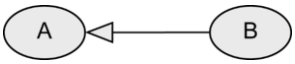
OO Analysis: Use Case Model
Creating Use Case Diagram

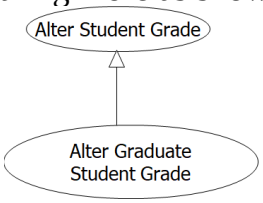
Factoring Use Cases (Relationships between Use Cases)
You may notice common behavior across different use cases. Factoring these use cases would make it possible to define such behavior only once and reuse it whenever required just like a subroutine or function defined in a program and used in different places in the flow of the program. So it is desirable to factor out common usage such as error handling from a set of use cases. Complex use cases need to be factored into simpler use cases. UML offers following three mechanisms for factoring of use case:

- Generalization: Use cases that are specialized versions of other use cases.
- Include: Use cases that are included as parts of other use cases and it enable to factor common behavior.
- Extend: Use cases that extend the behavior of other core use cases and it enable to factor variants

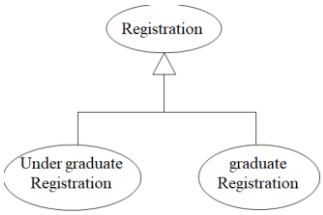
Now we describe the factoring of use cases in detail by using simple examples.
Generalization: Used to group common properties and common behavior of different use cases in a parent use case. Use cases that are specialized versions of other use cases. If a use case A generalizes a use case B, B inherits the behavior of A, which B can either extend or overwrite. B also inherits all relationships from A. This relationship is depicted as shown below. Use case B adopts the basic functionality of use case A but decides itself what part of A is executed or changed



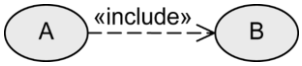
Generalization can be used when one use case that is similar to another, but does something slightly differently or something more as shown in the following diagram:



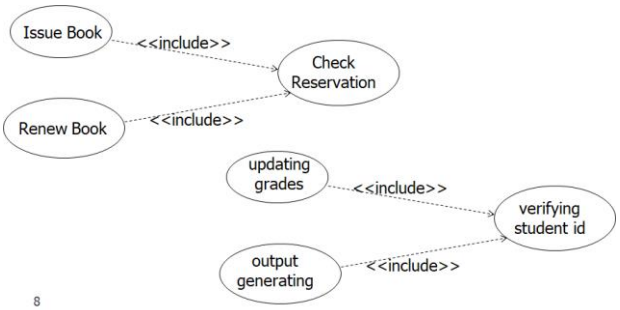
There is parent-child relationship and the child use case inherits the behavior and meaning of the parent use case. The child may add to or override the behavior of its parent as shwin below:



«include»: If a use case A includes a use case B, it is represented as a dashed arrow from A to B labeled with the keyword «include» as shown below:



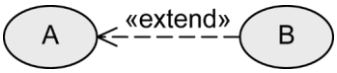
Behavior of B is integrated into the behavior of A. The use of «include» is analogous to calling a subroutine in a procedural programming language. This relationship depicts use cases that are included as parts of other use case as shown below:



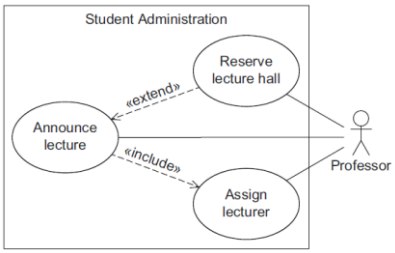
8

For Library example use cases depicted above, before a book is Issued or a book is Renewed, better to check whether it has been Reserved by any member. Similarly, for updating grades or generation output (result), student-id is verified.

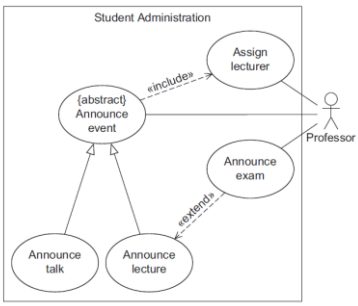
«extend»: If a use case B is in an «extend» relationship with a use case A, then A may (! mandatory) use the behavior of B. Both use cases can also be executed independently of one another. A is referred to as the base use case and B as the extending use case as shown below:



An «extend» relationship is shown with a dashed arrow from the extending use case B to the base use case A. Note that the arrow indicating an «extend» relationship points towards the base use case, whereas the arrow indicating an «include» relationship originates from the base use case and points towards the included use case. In student administration example use case diagram shown, the use cases Announce lecture and Assign lecturer are in an «include» relationship, whereby Announce lecture is the base use case.
Whenever a new lecture is announced, the use case Assign lecturer must also be executed.

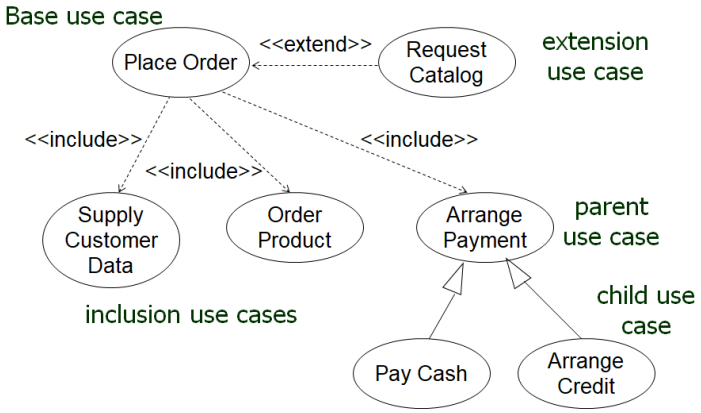


The actor Professor is involved in the execution of both use cases. Lecturers can also be assigned to an existing lecture as the included use case can be executed independently of the base use case.
Sometimes an abstract use case is defined and is labeled {abstract}. The abstract use case cannot be executed directly; only the specific use cases that inherit from the abstract use case are executable. In the student administration example use case diagram shown below the abstract use case Announce event passes on its properties and behavior to the use cases Announce lecture and Announce talk.



As a result of an «include» relationship, both use cases must execute the behavior of the use case Assign lecturer. When a lecture is announced, an exam can also be announced at the same time. Both use cases inherit the relationship from the use case Announce event to the actor Professor Generalization allows us to group the common features of the two use cases Announce lecture and Announce talk.

Use cases relationships in summarized form are depicted in the following diagram:

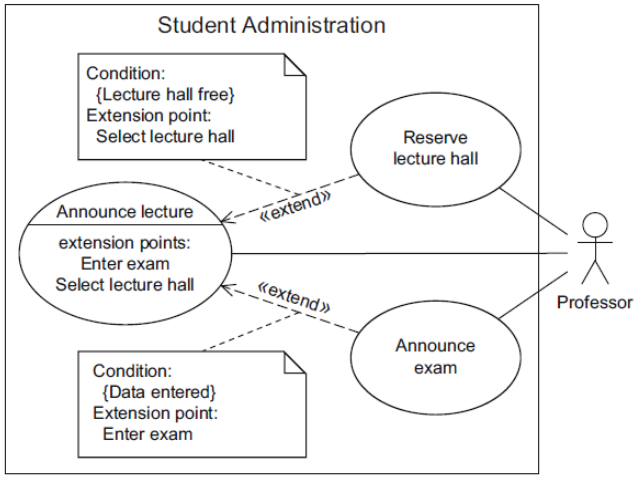


Condition

A condition that must be fulfilled for the base use case to insert the condition behavior of the extending use case can be specified for every «extend» relationship. The condition is specified, within curly brackets, in a note that is connected with the corresponding «extend» relationship. A condition is indicated by the preceding keyword Condition followed by a colon.

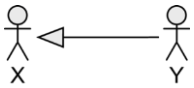
Extension points

By using extension points, you can define the point at which the behavior of the extending use cases must be inserted in the base use case. The extension points are written directly within the use case symbol and have a separate section that is identified by the keyword extension points. If a use case has multiple extension points, these can be assigned to the corresponding «extend» relationship via specification in a note similarly to a condition.
In the student administration example use case diagram shown below the use case Announce lecture the extension points have a separate section that is identified by the keyword extension points. A lecture hall can only be reserved if it is free. Furthermore, an exam can only be created if the required data has been entered.

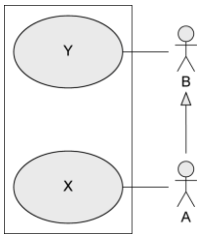


Relationships between Actors

Actors often have common properties, and some use cases can be used by various actors. To express this, actors may be depicted in an inheritance relationship (Generalization) with one another. Actor Y inherits from actor X and is depicted as shown below:

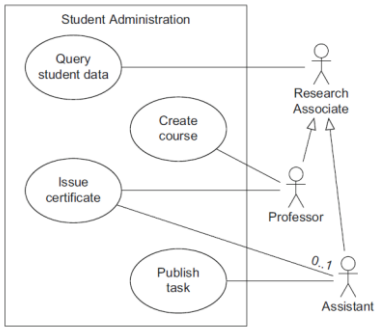


Generalization relationship expresses an “is a” relationship. When an actor Y (sub-actor) inherits from an actor X (super-actor), Y is involved with all use cases with which X is involved as shown below:



In above diagram, actor A can interact with use case X and Y, whereas actor B can only interact communicate with use case Y.

In the student administration example use case diagram shown below the actors Professor and Assistant inherit from the actor Research Associate, which means that every professor and every assistant is a research associate.



Every research associate can execute the use case Query student data and only professors can create a new course (use case Create course). In contrast, tasks can only be published by assistants (use case Publish task). To execute the use case Issue certificate an actor professor is required; in addition, an actor Assistant can be involved optionally, which is expressed by the multiplicity 0..1

We create use case diagram for an information system of functionality of a student office in accordance the student office of a university with the following specification:

- Many important administrative activities of a university are processed by the student office. Students can register for studies, enroll, and withdraw from studies here.
- Students receive their certificates from the student office. The certificates are printed out by an employee. Lecturers send grading information to the student office. The notification system then informs the students automatically that a certificate has been issued.
- There is a differentiation between two types of employees in the student office:
    - those that are exclusively occupied with the administration of student data (service employee, or ServEmp)

    - those that fulfill the remaining tasks (administration employee, or AdminEmp) whereas all employees (ServEmp and AdminEmp) can issue information.
- Administration employees issue certificates when the students come to collect them. Administration employees also create courses. When creating courses, they can reserve lecture halls.

To create a use case diagram from this simplified specification, we perform following activities:

- First identify the actors and their relationships to one another.
- Then determine the use cases and their relationships to one another
- Finally, associate the actors with their use cases

Our objective is to create Use-Case Model of the information system supporting the employees of a student office rather than modeling the functionalities the student office provides for the students.

Identify the Actors (Relationships to one another)

By looking at the textual specification, we can identify five potential actors:

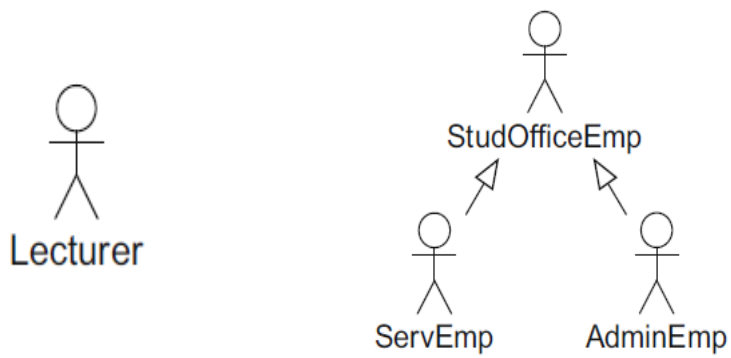
Lecturer
Student
Notification System
employees of the types ServEmp and AdminEmp

As both types of employees demonstrate common behavior, namely issuing information, it makes sense to introduce a common super-actor StudOfficeEmp from which ServEmp and AdminEmp inherit.

We assume that the Notification System is not part of the student office, rather notification is sent by lecturer action “send certificate” so we do not consider it as an actor (outside the system boundary).

Since we are modeling the information system supporting the employees of a student office, so we will not consider student as an actor to collect certificate.

Final list of actors and their relationships are illustrated below:



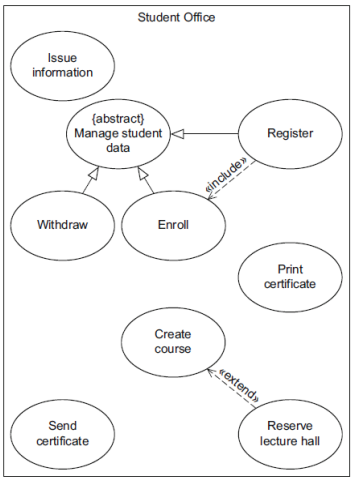
Identify Use Cases (relationships to one another)

To identify use cases, we determine which functionalities the student office must fulfill. From the specification, we have the functions Register, Enroll, and Withdraw. We could group these in one use case Manage student data as they are all performed by an actor ServEmp.

Since certificate information (in the form grade) is sent to student office by the lecturer, where a certificate is printed, so we have Print certificate and Send certificate use cases. From system specification we can identify Issue information, Reserve lecture hall, and Create course use cases.

- There is a generalization relationship between Manage student data an abstract use case and Register, Enroll, and Withdraw use cases.
- There is a relationship between Reserve lecture hall and Create course use case. This relationship is that of an extension, where Reserve lecture hall extends the use case Create course.
- Register and Enroll use cases have dependency relationship, which is expressed as include relationship, where Enroll use case is included with Register use case.

Final list of Use Cases and their relationships are illustrated below:



Association between Actors and their Use Cases

We have four actors and nine use cases, where actors directly or indirectly interact with use cases as depicted below: