COMP2009
Software Engineering

Use Cases
Part II

Use Case Modelling - Additional Features

- Scenarios.
- Relationships between use cases: stereotypes.
- Generalisations.
  - 2 actors or 2 use cases can be related by *generalisation*.
  - Generalisation between actors.
  - Generalisation between use cases.

Scenarios

- Each time an actor interacts with a system, the triggered use case instantiates a *scenario*.
- A scenario is an *instance* of a use case.
- A scenario corresponds to one possible interaction between the system and its actors.
- This interaction can be viewed as a *sequence of messages*.
- Each scenario thus corresponds to a specific path through a use case with no branching.

More on Scenarios

- Scenarios of the same use case can involve different behaviour.
  - The only thing they have in common is that they are all attempts to carry out the same task.
  - There can be one primary scenario plus one or more secondary scenarios.
  - Secondary scenarios correspond to alternatives, exceptions, errors, interrupts, etc.
- Scenarios can help surface requirements, facilitating an iterative and incremental approach to requirements specification.
- Scenarios can be documented as text alongside the use case.

Example Scenarios - Borrow Copy of Book

- **Scenario 1**
  - BookBorrower Joe borrows the Library's only copy of Using UML: Software Engineering with Objects & Components, when he has no other Book out on loan. The System grants the Loan and updates Joe's Status accordingly.
- **Scenario 2**
  - BookBorrower Ann tries to borrow the Library's 3rd copy of The Mythical Man Month, but is refused because she already has six Books out on loan, which is her maximum allowance. The system refuses the Loan and makes no change to Ann's Status.

Normal v. Alternatives v. Errors

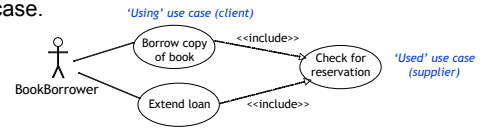
- A use case needs to capture alternatives and errors *as well as* the normal flow of events.
- Use scenarios to help identify all possible different outcomes of a use case.
- The alternatives/error flows need to be included when the system is designed and implemented.
  - Otherwise there will be missing behaviour.

Stereotypes

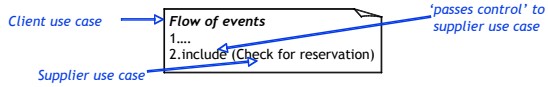
- Stereotypes are UML's way of attaching additional information to model elements.
  - Stereotypes make UML extensible.
- A stereotype is depicted in the following way: **«stereotype_name»**
- We use two pre-defined stereotypes to show relationships between use cases
  - **«include»**: Indicates a use case reused by another use case.
  - **«extend»**: Separates variant behaviour between use cases.

«include»

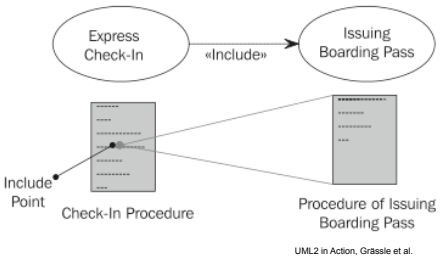
- Used to factor out common behaviour from use cases.
  - So that behaviour can be shared as a supplier use case.



- Supplier use cases are included in client use case specifications in the following way:



Include relationship



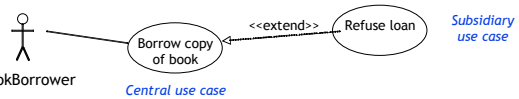
UML2 in Action, Grassle et al.

When to Use «include»

- Consider using an «include» relationship between use cases:
  - to show common functionality between use cases.
  - to avoid duplication by allowing sharing.
  - to show how the system can use a pre-existing element.
  - to document the fact that the project has developed a new reusable element.
- Dangers
  - Sharing may be difficult to realise in the design.
  - Visually harder to read.
  - Harder to keep up to date.

«extend»

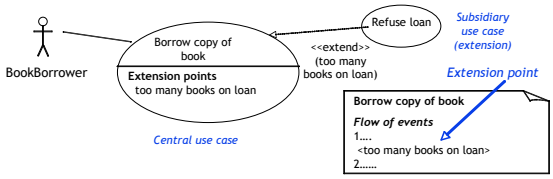
- A way of adding new behaviour to existing use cases.
- If a use case incorporates a number of significantly different scenarios, it is sometimes clearer to show them as one main case and a number of subsidiary cases.



- Included in the central use case specification using extension points.
  - But the central use case doesn't actually know *about* the extension (i.e. encapsulation).
  - Note direction of arrow.

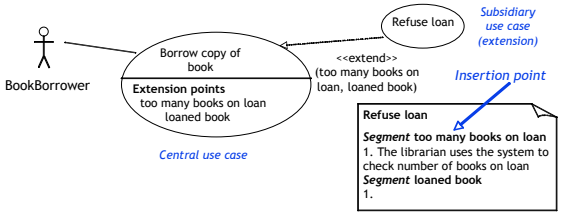
Extension Points

- Include in the description of the central use case
  - Condition under which the exceptional case applies.
  - *Extension point*: Point at which the condition is tested and the behaviour may diverge.
- Can have one or more extension points in the central case.

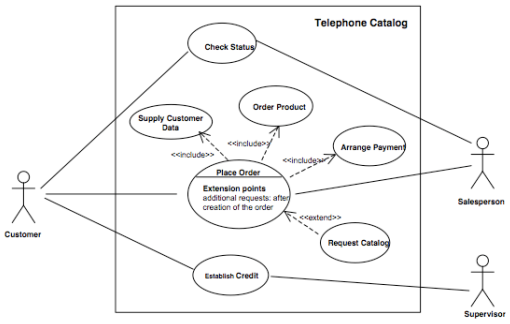


Insertion Points

- Insertion points are included in the specification of the subsidiary use case.
  - Specified as segments.



Another Example



When to Use «extend»

- Don't use it...
  - Unnecessarily complex most of the time.
- Consider using an «extend» relationship between use cases:
  - to reuse the same behaviour a number of times in different use cases.
  - when the details of this behaviour can be delayed or altered.
- Dangers
  - Visually harder to read, stakeholders won't understand.
  - Harder to keep up to date.

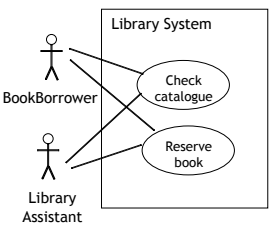
Beware...

- You don't have to use «include» or «extend».
- Keep your use cases straightforward.
- Don't fall into the trap of using them because you can.
  - Use cases are meant to be readable by stakeholders - could they understand your use of «include» or «extend»?

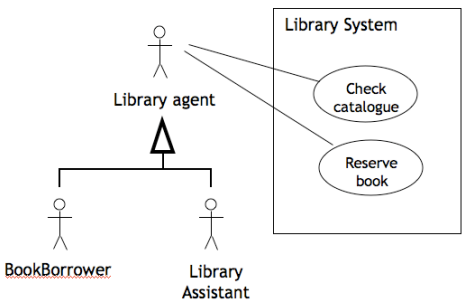
Actor Generalisation

- Used to factor out commonality between two actors.
- Look for crossed lines in a use case diagram and see if there are two actors communicating with same the set of use cases in the same way.

*The two actors can be generalised as a **Library Agent***

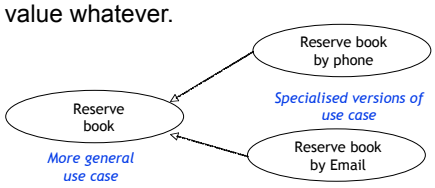


Actor Generalisation (2)

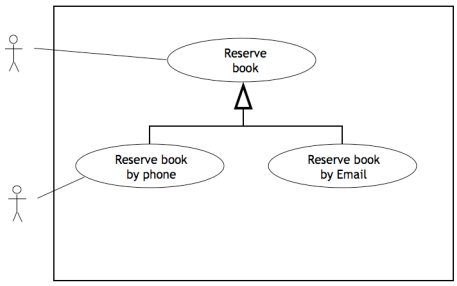


Use Case Generalisation

- In principle can be used to factor out commonality between two use cases.
  - Look for use cases that are specialisations of a more general case.
  - Children inherit from their parents and may add new features.
- In practice, use case generalisation is almost always of no value whatever.

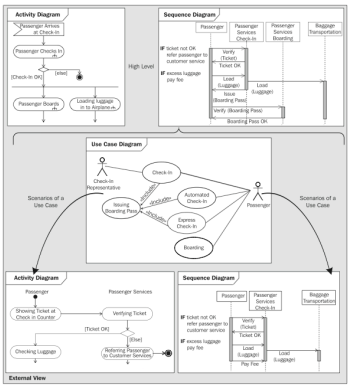


Use Case Generalisation (2)



Better off just having two use cases.

Using Sequence and Activity Diagrams

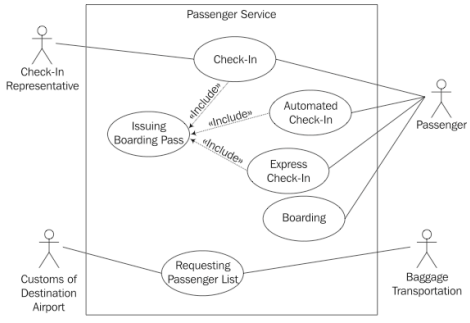


UML2 in Action, Grässle et al.

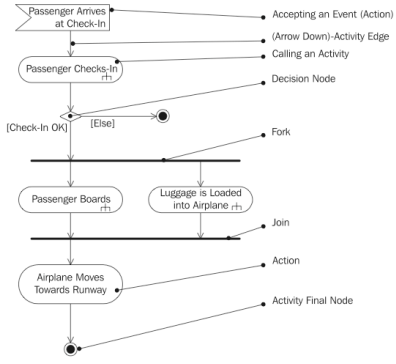
Use Case v. Activity v. Sequence Diagrams

- Use Case: show actors, use cases and their relationships. An overview.
  - But not procedures, sequences or alternative scenarios.
- Activity: describes procedures at various levels of detail.
  - Sequences, alternatives and parallel events.
- Sequence: chronological chain of interactions in a scenario.
  - Show information exchanged between entities (objects).
  - May not include every alternative, or parallel behaviour.

Use Case Diagram for Airport System



Activity Diagram "Passenger Services"



Description of the business process.

Covers several use cases.

UML2 in Action, Grässle et al.

Activity Diagram "Passenger Checks In"

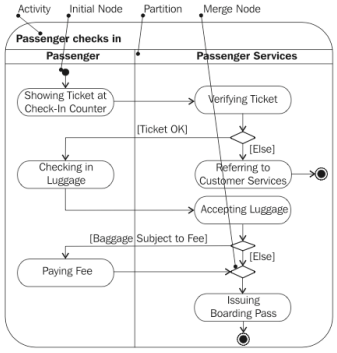
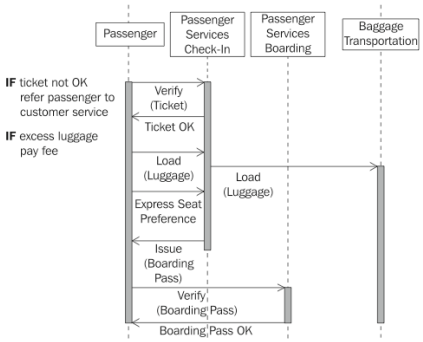


Diagram for a single use case.

UML2 in Action, Grässle et al.

Sequence Diagram "Passenger Services"



Another Use Case Example

- See ATM Withdraw Cash handout.

Common problems with Use Case Diagrams(1)

- System boundary box must be shown (i.e., the box around all the use cases, with the actors outside it).
- In a use case diagram there are no associations (lines) between use case icons, except for include and extend relationships (and possibly others that may have been defined).
- The use case diagram should be a single diagram showing all the use cases and actors in the system. For larger projects, beyond the scope of this coursework, you can have multiple diagrams showing different behavioural aspects of the application.
- Beware of over using includes and especially extends relationships between use cases.
- The extends relationship is more than just an arrow on the use case diagram. The extension point(s) must be clearly shown in the diagram and the use case specifications.
- Association lines between actors and use cases do not have arrow heads.

Common problems with Use Case Diagrams(2)

- Actor icons are placed outside of the system boundary box, use case icons inside.
- Not UML notation! Don't invent your own notation and expect others to understand it correctly.
- If using include or extends associations between use case icons make sure the arrow head is put on the correct end of the association.

Common problems with Use Cases(1)

- A use case is a dialogue between the system and an actor (or actors). The main flow should reflect this by describing the use case in terms of the system doing something, an actor responds or enters input, the system responds, etc.
- A step in the main flow should clearly state a single action or response. Beware of lumping a multi-part action or behaviour into a single step.
- Use cases describe the system behaviour and actor interaction only. Don't include behaviours outside of the software system (for example, the hotel manager going to a room to do something).
- The main flow should usually start with the step "The use case starts when ...", so there is a clear start state for the use case.
- An actor is an entity that interacts directly with the system via a use case. A guest who interacts with the receptionist but not the software system would not be an actor.

Common problems with Use Cases(2)

- Steps in the main flow should name the relevant actors, specified earlier in the use specification, not introduce other names/labels. For example, the main flow should refer to the Receptionist and/or Hotel Manager, not the “user”.
- The main flow is too short to be of use. Either expand it (if too vague or high-level) or remove the use case, possibly merging it with another use case.
- The main flow should be a numbered sequence.
- The main flow should have steps that directly involve the named actors, i.e., the actor performs some action in the flow.
- The Alternative Flows section names the use cases that provide alternate outcomes for the use case. Each Alternative Flow should be documented as a use case in its own right, suitably labelled as an alternative flow. In addition, the step in the use case main flow where the alternative flow starts should state that the alternative is being followed.

Common problems with Use Cases(3)

- The actor(s) don’t appear in main flow. Rewrite flow or delete use case.
- Too few use cases to provide a proper description of the system.
- The main flow stops without completing all the steps implied by the use case.
- Use cases should not include design or implementation information. For example, it is valid to state that “the input is stored in the database” but not to state “the input is stored as a row in the Person table, using an SQL update”. Also use cases should not refer to design classes.
- A description of a scenario has been provided instead of a use case. A scenario is an instance of a use case, a specific example of one execution of the use case. The use case should describe the sequence of events (main flow) from which the scenario can be instantiated.

Summary

- Scenarios
- Stereotypes
- Generalisation
- Keep use cases straightforward!

- See the course text book, UML2 and the Unified Process, for more information about and examples of use cases.