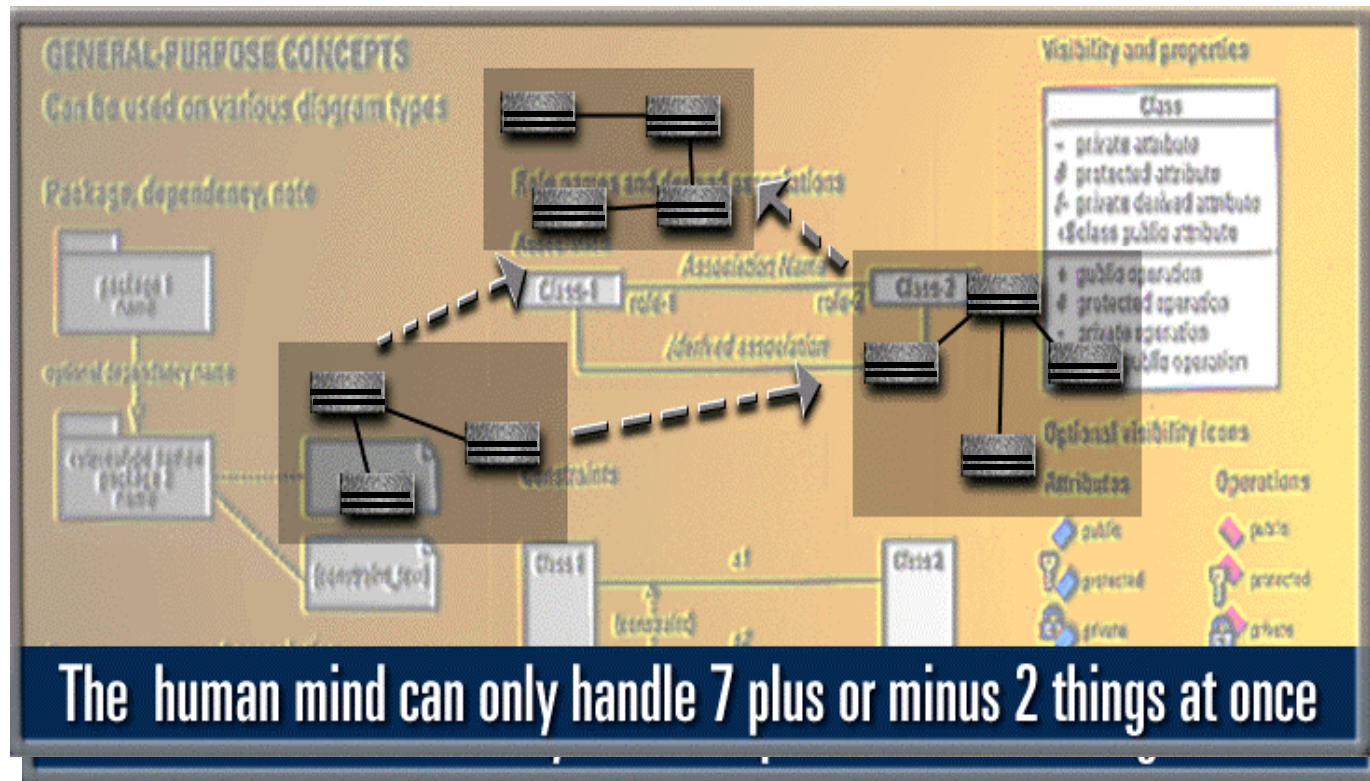
Visual Modeling is a Communication Tool

Use visual modeling to capture business objects and logic

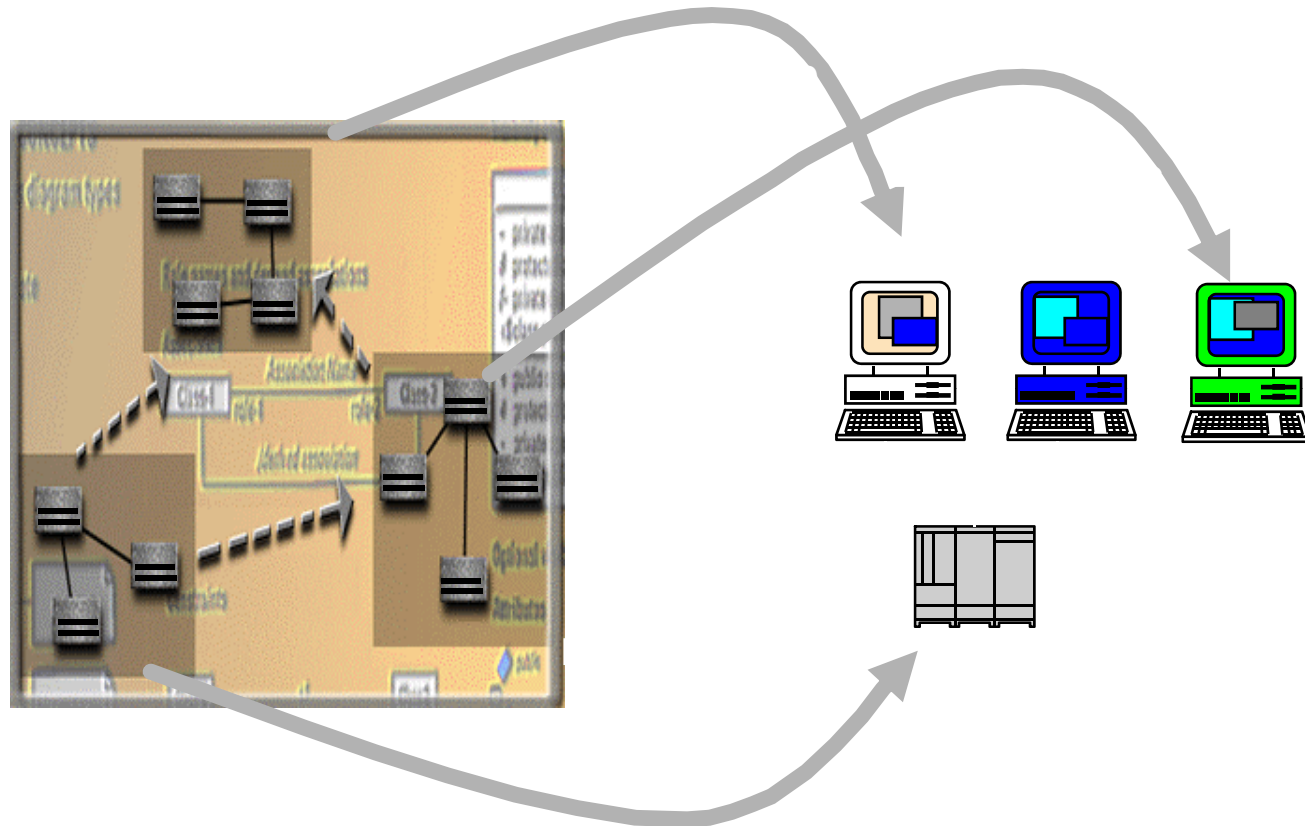


Use visual modeling to analyze and design your application

Visual Modeling : Manages Complexity



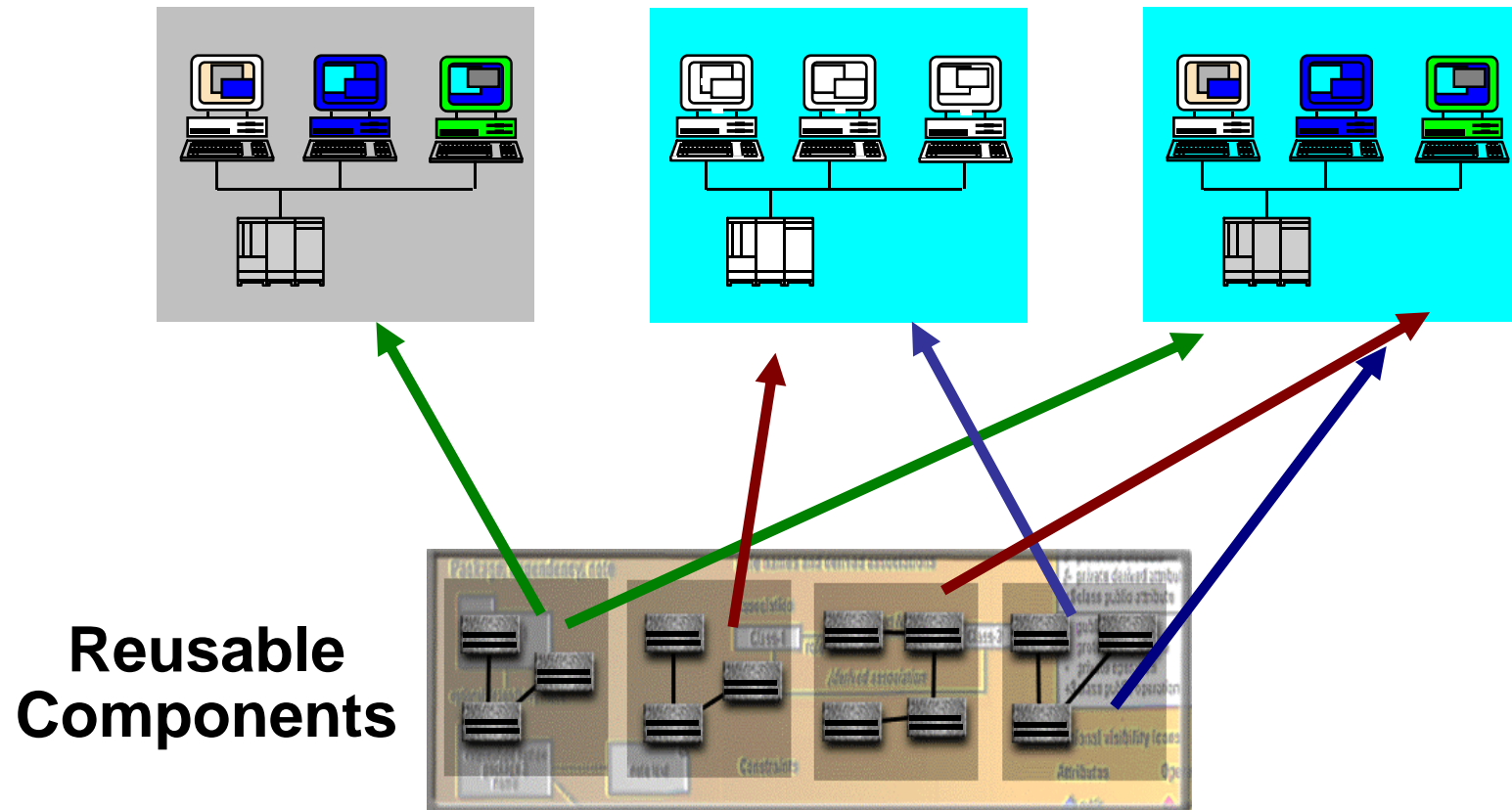
Visual Modeling Defines Software Architecture



**Model your system
independent of
implementation language**

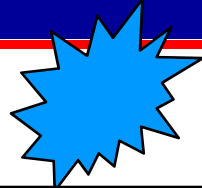
Visual Modeling : Promotes Reuse

Multiple Systems

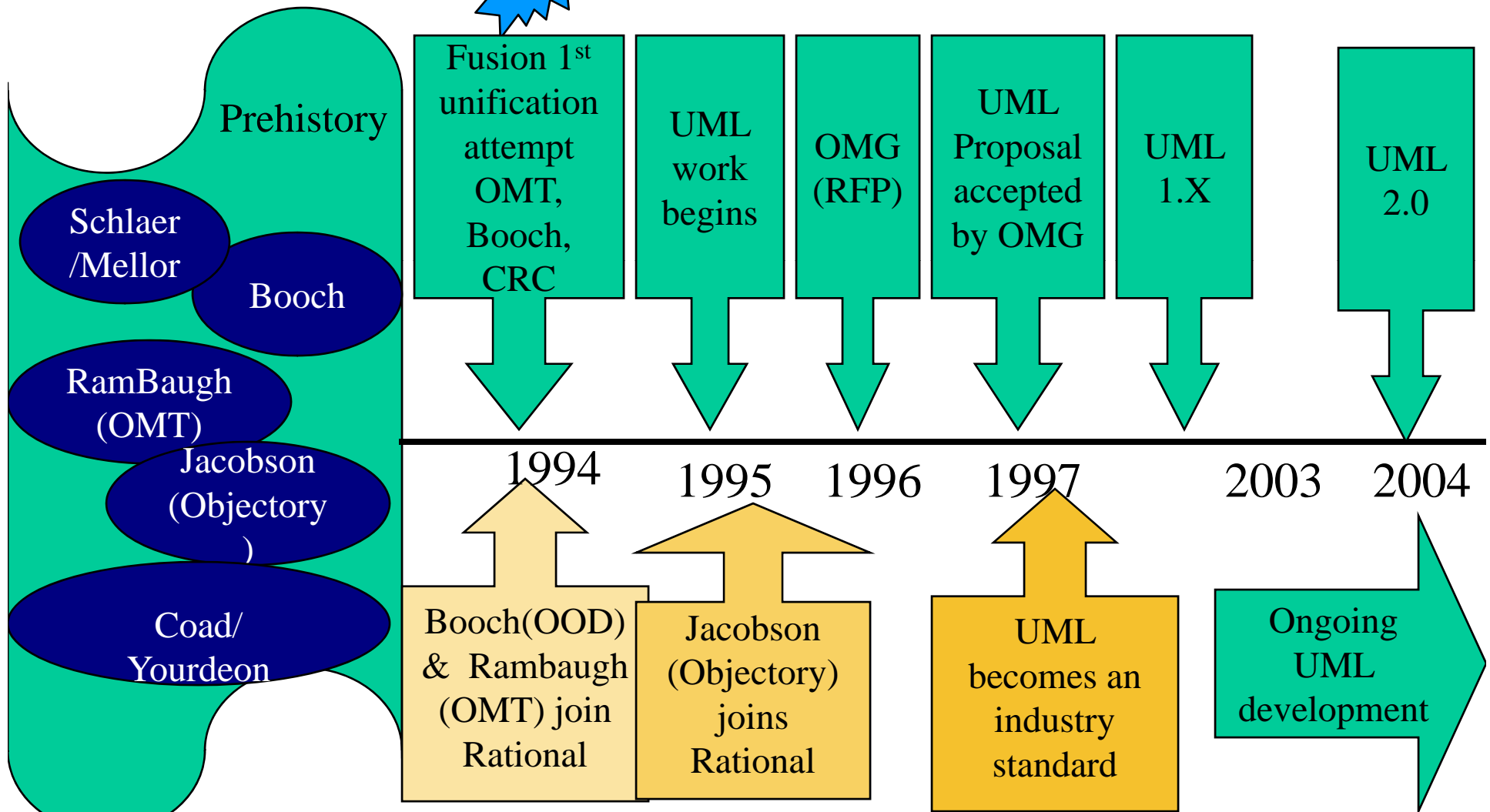


History of the UML

Nov '97



UML approved by the OMG



Unified Modeling Language (UML)

An effort by Rational to standardize OOA&D notation

- “a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system” [Booch]

UML attempts to combine the best of the best from

- Data Modeling concepts (Entity Relationship Diagrams)
- Business Modeling (work flow)
- Object Modeling
- Component Modeling
- **Offers vocabulary and rules for communication**
- **Focus on conceptual and physical representations of a system**
 - **Not a process but a language**



UML is for Visual Modeling

A picture is worth a thousand words!

- Uses standard graphical notations
- Semi-formal
- **Captures Business Process from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems**

UML is also for ...

Specifying

- Building models that are
 - Precise
 - Unambiguous
 - Complete
- Symbols are based on
 - well-defined syntax
 - semantics.
- Addresses the specification of all important analysis, design, and implementation decisions.

UML is also for ...

Constructing

- Models are related to OO programming languages.
- Round-trip engineering requires tool and human intervention to avoid information loss
 - Forward engineering — direct mapping of a UML model into code.
 - Reverse engineering — reconstruction of a UML model from an implementation.

UML is also for ...

Documenting

- Architecture
- Requirements
- Tests
- Activities
 - ✓ **Project planning**
 - ✓ **Release management**



Three (3) Basic Building Blocks of UML

- **Things**

important modeling concepts

- **Relationships**

tying individual things together

- **Diagrams**

grouping interrelated collections of things and relationships



3 Basic Building Blocks of UML - Things

Structural — nouns of UML models.

Behavioral — dynamic (verbal) parts of UML models.

Grouping — organizational parts of UML models.

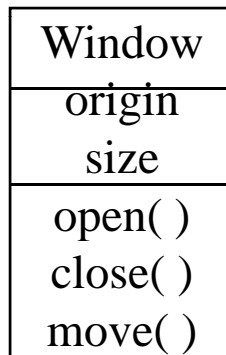
Annotational — explanatory parts of UML models.

Structural Things in UML- 7 Kinds

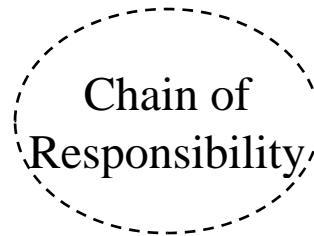
Nouns of UML models

Conceptual or physical elements.

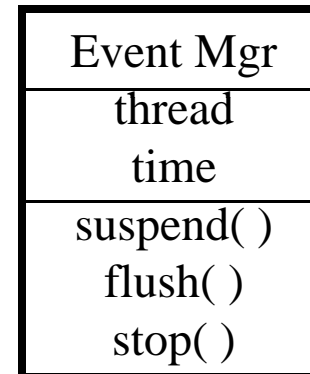
Class



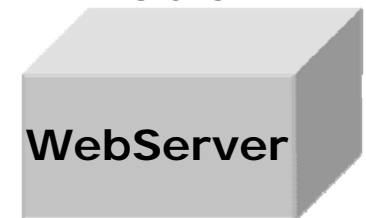
Collaboration



Active Class



Node

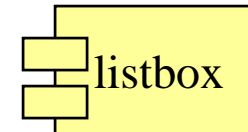


IWindow

Interface



Use Case

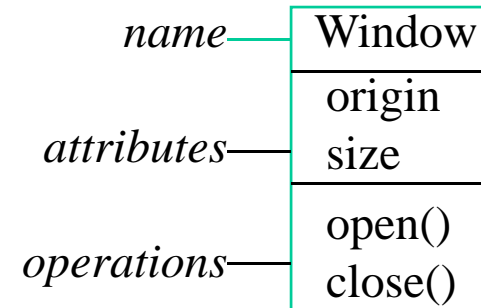


Component

Structural Things in UML

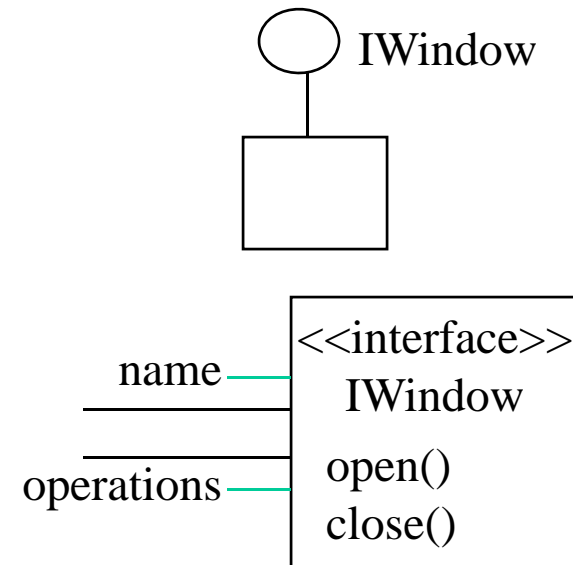
1. Class

A description of a set of objects that share the same attributes, operations, relationships, and semantics.



2. Interface

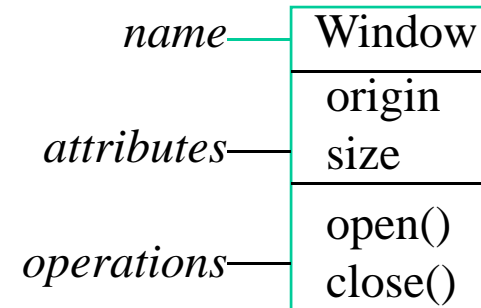
A collection of operations that specify a service (for a resource or an action) of a class or component. It describes the externally visible behavior of that element.



Structural Things in UML

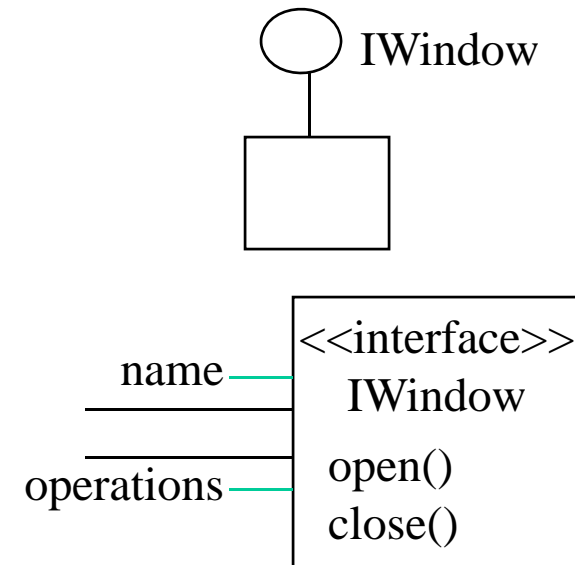
1. Class

A description of a set of objects that share the same attributes, operations, relationships, and semantics.



2. Interface

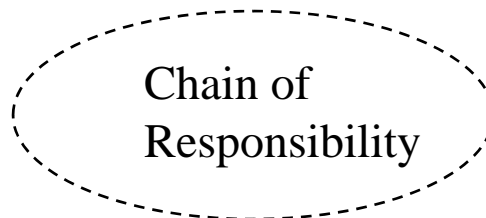
A collection of operations that specify a service (for a resource or an action) of a class or component. It describes the externally visible behavior of that element.



Structural Things in UML..

3. Collaboration

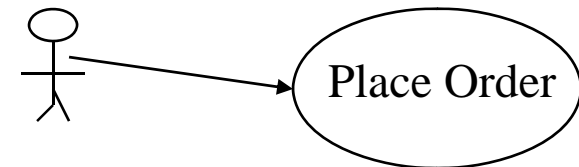
- Define an interaction among two or more classes.
- Define a society of roles and other elements.
- Provide cooperative behavior.
- Capture structural and behavioral dimensions.
- UML uses ‘pattern’ as a synonym



Structural Things in UML..

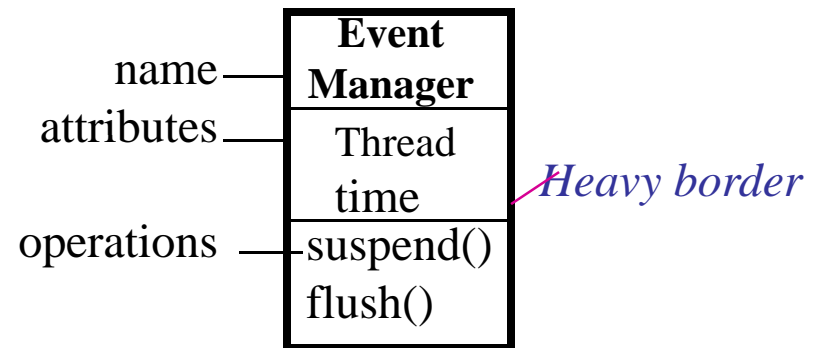
4. Use Case

- A sequence of actions that produce an observable result for a specific actor.
- A set of scenarios tied together by a common user goal.
- Provides a structure for behavioral things.
- Realized through a collaboration



Structural Things in UML..

5. Active Class

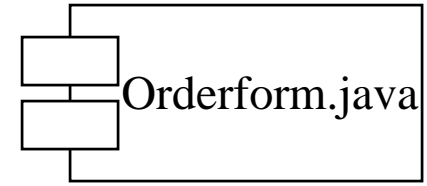


- Special class whose objects own one or more *processes* or *threads*.
- Can initiate control activity.

Structural Things in UML..

6. Component

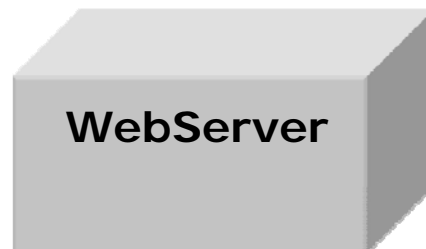
- Replaceable part of a system.
- Components can be packaged logically.
- Conforms to a set of interfaces.
- Provides the realization of an interface.
- Represents a physical module of code



Structural Things in UML..

7. Node

- Element that exists at *run time*.
- Represents a *computational resource*.
- Generally has memory and processing power.





Behavioral Things in UML

- Verbs of UML models.
- Dynamic parts of UML models: “behavior over time and space”
- Usually connected to structural things in UML.

Behavioral Things in UML..

Two primary kinds of behavioral things:

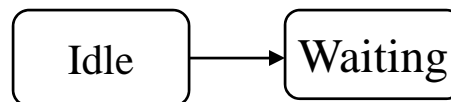
Interaction

behavior of a set of objects comprising of a set of message exchanges within a particular context to accomplish a specific purpose.



State Machine

behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.



Grouping Things in UML

Packages –

- one primary kind of grouping.



Meeting Scheduler

A UML Package Diagram showing a package named 'Meeting Scheduler'. The package is represented by a rectangle with a small tab on the top-left corner.

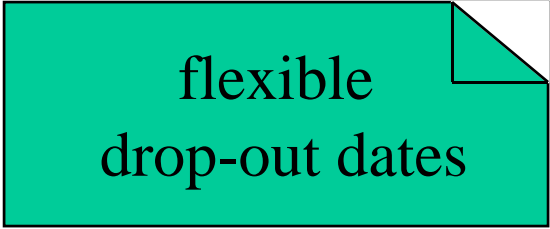
- General purpose mechanism for organizing elements into groups.
- Purely conceptual; only exists at development time.
- Contains behavioral and structural things.
- Can be nested.
- Variations of packages are: Frameworks, models, & subsystems.

Annotational Things in UML

- Explanatory parts of UML models
- Comments regarding other UML elements (usually called adornments in UML)

Note is the one primary annotational thing in UML

- best expressed in informal or formal text.



flexible
drop-out dates



3 basic building blocks of UML - Relationships

- Dependency
- Association
- Generalization

Generalization

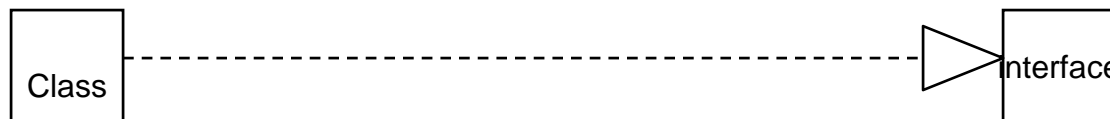
Inheritance:

- a solid line with a solid arrowhead that points from a sub-class to a superclass or from a sub-interface to its super-interface.



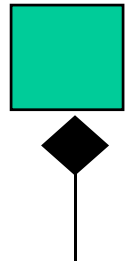
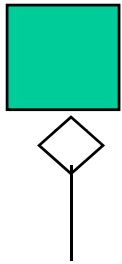
Implementation:

- a dotted line with a solid arrowhead that points from a class to the interface that it implement



Association

- a solid line with an open arrowhead that represents a "has a" relationship.
- The arrow points from the containing to the contained class.
- Associations can be one of the following two types or not specified.



1.

Composition

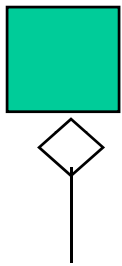
2.

Aggregation

Association

Composition:

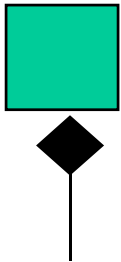
- Represented by an association line with a solid diamond at the tail end.



- A composition models the notion of one object "owning" another and thus being responsible for the creation and destruction of another object.

Aggregation:

- Represented by an association line with a hollow diamond at the tail end.



- An aggregation models the notion that one object uses another object without "owning" it and thus is not responsible for its creation or destruction.

Composition

// WebServer is composed of a HttpListener and a RequestProcessor

```
public class WebServer {  
    private HttpListener listener;  
    private RequestProcessor processor;  
    public WebServer(HttpListener listener, RequestProcessor  
        processor) {  
        this.listener = listener;  
        this.processor = processor;  
    }  
}
```

Aggregation

// WebServer is an aggregate of HttpListener and RequestProcessor

// and controls their lifecycle

```
public class WebServer {  
    private HttpListener listener;  
    private RequestProcessor processor;  
    public WebServer() {  
        this.listener = new HttpListener(80);  
        this.processor = new RequestProcessor("/www/root");  
    }  
}
```

Dependency

- a dotted line with an open arrowhead that shows one entity depends on the behavior of another entity.
- Typical usages are to represent that one class instantiates another or that it uses the other as an input parameter

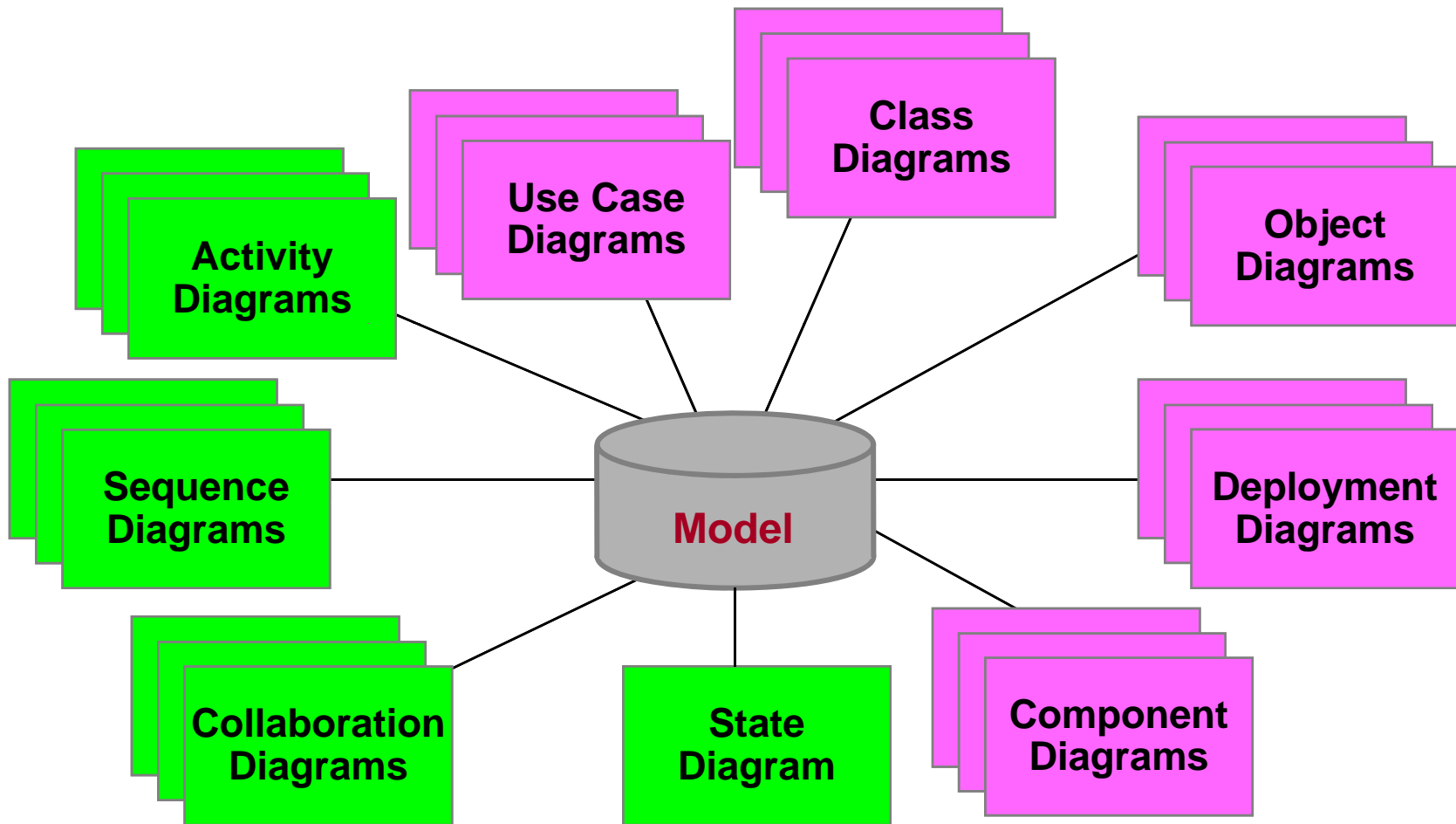


3 basic building blocks of UML - Diagrams

- Graphical representation of a set of elements.
- Represented by a connected graph:
 - Vertices are things;
 - Arcs are behaviors.
- 5 most common views built from 9 diagram types.

- 1. Class Diagram; Object Diagram**
- 2. Use case Diagram**
- 3. Sequence Diagram; Collaboration Diagram**
- 4. Statechart Diagram**
- 5. Activity Diagram**
- 6. Component Diagram**
- 7. Deployment Diagram**

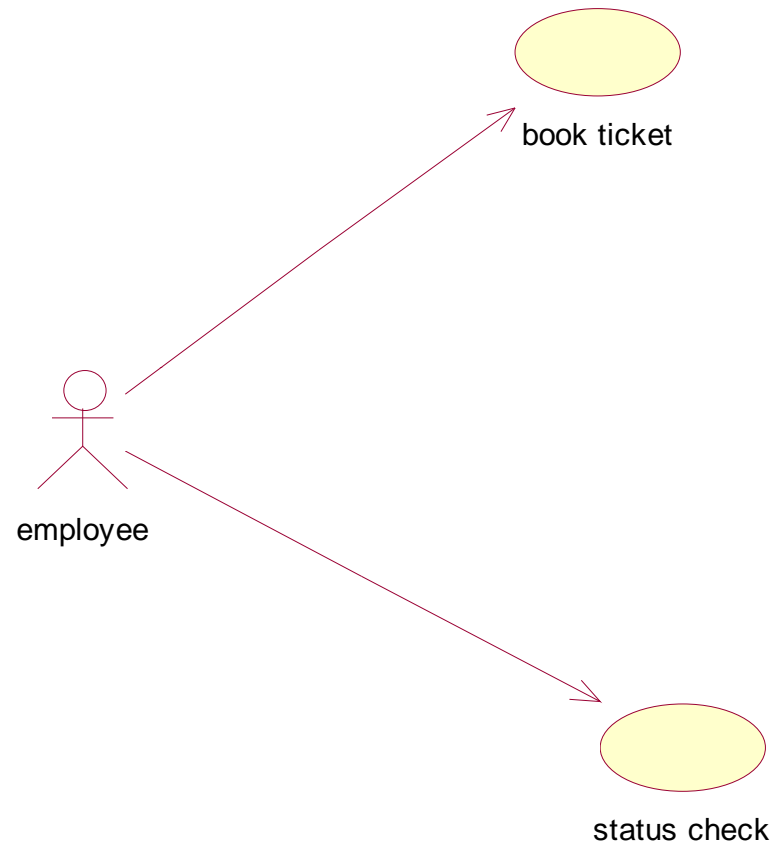
The UML Provides Standardized Diagrams



Use-case Diagram

- It shows a set of use cases and actors and their relationships. It
- helps the analyst to discover the requirements of the target
- system from the user's perspective.
- It is depicted as follows in Rational Rose :

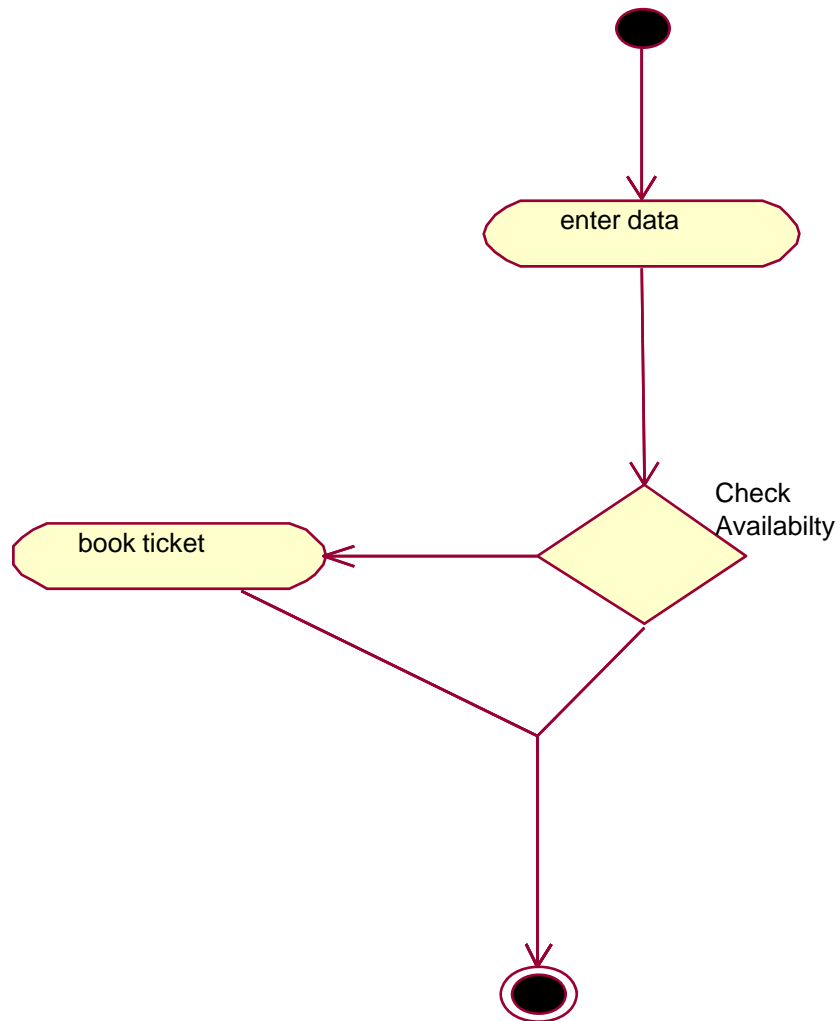
Use-case Diagram



Activity Diagram

- It's a special kind of a state chart diagram that shows the flow
- from activity to activity within the system.
- Its depicted as follows in Rational Rose :

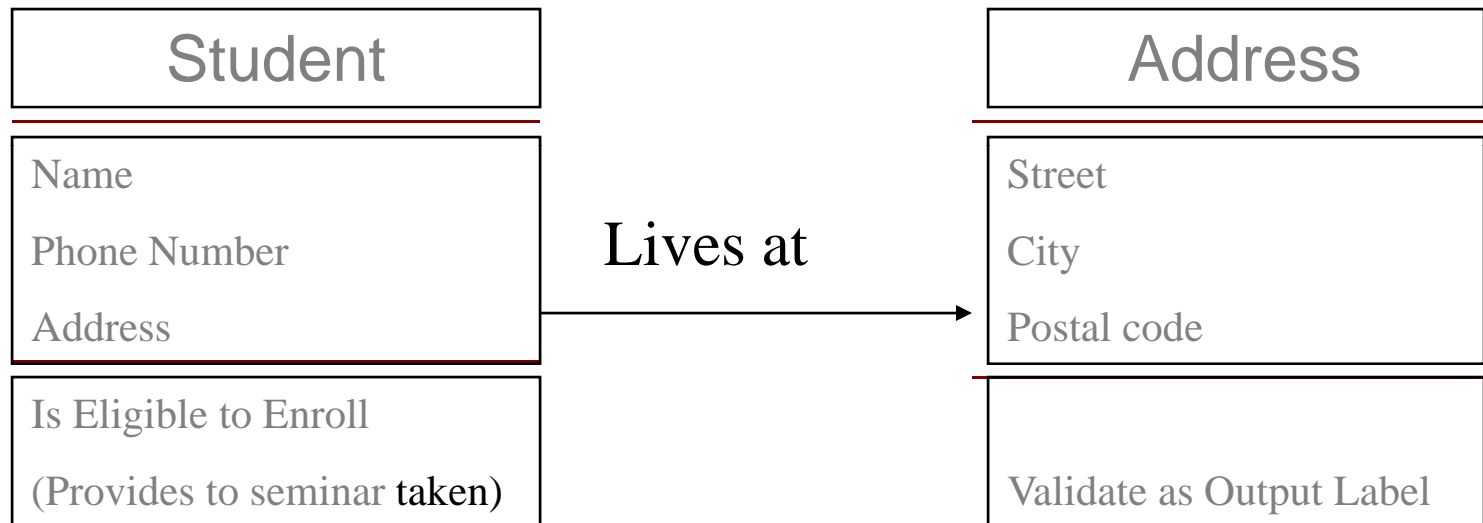
Activity Diagram



Class Diagram

- It shows a set of classes and their relationships. It addresses the
- static design view of a system.
- Its depicted as follows in Rational Rose :

Class Diagram

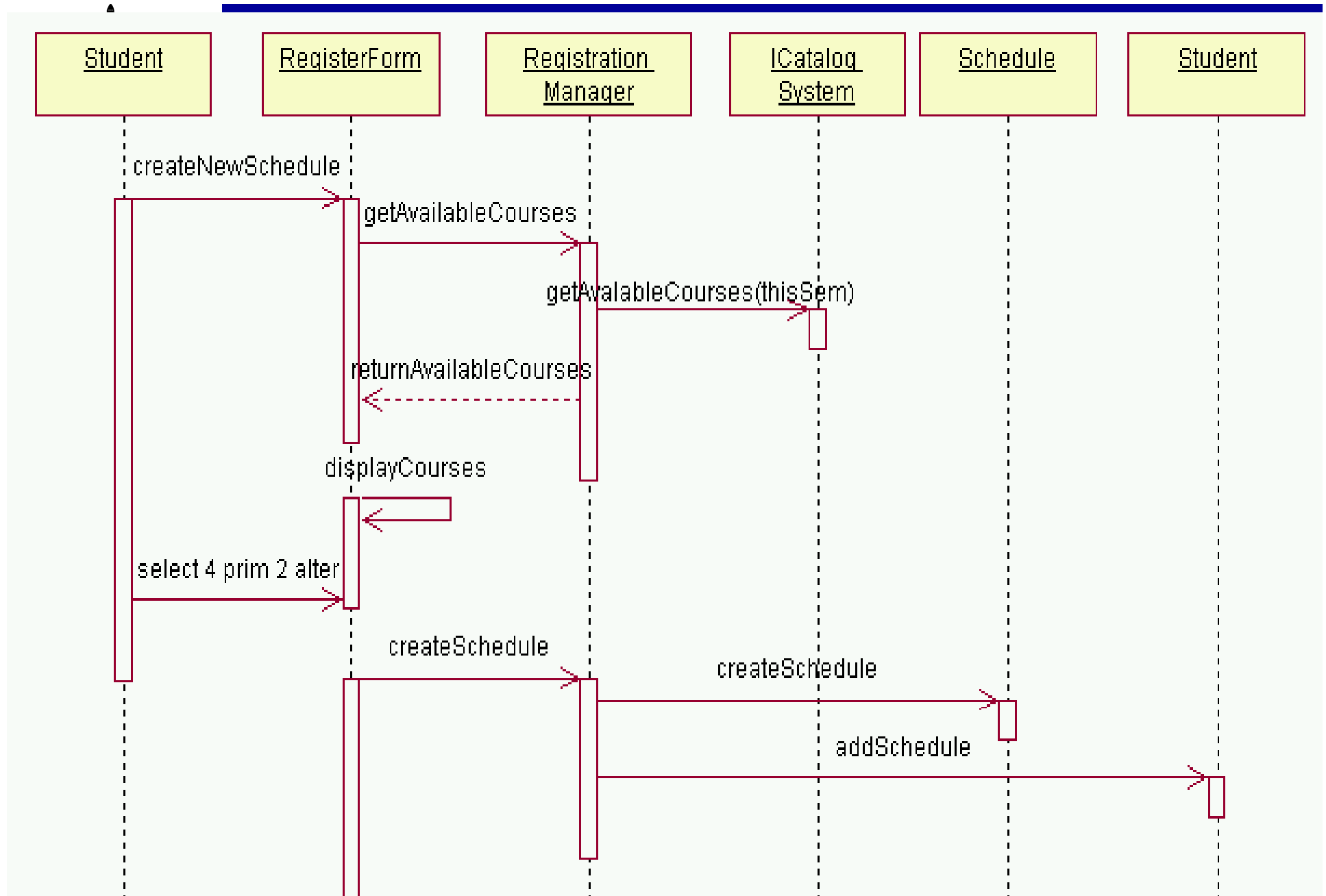


Interaction Diagrams

- An interaction diagram shows an interaction,
- consisting of a set of objects and their relationships,
- include the messages that may be exchanged between them
- Model the dynamic aspect of the system
- Contain two sort of diagrams:
 - **Sequence diagrams,**
 - ✓ **show the messages objects send to each other in a timely manner**
 - **Collaboration diagrams,**
 - ✓ **show the organization of the objects participating in an interaction**

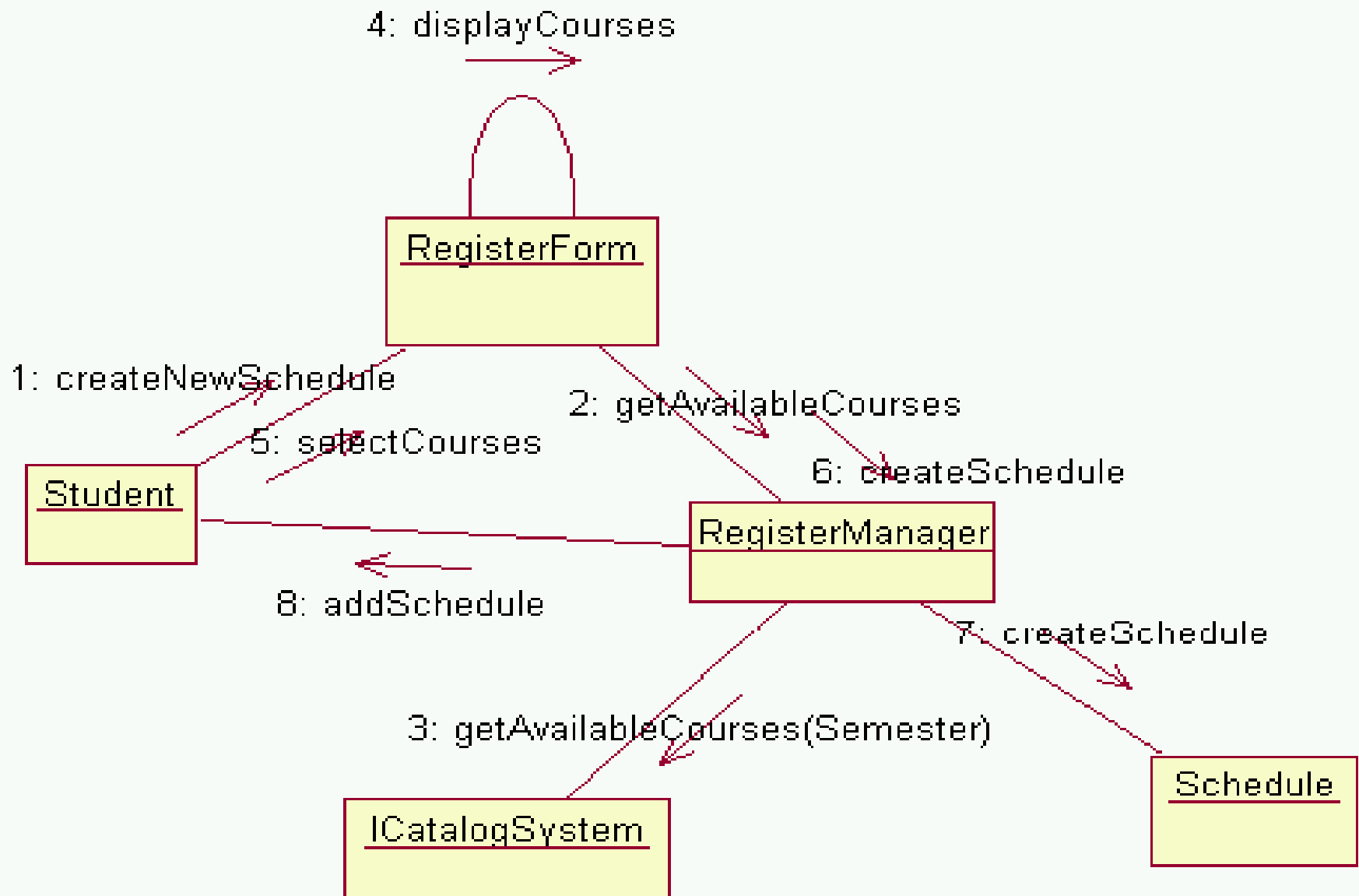
Sequence Diagram

- It is an interaction diagram that emphasizes the time ordering of messages.
- It is drawn in Rational Rose as in next slide



Collaboration Diagram

- Its an interaction diagram that emphasizes the structural
- organization of the objects that send and receive messages.
- It is drawn in Rational Rose as in next slide



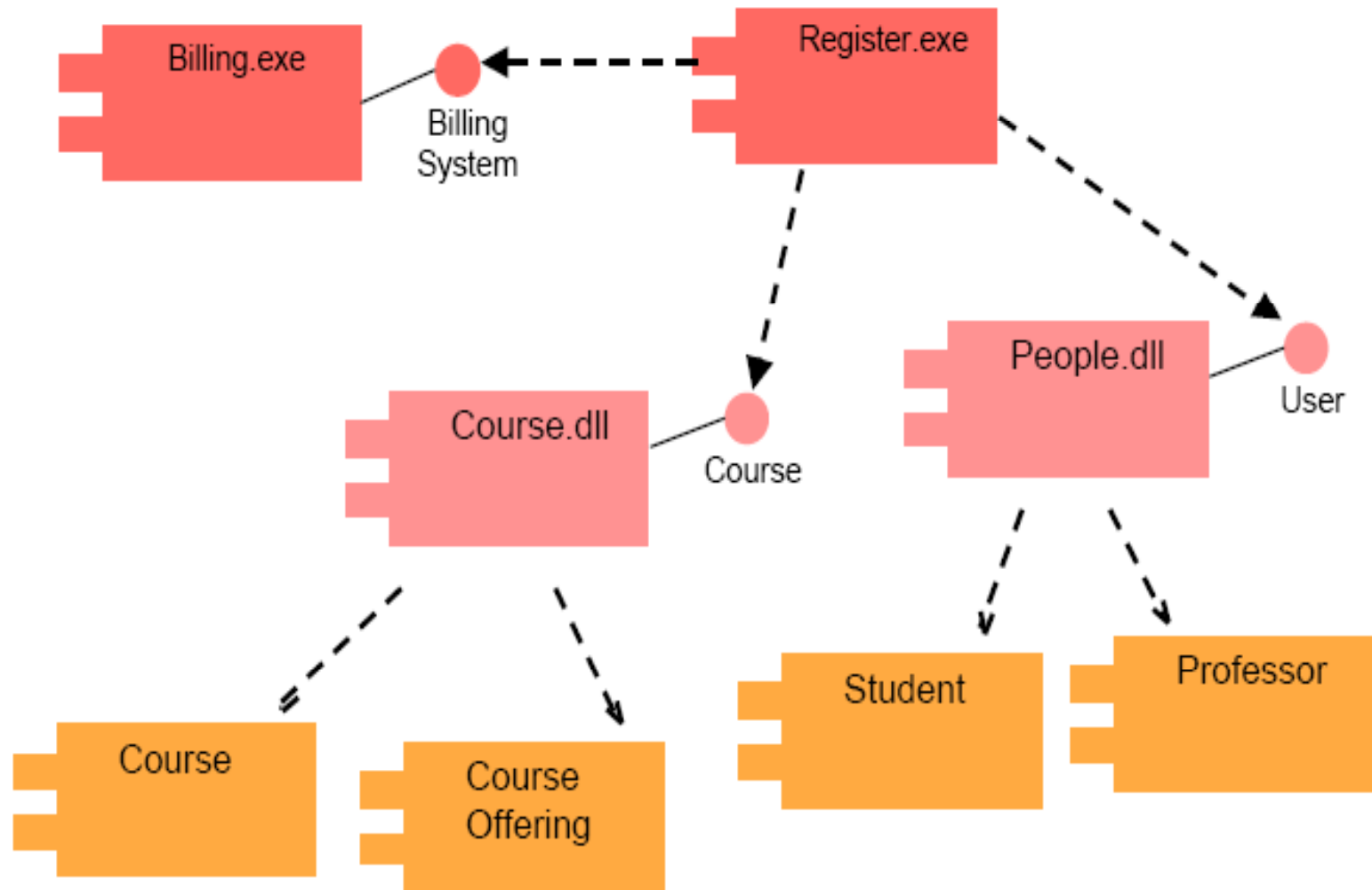
Sequence v/s Collaboration

- Semantically both are the same
- Express different sides of the model
- Sequence diagram expresses time ordering
- Collaboration diagram is used to define class behavior

Component Diagram

- The main purpose of component diagram is to show the structural relationship between components of the system.
- Its depicted as follows in Rational Rose :

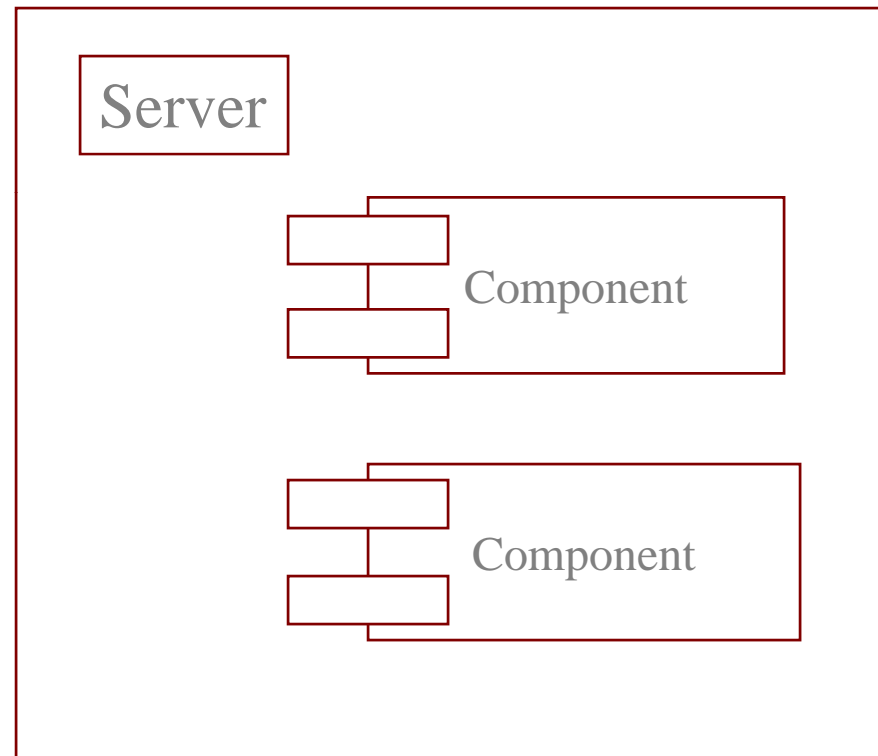
Component Diagram



Deployment Diagram

- The deployment diagram shows how a system will be physically deployed in the hardware environment .
- Its depicted as follows in Rational Rose :

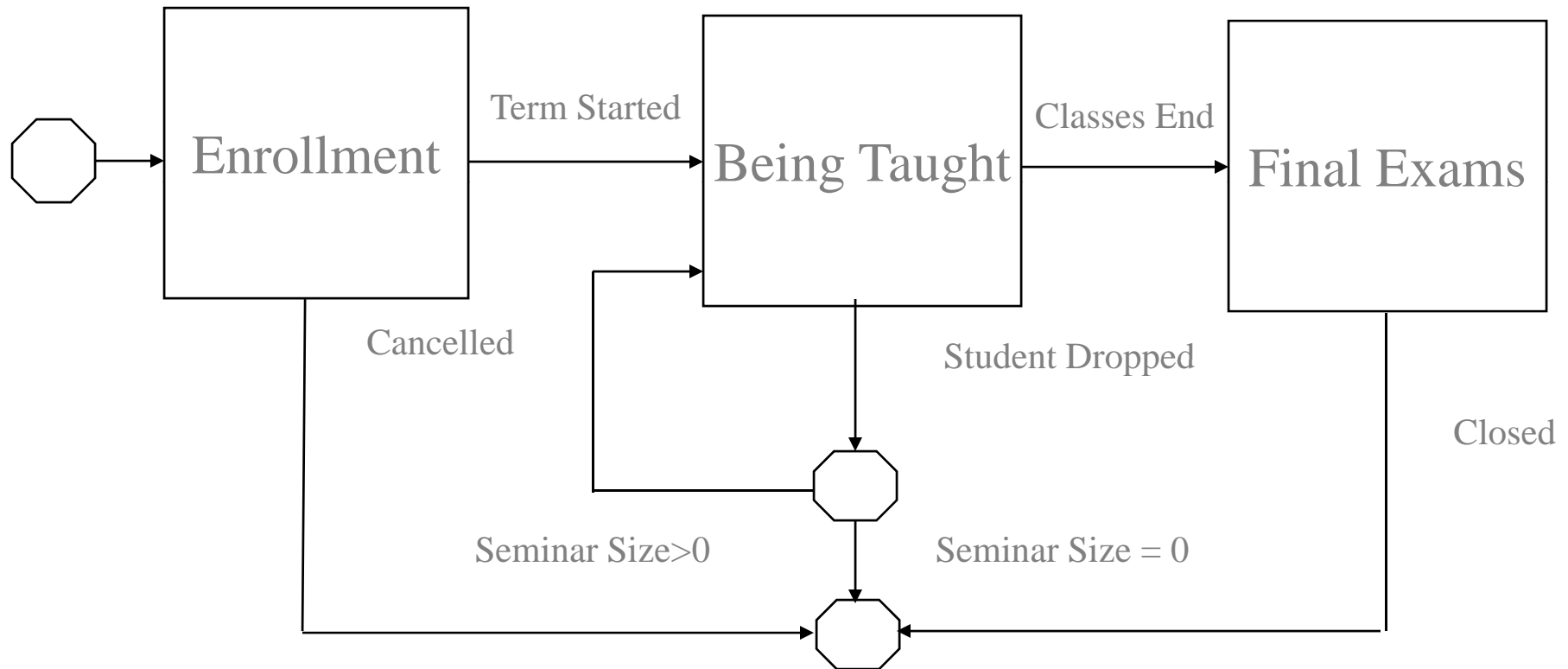
Deployment Diagram



State Chart Diagram

- It shows a state machine, consisting of states , transitions , events and activities.
- It addresses the dynamic view of a system.
- Its depicted as follows in Rational Rose :

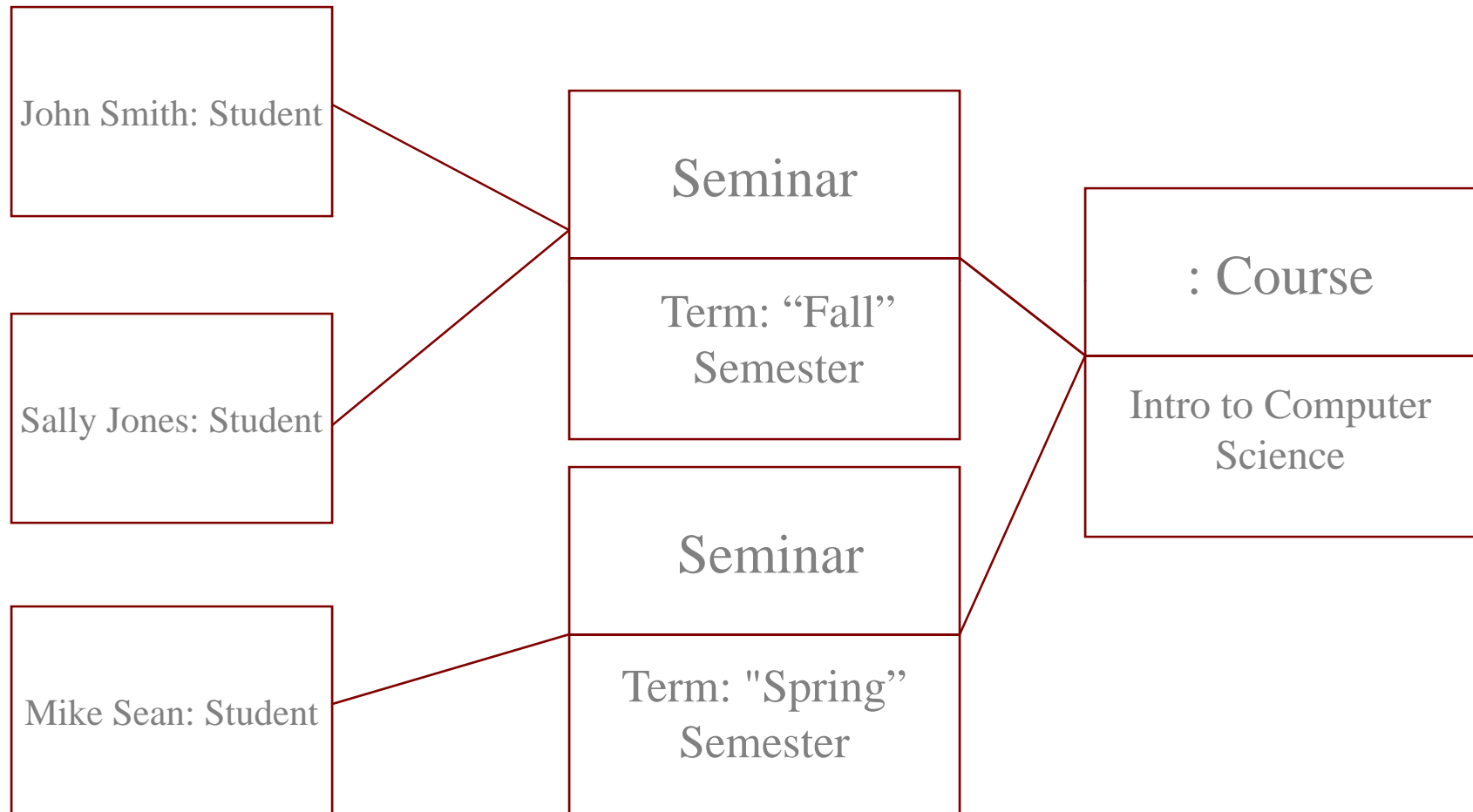
State Chart Diagram



Object Diagram

- A pictorial representation of the relationships between the
- instantiated classes at any point of time.
- Its depicted as follows in Rational Rose:

Object Diagram



Extensibility of UML

- Stereotypes can be used to extend the UML notational elements (*extends vocabulary.*)
 - Name shown in guillemets <<*stereotype*>>
 - Stereotypes may be used to classify and extend associations, inheritance relationships, classes, and components
 - Examples:
 - **Class stereotypes:** boundary, control, entity, utility, exception
 - **Inheritance stereotypes:** includes and extends
 - **Component stereotypes:** subsystem

Extensibility of UML...

- **Tagged values**

- extends properties of UML building blocks.
- Placed in braces beneath the name of the element to which the property applies

- **Constraints**

- extend the semantics of UML building blocks.
- A string placed in braces
{if Order.customer.credit.Rating is “poor” then Order.isPrepaid must be true}

Architecture & Views

- Architecture - set of significant decisions regarding:
 - Organization of a software system.
 - Selection of structural elements & interfaces from which a system is composed.
 - Behavior or collaboration of elements.
 - Composition of structural and behavioral elements.
 - Architectural style guiding the system.

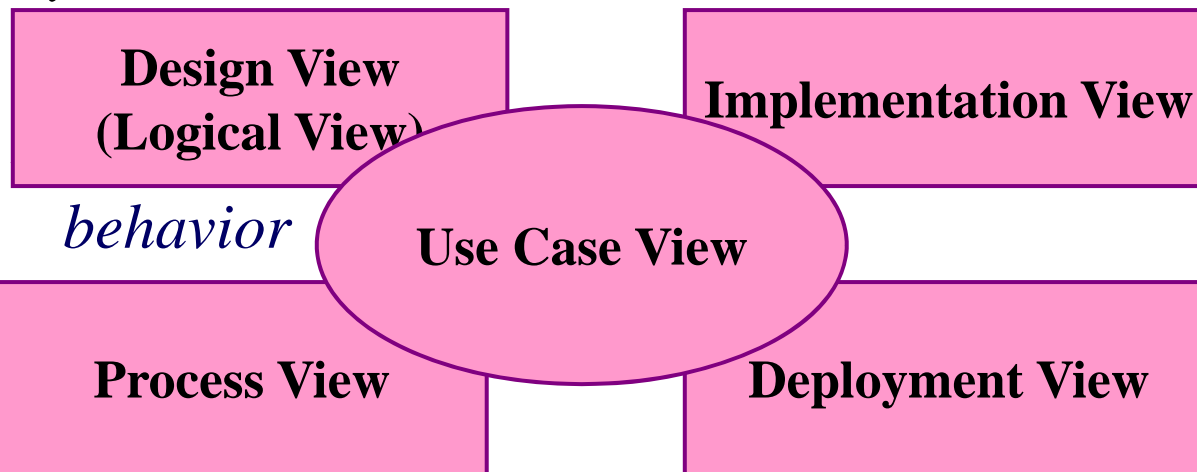
UML have five different views

- Usecase View
- Design View
- Implementation View
- Process View
- Deployment View

Architecture & Views..

*vocabulary
functionality*

*system assembly
configuration mgmt.*



behavior

*performance
scalability
throughput*

*system topology
distribution
delivery
installation*

Use Case View

- Technique to capture business process from user's perspective.
- Encompasses the behavior as seen by users, analysts and testers.
- Specifies forces that shape the architecture.
- Static aspects captured in use case diagrams.
- Dynamic aspects captured in interaction diagrams, statechart diagrams, and activity diagrams.



Design View (Logical View)

- Encompasses classes, interfaces, and collaborations that define the vocabulary of a system.
- Supports functional requirements of the system.
- Static aspects captured in class diagrams and object diagrams.
- Dynamic aspects captured in interaction, statechart, and activity diagrams.

Process View

- Encompasses the threads and processes defining concurrency and synchronization.
- Addresses performance, scalability, and throughput.
- Static and dynamic aspects captured as in design view; emphasis on active classes.

Implementation View

- Encompasses components and files used to assemble and release a physical system.
- Addresses configuration management.
- Static aspects captured in component diagrams.
- Dynamic aspects captured in interaction, statechart, & activity diagrams.



Deployment View

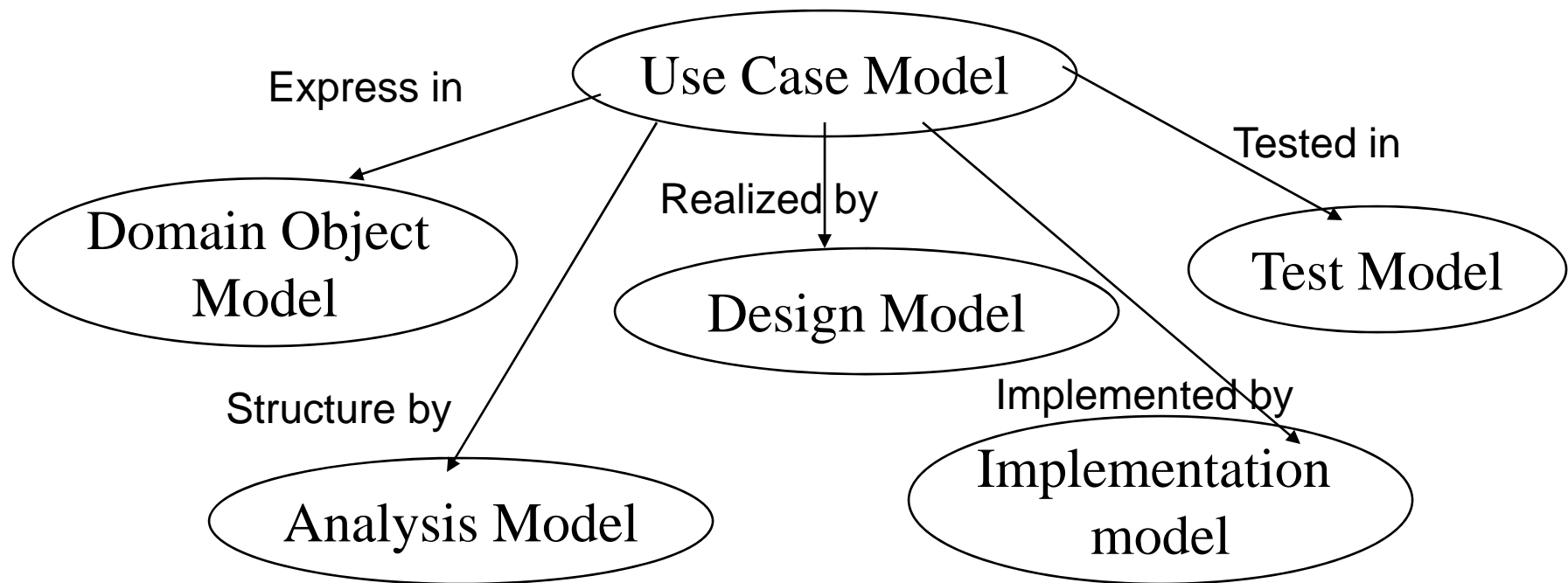
- Encompasses the nodes that form the system hardware topology.
- Addresses distribution, delivery, and installation.
- Static aspects captured in deployment diagrams.
- Dynamic aspects captured in interaction, statechart, & activity diagrams.



Using UML Concepts in a Nutshell

- Display the boundary of a system & its major functions using use cases and actors
- Illustrate use case realizations with interaction diagrams
- Represent a static structure of a system using class diagrams
- Model the behavior of objects with state transition diagrams
- Reveal the physical implementation architecture with component & deployment diagrams
- Extend your functionality with stereotypes

Using UML Concepts in a Nutshell..



Rules of UML

- Well formed models — *semantically self-consistent and in harmony with all its related models.*
- Semantic rules for:
 - **Names** — what you can call things.
 - **Scope** — context that gives meaning to a name.
 - **Visibility** — how names can be seen and used.
 - **Integrity** — how things properly and consistently relate to one another.
 - **Execution** — what it means to run or simulate a dynamic model.

Rules of UML

Avoid models that are

- Elided
 - certain elements are hidden for simplicity.
- Incomplete
 - certain elements may be missing.
- Inconsistent
 - no guarantee of integrity.

Process for Using UML

How do we use UML as a notation to construct a good model?

- Use case driven
 - use cases are primary artifact for defining behavior of the system.
- Architecture-centric
 - the system's architecture is primary artifact for conceptualizing, constructing, managing, and evolving the system.
- Iterative and incremental
 - managing streams of executable releases with increasing parts of the architecture included.

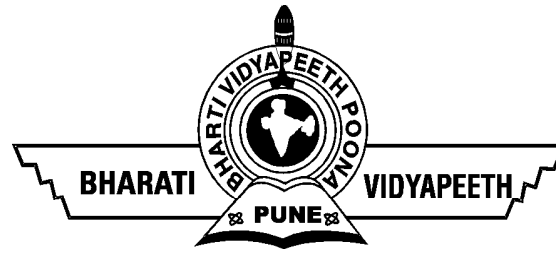
The Rational Unified Process (RUP)

Process for Using UML - But No Silver Bullet

- **main reason for using the iterative life cycle:**
 - Not all the needed information up front
 - Changes throughout the development period
- **expect**
 - To face some persistent, recurring risks
 - To discover new risks along the way
 - To do some rework; to throw away some lines of code
 - To change requirements along the way

- ✓ Background
- ✓ What is UML for?
 - ✓ for visualizing, specifying, constructing, and documenting models
- ✓ Building blocks of UML
- ✓ Things, Relationships and Diagrams
- ✓ Process for Using UML
- ✓ Use case-driven, Architecture-centric, & Iterative and incremental

- Activity Diagram



Use Case Modeling



Learning Objectives

- UseCase Modeling Overview
- Purpose
- Role & Frequency
- Software Requirement Meta Model
- Usecase Modeling
- Requirements Workflow Details
- UseCase Model Characteristics
- Actors
- Finding Actors
- Usecase
- Finding Usecase



Learning Objectives..

- Usecase Modeling Most Applicability
- Usecase Modeling Least Applicability
- Use of Advance Features
- Write Usecase Specifications
- Project Glossary
- Requirement Tracing
- Case Study



Use Case Modeling: Overview

The *Use Case Model* consists of the following:

- Actors
- Use cases
- Relationships
- System boundary



Use Case Modeling: Overview..

Steps of use case modeling:

- Find the system boundary
- Find the actors
- Find the use cases:
 - Describe how Actors and Usecase interacts
 - Package Usecases nad Actors
 - Draw Usecase diagrams
 - Evaluate your Results

Purpose

- To define the scope of the system
 - what will be handled by the system
 - what will be handled outside the system.
- To define who and what will interact with the system.
- To outline the functionality of the system.

Role & Frequency

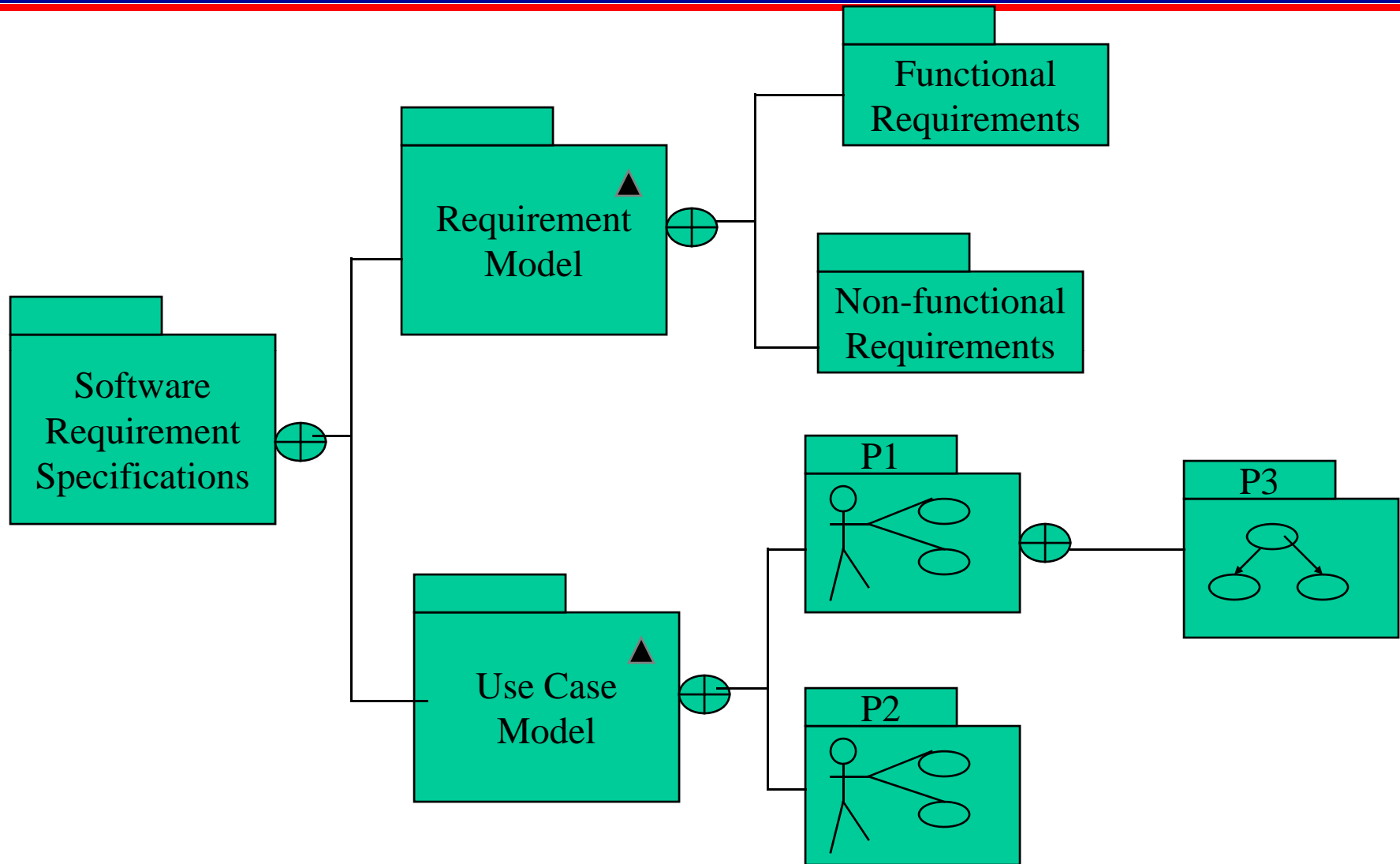
Role

✓ **System Analyst**

Frequency

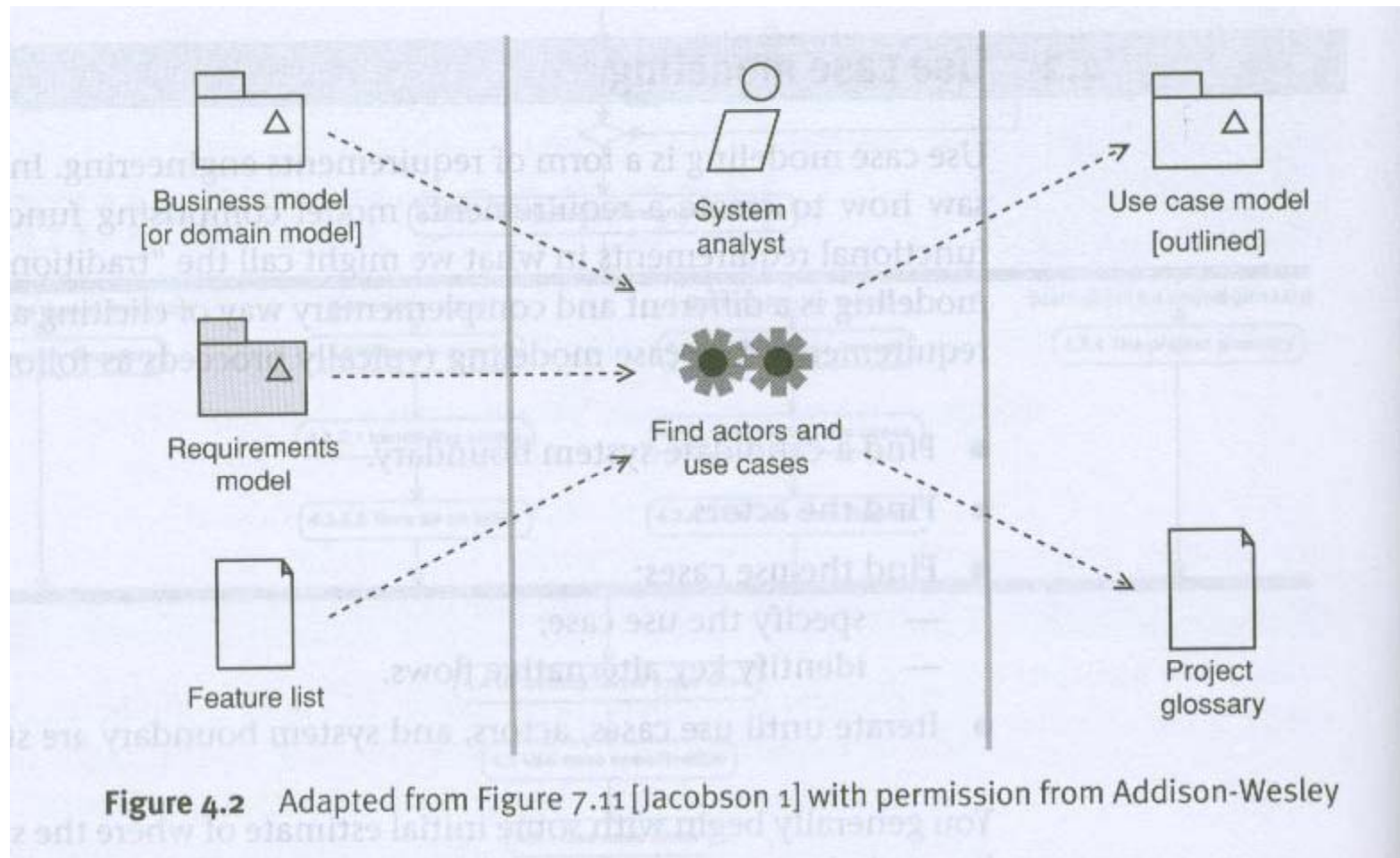
- As required,
- typically occurring multiple times per iteration
- most frequently in Inception and Elaboration iterations.

Software Requirement Meta Model

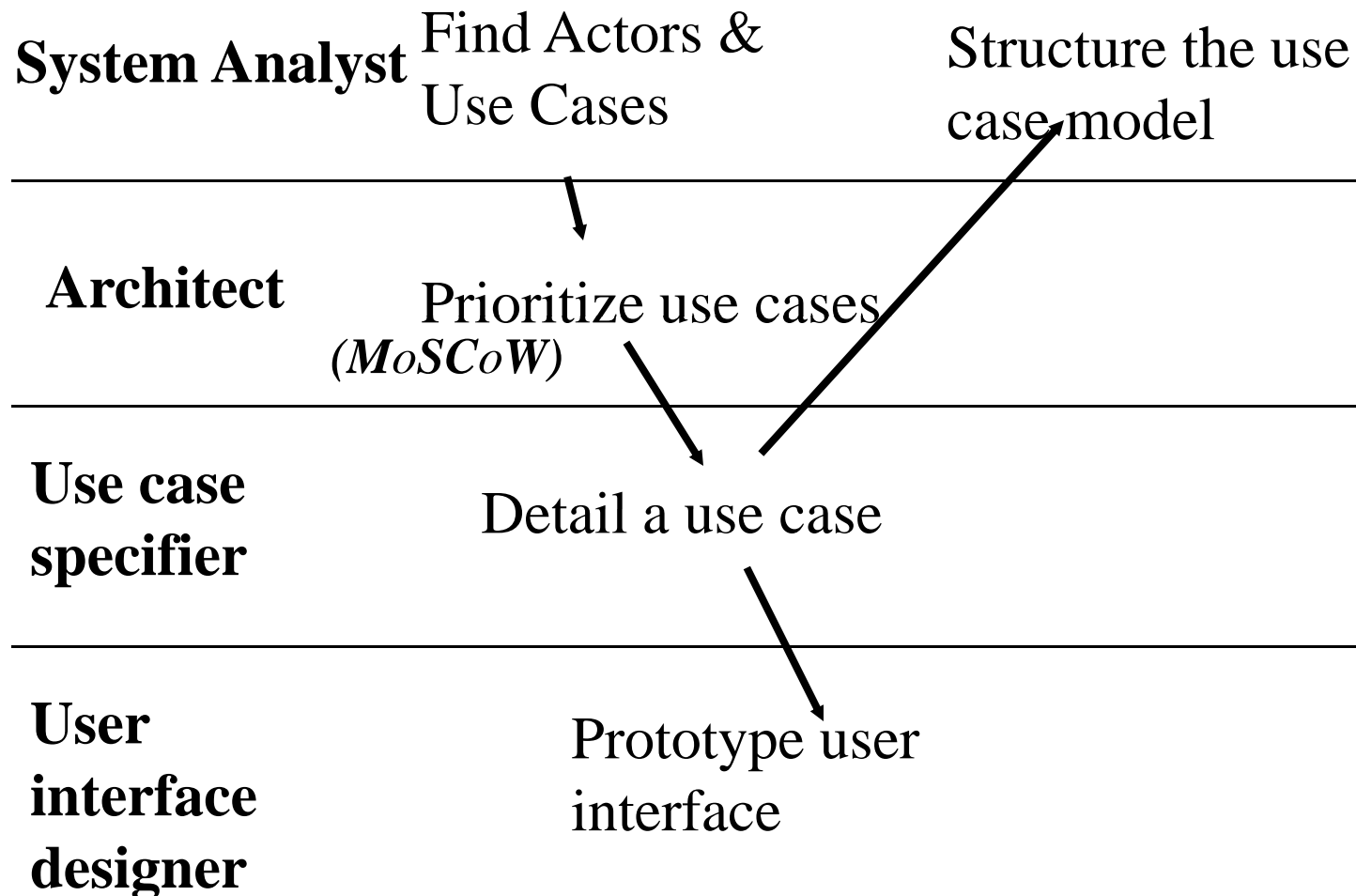


UseCase Modeling

Fig. 4.2 [Arlow & Neustadt 2005]



Requirements Workflow Details

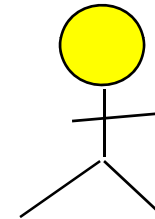


Use Case Model- Characteristics

- Need to be verified by users/managers
- Will be used as basis for rest of the development
- Therefore, It must be
 - Simple
 - Correct
 - Complete
 - Consistent
 - Verifiable, Modifiable, Traceable
 - Rankable (for iteration)

Actors

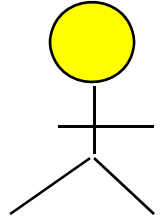
- a role taken by an external entity when interacting with the system directly
- An actor is a stereotype of class with its own icon
- An actor:
 - Is always external to the system
 - Interacts directly with the system
 - Represents a role played by people or things, not specific people or specific things



Finding Actors

Answer the questions

- Who will supply, use, or remove information?
- Who will use this functionality?
- Who is interested in a certain requirement?
- Where in the organization is the system used?
- Who will support and maintain the system?
- What are the system's external resources?
- What other systems will need to interact with this one?



Finding Actors...

several different aspects of a system's surroundings that you will represent as separate actors:

- Users who execute the system's main functions.
- Users who execute the system's secondary functions, such as system administration.
- External hardware the system uses.
- Other systems interacting with the system

Use Case

- According to Rumbaugh, a *use case* is
 - “a specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors”
- Use cases:
 - Are always started by an actor
 - Are always written from an actor’s point of view

Use Cases...

- Names of use cases should be verb phrases
- Candidate use cases can be discovered starting from the list of actors (how they interact with the system?)
- Finding use cases is an iterative process

Finding Use Cases...

Questions you can ask to identify use cases:

- What functions a specific actor wants from the system?
- Does the system store and retrieve information? If yes, which actors are involved?
- Are any actors notified when the system changes its state?
- Are any external events that affect the system? What notifies the system about these events?

Describe How Actors and Use Cases Interact

- to show how actors relate to the use case,
- must define a communicates-association
- navigable in the same direction as the signal transmission between the actor and the use case



Package Use Cases and Actors

- Interact with the same actor.
- Have include- or extend-relationships between each other.
- Are all optional, and are offered by the system together or not at all.

Use Case Diagram

- Shows the system boundary,
- the use cases internal to the system,
- the actors external to the system, e.g.

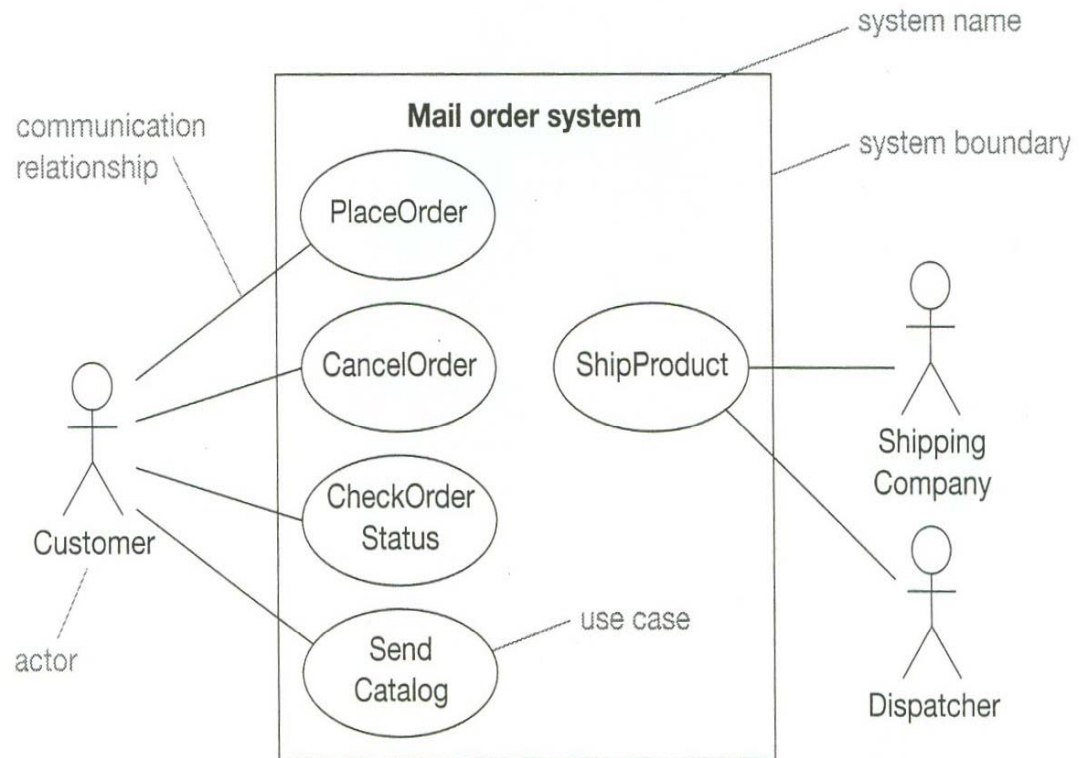


Figure 4.6

Use Case Specification...

- The output of this activity is
 - a more detailed use case
 - consists at least of the use case name and use case specification.
- Most common template for *use case specification*,

Use Case Specification...

use case name	Use case: PaySalesTax
use case identifier	ID: 1
brief description	Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
the actors involved in the use case	Primary actors: Time
	Secondary actors: TaxAuthority
the system state before the use case can begin	Preconditions: 1. It is the end of the business quarter.
the actual steps of the use case	Main flow: <i>implicit time actor</i> 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
the system state when the use case has finished	Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
alternative flows	Alternative flows: None.

Figure 4.8

Use Case Specification ...

Use case: ManageBasket
ID: 2
Brief description: The Customer changes the quantity of an item in the basket.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The shopping basket contents are visible.
Main flow: 1. The use case starts when the Customer selects an item in the basket. 2. If the Customer selects "delete item" 2.1 The system removes the item from the basket. 3. If the Customer types in a new quantity 3.1 The system updates the quantity of the item in the basket.
Postconditions: None.
Alternative flows: None.

Branching
using IF

Use Case Specification..

Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: <ol style="list-style-type: none"> 1. The use case starts when the Customer selects "find product". 2. The system asks the Customer for search criteria. 3. The Customer enters the requested criteria. 4. The system searches for products that match the Customer's criteria. 5. If the system finds some matching products then <ol style="list-style-type: none"> 5.1 For each product found <ol style="list-style-type: none"> 5.1.1 The system displays a thumbnail sketch of the product. 5.1.2 The system displays a summary of the product details. 5.1.3 The system displays the product price. 6. Else <ol style="list-style-type: none"> 6.1 The system tells the Customer that no matching products could be found.
Postconditions: None.
Alternative flows: None.

**Repetition
using FOR**

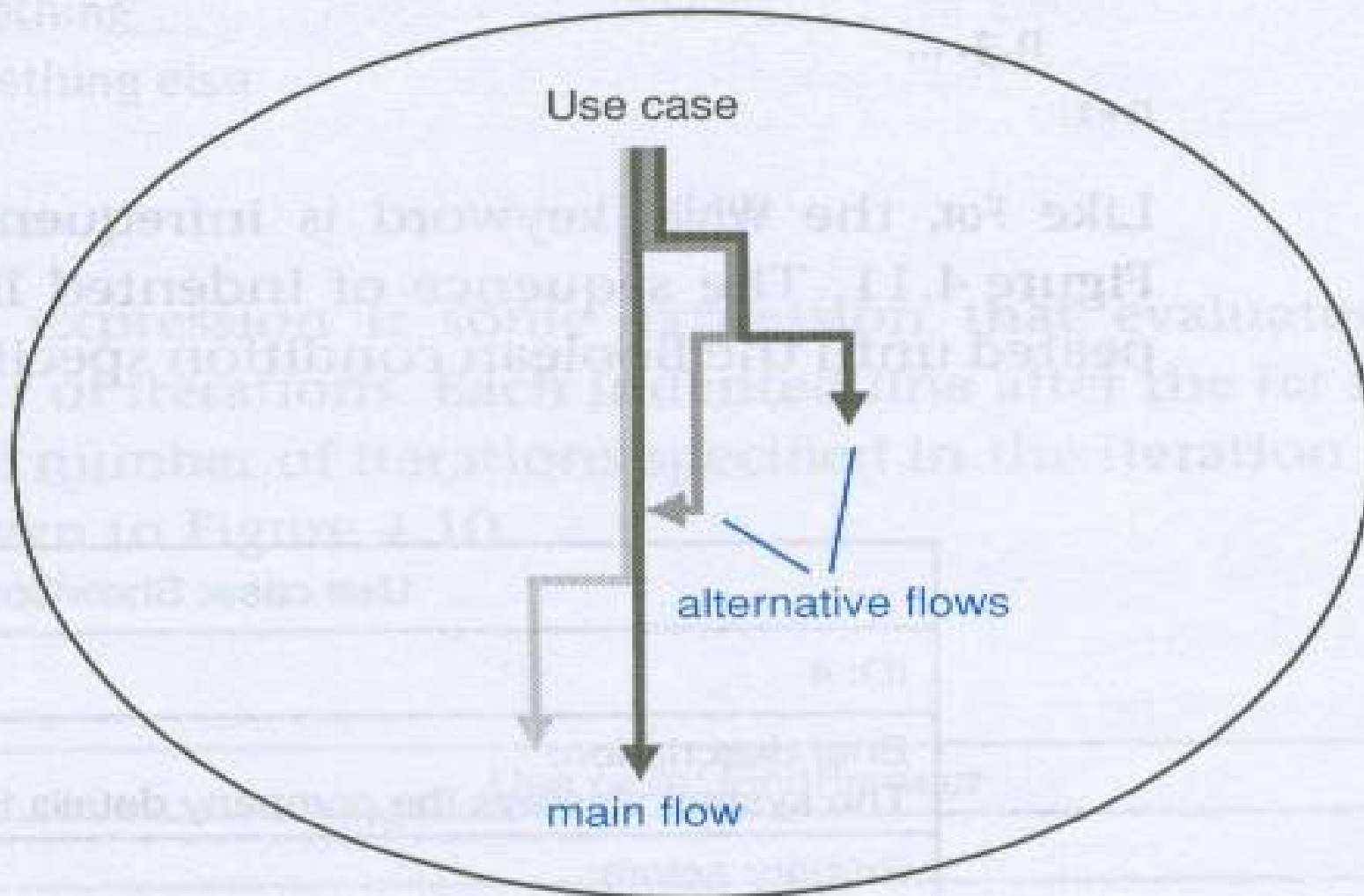
Figure 4.10

Use Case Specification..

Use case: ShowCompanyDetails	
ID: 4	
Brief description: The system displays the company details to the Customer.	
Primary actors: Customer	Repetition using While
Secondary actors: None.	
Preconditions: None.	
Main flow: 1. The use case starts when the Customer selects "show company details". 2. The system displays a web page showing the company details. 3. While the Customer is browsing the company details 3.1 The system plays some background music. 3.2 The system displays special offers in a banner ad.	
Postconditions: 1. The system has displayed the company details. 2. The system has played background music. 3. The system has displayed special offers.	
Alternative flows: None.	

Figure 4.11

Modelling Alternative Flows





Evaluate your Results

- If all necessary use cases are identified.
- If any unnecessary use cases are identified.
- If the behavior of each use case is performed in the right order.
- If each use case's flow of events is as complete as it could be at this stage.
- If the survey description of the use-case model makes it understandable.



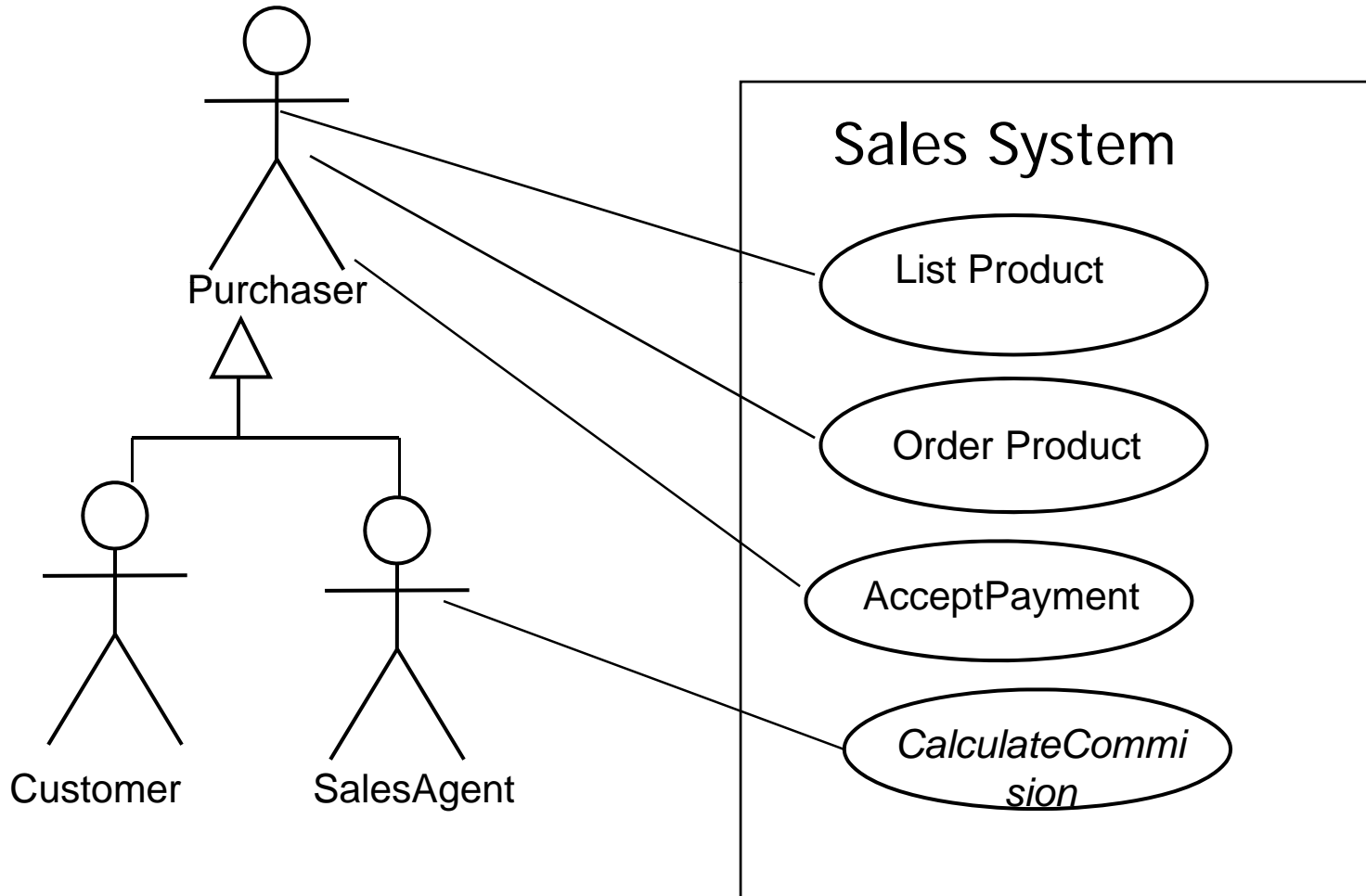
Advanced Use Case Modelling

- Start with the **priority** ones
- Add structure to the use case model:
 - identify generalization in Actors
 - identify generalization in use cases
 - include and extend relationships

Generalization - Actor

- Shows that one actor can participate in all the associations with use cases that the more specific actor can plus some additional use cases
 - Parent Actor more generalized and children are more specialized
 - Can substitute child actor in place of Parent Actor-substitutability principle
 - Parent actor is often abstract
 - Child Actors are concrete
 - Generalization can simplify use case diagram

Eg. Generalization Actor

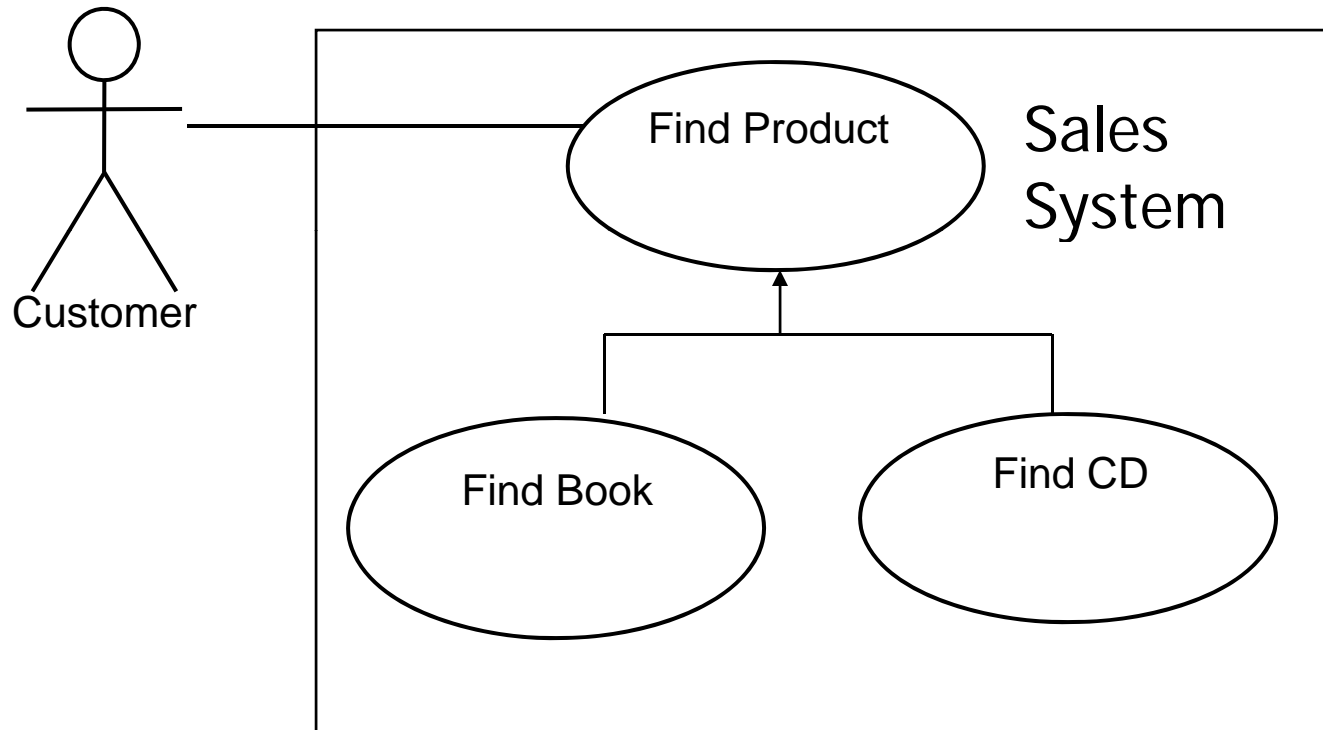


Generalization – Usecase..

Allow to factor out features that are common to two or more use cases into a parent usecase

- Child UC inherit all features of parent UC
- Child UC may add new features
- Child UC may override parent features except relationship and extension points

Eg. Generalization Usecase



Generalization – Usecase..

Use case features	Inherit	Add	Override
Relationship	Y	Y	N
Extention point	Y	Y	N
Preconditions	Y	Y	Y
Postcondition	Y	Y	Y
Step in main flow	Y	Y	Y
Alternative flow	y	Y	Y

Generalization – Usecase

- Simple tag conventions are
 - ✓ Inherited without change-3.(3.)
 - ✓ Inherited and renumbered- 6.2(6.1)
 - ✓ Inherited and overridden- 1.(01.)
 - ✓ Inherited, overridden and renumbered- 5.2(o5.1)
 - ✓ Added-6.3.

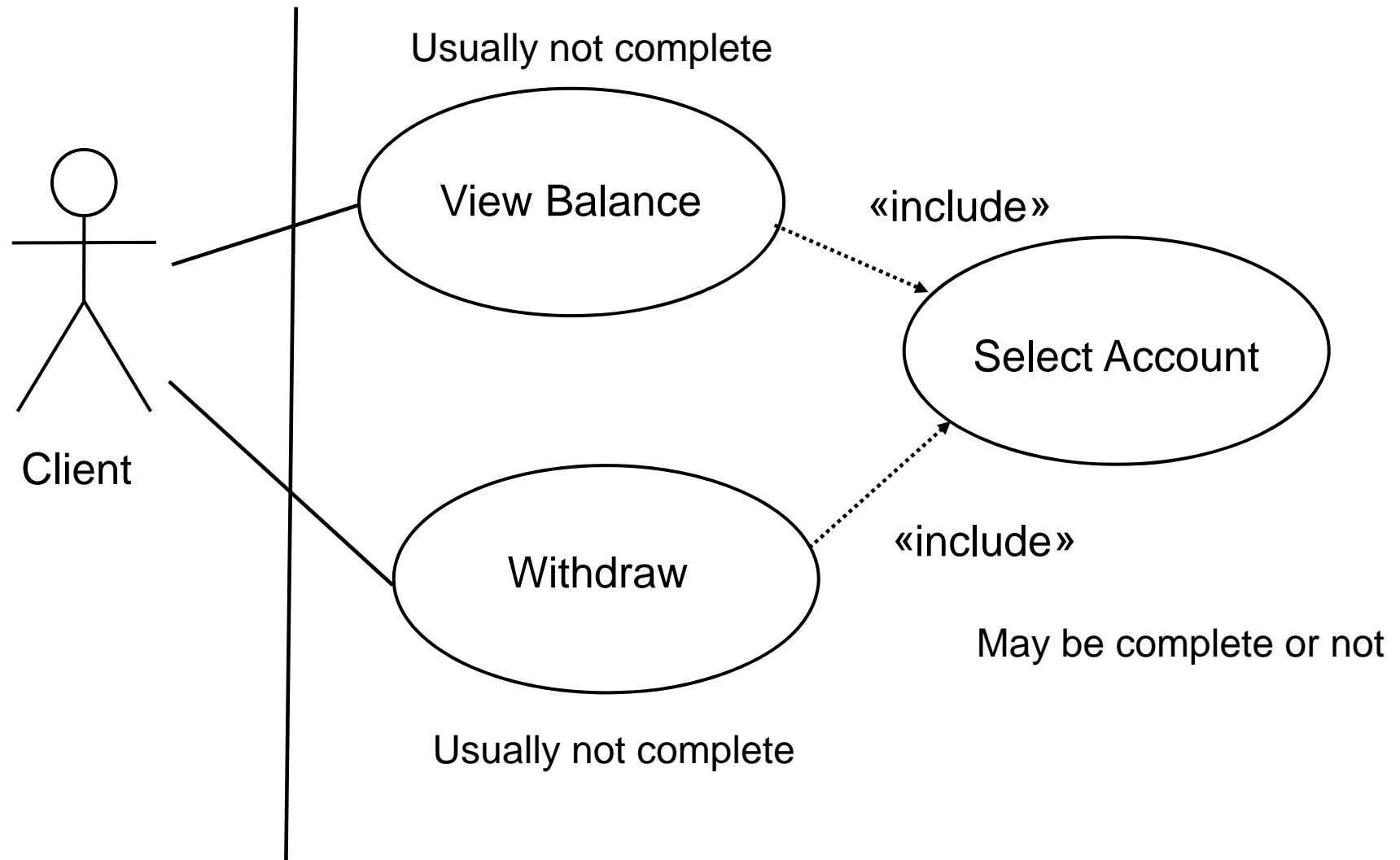
Usecase Dependencies

- Extend and Include relationships between use cases
- Shown as stereotyped dependencies
- Stereotypes are written as text strings in guillemots:
 - «**extend**»
 - «**include**»

<<Include>>

- used to separate out a sequence of behaviour that is used in **more than one** use cases
 - Including usecase is the base usecase
 - Included usecase is the inclusion usecase
 - Base usecase is not complete without inclusion
 - Inclusion may_be complete/ incomplete

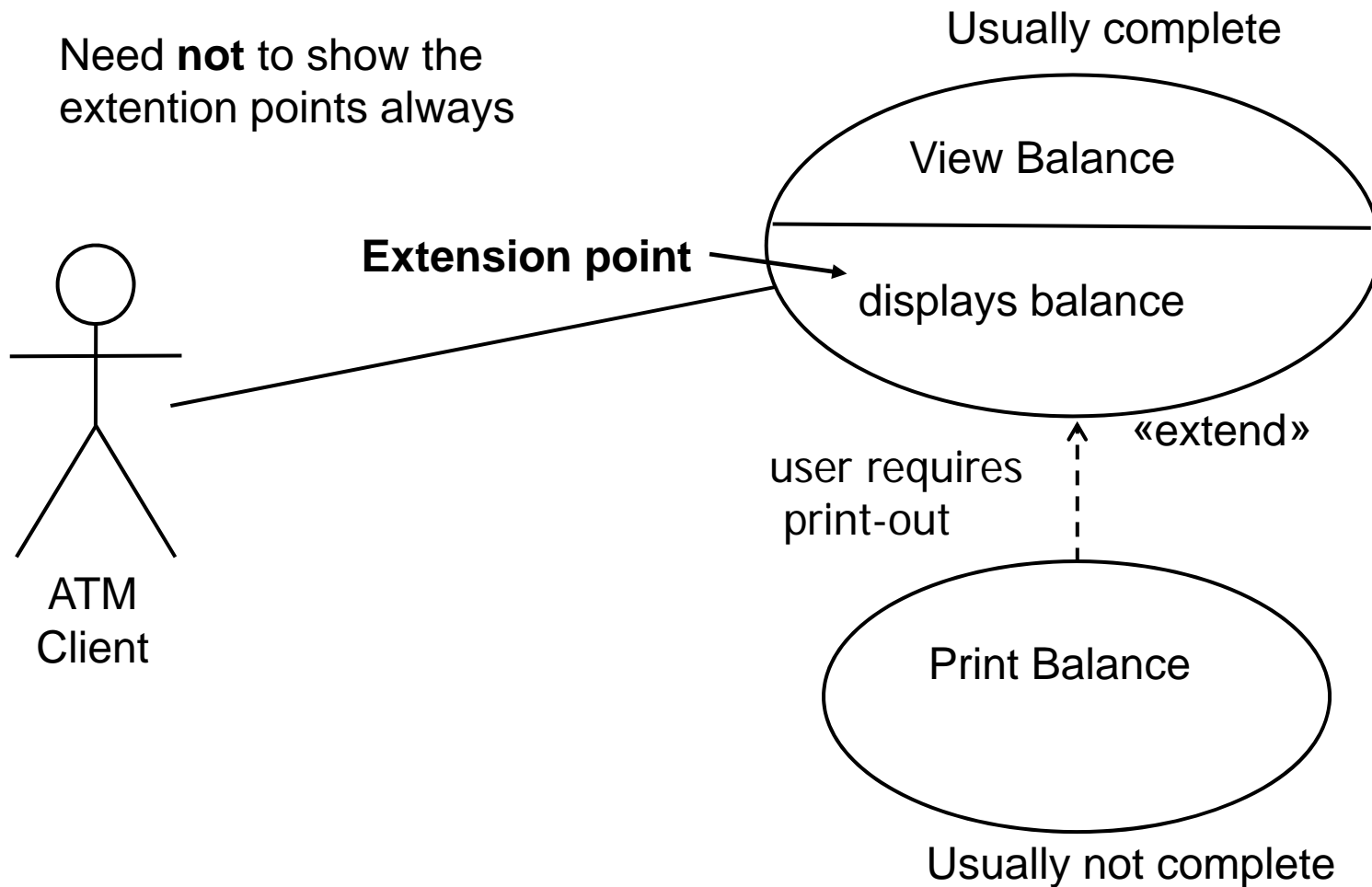
Include



Extend

- One use case provides additional functionality that **may** be required in another use case
- Base usecase is generally complete without the insertion segment
- Extension points are generally not complete

Eg. Extend





Use case Modeling most Applicability

- Usecase modeling is most appropriate for system that
 - Dominated by functional requirements
 - Have many types of users
 - Have many interfaces to other systems



Use case Modeling least Applicability

- Dominated by non-functional requirements
- Have few users
- Have few interfaces

Use of Advance Features

Actor generalization-

- Only where it simplifies the model

Use case generalization

- Consider not using, or only using with abstract parents

<<include>>

- Where it simplifies model
- Beware of overuse, as it makes functional decomposition

<<extend>>

- Use carefully to ensure all stakeholder exposed to the model understand and agree on its semantics



Hints to Write Use Case

- Keep use case short and simple
- Focus on the what not the how
- Avoid functional decomposition

Project Glossary

The *project glossary*

- Important project artifact
- Provides a dictionary of key business terms
- Captures business language and jargon
- Should resolve synonyms and homonyms
- Should be understandable by all stakeholders
- UML does not set a standard for the project glossary
- Synchronization between the project glossary and the UML model is needed

Requirements Tracing

Table 4.2

	Use case			
	UC ₁	UC ₂	UC ₃	UC ₄
R1	X			
R2		X	X	
R3			X	
R4				X
R5	X			



Putting the UML to Work

The ESU University wants to computerize their registration system

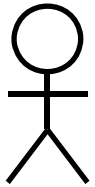
- The Registrar sets up the curriculum for a semester
 - ✓ One course may have multiple course offerings
- Students select 4 primary courses and 2 alternate courses
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to receive their course offering rosters
- Users of the registration system are assigned passwords which are used at logon validation



System Boundary

- Semester curriculum
- Student registration for semester

Actors-ESU



Registrar



Professor



Student

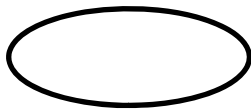


Billing System

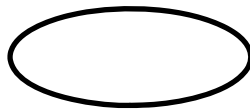
ESU-Use Cases

Actors are examined to determine their needs

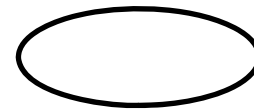
- **Registrar** -- maintain the curriculum
- **Professor** -- request roster
- **Student** -- maintain schedule
- **Billing System** -- receive billing information from registration



Maintain Curriculum

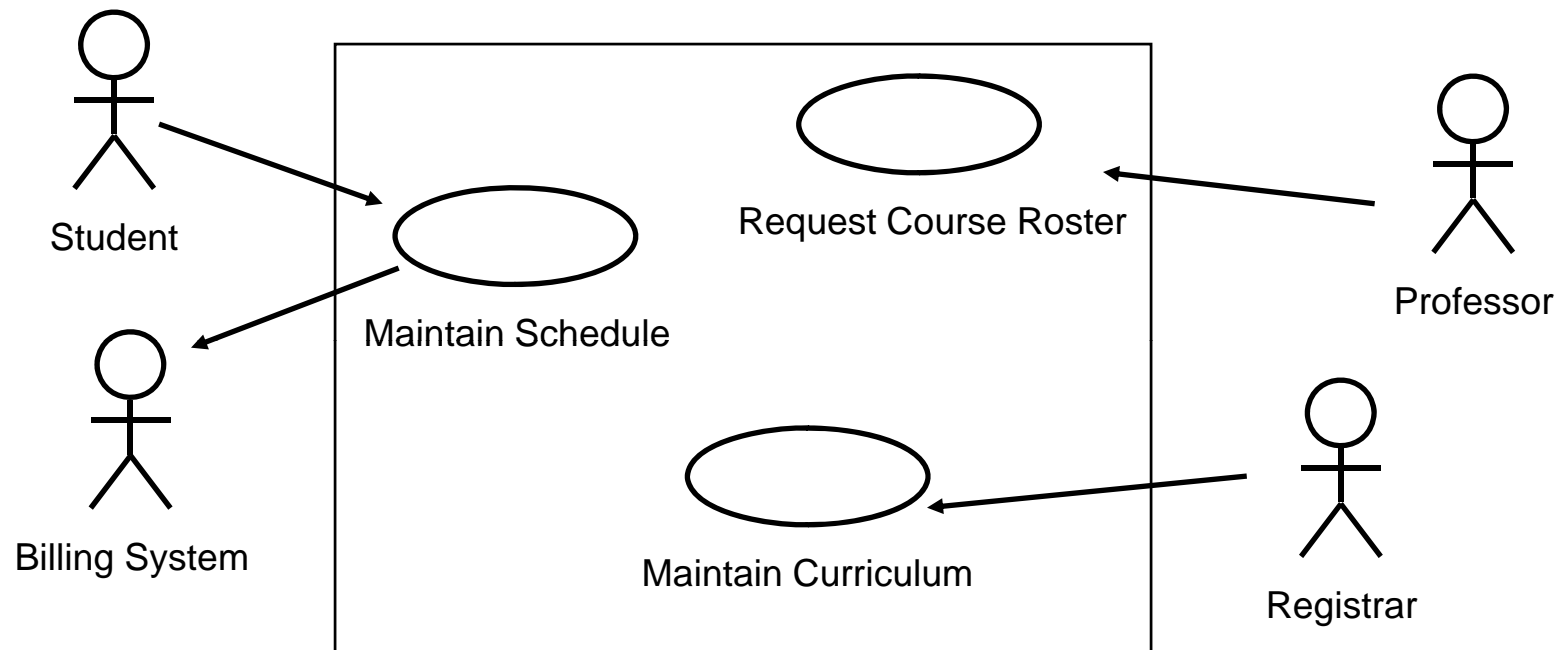


Request Course Roster



Maintain Schedule

Use Case Diagram





Use case-Maintain Curriculum..

ID:2

Brief Description:

- Registrar setup curriculum for semester

Primary Actors:

- Registrar

Secondary Actors:

- None

Precondition:

1. Registrar logs onto the Registration System

Use case-Maintain Curriculum..

Main Flow:

1. prompts the Registrar to select the current semester or a future semester
2. The Registrar enters the desired semester
3. system prompts the registrar to select the desired activity: ADD, DELETE, REVIEW, or QUIT
4. If the activity selected is ADD
 - 5.1 Add a Course subflow is performed
5. If the activity selected is DELETE
 - 5.1 Delete a Course subflow is performed.
6. If the activity selected is REVIEW
 - 6.1 Review Curriculum subflow is performed.
7. If the activity selected is QUIT
 - 7.1 the use case ends.



Use case-Maintain Curriculum

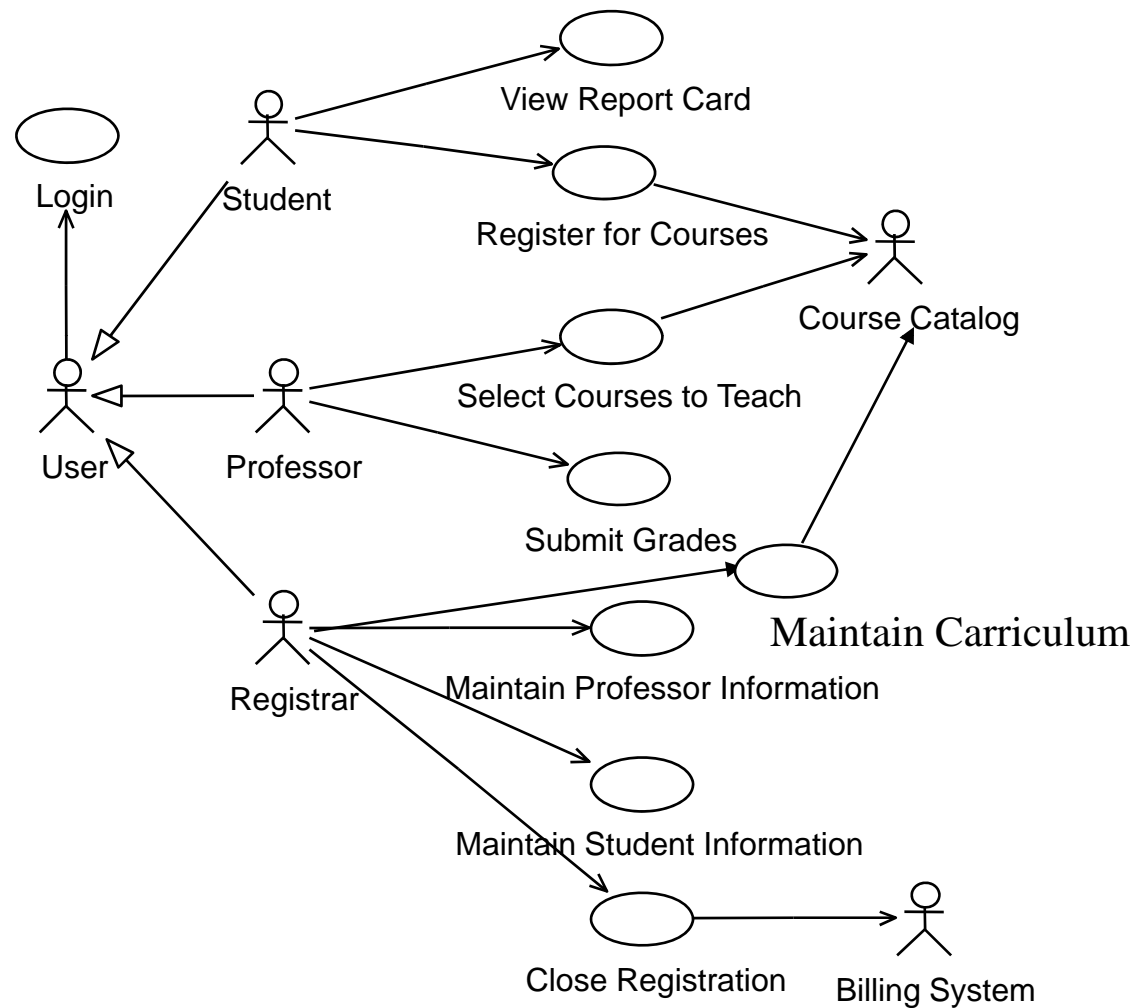
Post conditions:

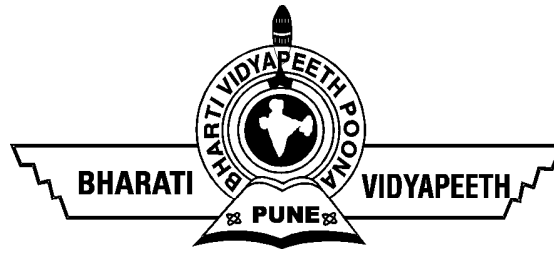
- None

Alternative Flow:

- None

Use Case Diagram





Activity Diagrams

modeling the logic captured by a single
use case or usage scenario, or for
modeling the detailed logic of a
business rule



Learning Objectives

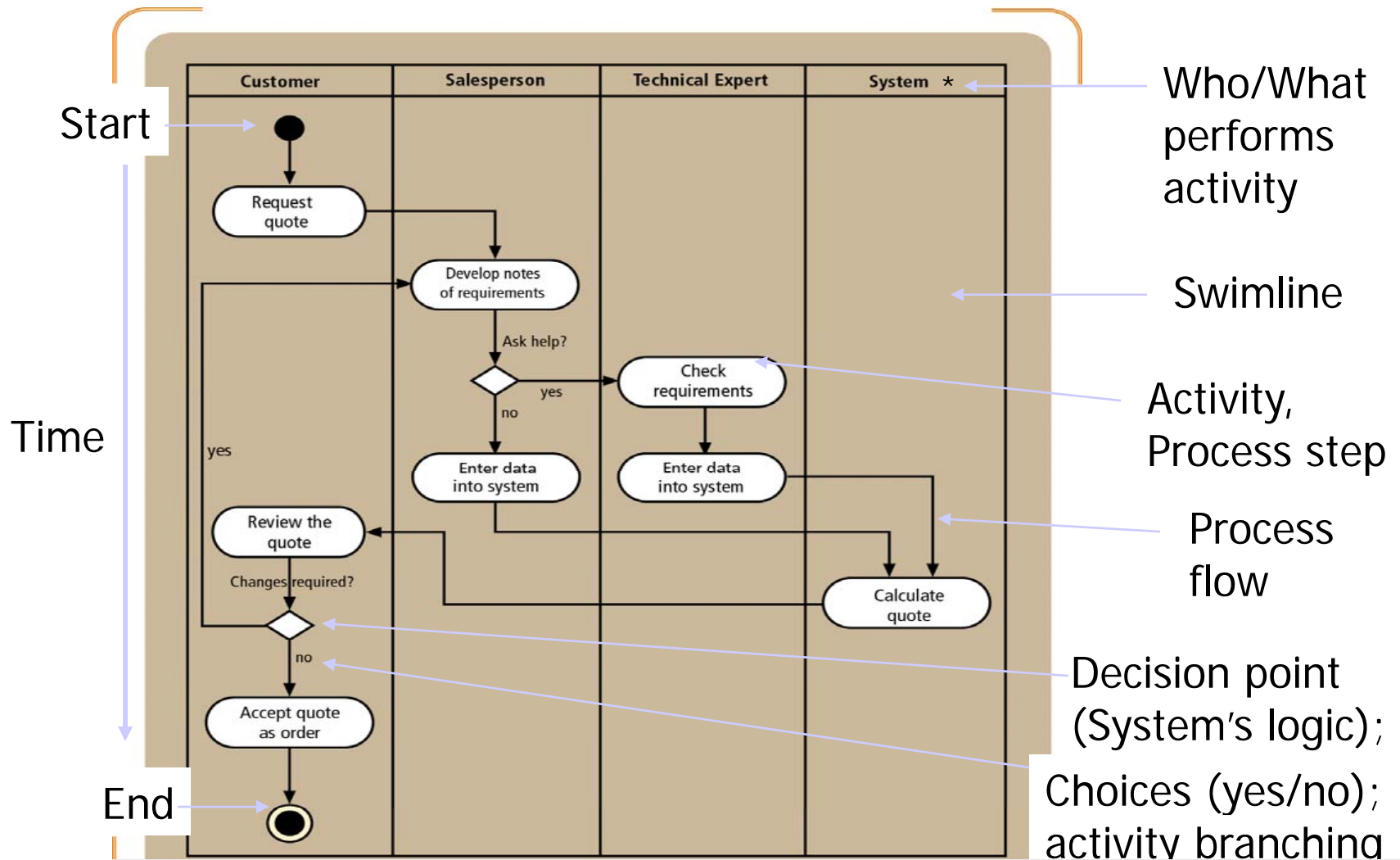
- Activity diagram concept
- Elements of activity diagram
- Reading activity diagrams
- Process logic in activity diagram
- Creating activity diagrams
- Case Study



Activity Diagram (AD) Concept

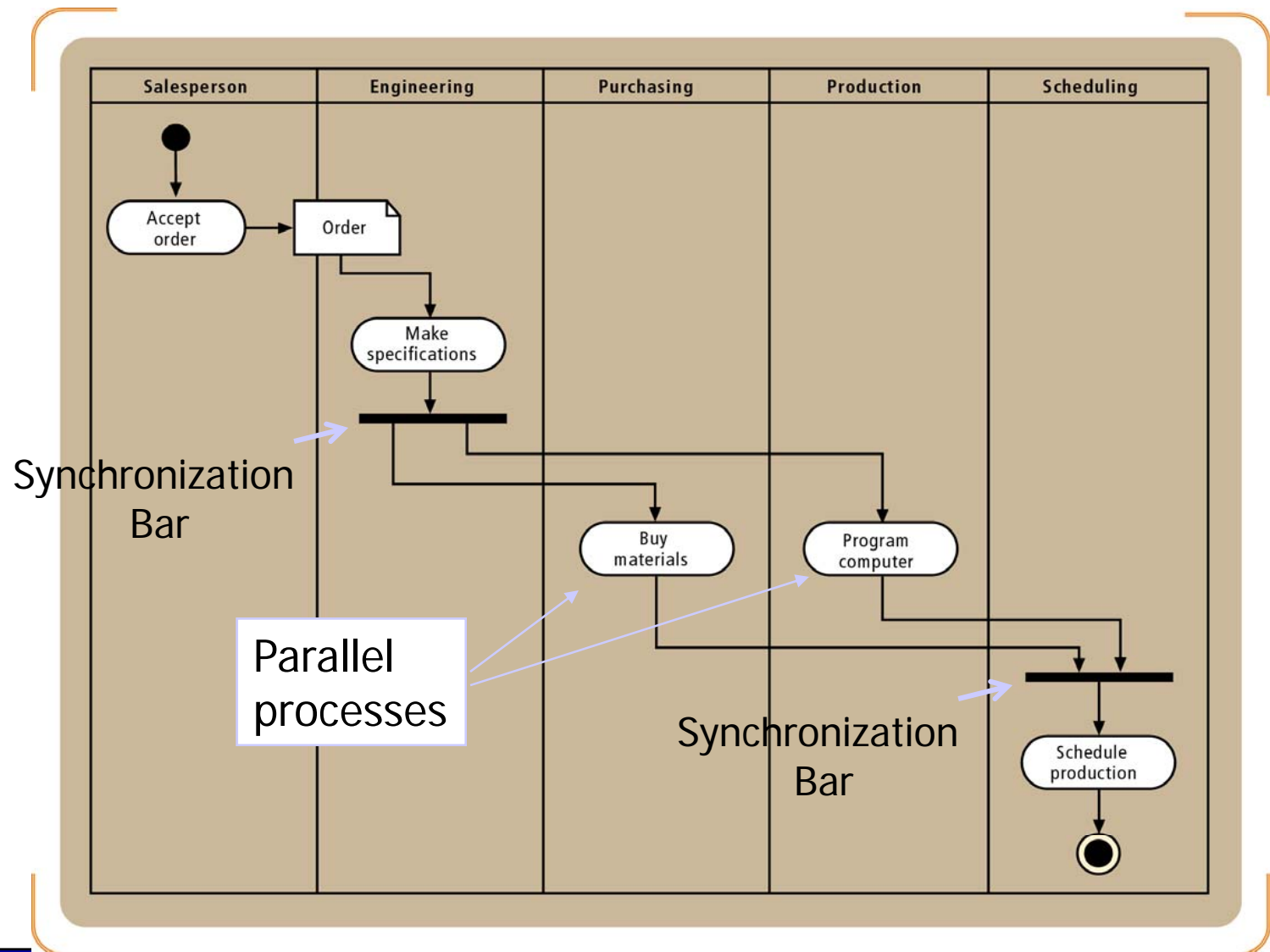
- AD used for requirements determination
- AD depicts:
 - **process (workflow)** – manual & automated
 - **process logic** – time order of process steps (activities) & decision points
 - **process performers**
- AD resembles old flow charts and somewhat data flow diagram

AD elements – Quote process

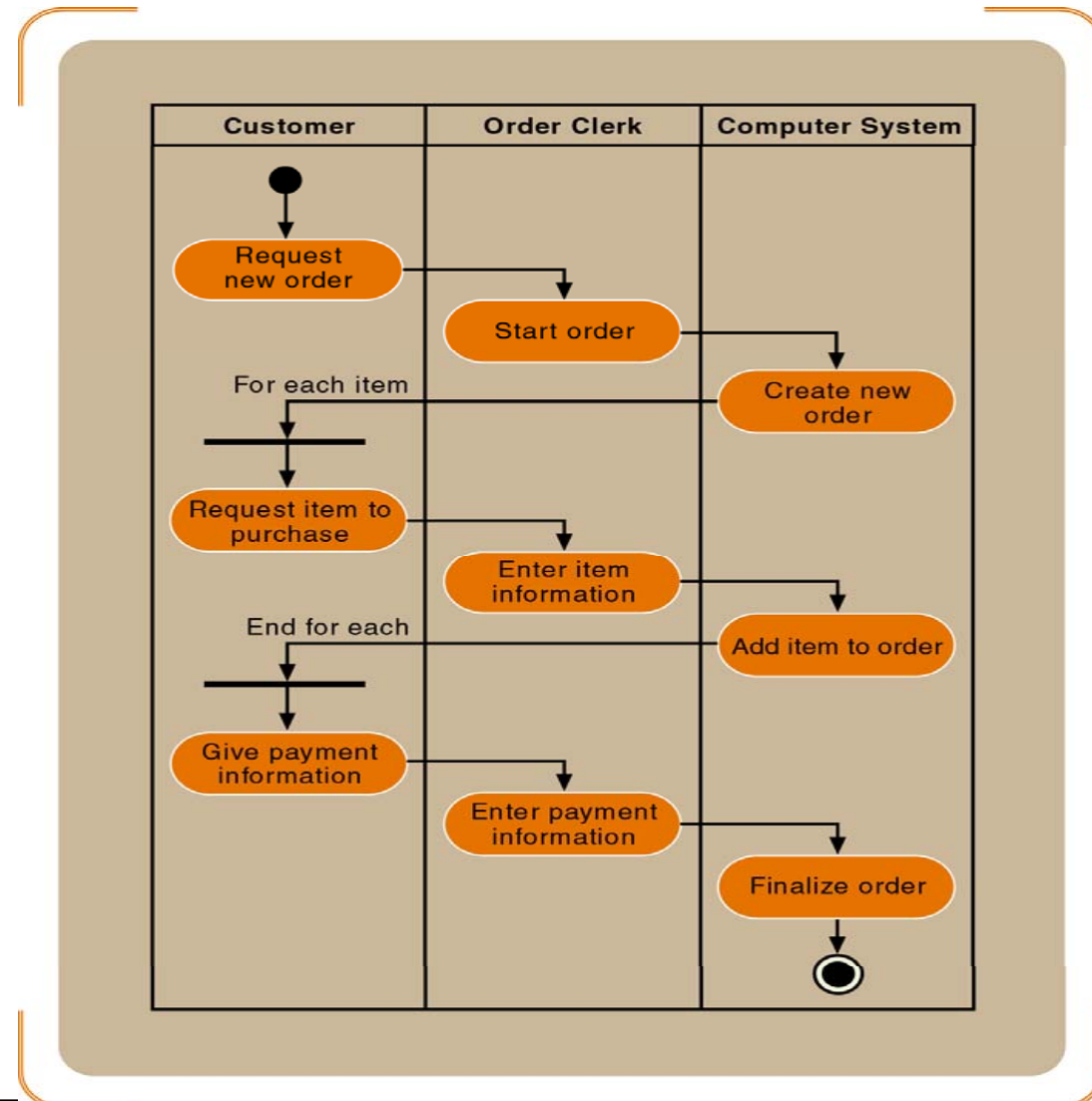


* System=computer, computer software and hardware; same as in use case descriptions

AD elements (cont.)



Telephone Order Process – simpler (cont.)



Process Logic

Sequential (step 1 → step 2)

Conditional, If-Then (decision points: if condition A, then step n)

Iteration, loops (feedback into a previous step while certain condition persists)

- Slide 145, feedback from the “Change required?” decision point)
- Slide 147, loop between bars *For each item to End of each*



What AD does not Show?

- Data passed between steps
- Objects (directly, can be inferred)
- User interface

How to Create AD

1. Identify activities (steps) of a process
2. Identify who/what performs activities (process steps)
3. Draw swimlines
4. Identify decision points (if-then)
5. Determine if step is in loop (*For each...*, if-then based loop)
6. Determine if step is parallel
7. Identify order of activities, decision points

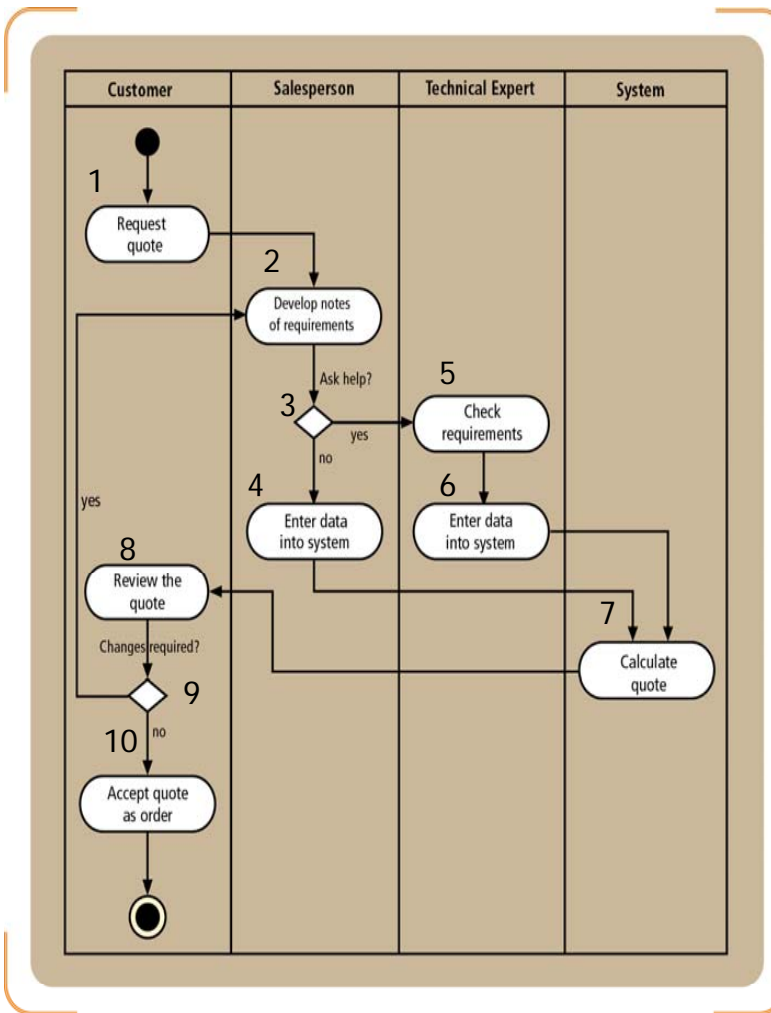
Continues...

How to create AD (cont.)

8. Draw the start point of the process in the swimline of the first activity (step)
9. Draw the oval of the first activity (step)
10. Draw an arrow to the location of the second activity
11. Draw subsequent activities, while inserting decision points and synchronization/loop bars where appropriate
12. Draw the end point after the last activity.

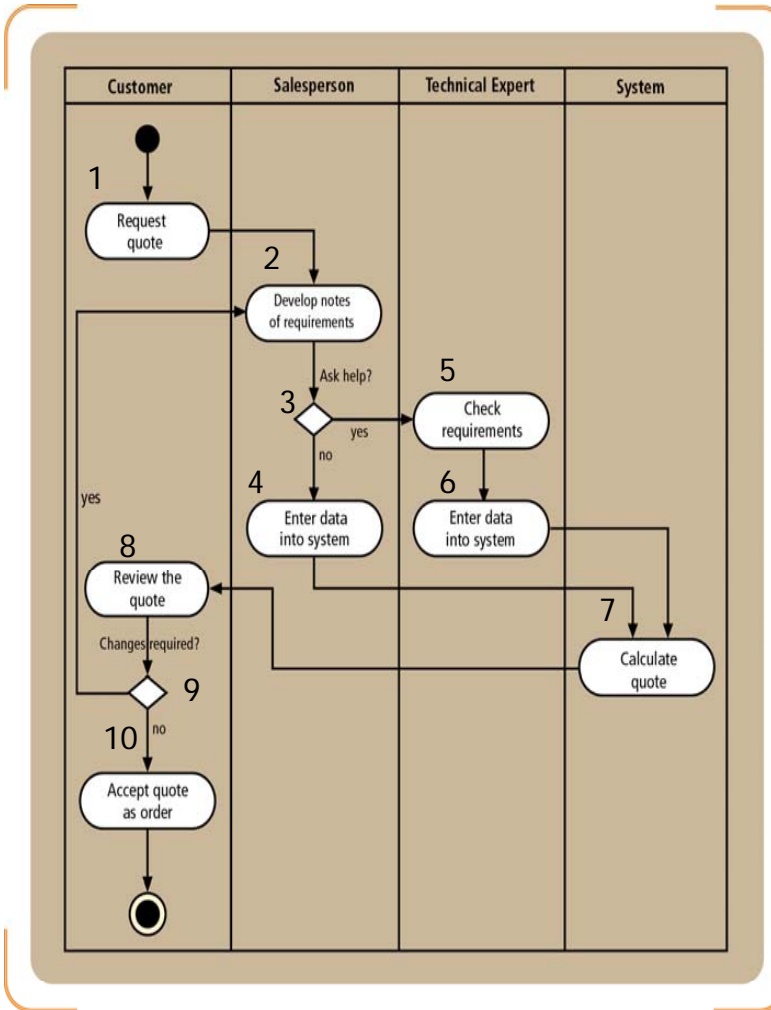
You can tabulate this information (see next slide).

How to Create AD (cont.)



Step ID	Process Step (Activity, Decision)	Who/What Performs Step	Parallel Activity	Loop	Preceding Step
1	Request quote	Customer	No	No	-
2	Develop requirement notes	Salesperson	No	Yes	1
3	Decision: Help?	Salesperson	-	Yes	2
4	Salesperson enters data	Salesperson	No	Yes	3
5	Check requirements	Technical Expert	No	Yes	3

How to Create AD (cont.)



Step ID	Process Step (Activity, Decision)	Who/What Performs Step	Parallel Activity	Loop	Preceding Step
6	Tech. expert enters data	Technical Expert	No	Yes	5
7	Calculate quote	System	No	Yes	4, 6
8	Review quote	Customer	No	Yes	7
9	Decision: Changes?	Customer	No	Yes	8
10	Accept quote as order	Customer	No	No	9

Case Study - Agate Ltd; B,M&F-85

- Agate is an advertising company
- Many companies are its clients
- Clients have advertising campaigns
- Each campaign includes one or more adverts
- Staff may work on several campaigns at a time
- Bonus is calculated depending on salary grade



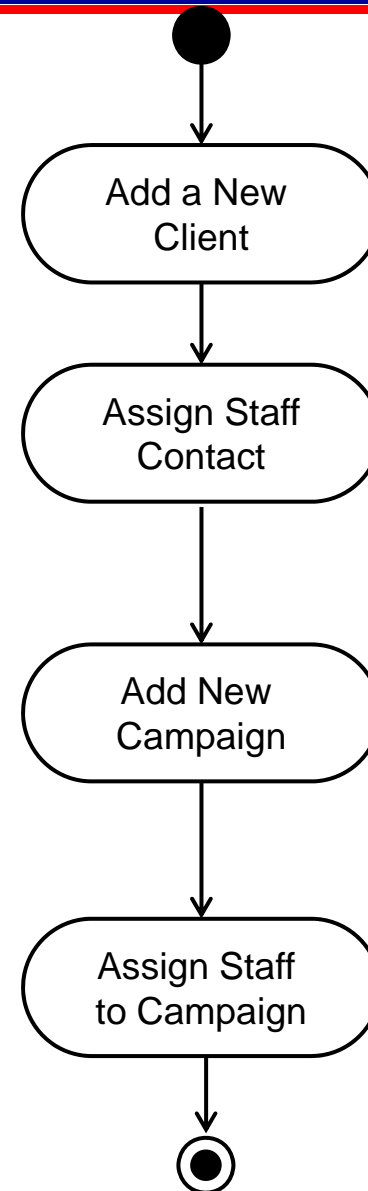
Drawing Activity Diagrams

Identify activities

- What happens when a new client is added in the Agate system?
 - ✓ Add a New Client
 - ✓ Assign Staff Contact
 - ✓ Add New Campaign
 - ✓ Assign Staff to Campaign

Organise the activities in order with transitions

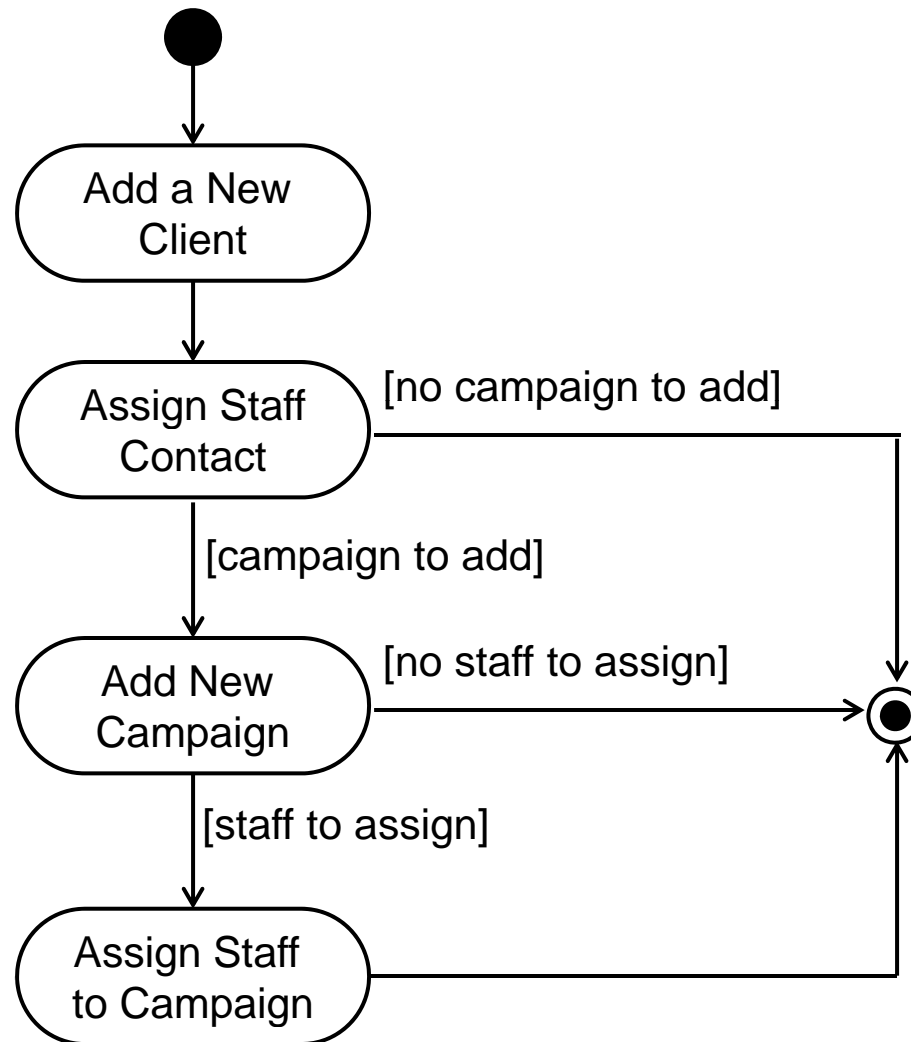
Drawing Activity Diagrams..



Drawing Activity Diagrams..

- Identify any alternative transitions and the conditions on them
 - sometimes there is a new campaign to add for a new client, sometimes not
 - sometimes they will want to assign staff to the campaign, sometimes not
- Add transitions and guard conditions to the diagram

Drawing Activity Diagrams..

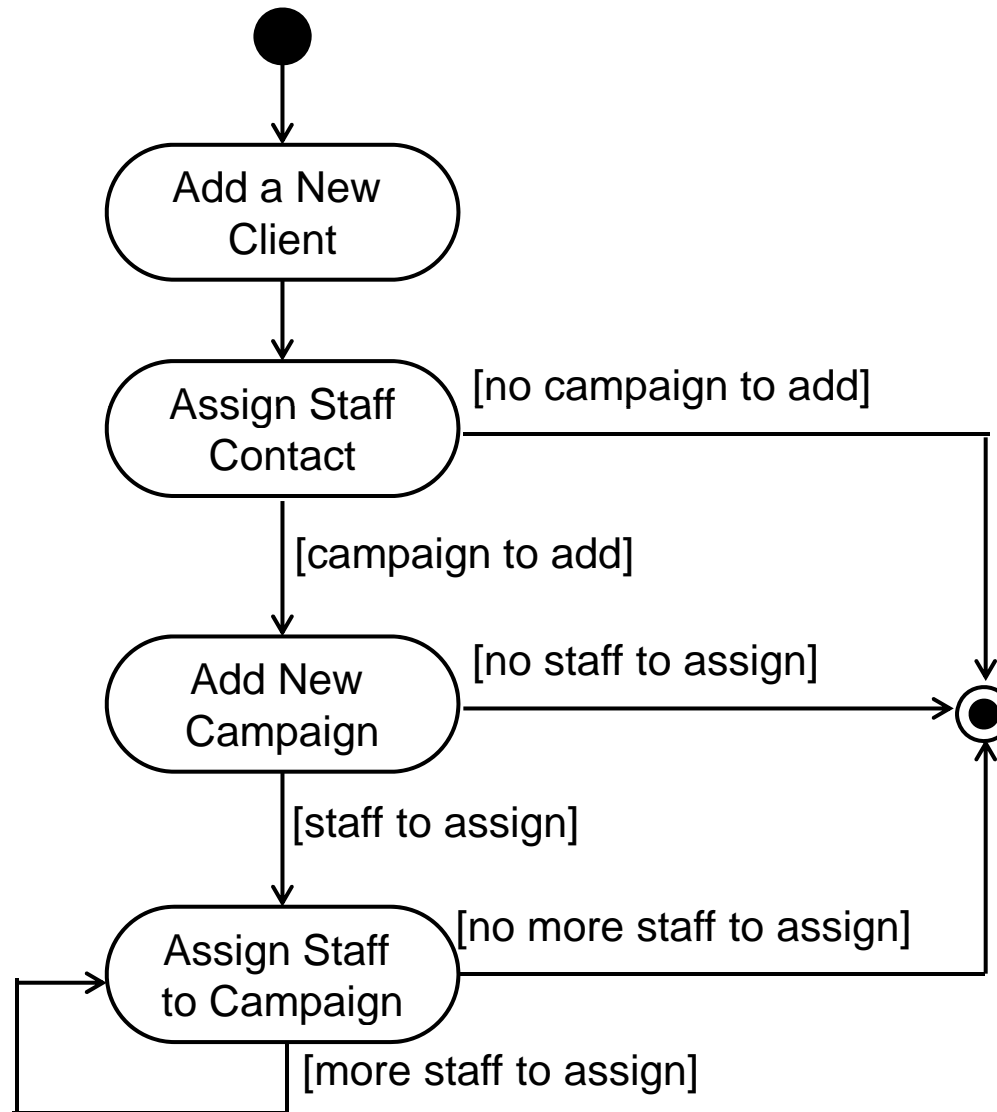




Drawing Activity Diagrams..

- Identify any processes that are repeated
 - they will want to assign staff to the campaign until there are no more staff to add
- Add transitions and guard conditions to the diagram

Drawing Activity Diagrams..

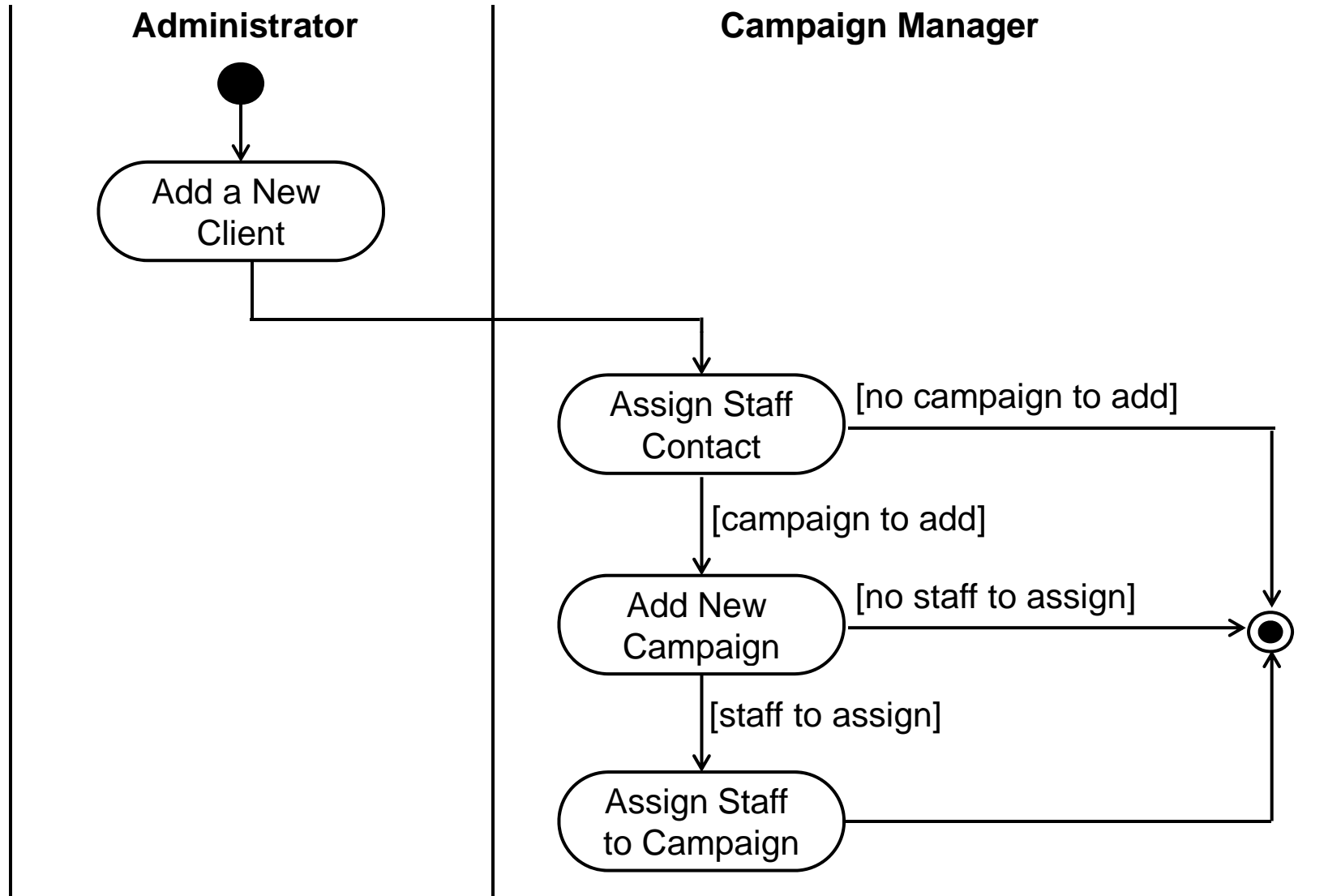




Drawing Activity Diagrams..

- Are all the activities carried out by the same person, organisation or department?
- If not, then add swim lanes to show the responsibilities
- Name the swim lanes
- Show each activity in the appropriate swim lane

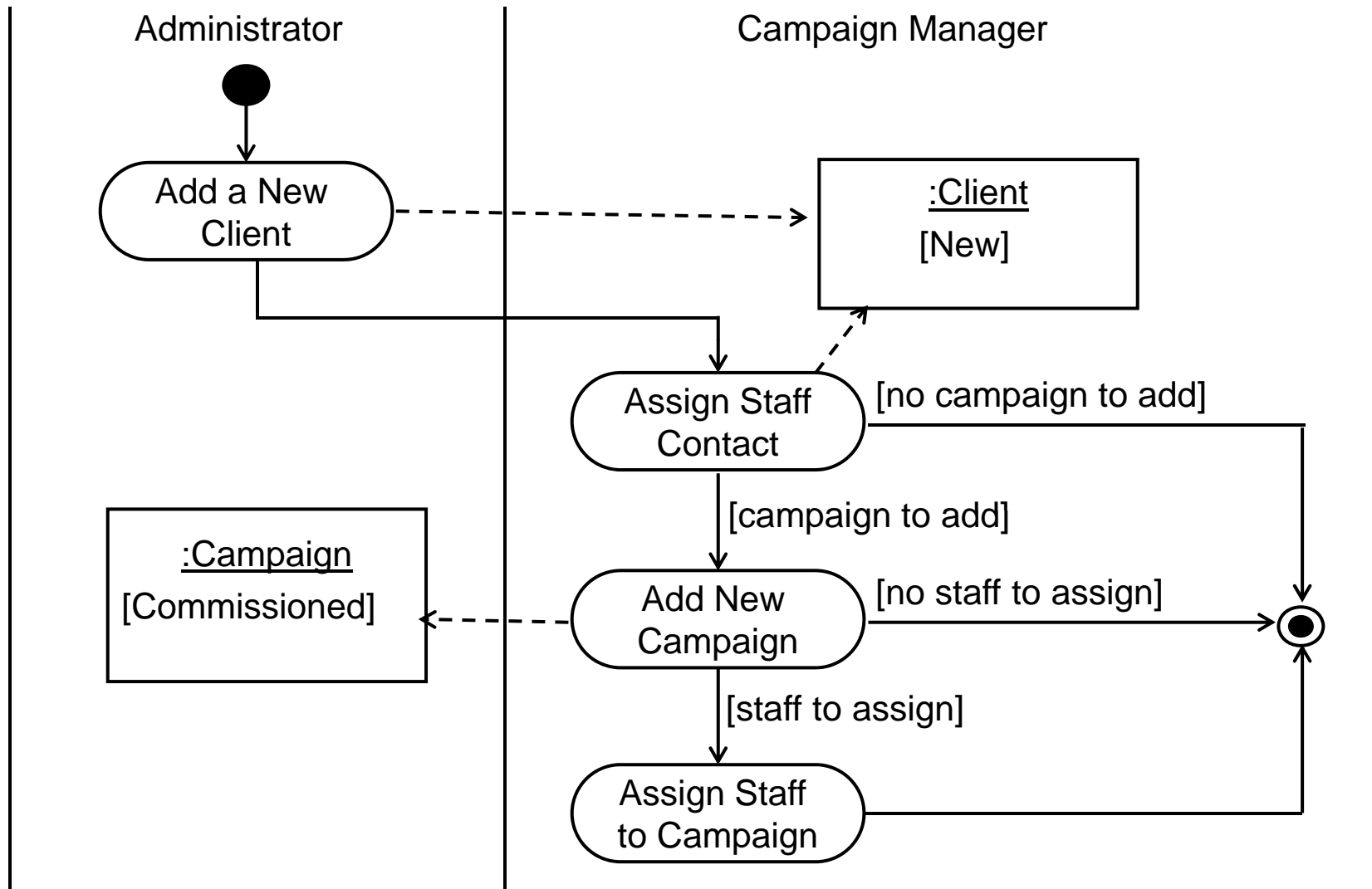
Drawing Activity Diagrams..



Drawing Activity Diagrams..

- Are there any object flows and objects to show?
 - these can be documents that are created or updated in a business activity diagram
 - these can be object instances that change state in an operation or a use case
- Add the object flows and objects

Drawing Activity Diagrams..



Exercise

Create an activity diagram based on the following narrative.

The purchasing department handles purchase requests from other departments in the company. People in the company who initiate the original purchase request are the "customers" of the purchasing department. A case worker within the purchasing department receives that request and monitors it until it is ordered and received. Case workers process the requests for purchasing products under \$1,500, write a purchase order, and then send it to the approved vendor. Purchase requests over \$1,500 must first be sent out for a bid from the vendor that supplies the product. When the bids return, the case worker selects one bid. Then, the case worker writes a purchase order and sends it to the approved vendor

Exercise

- **Develop an activity diagram based on the following narrative.**

The purpose of the Open Access Insurance System is to provide automotive insurance to car owners. Initially, prospective customers fill out an insurance application, which provides information about the customer and his or her vehicles. This information is sent to an agent, who sends it to various insurance companies to get quotes for insurance. When the responses return, the agent then determines the best policy for the type and level of coverage desired and gives the customer a copy of the insurance policy proposal and quote.

Sequence Diagram

Interaction Diagrams

- UML Specifies a number of interaction diagrams to **model dynamic aspects** of the system
- Dynamic aspects of the system
 - Messages moving among objects/ classes
 - Flow of control among objects
 - Sequences of events

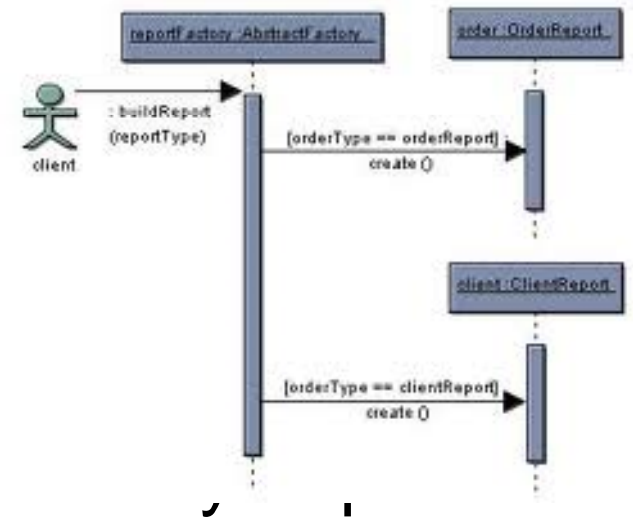


Dynamic Diagram Types

- **Interaction Diagrams** - Set of objects or roles and the messages that can be passed among them.
 - **Sequence Diagrams** - emphasize time ordering
 - **Communication Diagrams** - emphasize structural ordering
- **State Diagrams**
 - State machine consisting of states, transitions, events and activities of an object
- **Activity & Swimlane Diagrams**
 - Emphasize and show flow of control among objects

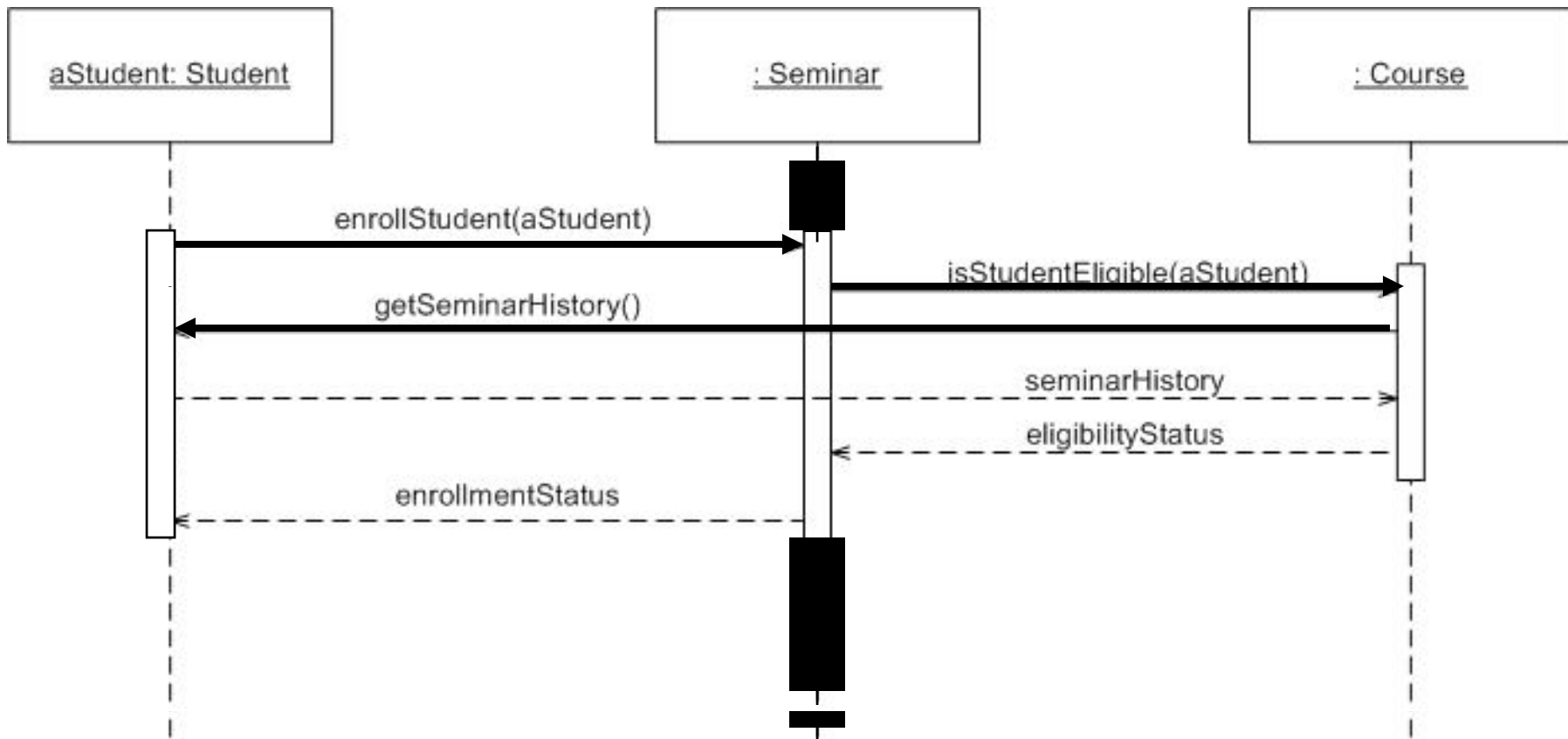
Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

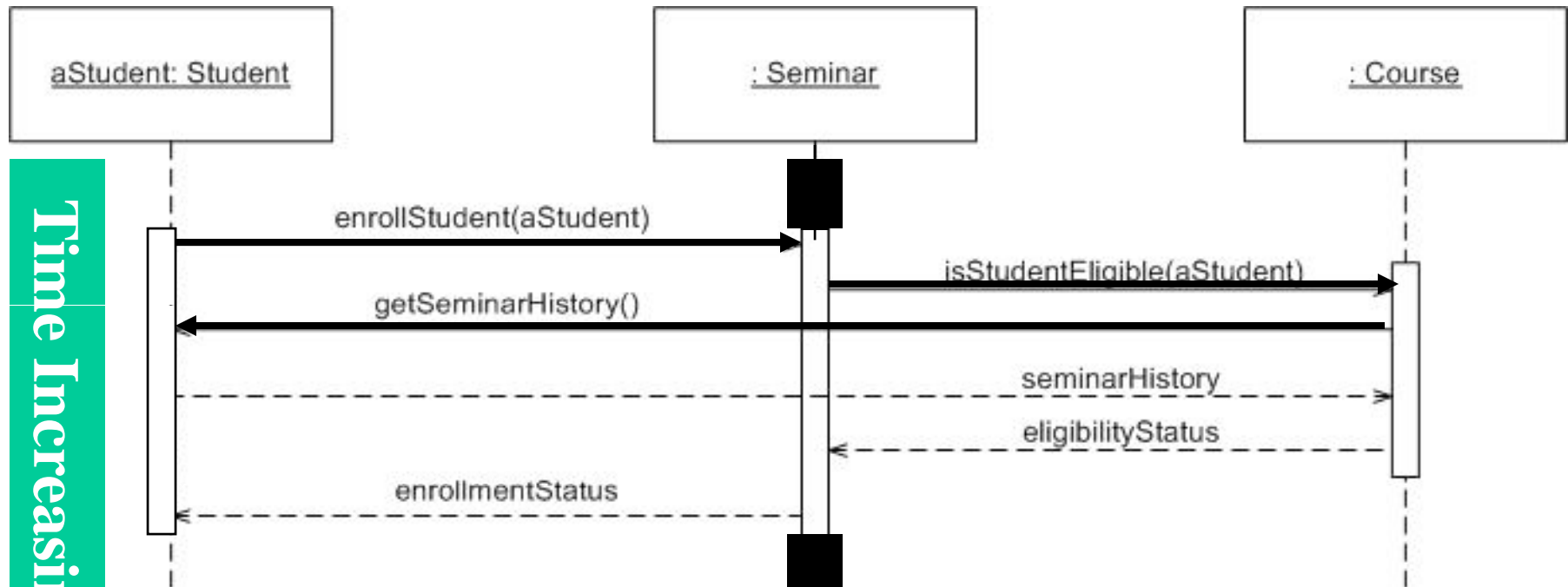


Emphasis on time ordering!

Sequence Diagram

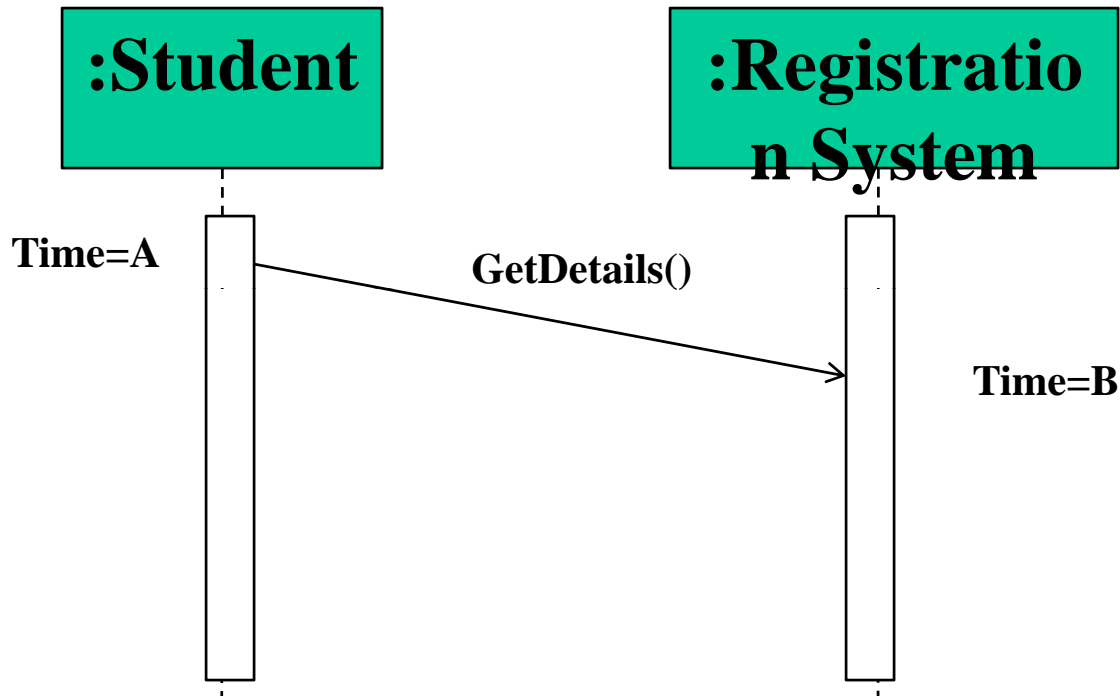


Sequence Diagram



All lines should be horizontal to indicate instantaneous actions. Additionally if ActivityA happens before ActivityB, ActivityA must be above activity A

Diagonal Lines



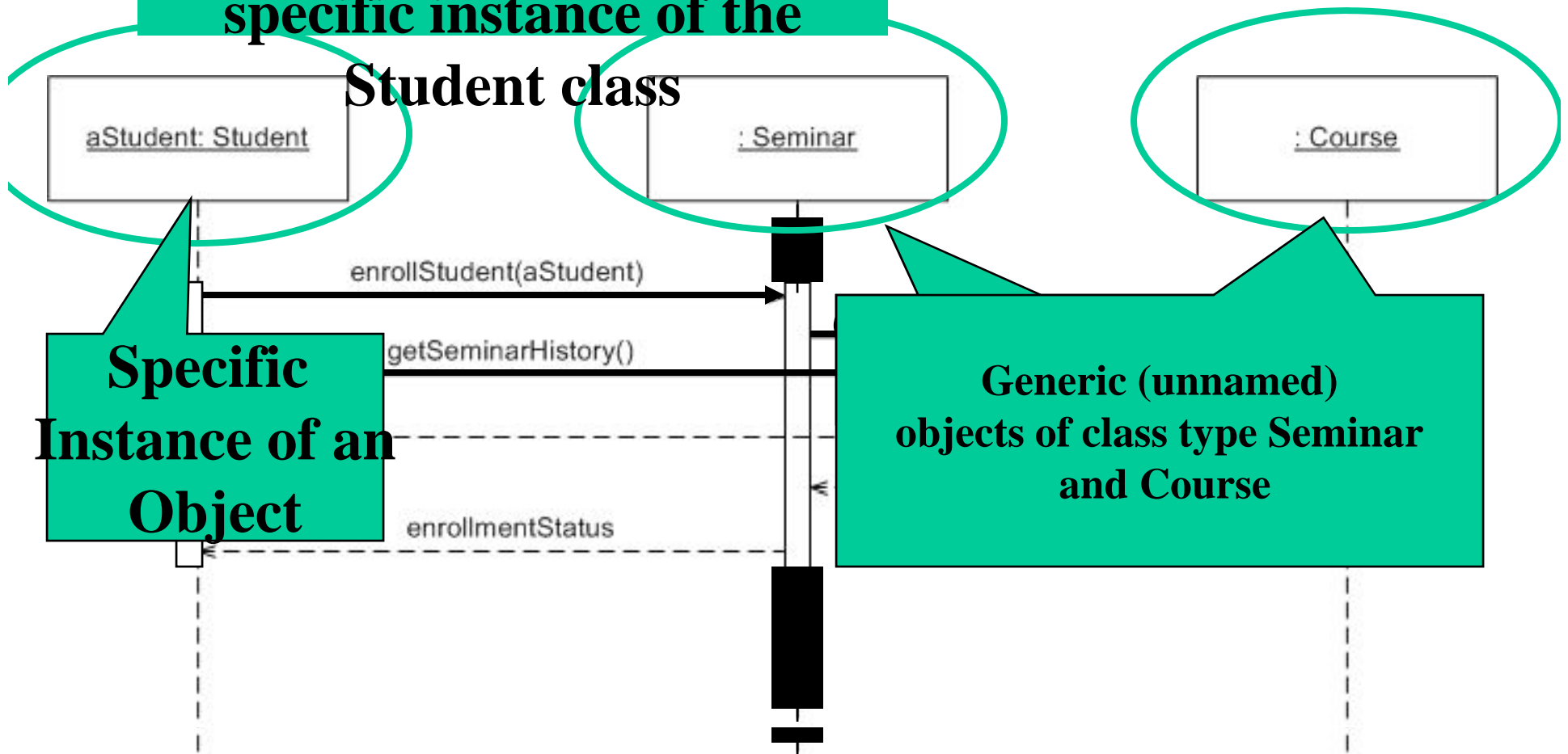
- What does this mean?

Do you typically care?

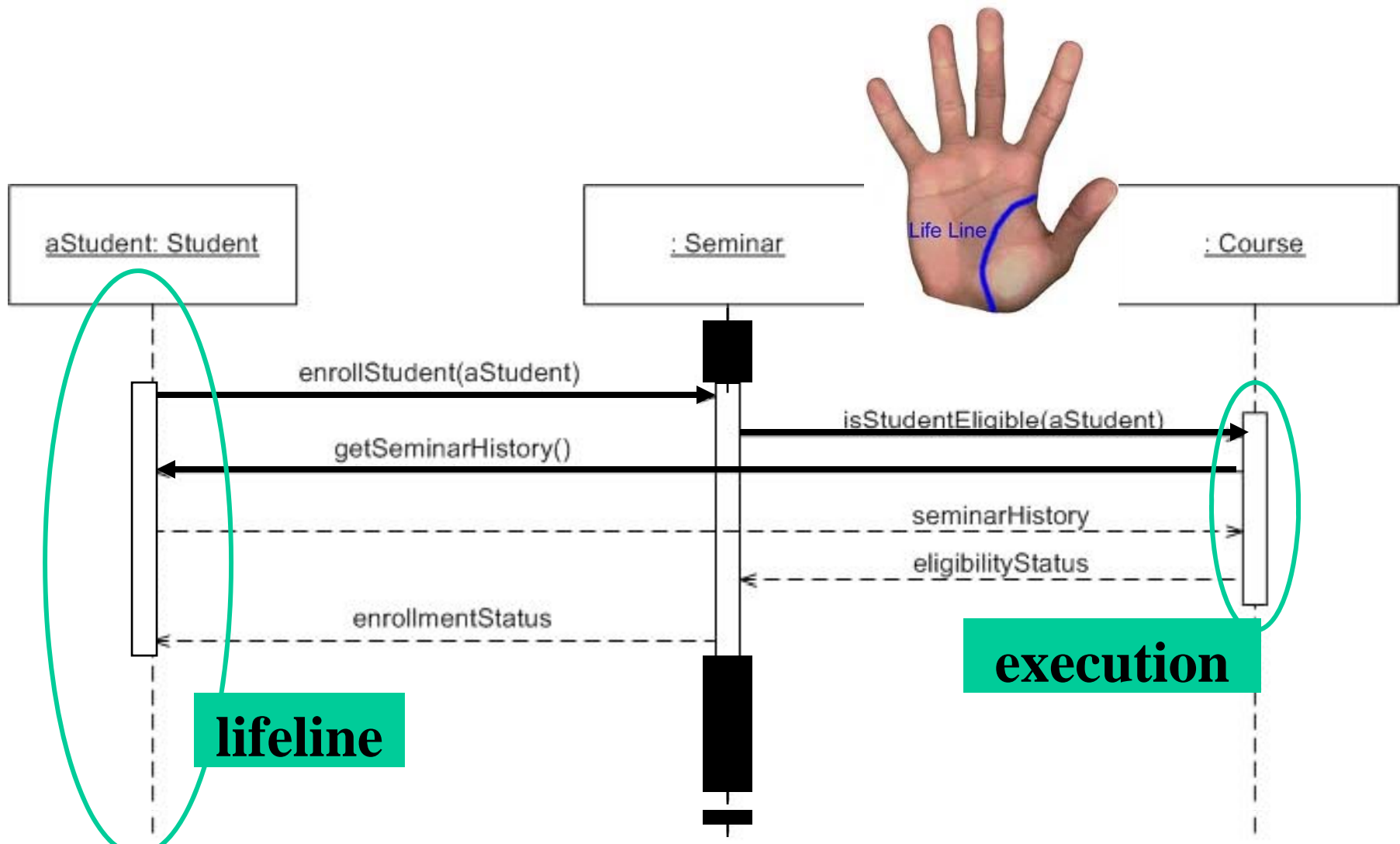


Components

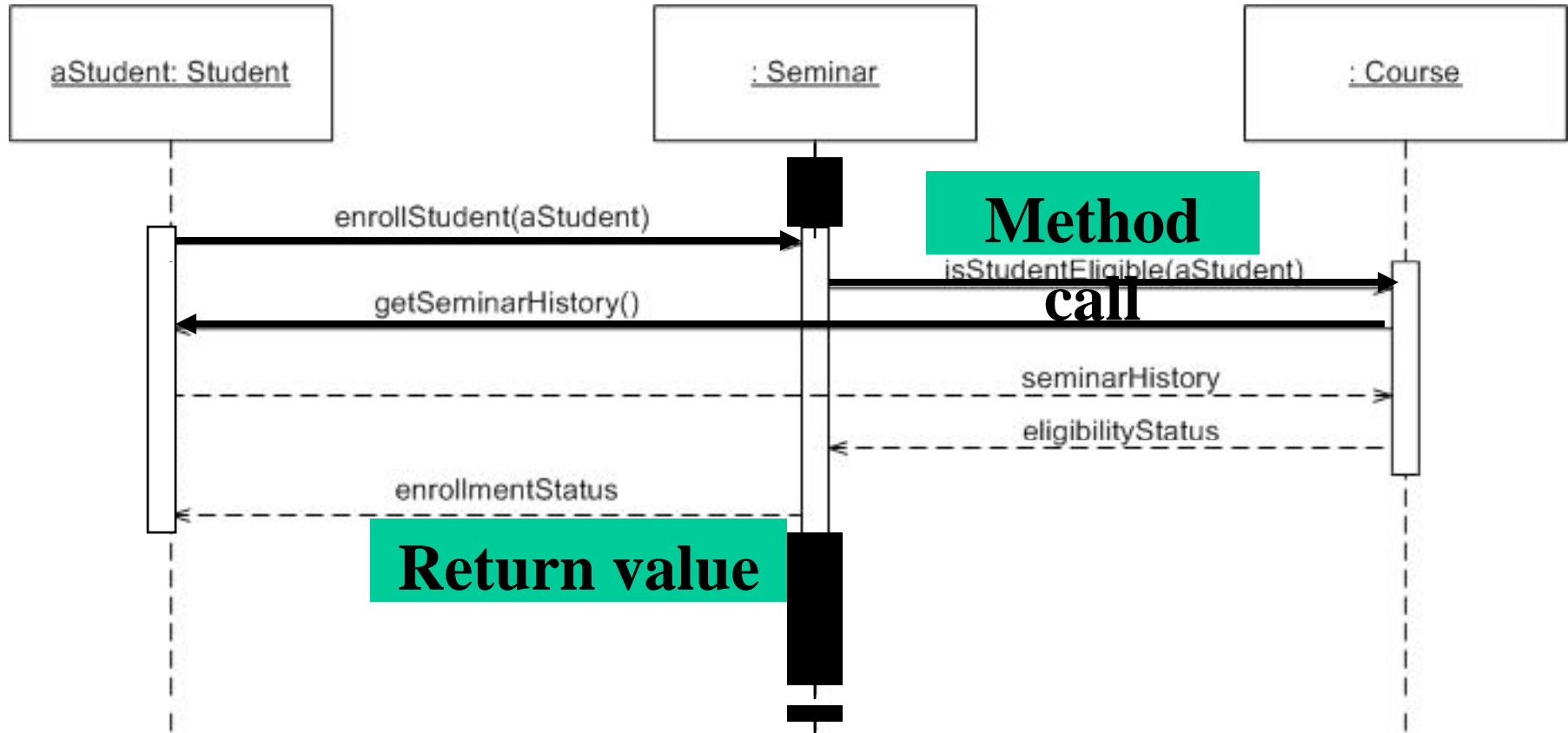
Objects: aStudent is a specific instance of the Student class



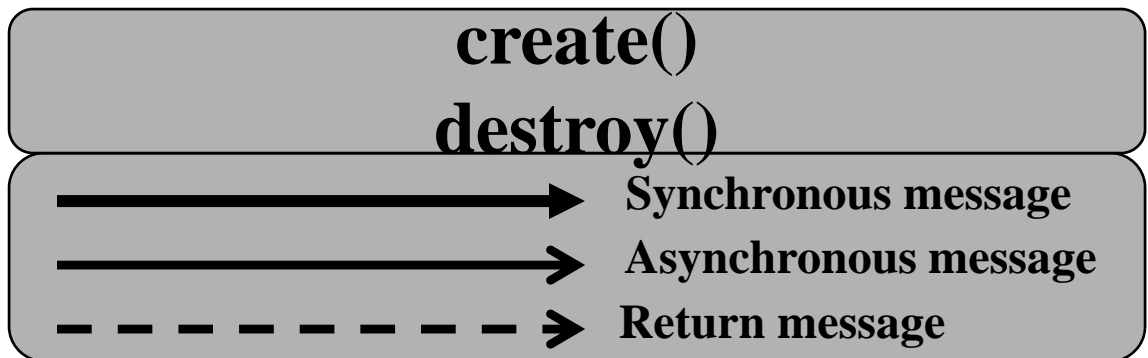
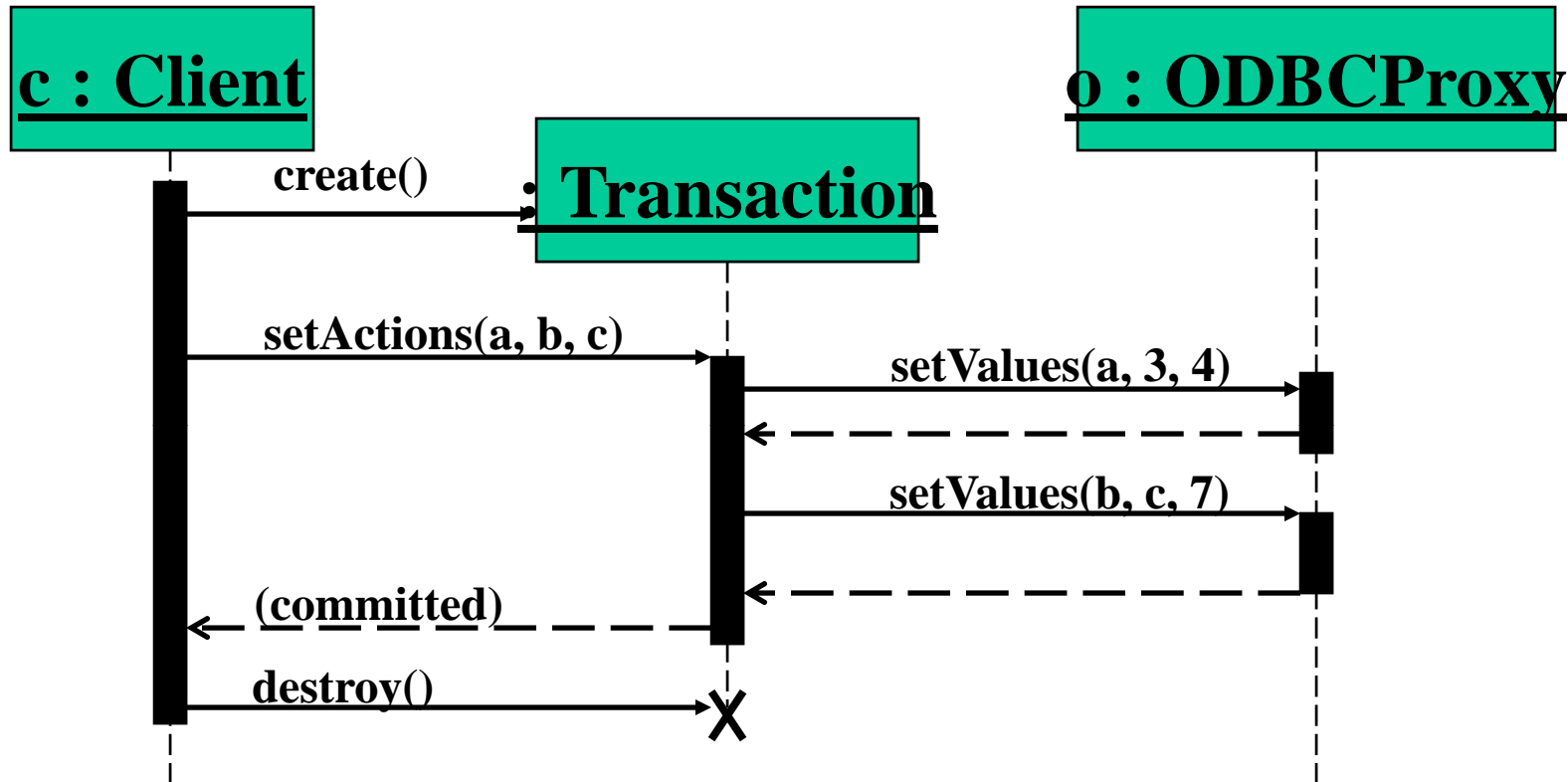
Components



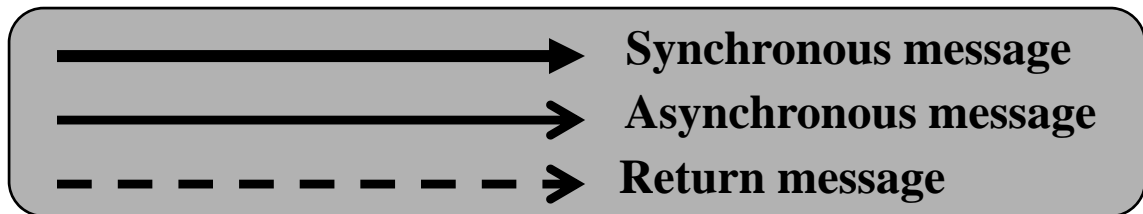
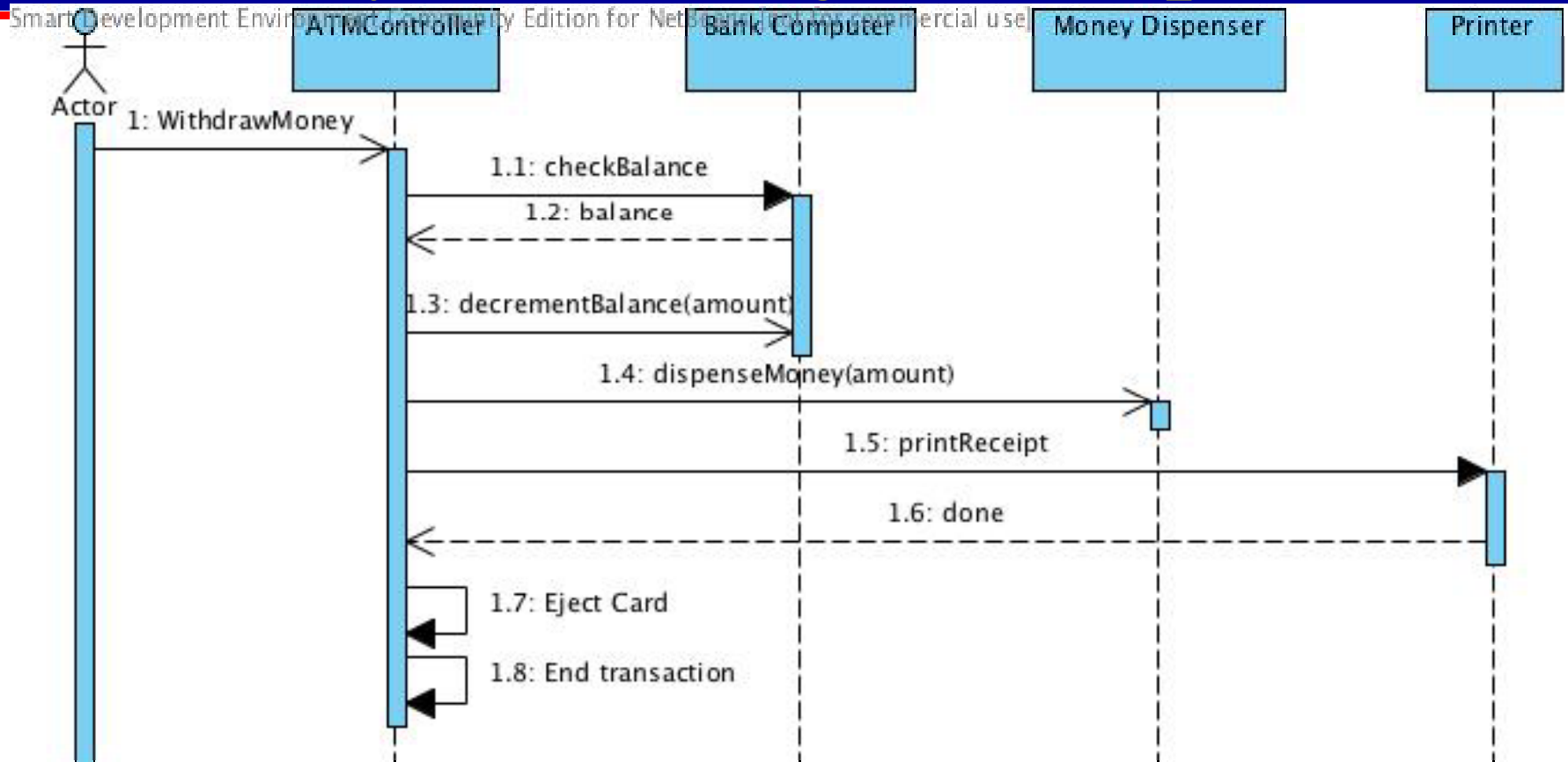
Components



Components

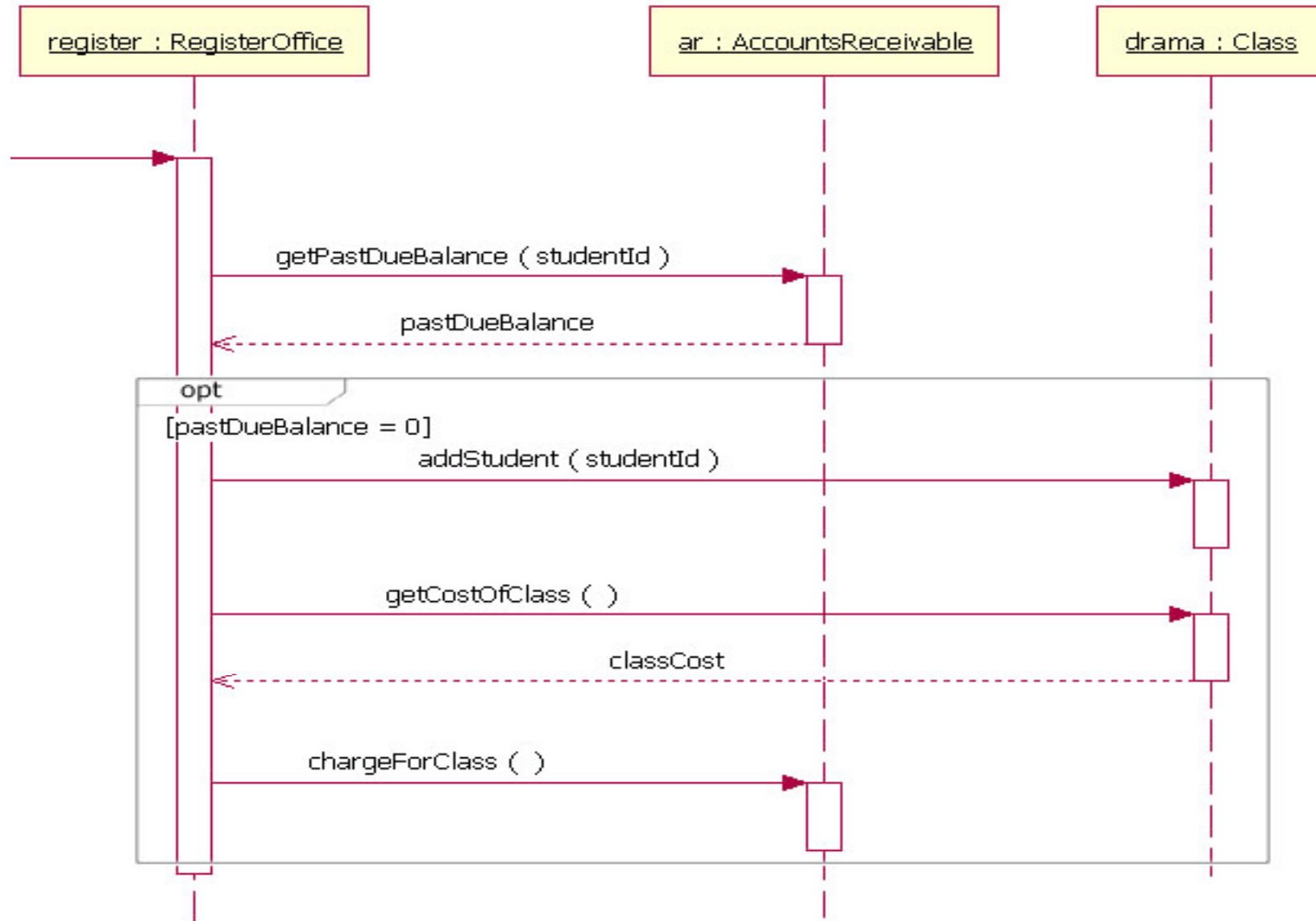


Async Message Example

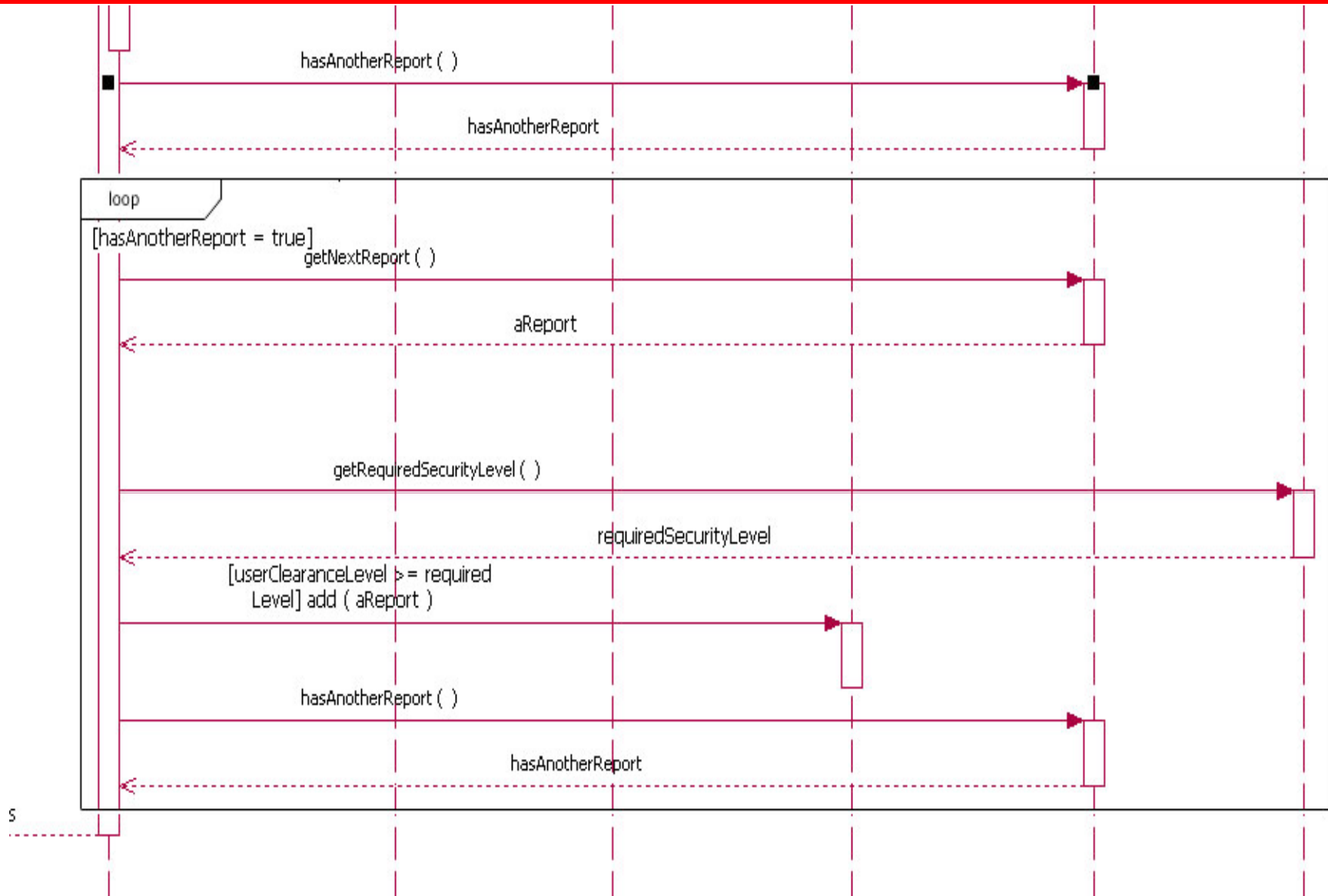




Components: option



Components: loop

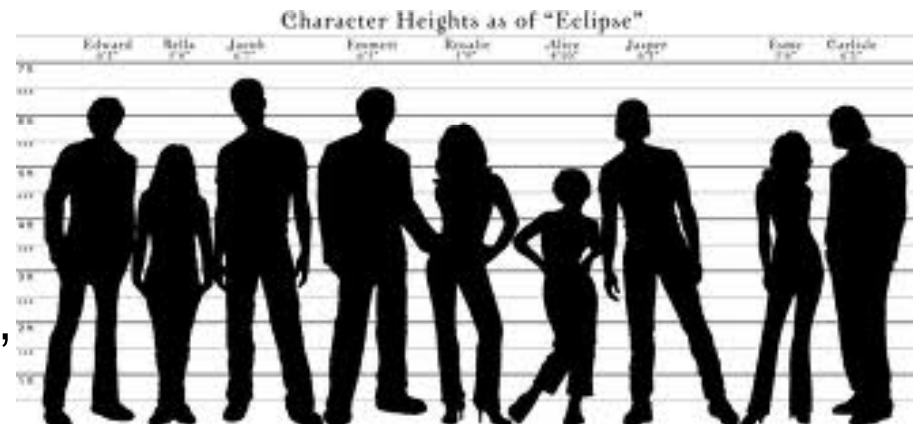


Rules of thumb

- Rarely use options, loops, alt/else
 - These constructs complicate a diagram and make them hard to read/interpret.
 - Frequently it is better to create multiple simple diagrams
- Create sequence diagrams for use cases when it helps clarify and visualize a complex flow
- Remember: the goal of UML is communication and understanding

Summary

- Sequence diagrams model object interactions with an emphasis on time ordering
- Method call lines
 - Must be horizontal!
 - Vertical height matters!
“Lower equals Later”
 - Label the lines
- Lifeline – dotted vertical line
- Execution bar – bar around lifeline when code is running
- Arrows
 - Synchronous call (you’re waiting for a return value) – triangle arrow-head
 - Asynchronous call (not waiting for a return) – open arrow-head
 - Return call – dashed line

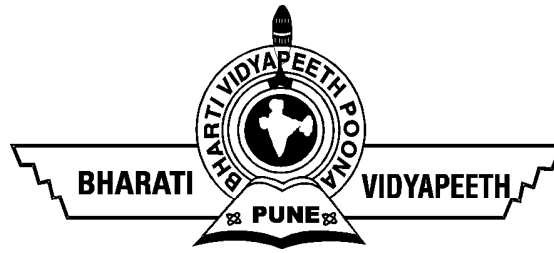


Internet shop PlaceOrder Usecase

- The customer opens an order form in his/her web browser and selects a product as well as the amount he/she wants to buy.
- The customer submits the form.
- The system will send an order confirmation via email to the customer.
- The system will check the magazine, to see if the requested amount is on stock.
 - If this is the case, the order is “shipped” immediately.
 - Otherwise
 - ✓ The system orders the missing goods via email from a major supplier.
 - ✓ If parts of the ordered goods are on stock, they are reserved for that order.
 - ✓ The system adds the order to the list of open orders.
- Draw a sequence diagram for this use case.

In class exercise

- Draw a sequence diagram for:
 - Adding a picture to Flickr (or any online image database). Login, pick an album, upload a picture, etc... Think about the software classes that would be involved – WebGUI (think of this as reporting what the user does), UserAccount, Album, AlbumList, etc...
 - Don't forget to check and update their current disk usage. For this diagram show the check coming back as acceptable.. you would do a second diagram for them running over quota.



Class & Object Diagram



Class Diagrams

UML modeling elements in class diagrams

- Classes and their structure and behavior
- Association, dependency, and inheritance relationships
- Multiplicity and navigation indicators
- Role names

Classes

- A class is a collection of objects with
 - common structure
 - common behavior
 - common relationships
 - common semantics
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
 - Naming standards should be created
 - e.g., all classes are singular nouns starting with a capital letter

Operations

- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams

RegistrationManager
addCourse(Student,Course)

Attributes

- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge

- Syntax

visibility name: type multiplicity = default {property-string}

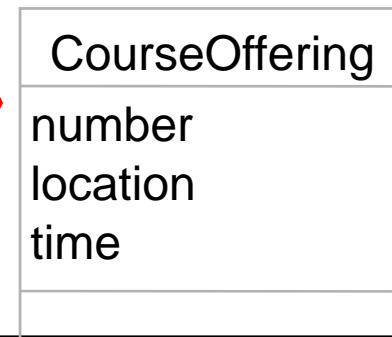
{+, -, ...}

{Integer, Boolean, String, Date, Money, ...}
{ [1], [n], [0..1], [m..n], [*], [1..*], [n..*] }

{0, "foo", 12-05-06, 100£, ...}

{ readOnly, ordered, nonunique, unique, bag, set, ...}

Each course offering has a number, location and time



Attributes Structural Features

- An attribute describes a property as a line of text within the class box

- Syntax

visibility name: type multiplicity – default {property-string}

{+, -,
#, `}

{Integer,
Boolean,
String,
Date,
Money,
...}

{ [1],
[n]
[0..1],
[m..n],
[*],
[1..*],
[n..*]}

{0,
"foo",
12-05-06,
100£,
...}

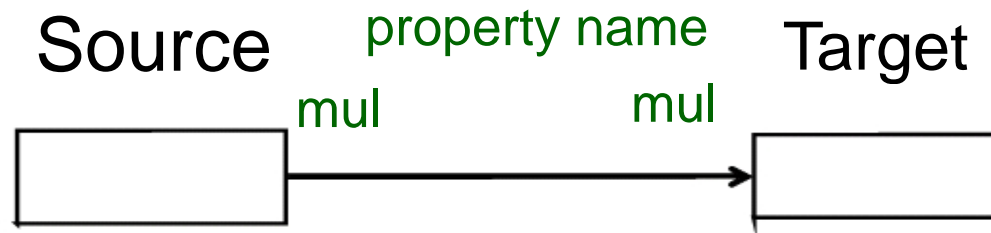
{ readOnly,
ordered,
nonunique,
unique,
bag,
set
...}

Relationships

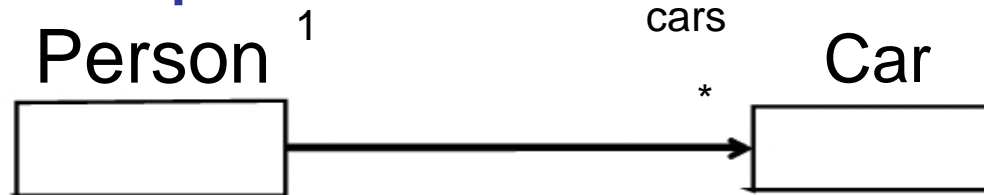
- Relationships provide a pathway for communication between objects
- if two objects need to “talk” there must be a link between them
- Three types of relationships are:
 - Generalization
 - Association
 - Dependency

Associations (structural features)

- An association is a bi-directional connection between classes
 - An association is shown as a line connecting the related classes



Example



Multiplicity Notation

1
— Class

exactly one

*
— Class

many (zero or more)

0..1
— Class

optional (zero or more)

m..n
— Class

numerically specified

{ordered} *
— Class

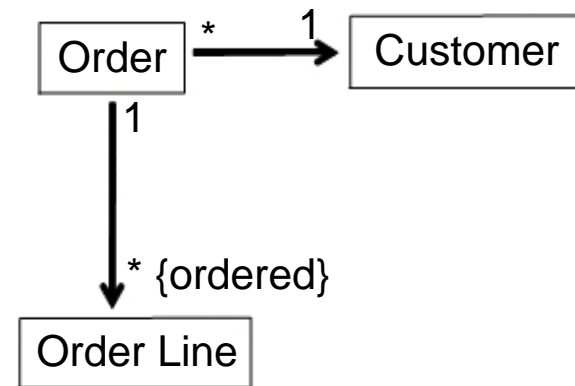
ordered

Order = class fields

dataReceived: Data
isPrepaid: Boolean
number: String
price: Money
customer: Customer
lineItems: List<Order Line>

methods

dispatch
close



Putting the UML to Work

The ESU University wants to computerize their registration system

- The Registrar sets up the curriculum for a semester
 - ✓ One course may have multiple course offerings
- Students select 4 primary courses and 2 alternate courses
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to receive their course offering rosters
- Users of the registration system are assigned passwords which are used at logon validation



Classes

RegistrationForm

Schedule Algorithm

RegistrationManager

Course

Student

Course Offering

Professor



Operations & Attributes

RegistrationForm

RegistrationManager

addStudent(Course, StudentInfo)

ScheduleAlgorithm

Course

name
numberCredits

open()
addStudent(StudentInfo)

Professor

name
tenureStatus

Student

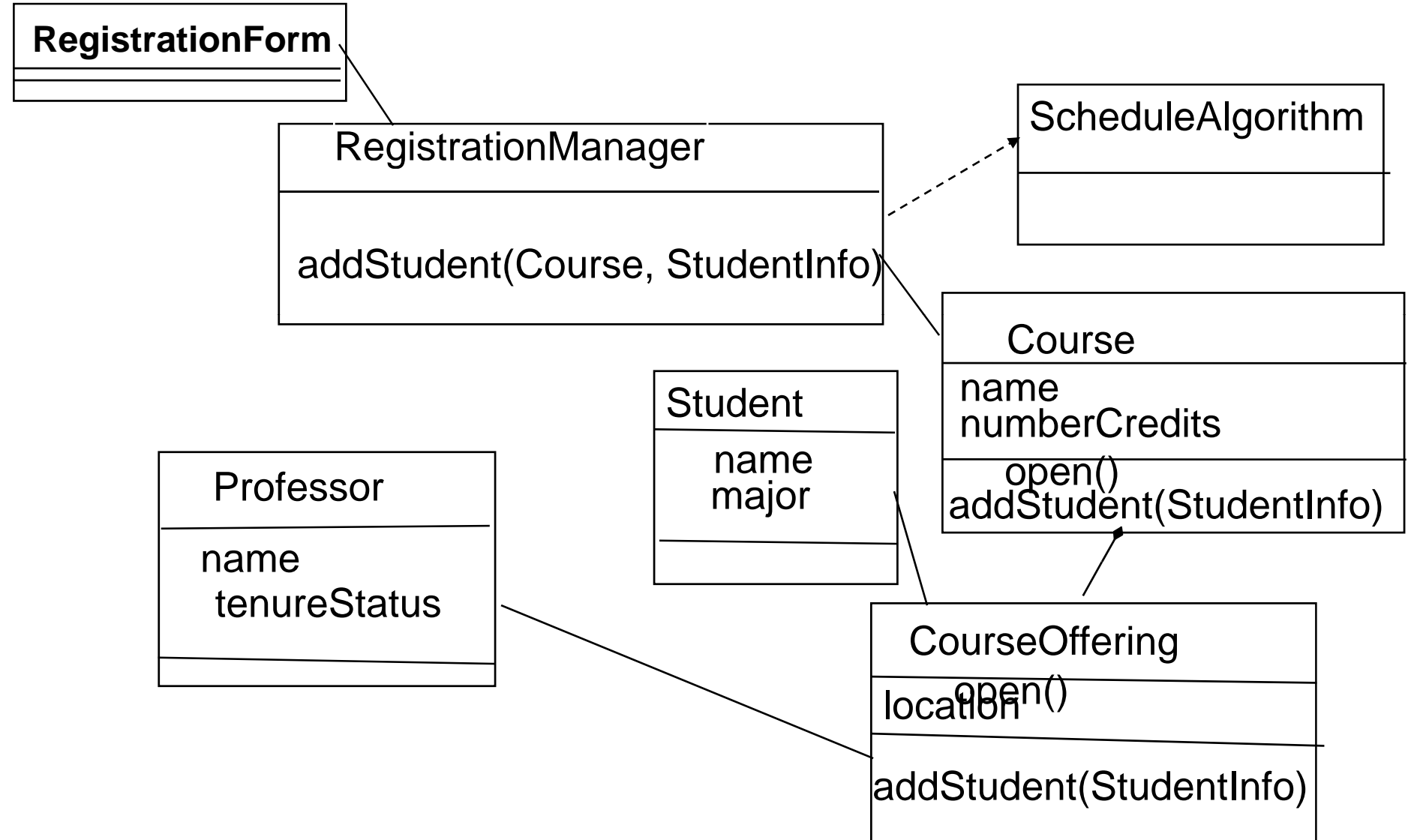
name
major

CourseOffering

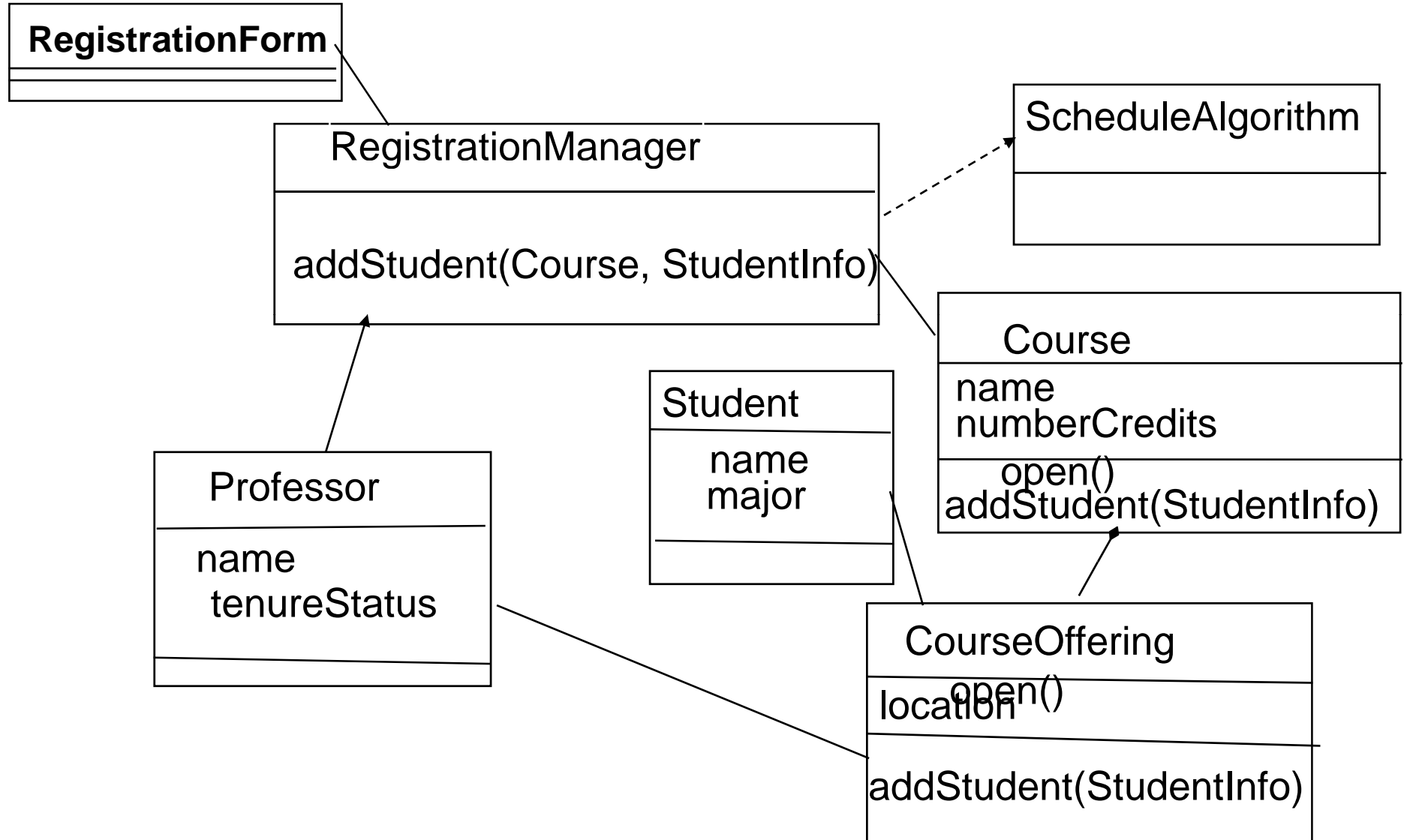
location
open()

addStudent(StudentInfo)

Relationship



Generalization



Object Diagrams

- An object diagram is a snapshot of the objects in the system
- Useful for showing complex structures
- Object names are underlined
- Syntax

instance name : class name

instance name

:class name

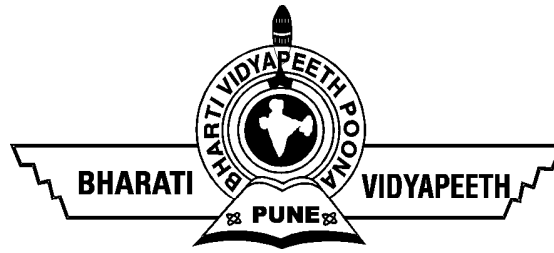


Case Study Dog Fun

- Each customer has one or more dogs
- I keep track of the customers' kind of dogs and email so I can send them greetings at Christmas and new offers [if they want]. I also keep track of how much they have spent
- The system helps me compute how much the customer should pay for the treatments of the dogs the customer brings
- A treatment starts with a wash that has a fixed cost of 10\$ no matter what kind of dog

Dog Fun ..

- **Then there are three kinds of ways to continue a treatment:**
 1. puddle-cut which can be NY-style to \$100 or London to \$50 (London style may include coloring for an extra \$10) [I decline to puddle cut a dog that's not a puddle]
 2. trim-cut for \$30, or
 3. style-cut for \$40
- **Some of my costumers get a discount of some percent, but not all**



State Diagram

Terms & Def.

- **State Diagram** shows *state Machine*
- **State Machine** is a behaviour that specifies the sequence of **states** an object goes through during its life time in response to **event**
- State is a condition or situation in the life of an object during which
 - It satisfies certain condition
 - Perform some activities
 - Or wait for some event

Terms & Def.

- **event** is the specification of a significant occurrence that has a location in time & space
- In context to state machine
 - An event is an occurrence of a stimulus that can trigger a state transition
- **Transition** is a relationship between two states
- An **activity** specifies non atomic on going execution

Terms & Def.

- Types of events
 - External
 - ✓ Signals
 - ✓ Call Events
 - Internal
 - ✓ Change Event
 - ✓ Time Event

State Diagrams

- A **state diagram** is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies the state sequences caused by event sequences.
- State names must be unique within the scope of a state diagram.
- All objects in a class execute the state diagram for that class, which models their common behavior.

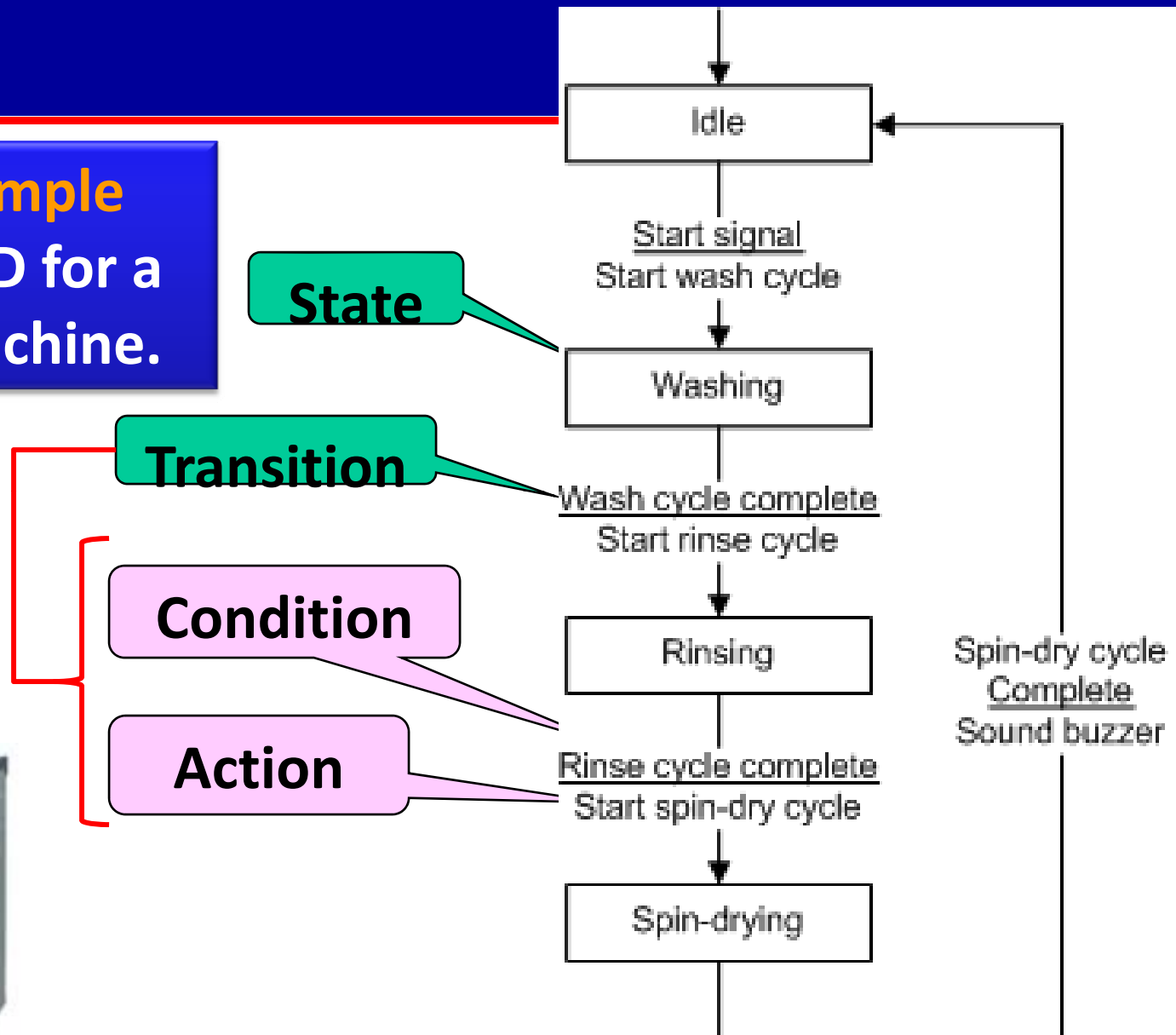
Purpose

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object.

Where to use

- To model object states of a system.
- To model reactive system. Reactive system consists of reactive objects.
- To identify events responsible for state changes.
- Forward and reverse engineering.

Here's a **simple** example STD for a washing machine.



A **condition** is typically some kind of **event**, e.g.:

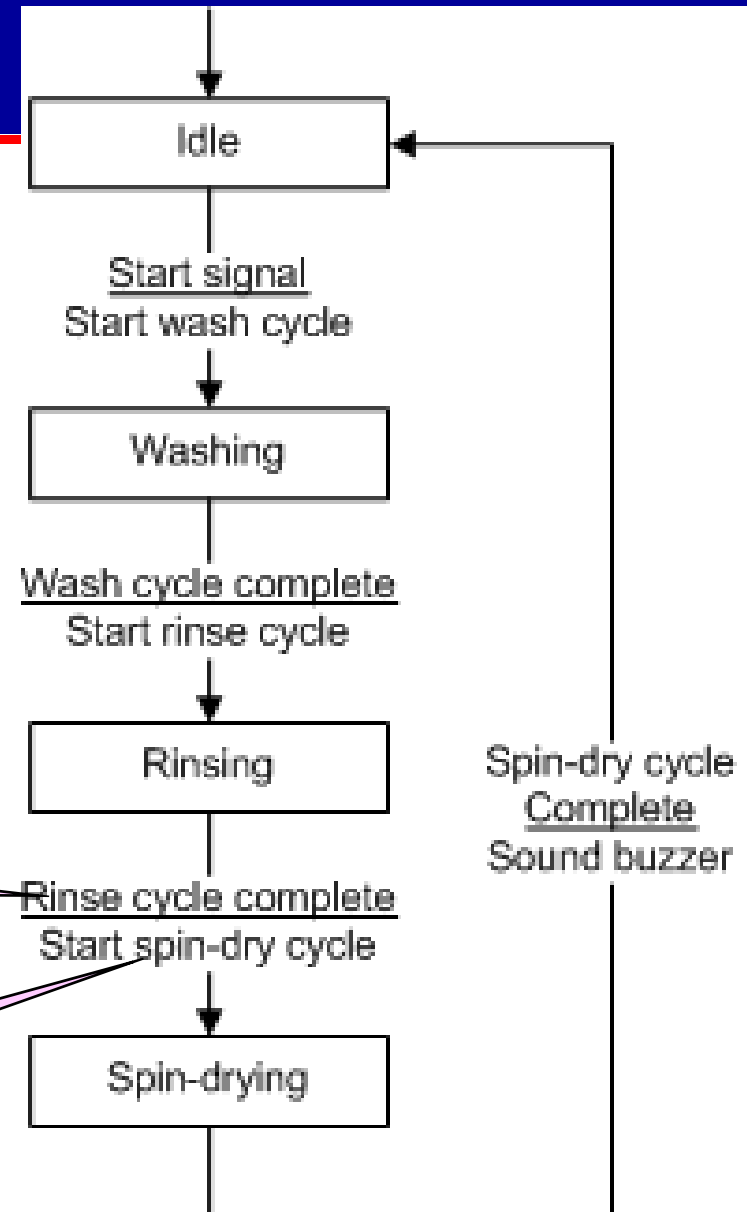
- Signal
- Arrival of an object (data/material),
- Etc...

An **action** is the appropriate **output** or **response** to the event, e.g.:

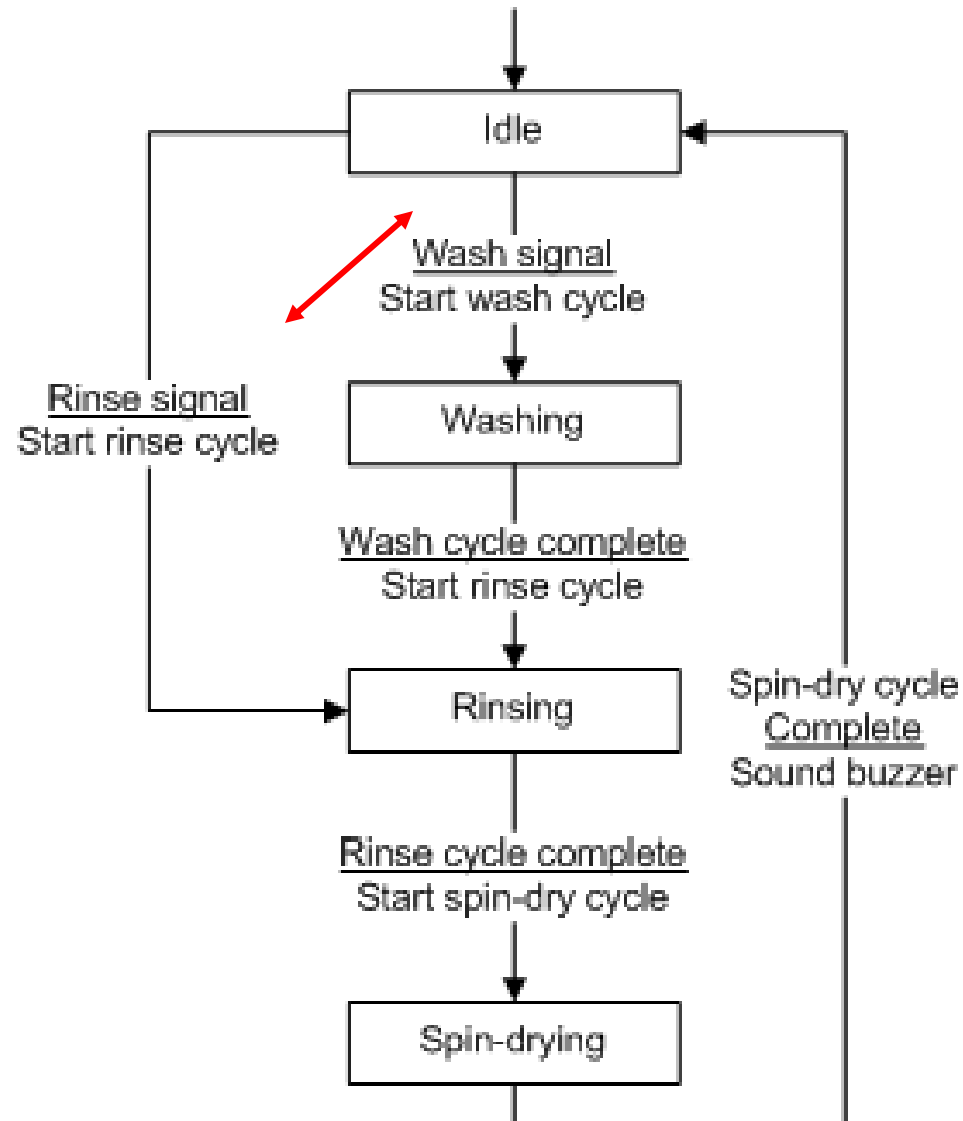
- Signal or message
- Transfer of an object,
- Calculation,
- Etc...

Condition

Action



Here's an even more interesting example.



It is now perfectly clear that you:

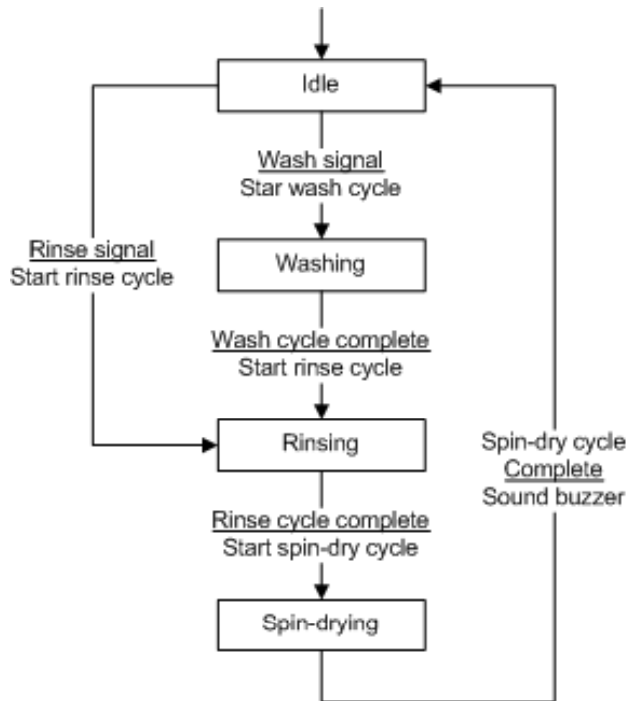
- Can go right to rinsing without washing, but..
- Can never go to spin-drying without rinsing.

So, what would I use such a modeling technique for?

State Transition Matrix

		From			
		State 1	State 2	State 3	etc...
To	State 1				
	State 2	<u>Condition</u> Action			
	State 3		<u>Condition</u> Action		
	etc...				

State Transitions Matrix

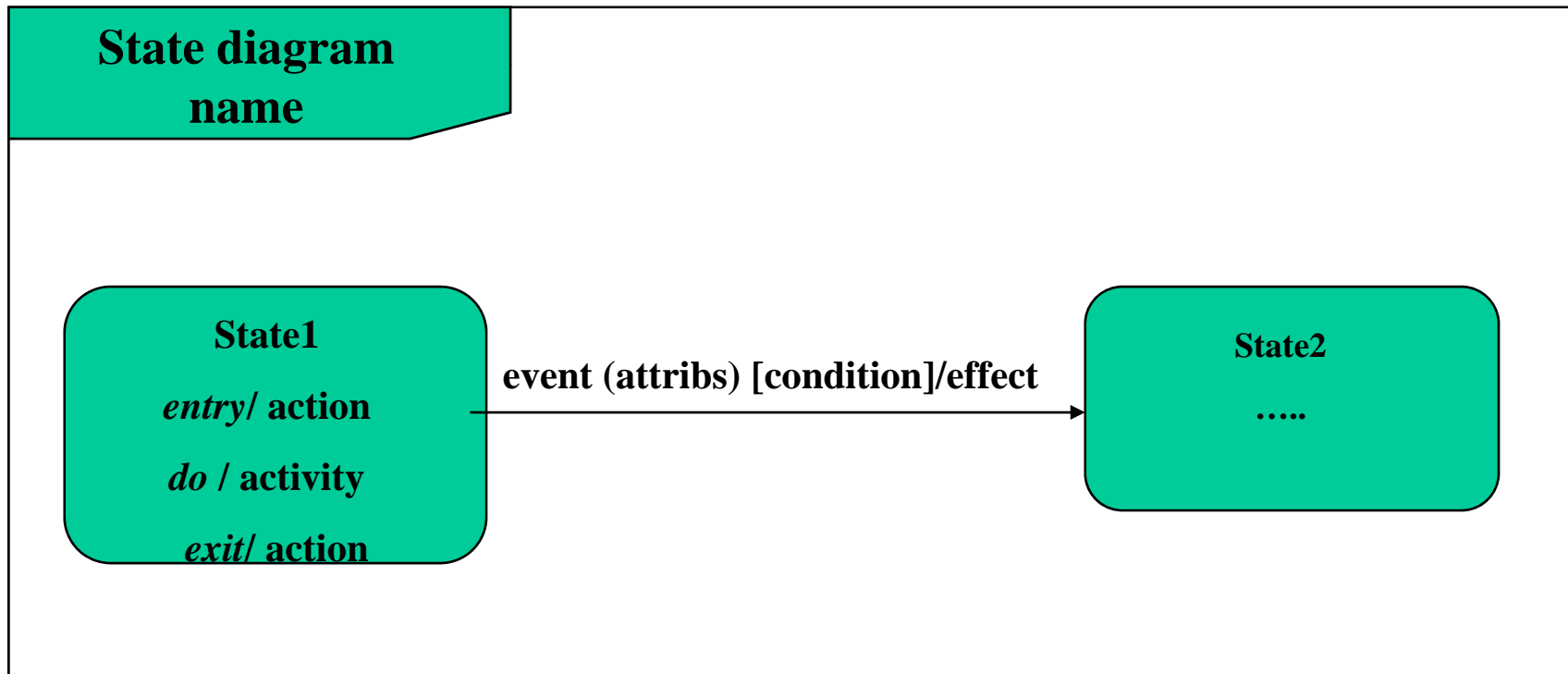


Do this first to define desired operation.

Check blank cells – have we covered all possible desired transitions?

		From			
		Idle	Washing	Rinsing	Spin-Drying
To	Idle				Spin-dry cycle complete Sound buzzer
	Washing	Wash signal			
		Start wash cycle			
	Rinsing	Rinse signal	Wash cycle complete		
		Start rinse cycle	Start rinse cycle		
	Spin-Drying			Rinse cycle complete	
				Start Spin-dry cycle	

Basic State Diagram Notation



Do-Activities

- an activity that continues for an extended time.
- can only occur within a state and cannot be attached to a transition.
- include
 - continuous operations,
 - ✓ such as displaying a picture on a television screen
 - Sequential operations
 - ✓ that terminate by themselves after an interval of time
 - ✓ such as closing a valve.

Do-Activities

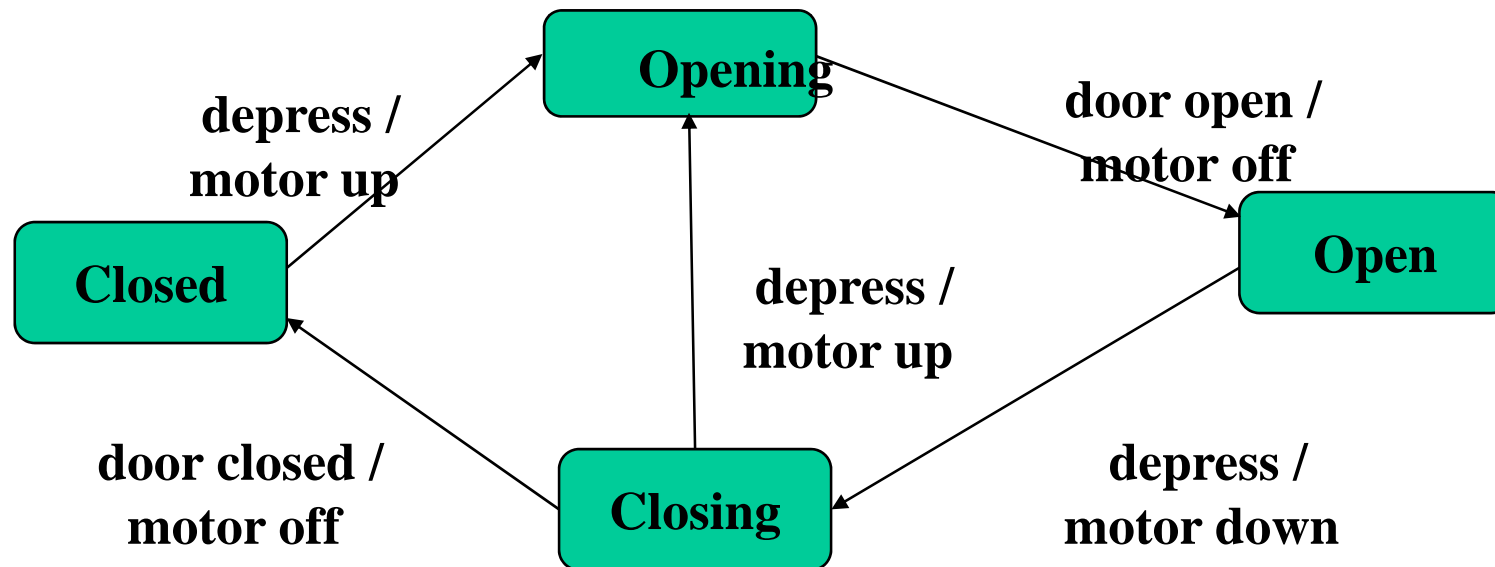
- may be performed for all or part of the duration that an object is in a state.
- may be interrupted by an event that is received during its execution
- such an event may or may not cause a transition out of the state containing the do-activity.

Paper jam

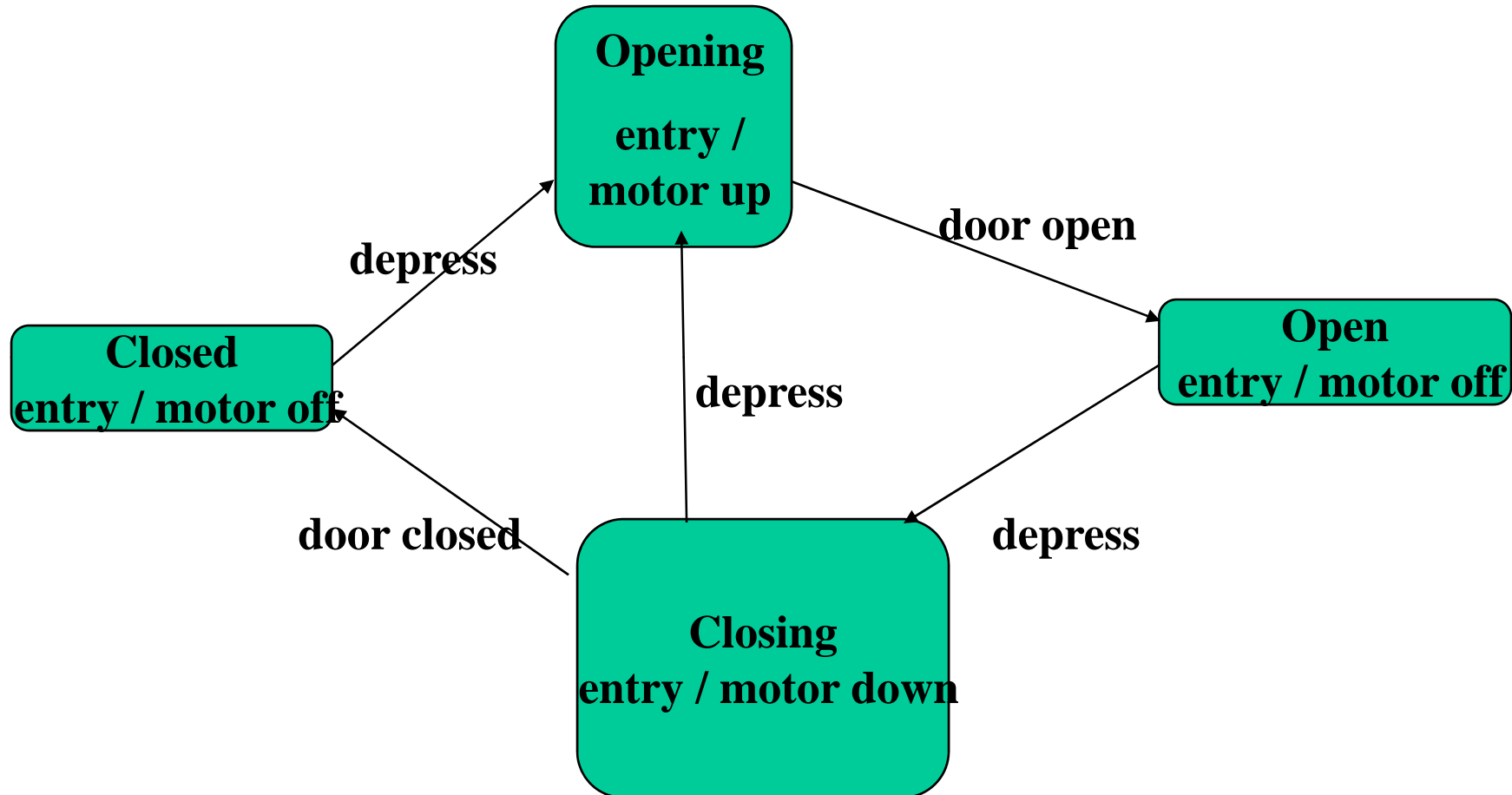
do/flash warning light

Entry and Exit Activities

- As an alternative to showing activities on transitions,
- The next figure shows the control of a garage door opener.



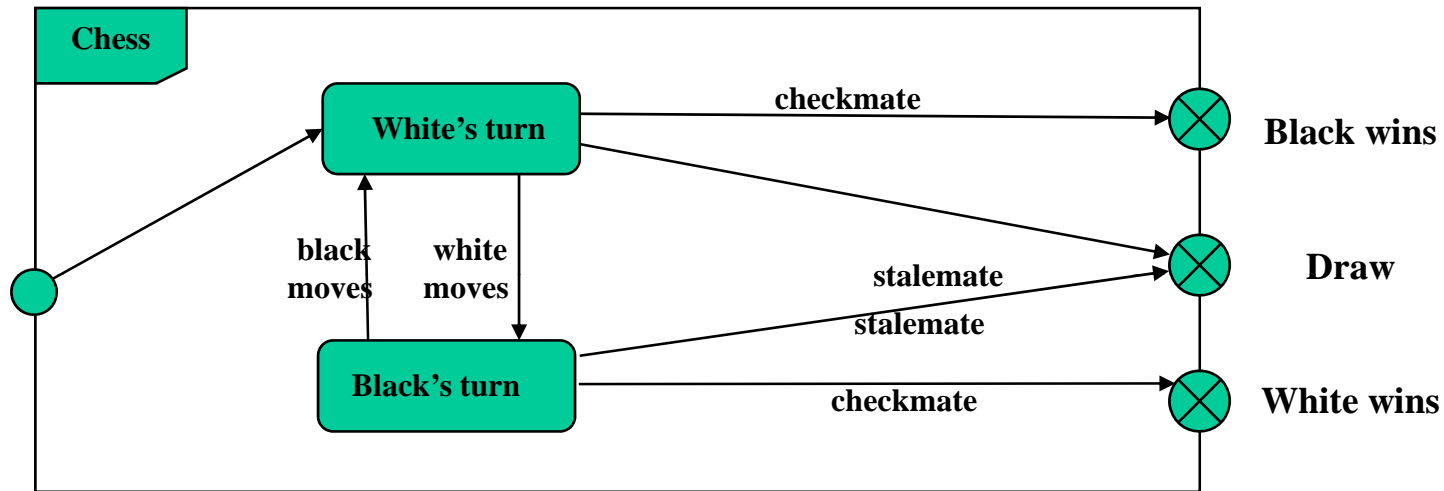
Activities on entry to states



Exit Activity

- less common than entry activities, but they are occasionally useful.
- Whenever the state is exited, by any outgoing transition, the exit activity is performed first.
- If a state has multiple activities, they are performed in the following order: activities on the incoming transition, entry activities, do-activities, exit activities, activities on the outgoing transition.
- Events that cause transitions out of the state can interrupt do-activities.
- If a do-activity is interrupted, the exit activity is still performed.
- In general, any event can occur within a state and cause an activity to be performed. *Entry* and *exit* are only two examples of events that can occur.

One shot State Diagram



State Diagram for a Phone Line

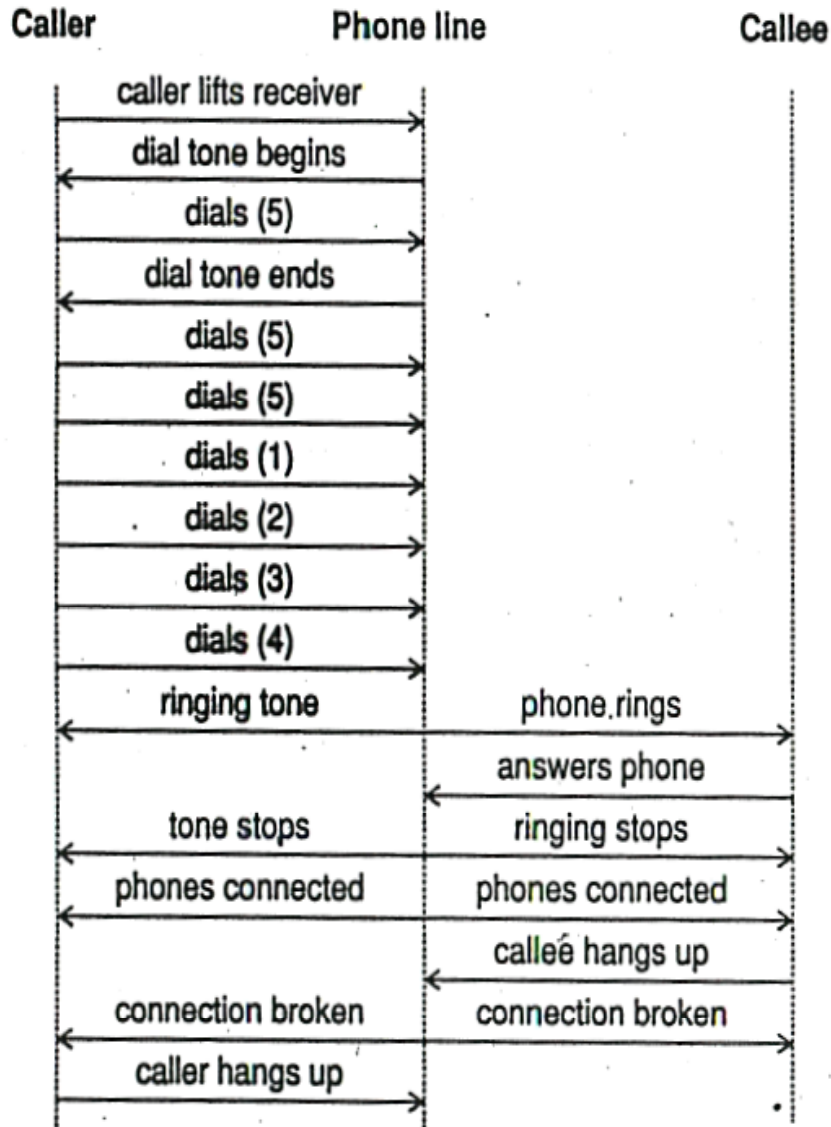


Figure 5.3 Event trace for phone call

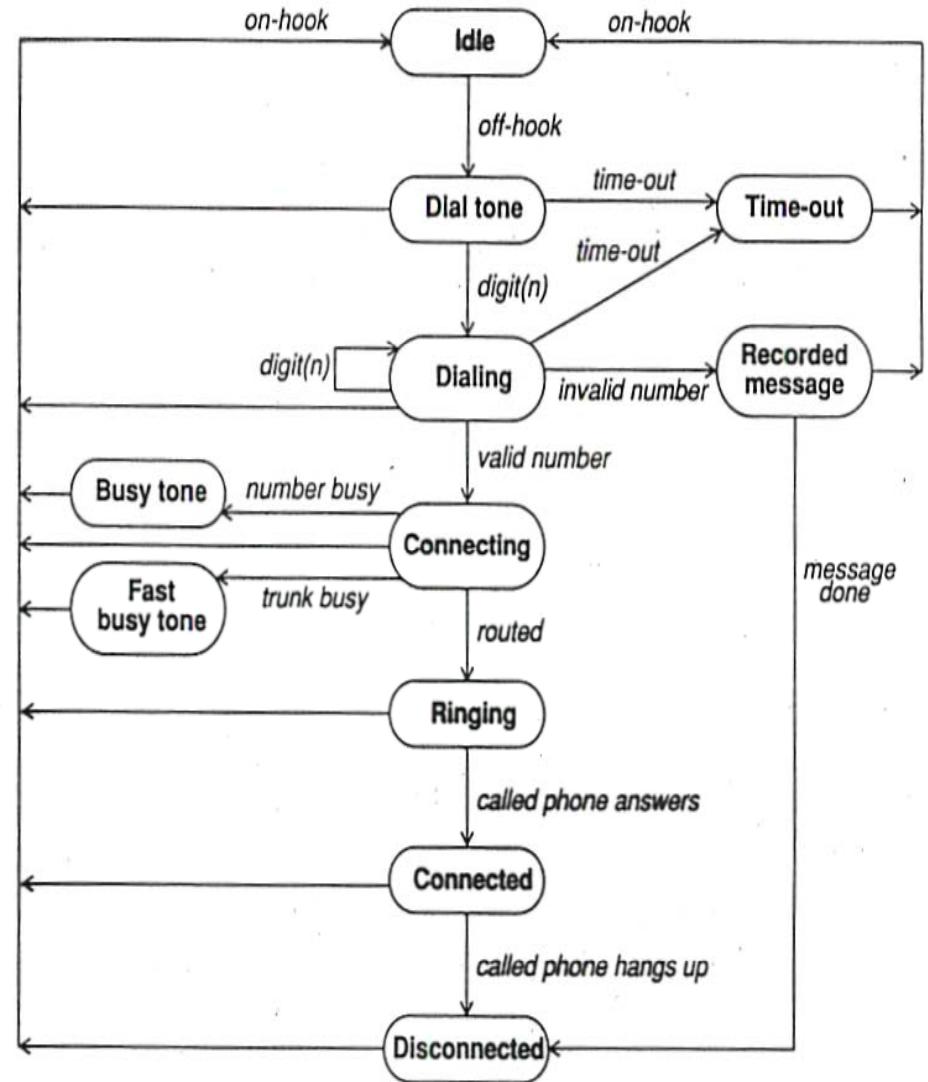
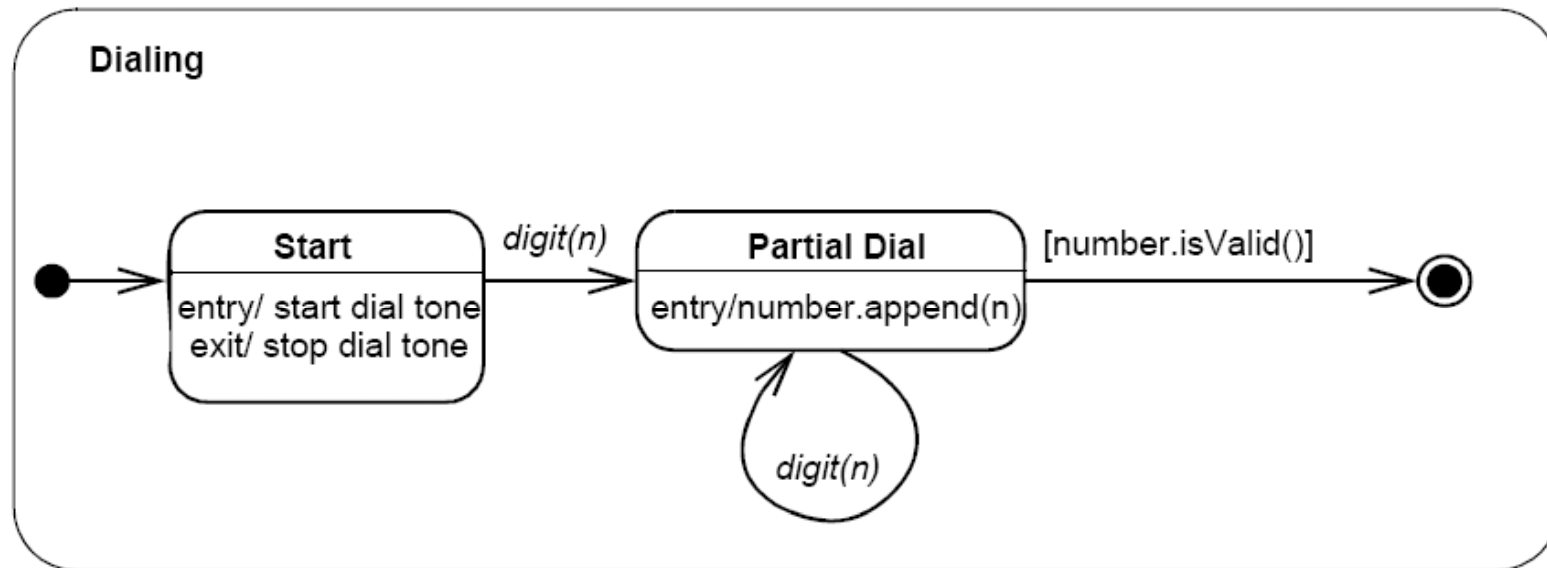


Figure 5.5 State diagram for phone line

Composite State

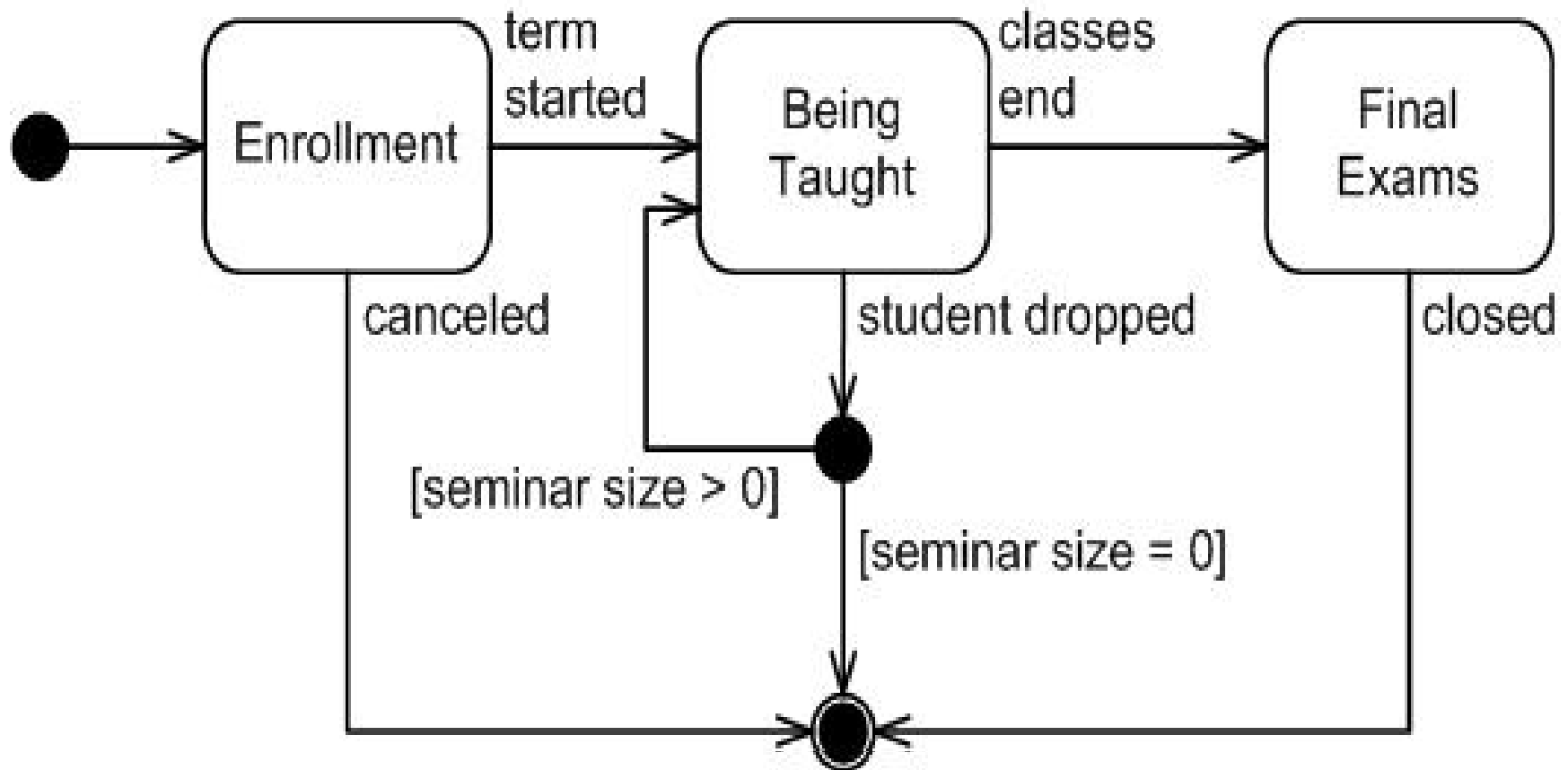
Concept of substate



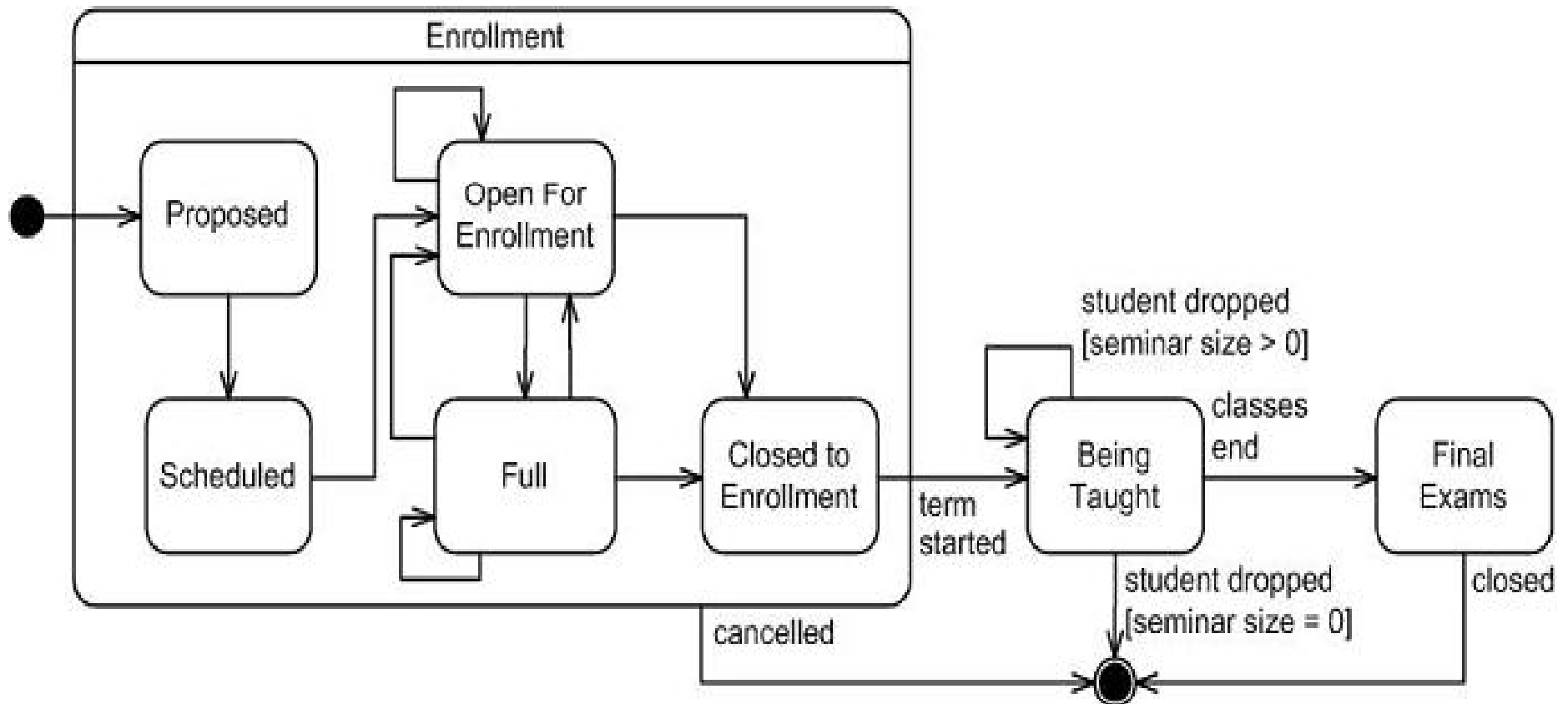
Substates

- **Substates**
 - ✓ **Orthogonal (concurrently active)**
 - ✓ **Nonorthogonal (Sequential active)**

State Diagram in UML..

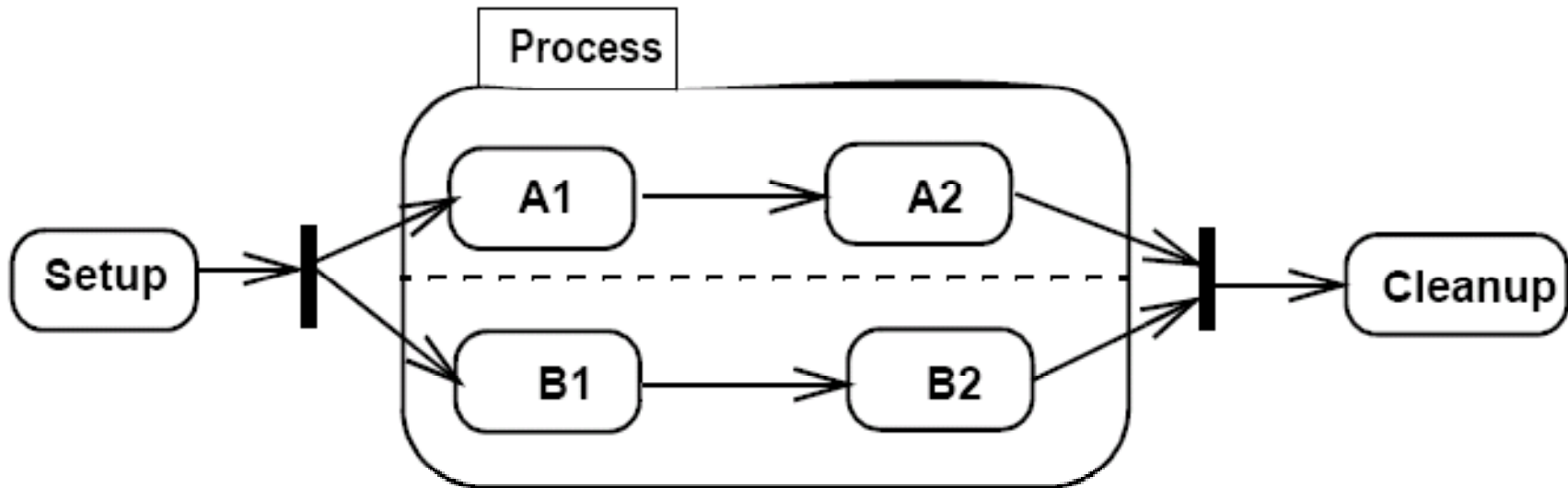


State Diagram in UML..

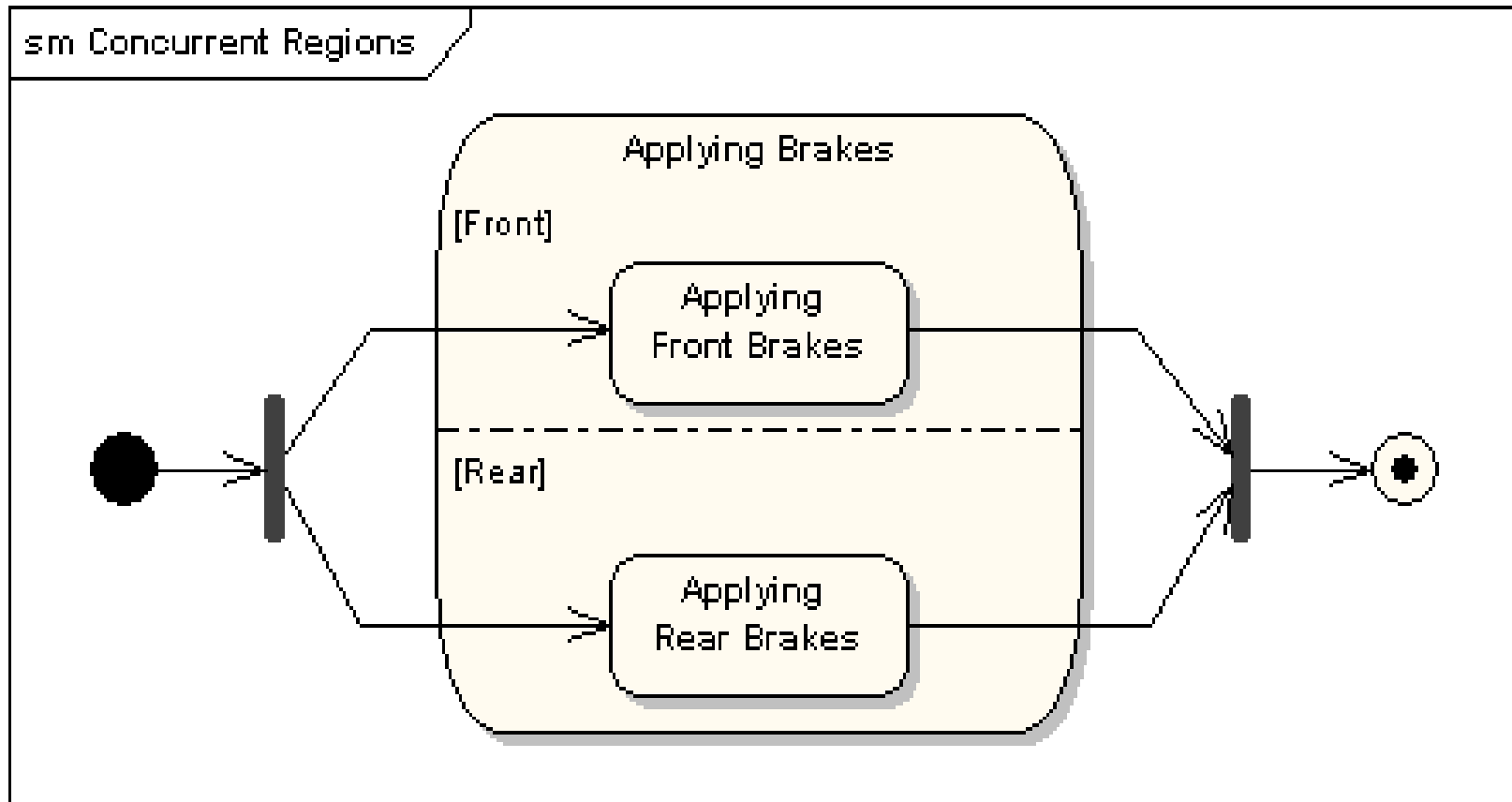


Concurrent States

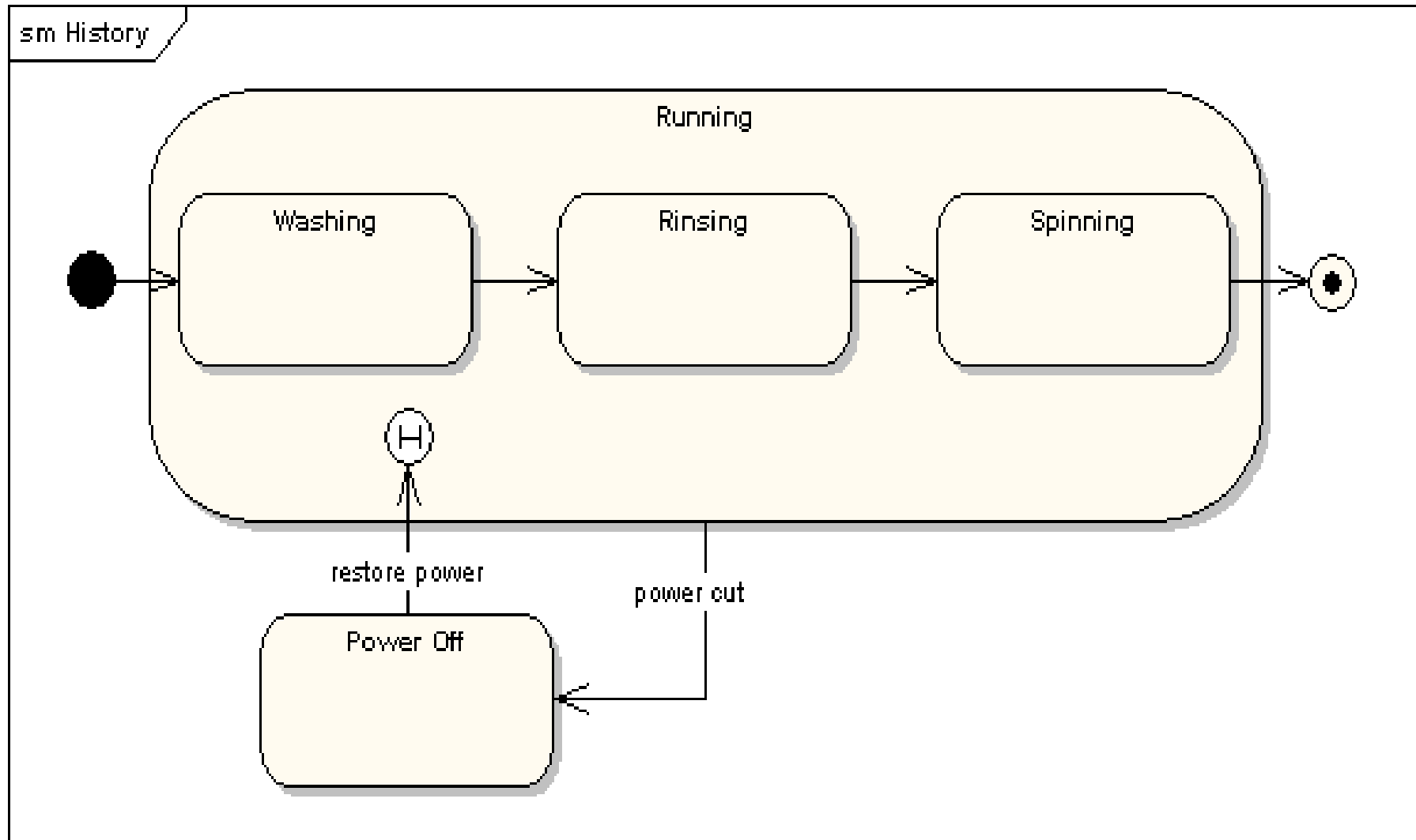
- Transitions to and from Concurrent States
multiple source states and target states



Concurrent States

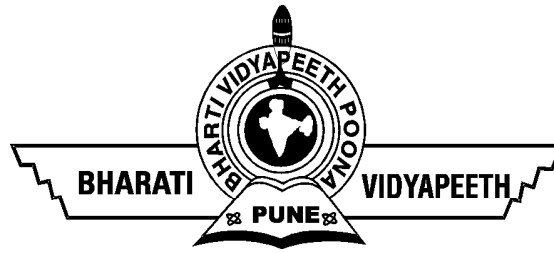


History States



Problem # 1

- A simple digital watch has a display and two buttons to set it, the A button and the B button. The watch has two modes of operation, display time and set time.
- In the display time mode, the watch displays hours and minutes, separated by a flashing colon.
- The set time mode has two submodes, set hours and set minutes. The A button selects modes. Each time it is pressed, the mode advances in the sequence: display, set hours, set minutes, display, etc.
- Within the submodes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event.
- Prepare a state diagram of the watch.

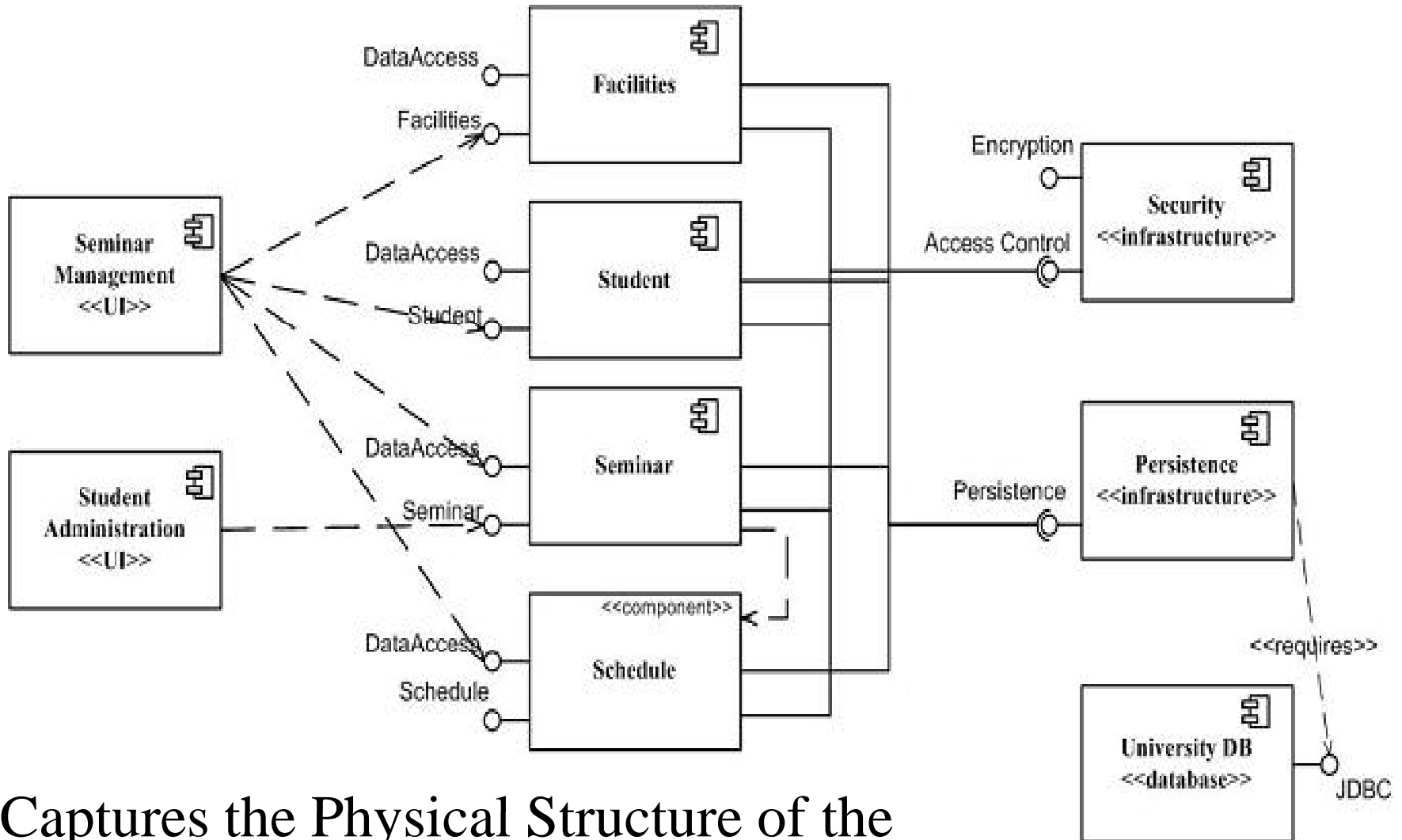


Component & Deployment Diagram

Component Diagram

- Component Diagram:
 - High-Level Interaction
 - Dependencies Among Software Components
- Captures the Physical Structure of the Implementation
- Built As Part of Architectural Specification
- Purposes:
 - Organize Source Code
 - Construct an Executable Release
 - Specify a Physical Database
- Main Concepts:
 - Component, Interface, Dependency, Realization
- Developed by Architects and Programmers

Component Diagram



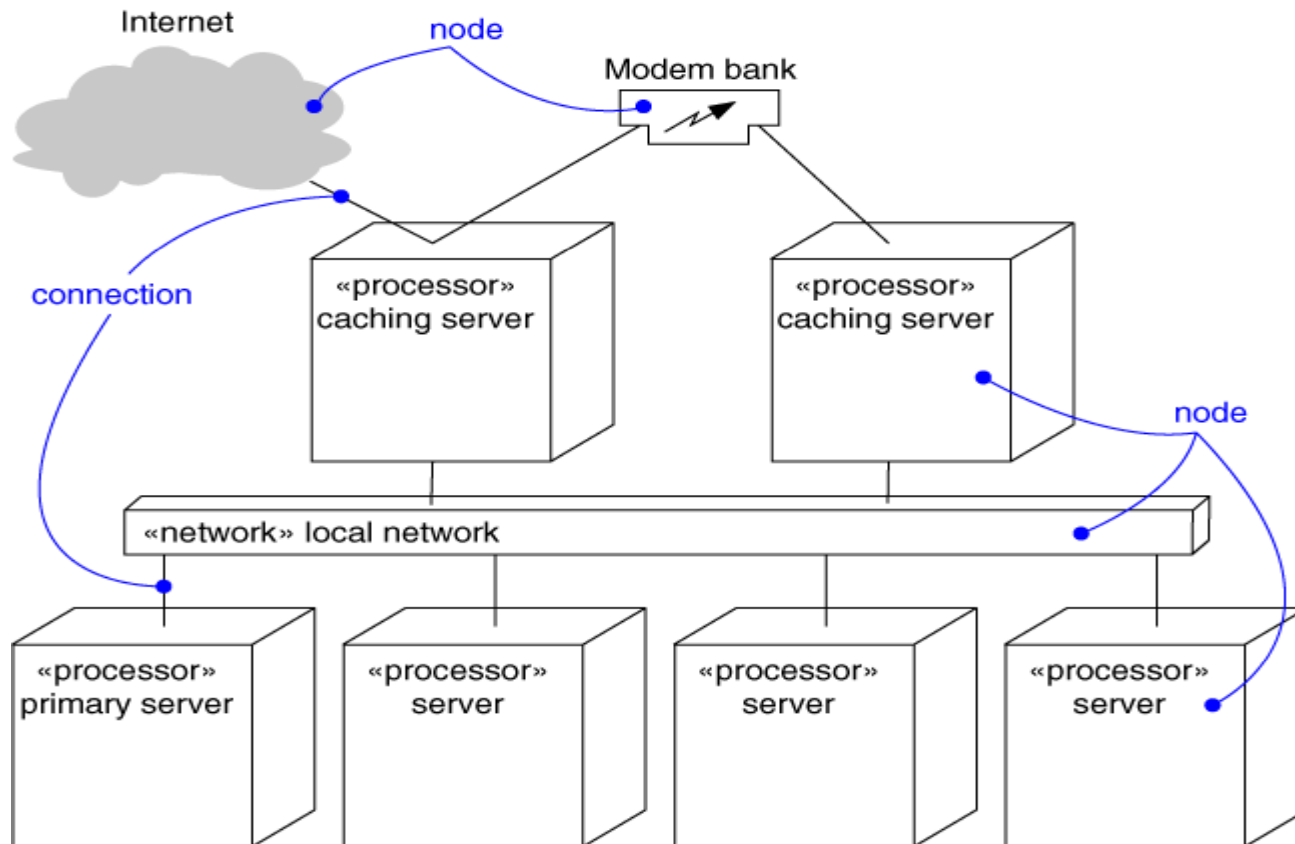
Captures the Physical Structure of the Implementation

Deployment Diagram

- Deployment Diagram:
 - **Focus on the Placement and Configuration of Components at Runtime**
- Captures the Topology of a System's Hardware
- Built As Part of Architectural Specification
- Purposes:
 - Specify the Distribution of Components
 - Identify Performance Bottlenecks
- Main Concepts:
 - **Node, Component, Dependency, Location**
- Developed by
 - **Architects, Networking Engineers, and System Engineers**

Deployment Diagram

Captures the Topology of a System's Hardware

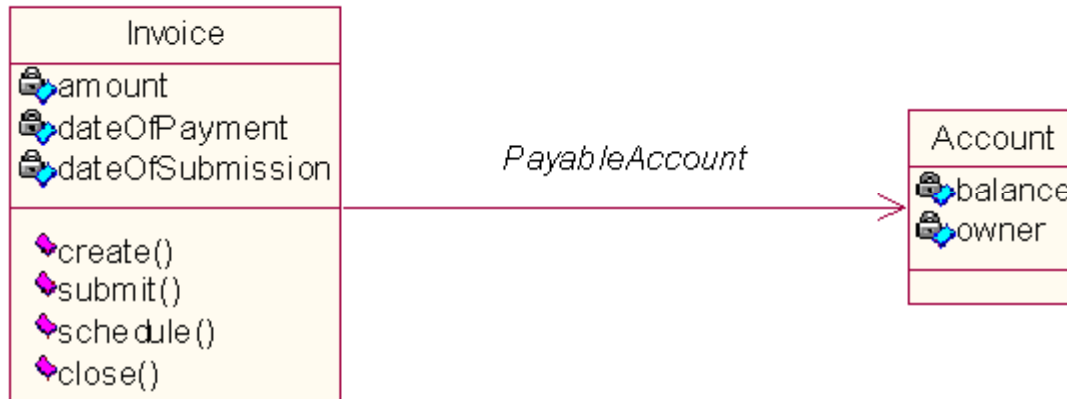




UNIT IV Learning's

- ✓ Modeling with UML
- ✓ Basic Building Blocks of UML
- ✓ A Conceptual Model of UML
- ✓ Basic
- ✓ Structural Modeling
- ✓ UML Diagrams
- ✓ Case Studies

Objective Questions



Q1. The arrow in the above diagram indicate

1. Refers to
2. Association
3. Dependency
4. Navigability

Q2. At Conceptual level, a Class can consist attributes of:

- a. Primitive data type
- b. Value object data type e.g. Date, Money
- c. Complex data type e.g. Customer, Account, Inventory
- d. All of the above
- e. A, b ONLY
- f. A, c ONLY

Objective Questions..

Q3. Class diagrams at conceptual level should include:

1. attributes ONLY
2. operations ONLY
3. both attributes and operations

Q4. What are the strengths and weakness of Interaction Diagrams?

1. when you want to look at the behavior of several objects within a single use case
2. they are good at precise definition of the behavior
3. they are good at showing collaborations among objects
4. they are good at exploring concurrency and multi-thread issues

Q5. Package diagrams are designed for:

1. organizing a large project into components
2. depicting the overall structure of a system
3. assisting testing
4. assisting deployment
5. reducing dependency

Objective Questions..

Q6. Which statements are true about Class Diagrams?

1. From the conceptual perspective, associations represent conceptual relationships between classes
2. Naming role is optional. If missing, it is named after the source class
3. Multiplicities are normally 0, 1, *. It does not support for a range number e.g. 2-4
4. Within the specification perspective, associations represent methods.
5. From the conceptual perspective, associations have no arrow heads meaning that they are non-directional
6. From specification perspective, association lines with arrows indicate navigability. The source class has responsibility of 'knowing' the target class but not the other way round.

Q7. Which statements are true about associations in Class Diagrams?

1. It is good practice to name every association and most data modelers prefer to name association using a "verb"
2. Most object modelers prefer to assign role name using a "noun" to association
3. If there is no name on the role, the implied name is the name of target class

Objective Questions..

Q8. What is the difference between an attribute and an association?

1. From the conceptual perspective, there is a distinct difference.
2. From specification and implementation perspective, an attribute is owned by a type while an association allows one to navigate from one type to another
3. UML syntax for attribute - visibility name: type = default value

Q9. Which of the following are true about operations of a class?

1. At conceptual level, operations are used to specify the interface of a class
2. The UML syntax for operation is: visibility name (parameter-list) : return-type-expression {property-string}
3. Most people use operation and method interchangeably. Strictly speaking, operation is method declaration while method is referring to the body of procedure.

Q10. Given that classes A -> B -> C are residing on different packages and their dependency is illustrated as above. Which of the following statements is true?

1. Change of a private method of B may affect A.
2. Change of a public method of B may affect A.
3. Change of a public method of C may affect A.
4. Change of a public method of C may affect B.

Q11. Differentiate Attribute and Association.

Objective Questions..

Q12. What is guard condition?

Q13. Knowing UML means one can handle object-oriented analysis and design.

1. True
2. False

Q14. Which of the following statements are true for Activity diagrams?

1. can be used to depict workflow for a particular business activity
2. can be used to explore/discover parallel activities
3. do not tell you who does what and are difficult to trace back to object models
4. all of the above

Q15. Which of the following statement is false?

1. Aggregation is a special kind of Association.
2. Both Aggregation and Composition are 'part-of' relationship.
3. When the whole is deleted, parts in aggregation are also removed
4. All of the above
5. None of the above

Q16. Which statement is correct to test subtyping?

1. Use "is a" e.g. Silky is a Cat; Cat is an Animal
2. Cat is a kind of Animal
3. Animal is a kind of Cat

Objective Questions..

Q17. Which of the following statements are true about Package Diagrams?

1. Package is a grouping mechanism that can be applied to classes only
2. Package in UML is similar to Java, to avoid name collision
3. Package diagrams are particularly useful for testing
4. Package is an object-oriented approach in managing system structure
5. A package may contain class(es), list of classes, another package
6. Whenever a class diagram that encompasses the whole system is no longer legible on a single letter-size sheet of paper

Q18. Which of the following statements is false about the goals of Inception?

1. What is the vision and business case for this project
2. Is it feasible?
3. Are we going to buy or build?
4. Provide accurate estimates of cost
5. Produce a development schedule
6. Get decision from management to proceed or stop

Q19. Would you use sequence diagram or an activity diagram to model process flow that has lot of conditional flows and concurrent processing?

Q20. What is the difference between an attribute and an association?



Objective Questions..

Q21. State typical applications of deployment diagrams.

Q22. Define the following

- Magic State
- Guard
- A minimal state machine
- Association
- Generalization
- Aggregation

Short Questions

- Q1. Write short Note on Activity diagrams
- Q2. Write short note on Conceptual model of UML
- Q3. Discuss the role of Use Case model in object oriented requirement analysis.
- Q4. What is a history state? How it helps in the development of a activity diagram?
- Q5. What is the difference between Use-Case Model and Domain-Object model?
- Q6. Differentiate between Sequence diagram and collaboration diagram
- Q7. What is the guard condition? Explain its significance with the help of an example.
- Q8. Differentiate between component and deployment diagram.
- Q9. Explain various substates in activity diagram with the help of examples.
- Q10. Draw state transition diagram for inserting and removing items in a queue.
- Q11. What is the way to describe a modeling technique?
- Q12. Write short note on Component Diagram and Deployment Diagrams
- Q13. Differentiate Association, Aggregation and Composition with example
- Q14. Discuss the UML history.

Long Questions

- Q1. What is architecture of UML? How many types of diagrams are used in UML? Explain each.
- Q2. Write about features of UML system.
- Q3. List UML diagrams and explain any two in detail.
- Q4. What is the purpose of Extension and Inclusion association between Use Cases? Explain with the help of an example.
- Q5. Write the history of UML in short. What are the various UML diagrams write function of each diagram in very short. Describe basic building blocks of UML.
- Q6. Describe sequence diagram. Draw the sequence diagram for the use case Returning Item.
- Q7. Discuss State Machine diagram. Draw the state transition diagram for a stack.
- Q8. Discuss the different types of relationships in class diagram with example.
- Q9. Discuss the different views and their responsibilities in UML architecture.

Research Problems

- P1. Consider a car rental application. The rental agency has multiple offices/ locations where customer can test drive and then select a car for rental (local or to outstation). The period of rental, terms and conditions for rental is flexible. Software has to take responsibility for loaning cars, keeping track of availability of cars, return of cars, billing, maintenance activities for cars and keeping track of drivers availability and assignment in case of chauffeur driver car rentals. Draw use case diagram for above application taking advantage of full UML notation for use case diagrams
- P2. Prepare an activity diagram for computing a restaurant bill. There should be a charge for each delivered item .The total amount should be subject to tax and a sevice charge of 18% for groups of six or more. For smaller groups, there should be a blank entry for a gratuity according to the customer discretion. Any coupons or gift certificates submitted by a customer should be subtracted

References

1. Ivar Jacobson, "Object Oriented Software Engineering", Pearson, 2004.
2. Grady Booch, James Runbaugh, Ivar Jacobson, "The UML User Guide", Pearson, 2004
3. R. Fairley, "Software Engineering Concepts", Tata McGraw Hill, 1997.
4. P. Jalote, "An Integrated approach to Software Engineering", Narosa, 1991.
5. Stephen R. Schach, "Classical & Object Oriented Software Engineering", IRWIN, 1996.
6. James Peter, W Pedrycz, "Software Engineering", John Wiley & Sons
7. Sommerville, "Software Engineering", Addison Wesley, 1999.
8. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/userguide.html
9. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/statemachinediagram.html
10. <http://www.developer.com/design/article.php/2238131/State-Diagram-in-UML.htm>