
UNIFIED
MODELING
LANGUAGE™
UML®

Lecture 2,3

Use Case Modelling

Vasco Amaral
MDS 2019/2020

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA
Departamento de Informática

千里之行，始於足下

Models

… and Model-Driven Development

Models as first-class citizens

What is a model?

Models as first-class citizens

What is a model?

"A model is an abstraction of a (real or language-based) system allowing predictions or inferences to be made."

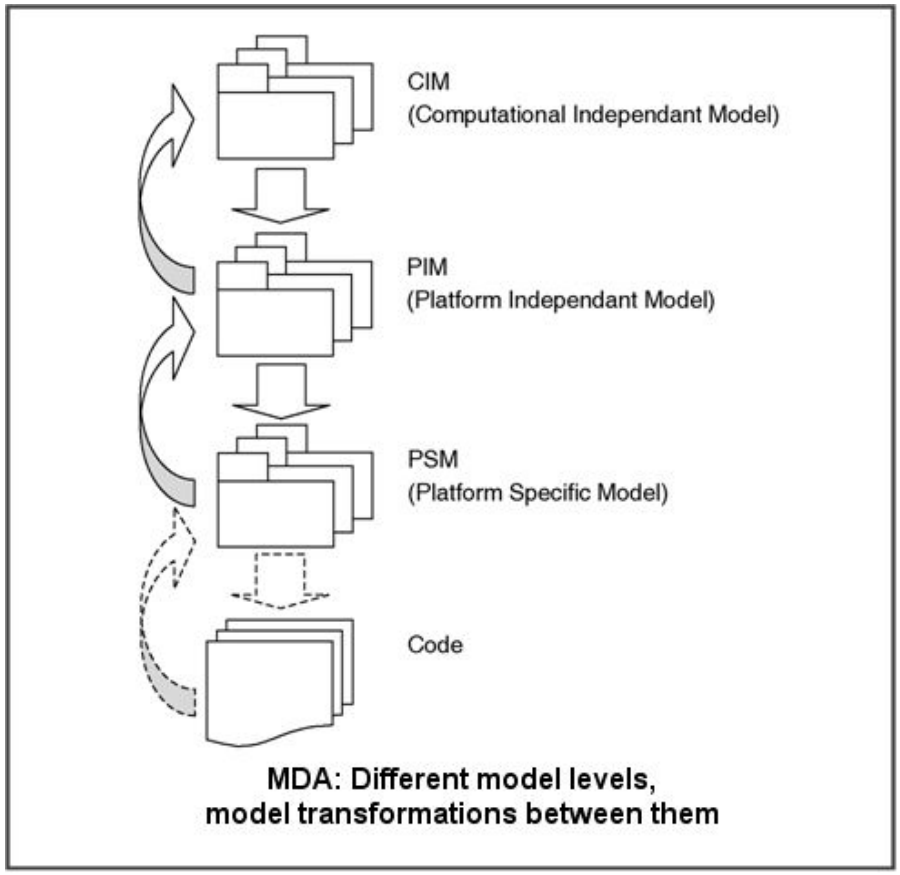
Thomas Khune, Matters of (Meta-) Modeling, SoSYM, 2006

Models as first-class citizens

Visual Models are excellent means for documenting, and more...

Model-Driven software - OMG's view of MDD

Promotes the use of models (and automatic translations) from requirements to code as a way to document and capture essential information for the perspective of the systems (at an adequate level of abstraction)



MDA: Different model levels, model transformations between them

UML

Unified Modeling Language

UML

- General purpose modelling visual modelling language
- Supports the development life cycle, from requirements to implementation
- Matches the Unified Process, but supports other engineering processes
- Systems modelling for several domains, including embedded, real-time and management and decision-support systems
- Mostly associated with Object Oriented modelling, but goes beyond it and is language and platform independent
- Includes a set of best practices

- Note: UML is **NOT** a methodology

Objects and the UML

- UML models software intensive systems as collections of objects
- Two cohesive structures are used:
  - Static structure – describes object types and their relationships
  - Dynamic structure – describes the objects' life cycles and the interactions among those objects so that system requirements are satisfied

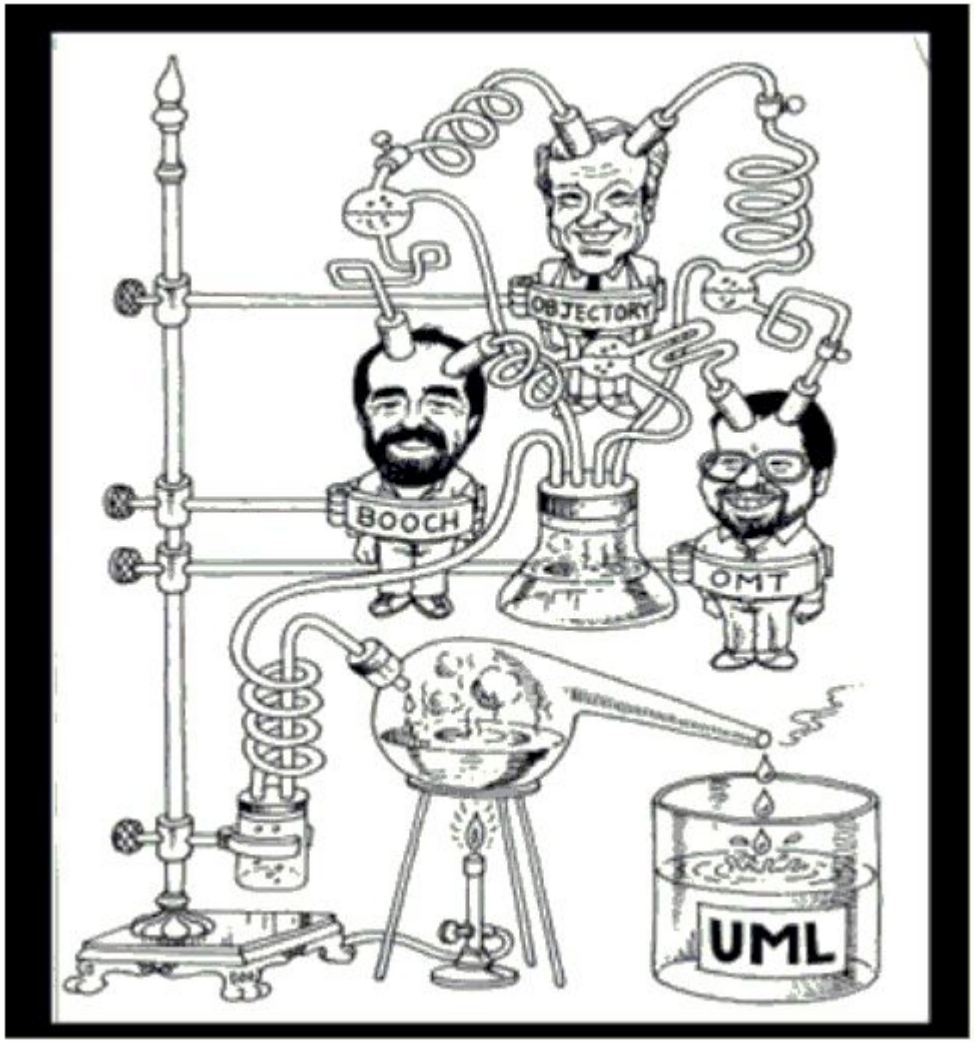
UML basic elements

- Things (model elements and model names)
- Relations (how to relate things)
- Diagrams (views of the model)

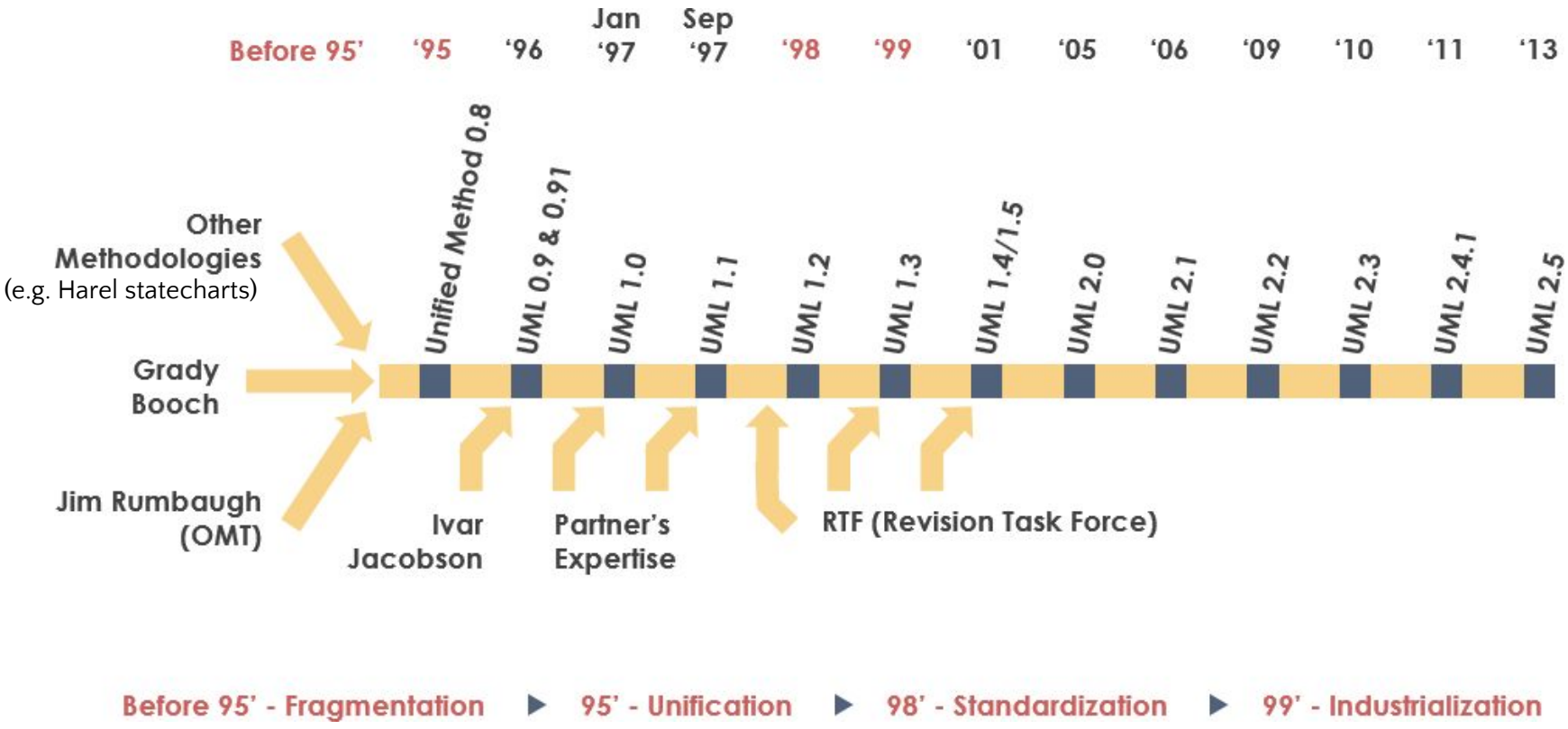
UML Creators – the three amigos

Ok, ok, not those three amigos… these guys! 😊

Each of the three amigos had his own approach

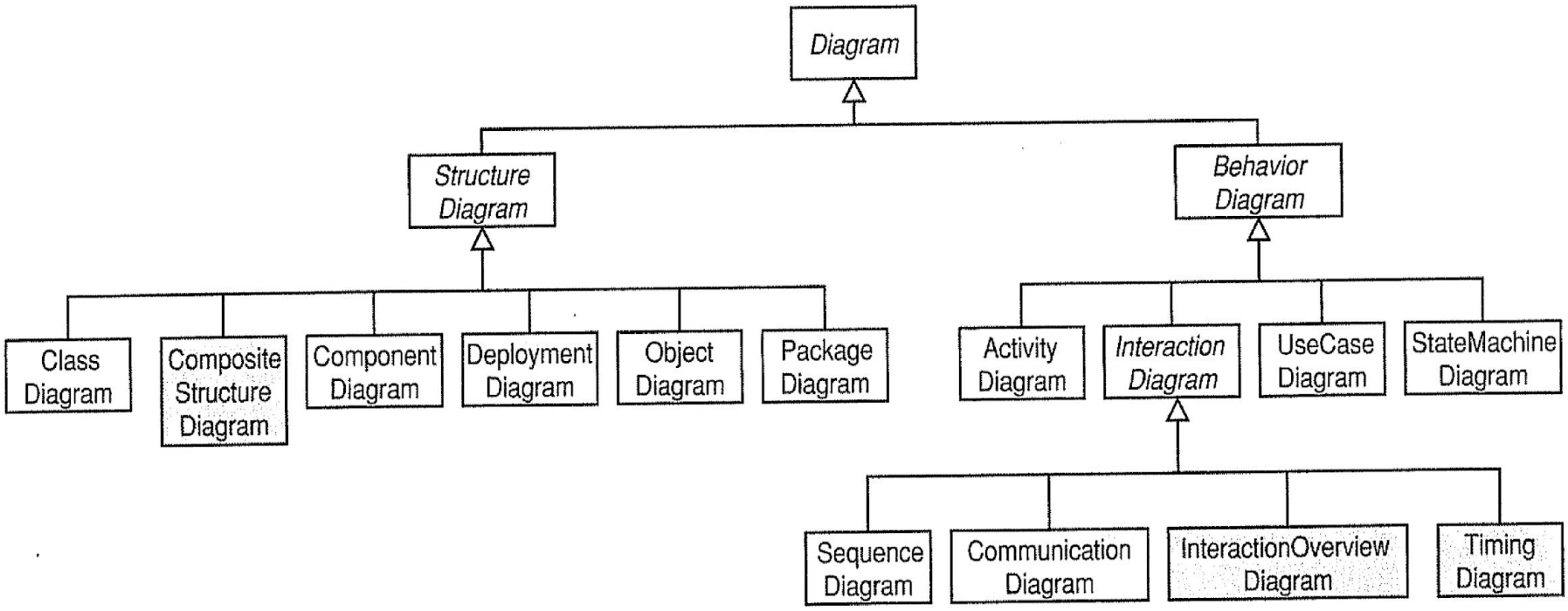


UML history



Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

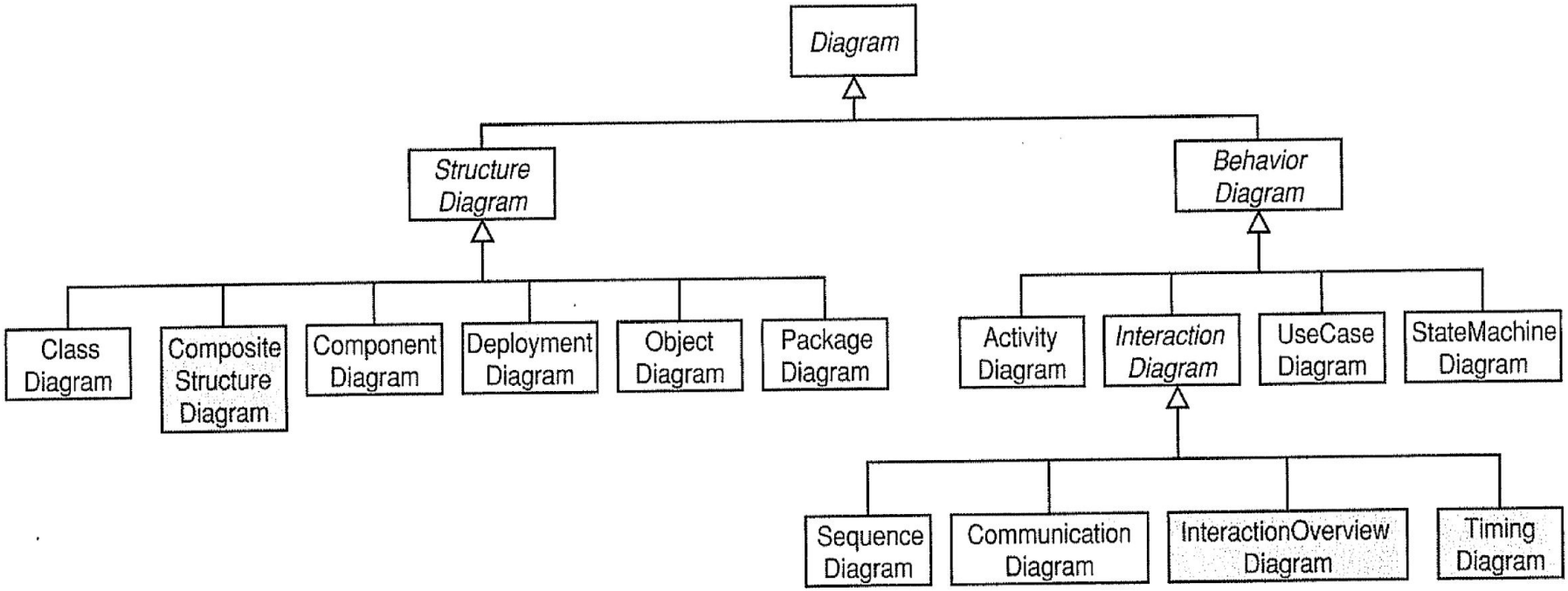
UML Diagrams



Requirements Analysis

Use case models

Use Case models



Use case models

- Sequence of interactions of outside entities (known as actors) with the system
- System actions that yield observable results of value to the actors
- A use case describes how the system will be used.

Use cases

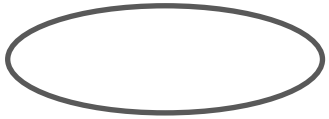
- Describes **how the system will be used**
- Describes the essence of the system **from the perspective of actors**
- This essence is captured **via use case diagrams** that illustrate it visually
- Further details are captured in **a textual definition**
- Means of communication between software engineers and other stakeholders (namely customers and end users)

Use case model: system, actors and use cases

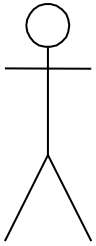
- System decomposition into use cases and their corresponding actors
  - Use cases represent functional requirements
  - Actors are external entities (human or device) that play a role in the system
  - Relations define which actors play which roles
- Building a use case model
  - Use case diagram
  - Detailed description of each use case.

Use cases: basic notation

System boundary



Use case



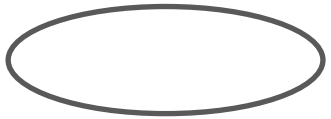
Actor

Is the actor of

System boundary

A box drawn around the use cases to denote the edge, or boundary, of the system being modelled.
This is known as the subject, in UML

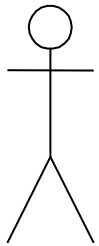
Use case



- A thing an actor can do with the system
- "Dialog" specifying a sequence of transactions between an actor and the system
- Describes system usage scenarios, **always started by an actor**
- Focuses on a complete functionality, including an optimistic vision, and error situations

- Use cases are gathered during requirements engineering and are used as input for the analysis, design, implementation, and system testing

Actor



- Actors are external entities (humans or devices) that play a role in the system
  - An entity can play more than one role
  - More than one entity can play the same role
- An actor may appear in more than one use case

Relationship

- Identifies which actor plays a role in a given use case

Can you identify the actors?

Consider a system for recycling bottles and cans where a customer may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the client wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the client.
The system is also used by na operator that, in the end of the day, requests a list of the returned items. The operator may also update system information.

Clearly, the client is an external entity

Consider a system for recycling bottles and cans where a client may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the client wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the client.
The system is also used by an operator that, in the end of the day, requests a list of the returned items. The operator may also update system information.

The **operator** is also an external entity

Consider a system for recycling bottles and cans where a **client** may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the **client** wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the **client**.
The system is also used by an **operator** that, in the end of the day, requests a list of the returned items. The **operator** may also update system information.

Bottles, cans , items, receipts, and the system itself are NOT actors

Consider a system for recycling bottles and cans where a client may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the client wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the client.
The system is also used by an operator that, in the end of the day, requests a list of the returned items. The operator may also update system information.

Which use cases can you identify here?

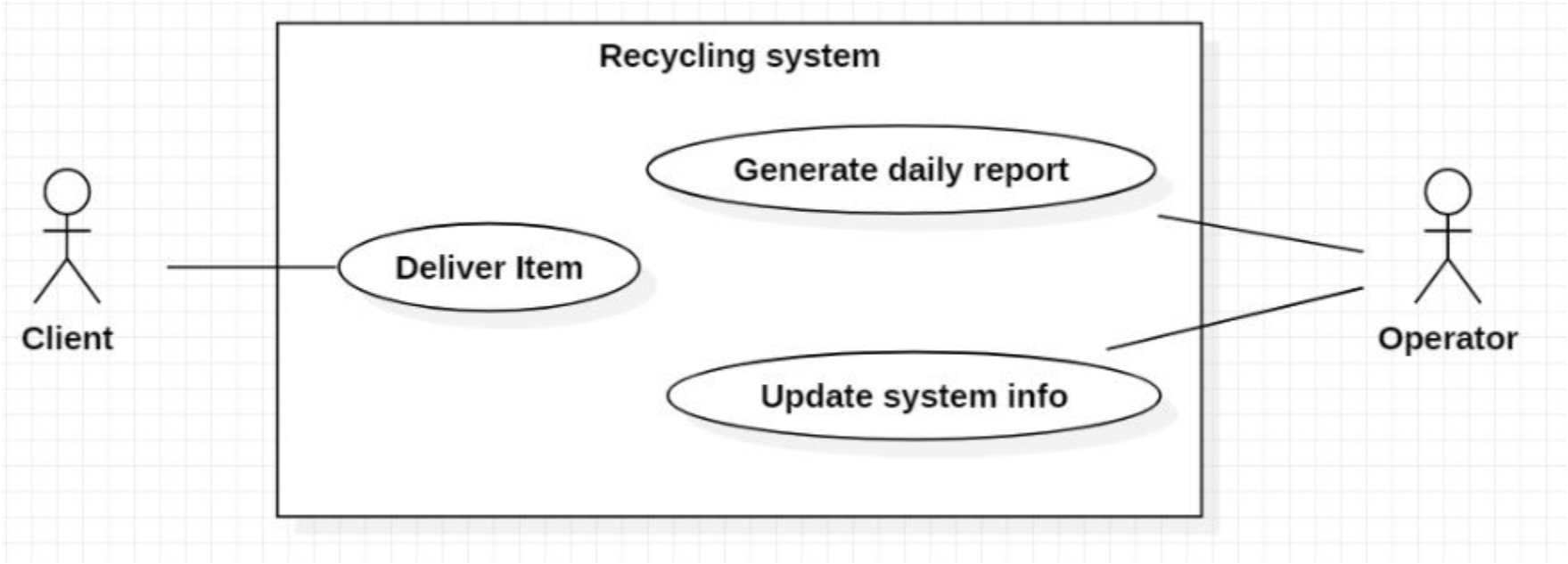
Consider a system for recycling bottles and cans where a client may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the client wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the client.
The system is also used by an operator that, in the end of the day, requests a list of the returned items. The operator may also update system information.

Which use cases can you identify here?

Consider a system for recycling bottles and cans where a **client** may deliver used bottles and cans and receive a payment for those. As each item has different dimensions and is worth a different value, the system has to identify the kind of item it has just received. The system registers the number of received items and, if the **client** wishes so, the system may print a receipt with the number and kind of received items, as well as the individual prices of each item and the total value paid to the **client**.
The system is also used by an **operator** that, in the end of the day, requests a list of the returned items. The **operator** may also update system information.

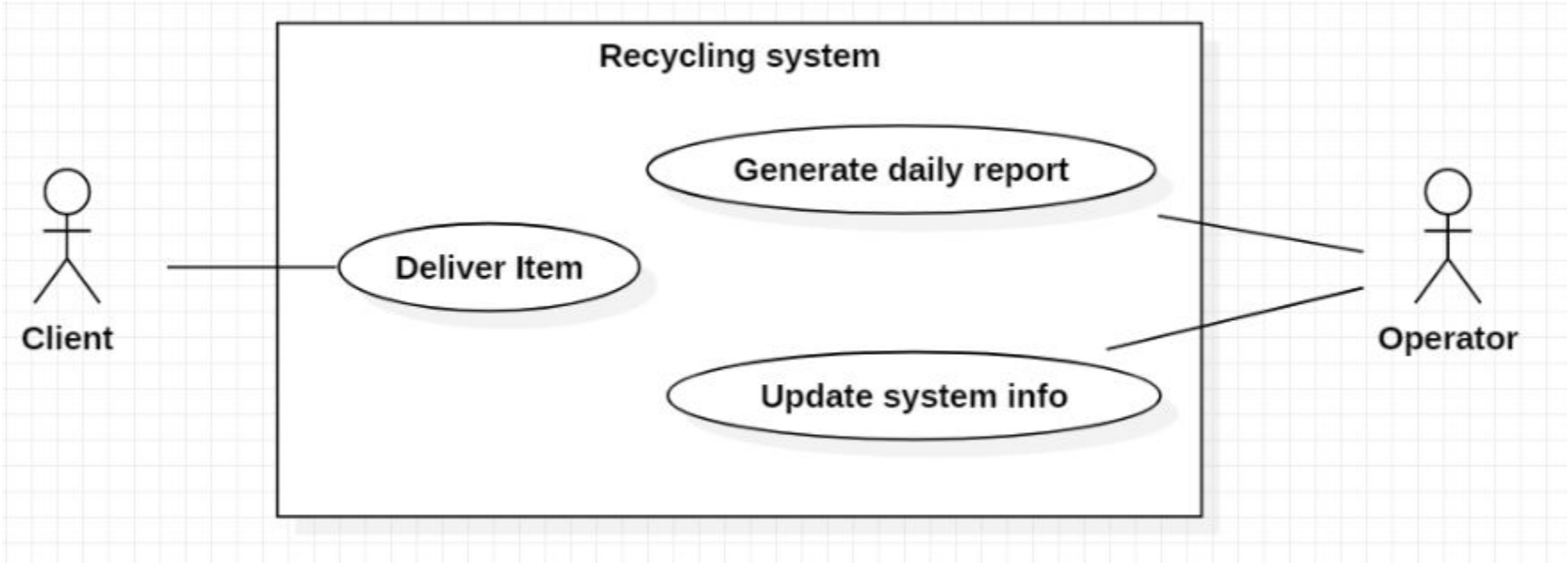
Actors: Client

- Deliver items (bottles and cans; this includes the whole process until the client receives money for them)



Actors: Operator

- Generate a daily report with the list of returned items
- Update system info



Use case specification (to complete)

Each use case must be described with more details.
Example for the ***Deliver Item*** use case:

- Deliver item starts when the client wants to return bottles or cans. For each item inserted into the machine, the system increments the counter of items delivered by this client and the counter of items of that type delivered by the client. The system also updates the daily total for items of that type. When the client has no more items to deliver, the client asks for a receipt by pushing a receipts button. The system does all the necessary computations and, for each type of delivered item, computes its price and number of returned items. A printer prints these, one by each line, in a receipt. It also computes and prints the total amount to return the client. Finally, the system delivers the receipt to the client, which is returned to the client. Whenever…
  <fill this with error situations and how to handle them>

Use cases are simple (well… they can actually be tricky)

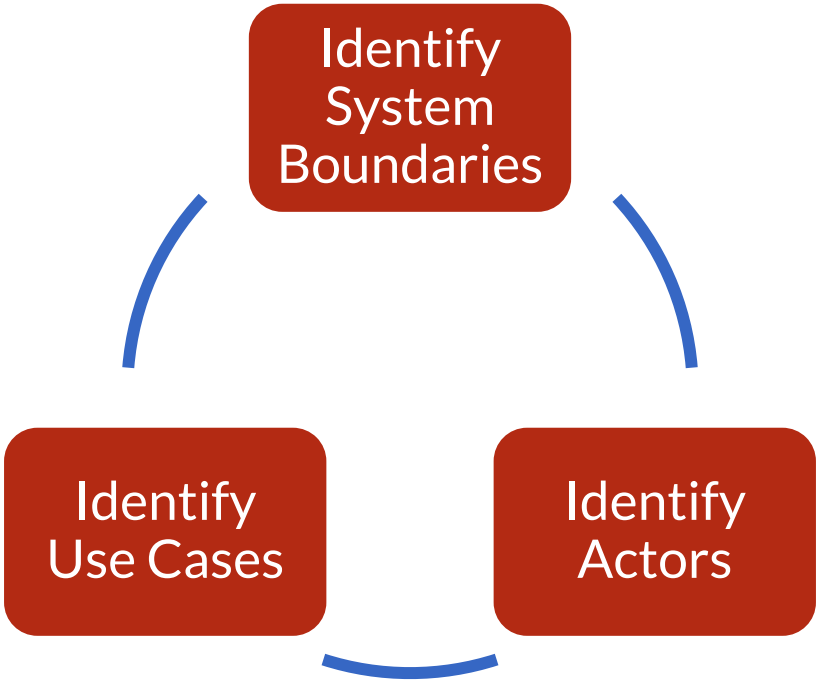
- Use cases are visually simple and an effective way of communicating with other stakeholders
- The tricky part is identifying the interesting ones and describing them correctly
- The use case model may be the single most important model you create
    - Fail here, and you won't really know what your systems is supposed to do, and what it is not supposed to do
    - "千里之行，始於足下", Laozi 😊
    - "If you don't know where you are going, you'll end up someplace else." Yogi Berra

Use case model

- Specifies the system boundaries and captures the functionality the system to be should offer
- May be a deliverable from the development team to the customer
  - It must be understandable even to non-UML experts
- It is a stepping stone for the next phases of the system development
  - It is structured by the analysis model

Use case modelling process

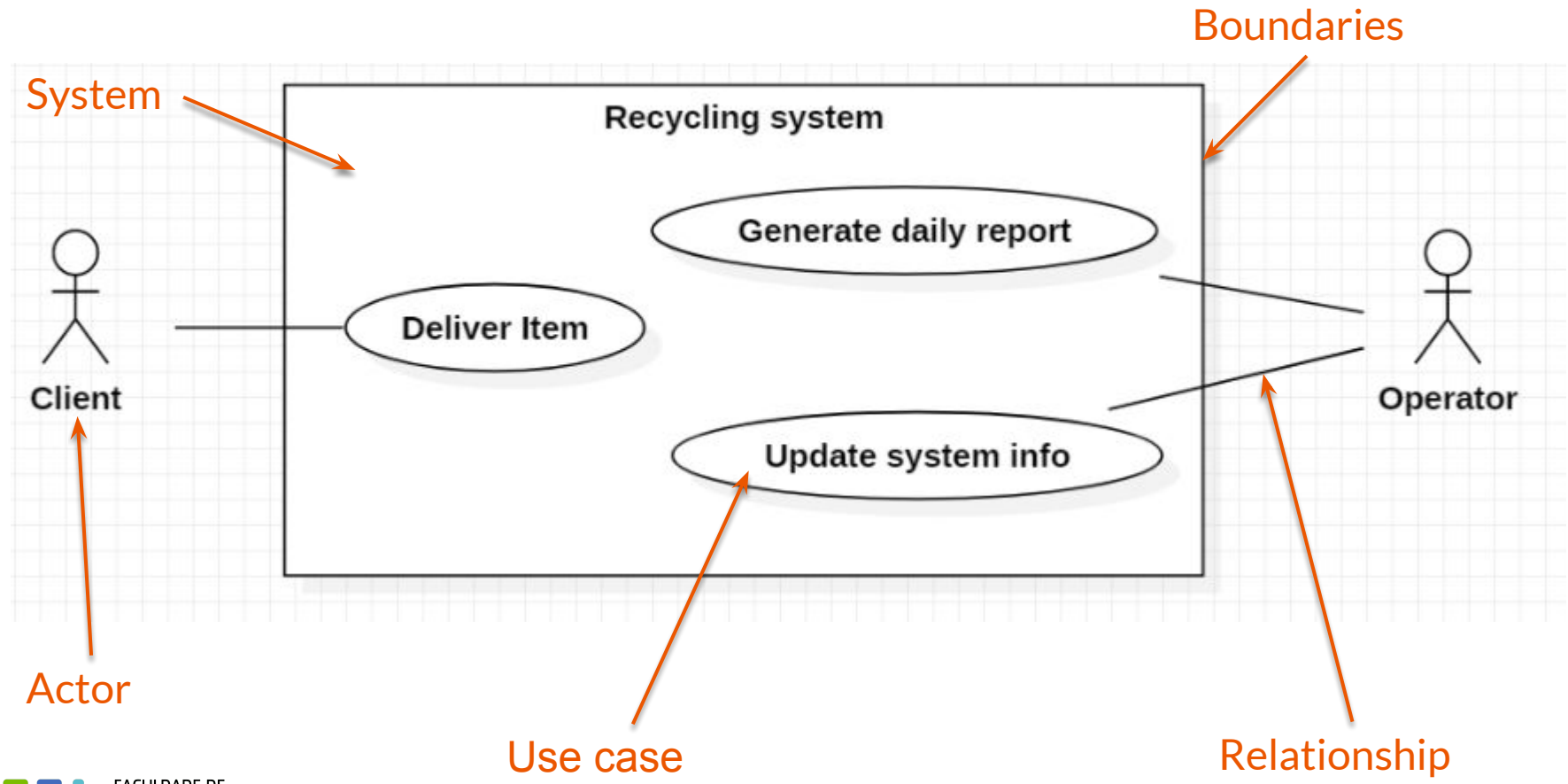
Process



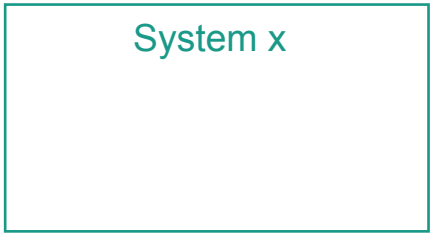
Process outcomes

- System boundaries
- Actors
- Use cases
- Relationships between use cases and actors

Use case diagram syntax (partial)



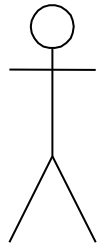
System boundary (aka subject) definition



- It is essential to define the system boundaries
  - What is part of the system?
  - What is NOT part of the system?
- System represented as a rectangle
- Actors represented outside of the system
- Use cases represented inside the system

This may look trivial, but failing the system boundaries definition may lead (**and often does lead**) to a project failure! Boundaries definition has a tremendous impact in the definition of system functional and non-functional requirements.

Actors (revisited)



- An actor specifies a role which an external entity may play while interacting directly with the system
- All actors interact with the system. If they don't, they are not actors!
- Actors may be:
  - People
  - Other systems
  - Other devices

Do not mistake the role an entity plays in a system with the entity itself. The same role may be played by several different entities.

When identifying actors, ask these questions

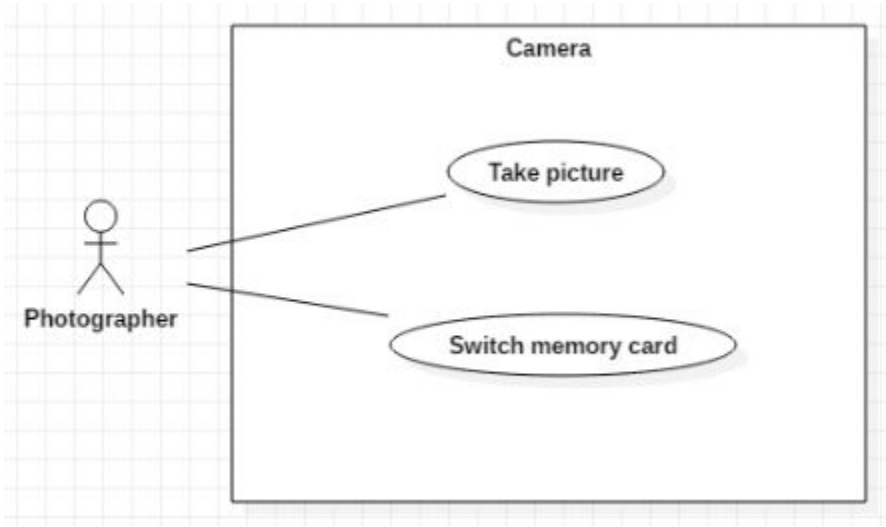
- Who (or what) needs the system to conduct his activities?
- What role does the candidate actor play in the interaction with the system?
- Who is interested in the results of the system?
- Who is responsible for administering the system?
- With which other systems is this system supposed to communicate with?
- Who supplies information to the system?

Sanity check when modelling actors

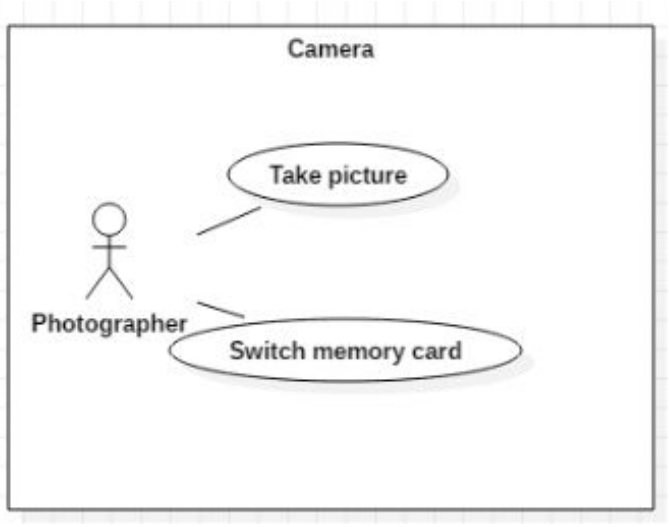
- Actors are always external to the system
- Actors interact directly with the system
- Actors represent a role in the system
- Actors do not represent someone, or something specific within the system
- Someone, or something, may represent more than one role in the system – in that case, he/it is represented by more than one actor
- An actor must always have a name and a short description of his role from the perspective of the business
- Time may be represented as an actor if there is a periodic event triggered by the passage of time.

Actors are always outside the system

Correct

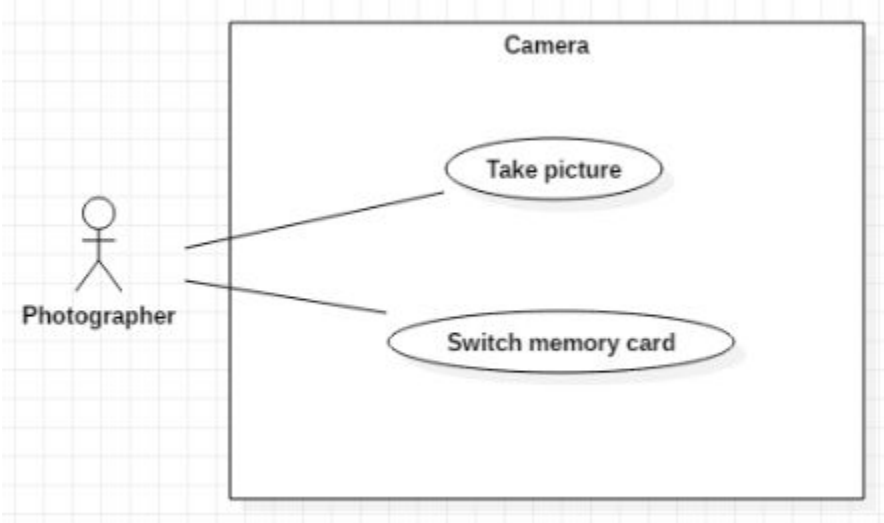


Incorrect

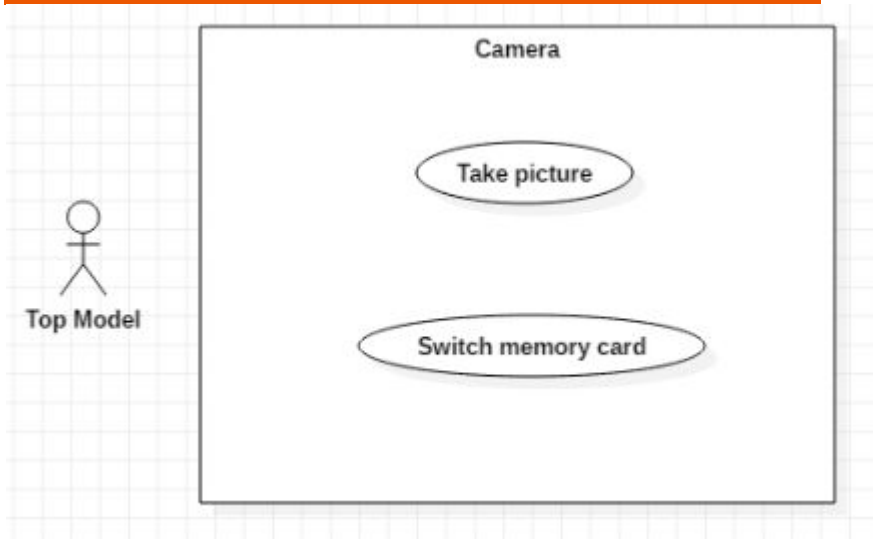


Actors must always participate in at least one use case. Otherwise, they are not really actors!

Correct

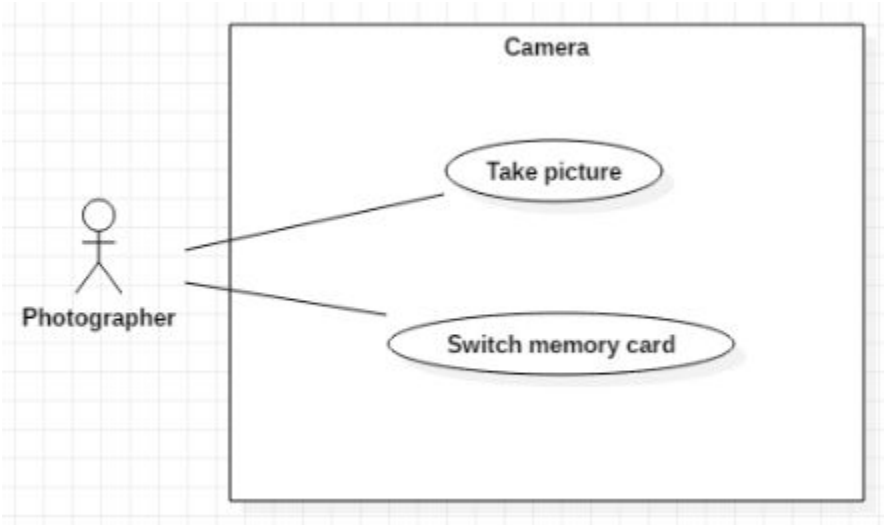


Incorrect

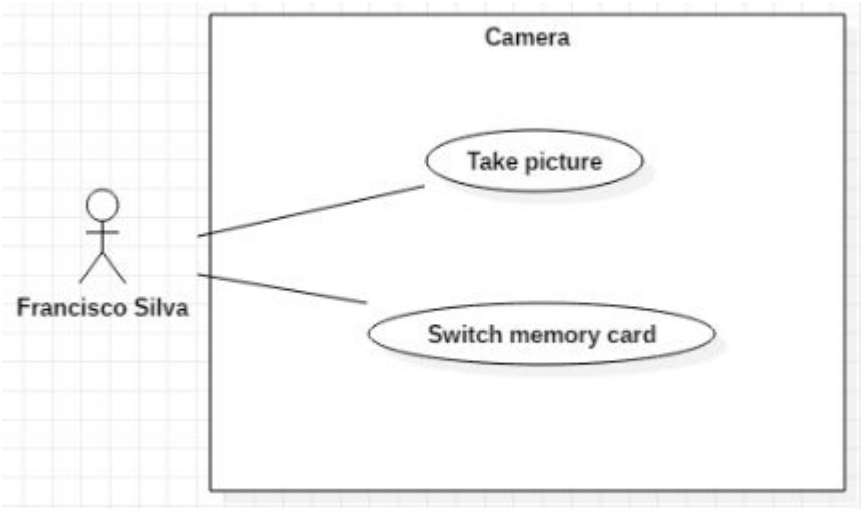


Actors represent a role in the system, rather than specific people, or specific things

Correct

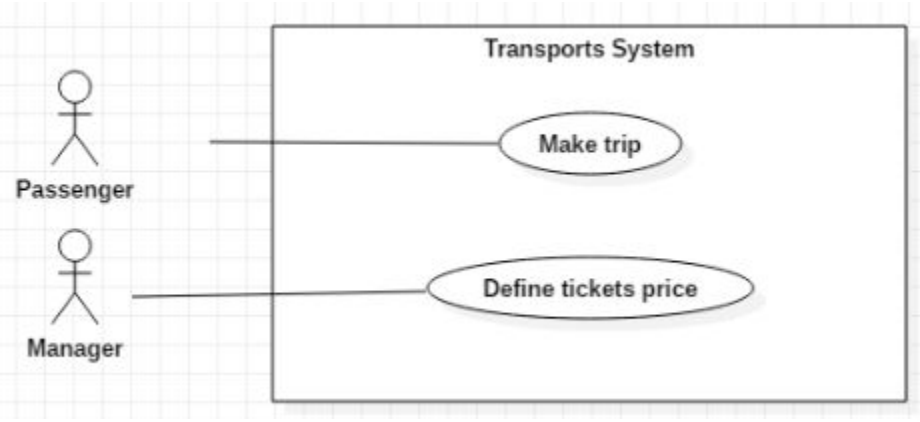


Incorrect

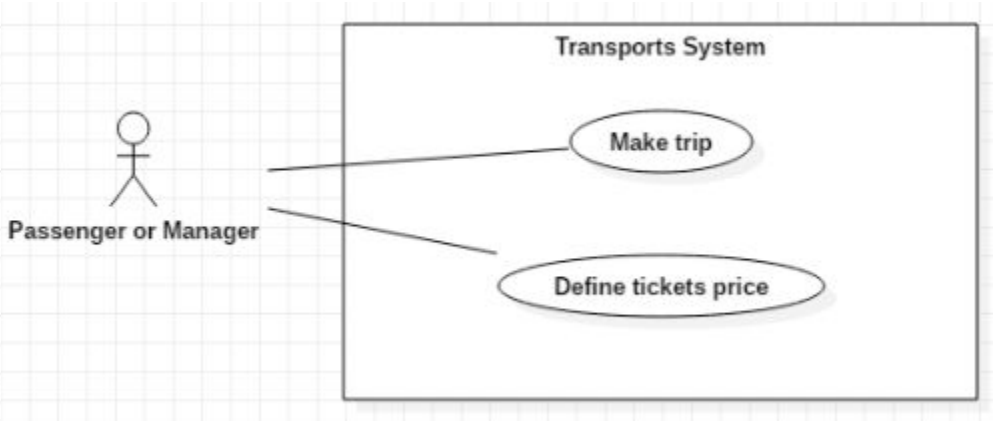


A person, or thing, may play more than one role, thus being represented by more than one actor

Correct



Incorrect



The actor name must make sense from a business perspective

Correct

Name: Passenger
Description: Public transportation passenger

Incorrect

Name: Person
Description: Person who moves between several different places in town to take care of some businesses which are totally irrelevant for our system, but that for some really stupid reason we decided to describe here anyway. Just because we can.

The actor must have a short description that describes it from a business perspective

Correct

Name: Passenger
Description: Public transportation passenger

Incorrect

Name: Person
Description: Person who moves between several different places in town to take care of some businesses which are totally irrelevant for our system, but that for some really stupid reason we decided to describe here anyway. Just because we can.

Time can be an actor too!

- Sometimes, we need to model events that occur in the system in a well specified moment in time, but which do not seem to have been triggered by any particular actor
- As it turns out, they were triggered by the passage of time
- If that is so, then we should introduce a special actor: Time

Example:

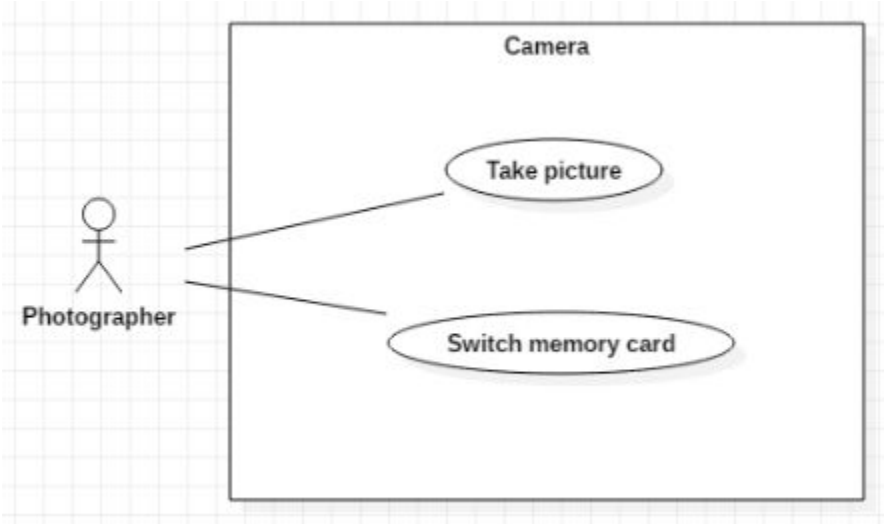
Every month, the email system deletes all the email messages classified as spam

How can we identify use cases?

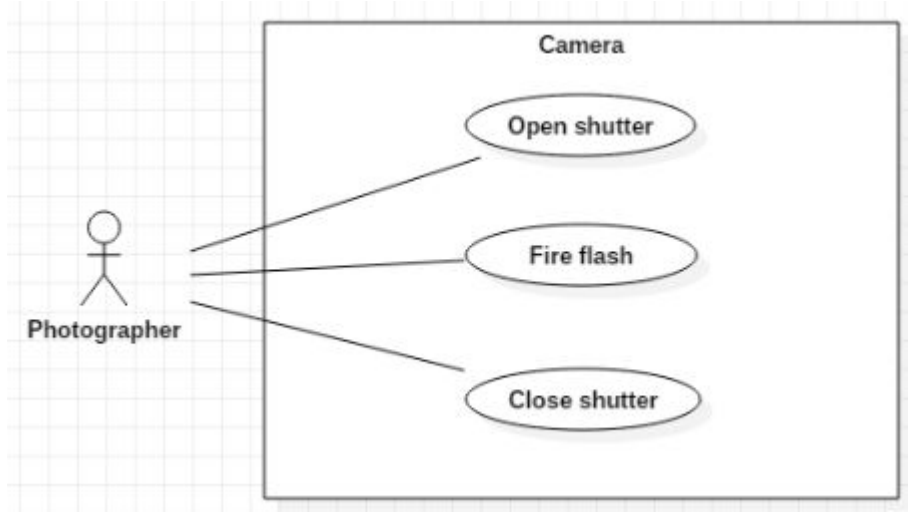
- Start with the list of actors and consider how each of them will use the system
- Consider the following list of helpful questions (if the answer to any of them is yes, you probably identified a use case):
  - What functions will the actor want from the system?
  - Does the system store and retrieve information? If so, which actors trigger these behaviors?
  - What happens when the system changes state (e.g. system start and stop)? Are any actors notified?
  - Do any external events affect the system? If so, which actor notifies the system about those events?
  - Does the system interact with any external system?
  - Does the system generate any reports?

Only represent high level use cases. Internal system behavior is not represented here!

Correct



Incorrect



What goes inside a system in the use case diagram? And what does not go in there?

Internal behaviors that are just used by other elements of the system do not belong in the use case diagram.

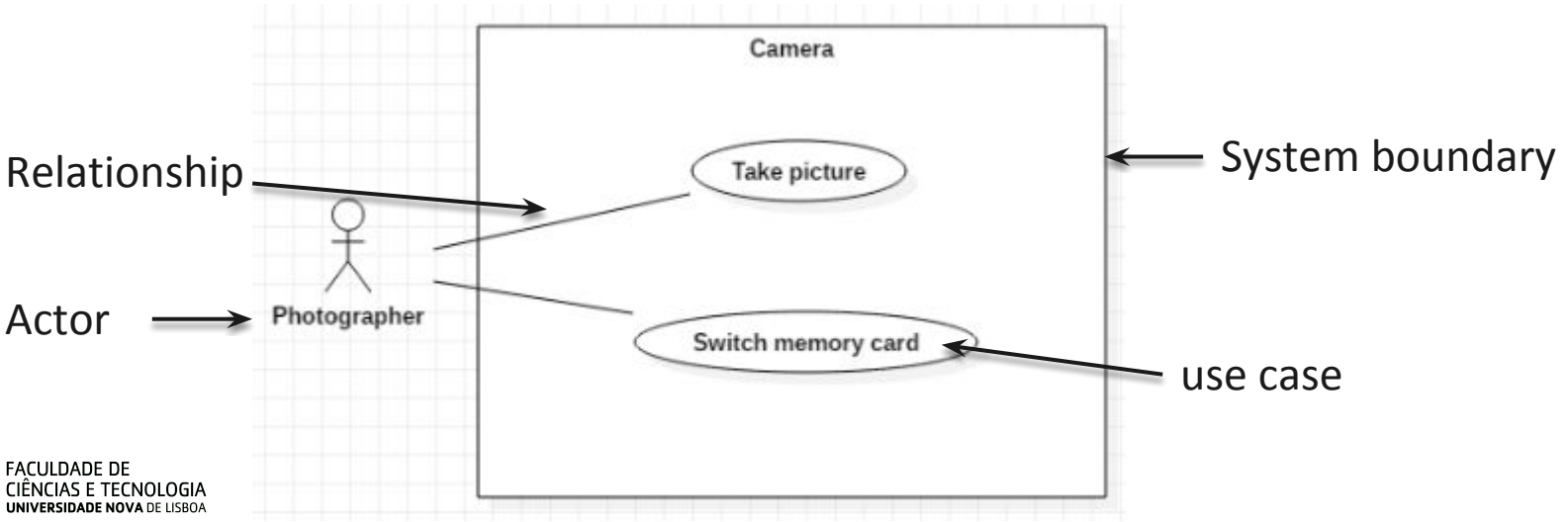
If a use case represents a high level service provided by the system, then it makes sense that an actor may use that service during a “session” using the system, i.e., using that service makes sense as a single action the user has to perform with the system.

Use cases – Summary (so far)

The set of all use cases describes all the possible ways of using the system.

If you start by identifying all actors, this makes it easier to identify the use cases. Each actor must play a role in at least one use case.

A use case is represented visually by an ellipse and must have at least one actor initiating it.



Use case specification

What do use cases represent?

- A use case is composed (i.e. defined) by a set of flows
- Each flow contains a sequence of steps that describe an interaction between the user and the system
- A use case joins a set of flows that, as a whole, satisfy a given user's objective

How should we describe use cases?

- By describing their scenarios:
  - A main scenario (primary flow)
  - Several alternative flows (secondary flows)
- Using semi-structured descriptions: templates
- Textually
- Using formal techniques, such as mathematical expressions, algebraic languages, and so on…
- Using other UML diagrams, namely activity and sequence diagrams

Flows

A flow is a sequence of events showing the typical interactions and the exchanged information exchanged between the user and the system.

- Occurs during a particular execution of the system.
- May be written in natural language
- Is a specific instance of a use case

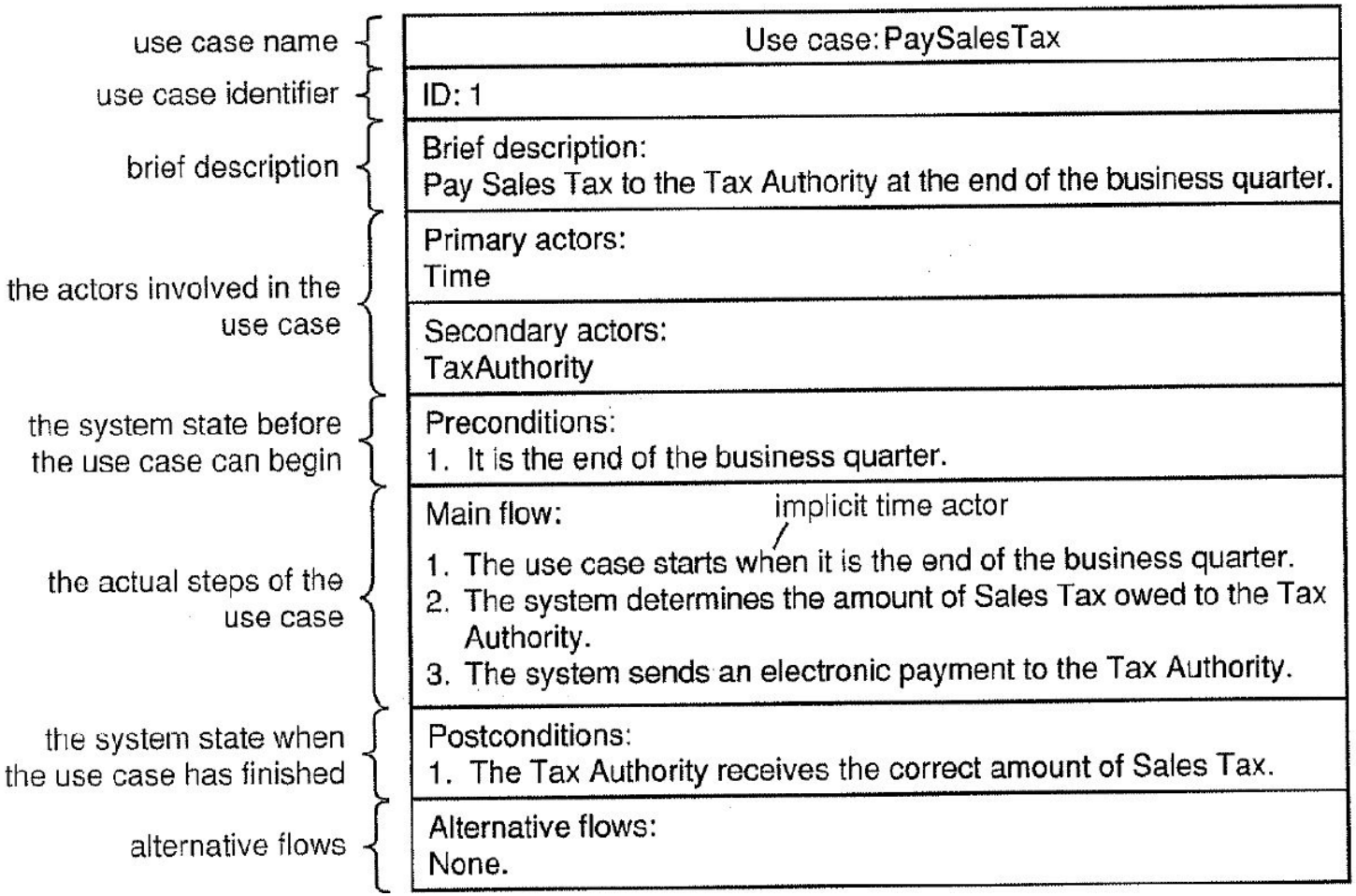
Any use case has always at least two flows

- Main, or primary flow: describes the situation when everything goes well (optimistic – happy day scenario)
- Secondary, or alternative flow: allows for a different sequence of events when compared to the the main scenario, dealing with exceptions and error situations

Use case template

Name: Use case name
ID: use case id
Description: executive summary
Actors: actors participating in the use case
    Primary
    Secondary
Pre-conditions: prerequisites for the successful execution of the use case
Main flow: atomic steps of the use case
Secondary flows: deviations from the main flow
Post-conditions: system status, after a successful completion of the use case.

Use case specification example



Use cases specification in detail

Choosing a use case name

- Pick short but descriptive names.
- The name should be clear for a customer without any specialization in Software Development Methods, so that the customer gets a clear idea of which function, or business process is, just by reading the name of the use case

Specifying use case description

- The use case description should fit in a single paragraph, summarizing the goal of the use case
- The description should capture the essence of the use case, that is, the business benefit for each of the actors involved

Use cases specification

- Actors – use the same names as in the Use Cases diagram
- Primary actor – the actor initiating the use case
  - Any actor initiating a use case is considered a primary actor
- Secondary actor – actor participating in the use case, but not starting it
  - Any actor participating in a use case, but not starting it is considered a secondary actor for that use case
- Each use case is initiated by a single actor
  - Different actors may initiate the same use case

If there are no secondary actors in the use case, it is good style to write “None” in the corresponding Use Case specification section. Let the reader know that you considered that there were no secondary actors to define, rather than forgot about them.

Use cases specification

Pre-conditions – restrict system usage before the use case starts

- The use case cannot start if the pre-conditions are false

Post-conditions – restrict the system status after the use case is executed

- Specify what must be true in the end of the use case

If the use case does not have pre- or post-conditions, write “None” in the corresponding section. Leaving it blank is ambiguous. Make sure whoever reads the use case specification knows that you considered the possibility of specifying them, and decided there were none to specify, rather than forgot that you were supposed to specify them.

Use cases specification – main flow

- Describes the use case, step by step
- Each step should be numbered
- The flow always starts with a main actor initiating the use case

<number> The use case starts when <actor> <action>

- Each step should be numbered and have the format

<number> <actor> does <action>

Good example

1. The use case starts when the client selects the operation “place order”.
2. The client fills in his name and address in the form

In both cases, we have simple, declarative sentences in which it becomes clear who is the actor participating in the use case and what is the action being performed

<number> The use case starts when an **<actor>** **<function>**
<number> The something**<actor>** does **<action>**

Bad example

2. Customer details are filled in.

This is bad in so many ways. We do not know:

- **Who** filled in the data?
- **What** data was filled in?
- **Where** were the data stored?

We must be able to answer clearly and with specific details, to questions such as:

- **Who**?
- **What**?
- **Where**?
- **When**?

Branching within a flow

- There is no standard way of doing it in UML
- We can use an idiom – the keywords if and else – to show branching in a simple way, without writing a separate alternative flow
- Branching within a flow is possible. It reduces the number of use cases, leading to a more compact representation of requirements
- Don't get carried away. Use this sparingly

Branching example

Use case: ManageBasket
ID: 2
Brief description: The Customer changes the quantity of an item in the basket.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The shopping basket contents are visible.
Main flow: 1. The use case starts when the Customer selects an item in the basket. 2. If the Customer selects "delete item" 2.1 The system removes the item from the basket. 3. If the Customer types in a new quantity 3.1 The system updates the quantity of the item in the basket.
Postconditions: None.
Alternative flows: None.

Repetition within a flow - for

- Sometimes (not very often) you need an actor to repeat an action several times
- The idioms for and while are not standard, but they can be useful in these situations

n. For (iteration expression)
  n.1. Do something
  n.2. Do some other thing
  n.3. ...
n+1.

Repetition example - for

Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: 1. The use case starts when the Customer selects "find product". 2. The system asks the Customer for search criteria. 3. The Customer enters the requested criteria. 4. The system searches for products that match the Customer's criteria. 5. If the system finds some matching products then 5.1 For each product found 5.1.1 The system displays a thumbnail sketch of the product. 5.1.2 The system displays a summary of the product details. 5.1.3 The system displays the product price. 6. Else 6.1 The system tells the Customer that no matching products could be found.
Postconditions: None.
Alternative flows: None.

Repetition example – while

- Use the while idiom to model a sequence of actions that will be repeated as long as the while condition remains true

n. While (Boolean condition)
    n.1 Do something
    n.2. Do something else
    n.3. …
n+1.

Repetition example – while

Use case: ShowCompanyDetails
ID: 4
Brief description: The system displays the company details to the Customer.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: 1. The use case starts when the Customer selects "show company details". 2. The system displays a web page showing the company details. 3. While the Customer is browsing the company details 3.1 The system plays some background music. 3.2 The system displays special offers in a banner ad.
Postconditions: 1. The system has displayed the company details. 2. The system has played background music. 3. The system has displayed special offers.
Alternative flows: None.

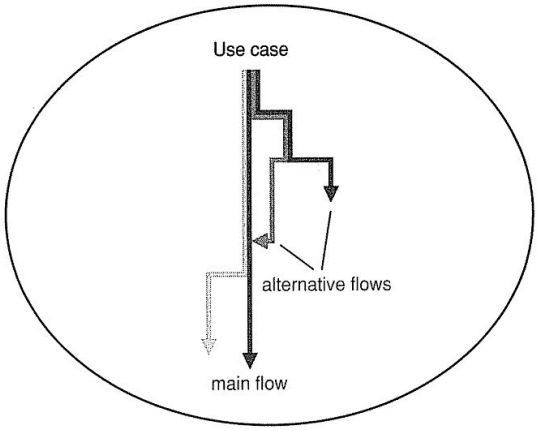
Don’t get carried away!

If, for and while are not frequently used
Detractors of these idioms prefer to use branches of alternative flows

Modelling secondary (aka alternative) flows

- A use case has always one main flow
- It may also have several alternative flows
- Alternative flows handle errors, branches and interruptions to the main flow
- Alternative flows frequently do not return to the main flow
    - Alternative flows frequently have different post-conditions as they handle errors and exceptions

Alternative flows should not have alternative flows among themselves.



Alternative flow example

Use case: CreateNewCustomerAccount
ID: 5
Brief description: The system creates a new account for the Customer.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: 1. The use case begins when the Customer selects "create new customer account". 2. While the Customer details are invalid 2.1 The system asks the Customer to enter his or her details comprising e-mail address, password, and password again for confirmation. 2.2 The system validates the Customer details. 3. The system creates a new account for the Customer.
Postconditions: 1. A new account has been created for the Customer.
Alternative flows: InvalidEmailAddress InvalidPassword Cancel

The alternative flow for invalid email address

Alternative flow: CreateNewCustomerAccount:InvalidEmailAddress
ID: 5.1
Brief description: The system informs the Customer that he or she has entered an invalid e-mail address.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The Customer has entered an invalid e-mail address.
Alternative flow: 1. The alternative flow begins after step 2.2 of the main flow. 2. The system informs the Customer that he or she entered an invalid e-mail address.
Postconditions: None.

Notes on alternative flows

- They have a separate list of actors used by the alternative
- Pre- and post-conditions do not necessarily match those of the main flow
  - If the alternative flow returns to the main flow, then its post-conditions are added to those of the main flow
- The steps within the alternative flow must also be specified
- Alternative flows should **not** have alternative flows of their own, as the model would quickly become too complex

How can one trigger an alternative flow?

- Alternative flow triggered instead of the main flow
- Alternative flow triggered after a particular step in the main flow
- Alternative flow triggered at any time during the main flow.

Alternative flow triggered instead of the main flow

- Alternative flow executes instead of the main use case
- Alternative flow triggered by the primary actor
- Use case replaced entirely

Alternative flow triggered after a particular step in the main flow

- This is a major deviation from the main flow and might not return to it
- When triggered after a particular step in the main flow, begin the alternative flow as follows:

1. The alternative flow begins after step X of the main flow.

Alternative flow triggered at any time during the main flow

- Use this sort of alternative flow to model something that could happen at any point in the main flow before the final step
- When an alternative flow can be triggered at any time during the main flow, you should begin it as follows

> 1. The alternative flow begins at any time

- If you wish to return to the main flow, you can express this as follows:

> N. The alternative flow returns to step M of the main flow

Alternative flow triggered at any time

Alternative flow: CreateNewCustomerAccount:Cancel
ID: 5.2
Brief description: The Customer cancels the account creation process.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Alternative flow: 1. The alternative flow begins at any time. 2. The Customer cancels account creation.
Postconditions: 1. A new account has *not* been created for the Customer.

How do you find alternative flows?

Look for:

- Possible alternatives to the main flow
- Errors that might be raised in the main flow
- Interrupts that might occur at a particular point in the main flow
- Interrupts that might occur at any point in the main flow

Limit the number of alternative flows to a necessary minimum

Two strategies for keeping a relatively low number of alternative flows,

- Select the most important alternative flows and document those
- Where there are groups of alternative flows that are very similar, document one member of the group as an exemplar, and add notes to this to explain how the others differ from it

Keep it simple, on a need to know basis

- Keep the information to the necessary minimum
  - Many alternative flows may never be specified at all
  - A one line description of them added to the use case may be enough detail to allow understanding the functioning of the system
- Your goal is to capture use cases to understand the desired behavior of the system
- Your goal is not to create a complete use case model just for the sake of it

Make sure you can trace your requirements

- Create traceability links between your requirements and the use case model
- Use this to check consistency:
  - If you have a requirement not covered by any use case, you are probably missing a use case
  - If you have a use case which is not traceable to any requirement, your requirements set is probably incomplete.

		Use case			
		UC1	UC2	UC3	UC4
Requirement	R1	X			
	R2		X	X	
	R3			X	
	R4				X
	R5	X			

When to apply use case modelling

- The system is dominated by functional requirements
- The system has many types of users to which it delivers different functionalities (i.e. there are many actors)
- The system has many interfaces (i.e. there are many actors)

Use cases are a poor choice when…

- The system is dominated by non-functional requirements
- The system has few users
- The system has few interfaces
- Examples
  - Embedded systems
  - Algorithmically complex systems with a few interfaces

Model everything
At the right level of abstraction
Using the most appropriate formalism

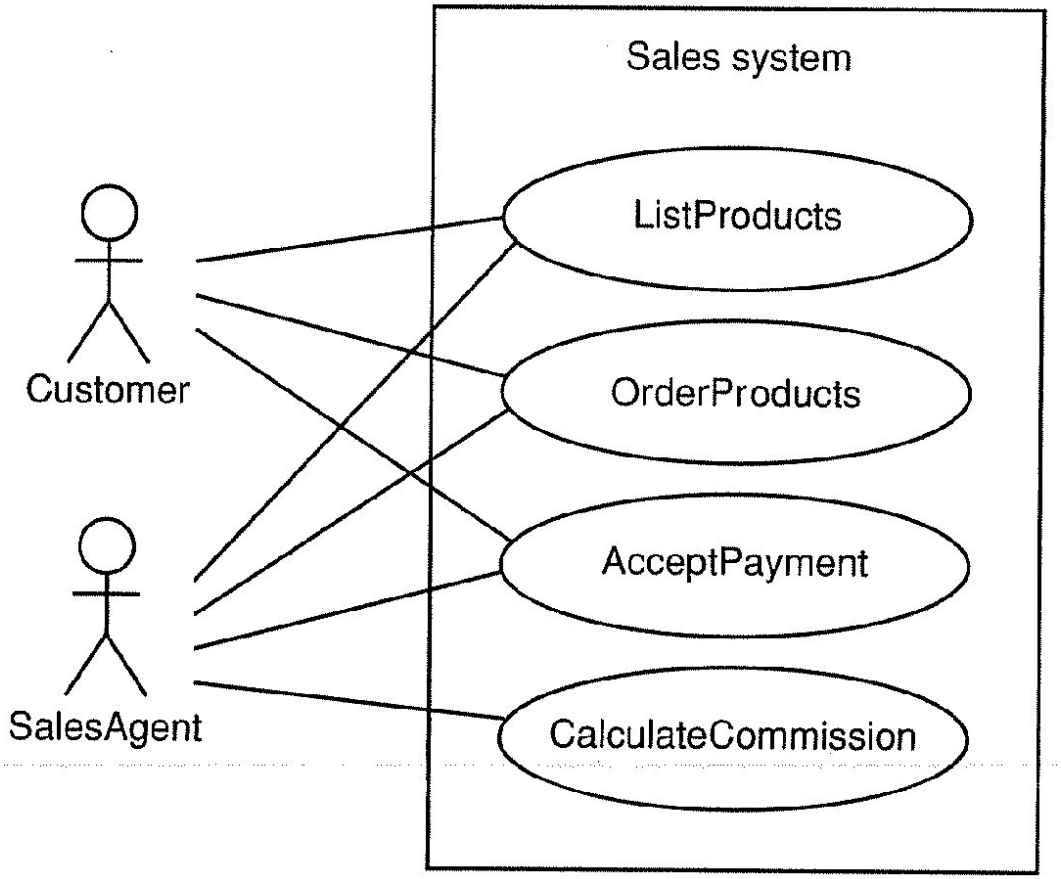
There are many other requirements engineering techniques – learn about them in the MIEI RAS course!

Advanced Use Case Modelling

Generalization among actors

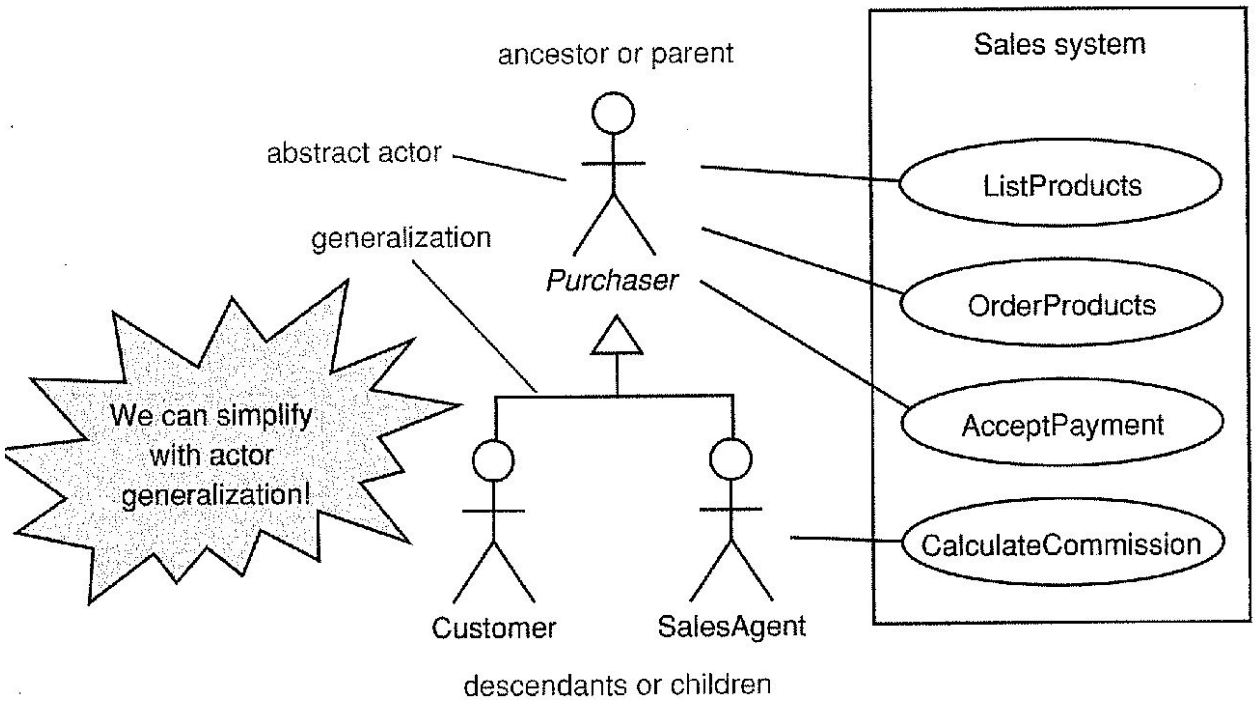
Consider the following sales system

Customer and SalesAgent have several use cases in common. We can use this to simplify the model.



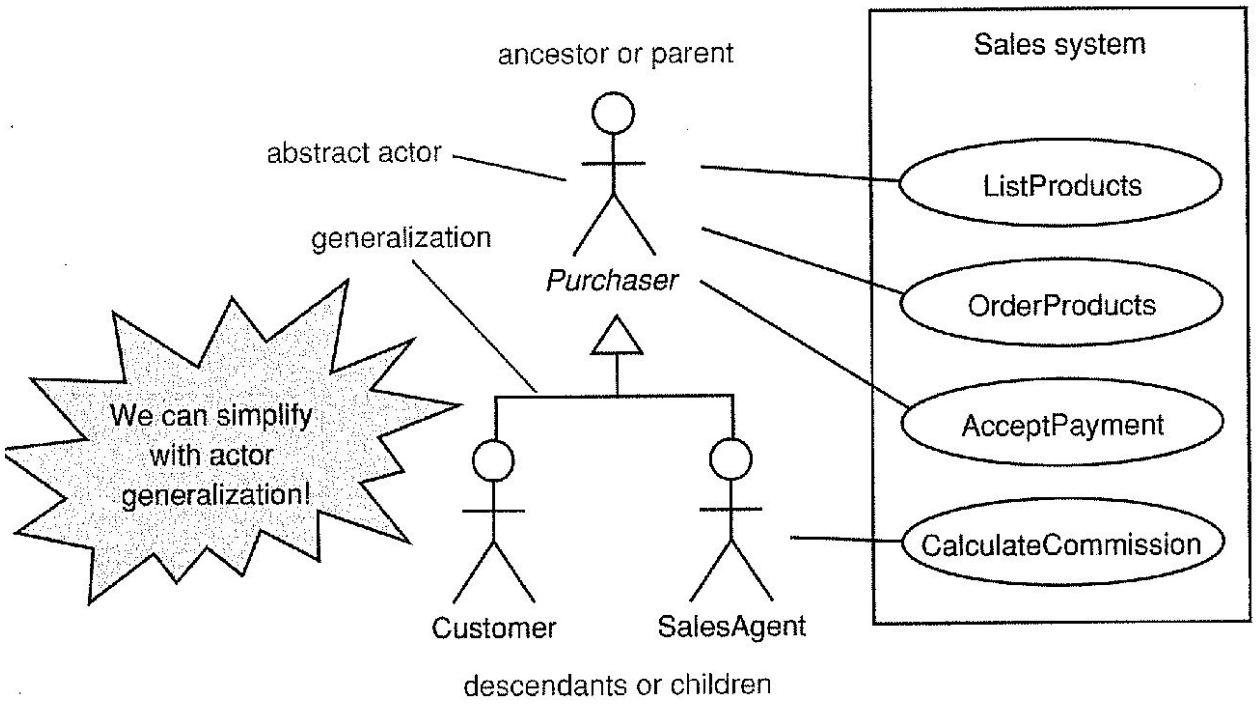
We can factor out the common use cases!

Actor generalization allows you to create an abstract actor *Purchaser* that interacts with the use cases ListProducts, OrderProducts and AcceptPayment. Customer and SalesAgent are the concrete actors. Real people, or systems, could play that role. In contrast, *Purchaser* cannot be instantiated.



Customer and SalesAgent inherit Purchaser's behavior

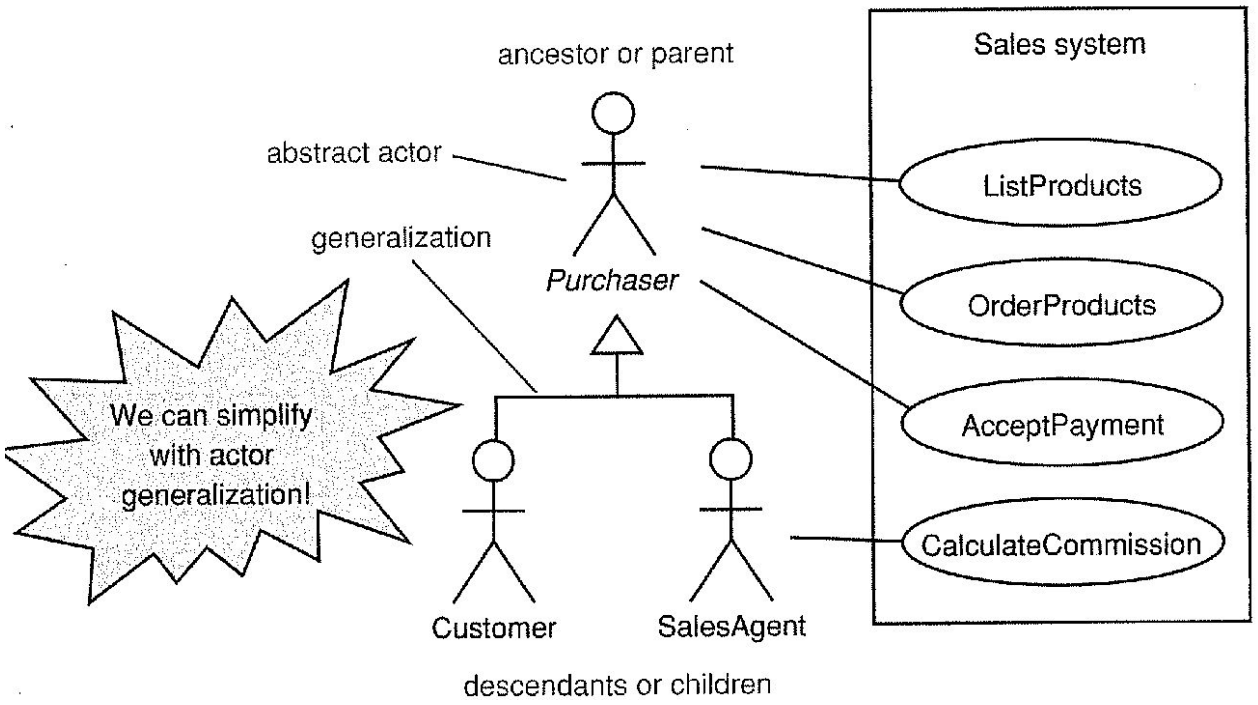
Both Customer and SalesAgent have interactions with ListProducts, OrderProducts and AcceptPayment.
Only the SalesAgent actor interacts with the CalculateCommission use case.
We can use a descendent anywhere the parent is expected!



Make sure you respect the substitution principle!

Could we make the SalesAgent a specialization of Customer, so that we would define the first three use cases for Customer and the forth for SalesAgent only?

That would be conceptually wrong. A SalesAgent is not a Customer. So, no. It would be a bad modelling practice to make that generalization.



Generalization: when to use?

- Having several use cases shared by different actors that are really playing a common role suggests na opportunity to generalize
- This only makes sense if all actors communicate with the use case in the same way
- It is usually a good idea to keep the generalized actor as abstract – we are generalizing a common role. Having an abstract actor simplifies the generalization semantics
- Descendant actors inherit the roles and relationships with use cases from their Ancestor actors
- Wherever an ascendant is expected, we may replace it by the descendant (substitution principle)

If the substitution principle fails, you are most likely generalizing something you really should not generalize.

When not to use generalization among actors

- The goal of using generalization is to simplify the model. If you use generalization and the model becomes harder to understand, then do not use it.

Key point: models are used, among other things, to **communicate effectively** with other stakeholders. If using generalization among actors makes communication more effective by making the model simpler to understand, use it. Otherwise, do not use it.

Relationships among use cases

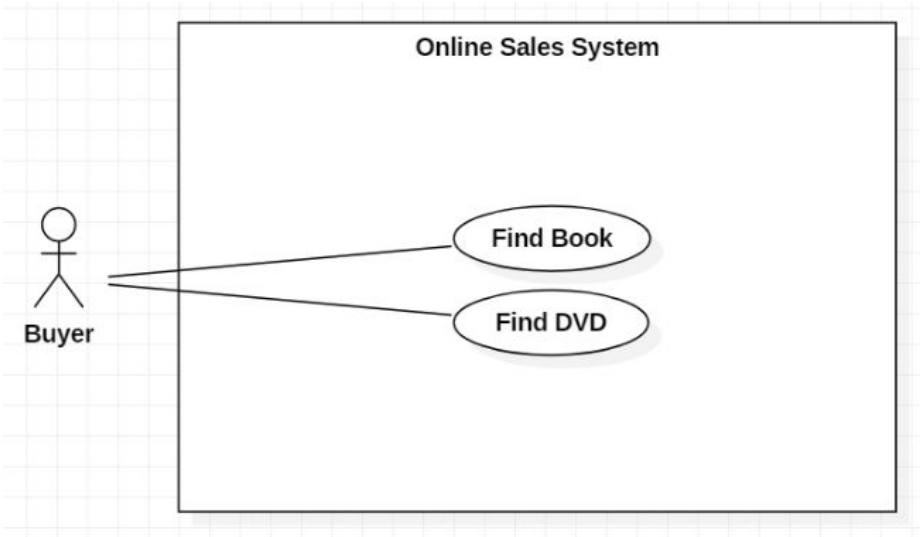
Generalization

Inclusion

Extension

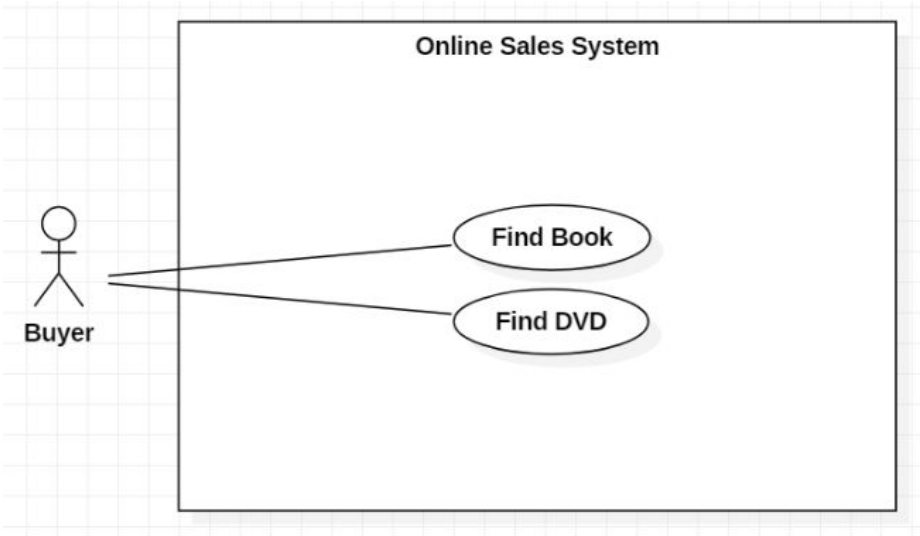
Use case generalization

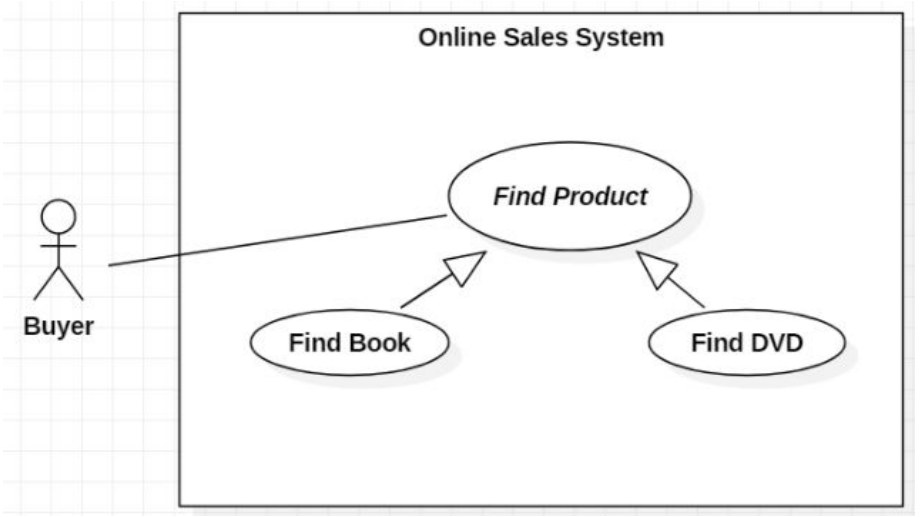
Find Book and Find DVD are pretty similar use cases



If you write down the use case specification for each of these use cases, you are likely to end up with two very similar use cases specification, the difference being on what the buyer is looking for (a Book, *vs.* a DVD).

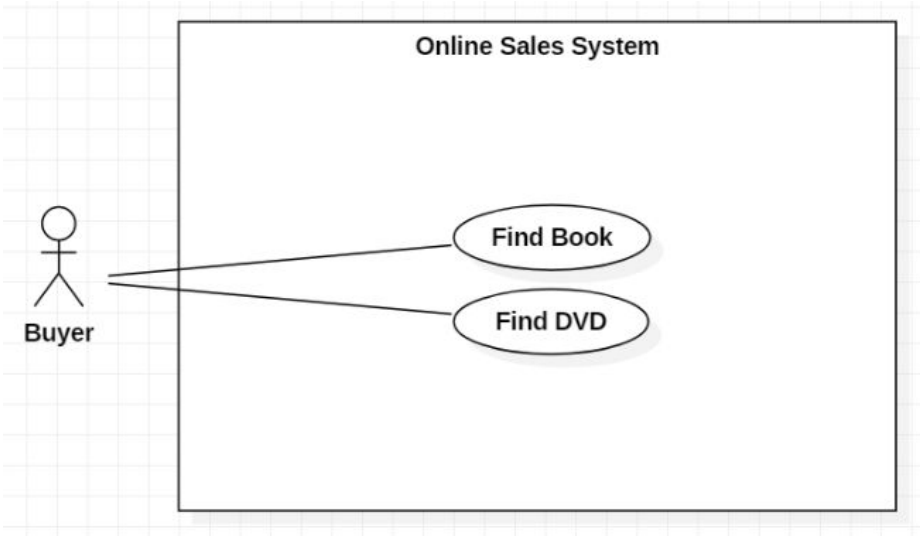
You can generalize them into the *Find Product* use case

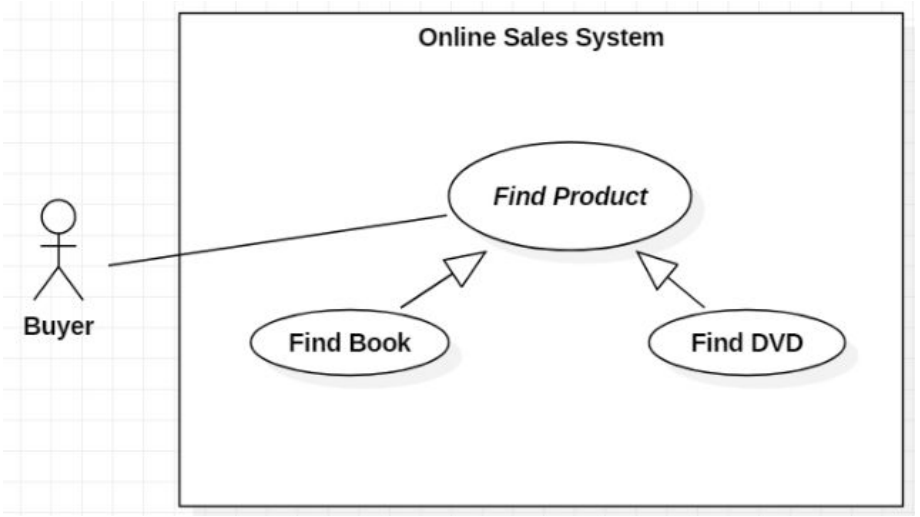




The italic font used in *Find Product* denotes this generalized Use Case is abstract. You cannot instantiate it.

Use cases generalization





Use cases generalization

- When a use case is a specialized version of another, more generic, use case, we may use **generalization**
- The goal must always be **keep the model as simple as possible**, but no more than that
- In Use Cases generalization the descendant represents a more specific version of its ancestor
- The descendant may:
  - Inherit characteristics from the ascendent
  - Add new characteristics
  - Change inherited characteristics

Specifying the *abstract* use case *Find Product*

Use case: *Find product*

ID: 1

Description: The buyer tries to find a product in the system.

Main actor: Buyer

Secondary actors: None

Pre-conditions: None

Main flow:

1. The case study starts when the buyer selects the option to find a product.
2. The system asks the buyer to define the search criteria to use.
3. The buyer introduces the search criteria.
4. The system seaches products satisfying the buyer's search criteria.
5. If the system finds products matching criteria for products
   5.1. The system shows the user a list of products matching the search criteria.
6. Else
   6.1. The system shows the buyer a message indicating no product was found.

Post-conditions: None

Alternative flows: None

Specifying the Find Book specialization

Use case: Find Book

ID: 2

Specializes: *Find Product*

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message indicating no product was found.**

Post-conditions: None

Alternative flows: None

Create a new unique ID

Use case: Find Book

ID: 2

Specializes: ***Find Product***

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message indicating no product was found.**

Post-conditions: None

Alternative flows: None

You can reuse some actions without changing them

Use case: Find Book

ID: 2

Specializes: *Find Product*

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message indicating no product was found.**

Post-conditions: None

Alternative flows: None

Override actions to make them specific to books. Annotate what is being overridden (e.g. (o2.) means action 2 is overriden)

Use case: Find Book

ID: 2

Specializes: *Find Product*

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message indicating no product was found.**

Post-conditions: None

Alternative flows: None

Add new actions as necessary

Use case: Find Book

ID: 2

Specializes: *Find Product*

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message explaining that no product was found.**

Post-conditions: None

Alternative flows: None

Re-label other actions, keeping the previous label available for easier tracing

Use case: Find Book

ID: 2

Specializes: *Find Product*

Description: The buyer searches a book in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a book.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for books satisfying the buyers search criteria.**
5. **(o5.) If the system finds books matching the search criteria for books**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each book, the system presents the author, title, price and ISBN
   - 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books.
6. (6.) Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message explaining that no product was found.**

Post-conditions: None

Alternative flows: None

Specifying the Find DVD specialization is similar, with the DVD-specific adjustments

Use case: Find DVD

ID: 3

Specializes: *Find Product*

Description: The buyer searches a DVD in the system.

Main Actor: Buyer

Secondary Actors: None

Pre-conditions: None

Main flow:

1. **(o1.) The use case starts when the buyer selects the option to find a DVD.**
2. **(o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, and genre.**
3. The buyer introduces the search criteria.
4. **(o4.) The system searches for DVDs satisfying the buyers search criteria.**
5. **(o5.) If the system finds DVDs matching the search criteria for DVDs**
   - 5.1. The system shows its best seller.
   - **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.**
   - 5.3. For each DVD, the system presents the **author, title, and genre.**
   - 5.4. While there are more DVDs to show, the system offers the buyer the possibility of requesting the next page with more DVDs.
6. Else
   - 6.1. The system shows its best seller.
   - **6.2. (6.1.) The system shows the buyer a message explaining that no product was found.**

Post-conditions: None

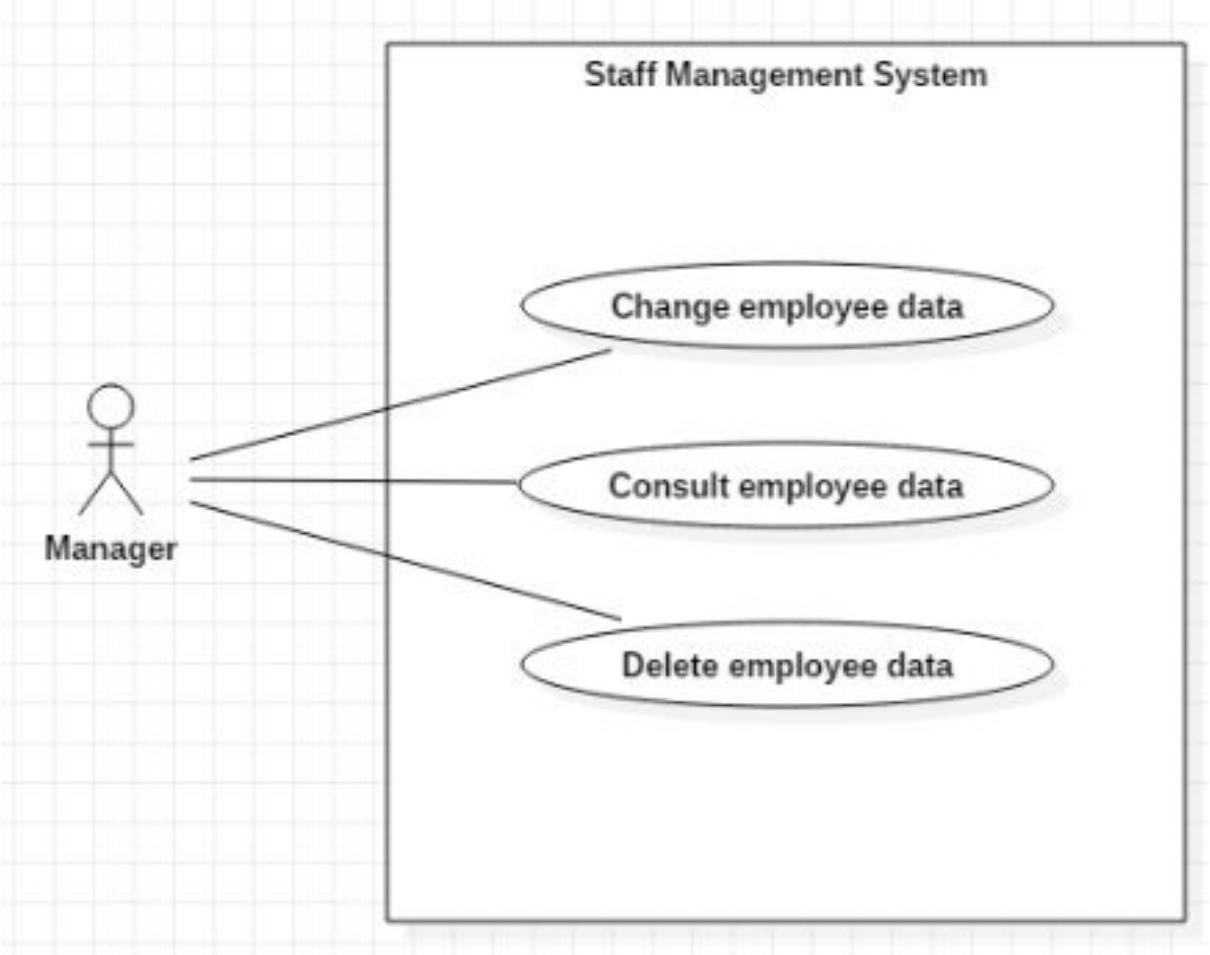
Alternative flows: None

Generalize Use Cases to factor common behavior among sibling use cases in a generalized use case

Characteristic	Inherit	Add	Redefine
Relationship	Yes	Yes	No
Extension points	Yes	Yes	No
Pre-condition	Yes	Yes	Yes
Post-condition	Yes	Yes	Yes
Action in the main flow	Yes	Yes	Yes
Alternative flows	Yes	Yes	Yes

Reusing Use cases

Consider the following staff management system



Change employee data

Use Case: Change employee data

ID: 4

Description: The manager changes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to change the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.
5. The manager changes the data of the employee.

...

Post conditions: The employee's data were changed

Alternative flows: None

View employee data

Use Case: View employee data

ID: 5

Description: The manager views the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to view the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.

...

Post conditions: None

Alternative flows: None

Delete employee data

Use Case: Delete employee data

ID: 6

Description: The manager deletes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to delete the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.
5. The manager deletes the data of the employee.

...

Post conditions: The employee's data were deleted.

Alternative flows: None

Factor the common subsequence

Use Case: Change employee data

ID: 4

Description: The manager changes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to change the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.
5. The manager changes the data of the employee.
6. ...

Post conditions: The employee's data were changed

Alternative flows: None

In all cases there is a common subsequence (which is actually longer, but in this case we only want to reuse just the search part of it). We can factor our model, to avoid unnecessary repetitions.

Use Case: View employee data

ID: 5

Description: The manager views the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to view the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.

...

Post conditions: None

Use Case: Delete employee data

Alternative flows: None

ID: 6

Description: The manager deletes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

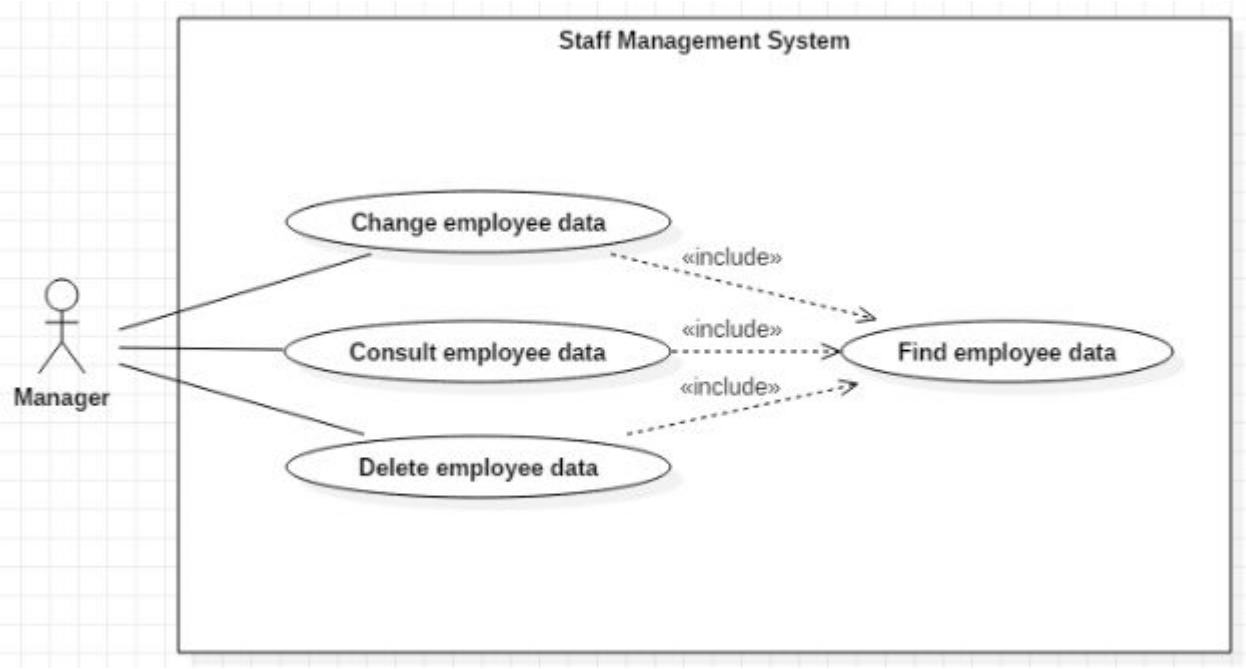
1. The use case starts when the manager selects the option to delete the employee's data.
2. The manager introduces the identification of the employee in the system.
3. The system finds the data of the employee.
4. The system shows the data of the employee.
5. The manager deletes the data of the employee.

...

Post conditions: The employee's data were deleted

Use cases reuse

The <<include>> relationship supports factoring the use case to find employee data. This use case is then reused by other use cases which need to find employee data.



Change employee data – new version

Use Case: Change employee data

ID: 4

Description: The manager changes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to change the employee's data.
2. The manager introduces the identification of the employee in the system.
3. Include (Find employee data)
4. The manager changes the data of the employee.

...

Post conditions: The employee's data were changed

Alternative flows: None

View employee data – new version

Use Case: View employee data

ID: 5

Description: The manager views the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to view the employee's data.
2. The manager introduces the identification of the employee in the system.
3. Include (Find employee data)

...

Post conditions: None

Alternative flows: None

Delete employee data – new version

Use Case: Delete employee data

ID: 6

Description: The manager deletes the employee's data

Main actor: Manager

Secondary Actor: None

Pre-conditions: The manager is authenticated in the system

Main flow:

1. The use case starts when the manager selects the option to delete the employee's data.
2. The manager introduces the identification of the employee in the system.
3. Include (Find employee data)
4. The manager deletes the data of the employee.

...

Post conditions: The employee's data were deleted.

Alternative flows: None

The <<includes>> association

- Promotes reuse: identical parts of a base use case can be extracted to create a new use case
- The base use cases can then be associated with the new use case through the <<includes>> association
- The new use case does not have a meaning on its own

Advantages:

- Fast to specify
- Eliminates “scattered” behavior
- More modularity
- More locality

The <<includes>> association

- The included use case provides reusable behavior to the base case
- Execution model:
  - The base use case executes until the inclusion point
  - The included use is executed
  - The execution goes back to the base use case, to continue its execution on the next step.
  - The base use case is only complete with the included use case
  - When the included use case is incomplete, this is known as a behavior fragment
    - The included use case is not directly initiated by actors
  - However, it is also possible to include complete use cases
    - Those included complete use cases

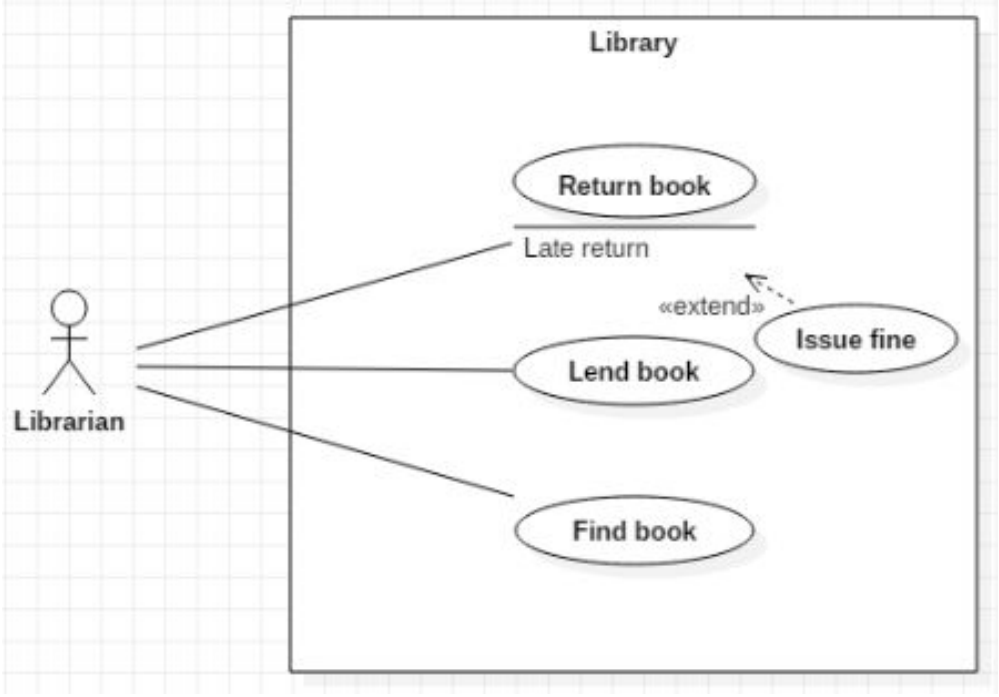
Use cases extension

Use cases extension

- Sometimes you may want to add a new behavior to a use case
- You can do so by creating an extension point
  - This implies also defining an extension use case specifying the behavior fragment to insert in the extension points
  - The base case is oblivious to its extending use cases
    - **This is fundamentally different from an include, where the base use case is incomplete without the included use case. Here, the base case is complete even in the absence of an extension.**

Consider a library management system

The Return book use case can be performed completely on its own. Issuing a fine as a consequence of a late return of a book to the library adds additional behavior, but the book could always be returned without issuing a fine, so return book is, in that sense, a complete use case.



Return book – base use case specification

Use case: return book

ID: 7

Description: The librarian returns a borrowed book

Main actor: Librarian

Secondary actors: None

Pre-conditions: The librarian is logged into the system

Main flow:

1. The use case starts when the librarian selects the option to return a book
2. The librarian introduces the identifier of the patron who has borrowed the book.
3. The system displays data on the patron who had borrowed the book, including information on the list of books borrowed by that patron.
4. The librarian finds the book to return in the borrowed books list

Extension point: late return

5. The librarian marks the book as “returned”

...

Post conditions: The book was returned.

Alternative flows: None

Extension use case specification

Use case: Issue fine

ID: 8

Description: The librarian registers and issues a fine

Main actor: Librarian

Secondary actors: None

Pre-conditions for segment 1: the book return is late

Segment 1 flow:

1. The librarian fills in the system the details of the fine (patron identifier, and amount to pay)
2. The system prints a fine.

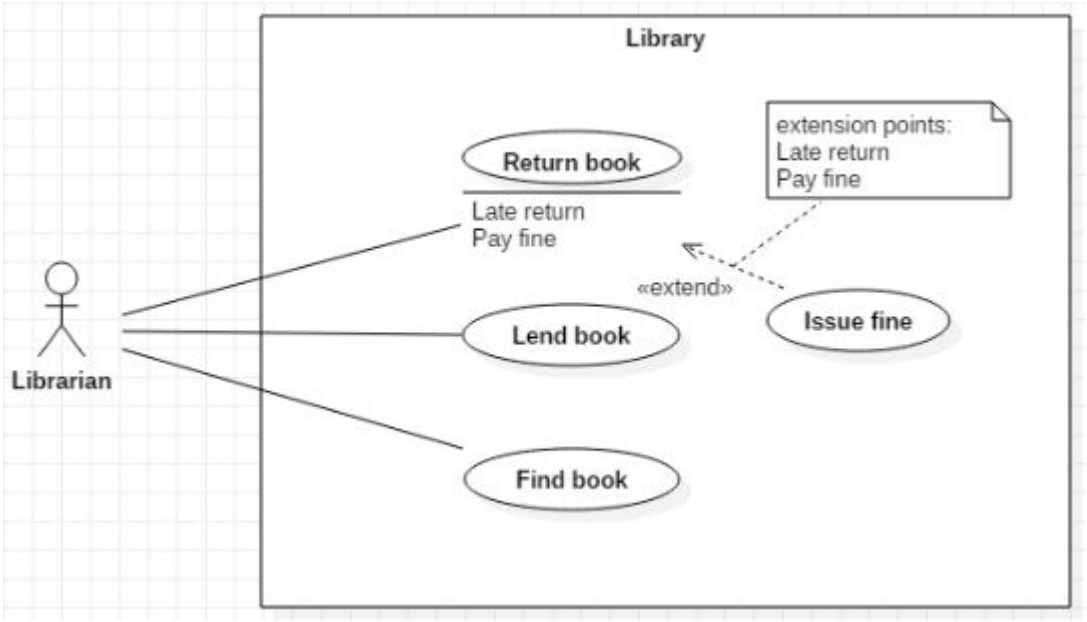
Post-conditions in segment 1:
The fine is registered in the system.
The system printed a fine.

The <<extend>> association

- The <<extend>> association has to specify one or more extension points in the base use case
- Otherwise, one may assume that the extension applies to all the extension points of that use case
- The extension use case has to have as many segments as the number of extension points as those identified in the <<extend>> association.
- There may be several extension use cases for a given same extension point, in a single base use case
- In those cases, the several extension points are executed, by a non-pre-determined order
- Extension use cases may also include pre- and post-conditions
- It is possible to extend extension use cases
- However, this is something normally to avoid, as the model tends to become too complex

Using multiple extensions

Suppose the system should also support the payment of the fine resulting from returning a book too late.



This mechanism is useful when we need to extend the base use case in several distinct points. In this example, when we detect the book return is late, we could model, in the base case, that we would also check whether that patron has additional late returns and list them, so that the patron could pay a single fine, with the sum of the outstanding fines.

Specification of the extension use case

Use case: Issue fine

Description:
Segment 1: The librarian registers and prints a fine
Segment 2: The librarian accepts the payment of the fine

Primary actor: librarian

Secondary actors: none

Pre conditions for segment 1: The book return is late

Segment 1 flow:
1. The librarian inserts in the system the fine details (patron id, and amount to pay)
2. The system prints the fine.

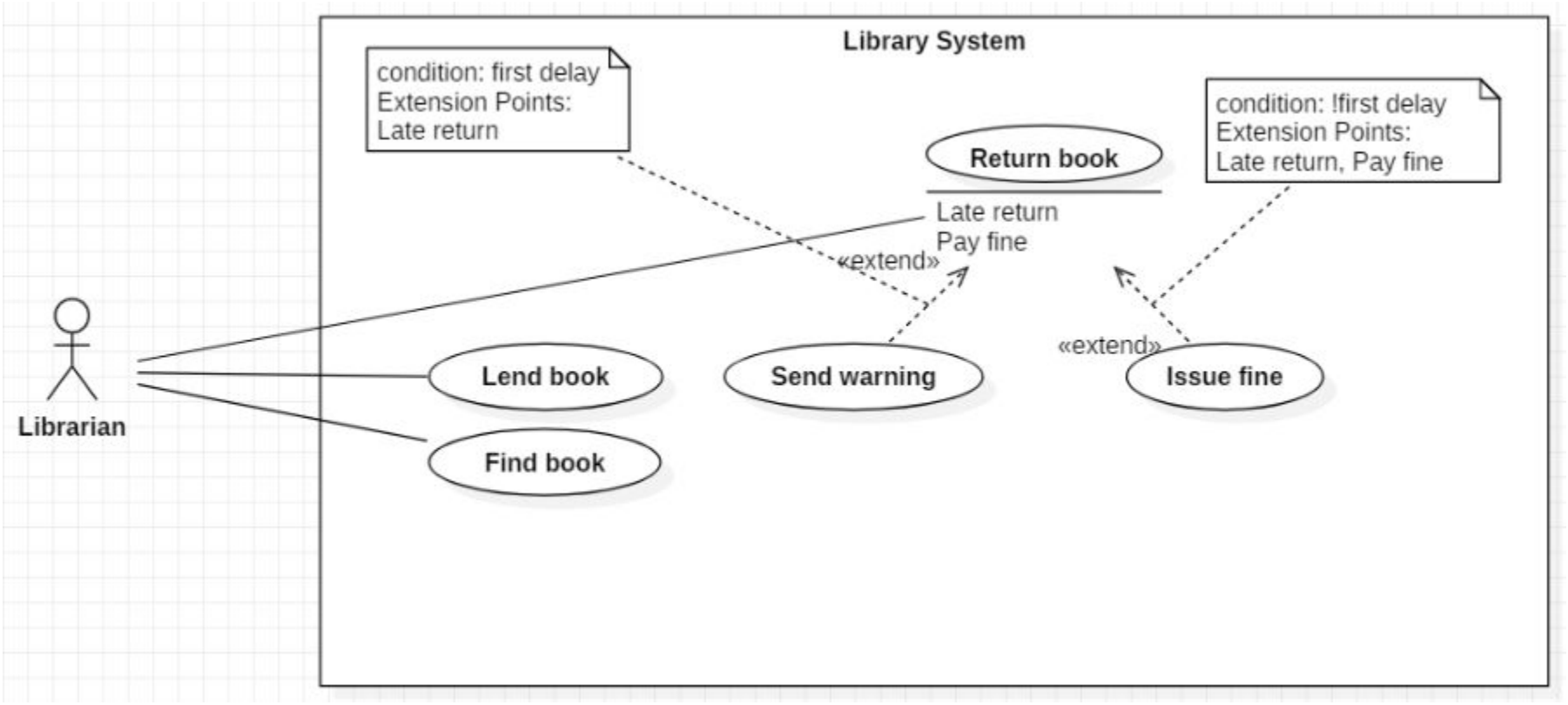
Post conditions of segment 1:
The fine is registered in the system.
The system prints a the fine.

Pre conditions for segment 2: There is a fine to be paid by the patron

Segment 2 flow:
1. The librarian accepts the ticket payment.
2. The librarian introduces into the system the fine data (patron id and amount payed)
3. The system prints the fine receipt.

Post conditions of segment 2:
1. The fine is registered as paid
2. The system printed a receipt for the paid fine.

Conditional extensions



Send warning use case specification

Use case: Send warning

ID: 9

Description:
Segment 1: The librarian sends a late return book warning.

Main actor: Librarian

Actores secondary actors: None

Pre-conditions for segment 1: The book return is late

Flow for segment 1:

1. The librarian inserts in the system the details of the fine (patron id and amount to pay)

Post-conditions for segment 1:
The warning is registered in the system.

Associations among use cases (summary)

<<includes>>

- The behavior of B is included in A
- The included use case B is necessary for ensuring the functionality of use case A
- A knows about B
- A is the base use case; B is the included use case

<<inherits>>

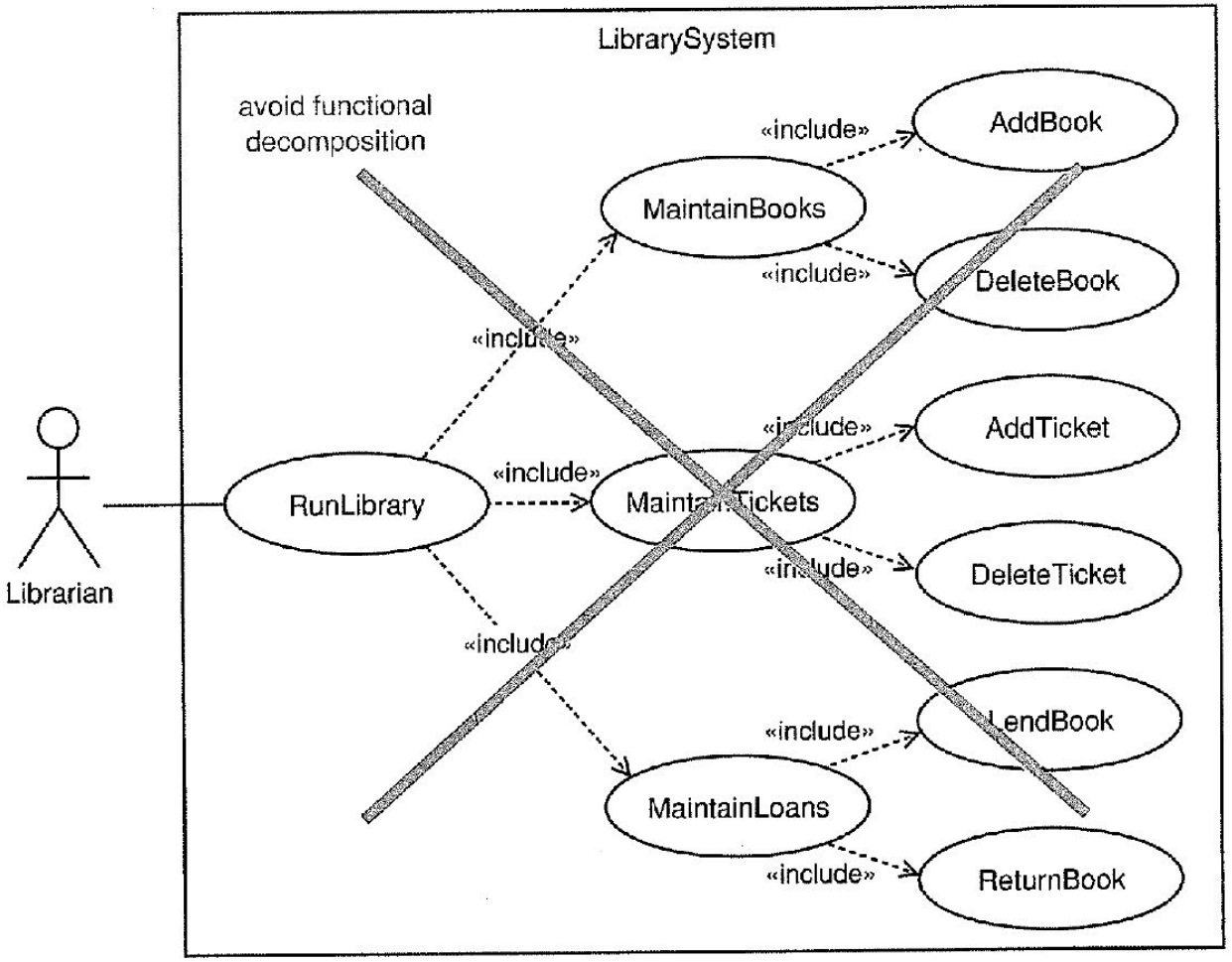
- Similar to class inheritance
- A inherits the behavior of B and may redefine it or extend it
- It is possible (and desirable) to define *abstract* use cases
- B is specialized by A

<<extends>>

- The behavior of B may extend the behavior of A
- A does not know B
- A is the base use case

Usage style

Avoid functional decomposition



About the usage of advanced mechanisms in use case modelling

- Often our stakeholders are able to understand the notion of actor and use case with relatively little training
- They struggle more to understand the notion of generalization among actors – make it easiear using generalized abstract actors
- The excessive use of <<include>> relationships makes models harder to understand, as you need to look into several use cases
- The extension mechanism is hard to understand for untrained modellers (and even for many trained software developers ☹) – use it with caution and parsimony
- When generalizing use cases, make the generalized use cases abstract; failing to to do so makes the specialized use cases much harder to understand

About the textual description style

Keep the use case specification compact

- Hint: make it fit in less than one page

Put your emphasis on **what** rather than **how**

- Leave design details… to the design

Avoid the functional decomposition of use cases

- A common mistake is to start from a too generic use case and then decomposing it until one reaches more “primitive” use cases which are, finally, of an adequate granularity so that one can detail them

Bibliography

Jim Arlow and Ila Neustadt, “UML 2 and the Unified Process”, Second Edition, Addison-Wesley 2006

- Chapters 1 to 5