Object Oriented Analysis & Design

Week 5

Use Case Diagram and the OO Model

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 4

- Software development best practices were developed to counter the huge number of failing software development projects. The best practices are: develop software iteratively, use component based architectures, continuously verify software quality, manage requirements, visually model software, control changes to software.
- Any modern object-oriented approach to developing information systems must be use-case driven, architecture-centric, and iterative and incremental.
- The UP project lifecycle is divided into 4 phases (inception, elaboration, construction and transition), with each phase having a deliverable (or milestone as we call them in project management).
- The core workflows of the UP are requirements, analysis, design, implementation and testing.
- The support workflows are project management, configuration and change management, and environment.
- The Rational Unified Process (RUP) is a process product developed and marketed by Rational Software Corporation that provides the details required for executing projects using the UP, including guidelines, templates, and tool assistance; essentially, it is a commercial process product providing the details or content for the UP framework.

Content

- Introduction
- The OO Model
- Use case diagram

Part 1

Introduction

Introduction

- Hitherto we have described the foundations of object orientation, introduced object oriented analysis and design, and described the unified process with its workflows.
- We also introduced the UML and described it as the language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems .
- The question now is at what point in our design will we start using the UML? Well, the answer is right from the requirements phase! It starts with the design of the use case diagram which is what we call a central view of the system (we shall see why).
- The rest of this course is about describing the different views of the system and expressing them using the UML. This helps us to understand what we wish to construct and makes the implementation easier.
- We begin by describing the views of the OO model and its relevance in the next section; thereafter an introduction to requirements follows.
- The lesson then examines the use case diagram in detail and includes examples to show how to build use cases from given scenarios.

Part 2

The OO Model

Introduction

- The object oriented model is made up of 4 models. These models represent the different views of the system.
- What is the purpose of these views? The views represent the full architecture of our software. Using the views we can understand the architecture of what is being developed and thus come up with a software that truly meets the requirements.
- Before examining these 4 models let us start by describing the architectural model as suggested by Kruchten (1995). These are shown in fig 1.

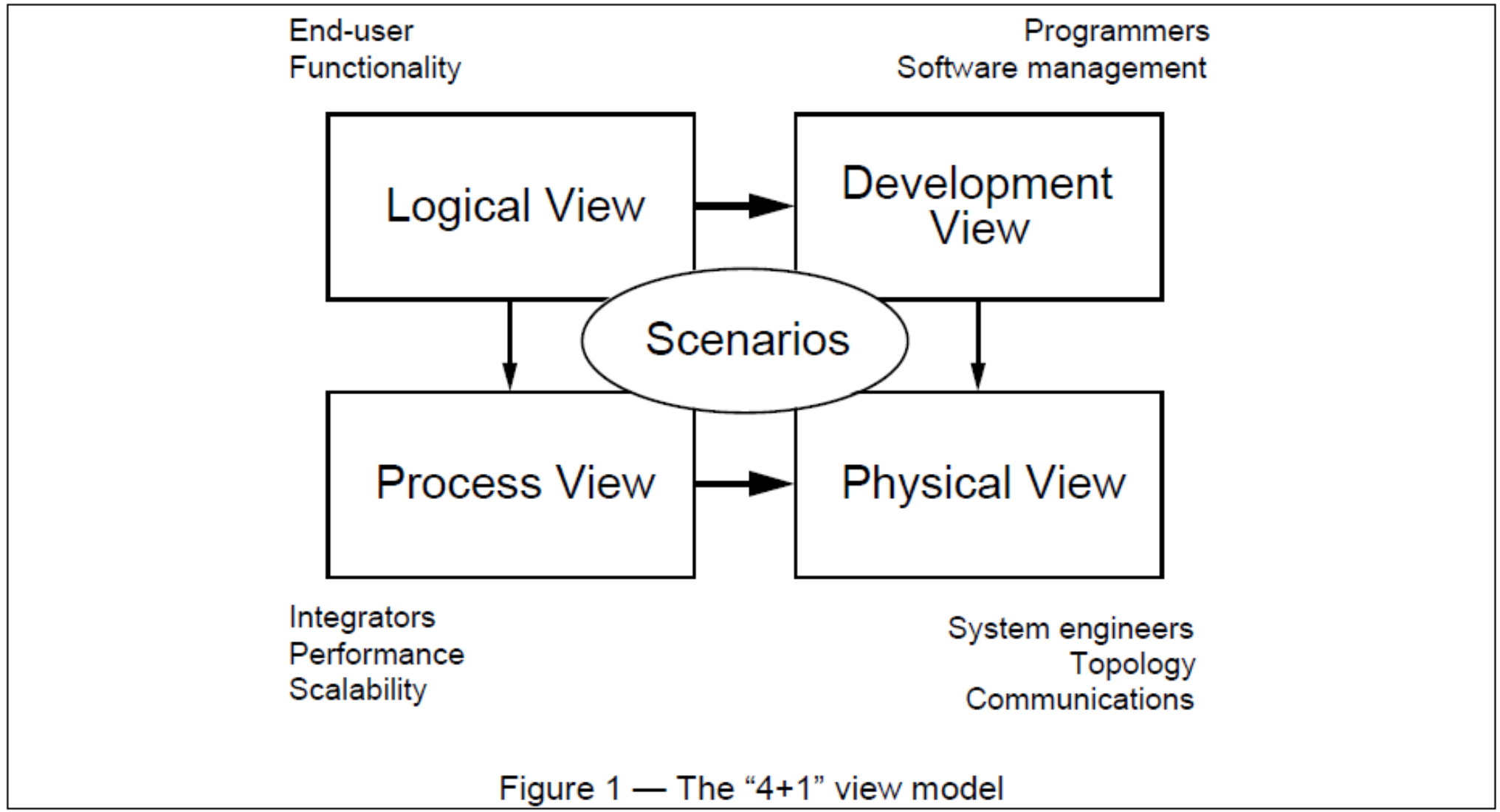

Fig 1. Five views of the architectural model. (Kruchten, 1995)

2.1 The architectural model

- Kruchten (1995) describes the architecture as "a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. This use of multiple views allows to address separately the concerns of the various 'stakeholders' of the architecture: end-user, developers, systems engineers, project managers, etc., and to handle separately the functional and non functional requirements."
- He describes the 5 views as follows:
  - Logical – this is the object model of the design
  - Process – concurrency and synchronization view of the design
  - Physical – mapping of software onto the hardware
  - Deployment – static configuration of the software in its development environment
  - Use case view – this is the +1 view and shows an illustration of the whole architecture.
- We extend this view concept further to describe the models of object oriented development as shown in fig 2.

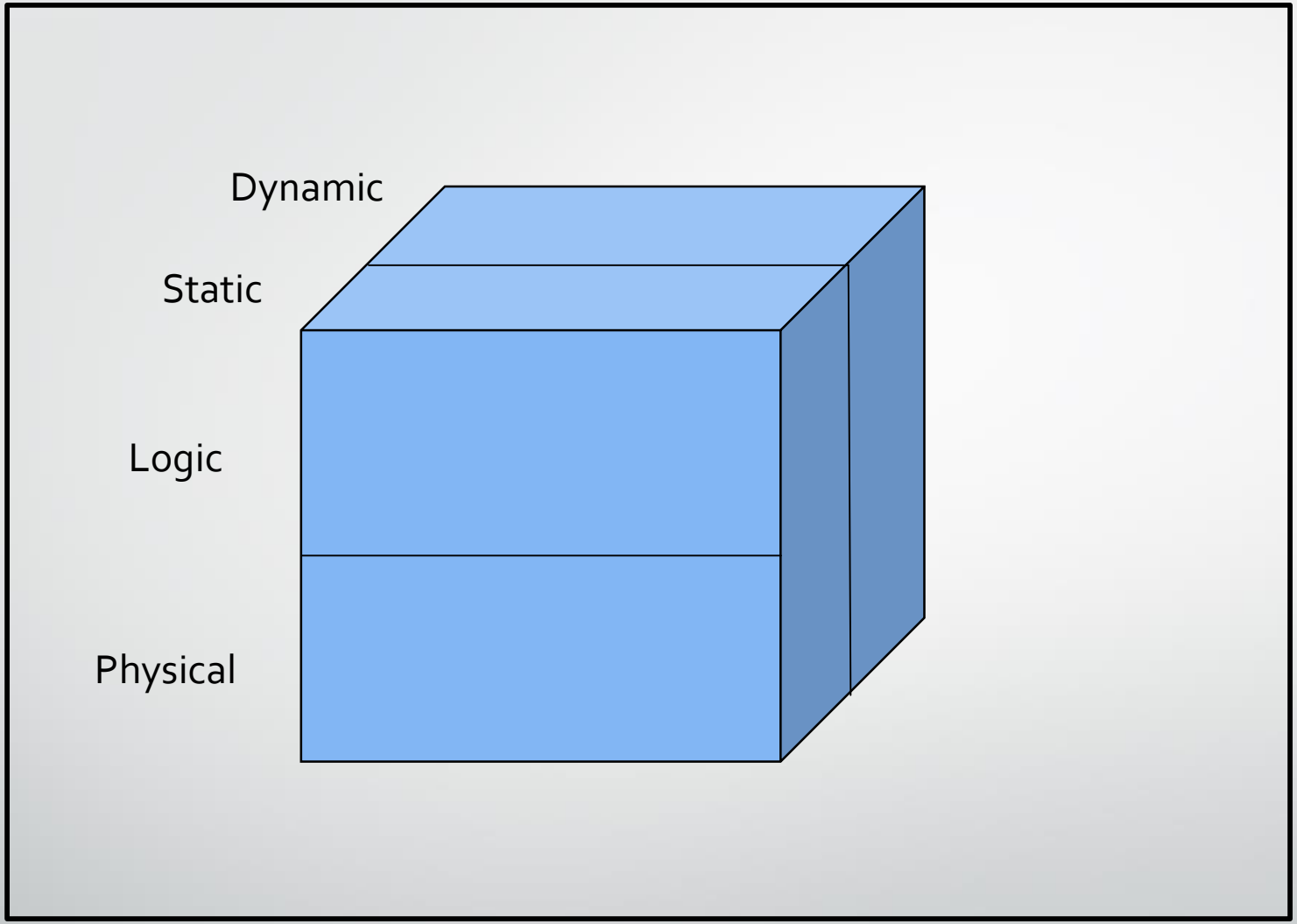

Fig 2. Models of object oriented development

2.2. Models of OO Development

- We are thus presented with a 4+1 view of the OO model. For each view there are a number of diagrams that denote the respective view of the system's model.
  - Process view – activity diagrams and state chart diagrams
  - Deployment view – deployment diagrams and component diagrams
  - Logical view – class diagram
  - Dynamic view – behavioral diagrams (interaction diagrams)

    +
  - Use case view – this is considered the pivot since it drives the development of all the other views from the requirements phase.

2.2 Models of OO Development (cont'd)

- The UML diagrams that are of interest in fulfilling the different architectural aspects of the system are:
- Use case diagram
- Class Diagram
- Behavioral diagrams
  - State chart diagrams
  - Object diagram
  - Activity diagrams
  - Interaction diagrams
    - Sequence diagrams
    - Collaboration diagrams

2.2 Models of OO Development

- Implementation diagrams

  - Component diagram

  - Deployment diagram

- In this lesson we discuss the use case diagram

Part 2

Use case diagram

What is the use case diagram?

- The use case diagram is core to the requirements modeling function of the system.
- It is used to depict all the functional requirements of the system.
- It is a view of the system from a user's perspective. It thus demonstrates the capability of the system as a whole.
- The components or parts of the use case diagram are:

  Actor

  Use case

  System boundary

  Relationship

  Actor relationship
- Since the users are the central focus of attention for this diagram it follows that they are the building blocks of the use case diagram.
- The use case diagram also the glue that ties the requirements to all the remaining design phases.

Actor

- An actor is anyone or anything that interacts with the system.
- The actor is a role and not an individual person or specific thing.
- It is the role that a user plays with respect to the system.
- The UML notation for an actor is a stickman. Actors perform use cases and a single actor may perform one or more use cases.
- Examples of actors in a university system may include student, lecturer, supervisor, manager, director, and so on. The emphasis is that it is a role that is played with respect to the system and therefore does not belong to an individual.
- Let us consider the student role in our university system: users can be created by the system administrator such as Tiffany, Jim, Jill, Tim, and so on. However, their role as far as the system is considered is "student"; thus they will have all the permissions that the role has been configured to have in the system.
- Fig 3 shows the notation of the actor (a stickman) together with the role of the actor underneath it. This is the proper way of showing the actor.

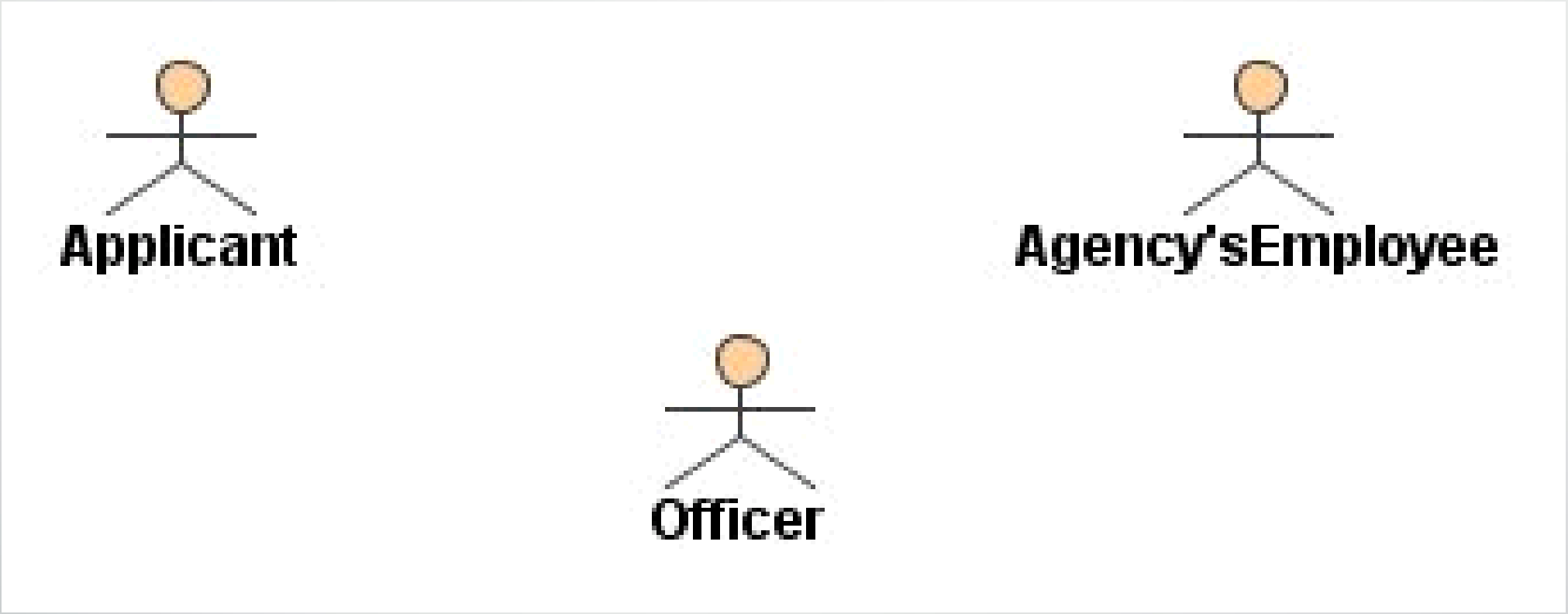

Fig 3. Actor notation example (Ojo and Estevez, 2005)

Actor (cont'd)

- How are the actors of a system identified?
- The actor is the one who interacts with the system or maintains it. An actor may perform 3 actions with respect to the system:
- They may give input to the system
- They may receive output from the system
- They may do both, that is, give input to and receive output from the system.
- Actors are always external to the system, and are therefore not part of the system.

Actor (cont'd)

- There are a number of ways in which the actors can be identified in a given scenario based on the definition of who an actor is:
  - Identify who is supposed to use the system (if they are a group what role then do they each play with respect to the system?)
  - Identify the person who installs the system
  - Identify the person who starts up the system
  - Identify external systems that use this system
  - Identify the people who get information from the system
  - Identify the people that give input to the system.
  - Based on the above identification a candidate list of all actors can be produced.

Actor (cont'd)

- The candidate list can be narrowed down further:
- If more than one actor is using the system in the same way then they are the same actor as far as the system is concerned; you can drop them all and retain one.
- If it is found that two or more actors are using the system in the same way then they are simply the same actor.
- A user may act as one or several actors as it interacts with the system, while several individual users may act as different instances of one and the same actor. (Ojo and Estevez, 2005)
- Since we know there are different actors playing different roles let us describe the different types or categories of actors.

Actor (cont'd)

- There are 4 different types of actors:
- Principle (also known as primary) – this is the one who uses the main functions of the system. The system is primarily made with him being the main user in mind.
- Secondary – this refers to the other actors who are not the primary actor.
- Other system – this refers to any other external system that interacts with this system
- External hardware – refers to the hardware that will be used and are part of the application domain (to be examined in detail in a later lesson).
- Further "when there is more than one actor in a use case, the one that generates the stimulus is called the initiator and the others are participants". (Ojo and Estevez, 2005)

Actors(cont'd)

- Ojo and Estevez (2005) suggest the following in naming actors:
- Group individuals according to how they use the system by identifying the roles they adopt while using the system
- Each role is a potential actor
- Name each role and define its distinguishing characteristics
- Do not equate job titles with roles; roles cut across jobs
- Use common names for existing system; avoid inventing new

Use cases

- A use case is a pattern of behavior exhibited by the system.
- Every use case is a pattern of transactions between the actor and the system; essentially this means the name of a use case generalizes the series of transactions between the two.
- Use cases describe or capture the functional requirements of the system.
- They are the backbone of the use case diagram.
- Ojo and Estevez (2005) define the use case as follows:
- A use case:
- 1. is a collection of task-related activities describing a discrete chunk of a system
- 2. describes a set of actions sequences that a system performs to present an observable result to an actor
- 3. describes a system from an external usage viewpoint

Use case (cont'd)

- To put it aptly a use case represents a dialog between an actor and the system.
- The use case can be considered a snapshot of one aspect of system.
- A use case typically represents a major piece of functionality that is complete from beginning to end; this means that the use case name represents a given sequence of transactions between actor and system that is complete end to end, it doesn't 'hang' at some point. This is why the name given to the use case should make the action clear.
- Most of the use cases are generated in initial phase of the process (requirements stage), but some more may be found as the cycle proceeds.
- A use case may be small or large. It captures a broad view of a primary functionality of the system in a simple manner that can be easily grasped by non technical user.

Use case (cont'd)

- Key attributes of a use case:
- 1) description
- 2) action sequence
- 3) includes variants
- 4) produces observable results
- A use case does not describe:
- 1) user interfaces
- 2) performance goals
- 3) non-functional requirements (Ojo and Estevez, 2005)
- Fig 4 shows the notation used for the use case in the UML. Note that the UML is not very strict regarding where to put the name of the use case.

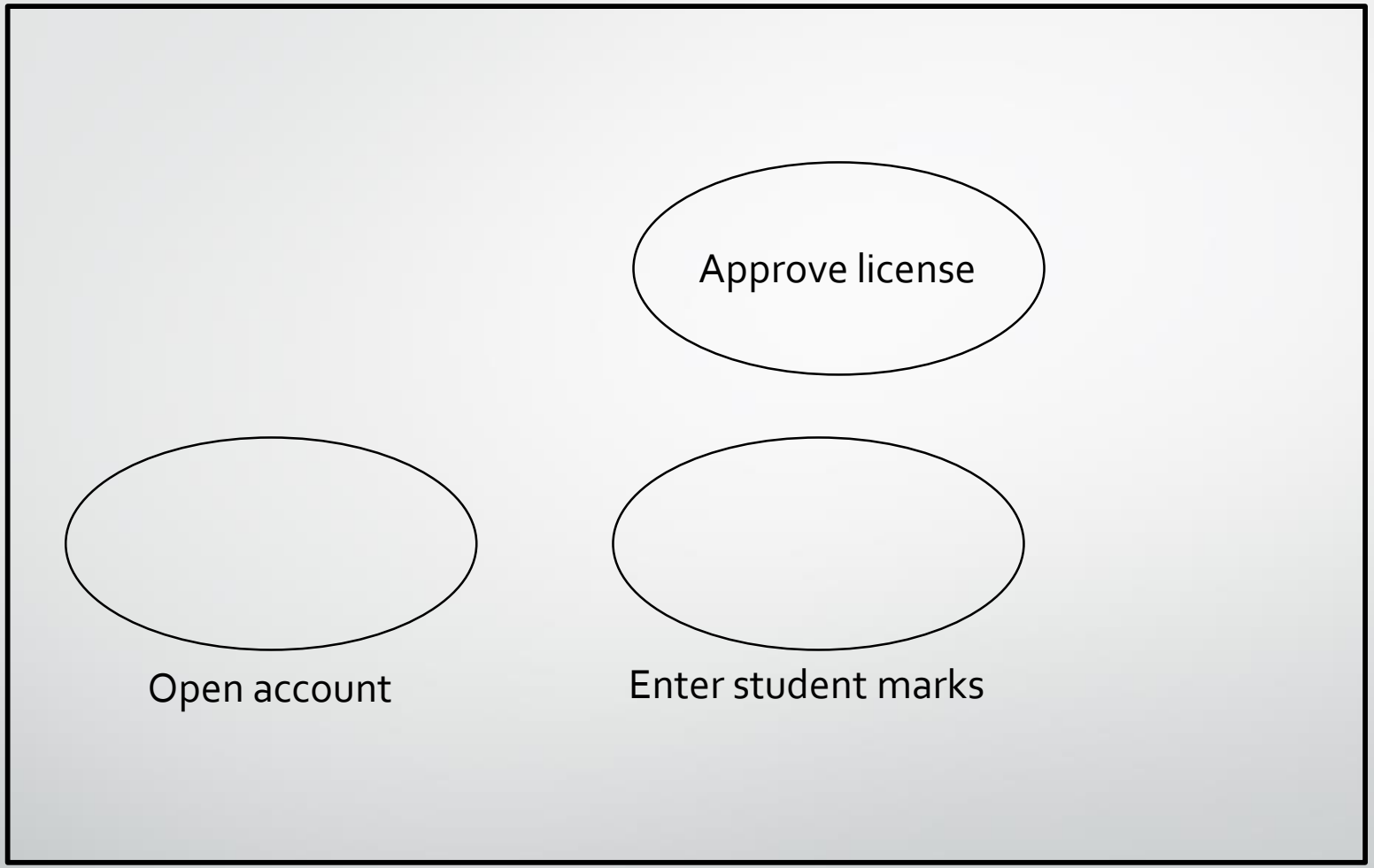

Fig 4. Use case notation

Use case (cont'd)

- Use cases can also be decomposed into other use cases. The key question again is to find a way to identify the use cases in your scenario. How can we do this?

How do we find the use cases?

It helps to ask the following questions in order to find the use cases:

- What functions will the actor want from the system?
- Does the system store information?
- What actors will create, read, update, or delete that information?
- Does the system need to notify an actor about changes in its internal state?

Use case (cont'd)

- Inasmuch as we can decompose use cases into smaller use cases we can also group them together. We do so in the following manner:
- Those use case functionality which are directly dependent on the system environment are placed in interface objects.
- Those functionality dealing with storage and handling of information are placed in entity objects
- Functionality's specific to one or few use cases and not naturally placed in any of the other objects are placed in control objects.
- Consequently, performing this division we obtain a structure which helps us to understand the system from logical view

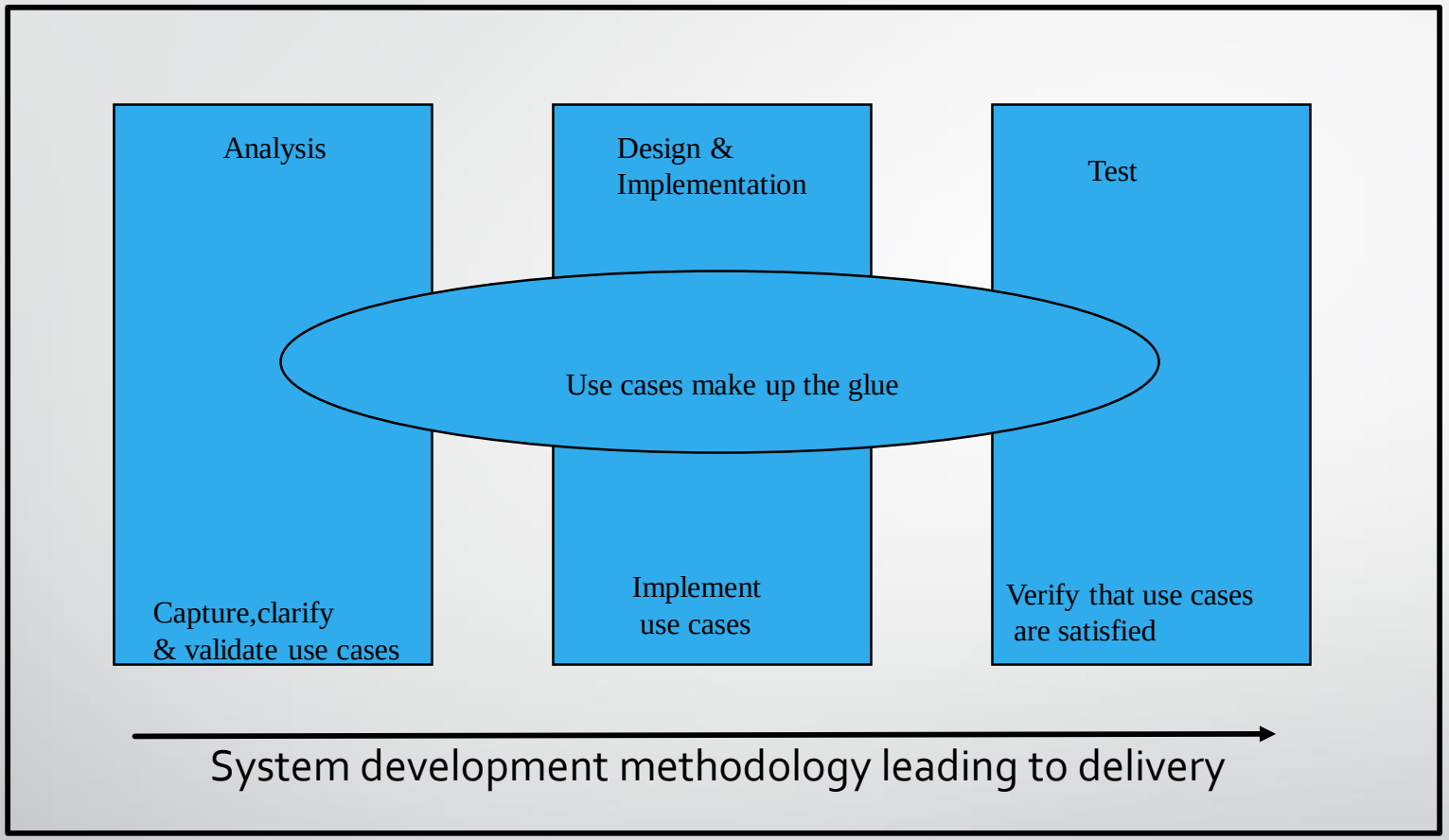

Fig 6. Use cases place in the analysis and design lifecycle

System boundary

- The system boundary shows where the system ends and the external environment begins.
- It is in the form of a rectangle. All uses cases will be found within the system boundary together with their relationships (if any).
- All actors (regardless of type/category) are found outside the system boundary.
- Showing the system boundary is very important in differentiating what belongs to the system and what is external to it.

Relationships

- Relationships exist between use cases, and between actors and use cases.
- The relationship between an actor and a use case is a communicates relationship.
- The relationship between use cases are of 3 types:
  - Generalization
  - Includes
  - Extends
- Note that the generalization relationship can also apply between actors. The UML notation for relationships is << *name of relationship, e.g. includes* >>
- Let us describe each of these relationships.

Relationships

- USES (a.k.a INCLUDES):
- This relationship comes about when multiple use cases share a piece of same functionality.
- This functionality is placed in a separate use case rather than documenting in every use case that needs it.
- Ojo and Estevez (2005) explain it a bit differently though the end result is the same: “the base use case explicitly incorporates the behaviour of another use case at a location specified in the base
- the include relationship never stands alone, but is instantiated as part of some larger base of use cases
- Fig 7 shows an example of implementation of a uses relationship.

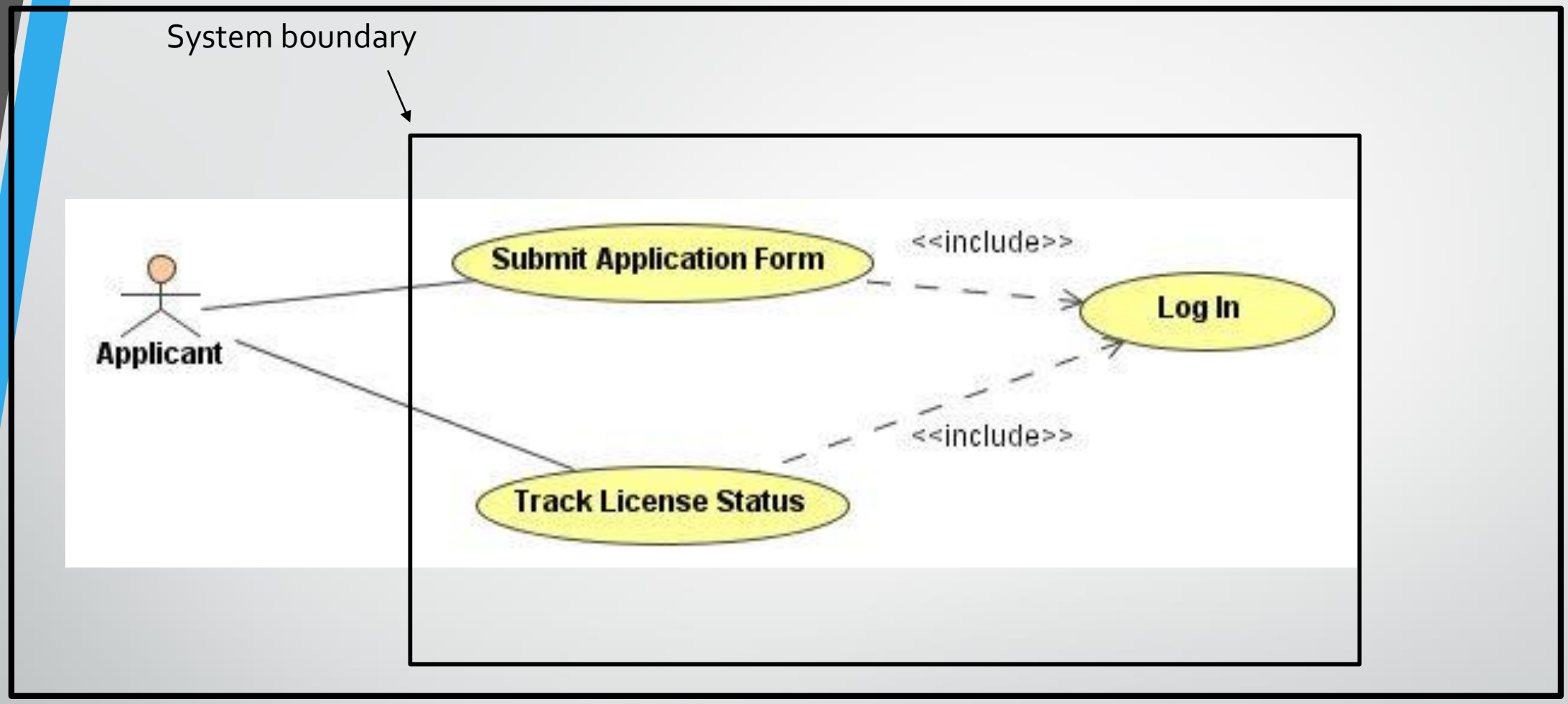

Fig 7. Includes (uses) relationship. (adapted from Ojo and Estevez, 2005)

Relationships (cont'd)

- EXTENDS:
- It is used to show optional behavior, which is required only under certain condition.
- a) the base use case implicitly incorporates the behaviour of another use case at a location specified by the extending use case (extension point)
- b) the base use case may stand alone and usually executes without regards to extension points
- c) depending on the system behaviour, the extension use case will be executed or not (Ojo and Estevez, 2005).
- GENERALIZATION:
- This shows relationships where a specialized use cases can be generalized into one general description use case. For example process student marks use case can refer to enter student marks, calculate grade, and so on, special use cases.
- Fig 8 shows an example of generalization.

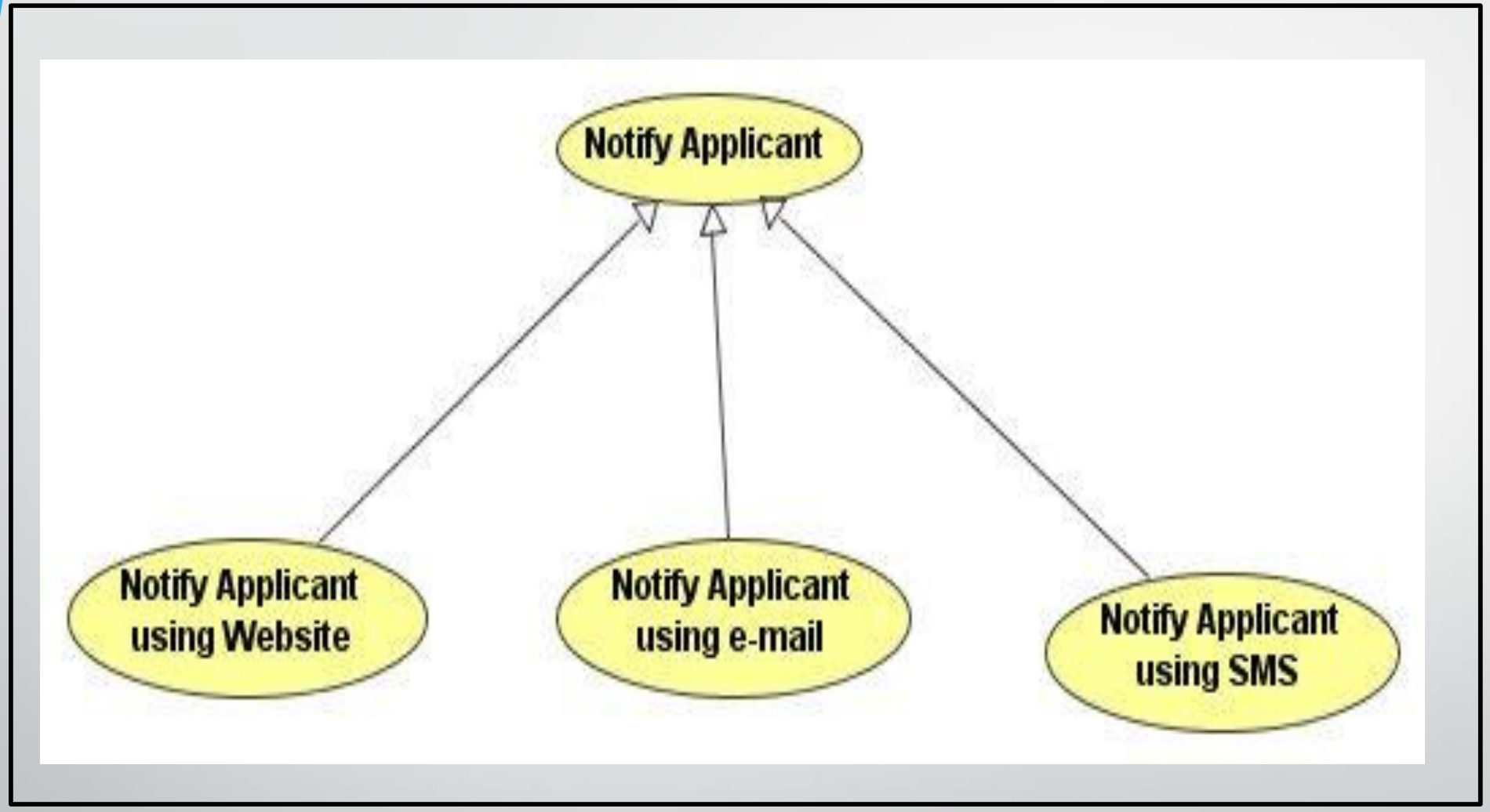

Fig 8. Generalization (Ojo and Estevez, 2005)

Example 1

- Dishi Poa Restaurant has recently commissioned your company to develop a unique restaurant system for them, details as follows:
- The system will service two types of customers: customers who pay by cash and customers who pay using credit or debit cards. It will store information about the customer, namely customer first and last names, age, sex and any known food allergies. It will also store the customers' contact details. Customers can place orders by phone or directly on the system via Internet. If a customer chooses to pay by phone then the system will allow the customer the option to simply place an order or to order takeout food. Placing an order allows the customer to visit the restaurant and give his order number directly to a waiter who will then serve him. If the customer chooses takeout then the system will issue a delivery order to the delivery department who will strive to ensure delivery within the hour. When a cash customer opts for takeout then they can either pay by Mpesa or cash on delivery. The system also provides a frequently asked questions (FAQ) section which will help customers to place orders whenever they require some form of help.
- Design and draw a use case diagram capturing the system above.

Solution

- Identify actors; who can you see?
- Identify use cases; how many use cases can you make out?
- Identify relationships between actors (if any), between actors and use cases (communicates relationship), and between use cases (again if any includes or extends relationships).
- After refinement you should have the following:

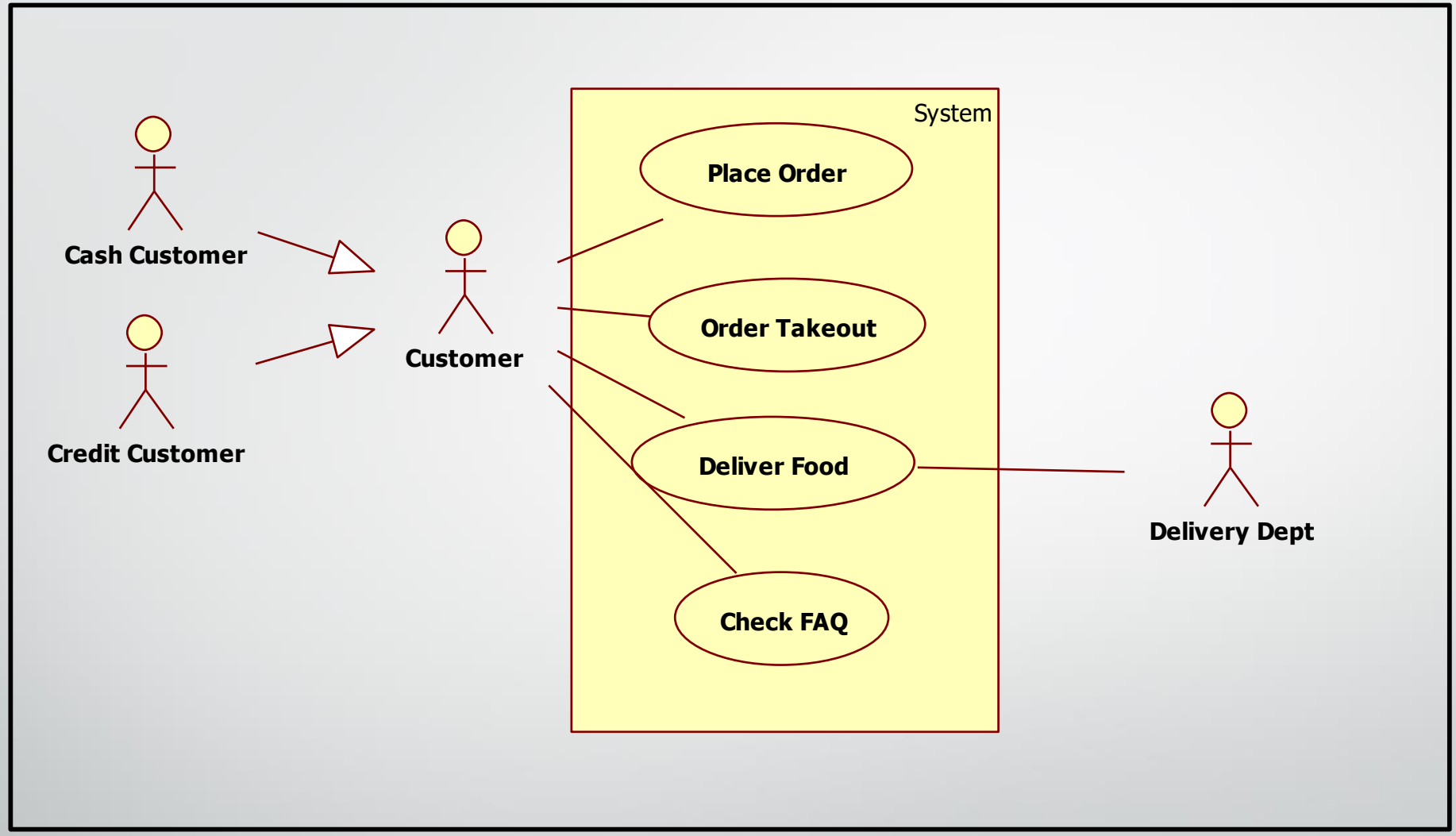

Fig 9. Solution to example 1

Example 2

- A University record system (URS) should keep information about its students and academic staff. Records for all university members are to include their id number, surname, given name, email, address, date of birth, and telephone number. The system should be able to handle the following commands:
  - Add and remove university members (students, and academic staff)
  - Add and Delete subjects
  - Assign and Un-assign subjects to students
  - Assign and Un-assign subjects to academic staff.
- Represent this information in a clearly labeled use case diagram

Summary

- Kruchten (1995) described the architectural views of software development as the 5 views as follows: logical, process, physical, deployment, and use case.
- There is a 4+1 view of the OO model. For each view there are a number of diagrams that denote the respective view of the system's model. The views are process, deployment, logical and dynamic. The +1 view is the use case view.
- The Use case view – this is considered the pivot since it drives the development of all the other views from the requirements phase.
- The components or parts of the use case diagram are: actor, use case, system boundary, relationships (between actors, actors and use cases, and between use cases themselves).

References

- Ojo, A., & Estevez, E. (2005). (rep.). *Object-Oriented Analysis and Design with UML - Training Course* (Vol. 1).
- Various class notes collected since 2012.