Lecture 4

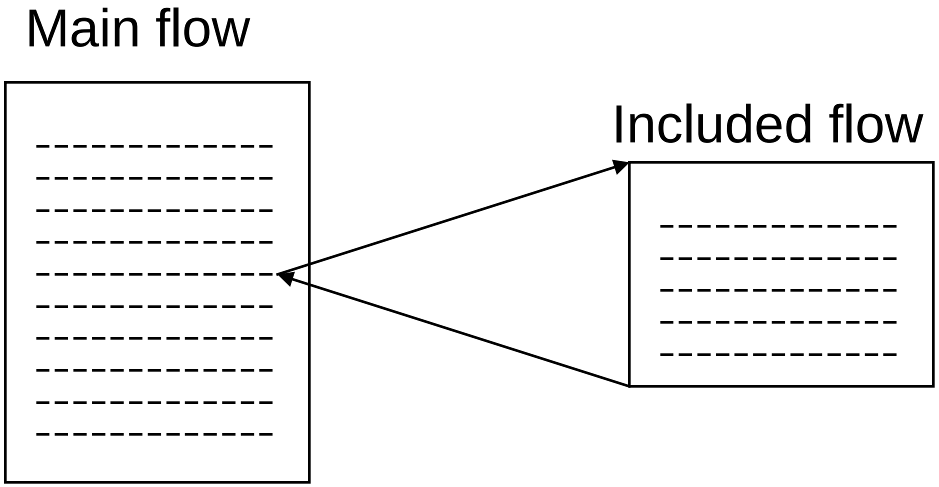
Use cases

Rogardt Heldal

Advanced topics

”Include”

- Often used to catch common action steps



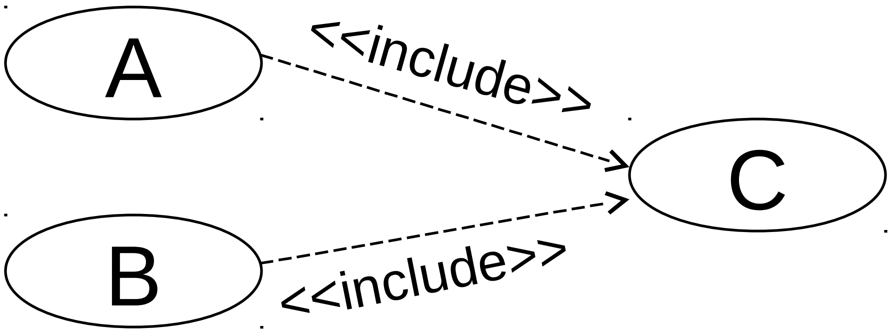
- Important: one has to leave and come back to the main flow at the same place.

- In the flow which is included add action step:

  include (the name of the included use case)

UML-syntax

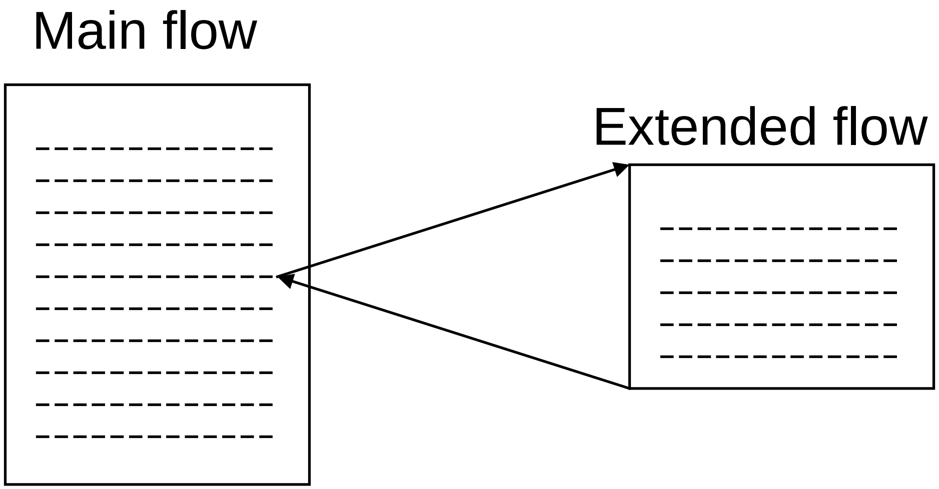
- Use cases A and B includes C:



  - A and B know about C, but not the other way round.
  - C must have at least as high priority as A and B.

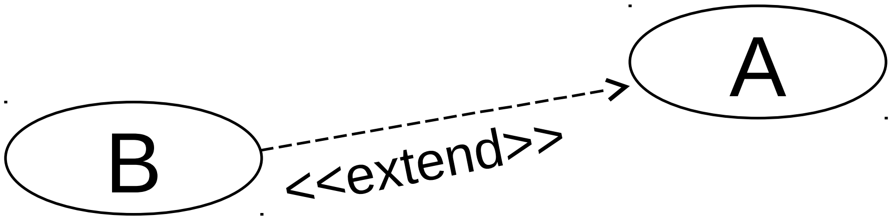
”Extend”

- Sometimes one wants to include extra action steps in a use case. Then ”extend” might be useful:



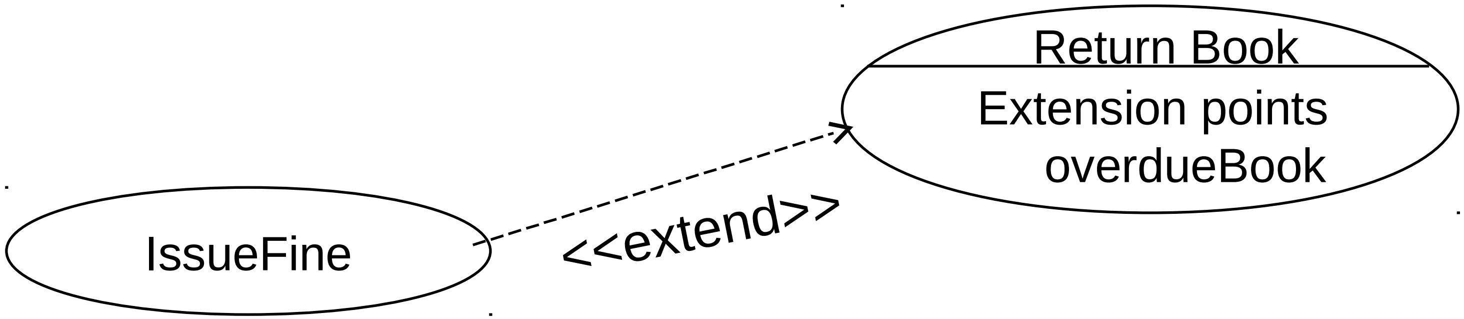
- Important: one has to leave and come back to the main flow at the same place.

UML Syntax



- A is extended with B
- Some condition has to be satisfied for the use case B to be used.
- A has to be a full use case without B.

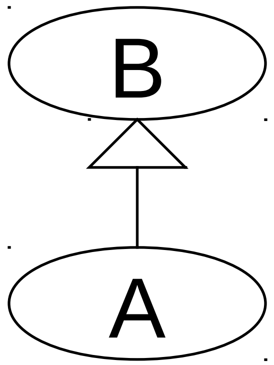
extend



- In the use case IssueFine’s flow:
- …
- …
- Extension point:overdueBook
- …

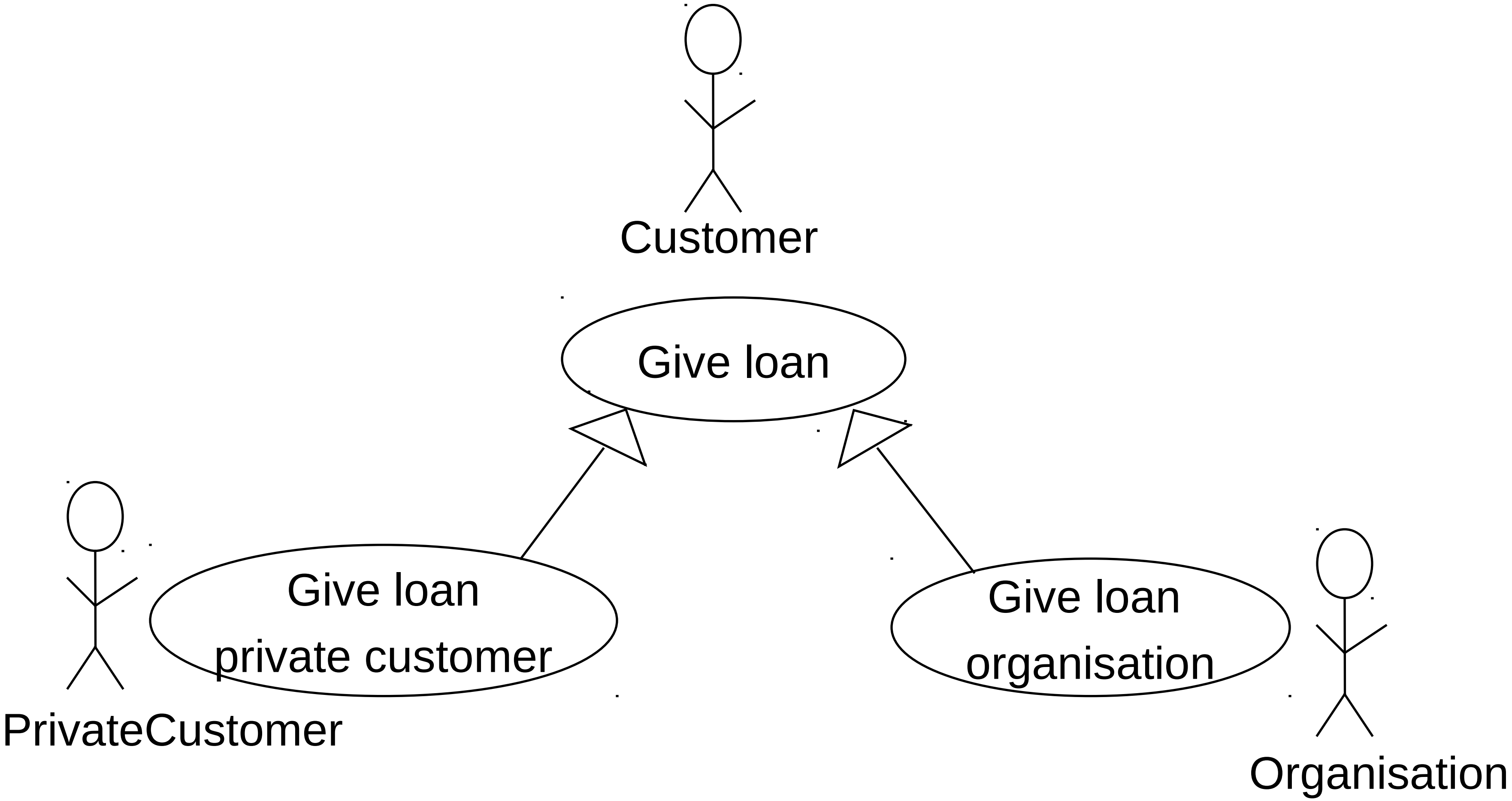
Inherit relation

- It might happen that several use cases have action steps which are similar. In this case one can use abstract use cases:



- A inherits from B.

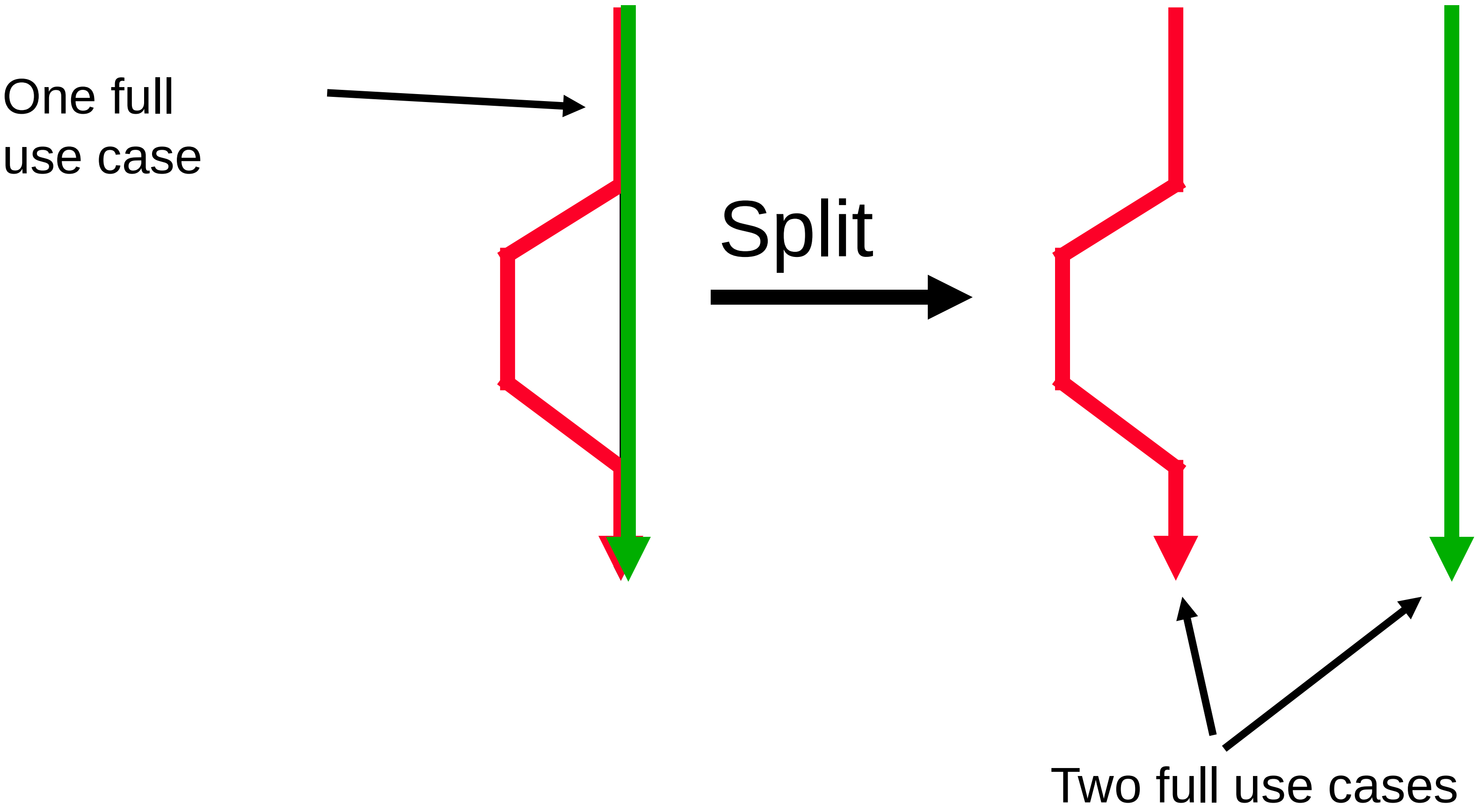
Example


Customer
Give loan
Give loan
private customer
Give loan
organisation
PrivateCustomer
Organisation

Uses

- Whether to use ”include”, ”extend”, and inherit is discussed a lot.
- Most important reason for using these features: they can improve readability

Split

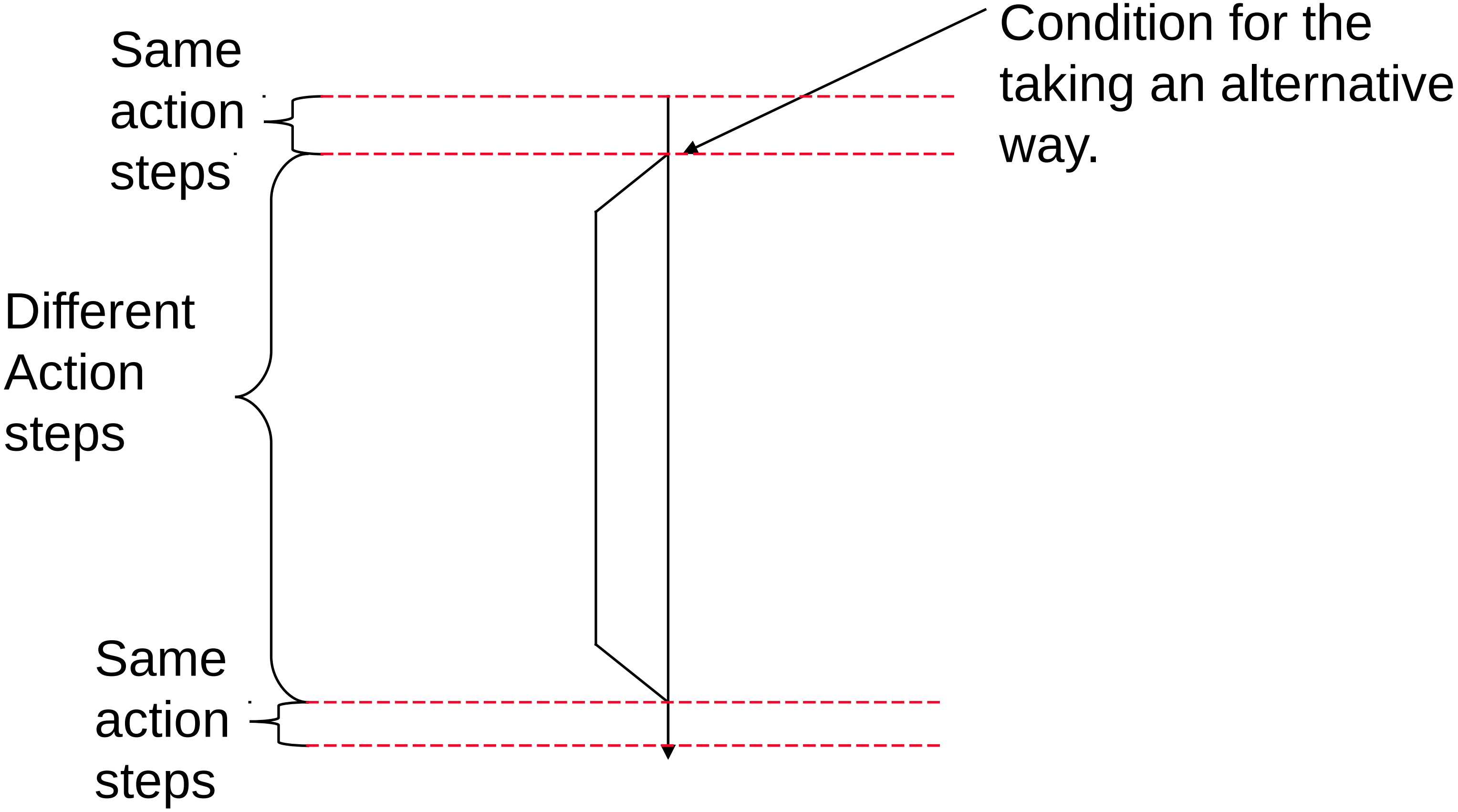


One full use case

Split

Two full use cases

Vertical split



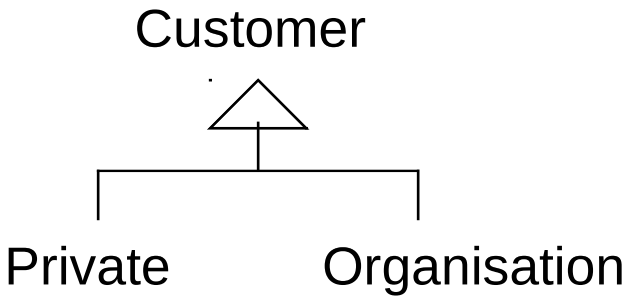
When to split

- If it is not important to show the condition
- If main flow and alternative flow do not have many steps in common
- If the two flows are complete use cases in themselves

- Sometimes one might want to combine use cases as well.

Actor Specialisation

- Vertical split can lead to more specialised actors, for example:

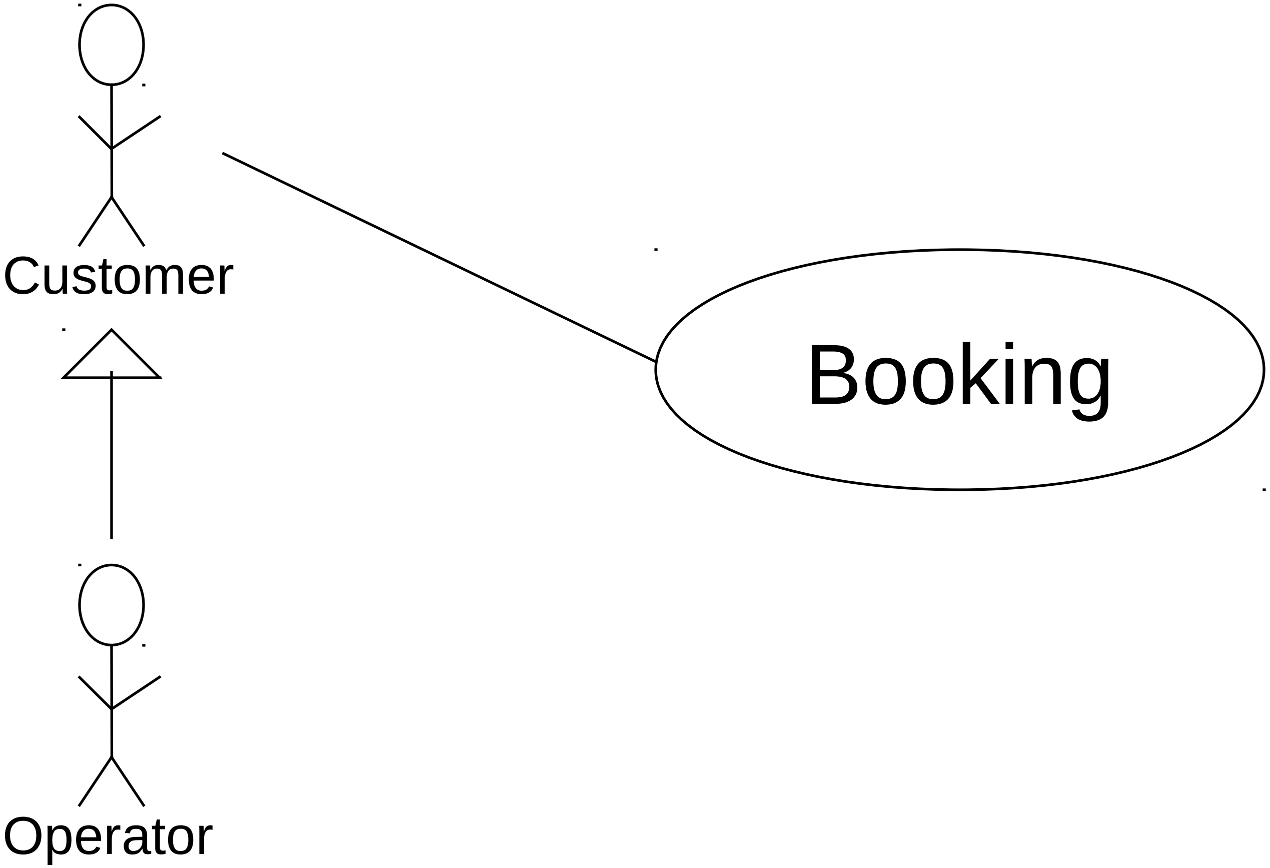


Customer

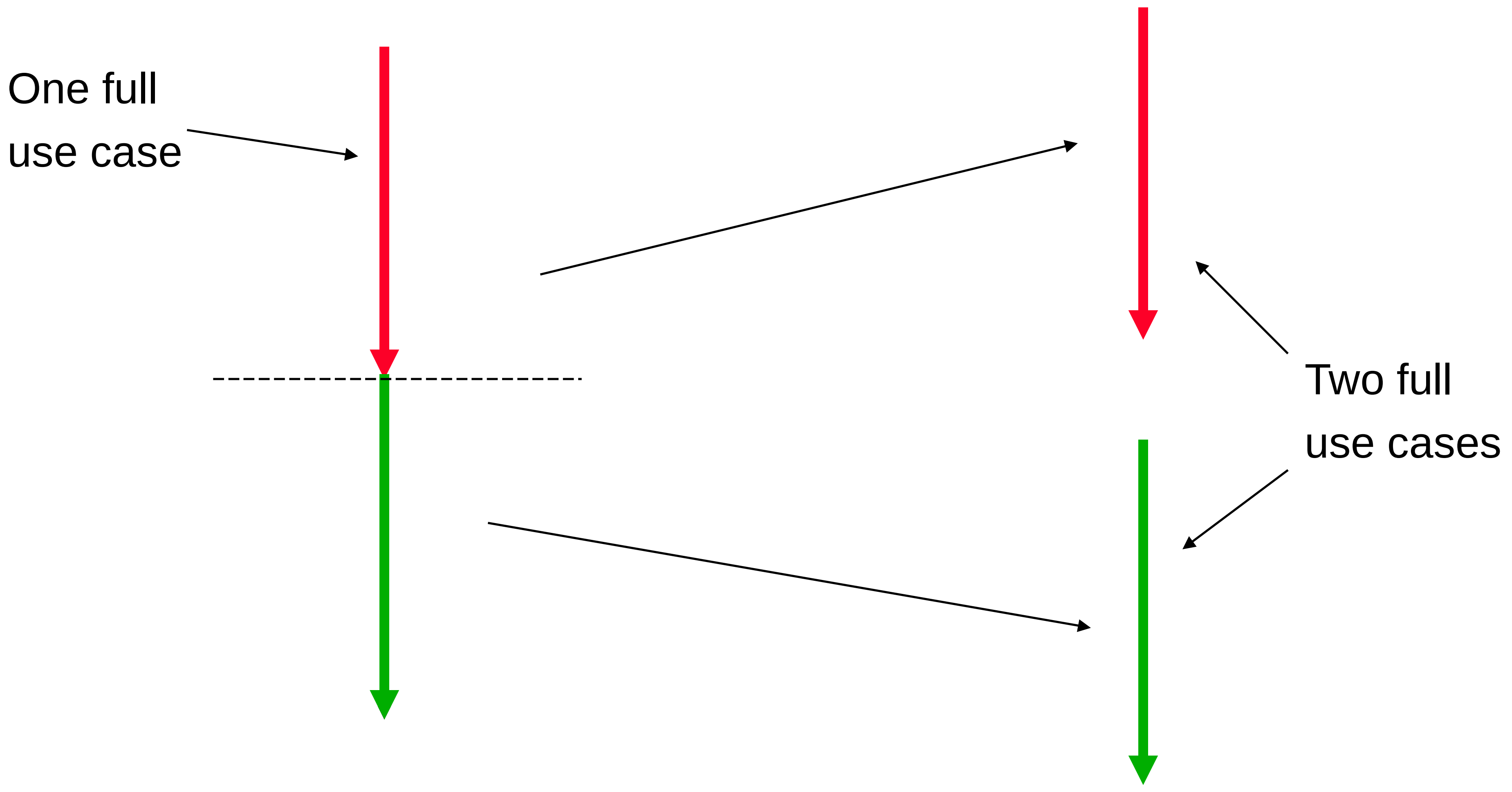
Private

Organisation

Example of Actor Inheritance



Horizontal Split



Business Use Cases

- Describes how a business works. May describe human behaviour as well.
- May contain system use cases.

System Sequence Diagram

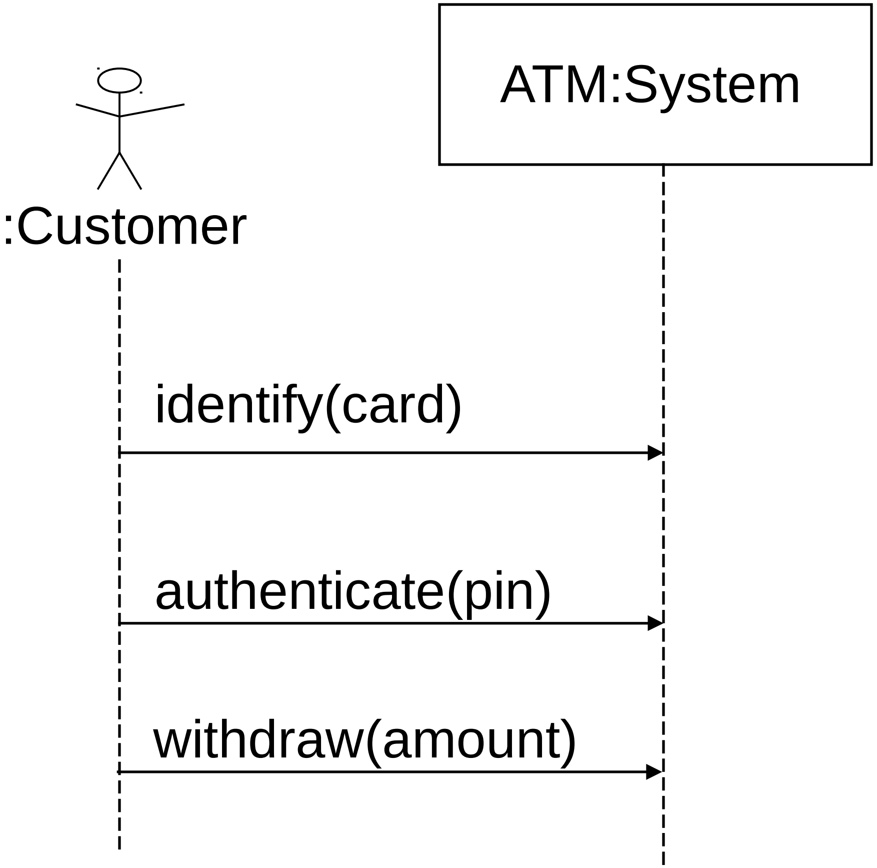
Use case: Withdraw Money

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

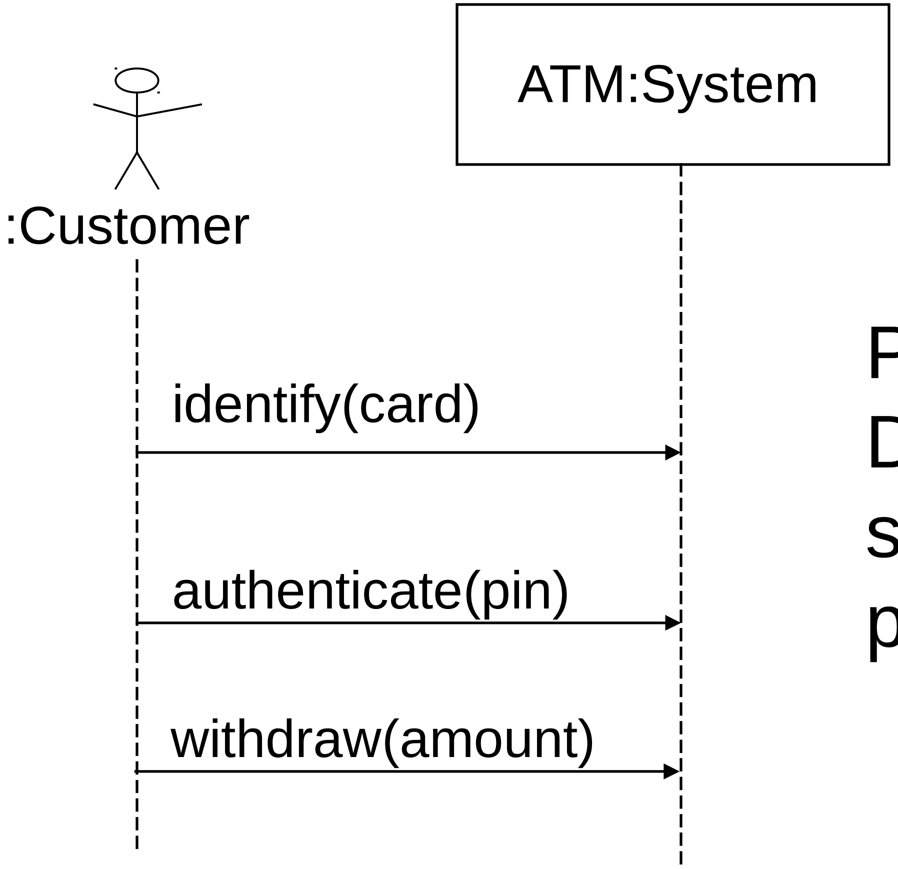
System Sequence Diagram Withdraw Money



Kinds of Sequence Diagrams

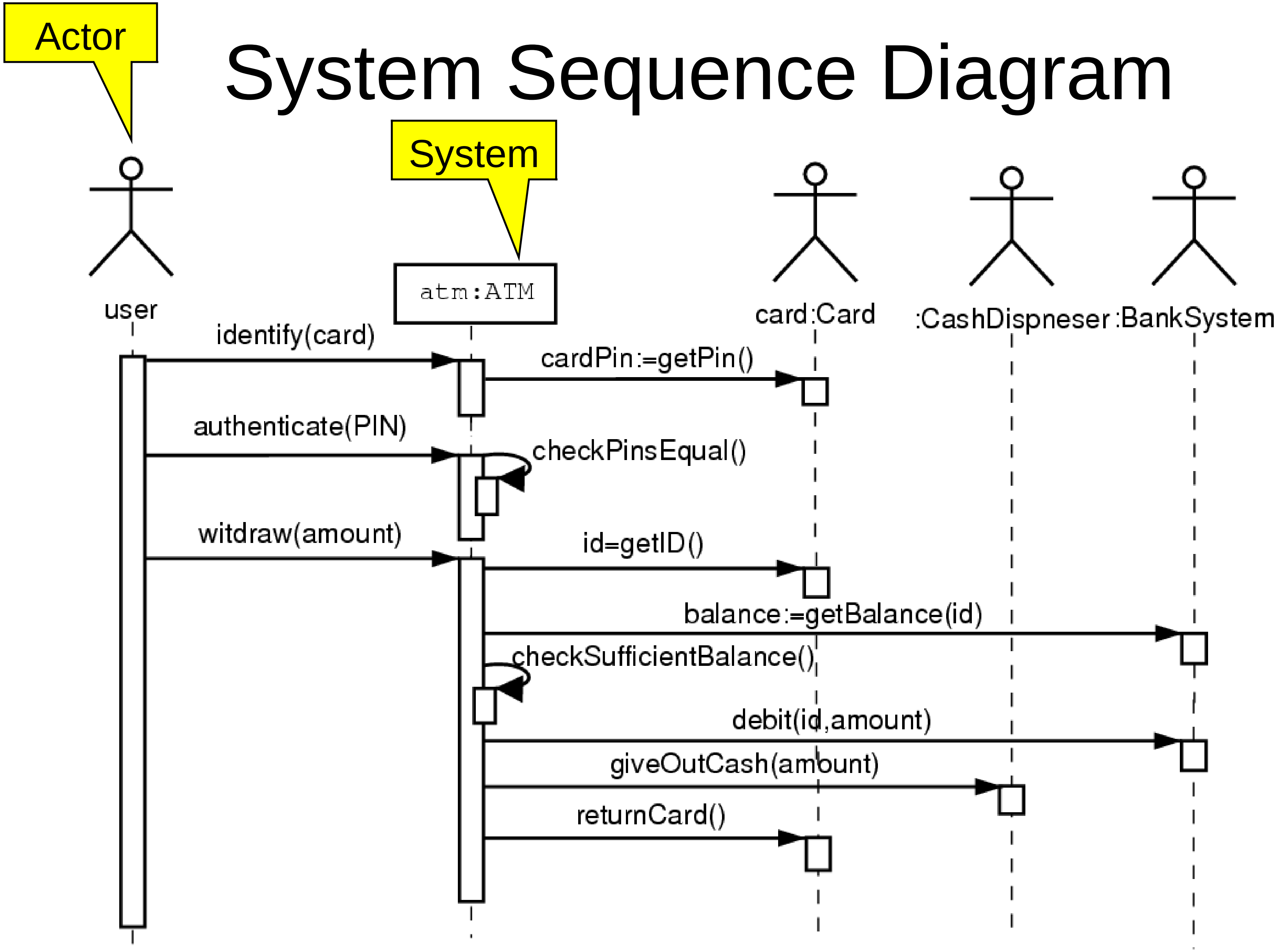
- Interaction Diagrams can be used on different levels:
  - System level ("System sequence diagrams"): Interaction between actors (primary, secondary, …) and system
  - Component level: Interaction between components
  - Object level: Interaction between objects, maybe in one system or component
- Notation is always the same

System Sequence Diagram
Withdraw Money

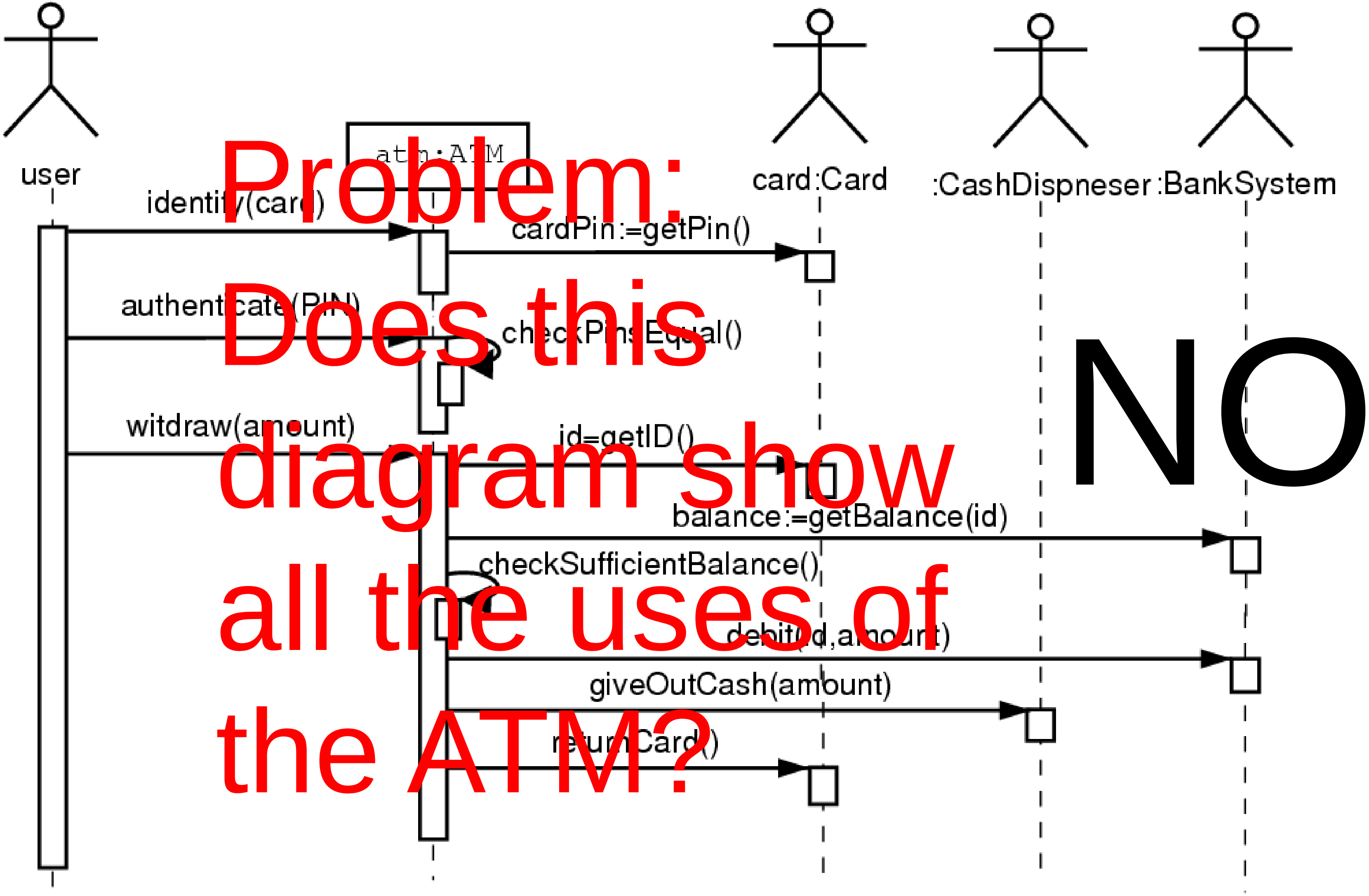


Problem:
Does this diagram show a realistic picture of the ATM?

System Sequence Diagram



Other Systems

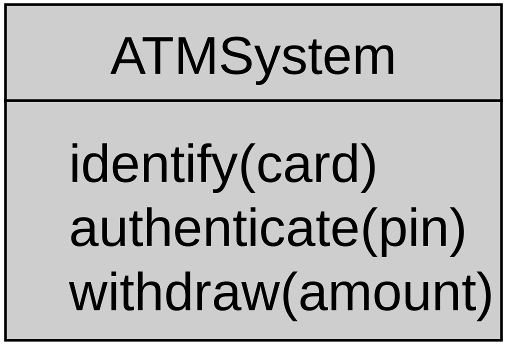


Problem: Does this diagram show all the uses of the ATM?

NO!

System Class

- We can consider a system class as a façade for the whole system.



ATMSystem

identify(card)
authenticate(pin)
withdraw(amount)

Contract

System operations

Example: Contract

- Operation: withdraw(amount:int)
- Postcondition:
  - **If** account contains enough cash

    then the balance of the account for the inserted card
    has been decreased by “amount” **AND**
    the card has been returned **AND**
    cash has been dispensed

    else the account balance has not been changed **AND**
    the card has been returned

Contract Template

- The signature of the operation:
  - Name, parameters, return value
- Description of the operation (optional), for instance
  - Informal meaning of operation
  - Implementation in pseudo-code
- Description of the parameters (optional)
- Description of the operation's result (optional)
- Cross-reference
- Precondition
- Postcondition

Use Domain Model to obtain pre- and post-conditions

- Furthermore, the domain model can be used as the basis for the creation of the contracts.
  - The precondition specifies what has to hold in the domain model before the call to the operation.
  - The postcondition specifies what has to hold in the domain model after the execution of the call.

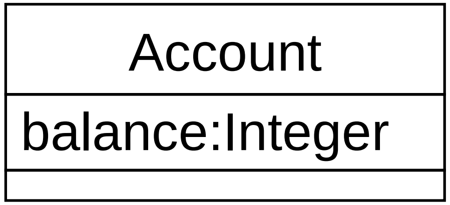
Postcondition

- The postcondition has to specify the following things:
  - What instances have been created?
  - What attributes have been modified?
  - What associations (to be precise, UML links) have been formed or broken?
  - What value is returned from the operation?

Example: Withdraw Money

- Which attributes are modified?

The balance attribute in the Account concept might be changed.



Account

balance:Integer

Problem

Write a contract for the operation authenticate.

…

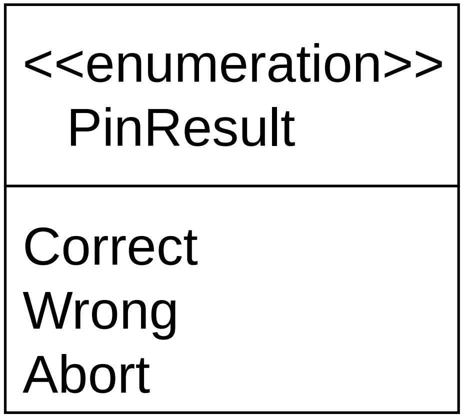
4. user authenticates himself by PIN
5. system validates that PIN is correct

…

- 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

Part of the solution

- Operation: Authenticate (userPin: Integer):PinResult
- Cross-ref: Withdraw Money
- Result:
  - PinResult::Correct if authentication successful,
  - PinResult::Wrong if authentication failed, but further tries are possible
  - PinResult::Abort if authentication failed
- post-condition: ?



<<enumeration>>
PinResult

Correct
Wrong
Abort

Solution

- Operation: Authenticate (userPin: Integer): PinResult
- …
- post-condition:
  - **if** userPin was equal to the pin of the inserted card

    then PinResult::Correct has been returned

    else if tries was at most 3

        then tries has been incremented by 1 **AND**

            PinResult::Wrong has been returned

        else card has been kept **AND**

            PinResult::Abort has been returned

More details about Contracts

- In contracts, one often is more precise than in use cases, even formal.
- On the next slide we show a formal contract written in Object Constraint Language (OCL) for Withdraw Money.
- We might come back to OCL later in this course.

Formal Contract

```
Context ATMController::giveAmount(amount:long) post:
  if ( amount <= bank.getBalance(card.getID()) ) then
        cashDispenser^giveOutCash(amount)
    and   bank.getBalance(card.getID())
        = bank.getBalance@pre(card.getID()) - amount
    and card^returnCard()
  else
        not cashDispenser^giveOutCash(?)
    and   bank.getBalance(card.getID())
        = bank.getBalance@pre(card.getID())
    and card^returnCard()
```

Problem

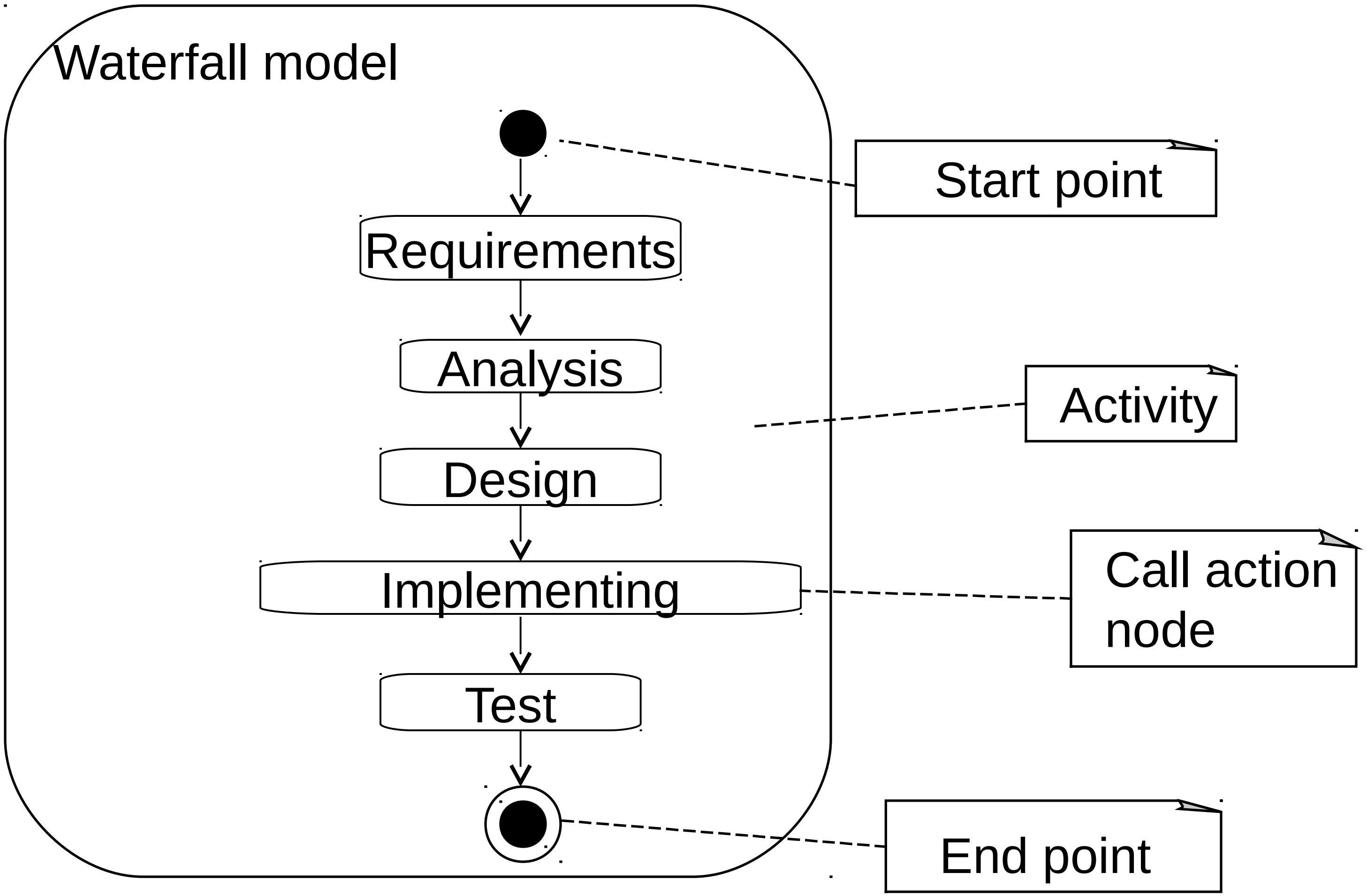
- Write a contract for the system operations obtained from “register on course”.

Activity diagram

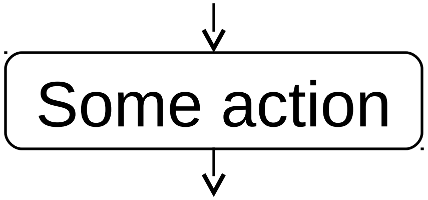
Activity diagrams

- Can be used to:
  - Describe sequences of activities
  - Both sequential and parallel
- Can be useful for
  - business modelling
  - describing the flows of a use case
  - …

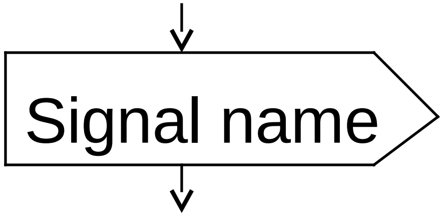
Work flow/process



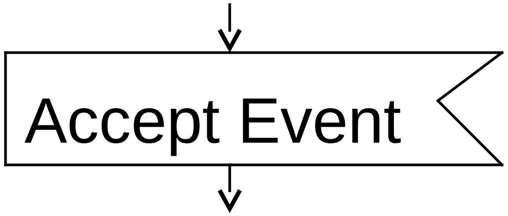
Action nodes



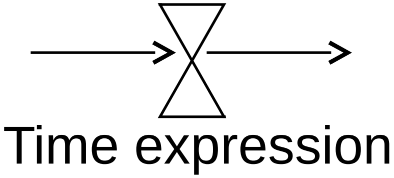
Call action node



Send signal

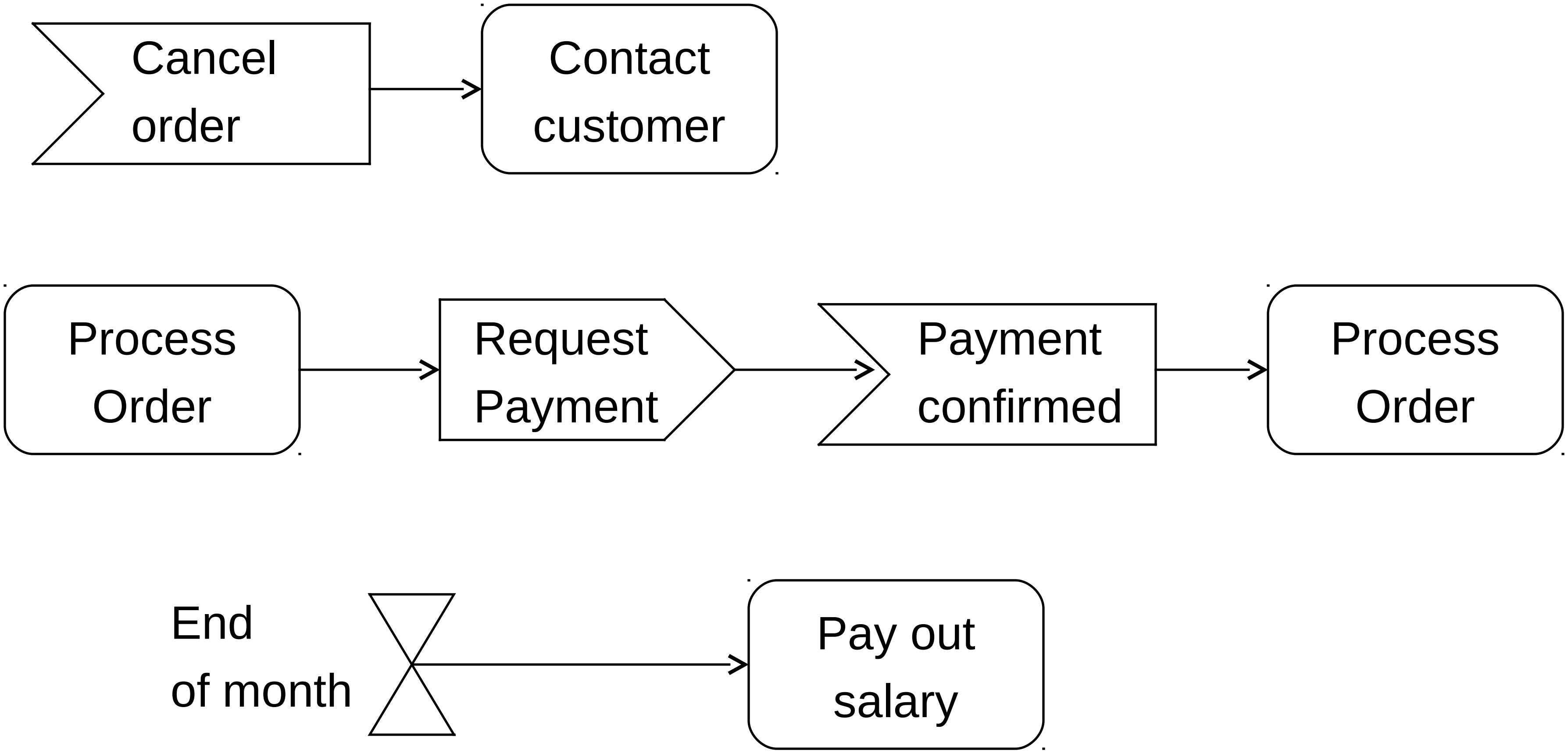


Accept event action node

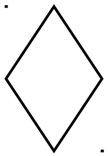


Accept time event action node

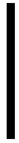
Example


Cancel order
Contact customer
Process Order
Request Payment
Payment confirmed
Process Order
End of month
Pay out salary

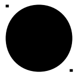
Control nodes



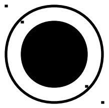
Decision node or Merge node



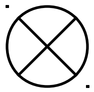
Fork node, join node



Initial node

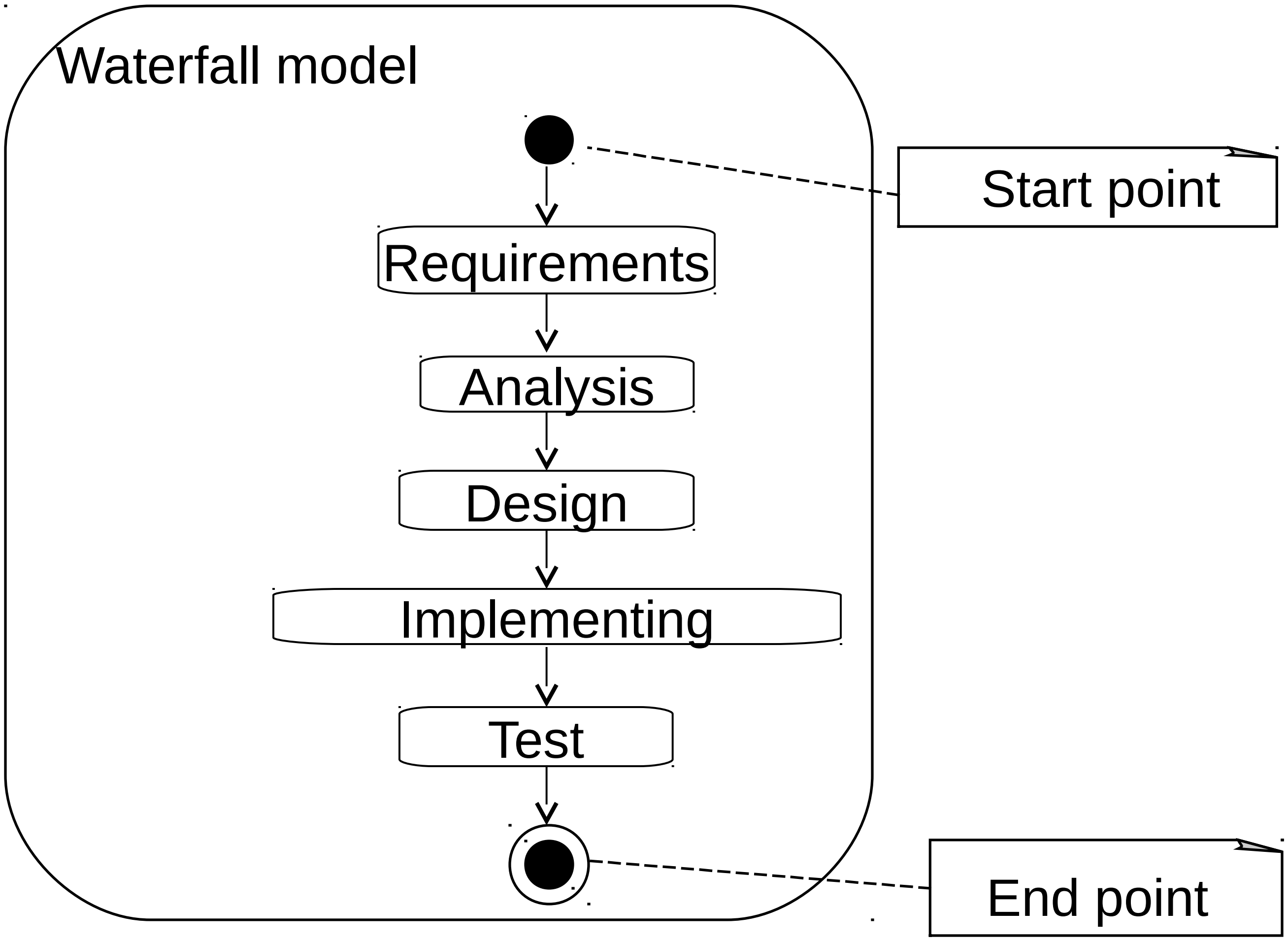


Activity final



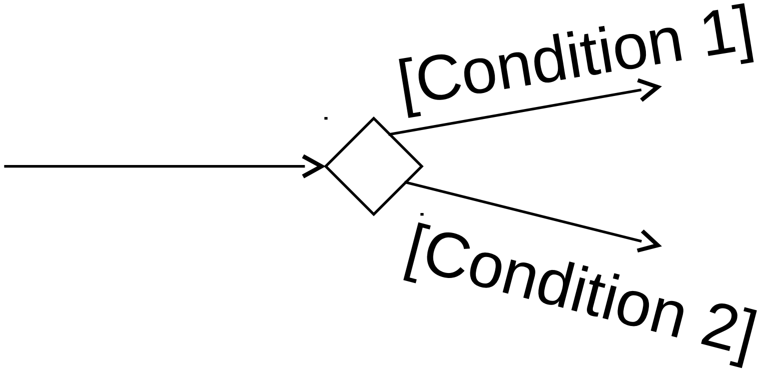
Flow final

Work flow/process

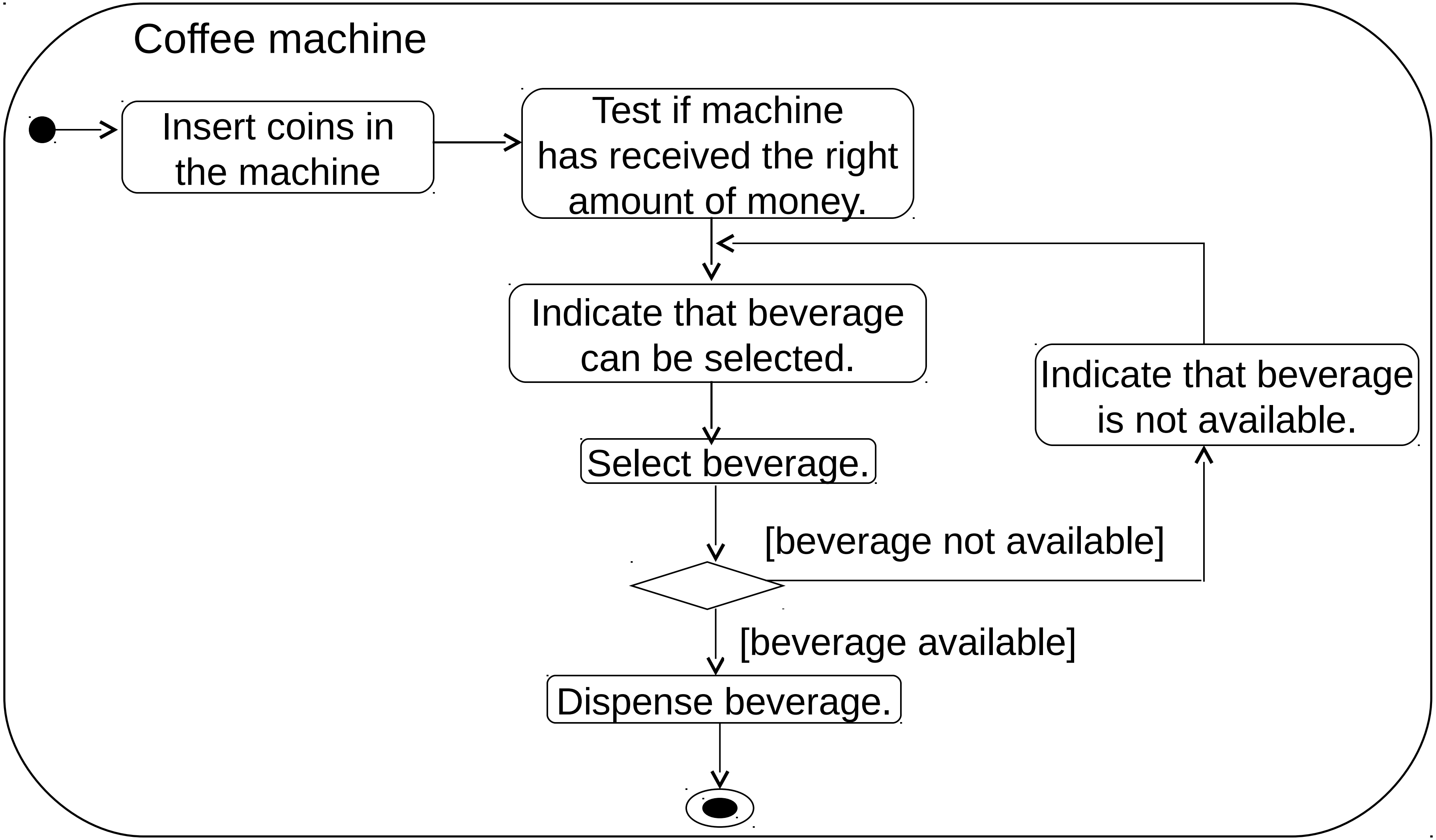


Decision node

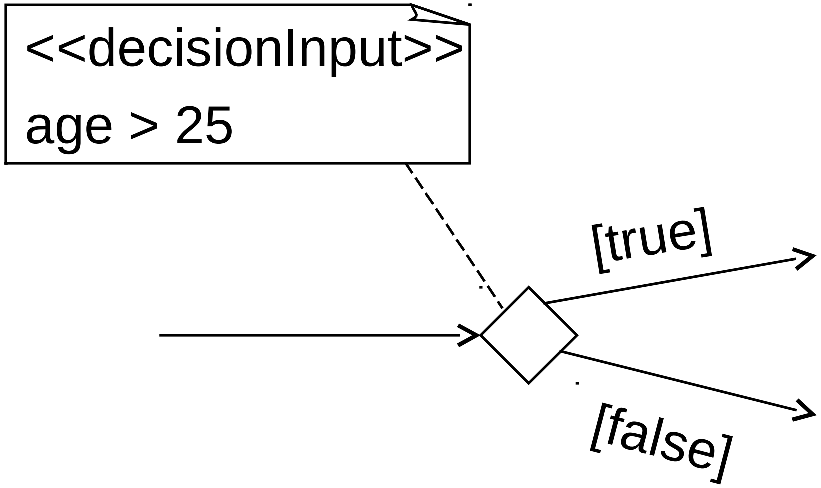
- The output edge whose guard condition is true is traversed.



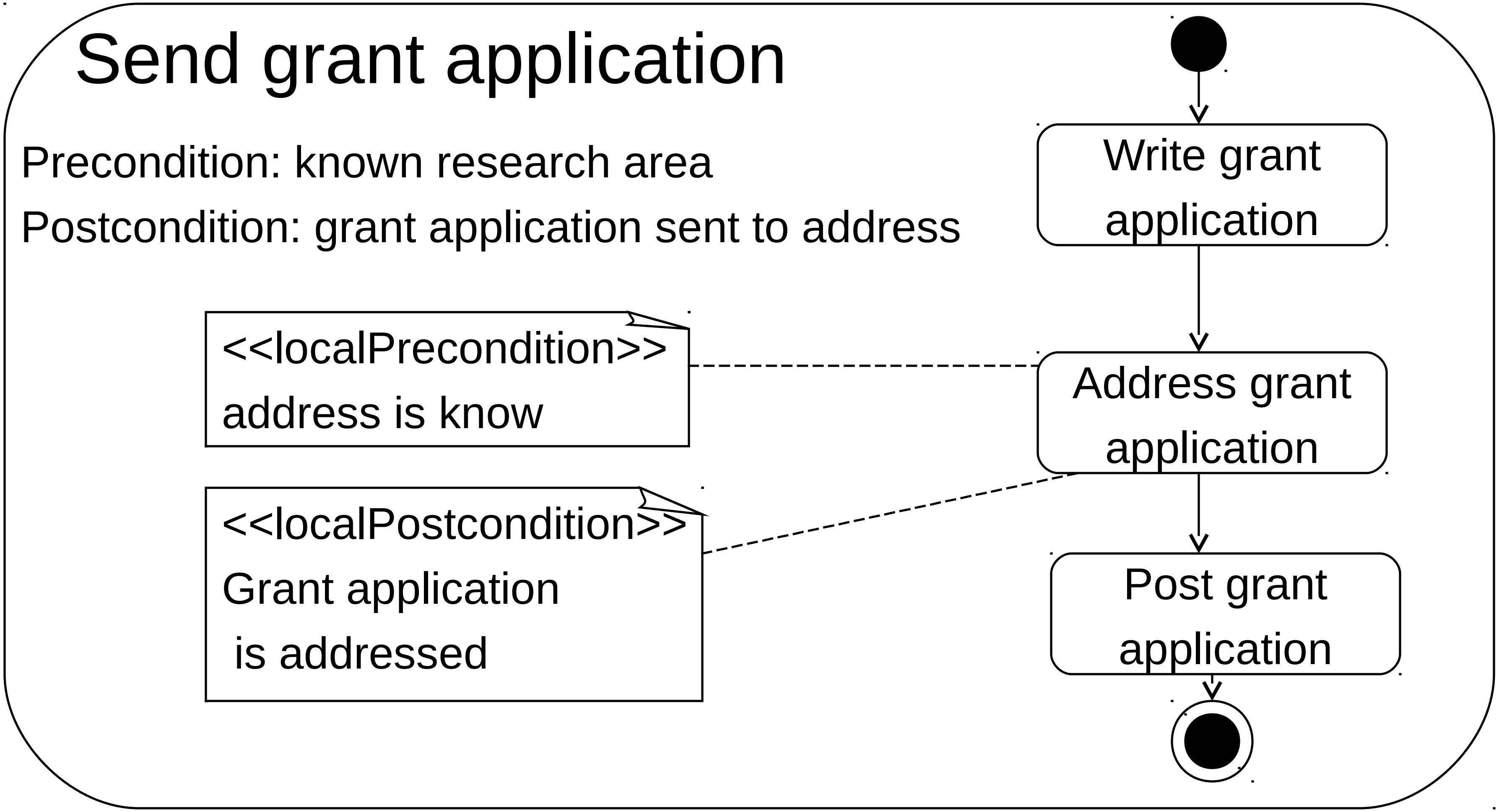
Example: Conditions and Iteration


Coffee machine
Insert coins in the machine
Test if machine has received the right amount of money.
Indicate that beverage can be selected.
Indicate that beverage is not available.
Select beverage.
[beverage not available]
[beverage available]
Dispense beverage.

Decision input



Pre- and Post-condition



Send grant application

Precondition: known research area

Postcondition: grant application sent to address

<<localPrecondition>>
address is know

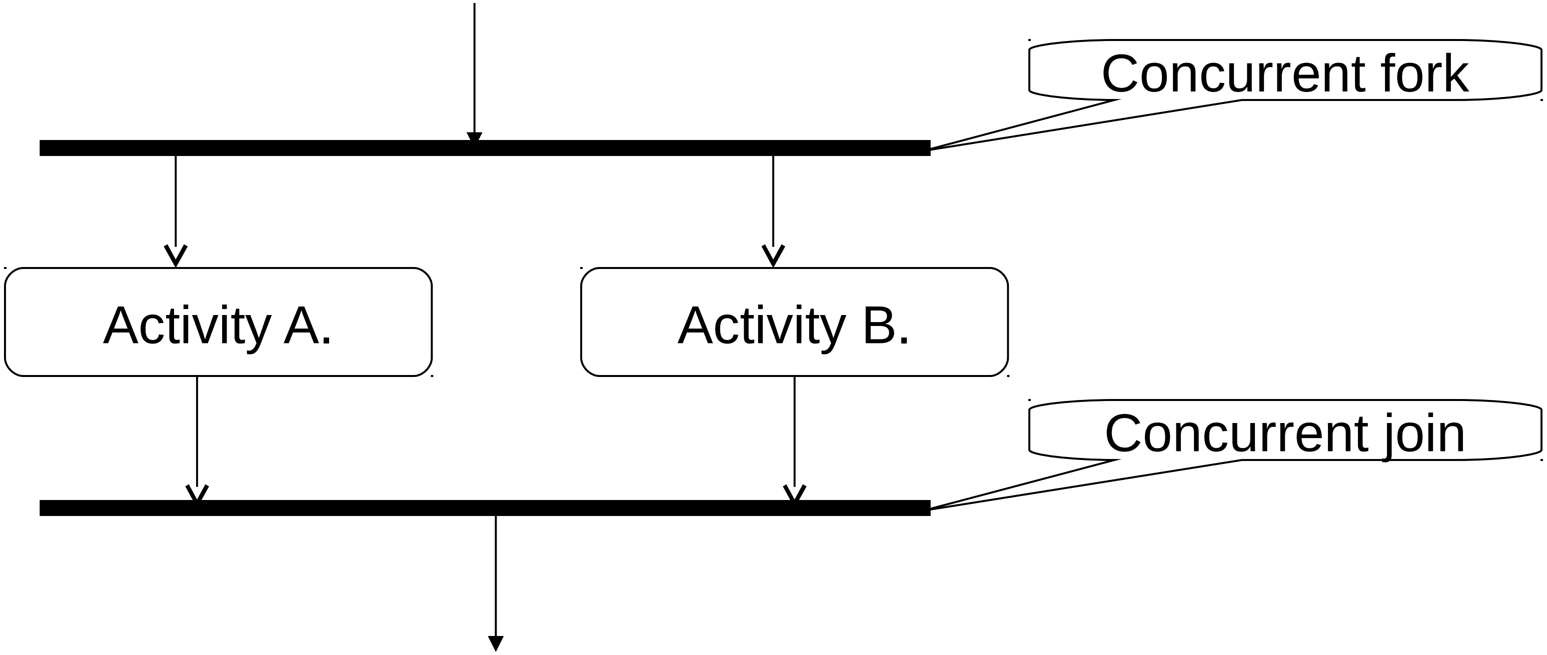
<<localPostcondition>>
Grant application
is addressed

Write grant application

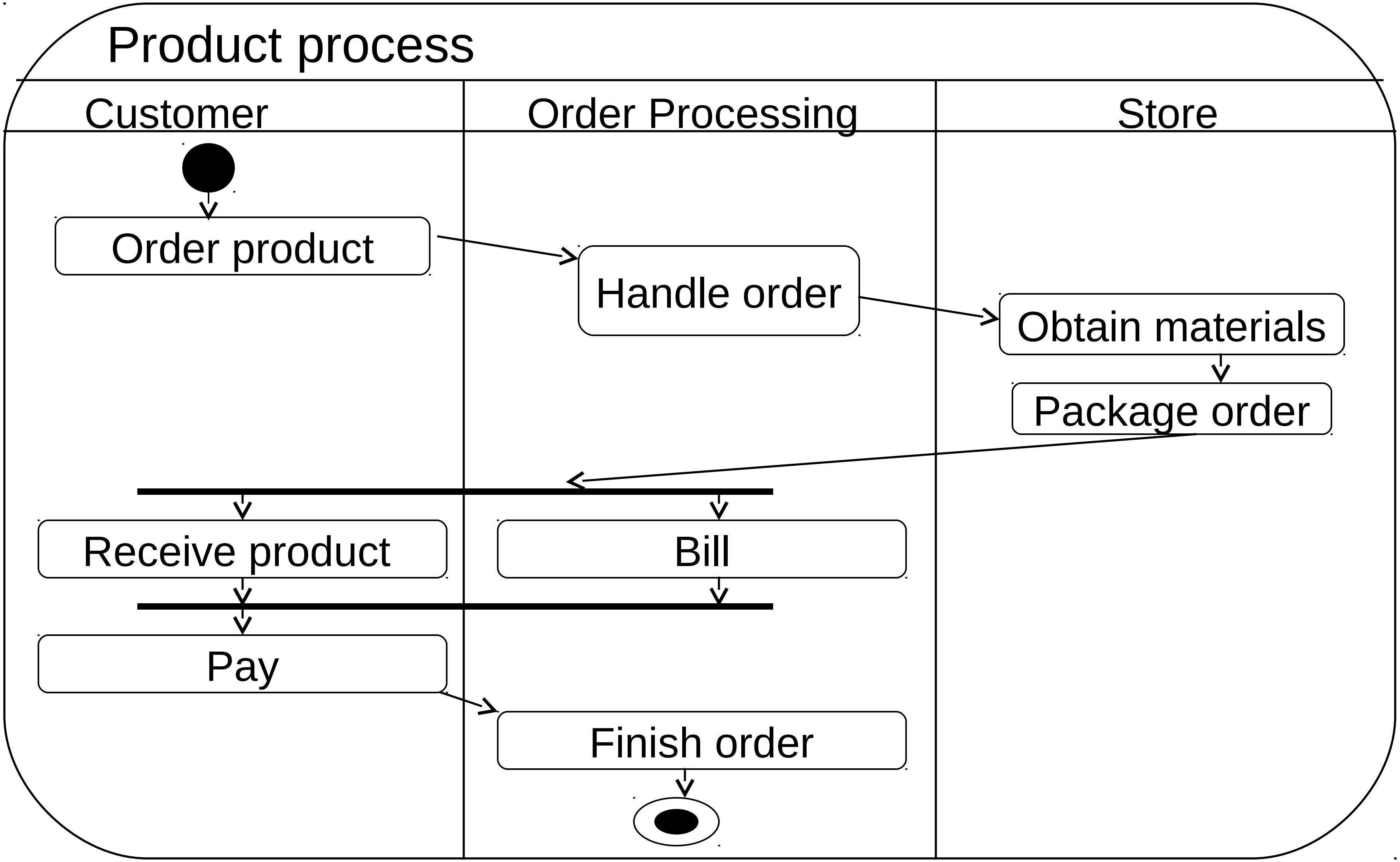
Address grant application

Post grant application

Parallel activities

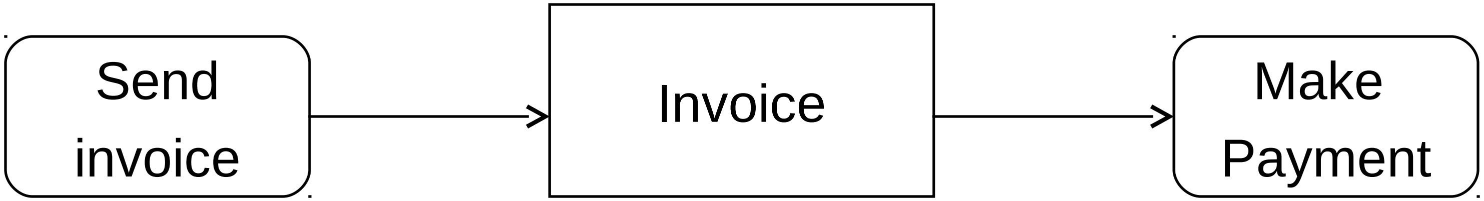


”Swimlanes”

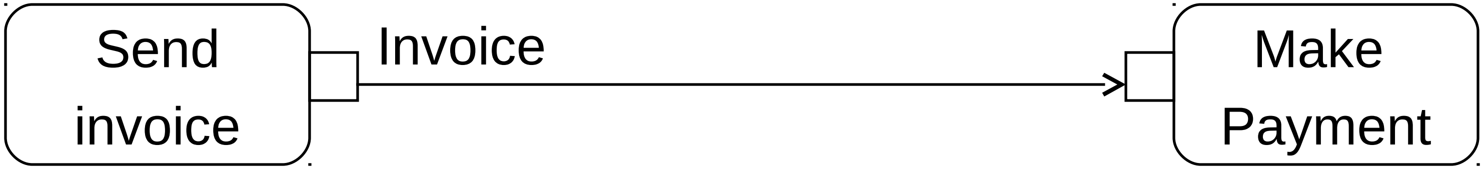

Product process
Customer
Order Processing
Store
Order product
Handle order
Obtain materials
Package order
Receive product
Bill
Pay
Finish order

Object nodes

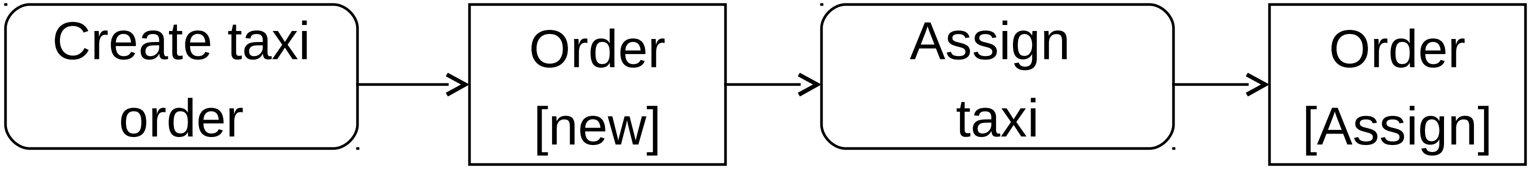
One can send objects:



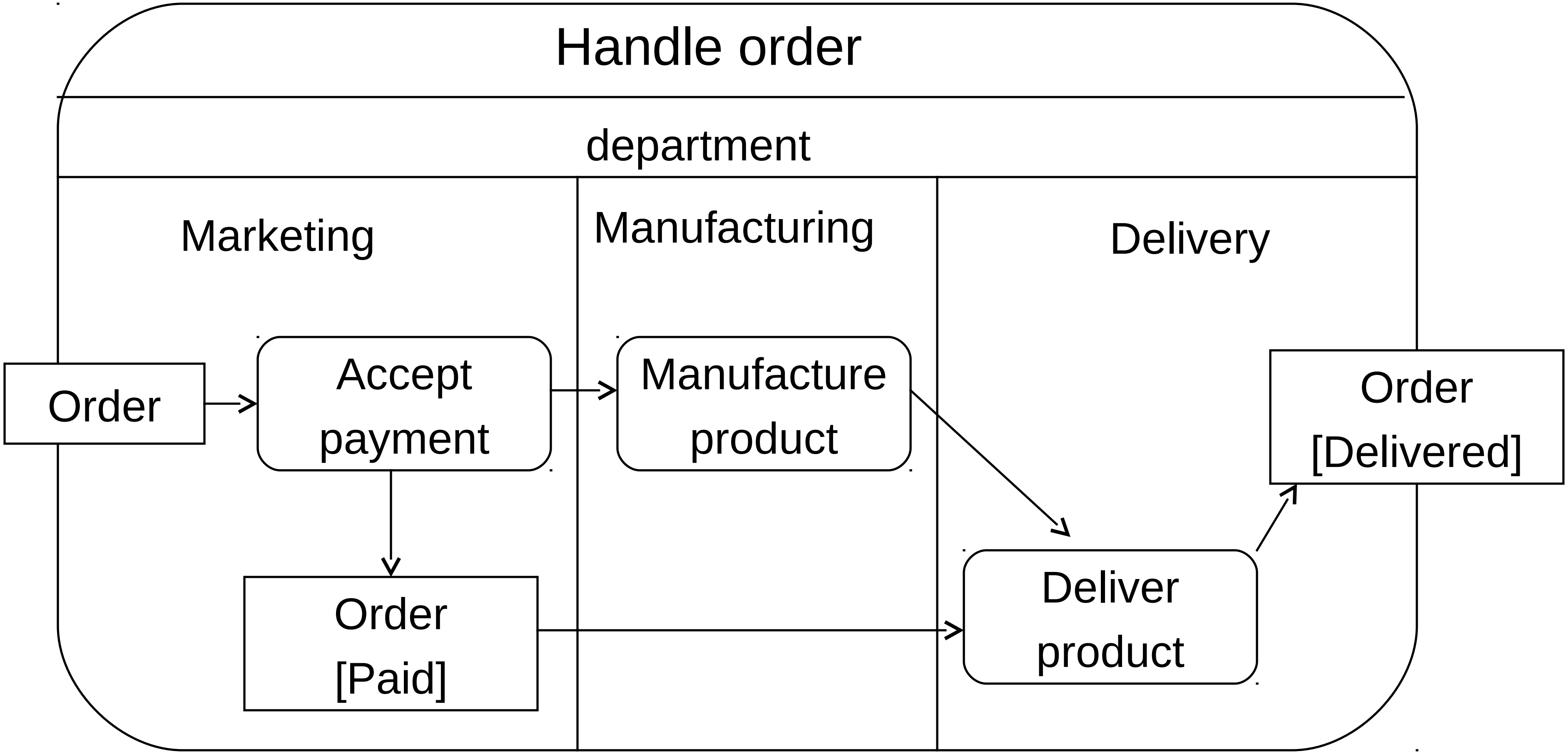
Using pins:



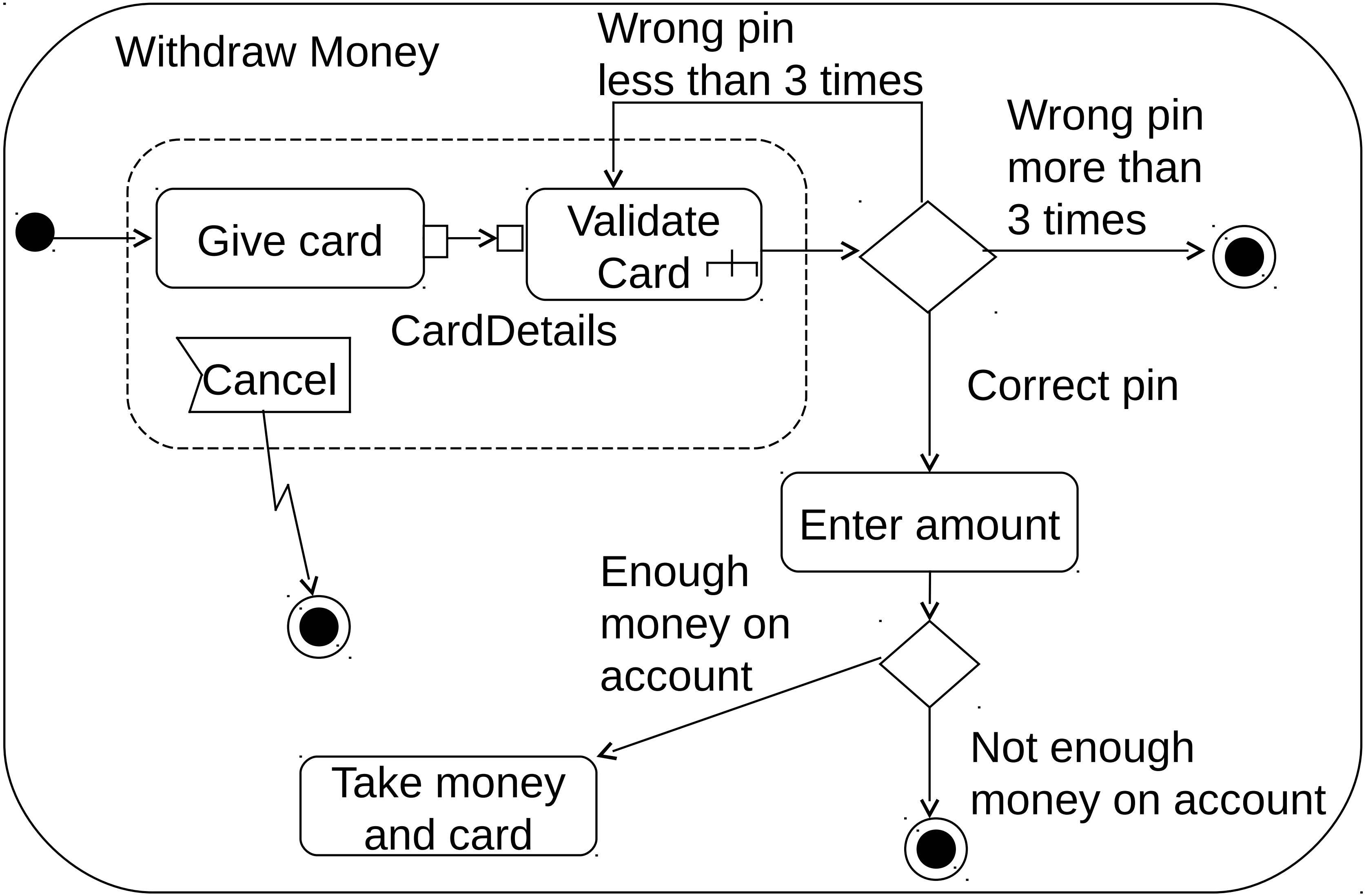
Object in State



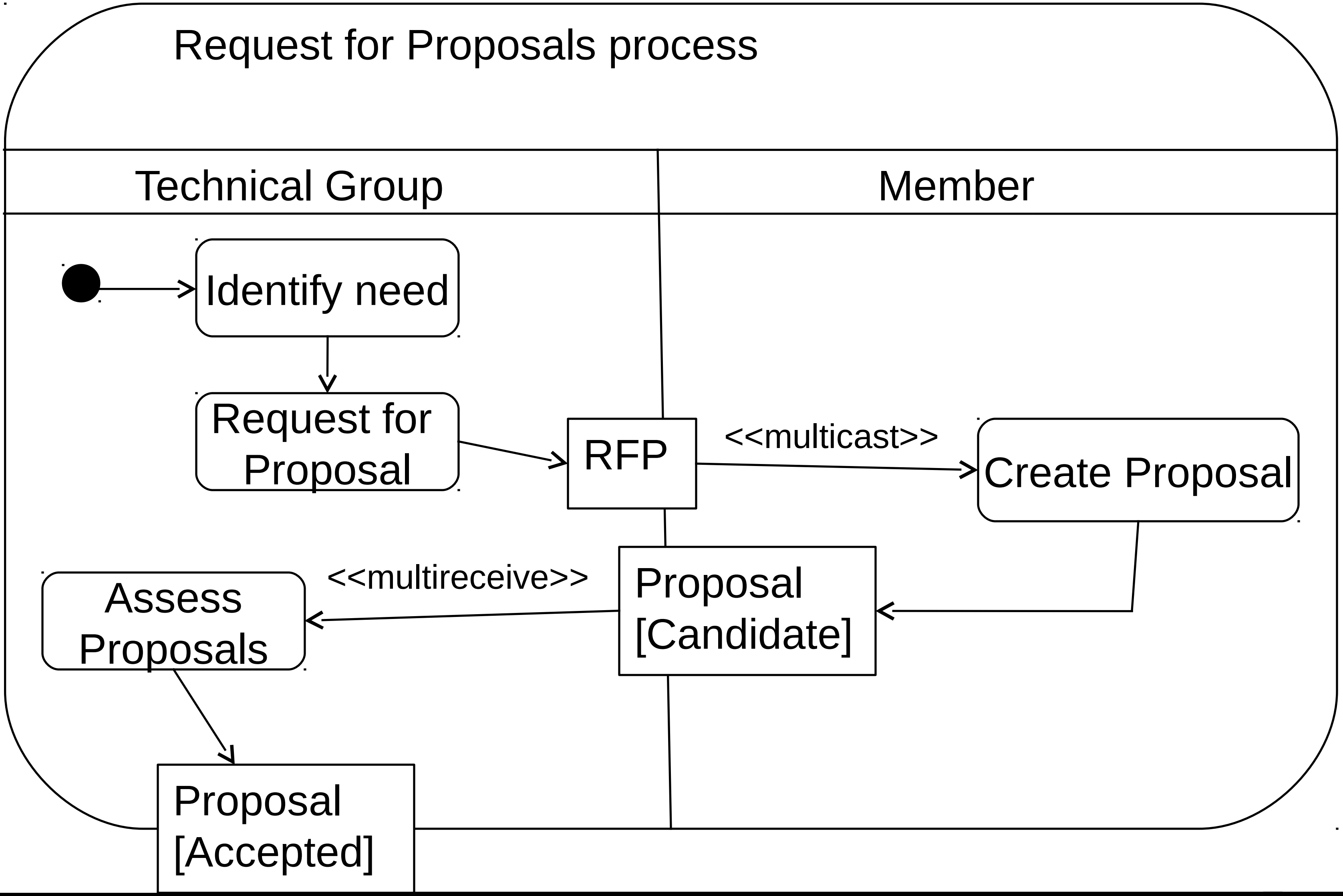
Activity parameters



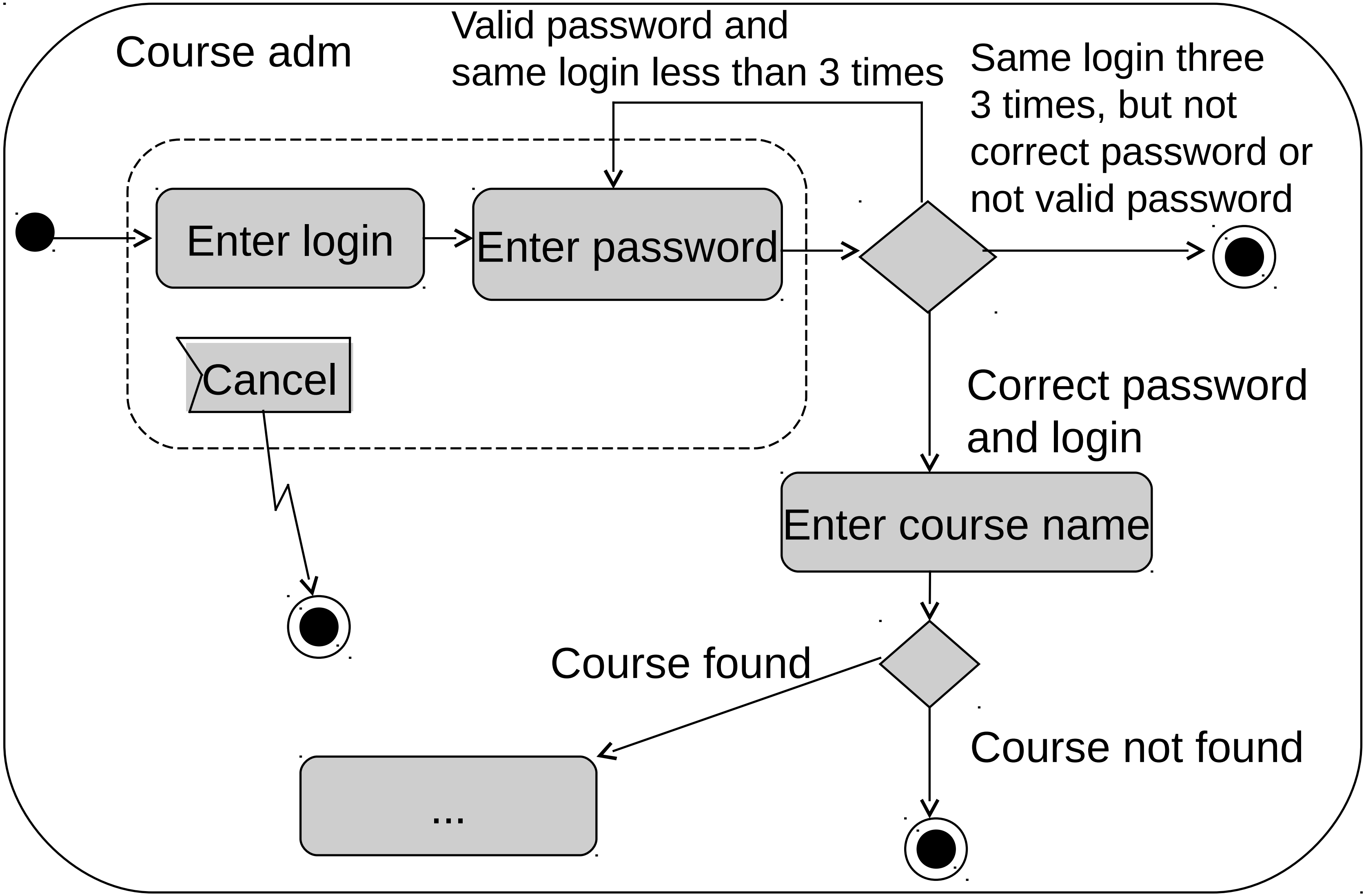
Activity Diagram



Multicast and multireceive

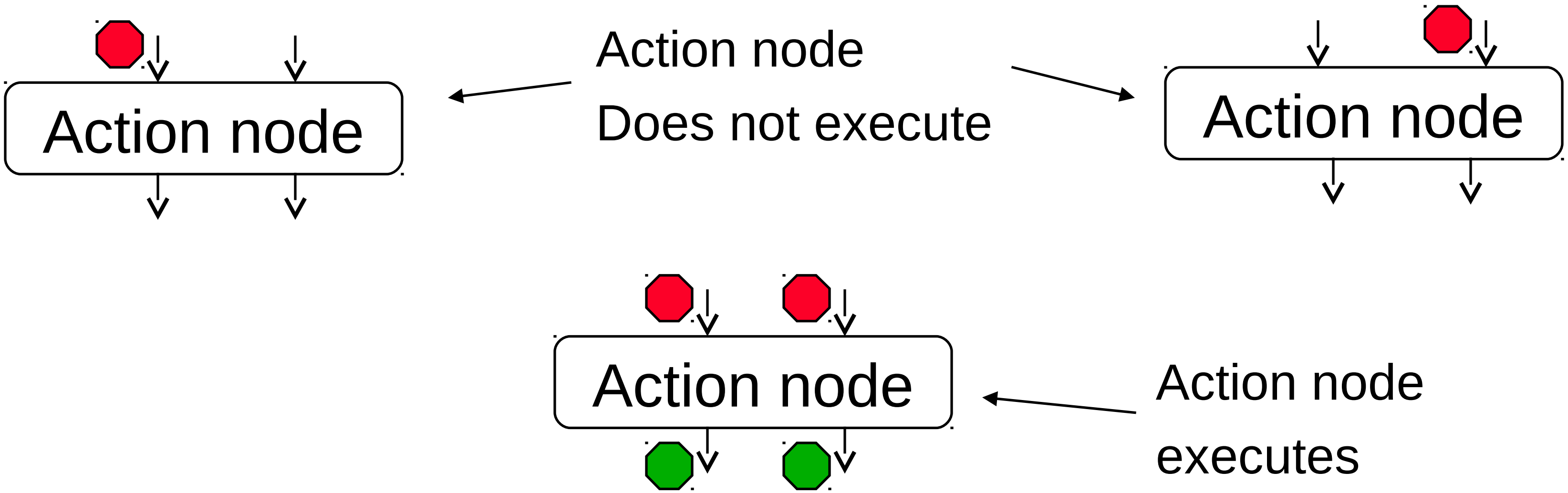


Activity Diagram

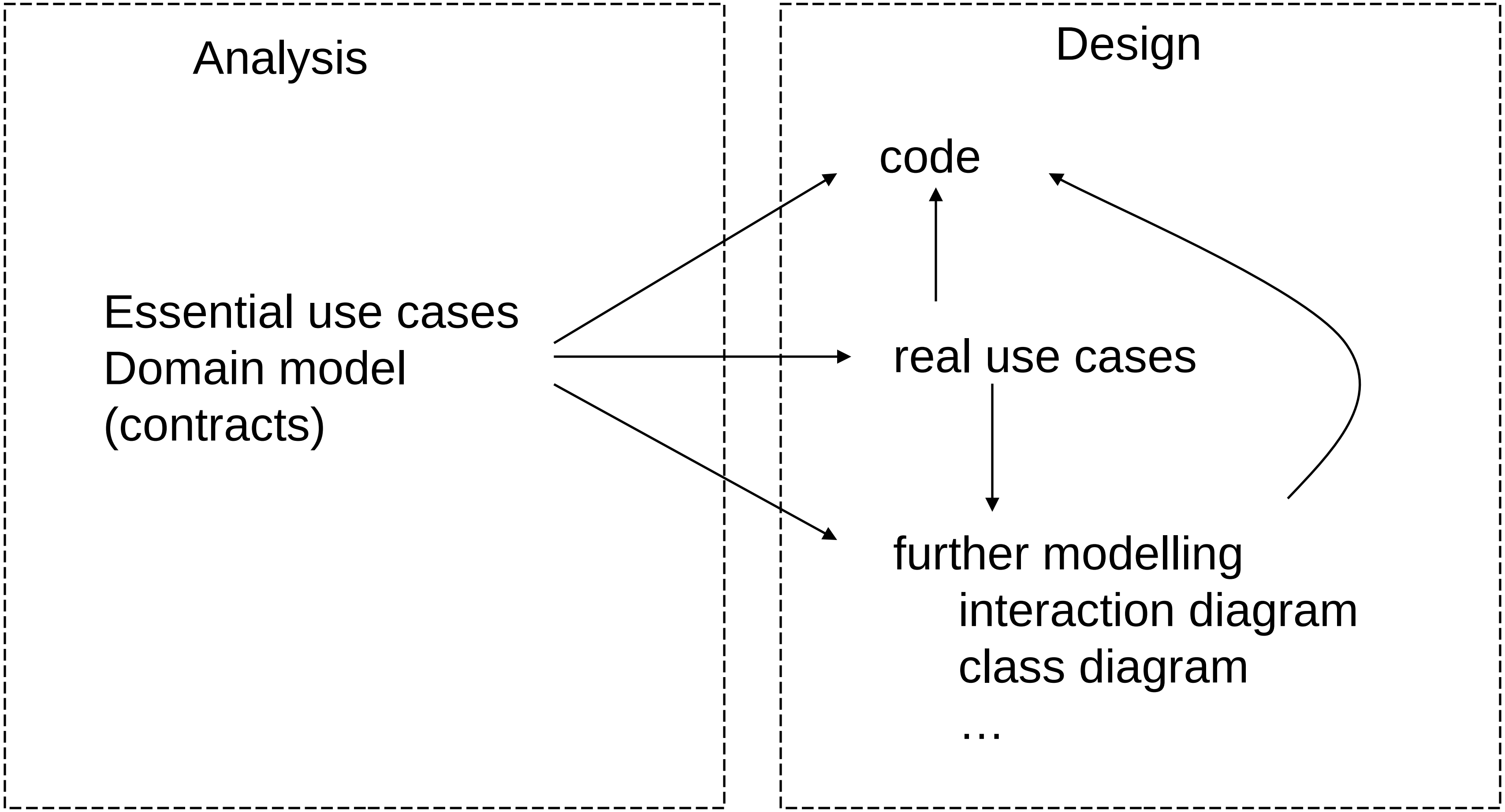


Semantics

- Action nodes execute when
  - There is a token simultaneously on each of the input edges AND
  - The input tokens satisfy all of the local action node preconditions



What next?



Analysis

Essential use cases
Domain model
(contracts)

Design

code

real use cases

further modelling
interaction diagram
class diagram
…

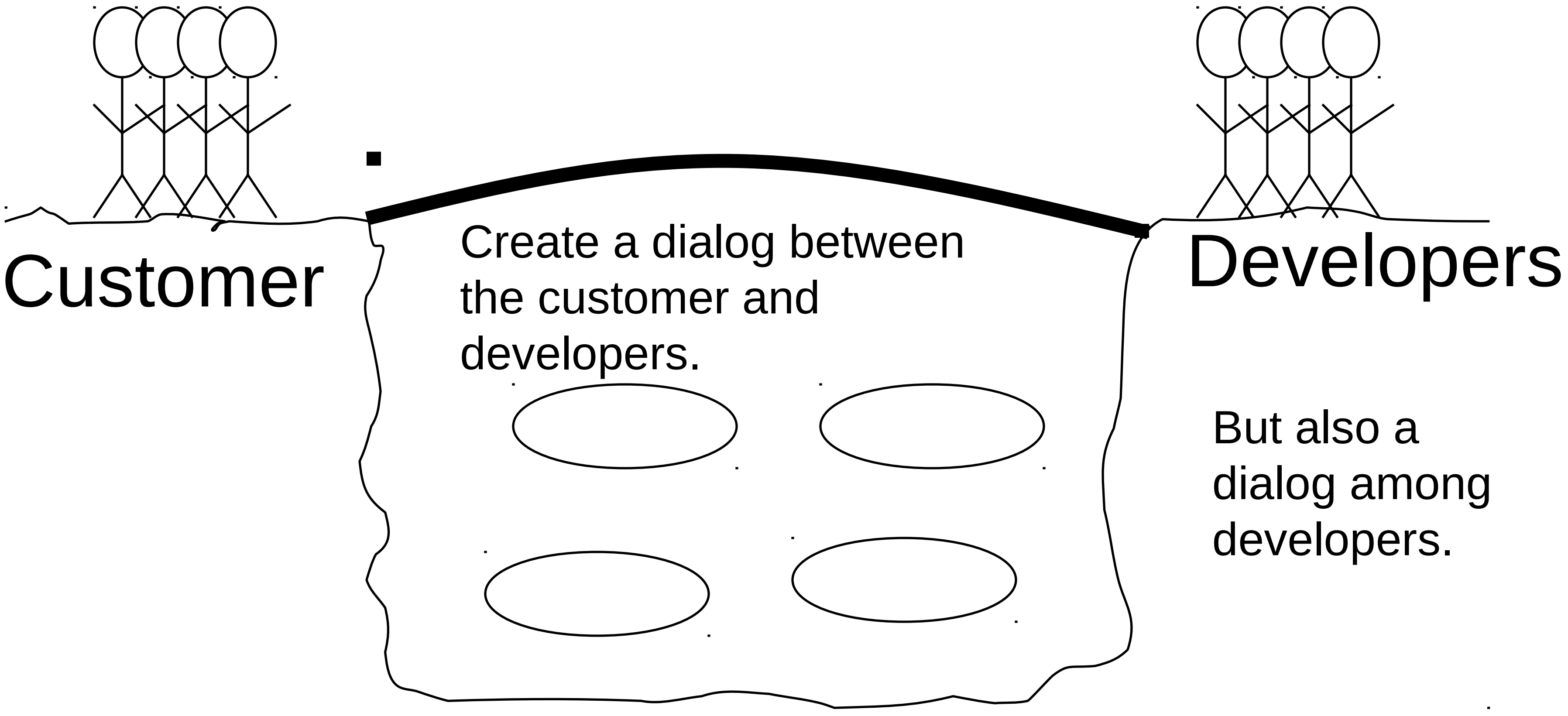
Appendix

Applications of Use Cases

Problem

- Why use Use Cases?
- In order to:
  - Describe the behavior of the system
  - Communicate with the customer
  - Catch functional requirements on the system
  - Obtain the user interface
  - Drive the development process, decide what should be done in each iteration
  - Obtain tests for the system

Communication



Customer

Developers

Create a dialog between the customer and developers.

But also a dialog among developers.

Requirement analysis

- Often these kinds of requirements have to be identified (FURPS+):
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability
  - ”+” represents further requirements

Example: ATM

1. ATM saves information about withdrawals
2. Can be given a code
3. Gives customer amount X of money if customer has at least X on the account.
4. Can be given a card
5. Can return a card when withdrawal is finished or when transaction is cancelled.
6. Can make transactions between accounts
7. Can insert money into the account
8. The amount of money inserted should be added to the account
9. Reduce the account by the amount withdrawn.
10. Can choose an amount.
11. Can choose to withdraw.
12. Check amount on account
13. Can obtain a receipt.
14. Can stop the process of withdrawal.
15. Can give code up to three times.
16. If wrong code three times then the ATM keeps the card.
17. …

Problem

- What is the problem with the list of requirements on the previous slide?
- Problems:
  - How to priorities requirements
  - How to group requirements
  - Often imprecise
  - Hard to obtain an overview
  - Is it a complete list of requirements?
  - Many!!! Can be several thousands.

Different kinds of functional requirements

- Business requirements:
  - A customer shall be able to book a taxi via telephone
- System requirements:
  - The system should estimate the time until a taxi arrives

- Use cases will help to separate these two types of requirements, since we write use cases only for describing system behavior.
- We will obtain system requirements from use cases.

Grouping Requirements

- Requirements can be grouped in several ways
- One way: Use cases
- For example, Withdraw Money relates to the requirements:
  - R1, R2, R3, …
- Implementing a requirement might not make a system more useful; implementing a use case does!
- Use cases tackle the problem of making requirements readable, understandable, and to choose priorities

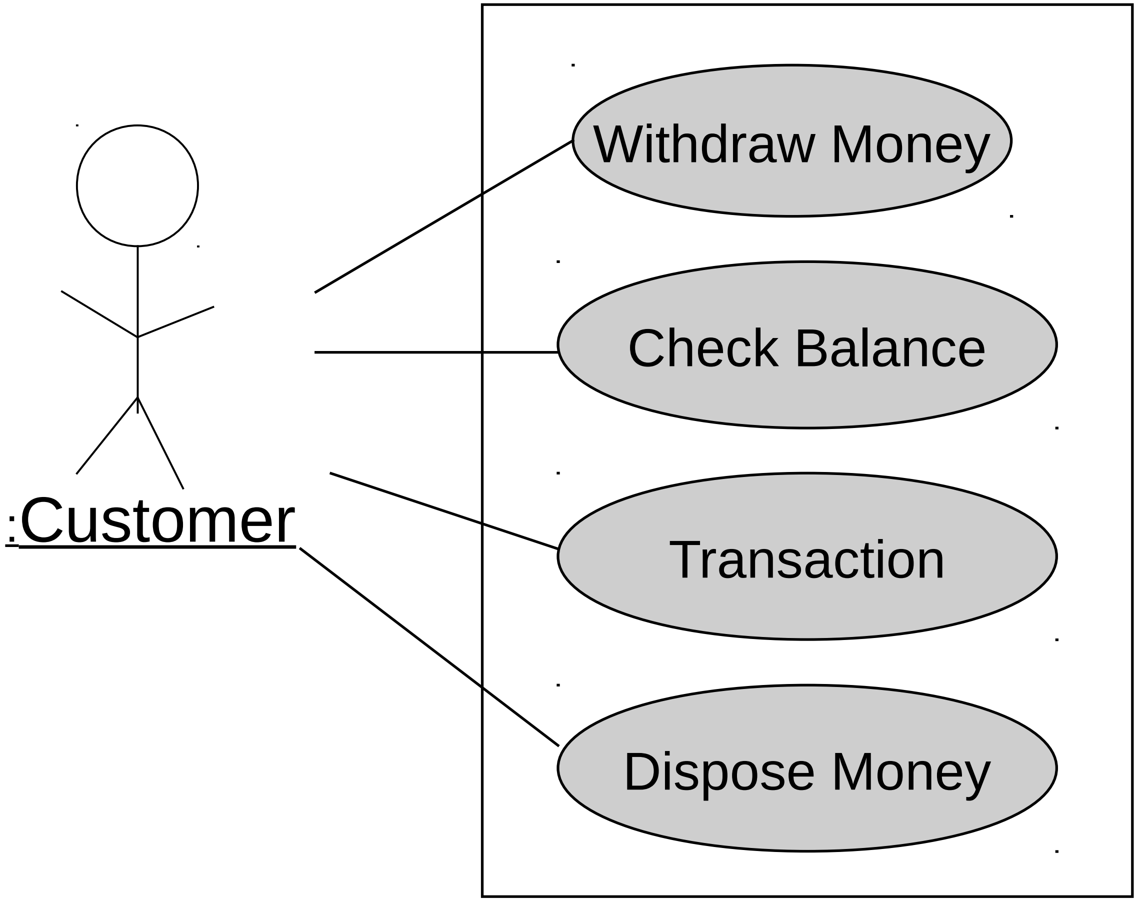
Functional requirements

- Use cases capture most functional requirements.
- But: Some functionality can be ”hidden” in several/all use cases
  - For instance: Logging occurring events

Dealing with Requirements

- Different ways of dealing with functional requirements:
  - Only having a requirement list
  - Only having use cases
  - A combination of both

Example: ATM


:Customer
Withdraw Money
Check Balance
Transaction
Dispose Money

Use Case/Requirement Matrix

	Withdraw Money	Check Balance	Transaction	Dispose Money
R1	X			
R2	X	X	X	X
R3	X			
R4	X	X	X	X
R5	X			

...

Non-functional requirements

- Non-functional requirements are hard to handle by use cases, but sometimes one can relate them to use cases.
- Further documents (apart from use cases) are needed

Example (1)

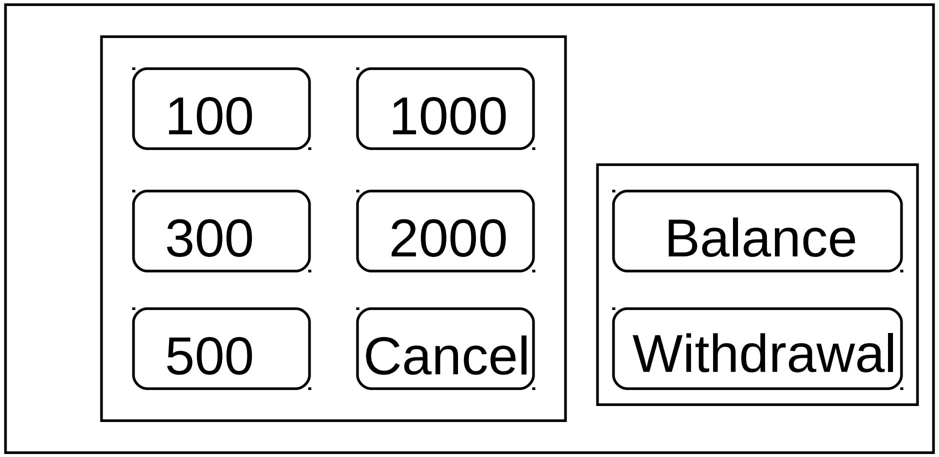
- Usability
  - ATM should be usable for colour blind persons
- Reliability
  - Frequency of failure
    - At most one failure per year (or per 10 sec)
  - Restart after an error
    - When restarting, account balance should be checked against bank to ensure right value (in case of unfinished transactions)

Example (2)

- Supportability
  - ATM system should be adaptable to
    - Different currencies
    - Different languages
    - Different bank computer systems
    - Different card types

User interface

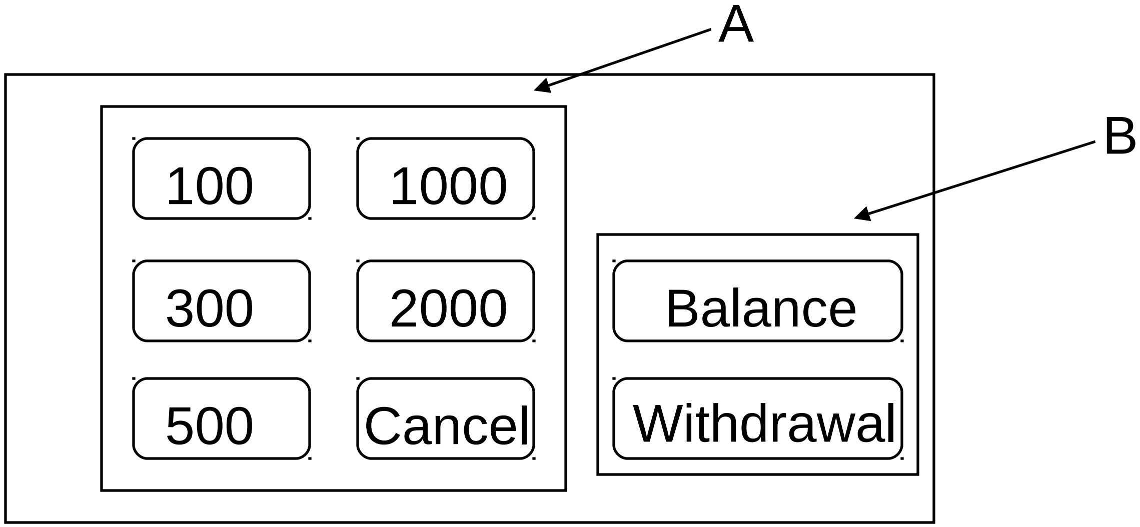
- Usually a ”user interface expert” will derive the user interface from use cases
- For instance:



- … and make a description of the interface

Connecting user interface and use cases

- For instance: ”Customer chooses amount in window A”



- Dangerous: Such use cases are very fragile concerning changes in user interface

Problem

- Should one consider user interface details when writing use cases?
- Should use cases contain information about the interface?
- Should one make the user interface before or after the use cases?

Interface first or last?

- Most people agree that use cases should be written before the user interface is designed
- Exception: Interface can be given, no changes are possible
- (Some people even recommend designing the user interface first)
- Interface is important, because customers might get new ideas by looking at it (less abstract than use cases, easier to understand, things become more concrete and more obvious)

Essential Use Cases

- A use case which abstracts from user interface, implementation details etc.
- Avoids premature design decisions of how to develop the system, such as the look of user interface, whether to use a database etc.

Summary

- We have considered:
- How to write brief use cases
- How to write complete use cases
  - How to write main flow
  - How to write alternative flows
  - How to write post-conditions
  - …

Role play

- To illustrate one flow through a use case, one can use a concrete case.
- One can play the interaction between the system and the actor.
- One person plays the system and for each actor there is a person playing the actor.