CITY UNIVERSITY LONDON

IN2013 Lecture 2

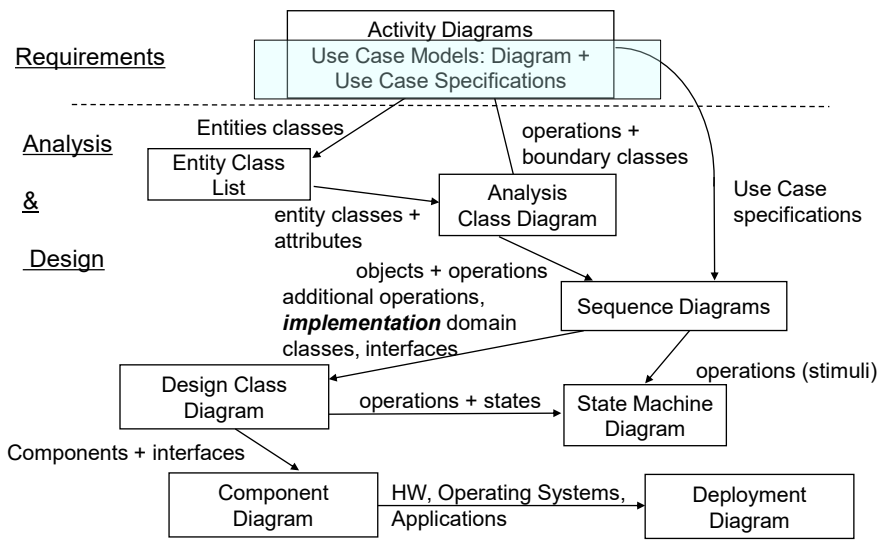
Object-oriented Analysis and Design: Use Case Models

Dr Peter Popov

7th of October 2025

1

From Requirements to Analysis and Design



2

Map

- Part 1: Basics of use case modelling
- Part 2: Advanced use case modelling
- Part 3: Examples and typical mistakes with use case models

3

Part 1: UML Use Case modelling – Basic Concepts

4

Objectives

Introduce use case modelling and the UML Use case diagrams

- Use cases (diagrams and specifications)
  - System boundaries
  - Actors
  - Use cases
    - Main and Alternative flows
    - Controlling the flow
      - Branching with "if"
      - Repetition with for and while
- Relationships between use cases and actors
- Requirements tracing

Use case modelling

Use case modelling is a form of ***requirements engineering***

Use case modelling proceeds as follows:

- Find the ***system boundary***
- Find actors
- Find use cases
  - Use case specification
  - Scenarios

It lets us identify the system boundary, ***who*** or ***what*** uses the system, and ***what*** functions the system should offer

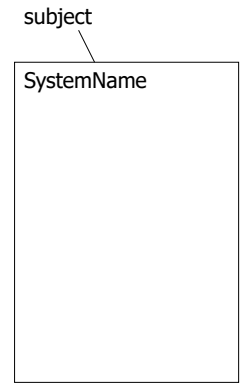
The subject

Before we can build a system, we need to know:

- Where the boundary of the system lies
- Who or what uses the system
- What functions the system should offer to its users

We create a Use Case model containing:

- Subject – the edge of the system
  - also known as the system boundary
- Actors – who or what uses the system
- Use Cases – things actors do with the system
- Relationships - between actors and use cases



What are actors?

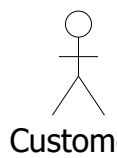
An actor is anything that interacts *directly* with the system

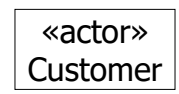
- Actors identify ***who*** or ***what*** uses the system and so indicate where the system boundary lies:
  - People, or
  - External devices (systems)

Actors are ***external*** to the system

- (i.e. outside system boundaries)

An Actor specifies a *role* that some external entity adopts when interacting with the system.

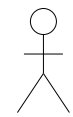




Identifying Actors

When identifying actors asking the following questions may help:

- Who or what uses the system?
- What roles do they play in the interaction?
- Who starts and shuts down the system?
- Who maintains the system?
- What other systems use this system?
- Who gets and provides information to the system?
- Does anything happen at a fixed time?
  - Automatic functions in the system are modelled as triggered by an actor called ***Time***.



9

What are use cases?

A use case is something an actor needs the system to do. It is a "case of use" of the system by a specific actor

Use cases are *always* started by an actor

- The ***primary*** *actor* triggers the use case.
  - Every use case MUST have a primary actor (one or more)
- Zero or more ***secondary*** *actors* interact with the use case in some way

Use cases are *always* written from the point of view of the actors

- A ***use case specification*** is provided for each use case to describe the interaction of actors and the system.

10

Naming use cases

Use cases describe something that happens
They are named using ***verbs*** or ***verb phrases***
Naming standard[1]: use cases are named using

UpperCamelCase e.g. **PaySalesTax**

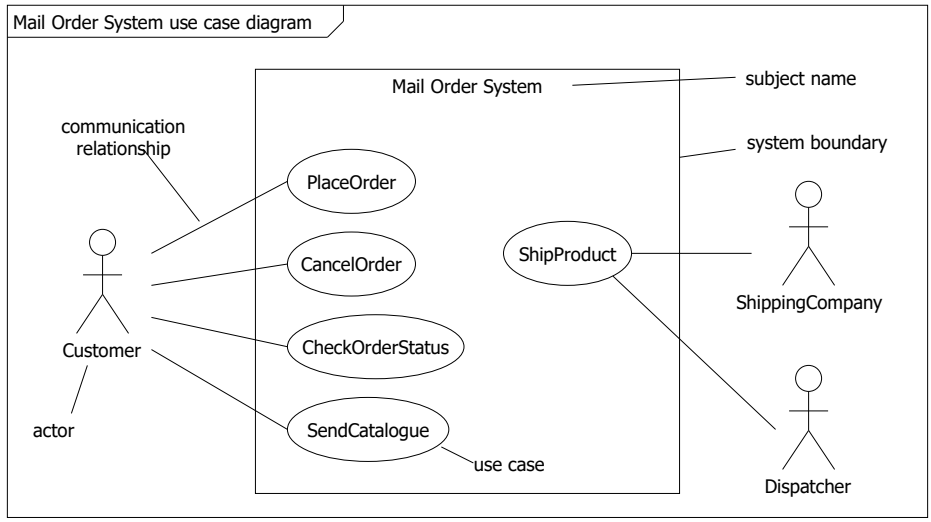
[1] UML 2 does not specify *any* naming standards.
All naming standards used in the module are suggested by Jim Arlow, based on "industry best practice".

Identifying use cases

Start with the list of actors that interact with the system
When identifying use cases ask:

- What functions will a specific actor want from the system?
- Does the system store and retrieve information? If so, which actors trigger this behaviour?
- What happens when the system changes state (e.g. system start and stop)? Are any actors notified?
- Are there any external events that affect the system? What notifies the system about those events?
- Does the system interact with any external system?
- Does the system generate any reports? Who triggers report generation?

The use case diagram



13

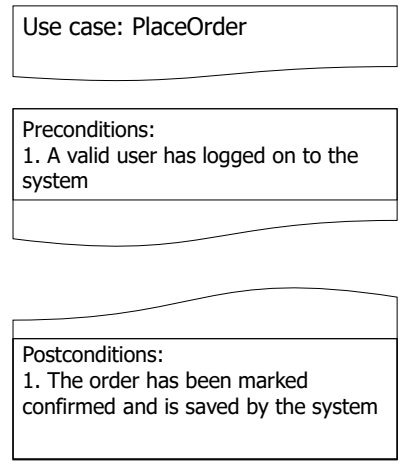
Use case specification

use case name	Use case: PaySalesTax
use case identifier	ID: 1
brief description	Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
the actors involved in the use case	Primary actors: Time
	Secondary actors: TaxAuthority
the system *state before* the use case can begin	Preconditions: 1. It is the end of the business quarter.
the actual steps of the use case	Main flow: (implicit time actor) 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
the system state when the use case has finished	Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
alternative flows	Alternative flows: None.

14

Pre- and postconditions

- Preconditions and postconditions are ***constraints which must be satisfied*** (i.e. these are ***facts***)
  - Preconditions constrain the state of the system ***before*** the use case can start. If the preconditions are NOT met, then the use case cannot start.
  - Postconditions constrain the state of the system ***after*** the use case has executed. Typically, we specify the ***changes of the state*** which occurred as a result of the use case completion.
- If there are no preconditions or postconditions write "None" under the respective heading



Examples of pre-/post-conditions

Examples of preconditions

- The user has a valid user account
  - Typically kept in a database
- The user has successfully logged in
- A connection to the system has been established (actor is an external computer system)
  - e.g. Connection descriptor is kept in the RAM

Examples of postconditions

- A new record (with the details of the booking made) is stored in the system database
- Connection is closed

Main flow

<number> The <something> <some action>

The flow of events lists the steps in a use case

It *always* begins by an actor doing something

- A good way to start a flow of events is:
  1) The use case starts when an <actor> <function>

The flow of events should be a sequence of short steps that are:

- Declarative
- Numbered,
- Time ordered

The main flow is always the *happy day* or *perfect world* scenario

- Everything goes as expected and desired, and there are no errors, deviations, interrupts, or anomalies
- Alternatives in the flow can be shown by branching or by listing under Alternative flows (see later)

A typical mistake in use case specifications

The actor does something that has *no consequences* for the system state

Consider the following step:

- X: Receptionist makes a phone call
  - This is not a valid step in a use case specification if the system does not maintain a ***record about the call***

Consider now this fragment:

- X: user makes a phone call
- X+1: The system records the call
  - this is a valid fragment.

Branching within the main flow: If

Use the keyword ***if*** to indicate alternatives within the flow of events
- There must be a Boolean expression immediately after ***if***

Use indentation and numbering to indicate the conditional part of the flow

Use ***else*** to indicate what happens if the condition is false

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.

19

Repetition within a flow: For

We can use the keyword ***For*** to indicate the start of a repetition within the flow of events

The iteration expression immediately after the ***For*** statement indicates the number of repetitions of the indented text beneath the ***For*** statement.

Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Actors: Customer
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. For each product found 5.1. The system displays a thumbnail sketch of the product. 5.2. The system displays a summary of the product details. 5.3. The system displays the product price.
Postconditions: None.
Alternative flows: NoProductsFound

20

Repetition within a flow: While

We can use the keyword ***while*** to indicate that something repeats while some Boolean condition is true

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 4. The system searches for products that match the Customer's criteria. 4.1. The system plays some background music. 4.2. The system displays special offers in a banner ad.
Postconditions: 1. The system has displayed the company details. 2. The system has played some background music. 3. The systems has displayed special offers.
Alternative flows: None.

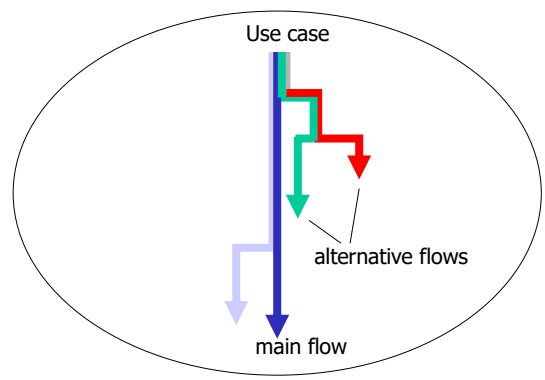
Branching: Alternative flows

We may specify one or more *alternative flows* through the flow of events:

- Alternative flows capture ***errors***, ***anomalies***, and ***interrupts***
- Alternative flows *typically* **do not return** to the main flow (although ***they might***)

Potentially very many alternative flows! You need to manage this:

- Pick the ***most important*** alternative flows and document those.
- If there are groups of similar alternative flows - document one member of the group as an exemplar and (if necessary) add notes to this explaining how the others differ from it.



Only document enough alternative flows to clarify the requirements!

How to find alternative flows

Find alternative flows by examining each step in the main flow and looking for:

- Anomalies
- Exceptions
- Interrupts

23

Look for ...

events – not happening, too frequent, too infrequent, wrong order, ...
actions – insufficient information, not completing, etc.
cognitive exceptions – slips, mistakes, lack of knowledge/ skill, ...
other human exceptions – age, size, gender, disability, etc.
machine exceptions – power failures, breakdowns, blockages, ...
human-machine exceptions – misinterpret interface, ...
machine-machine exceptions – communication failure, scrambled messages, ...

24

Referencing alternative flows in use case specifications

List the names of the alternative flows at the end of the use case

Find alternative flows by examining each step in the main flow and looking for:

- Anomalies
- Exceptions
- Interrupts

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 email 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

alternative flows

An alternative flow specification example

notice how we name and number alternative flows

Alternative flow: CreateNewCustomerAccount:InvalidEmailAddress
ID: 5.1
Brief description: The system informs the Customer that they have entered an invalid email address.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The Customer has entered an invalid email 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 email address.
Postconditions: None.

always indicate how the alternative flow begins.
In this case it starts after step 2.2 in the main flow

The alternative flow may be triggered *instead* of the main flow - started by an actor
The alternative flow may be triggered *after a particular step* in the main flow – after step X
The alternative flow may be triggered *at any time* during the main flow - at any time

Requirements tracing

Given that we can capture functional ***user*** requirements in a requirements model ***and*** in a use case model (a form of ***system*** requirements) we need some way of relating the two

There is a many-to-many relationship between user requirements and use cases:

- One use case covers many individual functional user requirements
- One functional user requirement may be realised by many use cases

Hopefully we have CASE support for requirements tracing:

- With UML tagged values, we can assign numbered requirements to use cases
- We can capture use case names in our Requirements Database

If there is no CASE support, we can create a Requirements Traceability matrix

Alternatively, we can use ***Volere templates*** and each functional user requirement provides a link to the use-case ID it is related to.

		Use cases			
		U1	U2	U3	U4
Requirements	R1	■			
	R2		■	■	
	R3			■	
	R4				■
	R5	■			

Requirements Traceability Matrix

When to use use case models

Use cases describe system behaviour from the point of view of ***one or more actors***. They are the *best* choice when:

- The system is dominated by functional requirements
- The system has ***many types of user*** to which it delivers ***different functionality***
- The system has many interfaces (GUI or with other systems)

Use cases are designed to capture ***functional*** requirements. They are a *poor* choice when:

- The system is dominated by non-functional requirements
- The system has ***few users***
- The system has ***few interfaces*** (for actors to interact with the system)

Tips for use case modelling

Consider the situation of *system use*

- It is critical to define the system boundaries
- Identify all actors
- Think about all the main tasks for which the system will be used
- Look for main alternative flows
- Always be prepared to ***revise*** your use case model
- Distinguish between use cases by frequency of use
- Only describe actions which involve interaction between actor and system
  - The ***internal processing*** that the system might undertake ***is not part*** of the use case specification. Internal working will be captured later, using other UML diagrams.

Part 1: Takeaway Messages

- Use cases are a form of *functional system requirements* engineering
- Use case models include:
  - Use case diagrams
  - Use case specifications, which in turn include:
    - Specifications of the main flows.
    - Specifications of the important alternative flows.
- We have looked at:
  - Use cases diagrams (subject, actors, use cases)
  - We looked at use case specifications:
    - Main flow
    - Alternative flows
    - Branching with *if*
    - Repetition with *for* and *while*
  - We also looked at Requirements tracing (linking user requirements with Use cases)

Further Reading

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 4.

31

Part 2: Advanced use case modelling

32

More relationships…

We have studied basic elements of use case diagrams, but there are relationships that we have still to explore:

- Actor generalisation
- Use case generalisation
- «include» – between use cases
- «extend» – between use cases

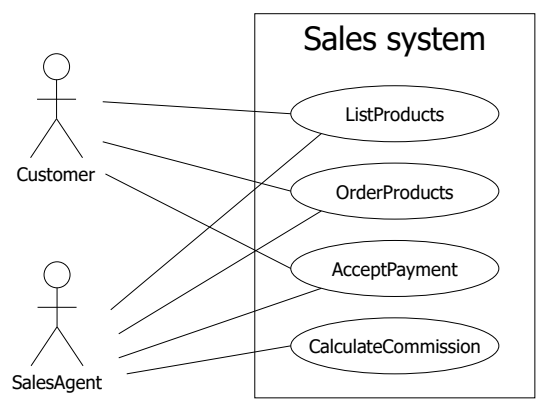
Actor generalization - example

The Customer and the Sales Agent actors are *very* similar

They both interact with "ListProducts", "OrderProducts", "AcceptPayment"

Additionally, the Sales Agent interacts with "CalculateCommission"

Our diagram is a *mess* – can we simplify it?

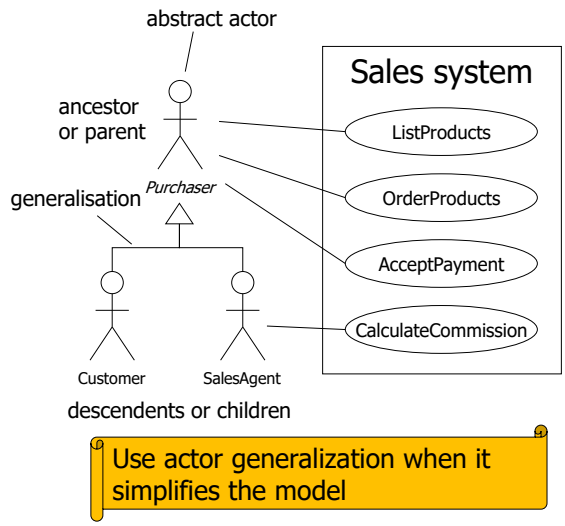


Actor generalisation

If two actors communicate with the same set of use cases in the same way, then we can express this as a generalisation to another (possibly ***abstract***) actor.

The descendent actors inherit the roles and relationships to use cases held by the ancestor actor.

We can substitute a descendent actor anywhere the ancestor actor is expected. This is the ***substitutability principle***.

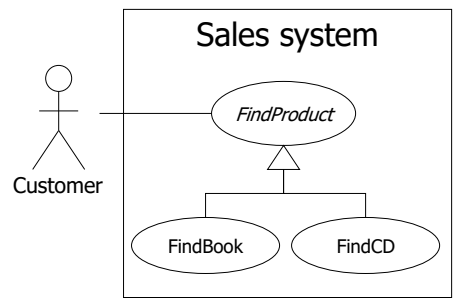


Use actor generalization when it simplifies the model

Use case generalisation

The ancestor use case must be a more general case of one or more descendant use cases
Child use cases are more specific forms of their parent
They can inherit, add and override features of their parent

Use case generalization semantics			
Use case element	Inherit	Add	Override
Relationship	Yes	Yes	No
Extension point	Yes	Yes	No
Precondition	Yes	Yes	Yes
Postcondition	Yes	Yes	Yes
Step in main flow	Yes	Yes	Yes
Alternative flow	Yes	Yes	Yes



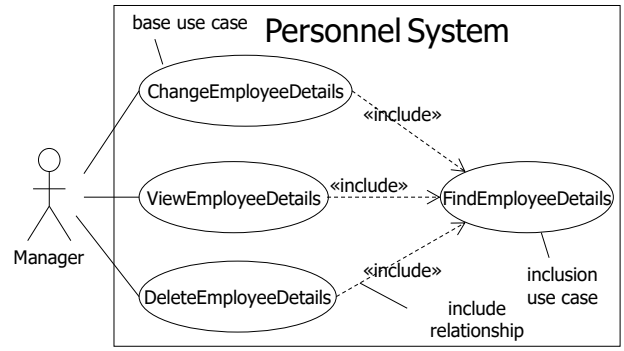
«include»

The base use case executes until the point of inclusion:
include(InclusionUseCase)

- Control passes to the inclusion use case which executes
- When the inclusion use case is finished, control passes back to the base use case which finishes execution

Note:

- Base use cases are ***not complete*** without the included use cases
- Inclusion use cases may be ***complete use cases***, or they may just specify a ***fragment of behaviour*** for inclusion elsewhere.



When use cases ***share common behaviour*** we can factor this out into a separate inclusion use case and «include» it in base use cases

«include» example

Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. include(FindEmployeeDetails). 2. The system displays the employee details. 3. The Manager changes the employee details. …
Postconditions: 1. The employee details have been changed.
Alternative flows: None.

Use case: FindEmployeeDetails
ID: 4
Brief description: The Manager finds the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. The Manager enters the employee's ID. 2. The system finds the employee details.
Postconditions: 1. The system has found the employee details.
Alternative flows: None.

Is anything missing in the previous example?

What if the included use case (i.e. the search) does not complete successfully and does not find an employee with matching details?

- In the example on the previous slide the Manager may type in the employee ID incorrectly and the search for employee record may fail. What happens then?

More generally, how shall I link two use-cases which are related (e.g. have an <<include>> relationship)?

- Consider using a 'session' (an element of design, but allows one to pass data between use cases)
  - Created when the user has logged-in
  - ***Passing data between use-cases***:
    - Creating a ***session***
    - Adding objects to the session
  - Then the system should refer to the session to find the result from the include/extend use case...
    - Step 2 of the main flow can be rewritten to refer to the session.

Revisiting the main use case: using if-else

Now the postcondition will be different for the two branch (if-else)

- the employee details get changed for the if branch
- the employee details are not changed if the else branch gets executed

Not good!

- Postconditions must be always true at the end of the use case flow!

Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Preconditions: 1. The Manager[illegible]on to the syste[illegible]
Main flow: 1. include(Fin[illegible] 2. If(session.employ[illegible] 2.1 The system [illegible]details. 2.2 The Ma[illegible]ails. 3. else 1. …<>
Postconditions: 1. The employee details have been changed. 2. The variable "session.employee_found" erased.
Alternative flows: None.

Modified FindEmployeeDetails

The results from a successful search are stored in session:
- "session.employee" contains the details of the employee found by the search .
- "session.employee_found" is set to true.

An alternative flow is specified with:
- "session.employee" set to NULL (no employee found with matching details).
- "session.employee_found" is set to false.

Use case: FindEmployeeDetails
ID: 4
Brief description: The Manager finds the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. The Manager enters the employee's ID. 2. The system finds the employee details and adds them to variable "session.employee". 3. The system sets variable "session.employee_found" to true.
Postconditions: 1. The system has found the employee details.
Alternative flows: No employee found.

Revisiting the main use case (2)

Consider this revised example:

The branching is moved to an <<include>> use case.

- The main flow only deals with the case of finding the employee's record.
- The other branch (employee record not found) is dealt with by an alternative flow, for which we can define a different post condition.

Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. include(FindEmployeeDetails). 2. System confirms that "session.employee_found" variable is set to true. 3. The system displays the employee details. 4. The Manager changes the employee details.
Postconditions: 1. The employee details have been changed. 2. The variable "session.employee_found" is erased.
Alternative flows: EmployeeRecordMissing

Alternative flow

Use case: ChangeEmployeeDetails:EmployeeRecordMissing
ID: 1.1
Brief description: System fails to locate an employee record.
Primary actors: Manager
Preconditions: 1. The Manager is logged on to the system. 2. "session.employee_found" is set to false.
Main flow: 1. The use case starts after the completion of step 1 of the main flow of use case ID 1. 2. The system notifies the actor that the employee record is not found. 3. Manager acknowledges the notification.
Postconditions: 1. The variable "session.employee_found" is erased.
Alternative flows: None.

43

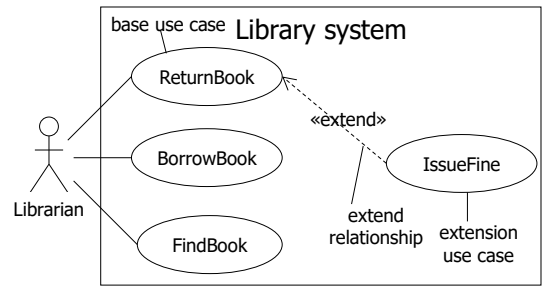
«extend»

«extend» is a way of adding new behaviour into the base use case by inserting behaviour from one or more extension use cases

- The base use case specifies one or more ***extension points*** in its flow of events

The extension use case may contain several insertion ***segments***

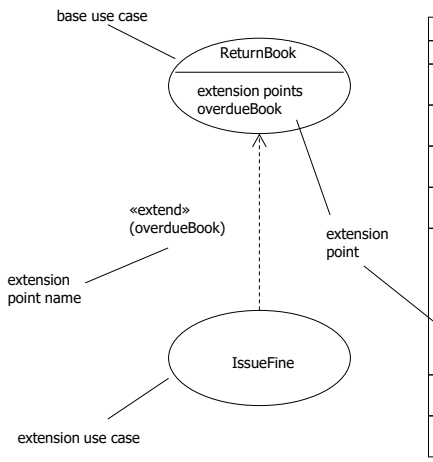
The «extend» relationship may specify *which* of the base use case extension points it is extending



The extension use case inserts behaviour into the base use case.
The base use case provides extension points, but *does not know* about the extensions.

44

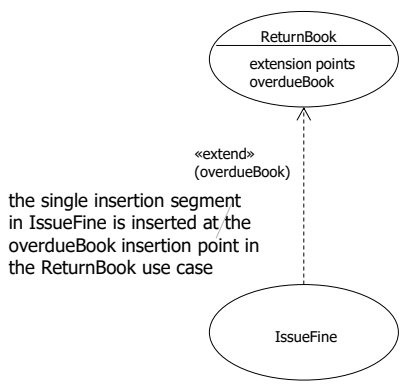
Base use case



Use case: ReturnBook
ID: 9
Brief description: The Librarian returns a borrowed book.
Primary actors: Librarian
Secondary actors: None.
Preconditions: 1. The Librarian is logged on to the system.
Main flow: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. extension point: overdueBook 4. The Librarian returns the book. ...
Postconditions: 1. The book has been returned.
Alternative flows: None.

There is an extension point overdueBook just before step 4 of the flow of events
Extension points are ***not*** numbered, as they are ***not*** part of the flow

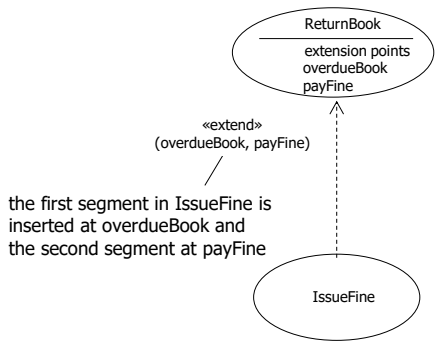
Extension use case



Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.

Extension use cases have one or more ***insertion segments*** which are behaviour fragments that will be inserted at the specified extension points in the base use case

Multiple extension points

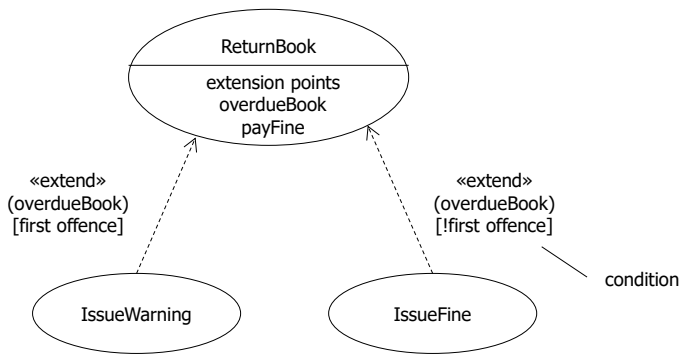


Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine. Segment 2: The Librarian accepts payment for a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.
Segment 2 preconditions: 1. A fine is due from the borrower.
Segment 2 flow: 1. The Librarian accepts payment for the fine from the borrower. 2. The Librarian enters the paid fine in the system. 3. The system prints out a receipt for the paid fine.
Segment 2 postconditions: 1. The fine is recorded as paid. 2. The system has printed a receipt for the fine.

If more than one extension point is specified in the «extend» relationship then the extension use case must have the ***same number*** of insertion segments

47

Conditional extensions



We can specify conditions on «extend» relationships

- Conditions are Boolean expressions
- The insertion is made if and only if the condition evaluates to true

48

Takeaway Messages (Part 2)

- We have learned about techniques for advanced use case modelling:
  - Actor generalisation
  - Use case generalisation
  - «include»
  - «extend»
- Use advanced features with discretion only where they make the model clearer!

49

Further Reading (Part 2)

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 5.

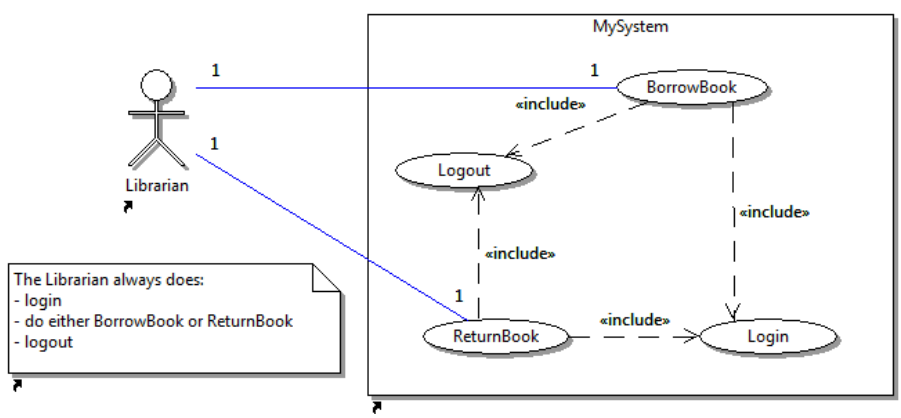
50

Part 3: Examples and common mistakes

51

An example: Librarian

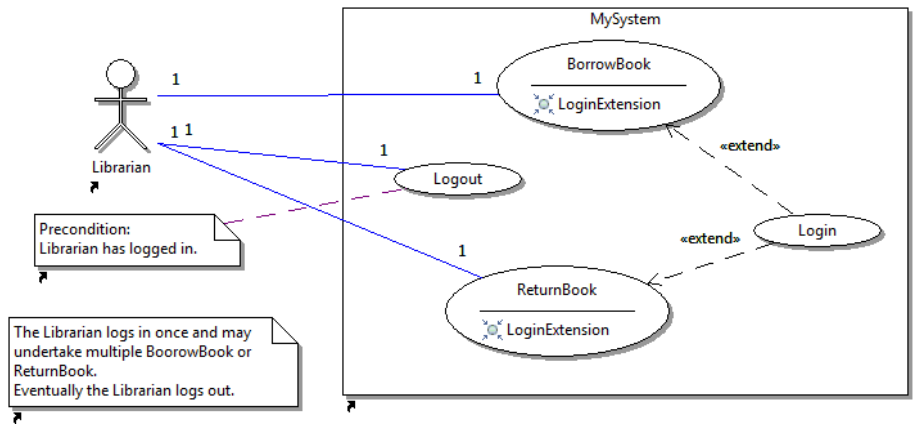
Version 1: using <<include>> relationship



52

An example: Librarian

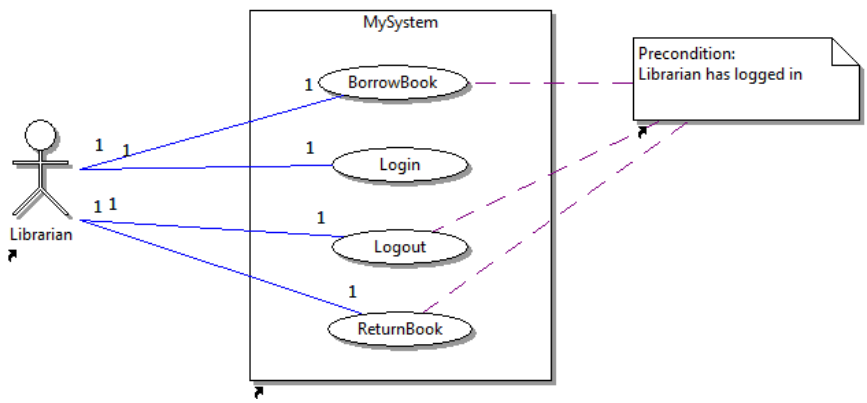
Version 2: using <<extend>> relationship



53

An example: Librarian

Version 3: expressing dependence as a precondition



54

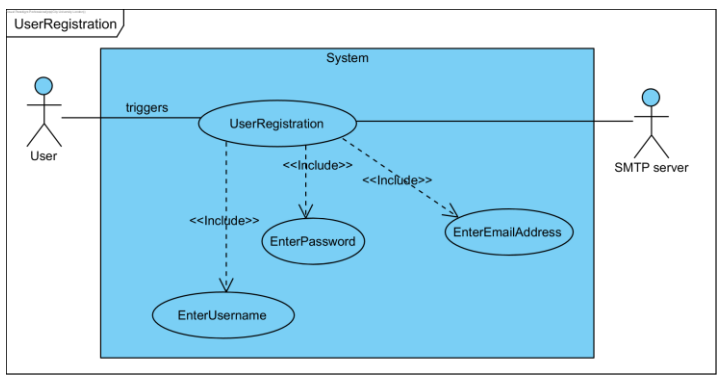
Examples with use case models

Some common mistakes

55

Example 1

Consider the following fragment:



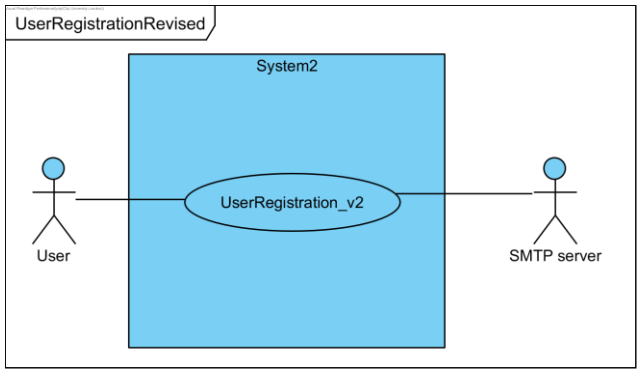
What is wrong with this model?

Use case: UserRegistration
ID: 1
Brief description: User registration to use the services offered by software/.
Primary actors: User
Preconditions: 1. The User is logged on to the system.
Main flow: 1. include(EnterUsernane). 2. include(EnterPassword) 3. Include(EnterEmailAddress) 4. System validates the user details 5. System creates a new User account
Postconditions: 1. A User account is created on the system with User details as provided.
Alternative flows: …

56

Example 1: Solution

The model should become:

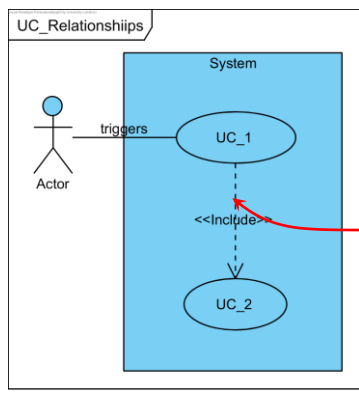


Use case: UserRegistration
ID: 1
Brief description: User registration to use the services offered by software/.
Primary actors: User
Preconditions: 1. The User is logged on to the system.
Main flow: 1. User enters Usernane. 2. User enters Password 3. User enters email Address 4. System validates the user details 5. System creates an new User account
Postconditions: 1. A User account is created on the system with User details as provided.
Alternative flows: <some can be defined here>

Steps 1, 2 and 3 should not be modelled as separate UCs. They are merely *steps* in the flow.

Example 2

Consider the following fragment:



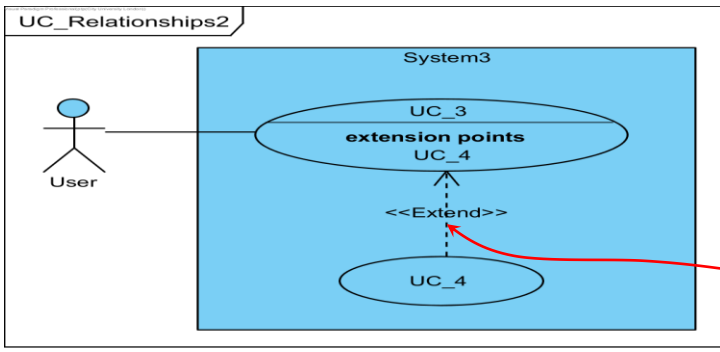
Use case: UserRegistration
ID: 2
Brief description: <some description>
Primary actors: Actor
Preconditions: 1. <some preconditions>
Main flow: 1. <do something> … Extension point: UC_2 …
Postconditions: 1. <specified post-conditions >
Alternative flows: …

What is wrong with this model?

The relationships <<include>> shown in the diagram between UC_1 and UC_2 does not match the Relationship in the specification (<<extend>> relationship is implied by using an extension point).

Example 2: Solution

Consider the following fragment:



Use case: UserRegistration
ID: 2
Brief description: <some description>
Primary actors: Actor
Preconditions: 1. <some preconditions>
Main flow: 1. <do something> … Extension point: UC_4 …
Postconditions: 1. <postconditions are specified>
Alternative flows: …

The type of relationship between UC_3 and UC_4 on the diagram and in the specification must be the same.

Takeaway Messages (Part 3)

- Simplicity of use case diagrams may be misleading, especially for novices:
  - Use case diagram should be clear and capture the essential elements, not every single step an actor may take when interacting with the system.
  - Too many «include» relationships usually indicate that the diagram is too detailed and should be simplified.
- Diagrams and specifications ***must be consistent***: they are two views on the same system
- «include» and <<extend>> have very different purpose
- «extend» and alternative flows serve different purposes in specifications:
  - <<extend>> capture *useful optional behaviour*
  - Alternative flows capture anomalies, errors, interruptions
    - Alternative flows *do NOT appear* in use case diagrams.

Further Reading (Part 3)

This material is not covered in the recommended textbook.

On Moodle you will find many more examples of use cases models (diagrams and specifications), e.g. in the model answers from the course works in the previous years.

Summary

- We have learned about UML use case modelling.
- Use case models are a form dealing with functional requirements.
- Use case diagrams allow one to define:
  - System boundary (Subject)
  - Actors
  - Use cases
  - Relationships between:
    - Actors may have a generalisation relationship
    - Use case may have several types of relationships:
      - <<include>>
      - <<extend>>
      - Generalisation
- Use case specifications are an important part use case model.
  - UML does not standardise a specific form of writing use case specifications.