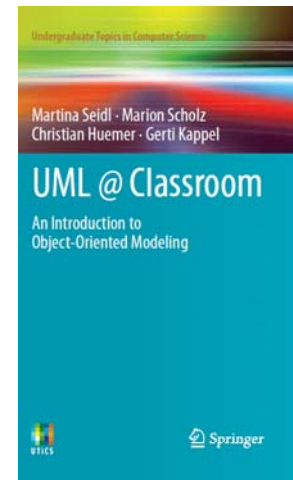


Object-Oriented Modeling

Use Case Diagram

Slides accompanying UML@Classroom
Version 1.0



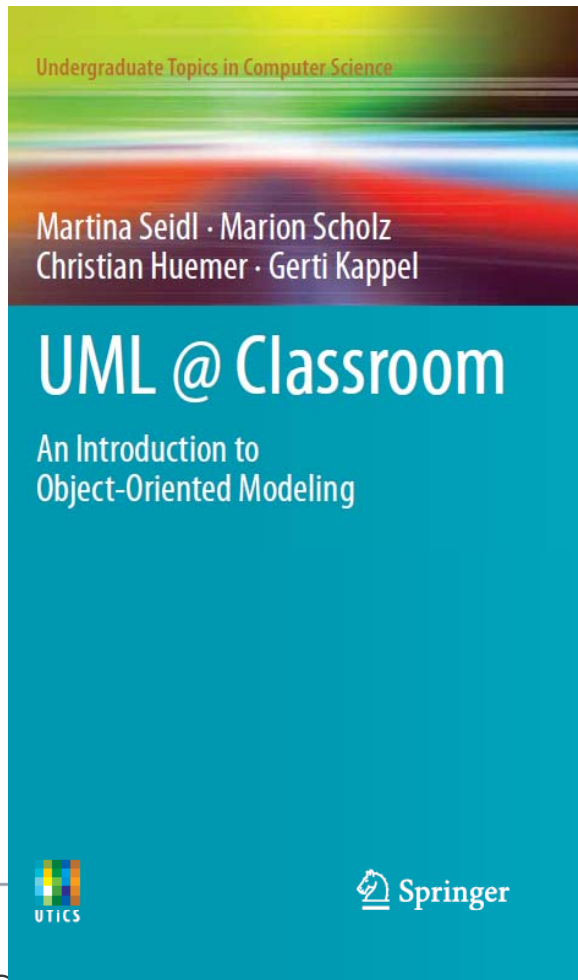
Business Informatics Group

Institute of Software Technology and Interactive Systems
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at

Literature

- The lecture is based on the following book:



UML @ Classroom: An Introduction to Object-Oriented Modeling

Martina Seidl, Marion Scholz, Christian Huemer
and Gerti Kappel

Springer Publishing, 2015

ISBN 3319127411

- **Use Case Diagram**
- Structure Modeling
- State Machine Diagram
- Sequence Diagram
- Activity Diagram

Content

- Introduction
- Use cases
- Actors
- Relationships between use cases and actors
- Relationships between use cases
- Relationships between actors
- Description of use cases
- Best practices
- Typical errors
- Notation elements



Introduction

- The use case is a fundamental concept of many object-oriented development methods.
- Use case diagrams express the expectations of the customers/stakeholders
 - essential for a detailed design
- The use case diagram is used during the entire analysis and design process.
- We can use a use case diagram to answer the following questions:
 - What is being described? (The system.)
 - Who interacts with the system? (The actors.)
 - What can the actors do? (The use cases.)



Example: Student Administration System

- **System**

(what is being described?)

- Student administration system

- **Actors**

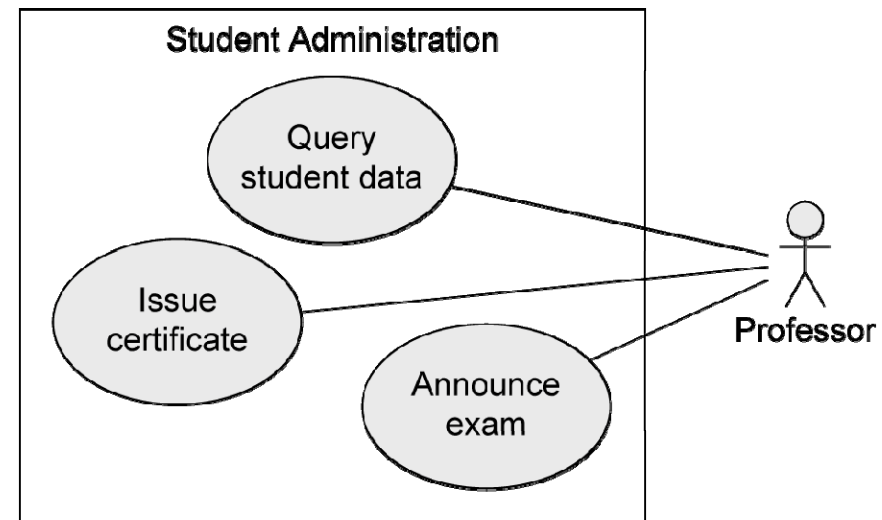
(who interacts with the system?)

- Professor

- **Use cases**

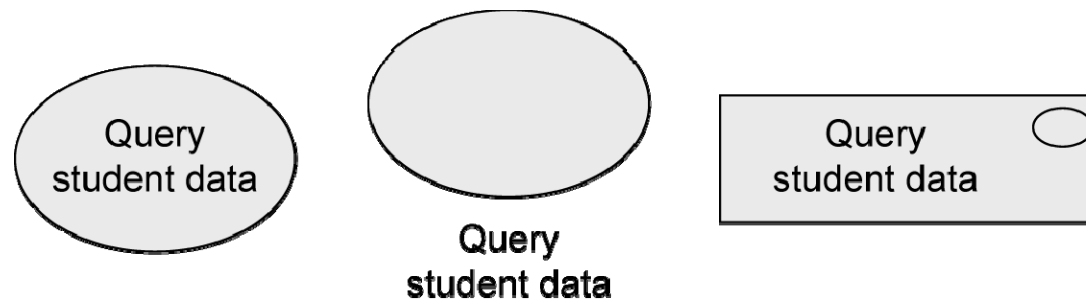
(what can the actors do?)

- Query student data
- Issue certificate
- Announce exam



Use Case

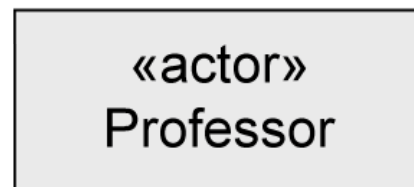
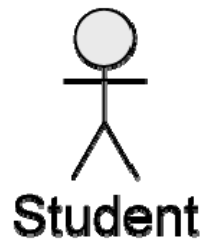
- Describes functionality expected from the system under development.
- Provides tangible benefit for one or more actors that communicate with this use case.
- Derived from collected customer wishes.
- Set of all use cases describes the functionality that a system shall provide.
 - Documents the functionality that a system offers.
- Alternative notations:





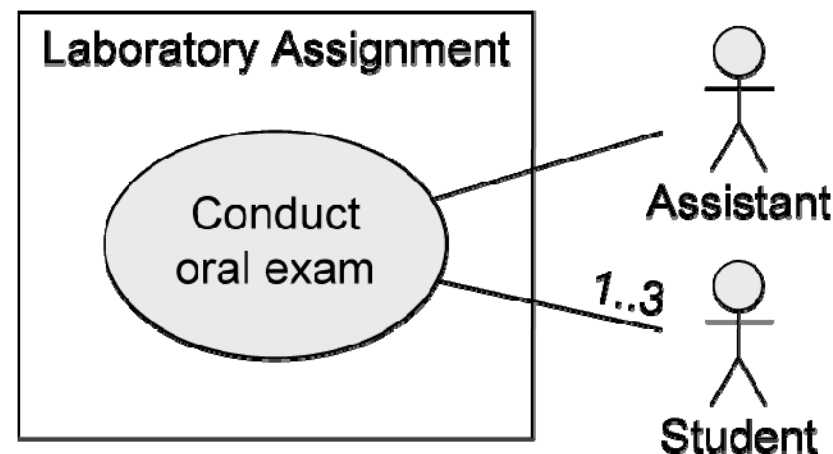
Actor (1/3)

- Actors interact with the system ...
 - by **using** use cases, i.e., the actors initiate the execution of use cases.
 - by **being used** by use cases, i.e., the actors provide functionality for the execution of use cases.
- Actors represent roles that users adopt.
 - Specific users can adopt and set aside multiple roles simultaneously.
- Actors are not part of the system, i.e., they are outside of the system boundaries.
- Alternative notations:



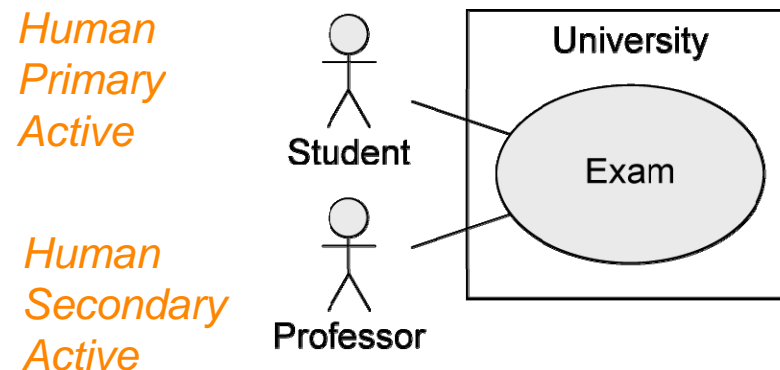
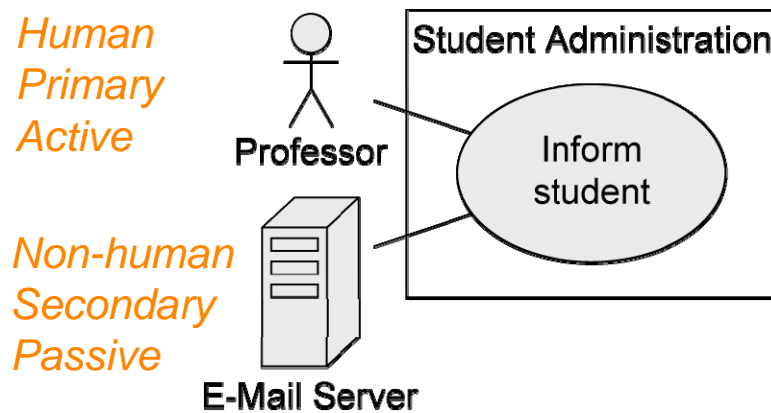
Actor (2/3)

- Usually user data is also administered within the system. This data is modeled within the system in the form of objects and classes.
- Example: actor **Assistant**
 - The actor **Assistant** interacts with the system **Laboratory Assignment** by using it.
 - The class **Assistant** describes objects representing user data (e.g., name, ssNr, ...).



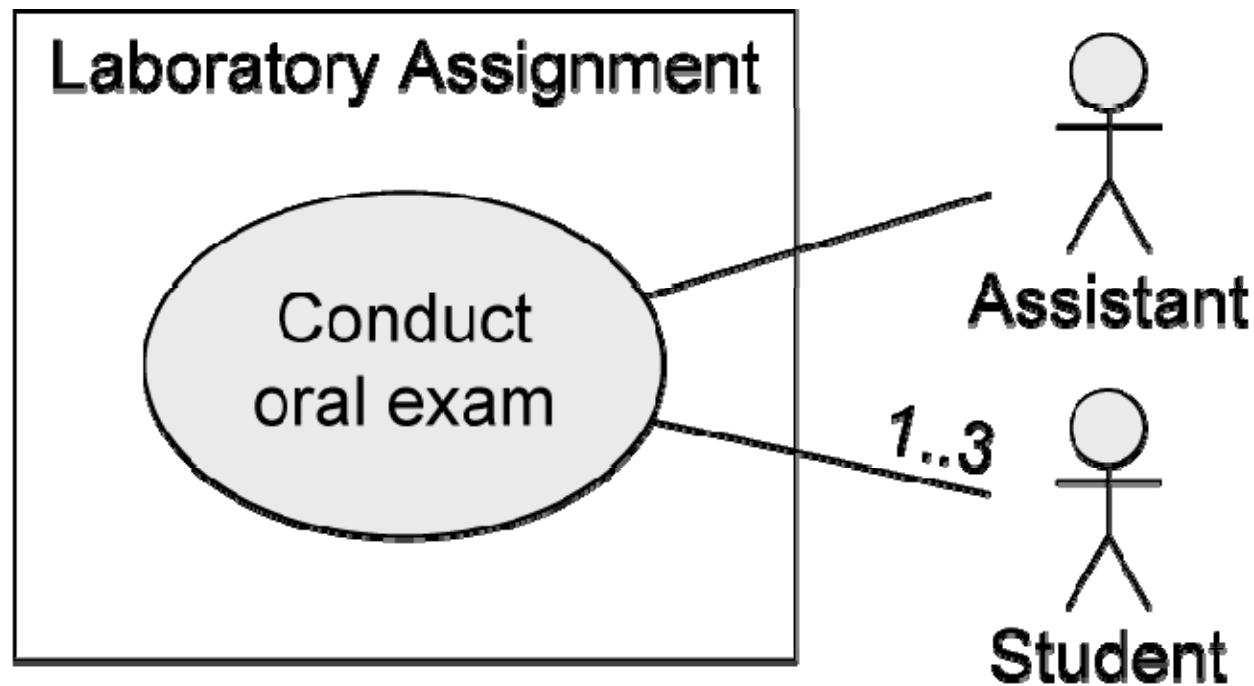
Actor (3/3)

- **Human**
 - E.g., Student, Professor
 - **Non-human**
 - E.g., E-Mail Server
 - **Primary:** has the main benefit of the execution of the use case
 - **Secondary:** receives no direct benefit
 - **Active:** initiates the execution of the use case
 - **Passive:** provides functionality for the execution of the use case
-
- **Example:**



Relationships between Use Cases and Actors

- Actors are connected with use cases via solid lines (*associations*).
- Every actor must communicate with at least one use case.
- An association is always binary.
- Multiplicities may be specified.

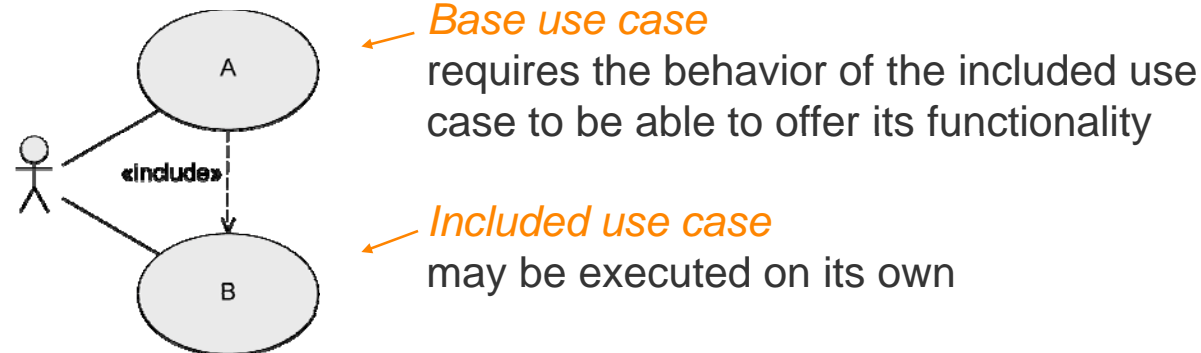


Relationships between Use Cases

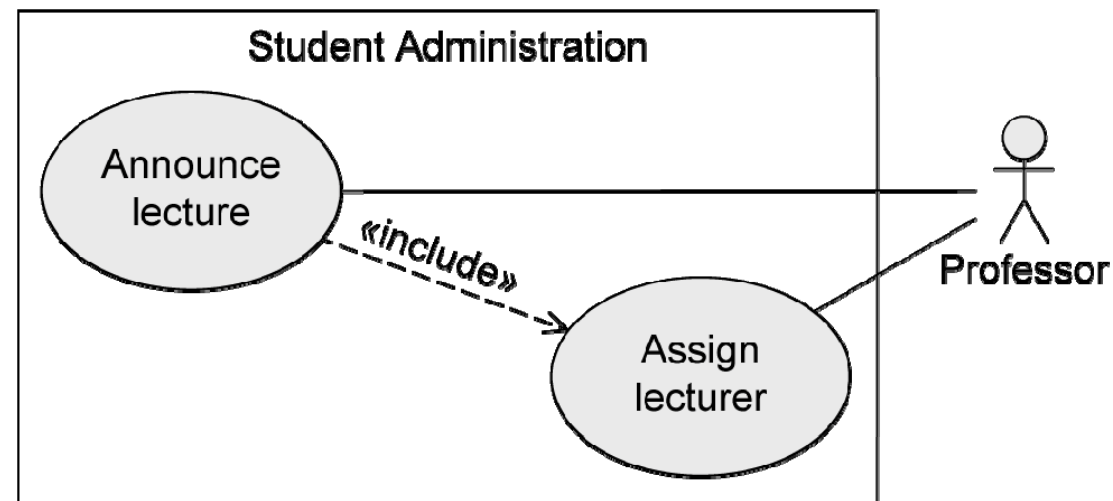
«include» - Relationship



- The behavior of one use case (included use case) is integrated in the behavior of another use case (base use case)



- Example:

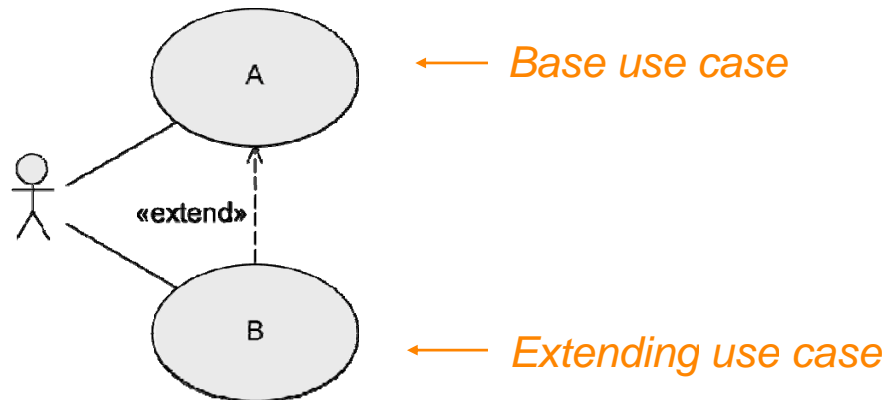


Relationships between Use Cases

«extend» - Relationship



- The behavior of one use case (extending use case) may be integrated in the behavior of another use case (base use case) but does not have to.
- Both use cases may also be executed independently of each other.



- A decides if B is executed.
- Extension points define at which point the behavior is integrated.
- Conditions define under which circumstances the behavior is integrated.

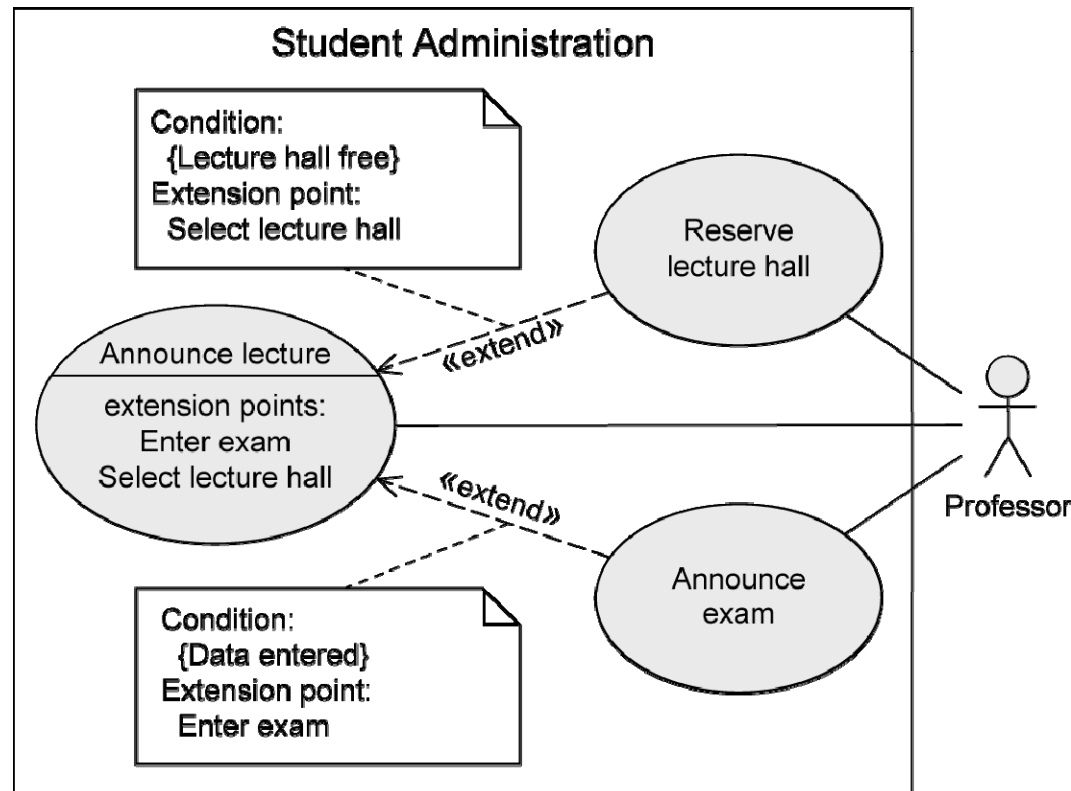
Relationships between Use Cases

«extend» - Relationship: Extension Points



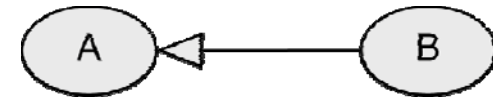
- Extension points are written directly within the use case.
- Specification of multiple extension points is possible.

Example:

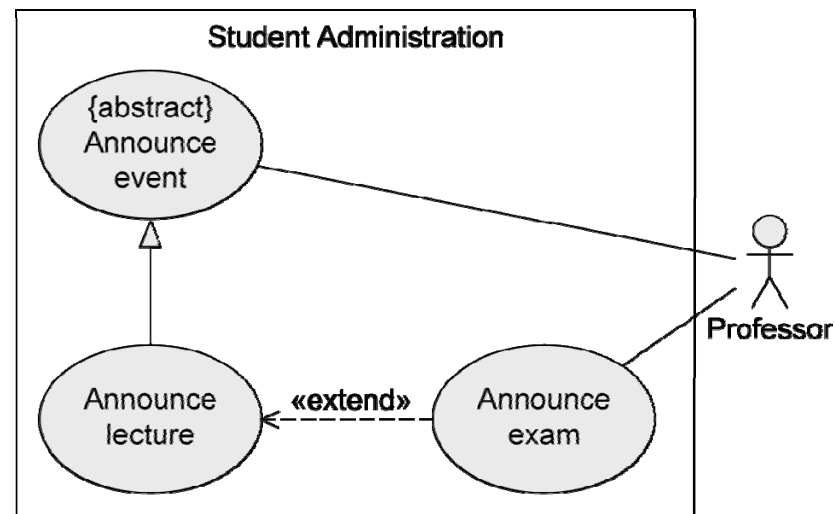
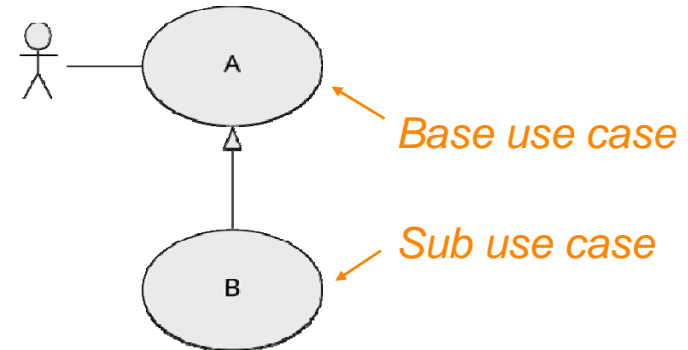


Relationships between Use Cases

Generalization of Use Cases

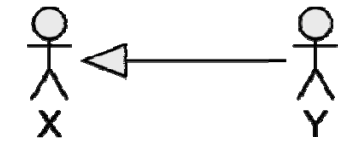


- Use case **A** generalizes use case **B**.
- **B** inherits the behavior of **A** and may either extend or overwrite it.
- **B** also inherits all relationships from **A**.
- **B** adopts the basic functionality of **A** but decides itself what part of **A** is executed or changed.
- **A** may be labeled **{abstract}**
 - Cannot be executed directly
 - Only **B** is executable
- Example:

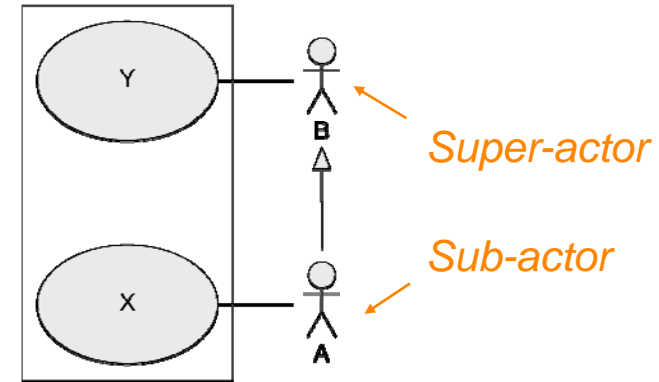


Relationships between Actors

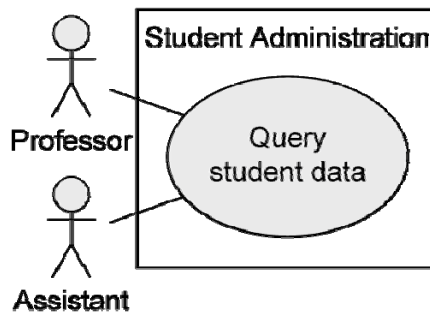
Generalization of Actors



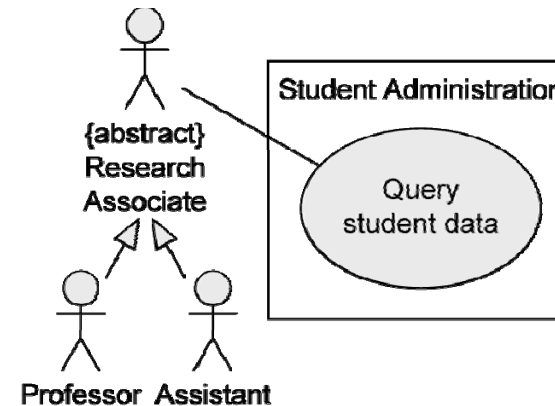
- Actor **A** inherits from actor **B**.
- **A** can communicate with **X** and **Y**.
- **B** can only communicate with **Y**.
- *Multiple inheritance* is permitted.
- *Abstract* actors are possible.



- Example:



≠



Professor AND Assistant needed for executing Query student data

Professor OR Assistant needed for executing Query student data



Description of Use Cases

- Structured approach
 - Name
 - Short description
 - Precondition: prerequisite for successful execution
 - Postcondition: system state after successful execution
 - Error situations: errors relevant to the problem domain
 - System state on the occurrence of an error
 - Actors that communicate with the use case
 - Trigger: events which initiate/start the use case
 - Standard process: individual steps to be taken
 - Alternative processes: deviations from the standard process

[A. Cockburn: Writing Effective Use Cases, Addison Wesley, 2000]



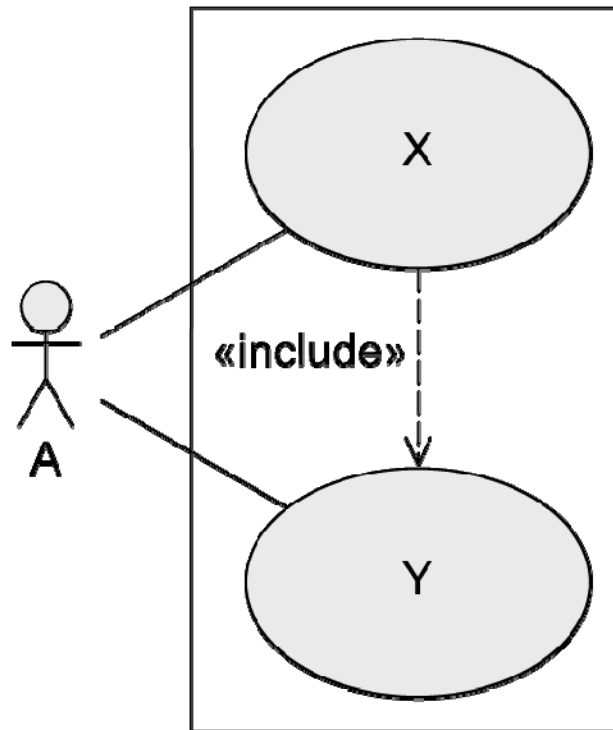
Description of Use Cases - Example

- Name: **Reserve lecture hall**
- Short description: An employee reserves a lecture hall at the university for an event.
- Precondition: The employee is authorized to reserve lecture halls.
- Postcondition: A lecture hall is reserved.
- Error situations: There is no free lecture hall.
- System state in the event of an error: The employee has not reserved a lecture hall.
- Actors: **Employee**
- Trigger: Employee requires a lecture hall.
- Standard process: (1) Employee logs in to the system.
(2) Employee selects the lecture hall.
(3) Employee selects the date.
(4) System confirms that the lecture hall is free.
(5) Employee confirms the reservation.
- Alternative processes: (4') Lecture hall is not free.
(5') System proposes an alternative lecture hall.
(6') Employee selects alternative lecture hall and confirms the reservation.

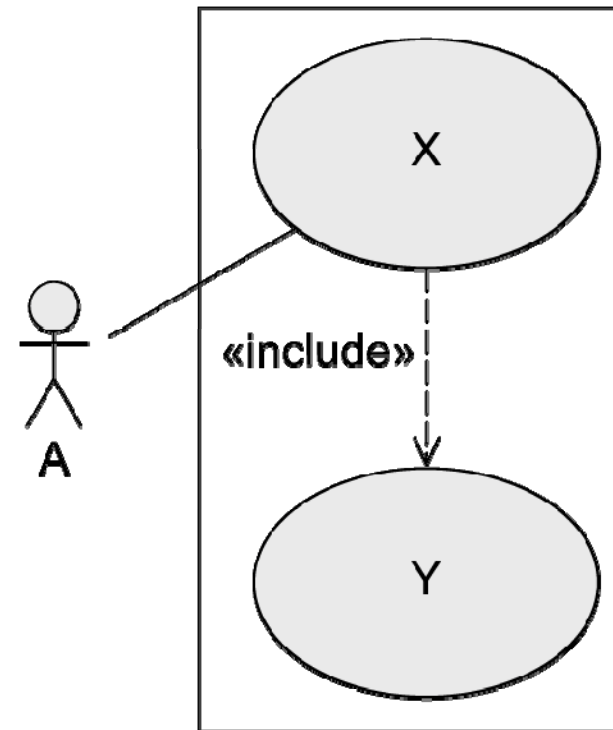
Best Practices

«include»

UML standard



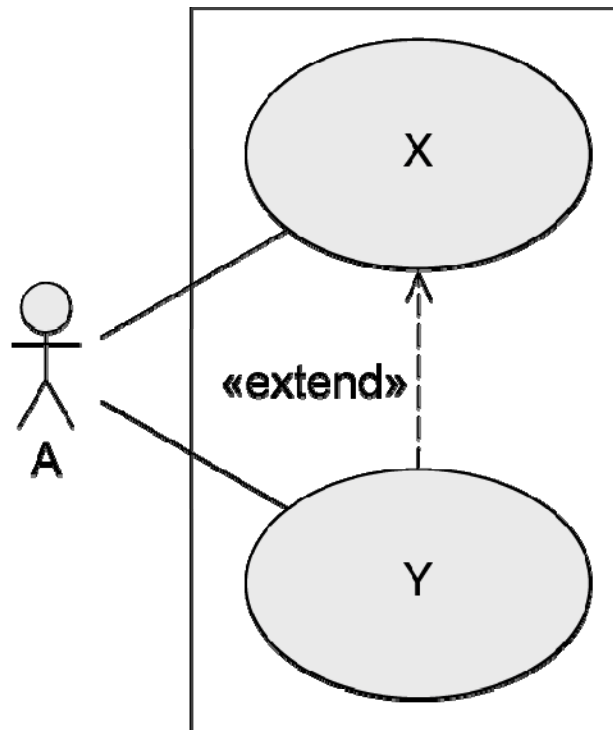
Best practice



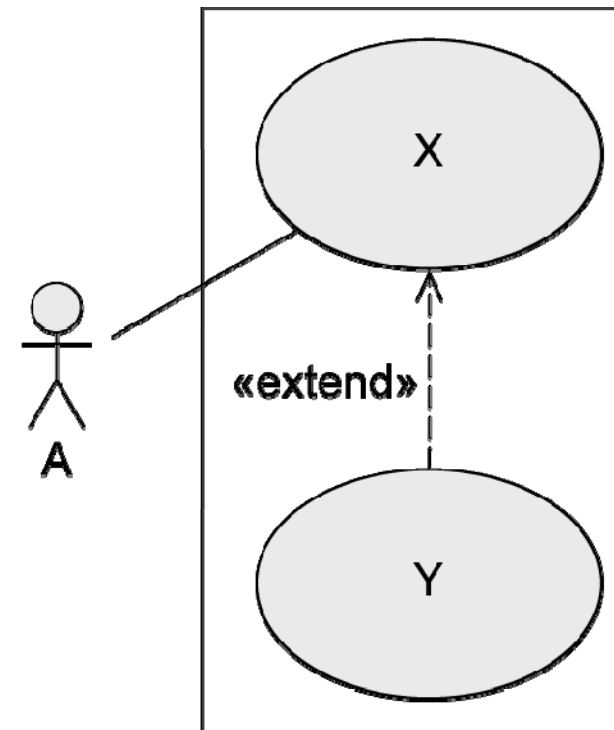
Best Practices

«extend»

UML standard



Best practice



Best Practices

Identifying Actors

- Who uses the main use cases?
- Who needs support for their daily work?
- Who is responsible for system administration?
- What are the external devices/(software) systems with which the system must communicate?
- Who is interested in the results of the system?

Best Practices

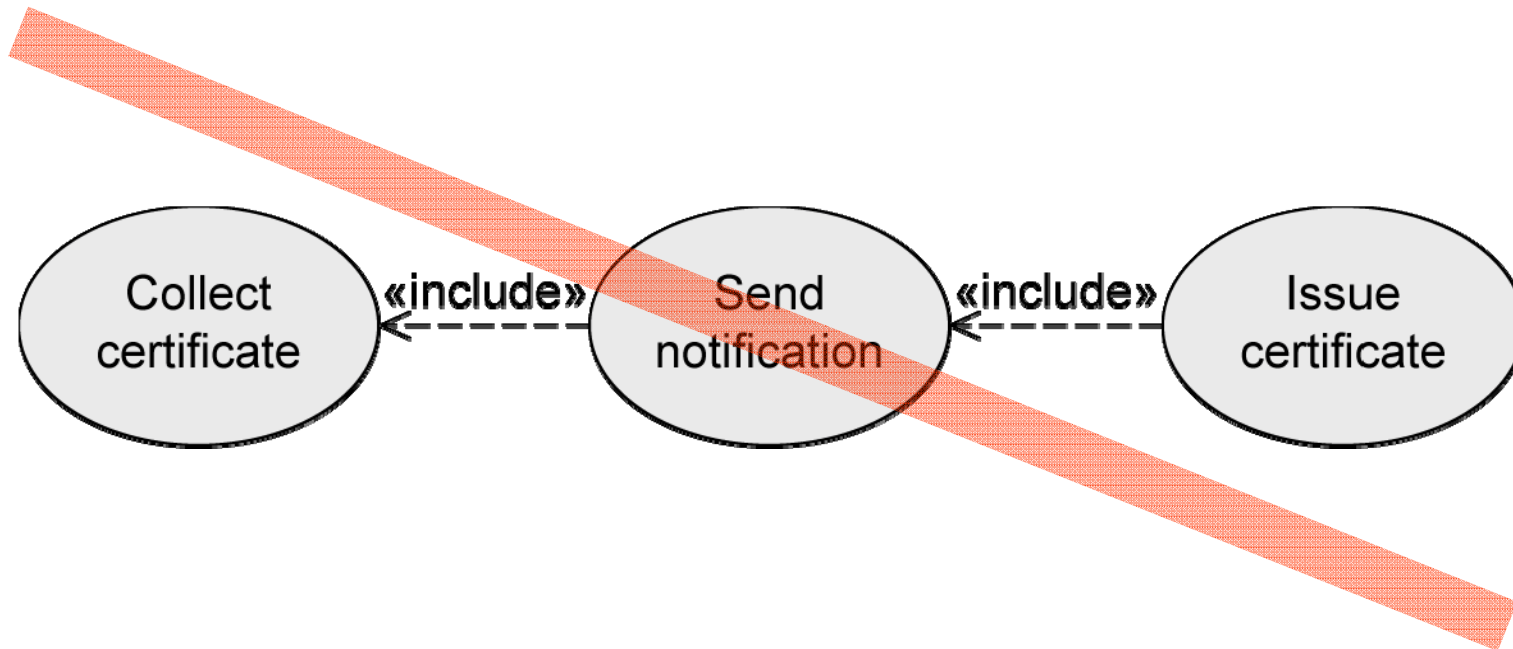
Identifying Use Cases

- What are the main tasks that an actor must perform?
- Does an actor want to query or even modify information contained in the system?
- Does an actor want to inform the system about changes in other systems?
- Should an actor be informed about unexpected events within the system?

Best Practices

Typical Errors To Avoid (1/5)

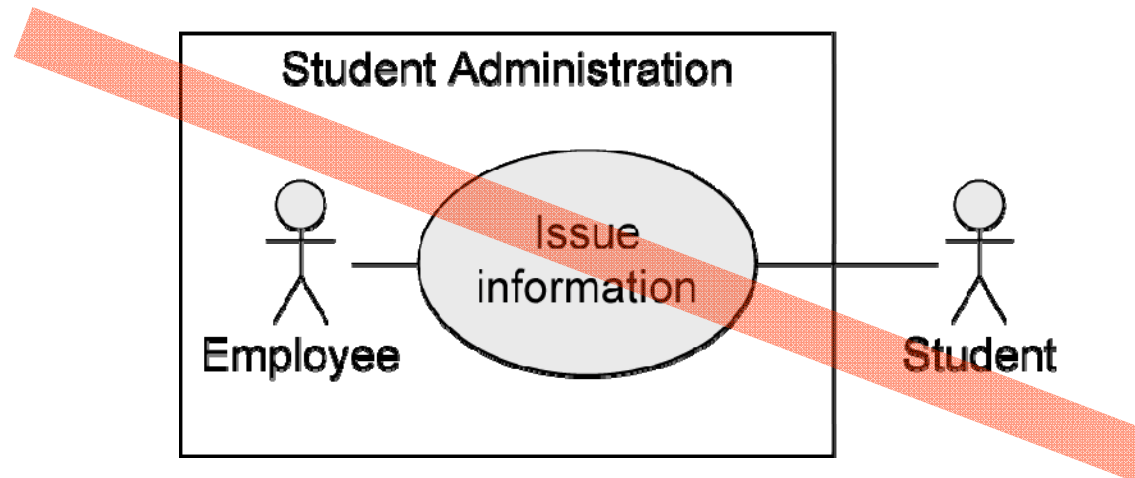
- Use case diagrams do not model processes/workflows!



Best Practices

Typical Errors To Avoid (2/5)

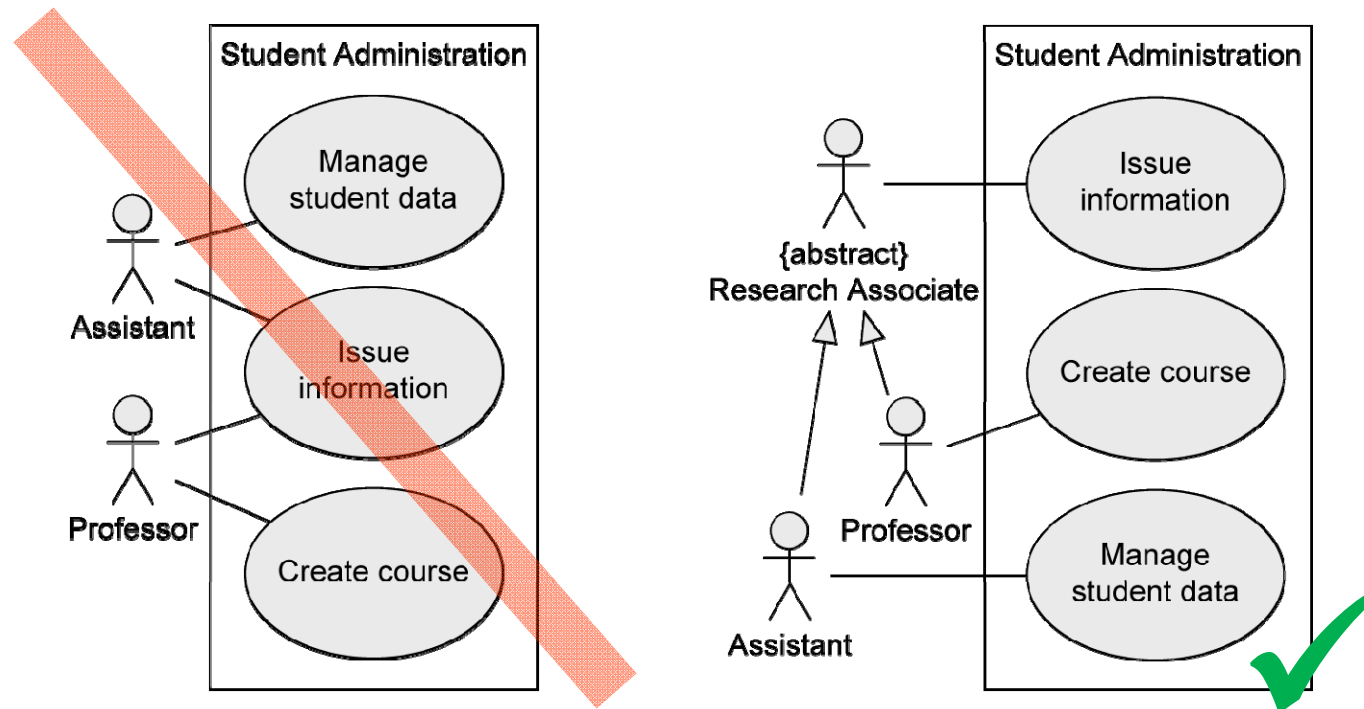
- Actors are not part of the system, hence, they are positioned outside the system boundaries!



Best Practices

Typical Errors To Avoid (3/5)

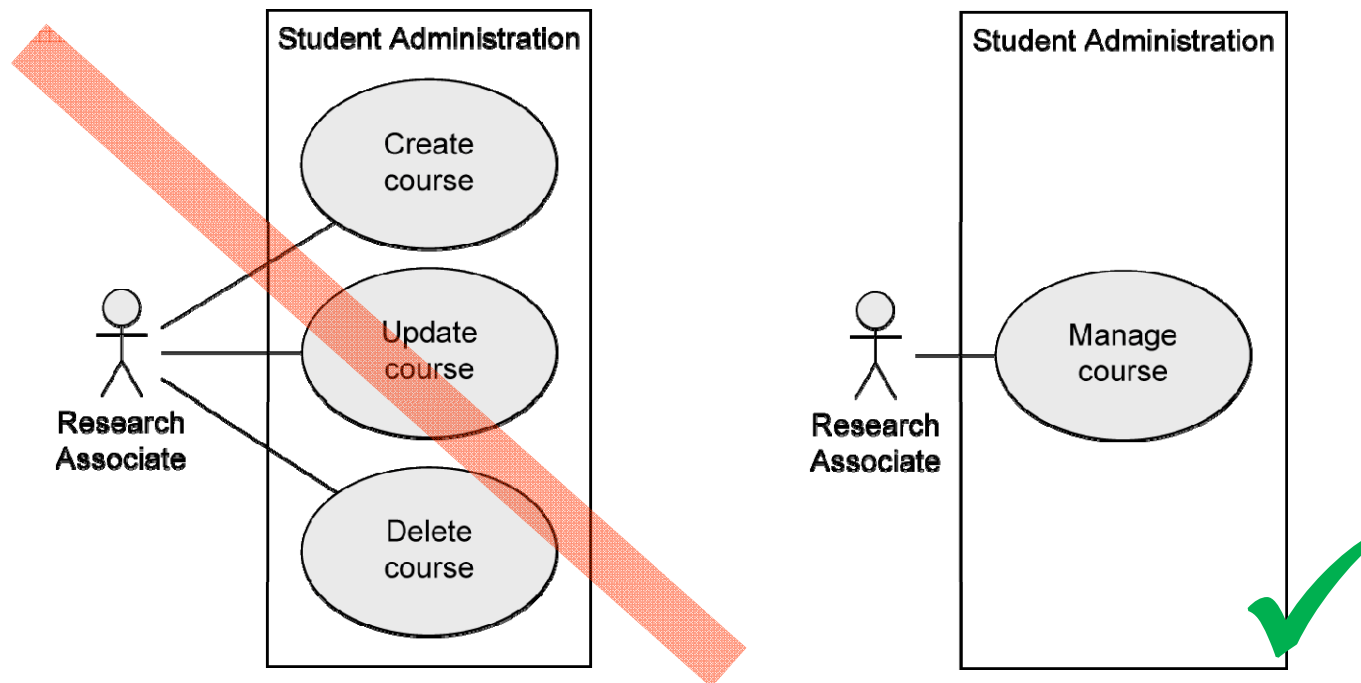
- Use case **Issue information** needs **EITHER** one actor **Assistant** **OR** one actor **Professor** for execution



Best Practices

Typical Errors To Avoid (4/5)

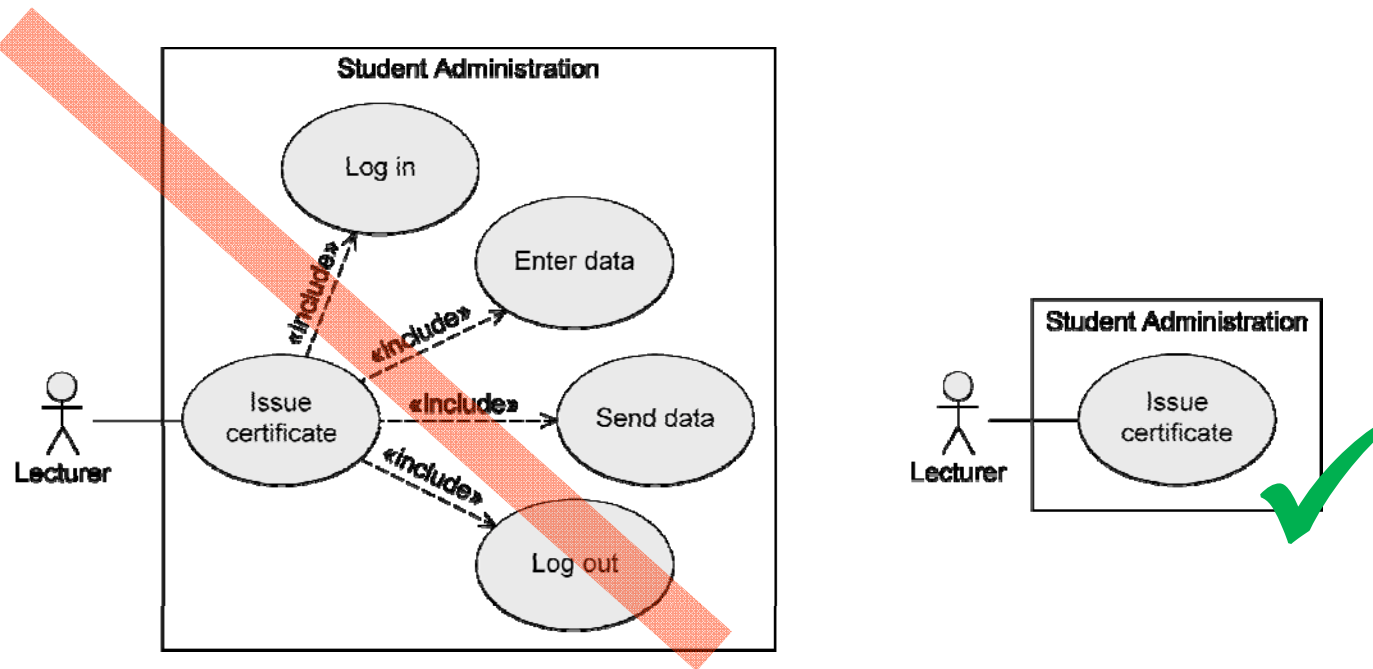
- Many small use cases that have the same objective may be grouped to form one use case



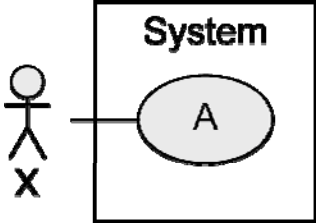


Best Practices

Typical Errors To Avoid (5/5)

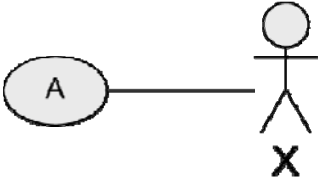

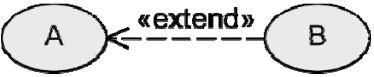
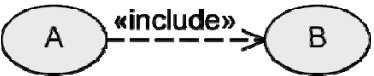
- The various steps are part of the use cases, not separate use cases themselves! -> NO functional decomposition



Notation Elements (1/2)

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system

Notation Elements (2/2)

Name	Notation	Description
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A