Lecture 3

Use cases

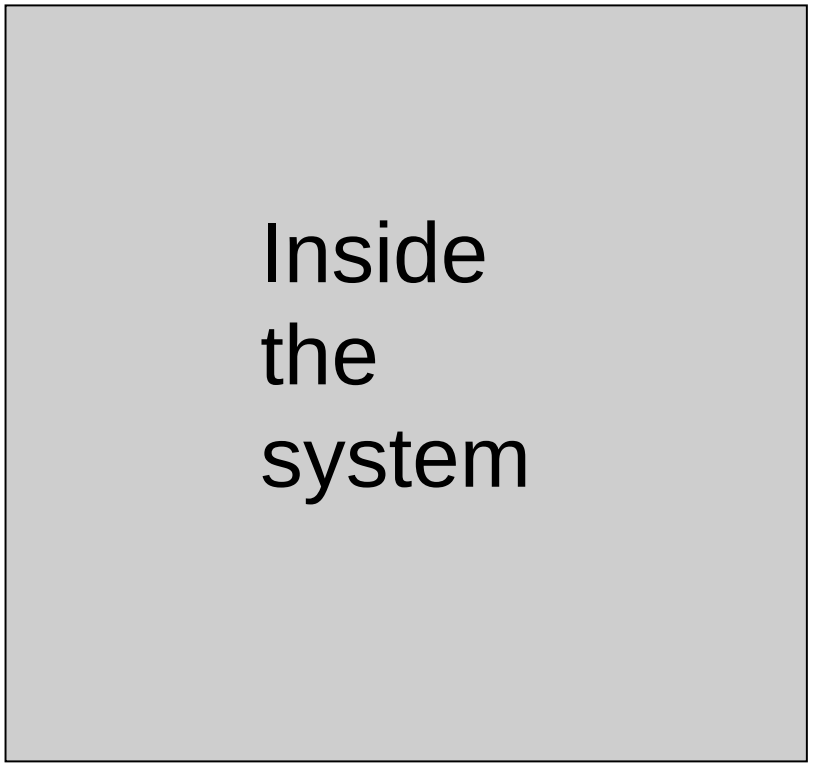
Rogardt Heldal

Objectives

- To be able to
  - Understand the difference between “inside” and “outside” of a system
  - describe the behaviour of a system using *use cases*
  - write use cases

System

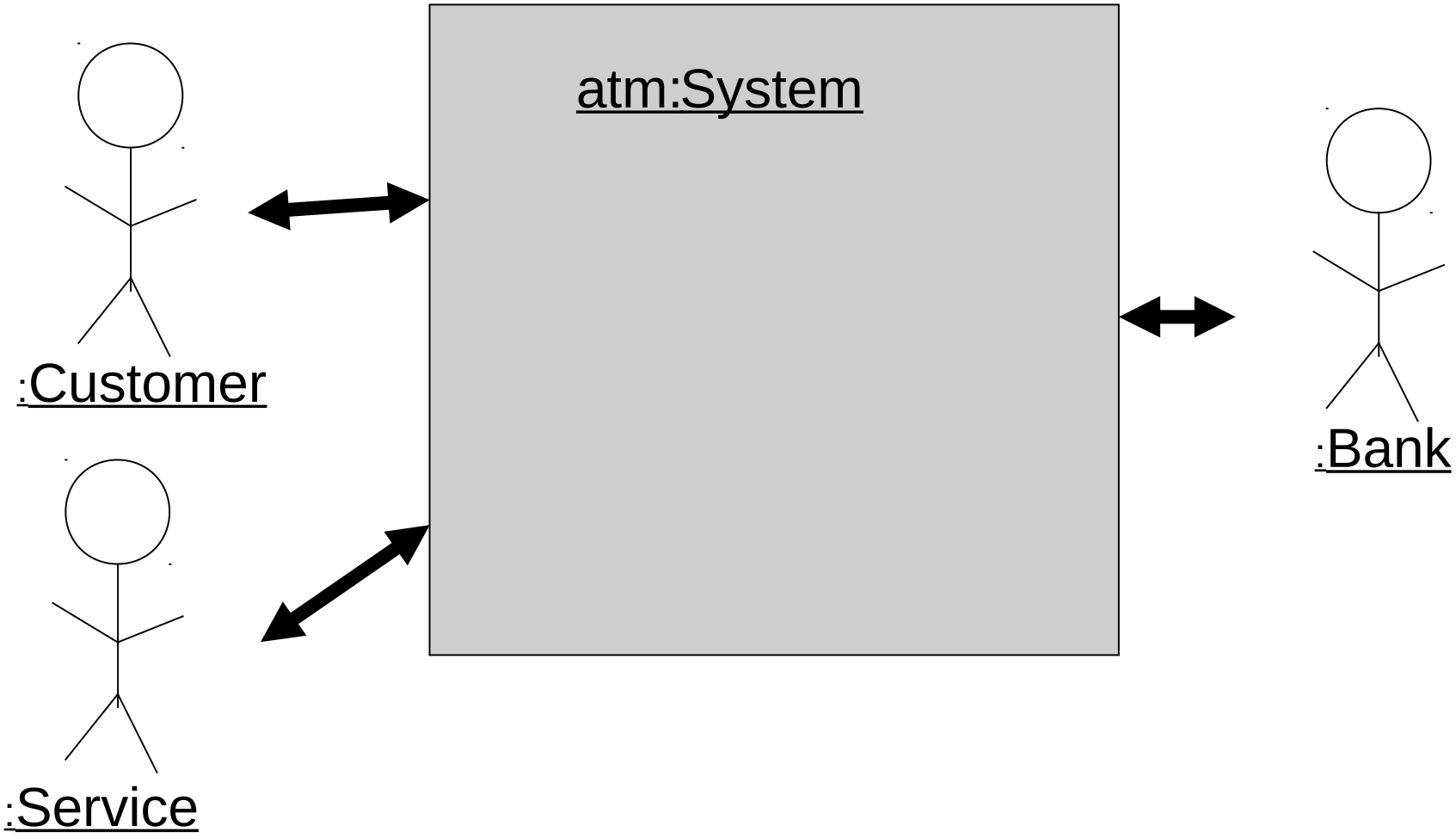
Outside the system



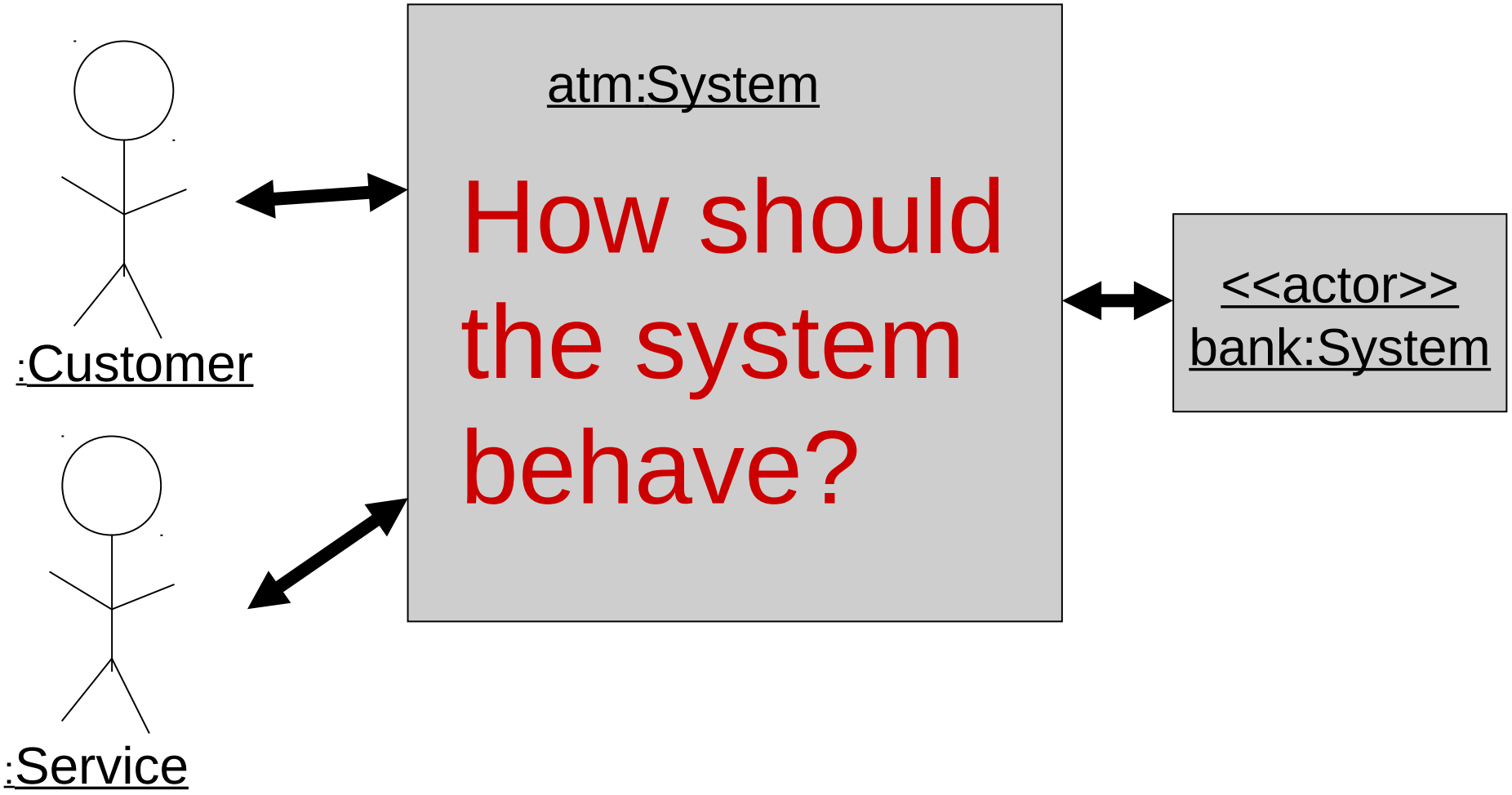
Outside the system

- Find the one who interacts with the system:
  - Actors

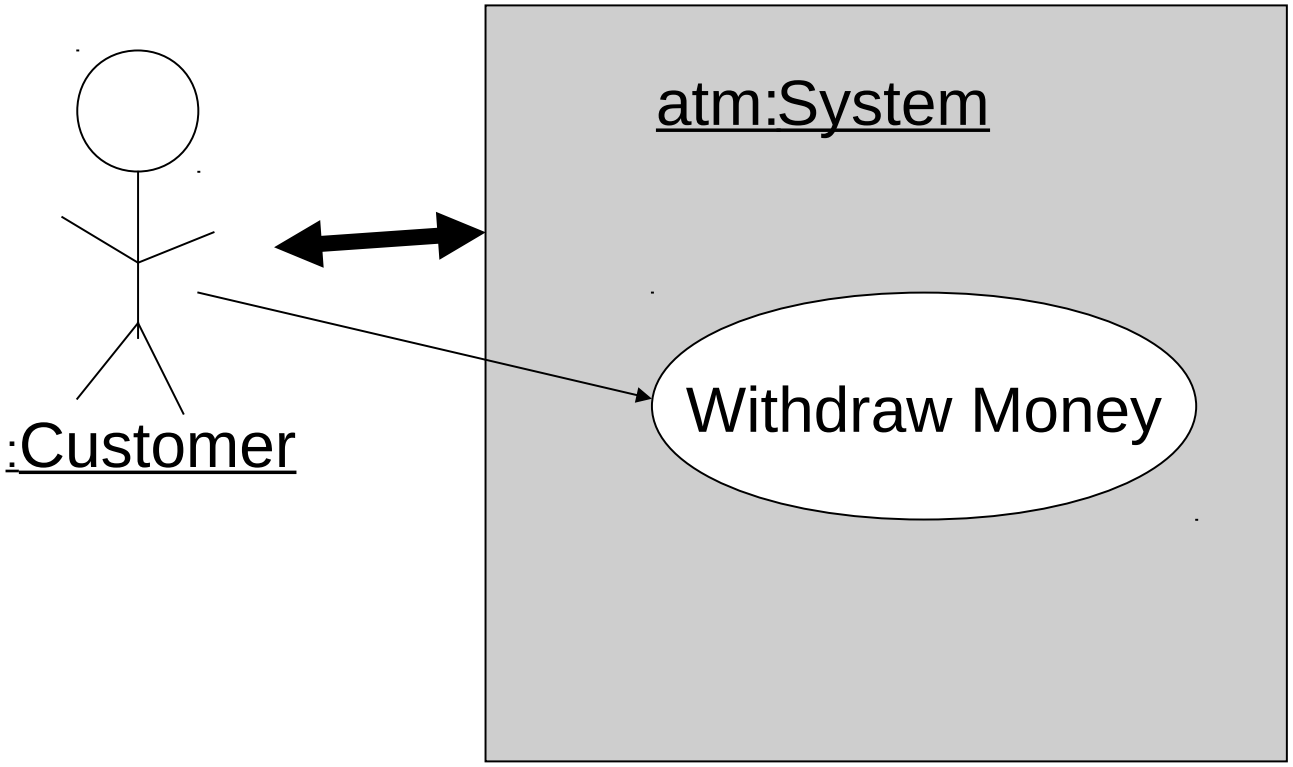
ATM



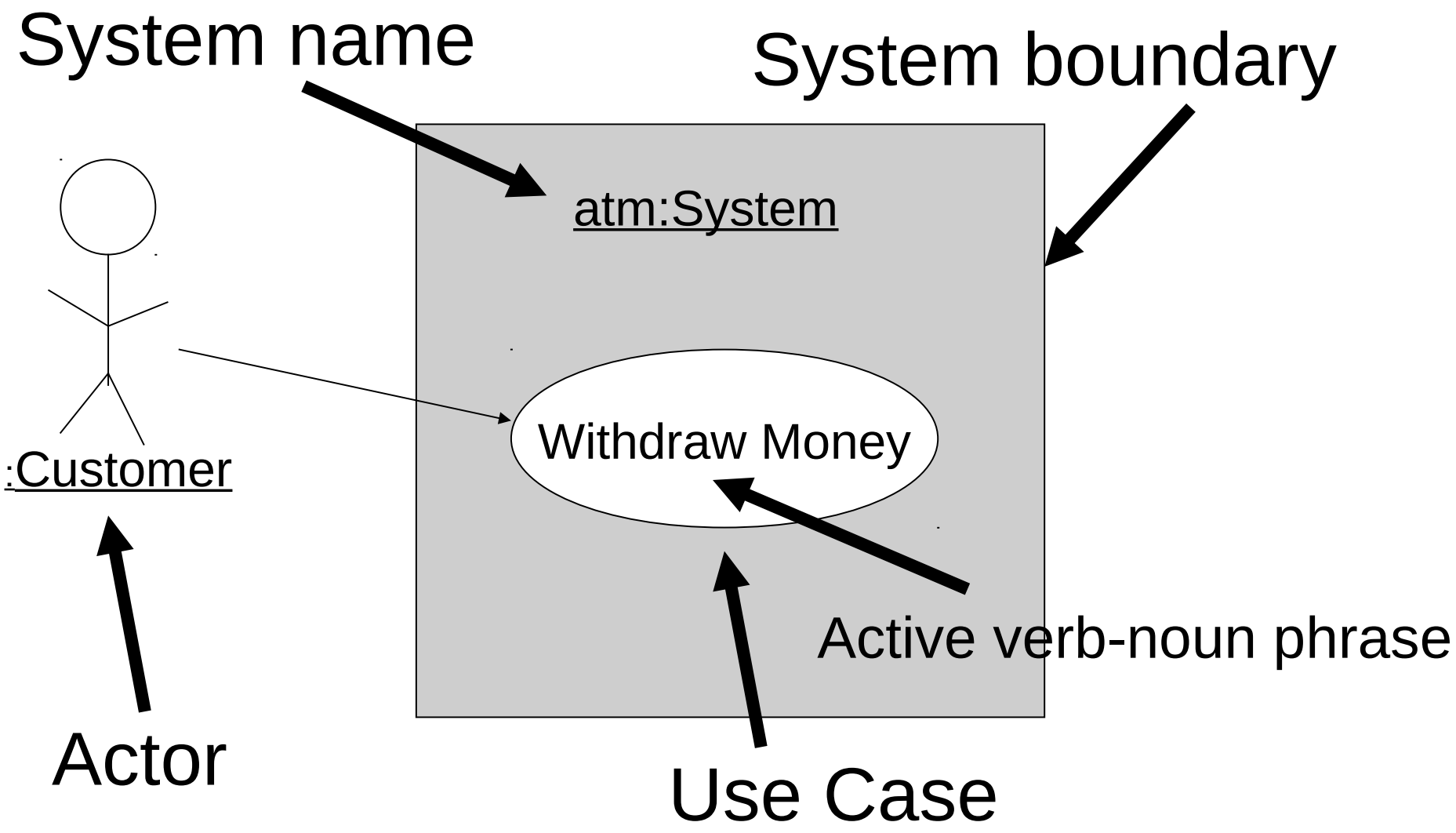
ATM


atm:System
How should the system behave?
:Customer
:Service
<<actor>>
bank:System

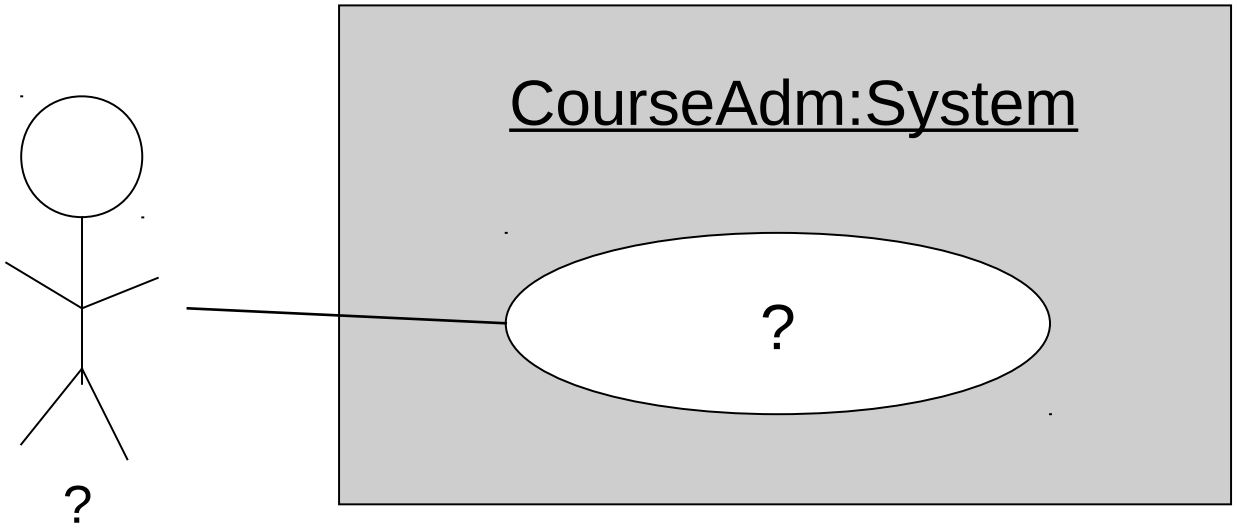
ATM



Use Case Diagram



Problem: Use Case Diagram



1. Find actors for this domain
2. Find use cases for this domain

Brief Use Cases

- A short description of the use case, for example:
  - Name: Withdraw Money
  - Actor: Customer
  - Goal: Take out money from an account
  - Description: The customer identifies himself and requests an amount of money. The ATM gives out money if the customer has sufficient funds in his account.

Problem: Brief Use Cases

- Write a brief use case for “register for course”.
  - Use Case Name: ?
  - Actor Name: ?
  - Goal: ?
  - Description: ?

Complete Use Case

- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event
  - Alternative Flows
  - Pre-Condition (if any)
  - Post-Condition
- Different organisations have different templates, but all things in this template should be part of any use case template.

Example: Flow of Events

Main flow of withdraw money:

1. user identifies himself by a card
2. system reads the bank ID and account number from card and validates them
3. user authenticates by PIN
4. system validates that PIN is correct
5. user requests withdrawal of an amount of money
6. system checks that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. system returns card and dispenses cash

Problem

- Write the main flow of events for the complete Use Case “register to course”.

Complete Use Case

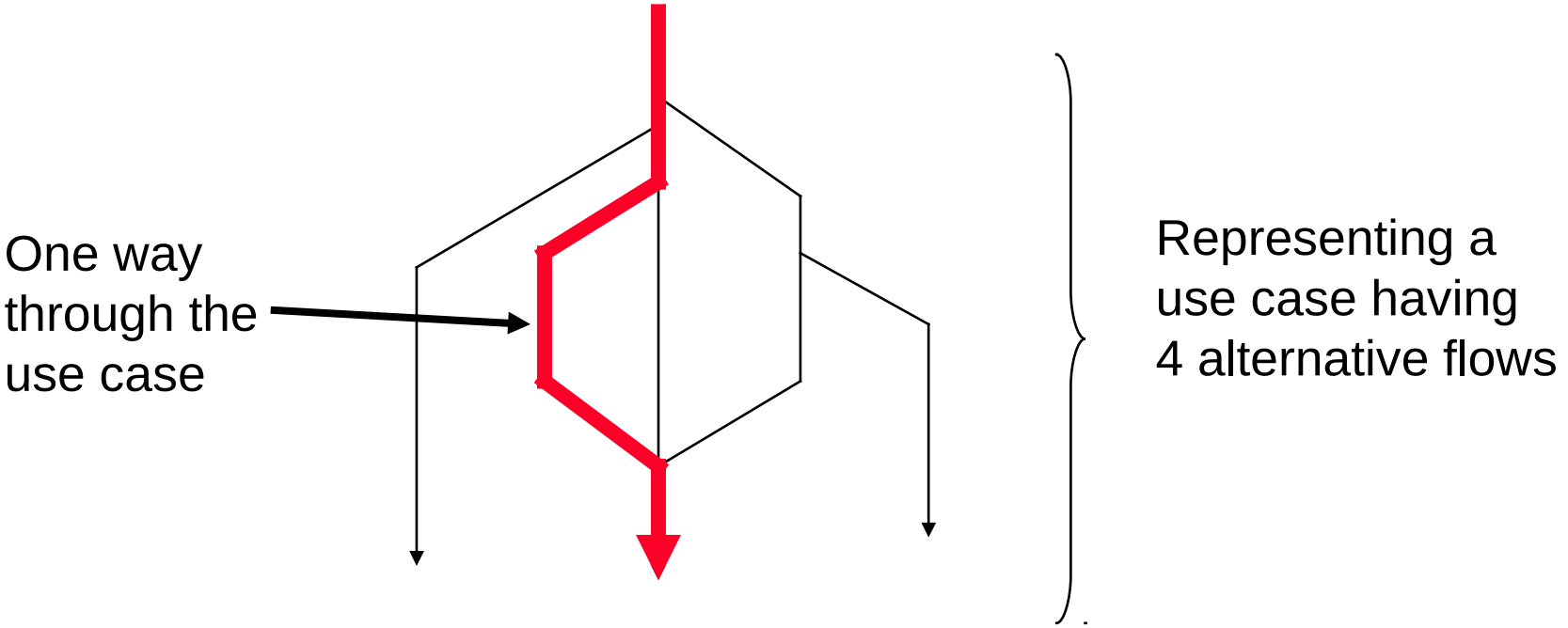
- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event (sequence of action steps)
  - Alternative Flows (sequence of action steps)
  - Pre-Condition (if any)
  - Post-Condition

Alternative flows

- Most use cases do not have just one flow, but several alternative flows.
  - Another frequent behaviour of the system
  - Another possible behaviour of the system
  - An error case
  - …
- The alternative flows depend on the input given by the actor, the system state, iterations … .

Scenarios

- One way through the use case.



One Flow of Events

Main flow of withdraw money:

1. user identifies himself by a card
2. system reads the bank ID and account number from card and validates them
3. user authenticates by PIN
4. system validates that PIN is correct
5. user requests withdrawal of an amount of money
6. system checks that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. system returns card and dispenses cash

Example: Alternative Flow

Fragment of the use case “Withdraw Money”

…

7. User requests withdrawal of an amount of money
8. System checks that the account balance is high enough
9. System subtracts from account the amount taken out from the ATM
10. System gives back card and dispenses cash

8-10a: Not enough money on account:

1. System does not change the account
2. System returns card

Numbering of Alternative Flows

1. …
2. …
3. …
4. …

3a – instead of point 3 in the main flow do this …

3b - instead of point 3 in the main flow do this …

2-3a – instead of point 2 to 3 in the main flow do this …

2-4a – instead of point 2 to 4 in the main flow do this …

2-4b – instead of point 2 to 4 in the main flow do this …

Problem

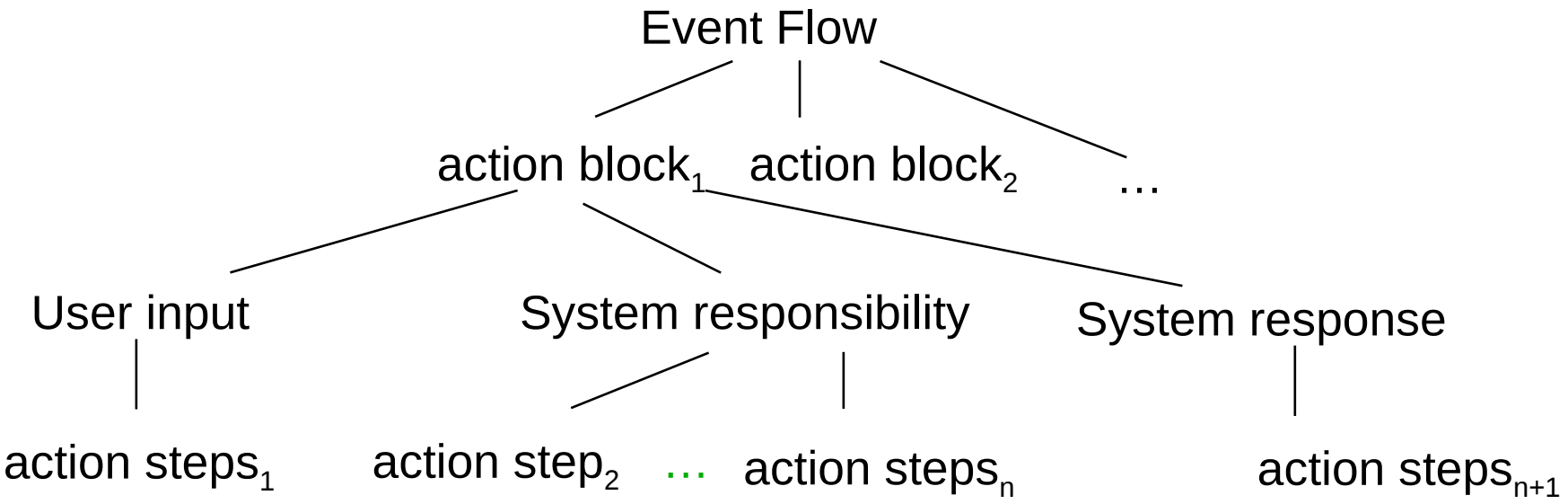
- In the case of a ATM, a user is permitted to give the wrong pin 3 times.
- If pin is given wrongly 3 times the card is kept.
- Write alternative ways for withdraw money which take this into account. Do not use WHILE loops!

Solution

- 4a. Wrong pin less than 3 times:
  - 1. System updates number of tries
  - 2. start from action step 3
- 4-8a. Wrong pin 3 times:
  - 1. System keeps the card

Support for writing better Use Case Flows

Action Block parts



Then one can have “assume” action steps when needed and a “choice” steps after return if necessary.

Withdraw Money

Only main flow:

Action block

1. user identifies himself by a card
2. Assume: bank ID and account number are the same

Action block

3. user authenticates by PIN
4. Assume: PIN is correct

Action block

5. user requests withdrawal of an amount of money
6. Assume: that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. Assume: that the ATM can communicate with the bank
9. system returns card and dispenses cash

User input

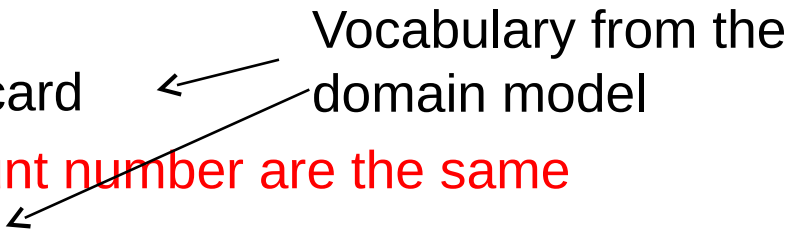
System responsibility

System response

Withdraw Money

Only main flow:

1. user identifies himself by a card
2. Assume: bank ID and account number are the same
3. user authenticates by PIN
4. Assume: PIN is correct
5. user requests withdrawal of an amount of money
6. Assume: that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. Assume: that the ATM can communicate with the bank
9. system returns card and dispenses cash



Assume and Choice

- Alternative flows can only be where there's an assume or a choice.
- For example if we have this action step in the main flow:
  - Assume: PIN is correct
- Then we can expect an alternative flow
  - Assume: PIN wrong three times

Buy Bike (Choice)

Only main flow:

1. …
2. …
3. …
4. System returns the price of the bike
5. Choice: customer wants to buy the bike
6. Customer gives the name, home address, card info

Action Block

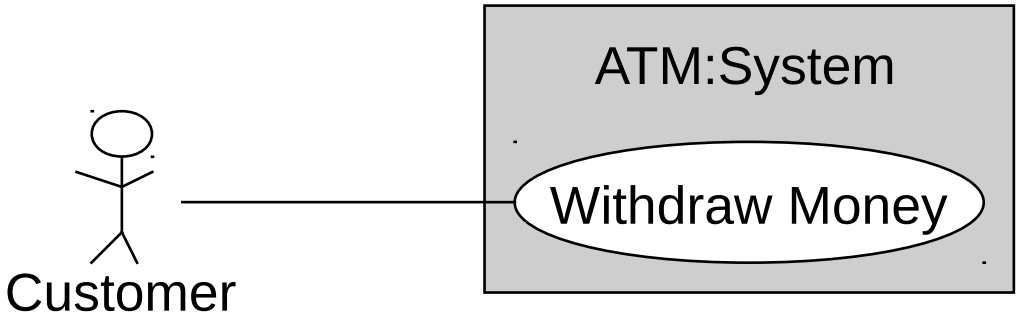
- A pattern for writing the event flows of use cases.

Objective:

- Write structured and informative use cases

When to use the Pattern

- Complete use cases
  - Containing all the important parts, in particular main and alternative flows.
- Essential style
  - Not containing implementation issues
  - Nor interface details
  - …
- System use cases (not business use cases), for example:



Complete Use Case

- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event (sequence of action steps)
  - Alternative Flows (sequence of action steps)
  - Pre-Condition (if any)
  - Post-Condition

Pre- and Post-condition

- Pre-condition is what holds, whenever the use case takes place, *before* the action steps happen.
  - For example “Withdraw Money”:
    - Customer has account
    - Customer has a bank card
    - …
  - But: Usually trivial stuff is left out
- Post-condition is what holds after the use case has taken place.
  - Examples later
- Beware: Conditions and action steps have to fit together!

Problems with pre-conditions

- Let us consider having “the customer entered the correct PIN” as a pre-condition:
  - This means that we will not specify what happens if the customer enters a wrong PIN.
  - Making the use case very weak!
- One should avoid pre-conditions as much as possible in use cases, because the usual understanding of a pre-condition is that the post-condition need only be guaranteed if the pre-condition is met before the use case.

Examples for post-conditions

- Post-condition for “Withdrawal”
  - If the customer entered the PIN on the Card, and the customer's balance was greater or equal to the requested amount, then the customer got the requested amount and the amount was deducted from the balance.
  - If the customer entered the wrong PIN three times, the card was retained.
  - If the customer requested too much money, the card was returned to the customer.

Problem

- Write a post-condition for “register for course”?

Use case Register Student

Pre: True

Post:
if Student id exists **and** course code exists **and** places exist on the course and student meets all the course prerequisites
then Student is registered on the course
else **nothing is changed in the system**

Use case Register Student

Pre: True

1. User inputs student id and course code
2. System finds student and course
3. Assume: that student and course exists
4. System registers student on course
5. Assume: enough places on the course
6. Assume: student has the required courses

Post:
if Student id exists **and** course code exists **and** places exist on the course and student meets all the course prerequisites
then Student is registered on the course
else **nothing is changed in the system**

Make implicit calls explicit

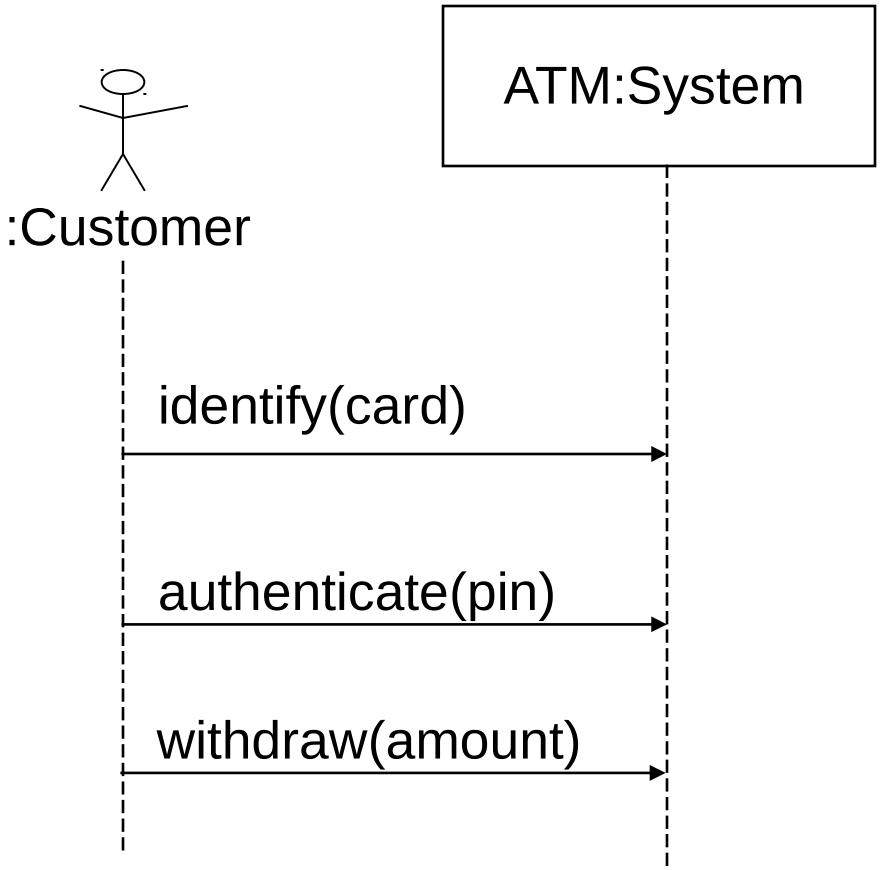
Use case: Withdraw Money

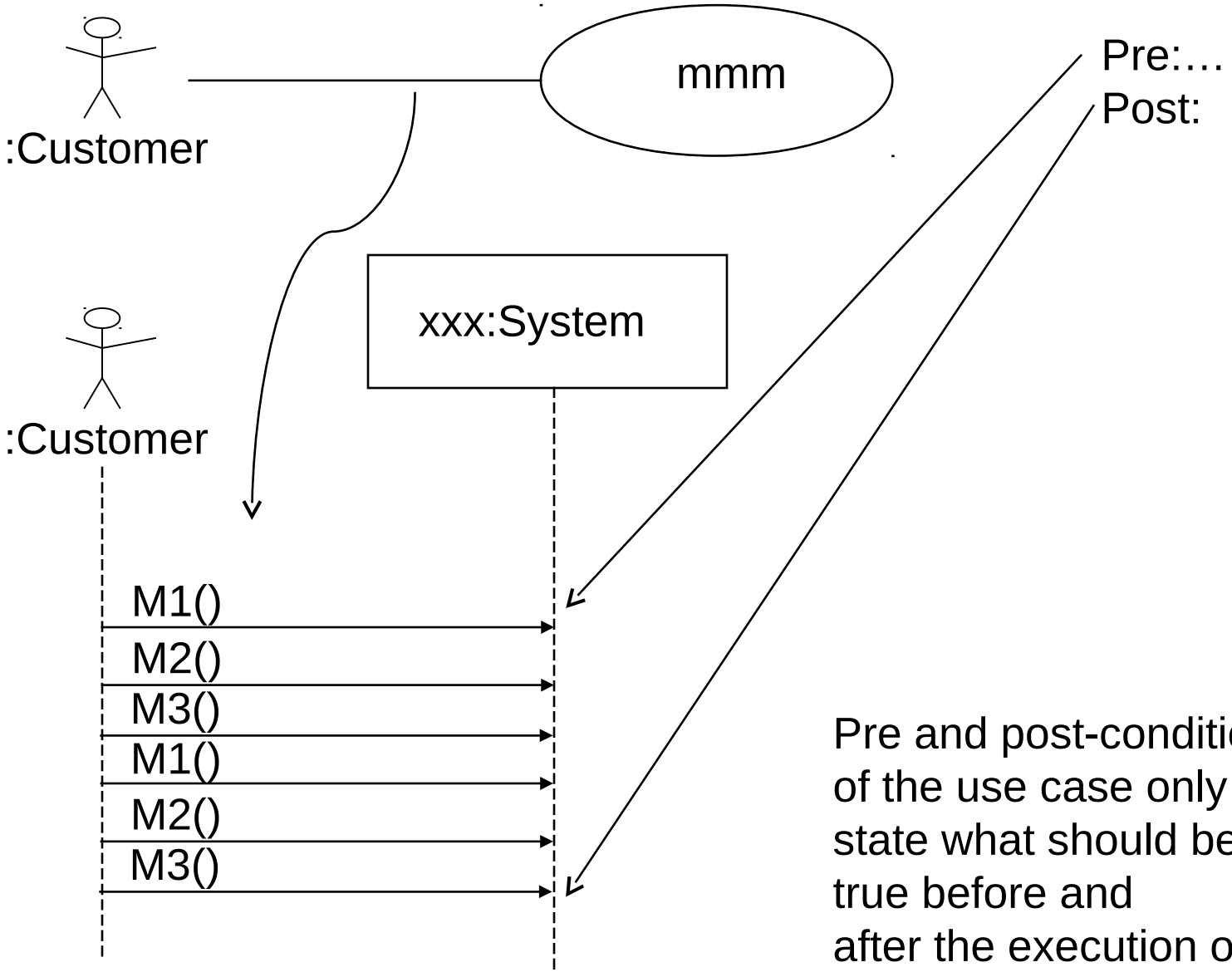
Only main flow:

1. user identifies himself by a card
2. system reads the bank ID and account number from card and validates them
3. user authenticates by PIN
4. system validates that PIN is correct
5. user requests withdrawal of an amount of money
6. system checks that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. system returns card and dispenses cash

Suggested names for operations

System Sequence Diagram Withdraw Money





Pre and post-condition of the use case only state what should be true before and after the execution of all the methods

Primary Actor

- A use case is always started by some actor : most often the primary actor
- There are also other types of actors:
  - Secondary actor
  - Helper actor
  - Time

Requirements on Use Cases

- Shall be a complete process
- Shall result in a given goal

External and Internal View

External View

User	System
identify himself	
	present choices
choose	
	dispense money
take money	

Can be viewed as a role play between user and system.

Example: Internal View

User	System
identify himself	
	verify identity
	present choices
choose	
	dispense money
take money	

Example: Internal View

1. User identifies himself
2. System verifies identity
3. System presents choices
4. User chooses
5. System dispenses money
6. User takes money

Action Block Details

- Two types:
  - Black box
    - User intention
    - System response
  - White box
    - User intention
    - System responsibility ← This is extra
    - System response
- In the white box case the system response can often be left out if it is clear from the system responsibility how the system will respond.

Abstraction level

Real Use Cases

- Contain design details.
- Different aspects:
  - Internal view of the system
  - Also gives information about user interface (i.e., not essential use case)
  - Consider also technology issues (like databases)

Real Use Case?

- Is this a real use case?
  - User authenticates himself by PIN
  - System validates that PIN is correct
- Less abstract than
  - Customer identifies himself
  - System verifies identity
- But the first fragment makes more sense for people working in the banking industry.
- The second fragment is often too general, can be used in several contexts:
  - ATM
  - Library system
  - Student Registration system

Which abstraction level to use?

- During analysis, one should write essential use cases.
- Later, during design, essential use cases can be refined to real use cases.
- But we believe that there are better ways of defining the design than using real use cases

Process for Brief Use Cases

- Find actors
- Consider the goals of each actor
- Write brief use cases, based on the goals of the use case:
  - Give a name of the use case
  - Give the actor(s)
  - Write the goal as one sentence
  - Write a brief description

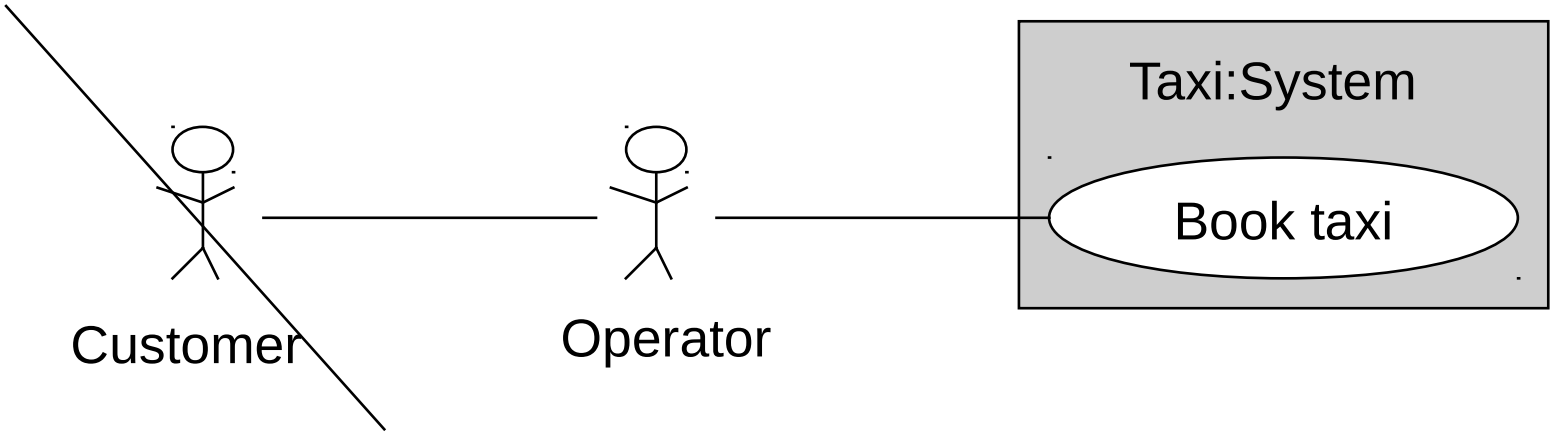
Prioritize Use Cases

- Based on the importance of a brief use case, a complete use case should be written.
- For ordering use cases take into consideration:
  - How important is the use case for the business?
  - Does the use case have important consequences for the system (technological, architectural)? These use cases should come early.

Process for Complete Use Case

- Based on a brief use case
  - Write the main flow of control
  - Write the alternative
  - Write pre-conditions (if any)
  - Write the post-conditions

Not good style



Not the way of doing it:

1) Customer calls operator
2) Customer gives bla bla to operator
3) Operator enter info bla bla into the system
4) Operator gives info bla bla back to customer

Only include interaction between actor and system!