SOFTWARE ENGINEERING LAB

Dr:- shimaa saad Assistant lecture,

Department of computer science

Email : shimaa_saad@yahoo.com

Telephone : 01222956341

EXERCISE NO. 5

Aim: Steps to draw the Use Case Diagram using Rational Rose or smart draw.

Hardware Requirements: Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.

Software Requirements: Rational Rose or smart draw, Windows XP,

Theory:

- According to the UML specification **a use case diagram** is –

a diagram that shows the relationships among actors and use cases within a system.‖

Use case diagrams are often used to:

• Provide an overview of all or part of the usage requirements for a system or organization in the form of an essential model or a business model

• Communicate the scope of a development project

• Model your analysis of your usage requirements in the form of a system use case model

- Use case models should be developed from the point of view of your project stakeholders and not from the (often technical) point of view of developers. There are guidelines for:

Use Cases, Actors, Relationships, System Boundary Boxes

1. Use Cases

- A use case describes a sequence of actions that provide a measurable value to an actor.
- A use case is drawn as a horizontal ellipse on a UML use case diagram.
- 1. Use Case Names Begin With a Strong Verb
- 2. Name Use Cases Using Domain Terminology
- 3. Place Your Primary Use Cases In The Top-Left Corner Of The Diagram
- 4. Imply Timing Considerations By Stacking Use Cases.

2. Actors

- An actor is a person, organization, or external system that plays a role in one or more interactions with your system (actors are typically drawn as stick figures on UML Use Case diagrams).
- 1. Place Your Primary Actor(S) In The Top-Left Corner Of The Diagram
- 2. Draw Actors To The Outside Of A Use Case Diagram
- 3. Name Actors With Singular, Business-Relevant Nouns
- 4. Associate Each Actor With One Or More Use Cases
- 5. Actors Model Roles, Not Positions
- 6. Use <> to Indicate System Actors
- 7. Actors Don‘t Interact With One Another
- 8. Introduce an Actor Called ―Time‖ to Initiate Scheduled Events

3. Relationships

- There are several types of relationships that may appear on a use case diagram:

• An association between an actor and a use case

• An association between two use cases

• A generalization between two actors

• A generalization between two use cases Associations are depicted as lines connecting two modeling elements with an optional openheaded arrowhead on one end of the line indicating the direction of the initial invocation of the relationship.

Generalizations are depicted as a close-headed arrow with the arrow pointing towards the more general modeling element.

(توضيحية)

1. Indicate An Association Between An Actor And A Use Case If The Actor Appears Within The Use Case Logic
2. Avoid Arrowheads On Actor-Use Case Relationships
3. Apply <<include>> When You Know Exactly When To Invoke The Use Case

4. Apply <<extend>> When A Use Case May Be Invoked Across Several Use Case Steps
5. Introduce <<extend>> associations sparingly
6. Generalize Use Cases When a Single Condition Results In Significantly New Business Logic
7. Do Not Apply <<uses>>, <<includes>>, or <<extends>>
8. Avoid More Than Two Levels Of Use Case Associations
9. Place An Included Use Case To The Right Of The Invoking Use Case
10. Place An Extending Use Case Below The Parent Use Case
11. Apply the –Is Likel Rule to Use Case Generalization
12. Place an Inheriting Use Case Below The Base Use Case
13. Apply the –Is Likel Rule to Actor Inheritance
14. Place an Inheriting Actor Below the Parent Actor

4. System Boundary Boxes

- The rectangle around the use cases is called the system boundary box and as the name suggests it indicates the scope of your system – the use cases inside the rectangle represent the functionality that you intend to implement.
- 1. Indicate Release Scope with a System Boundary Box.
- 2. Avoid Meaningless System Boundary Boxes.

Creating Use Case Diagrams

- we start by identifying as many actors as possible. You should ask how the actors interact with the system to identify an initial set of use cases. Then, on the diagram, you connect the actors with the use cases with which they are involved.
- If actor supplies information, initiates the use case, or receives any information as a result of the use case, then there should be an association between them.

Rational Rose

- Procedure (for rational rose):

• Click on the File menu and select New.

• Now from the Dialogue Box that appears ,select the language which you want to use for creating your model.

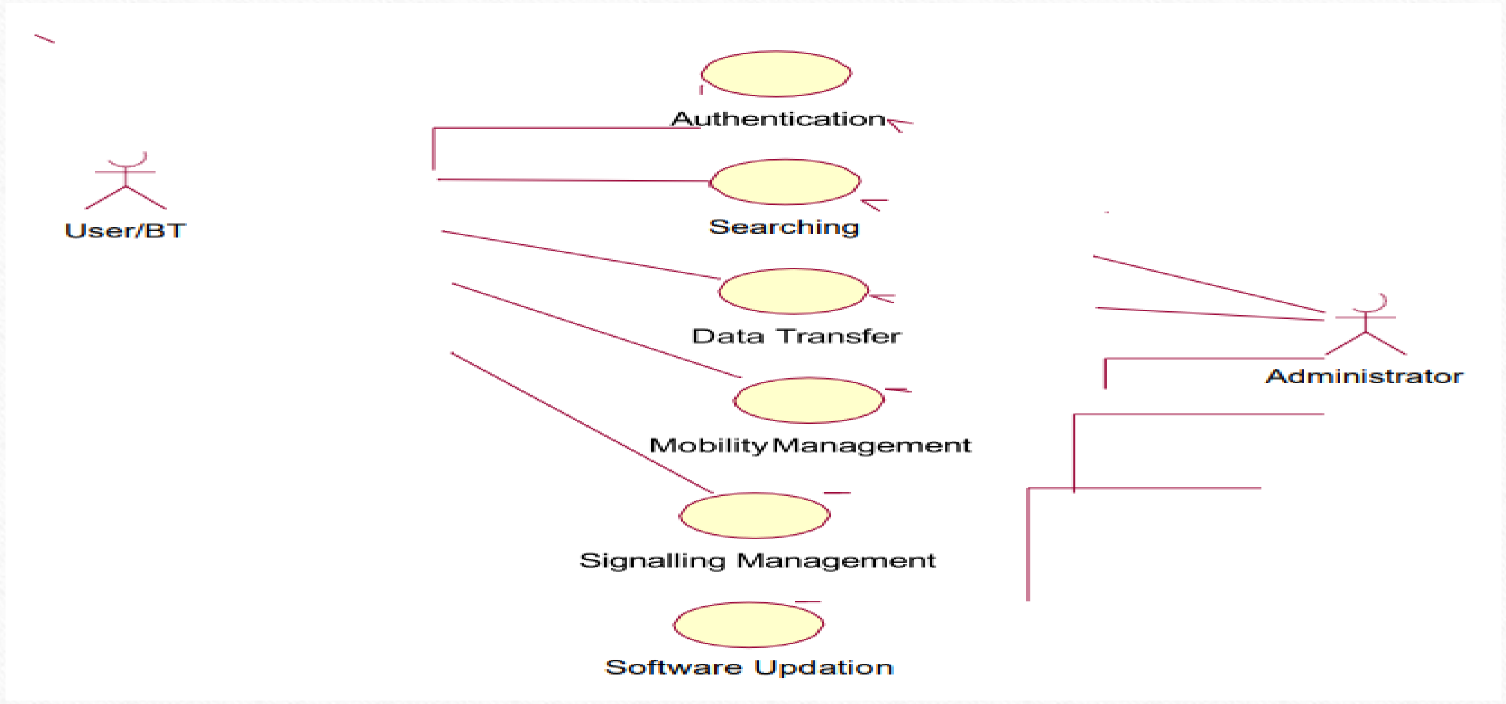
• In the left hand side window of Rational Rose right click on ―Use Case view‖ and select New>Use Case Diagram.

• Enter the name of new Use Case file in the space provided, and then click on that file name.

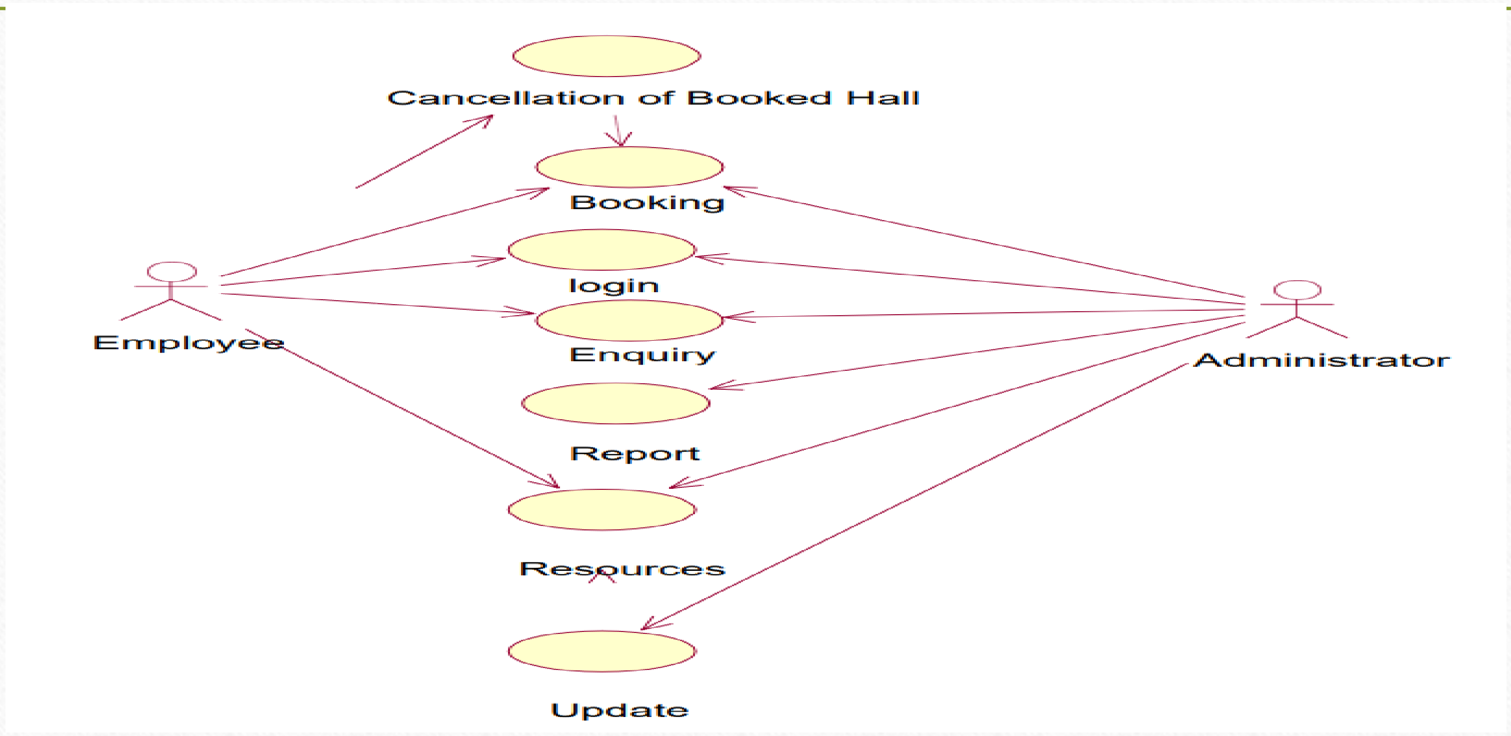
Rational Rose

- You can now use the window that appears on right hand side to draw your Use Case diagram using the buttons provided on the vertical toolbar.
- Conclusion: The use case diagram was made successfully by following the steps described above.

Some Sample Use Case Diagrams are given below for illustration purpose:



Use Case Diagram for Resource Management



EXERCISE NO. 6

- **AIM** :- To draw a sample activity diagram for real project or system.
- **Hardware Requirements:** Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.
- **Software Requirements:** Rational Rose, Windows XP,

THEORY

- UML 2 activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule.
- Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don‘t require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development.

Let‘s start by describing the basic notation :

- • Initial node. The filled in circle is the starting point of the diagram. An initial node isn‘t required although it does make it significantly easier to read the diagram.
- • Activity final node. The filled circle with a border is the ending point. An activity diagram can have zero or more activity final nodes.
- • Activity. The rounded rectangles represent activities that occur. An activity may be physical, such as Inspect Forms, or electronic, such as Display Create Student Screen.
- • Flow/edge. The arrows on the diagram. Although there is a subtle difference between flows and edges, never a practical purpose for the difference although.
- • Fork. A black bar with one flow going into it and several leaving it. This denotes the beginning of parallel activity.

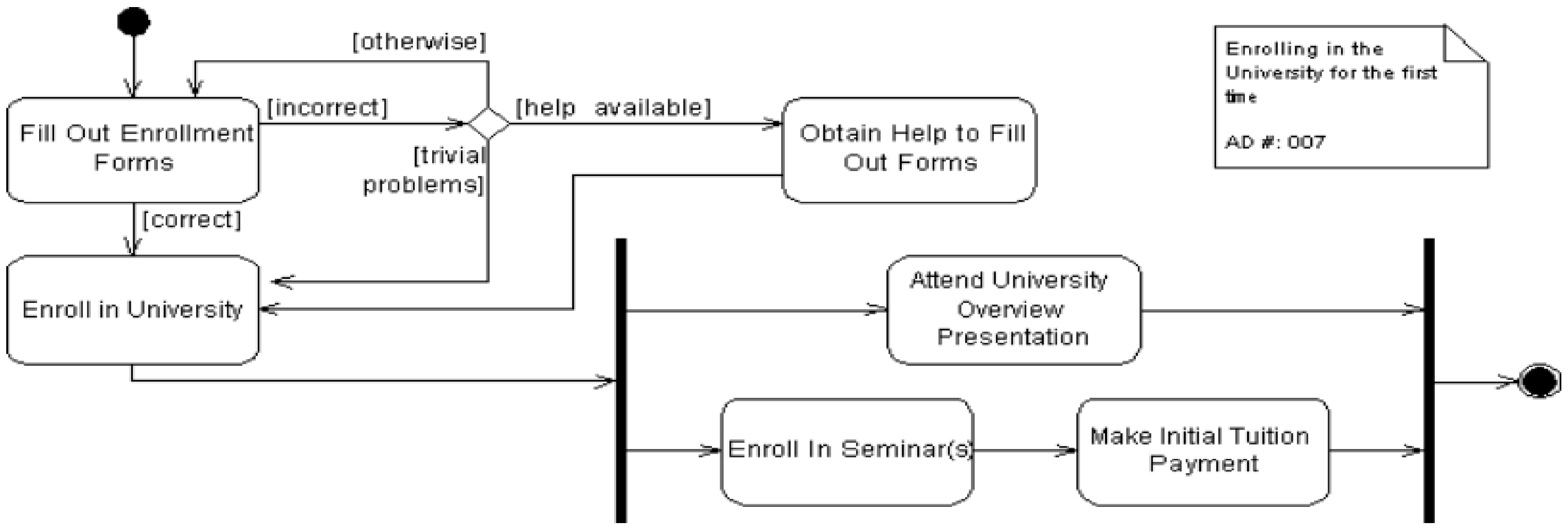
- • Join. A black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the end of parallel processing.
- • Condition. Text such as [Incorrect Form] on a flow, defining a guard which must evaluate to true in order to traverse the node.
- • Decision. A diamond with one flow entering and several leaving. The flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.
- • Merge. A diamond with several flows entering and one leaving. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow.
- • Partition. If figure is organized into three partitions, it is also called swimlanes, indicating who/what is performing the activities (either the Applicant, Registrar, or System).
- • Sub-activity indicator. The rake in the bottom corner of an activity, such as in the Apply to University activity, indicates that the activity is described by a more finely detailed activity diagram.
- • Flow final. The circle with the X through it. This indicates that the process stops at this point.

GUIDELINES ASSOCIATED FOR DRAWING AN ACTIVITY DIAGRAM

- 1.General Guidelines
- 2.Activities
- 3.Decision Points
- 4.Guards
- 5.Parallel Activities
- 6.Swimlane Guidelines
- 7.Action-Object Guidelines

1. General Guidelines

Figure1. Modeling a business process with a UML Activity Diagram.


[otherwise]
[incorrect]
[help available]
[trivial problems]
[correct]
Fill Out Enrollment Forms
Obtain Help to Fill Out Forms
Enroll in University
Attend University Overview Presentation
Enroll In Seminar(s)
Make Initial Tuition Payment
Enrolling in the University for the first time
AD #: 007

(توضيحية)

- 1. Place The Start Point In The Top-Left Corner. A start point is modeled with a filled in circle, using the same notation that UML State Chart diagrams use. Every UML Activity Diagram should have a starting point, and placing it in the top-left corner reflects the way that people in Western cultures begin reading. Figure1, which models the business process of enrolling in a university, takes this approach.
- 2. Always Include an Ending Point. An ending point is modeled with a filled in circle with a border around it, using the same notation that UML State Chart diagrams use. Figure1 is interesting because it does not include an end point because it describes a continuous process – sometimes the guidelines don‘t apply.
- 3. Flowcharting Operations Implies the Need to Simplify. A good rule of thumb is that if an operation is so complex you need to develop a UML Activity diagram to understand it that you should consider refactoring it.

2. Activities

- An activity, also known as an activity state, on a UML Activity diagram typically represents the invocation of an operation, a step in a business process, or an entire business process.
- 1. Question —Black Hole‖ Activities. A black hole activity is one that has transitions into it but none out, typically indicating that you have either missed one or more transitions.
- 2. Question —Miracle‖ Activities. A miracle activity is one that has transitions out of it but none into it, something that should be true only of start points.

3. Decision Points(توضيحية)

- A decision point is modeled as a diamond on a UML Activity diagram.
- 1. Decision Points Should Reflect the Previous Activity. In figure1 we see that there is no label on the decision point, unlike traditional flowcharts which would include text describing the actual decision being made, we need to imply that the decision concerns whether the person was enrolled in the university based on the activity that the decision point follows. The guards, depicted using the format [description], on the transitions leaving the decision point also help to describe the decision point.
- 2. Avoid Superfluous Decision Points. The Fill Out Enrollment Forms activity in FIGURE1 includes an implied decision point, a check to see that the forms are filled out properly, which simplified the diagram by avoiding an additional diamond.

4. Guards

A guard is a condition that must be true in order to traverse a transition.

- 1. Each Transition Leaving a Decision Point Must Have a Guard
- 2. Guards Should Not Overlap. For example guards such as x 0 are consistent whereas guard such as x <= 0 and x >= 0 are not consistent because they overlap – it isn‘t clear what should happen when x is 0.
- 3. Guards on Decision Points Must Form a Complete Set. For example, guards such as x < 0 and x >0 are not complete because it isn‘t clear what happens when x is 0.
- 4. Exit Transition Guards and Activity Invariants Must Form a Complete Set. An activity invariant is a condition that is always true when your system is processing an activity.
- 5. Apply a [Otherwise] Guard for ―Fall Through‖ Logic.
- 6. Guards Are Optional. It is very common for a transition to not include a guard, even when an activity includes several exit transitions.

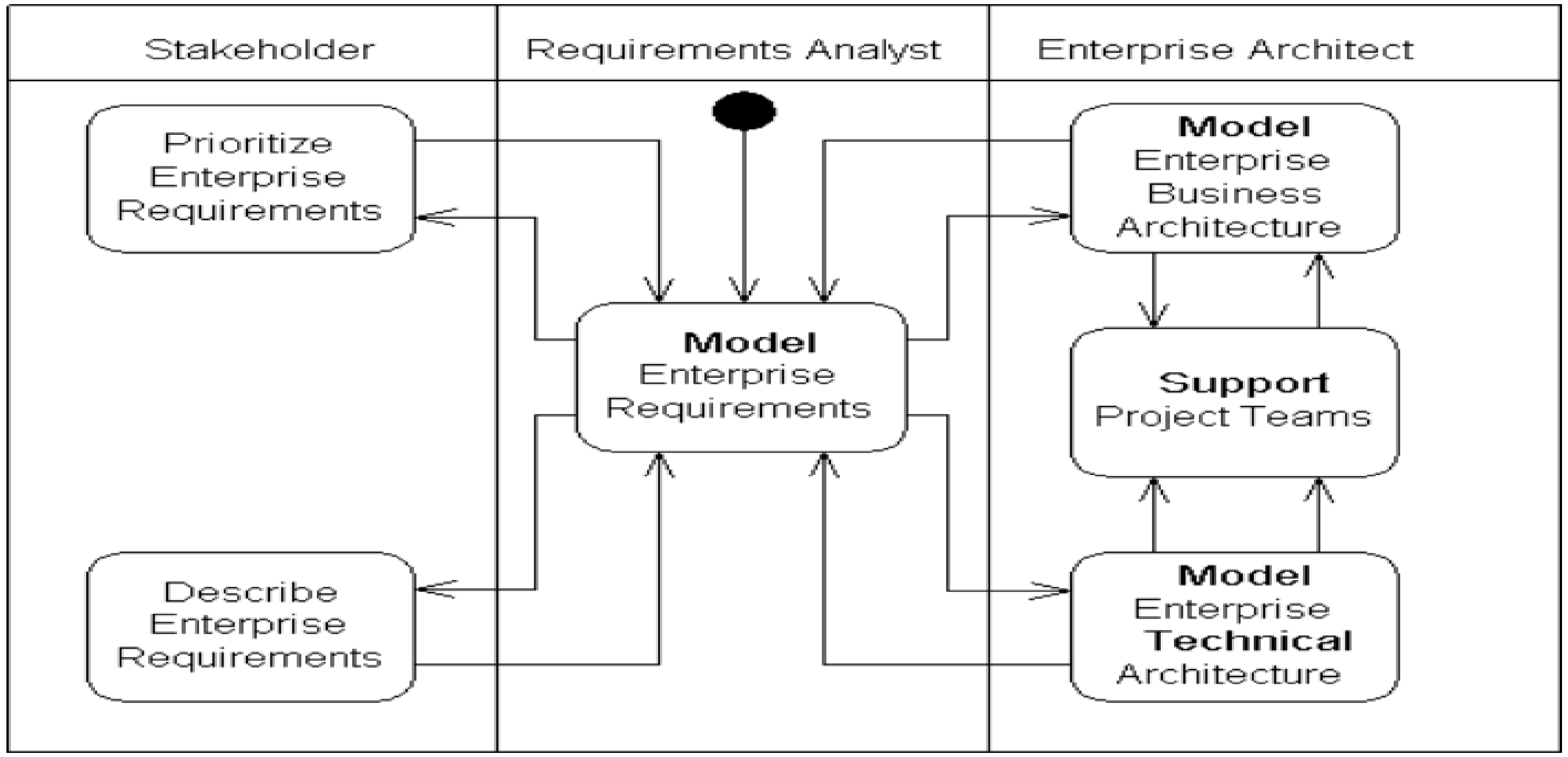
5. Parallel Activities

- It is possible to show that activities can occur in parallel, as you see in FIGURE 1 depicted using two parallel bars. The first bar is called a fork, it has one transition entering it and two or more transitions leaving it. The second bar is a join, with two or more transitions entering it and only one leaving it.

- 1. A Fork Should Have a Corresponding Join. In general, for every start (fork) there is an end (join). In UML 2 it is not required to have a join, but it usually makes sense.
- 2. Forks Have One Entry Transition.
- 3. Joins Have One Exit Transition
- 4. Avoid Superfluous Forks.

FIGURE 2 depicts a simplified description of the software process of enterprise architectural modeling, a part of the Enterprise Unified Process (EUP). There is significant opportunity for parallelism in this process, in fact all of these activities could happen in parallel, but forks were not introduced because they would only have cluttered the diagram.

Figure2. A UML activity diagram for the enterprise architectural modeling (simplified).

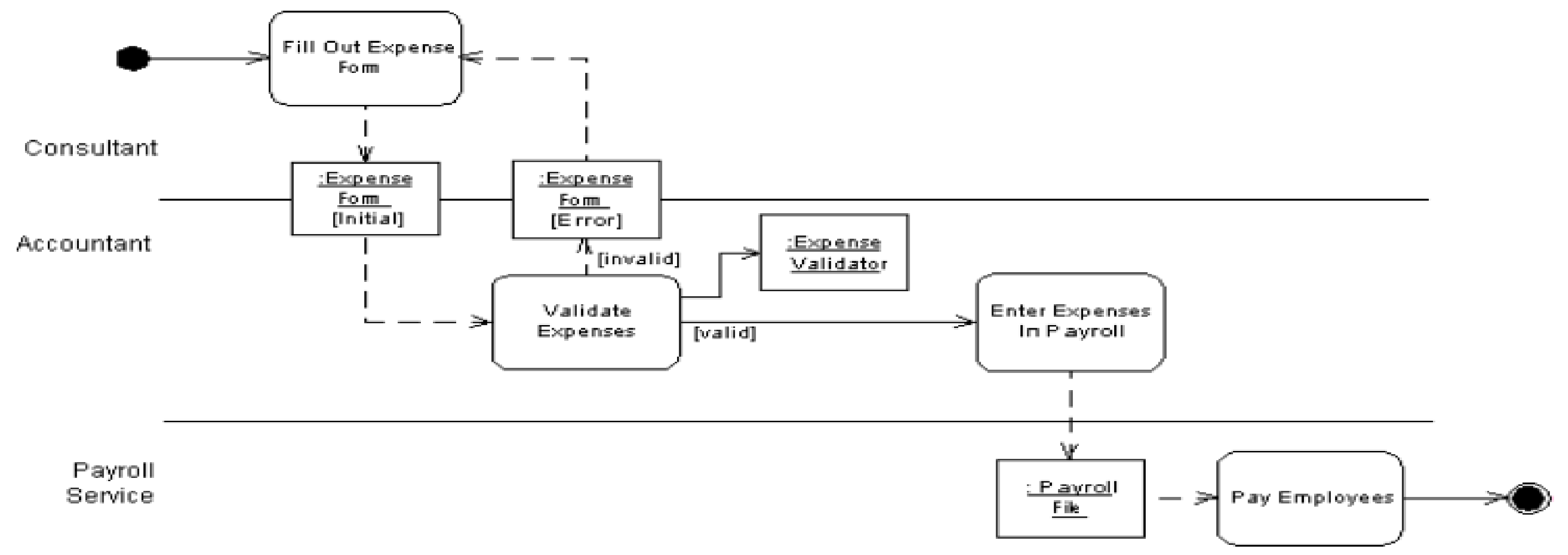

Stakeholder
Requirements Analyst
Enterprise Architect
Prioritize Enterprise Requirements
Describe Enterprise Requirements
Model Enterprise Requirements
Model Enterprise Business Architecture
Support Project Teams
Model Enterprise **Technical** Architecture

6. Swimlane Guidelines

- A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread.
- FIGURE 2 includes three swimlanes, one for each actor

Figure 3. Submitting expenses.

Figure 3. Submitting expenses.



- 1. Order Swimlanes in a Logical Manner.
- 2. Apply Swim Lanes To Linear Processes. A good rule of thumb is that swimlanes are best applied to linear processes, unlike the one depicted in FIGURE 3.
- 3. Have Less Than Five Swimlanes.
- 4. Consider Swimareas For Complex Diagrams.
- 5. Swimareas Suggest The Need to Reorganize Into Smaller Activity Diagrams.
- 6. Consider Horizontal Swimlanes for Business Processes.

In FIGURE 3 you see that the swimlanes are drawn horizontally, going against common convention of drawing them vertically.

7 Action-Object Guidelines

- Activities act on objects, In the strict object-oriented sense of the term an action object is a system object, a software construct. In the looser, and much more useful for business application modeling, sense of the term an action object is any sort of item.

For example in FIGURE 3 the Expense Form action object is likely a paper form.

- 1. Place Shared Action Objects on Swimlane Separators
- 2. When An Object Appears Several Time Apply State Names
- 3. State Names Should Reflect the Lifecycle Stage of an Action Object
- 4. Show Only Critical Inputs and Outputs
- 5. Depict Action Objects As Smaller Than Activities

Conclusion: The activity diagram was made successfully by following the steps described above

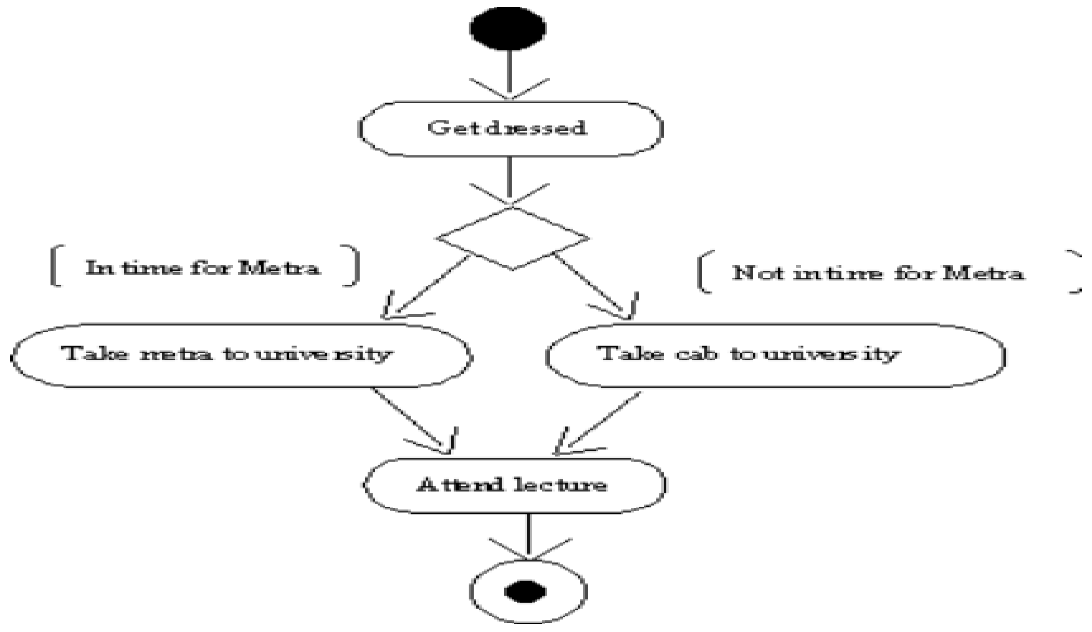
An example Activity diagram
SAMPLE ACTIVITY DIAGRAMS: SAMPLE 1

As you can see in Figure , the first activity is to get dressed to leave for the lecture. A decision then has to be made, depending on the time available for the lecture to start, and the timings of the public trains (metra). If there is sufficient time to catch the train, then take the train; else, flag down a cab to the University. The final activity is to actually attend the lecture, after which the Activity diagram terminates.

SAMPLE ACTIVITY DIAGRAMS: SAMPLE 1

SAMPLE 1:

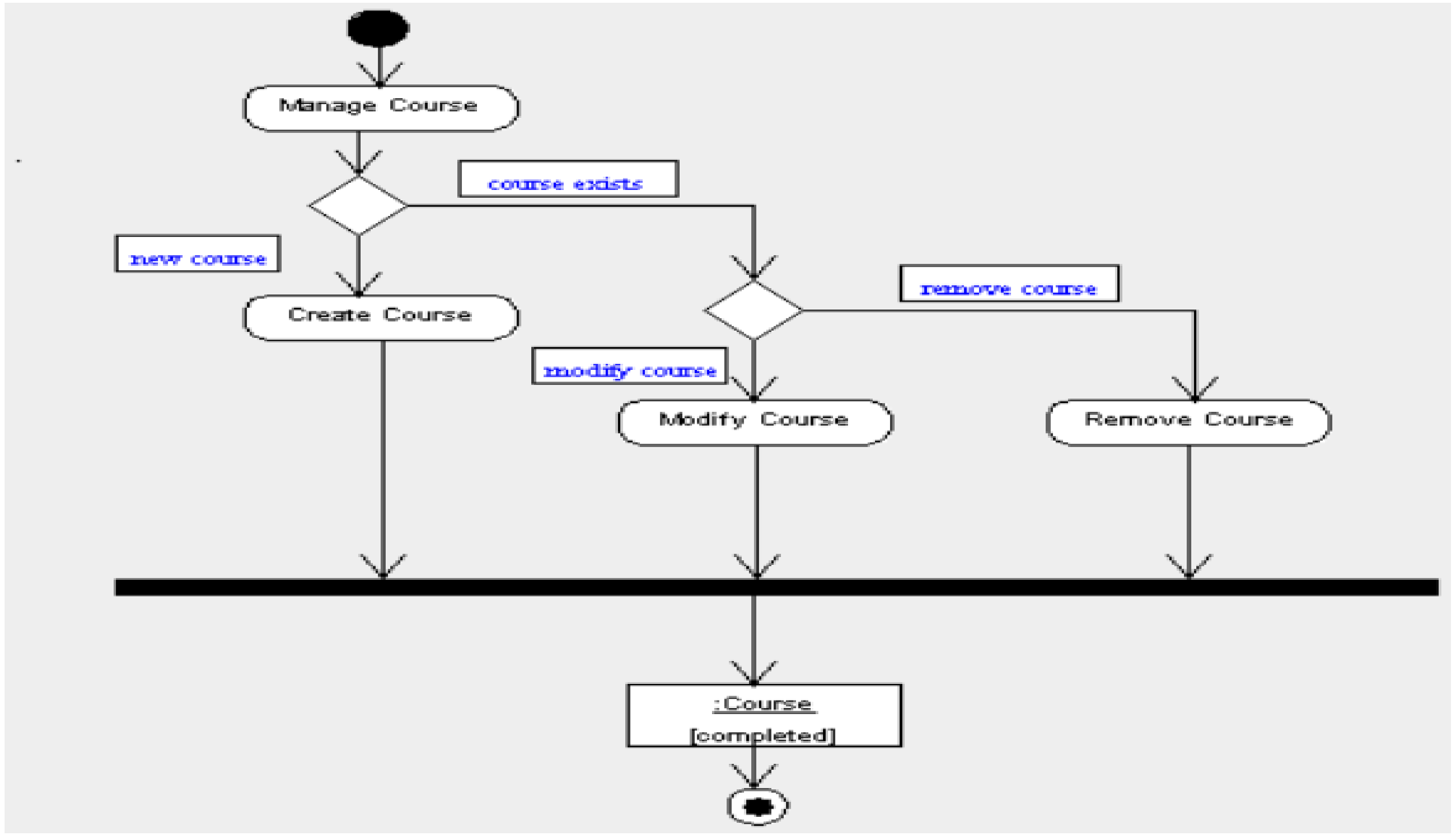
Let us consider the example of attending a course lecture, at 8 am.



SAMPLE 2: Identifying the activities and transitions **for managing course information**

- The course administrator is responsible for managing course information in the Courseware Management System. As part of managing the course information, the course administrator carries out the following activities:
- • Check if course exists
- • If course is new, proceed to the "Create Course" step
- • If course exists, check what operation is desired—whether to modify the course or remove the course
- • If the modify course operation is selected by the course administrator, the "Modify Course" activity is performed
- • If the remove course operation is selected by the course administrator, the "Remove Course" activity is performed In the first step in this Activity diagram, the system determines whether the course that is to be managed is a new course or an existing course. For managing a new course, a separate activity, "Create Course," is performed. On the other hand, if a course exists, the course administrator can perform two different activities—modify an existing course or remove an existing course. Hence, the system checks the type of operation desired based on which two separate activities can be performed—"Modify Course" or "Remove Course".

Activity diagram


Manage Course
course exists
new course
Create Course
remove course
modify course
Modify Course
Remove Course
:Course
[completed]