Interaction modelling: Use-cases

Anastasija Nikiforova

Institute of Computer Science

Based on materials developed by Marlon Dumas, Fabrizio Maria Maggi, Luciano García-Bañuelos and the book "**Object-Oriented Modeling and Design with UML** "

INFO

- ✓ **Practical session at 18:15 in Zoom**
- ✓ **Practical session #2 mode - ?**
- ✓ **Assignment#1 is announced; deadline for submission – October 15**
- ✓ **NB: class diagram is NOT ER model / database design diagram* *although IF you use it [in the real-world scenario] for DATA modelling, they can be quite similar. More traditional purposes – analysis and design!**

  Should ID be used in a class diagram?
- ✓ **NB: class diagram can be of two types – domain and application. *What is the difference?***
- ✓ **Do not rely on forums! IF you need more info, use more «authoritative» sources such as online courses (preferably by well-known universities). See some useful slinks in the slides**

Useful links

- ✓ https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/
- ✓ https://www.cs.unc.edu/~stotts/COMP145/CRC/class.html
- ✓ https://developer.ibm.com/articles/the-class-diagram/

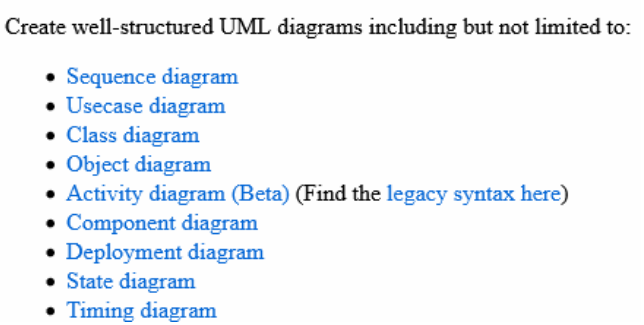
NB: use tree notation for inheritance / generalization (i.e., one incoming «arrow» to abstract class with multiple concrete classes)

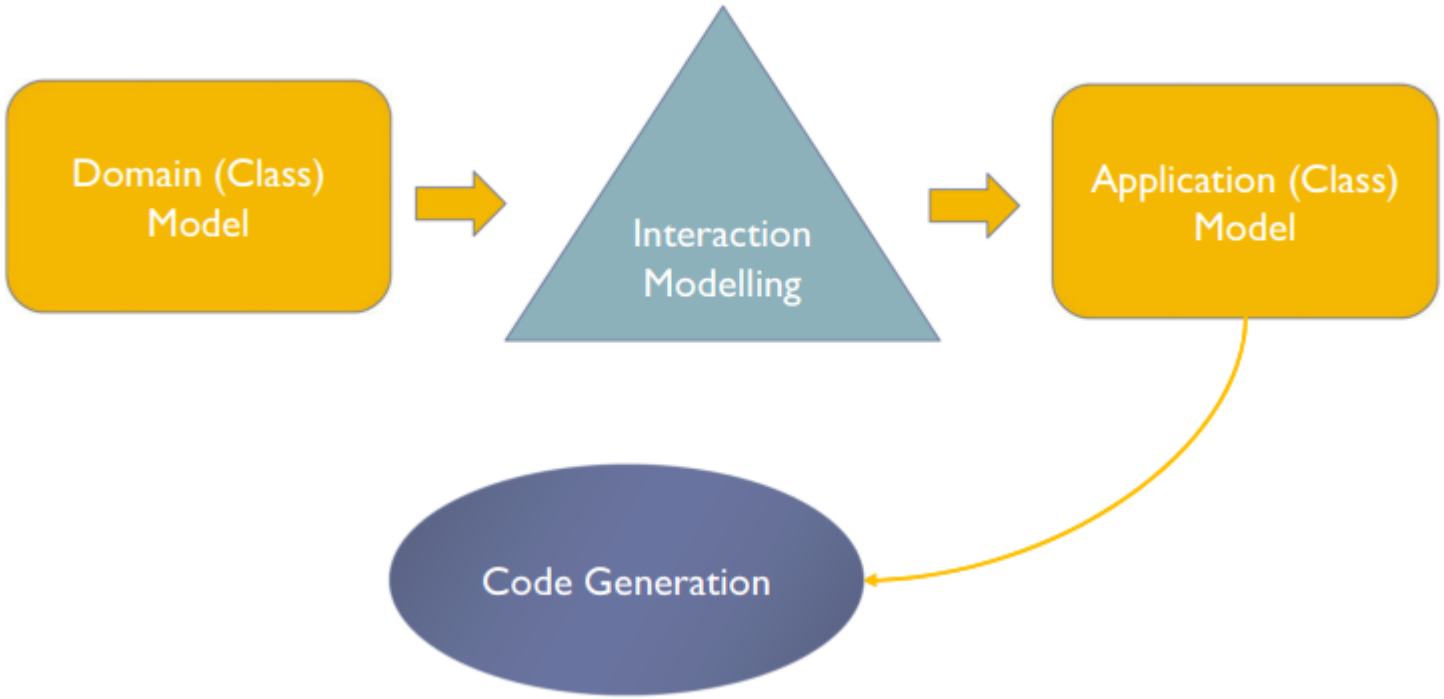
- ✓ https://www.uml-diagrams.org/
- ✓ Fully elaborated ATM example in UML by Russell Bjork

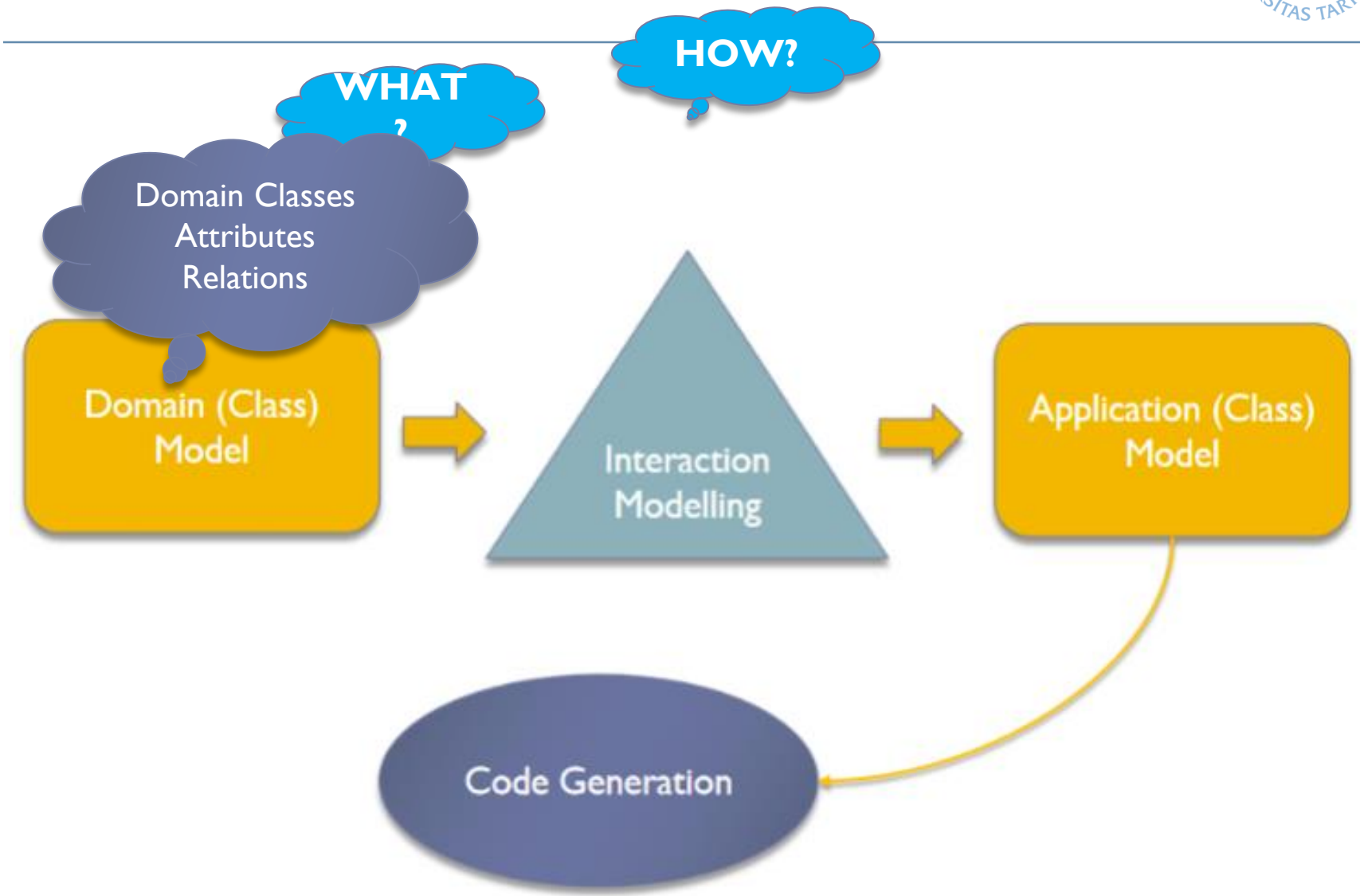
✓ And → Michael Blaha and James Rumbaugh. Object-Oriented Modeling and Design with UML

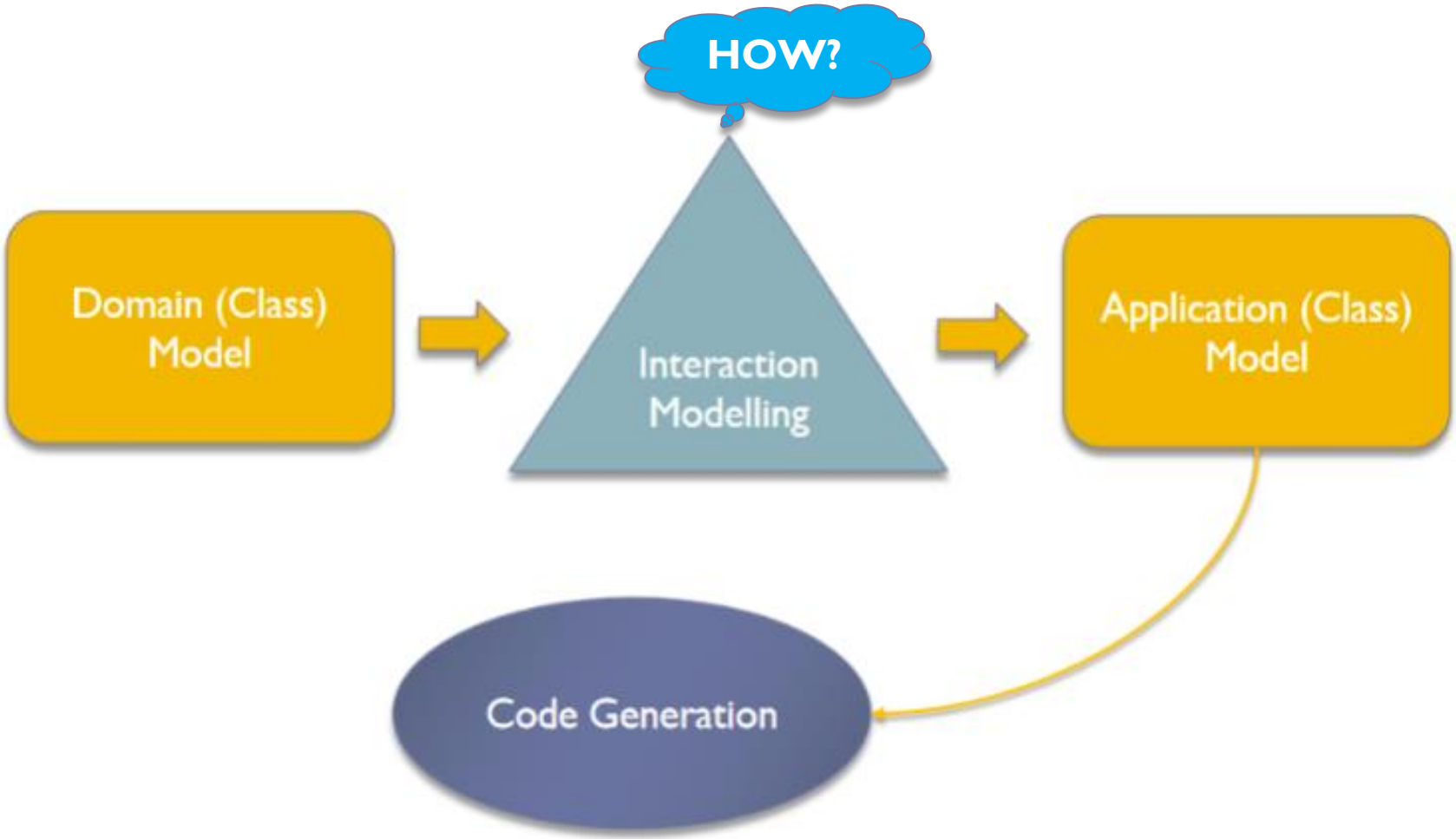
NB: some of you can find interesting

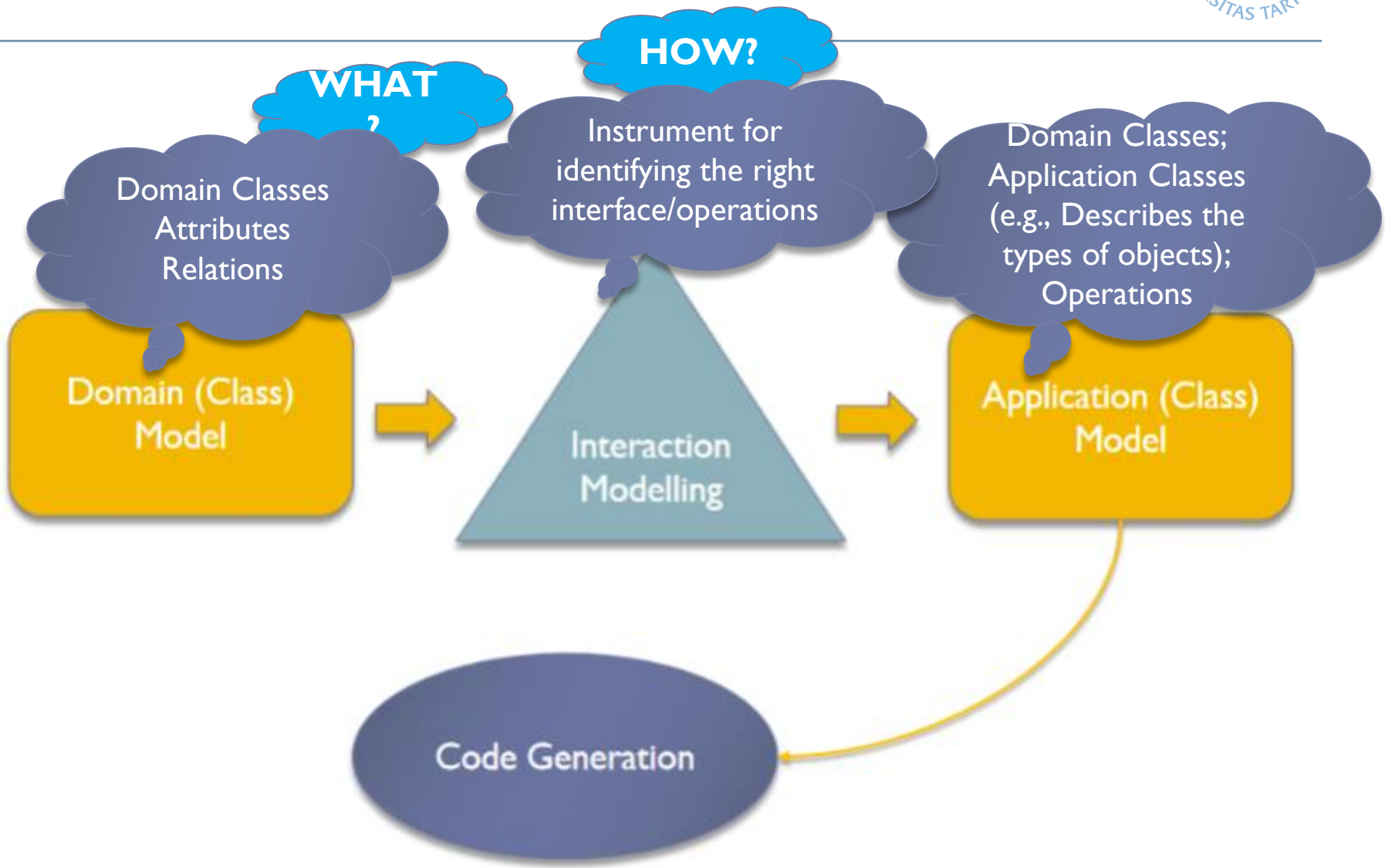
plantUML – «*a versatile component that enables swift and straightforward diagram creation. Users can draft a variety of diagrams using a simple and intuitive language***»**




Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation


HOW?
WHAT
?
Domain Classes
Attributes
Relations
Domain (Class)
Model
Interaction
Modelling
Application (Class)
Model
Code Generation


HOW?
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

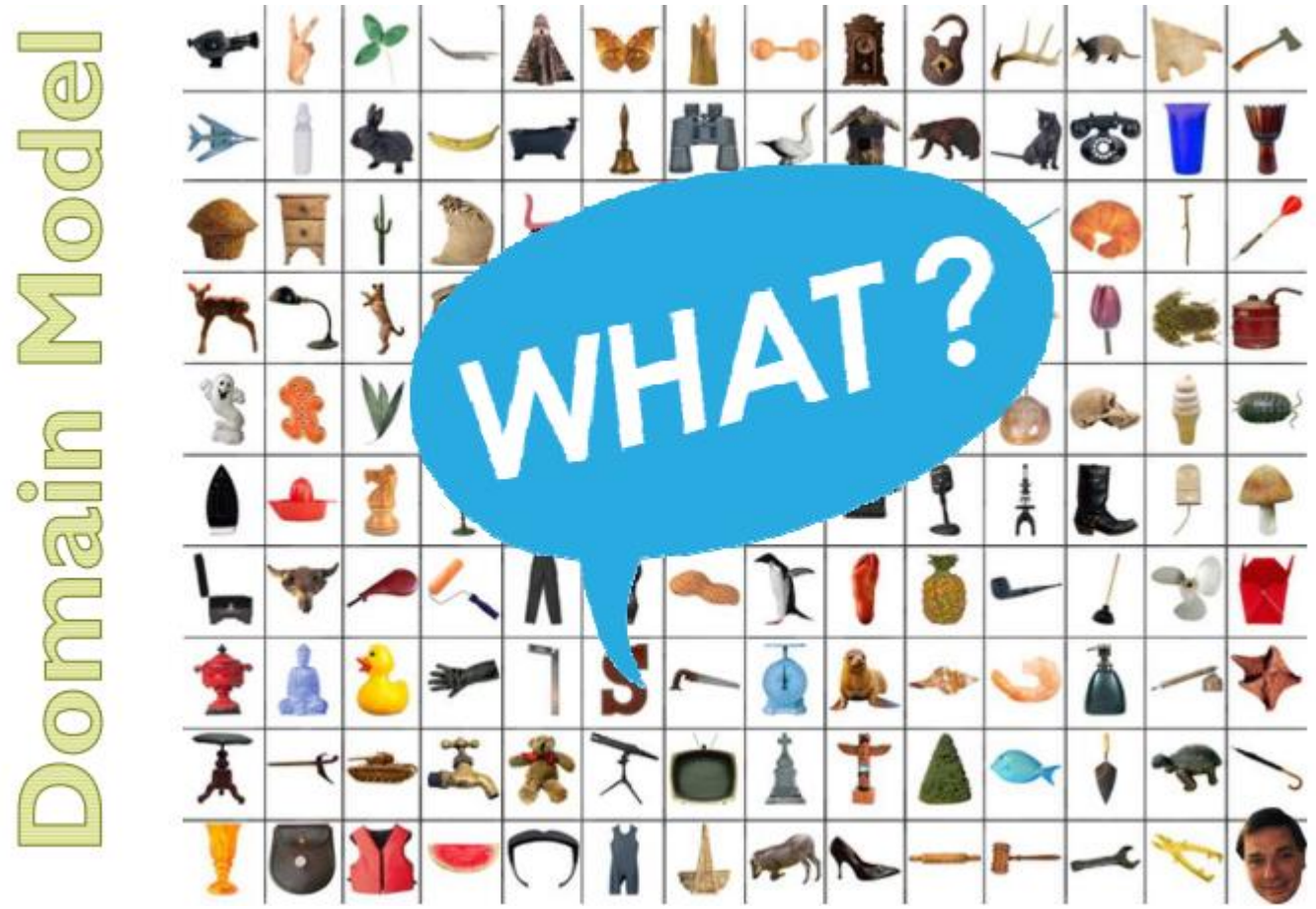

WHAT
?
HOW?
Domain Classes
Attributes
Relations
Instrument for identifying the right interface/operations
Domain Classes; Application Classes (e.g., Describes the types of objects); Operations
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

Interaction modelling

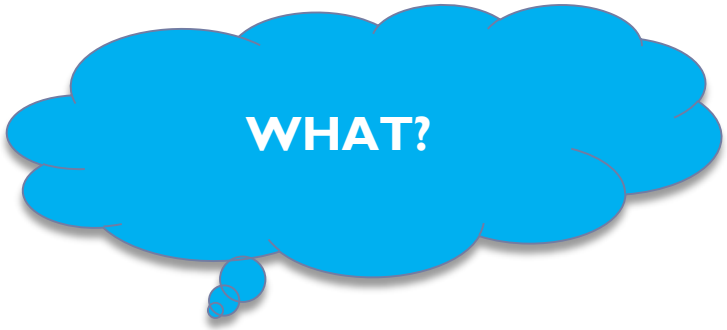
WHAT SERVES AS AN INPUT?

Interaction modelling: input

Interaction modelling: input



Interaction modelling: input



- To answer this question, the domain model provides classes with attributes and relations among them
- Operations are not specified

Interaction modelling: overview

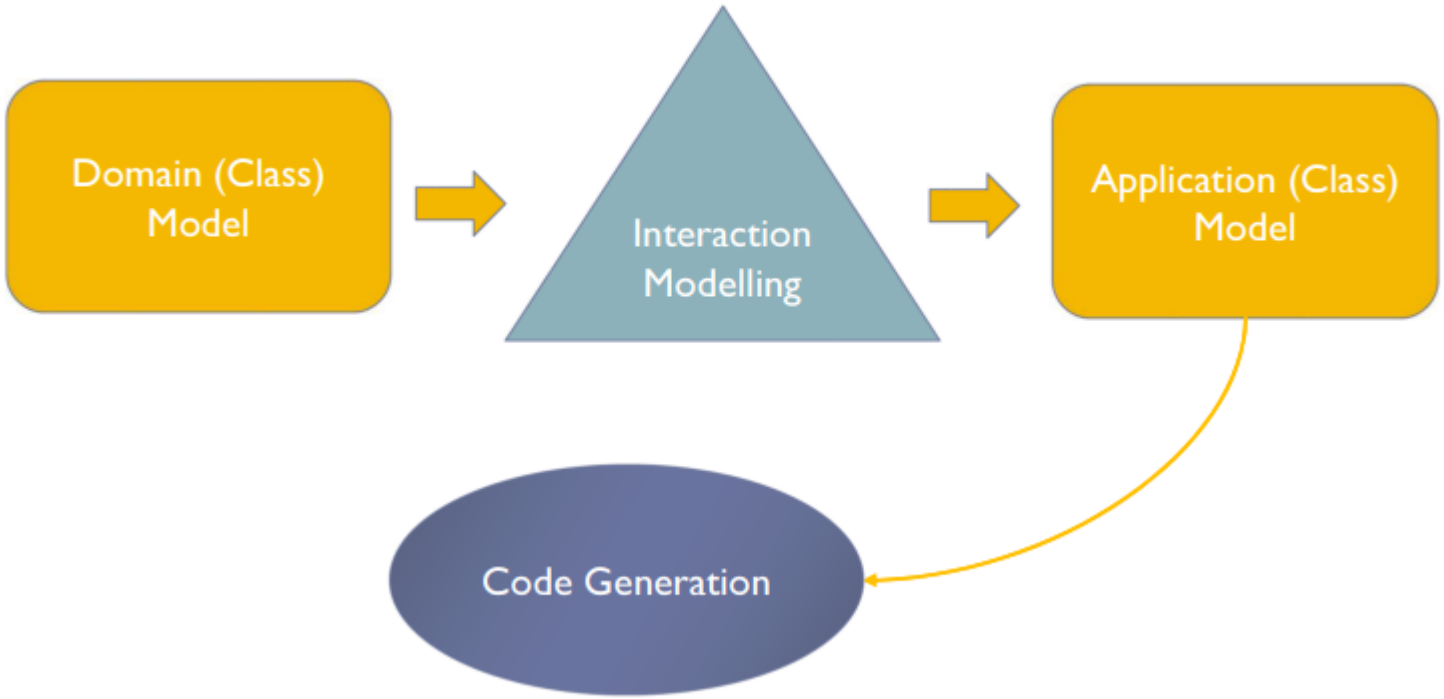
Interaction modelling: overview

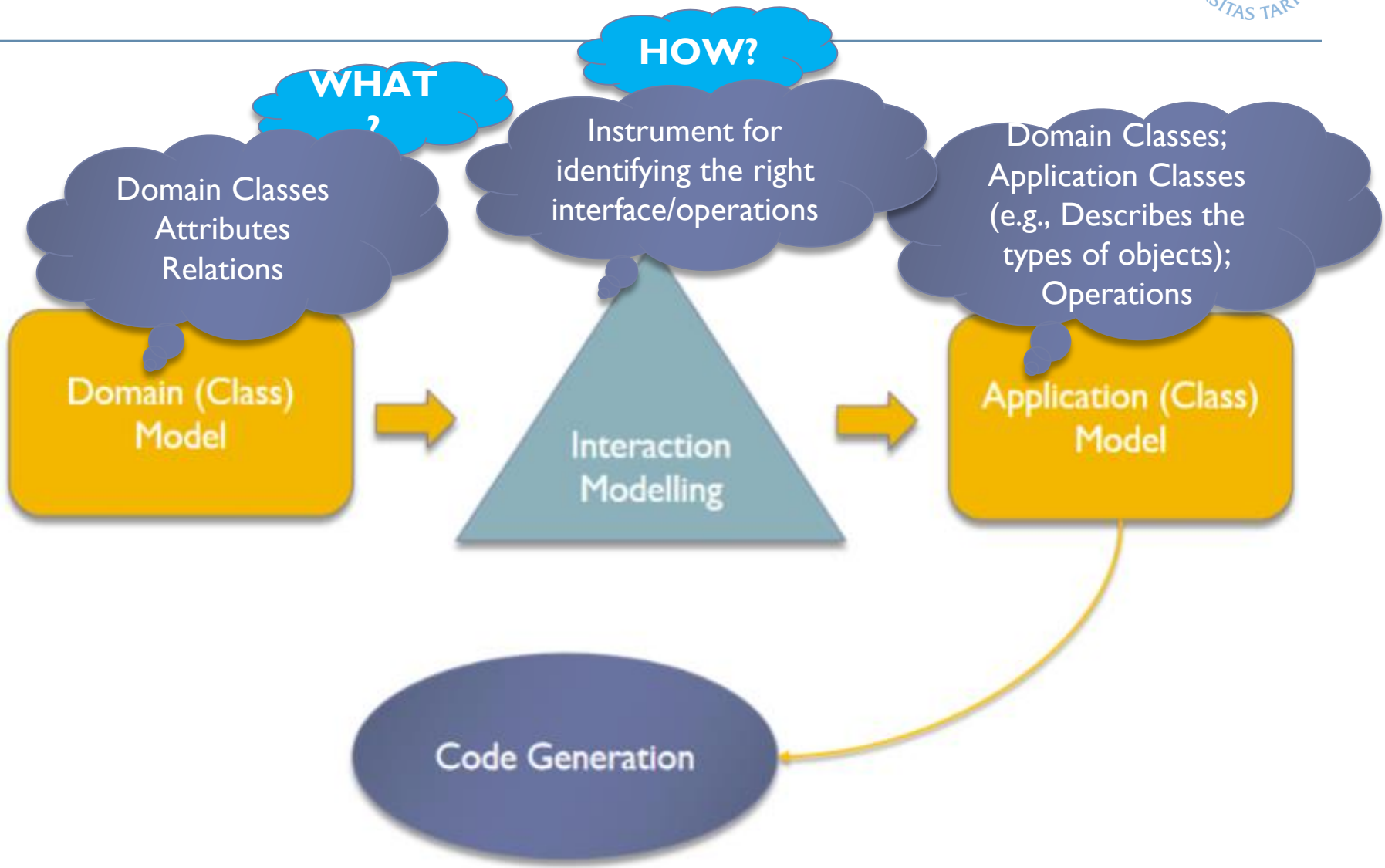
HOW DO OBJECTS INTERACT?

Interaction modelling: output

Interaction modelling: output


HOW?


Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation


WHAT
?
HOW?
Domain Classes
Attributes
Relations
Instrument for identifying the right interface/operations
Domain Classes; Application Classes (e.g., Describes the types of objects); Operations
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

Interaction modelling

- **Interactions** can be modeled **at different levels of abstraction**
  - **At a high level use cases** describe **how a system interacts with outside actors**
    - **Each use case represents a functionality that a system provides to the user**
    - Use cases are **helpful for capturing informal requirements**

- **Sequence diagrams** provide more details about **which operations need to be invoked in a specific scenario**

Interaction modelling: key concepts

- **Use case**
  - **Use case diagram**
  - **Use case description**

- **Scenario**
  - **Sequence diagram (next week)**

Interaction modelling: key concepts

- **Use case**
  - **Use case diagram**
  - **!Use case description!**
- **Scenario**
  - **Sequence diagram (next week)**

Use cases

- A use case is a **contract** of an **interaction between the system and its actor(s) to deliver a logical unit of functionality**

- A use case model usually comprises:
  - **A diagram**, describing **relations between use-cases and actors**.
  - **A textual description of each use case**

Use case in diagram

Use case description

⇒

Name
Actors
Triggers
Preconditions
Post-conditions
Success scenario
Alternative flows

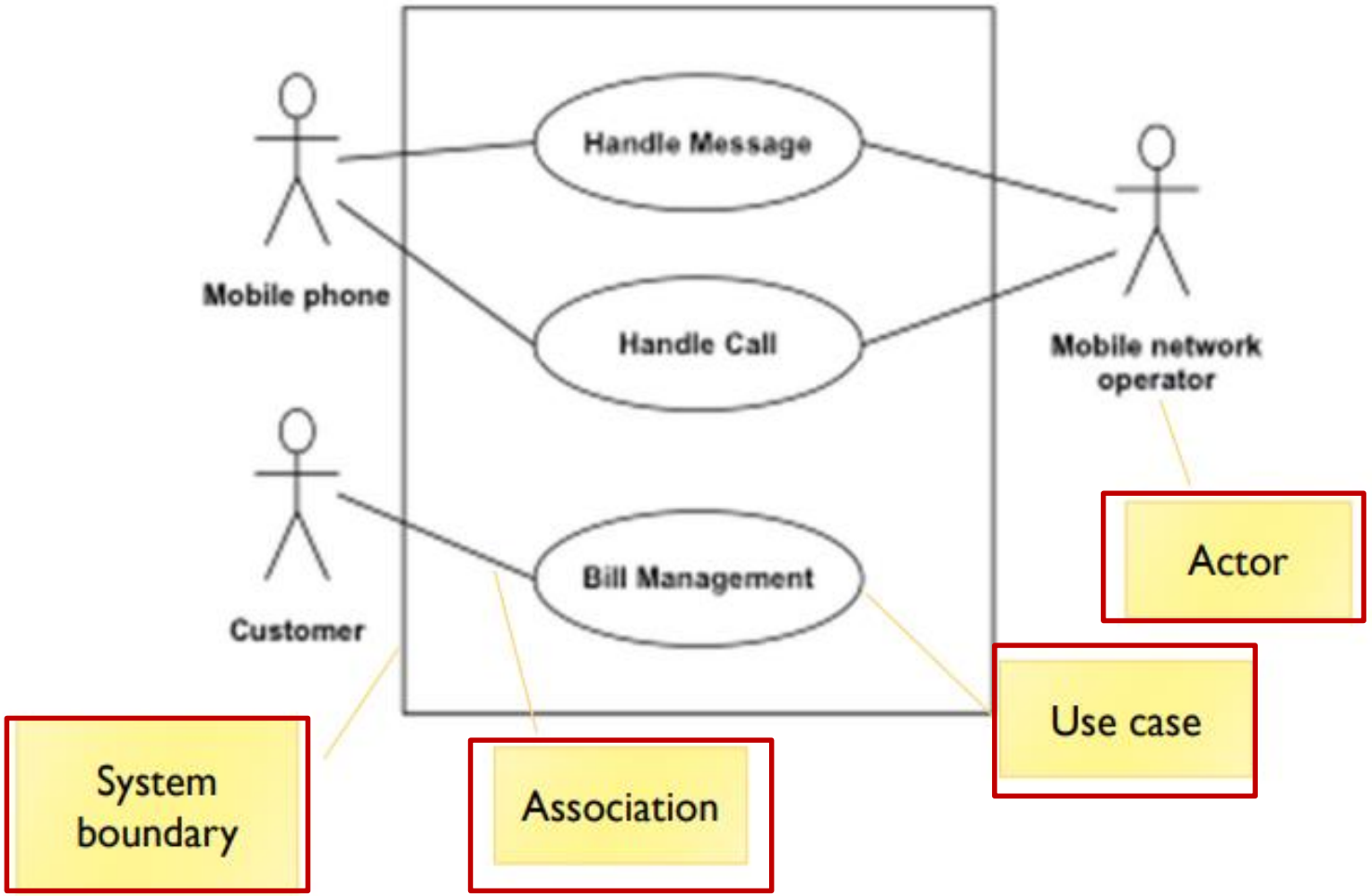
Each ellipse (use case) in the use case diagram is associated to a different description à one table per use case!

Use cases: purpose

- Create a **semi-formal model of the functional requirements**

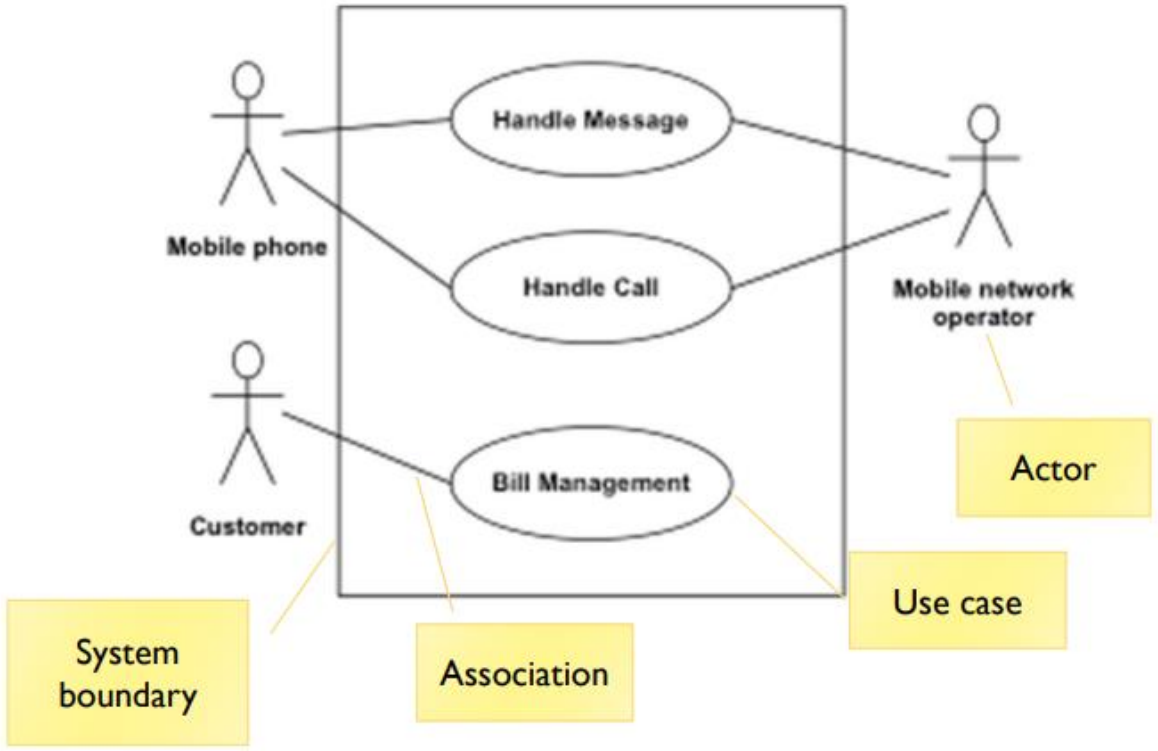
- **Analyze and define:**
  - Scope
  - External interfaces
  - Scenarios and reactions

Use case diagram



min set of elements for use-case diagram

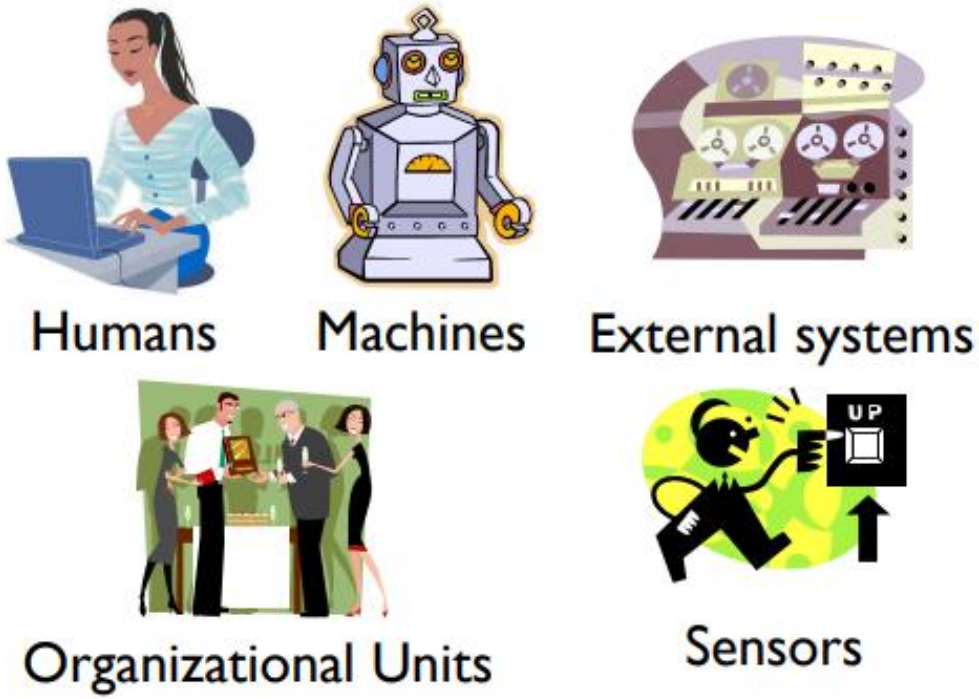
Use case diagram



STEP 1: finding actors

Use case models: finding actors

- **External objects that produce/consume data:**
  - **Must serve as sources and destinations for data**
  - **Must be external to the system**



Use case models: actors

- An **actor** is **a direct external user of a system**
  - **An object or a set of objects that communicates directly with the system but that is not part of the system**

Examples

- *Customer* and *repair technician* are actors of a vending machine
- *Traveler*, *agent* and *airline* are actors of a travel agency
- *User* and *administrator* are actors for a computer database system

Exercise

- **Who are the actors in ÕIS (Study Information System)?**
- **For each actor, name at least three use cases**

Use case models: actors

Actors can be **person, devices and other systems**

Anything that interacts directly with the system is the actor!

Use case models: actors

- **An actor represents a particular facet (i.e., role) of objects in its interaction with a system**

- **The same actor can represent different objects that interact similarly with a system**
  - ***E.g., many individual persons may use a vending machine but their behavior toward the vending machine can be summarized by the actors Customer and Repair Technician***
  - **Each actor represents a coherent set of capabilities for its objects**

Use case models: actors

- Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary
- An actor is directly connected to the system
  - An indirectly connected object is not an actor and should not be included as part of the system model
    - Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine
      - Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model

Use case models: actors

- Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary
- An actor is directly connected to the system
  - An indirectly connected object is not an actor and should not be included as part of the system model
    - Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine
      - Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model

Use case models: actors

A use case is a coherent piece of functionality that a system can provide by interacting with actors

- **Buy a beverage**. The vending machine delivers a beverage after a customer selects and pays for it.
- **Perform scheduled maintenance**. A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition.
- **Make repairs**. A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation.
- **Load items**. A stock clerk adds items into the vending machine to replenish its stock of beverages.

Figure 7.1 Use case summaries for a vending machine. A use case is a coherent piece of functionality that a system can provide by interacting with actors.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Use case models: actors

- **Each use case involves one or more actors as well as the system itself**

  Examples:

  - ***the use case "Buy a beverage" involves the Customer;***
  - ***the use case "Perform scheduled maintenance" involves the Repair Technician;***
  - ***in a telephone system the use case "Make a call" involves two actors, a Caller and a Receiver***

- **An actor is not necessarily a person**
  - ***Example: in an online shop the use case "Checkout" involves the Web Customer and the Credit Payment Service***

Use case models: actors

- **A use case partitions the functionality of the system into a *mainline behavior sequence*, *variations on normal behavior*, *exception conditions*, *error conditions*, *cancellations of a request***
- **Use cases should all be at a comparable level of abstraction**

  Examples:

  - ***"Make telephone call" and "Record voice mail message" are at a comparable level;***
  - ***"Set external speaker volume to high" is too narrow, "Set speaker volume" or even "Set telephone parameters" would be better***

Creating Use case models

- **Each use case involves one or more actors as well as the system itself**

  Examples:

  - *the use case "Buy a beverage" involves the Customer;*
  - *the use case "Perform scheduled maintenance" involves the Repair Technician;*
  - *in a telephone system the use case "Make a call" involves two actors, a Caller and a Receiver*

- **An actor is not necessarily a person**
  - *Example: in an online shop the use case "Checkout" involves the Web Customer and the Credit Payment Service*

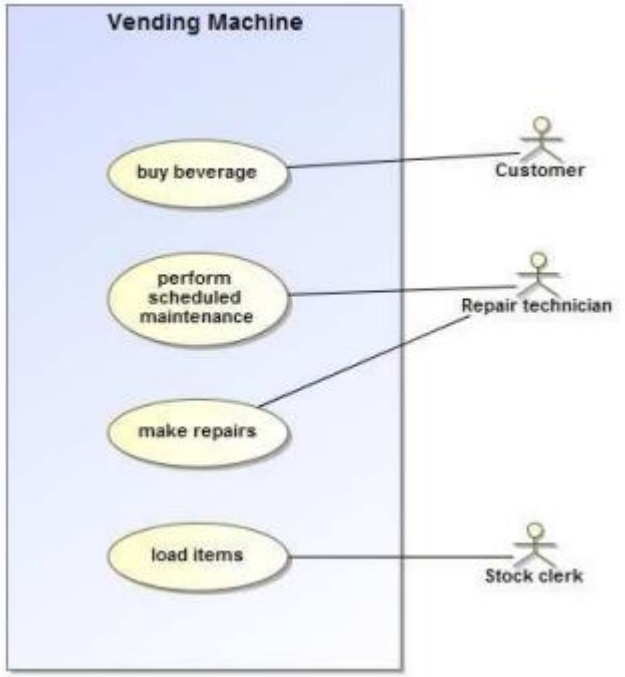
- **A use case partitions the functionality of the system into a mainline behavior sequence, variations on normal behavior, exception conditions, error conditions, cancellations of a request**
- **Use cases should all be at a comparable level of abstraction**

  Examples:

  - *"Make telephone call" and "Record voice mail message" are at a comparable level;*
  - *"Set external speaker volume to high" is too narrow, "Set speaker volume" or even "Set telephone parameters" would be better*

Use case diagrams

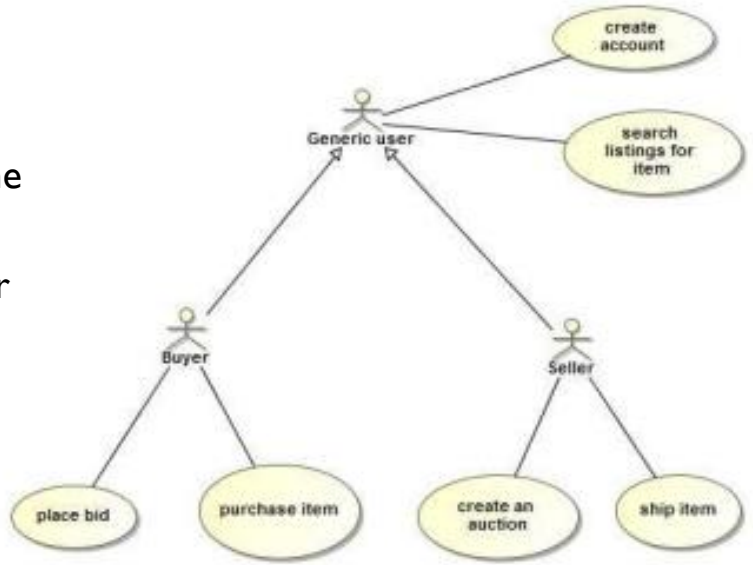
- UML has a graphical notation for summarizing use cases into use case diagrams
- A rectangle contains the use cases for a system with the actors listed on the outside
- The name of the system is written near a side of rectangle
- A name within an ellipse denotes a use case
- A "stick man" icon denotes an actor with the name placed below the icon
- Solid lines connect use cases to participating actors



Actor generalization

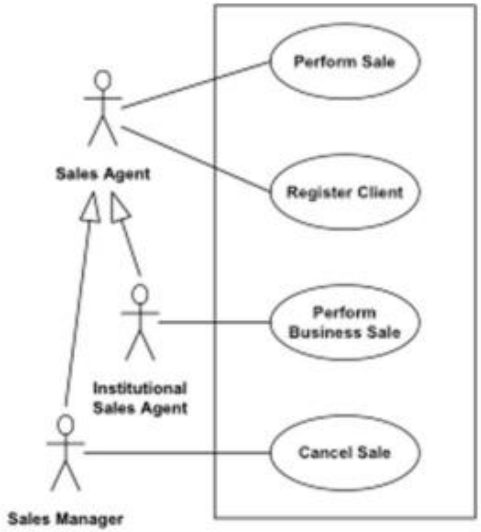
Generalization works in use case diagrams as well!

- The child actor inherits all use case associations from the parent
- Actor generalization should be used of the specific actor has more responsibility than the generalized one (i.e., associated with more use cases)
  - Example: *requirements management use case diagram*
  - duplicate behavior in both the buyer and seller, which includes "create an account" and "Search listings"
  - **Rather than having duplication**, use a **more general user** that has this behavior and then **the actors will "inherit" this behavior from the general user**



*Actor generalization should be used i**f the specific actor has more responsibility than the generalized one** (i.e. associated with more Use Cases)*

Actor generalization



*Actor generalization should be used i**f the specific actor has more responsibility than the generalized one** (i.e. associated with more Use Cases)*

Use case relationships -> linking use cases

- **For large applications complex use-cases can be built from smaller pieces!!!**

- **Linking enables flexibility in requirements specification**
  - Isolating functionality
  - Enabling functionality sharing
  - Breaking functionality into manageable chunks

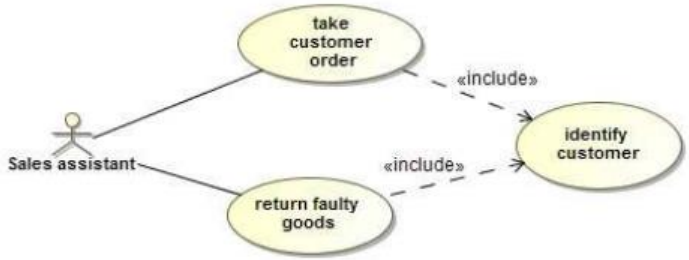
- Three linking mechanism are available in UML:
  - *Include*
  - *Extend*
  - *Generalization*

- And one grouping mechanism:
  - *Package*

Include

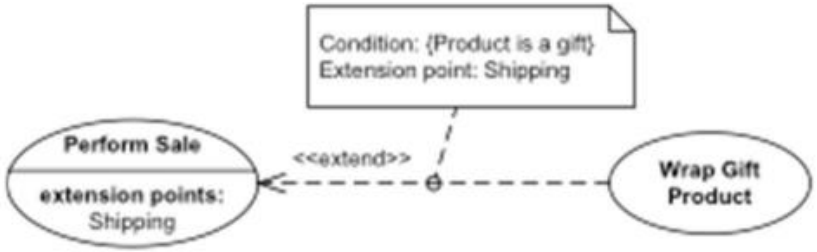
- **A use case can make use of another smaller use case**

- **Include** is used when:
  - **Decomposing complicated behavior**
  - **Centralizing common behavior // incorporates the behavior of another use case (e.g., subroutines)**
- Factoring a use case into pieces is appropriate **when the pieces represent significant behavior units**
- **The base use case explicitly incorporates the behavior of another use case at a location specified in the base**



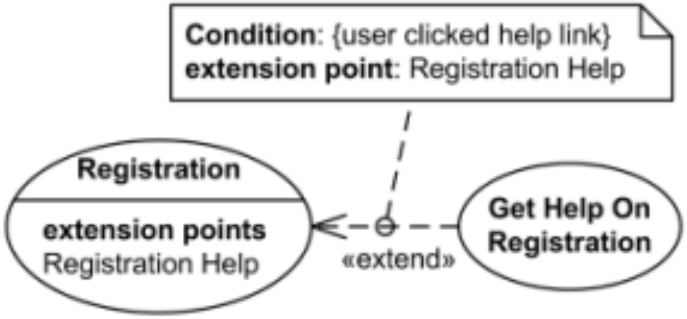
Extends

- **The base use case can incorporate another use case at certain points, called extension points**
- Adds an "extra behaviour" to a base use cases used **in the situation in which some initial capability is defined and later features are added modularly**
- **Base use case is meaningful on its own, it is independent of the extension**.
- Extension **typically defines optional behavior that is not necessarily meaningful by itself**
- Note the direction of the arrow
  - **The base use-case does not know which use-case extends it**



Extends

- Use case “Registration” is meaningful on its own.
- It could be optionally extended with “get Help On registration”



- **Extension points:** specify the location at which the behavior of the base use case may be extended.
- Extension points can have a condition attached.
- The extension behavior occurs only if the condition is true when the control reaches the extension point.