REPUBLIC OF CAMEROON

Peace- Work-Fatherland

REPUBLIQUE DU CAMEROUN

Paix-Travail-Patrie

THE UNIVERSITY OF BAMENDA

UNIVERSITE DE BAMENDA

THE COLLEGE OF TECHNOLOGY

(COLTECH)

ECOLE DE TECHNOLOGIE

COMPUTER ENGINEERING

SOFTWARE ENGINEERING

INFORMATION SYSTEMS:

UML METHODOLOGY

Failing to plan is planning to fail.

MBAH LESKY TAGWANG

UBA21PB015

2021/2022

INTRODUCTION

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems. UML combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the software development life-cycle and across different implementation technologies

What are models?

A model is a description of the problem we are set to solve. It simplifies the reality by using entities and relationships in the problem domain. A problem domain describes not only a particular problem but also the conditions under which the problem occurs. It's therefore a description of a problem and the relevant context of that problem.

A model shows us what the problem is and how we are going to tackle it. We may use diagrams, text, or any other agreed form of communication to present the model.

Models visualize the system we are about to build.

A modeling language, therefore, is a language for describing models. Modeling languages generally use diagrams to represent various entities and their relationships within the model.

UML was created to fulfill these tasks:

- To represent all parts of a project being built with object-oriented techniques.
- To establish a way to connect ideas, concepts and general design techniques with the creation of object-oriented code
- To create a model that can be understood by humans and also by computers - so that a computer can generate a major portion of the application automatically

UML accomplishes these tasks by having a series of different models. Each model represents a different view of the project. Some models are built from others, so there is a logical sequence in which the models are built.

The building blocks of the UML are things and relationships:

- Things or objects in the UML describe conceptual and physical elements in the application domain
- Relationships connect things together

These two elements are brought together in UML diagrams to help us visualize things and their relationships in a well-structured format.

UML diagrams represent two different views of a system model:

- **Static (or structural) view** This view emphasizes the static structure of the system using objects, attributes, operations, and relationships. Ex: Class diagram, Composite Structure diagram.
- **Dynamic (or behavioral) view** This view emphasizes the dynamic behavior of the system by showing how objects interact and changes to the internal states of objects. Ex: Sequence diagram, Activity diagram, State Machine diagram.

UML Diagrams

There are quite a few UML diagrams that we can use when designing our applications, and we can pick and choose those which will be of most use to us. However, there is a basic core set of diagrams that we will almost certainly use. This core set of diagrams includes:

- Use Case Models
- Interaction Diagrams
- Activity Diagrams
- Class Diagrams

We'll now run through these types of diagram. You'll notice, as we run through them, that some diagram types have sub-types themselves (such as collaboration and sequence diagrams).

1. Use case diagrams

The use case model translates the user's needs into an easy to understand model. It Describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases. The user may be an individual or an external system and is known as an actor. So, in a nutshell, the use case model is a representation of how the system, or part of the system, works from the actor's point of view. Use case models can be built from interviews with the user, and are the first step in converting the user's needs and requirements into a useful model.

Use cases are detailed enough to include all of the information on the project, but simple enough for even the most technically challenged user to understand. Use cases can also be associated with business rules, which explain special rules, related to the use case. Let's take an example. In an order entry application, the use cases could include descriptions of various sub-parts of the system. These sub-parts, that together could make up the whole system, could be such things such as login in, creating a new student account or registering a course. For the use case Create New Student, there could be a verbal description of the process of creating a new student that looked as follows.

Use case: create new student

Overview
The main purpose of this is to create a new student account.

Primary actor
Student

Starting point
The use case starts when the actor makes a request to create a new student or login

End point
The actor's request to create or login into account is either completed or cancelled

Flow events
The actor is prompted to enter information that defines the students, such as Name, Address, etc. The actor will then enter the information on the student. The actor can choose to save the information or cancel the operation. If the actor decides to save the information the new student is created in the system, and the list of students is updated

Alternative Flow of Events
The actor attempts to add a student that already exists. The system will notify the user and cancel the create operation.

Measurable Result
A student is added to the system

Business Rules
Students
Student Fields
Restrict Student Create

Use Case Extensions

This is a verbal description of what happens when a potential user of the program we want to design needs to create a new student. Possible flows of events are identified to explain how the system can get from the start point to a definite end point. Measurable results are defined, and some business rules are created. One of these business rules is called Restrict Student Create and might be written as follows:

BUSINESS RULE: RESTRICT STUDENT CREATE

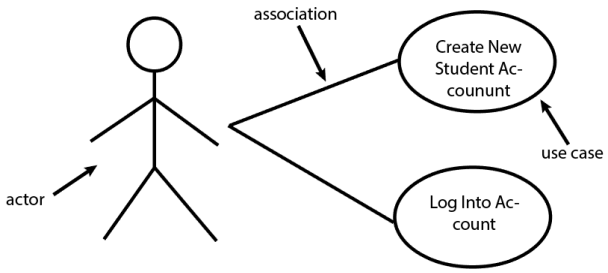
Overview
This rule is for when a Student is added to the system

Business Rule Type
Requirement

Business Rule
Each Student must have a unique StudentID
Each Student should only be listed once in the system

Derived Business Rules
None

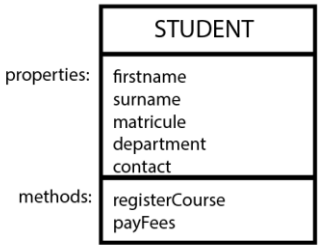
To put it in simpler terms, the use case model describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.



2. Class Diagrams.

The class diagram shows how the different entities (people, things, and data) relate to each other; in other words, it shows the static structures of the system. Class diagrams can also be used to show implementation classes, which are the things that programmers typically deal with. Class diagrams are built from use case, activity and sequence diagrams which we will see later. These classes are use to create objects (specific people or things). Classes have attributes (properties) and methods.

An example of class diagram:

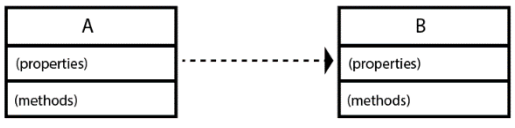


Relationships

Relationships between classes are generally represented in class diagrams by a line or an arrow joining the two classes. UML can represent the following, different sorts of object relationships.

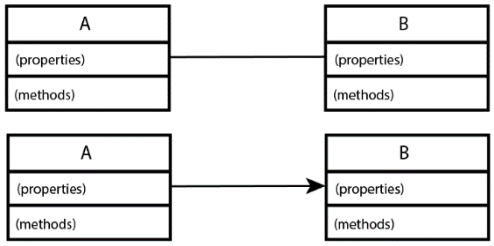
- **Dependency**

  If class A depends on class B, then this is option by a dashed arrow between A and B, with the arrowhead pointing at B:



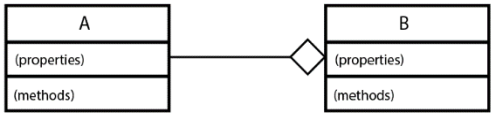
- **Association**

An association between A and B is shown by a line joining the two classes: If there is no arrow on the line, the association is taken to be bi-directional. A unidirectional association is indicated like this:



- **Aggregation**

An aggregation relationship is indicated by placing a white diamond at the end of the association next to the aggregate class. If B aggregates A, then A is a part of B, but their lifetimes are independent:

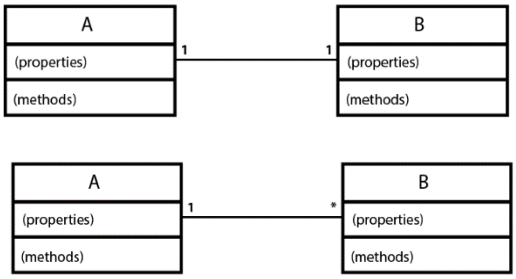


- **Composition**

Composition, on the other hand, is shown by a black diamond on the end of association next to the composite class. If B is composed of A, then B controls the lifetime of A.

Multiplicity

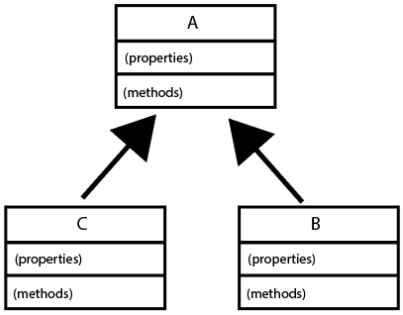
The multiplicity of a relationship is indicated by a number (or *) placed at the end of an association. The following diagram indicates a one-to-one relationship between A and B: This next diagram indicates a one-to-many relationship: A multiplicity can also be a range of values. Some examples are shown in the table below:



1	**One and only one**
*****	**Any number from 0 to infinity**
0..1	**Either 0 or 1**
n..m	**Any number from n to m inclusively**
1..*	**Any positive number**

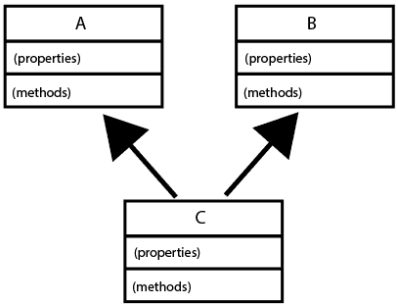
Inheritance

An inheritance (generalization/specialization) relationship is indicated in the UML by an arrow with a triangular arrowhead pointing towards the generalized class. If A is a base class, and B and C are classes derived from A, then this would be represented by the following class diagram:



Multiple Inheritance

The next diagram represents the case where class C is derived from classes A and B



INTERACTION DIAGRAMS

Interaction diagrams are the next step of the UML design process. Interaction diagrams concentrate on showing how objects or things in the system interact with each other to give a dynamic view of the system. There are two basic types of interaction diagram.

- Sequence diagrams
- Collaboration diagrams

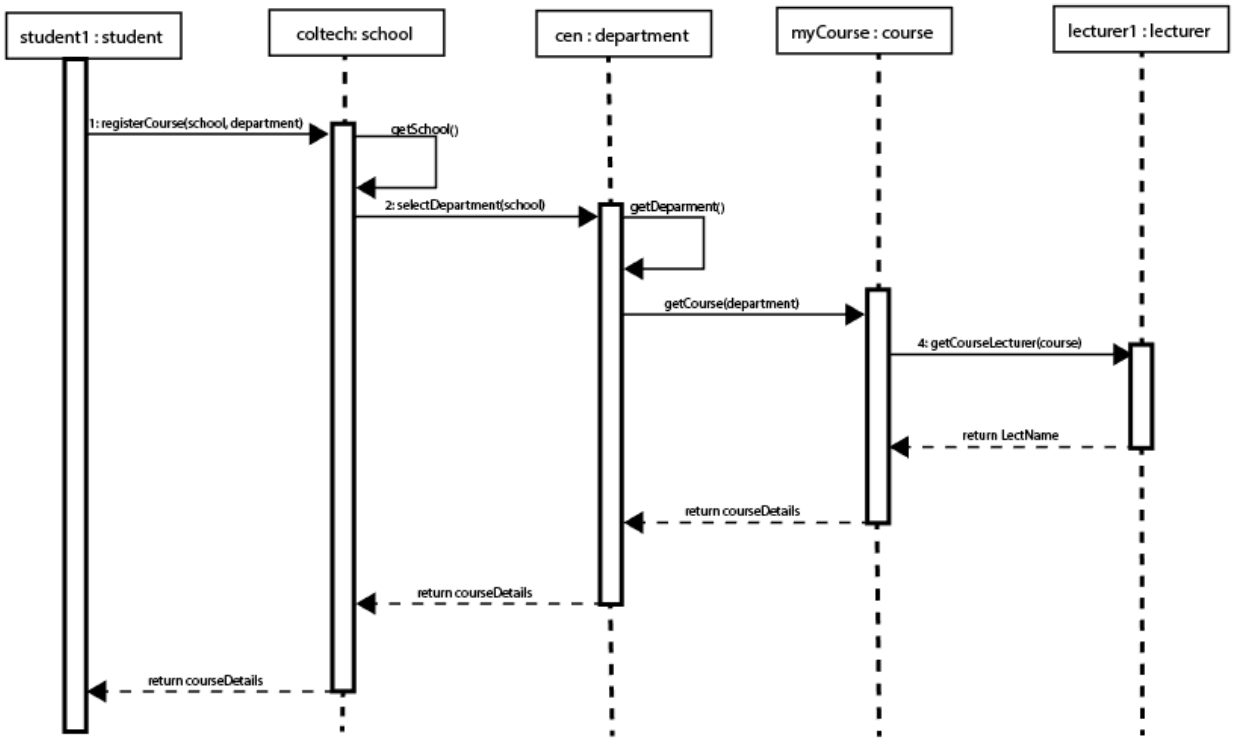
Essentially these both model the same information, except that sequence diagrams emphasize time ordering whereas collaboration diagrams spatial or structural organization

3. Sequence Diagrams

This type of diagram can be used to convert the written use case models that we saw in the previous section into a clearer visual model. This visual model will show how the objects associated with a particular use case communicate with each other and with users over time. A sequence diagram has two dimensions: The vertical dimension shows the sequence of

messages/calls in the time order that they occur; the horizontal dimension shows the object instances to which the messages are sent.

A sequence diagram is very simple to draw. Across the top of your diagram, identify the class instances (objects) by putting each class instance inside (see diagram below). In the box, put the class instance name and class name separated by a space/colon/space " : " (e.g., student1 : student). If a class instance sends a message to another class instance, draw a line with an open arrowhead pointing to the receiving class instance; place the name of the message/method above the line. Optionally, for important messages, you can draw a dotted line with an arrowhead pointing back to the originating class instance; label the return value above the dotted line.
Reading a sequence diagram is very simple. Start at the top left corner with the "driver" class instance that starts the sequence. Then follow each message down the diagram.



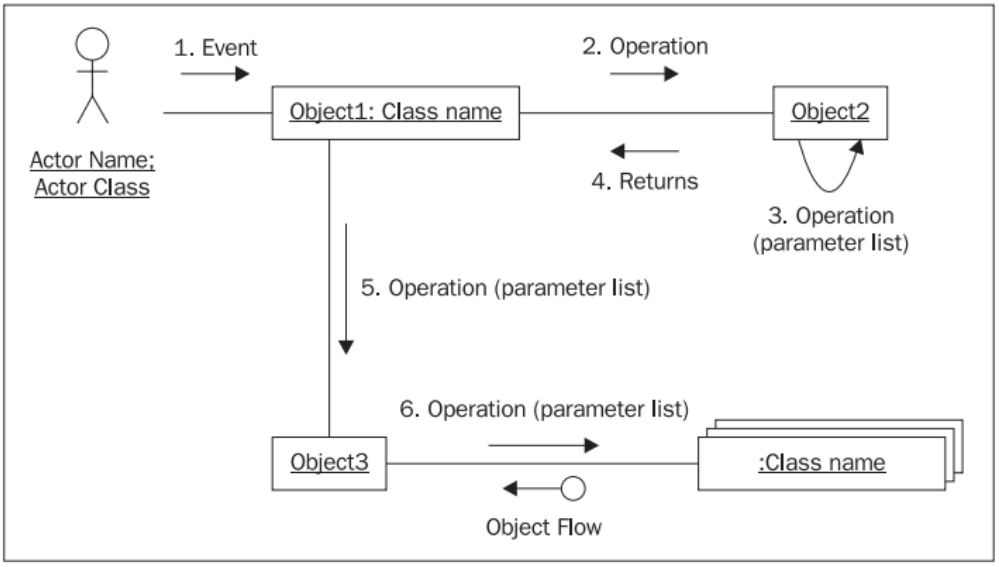
Sequence diagram to register a course (use case).

The rectangles show their lifetimes.

Sequence diagrams map out every possible sequence of events that can be performed within each use case, including correct and incorrect paths. The correct paths in the sequence diagrams can be used to design the GUI of the project as they show what the user will need to do to interact with the application. Incorrect sequences will later be used to map out errors and how to handle these errors.

4. Collaboration diagrams

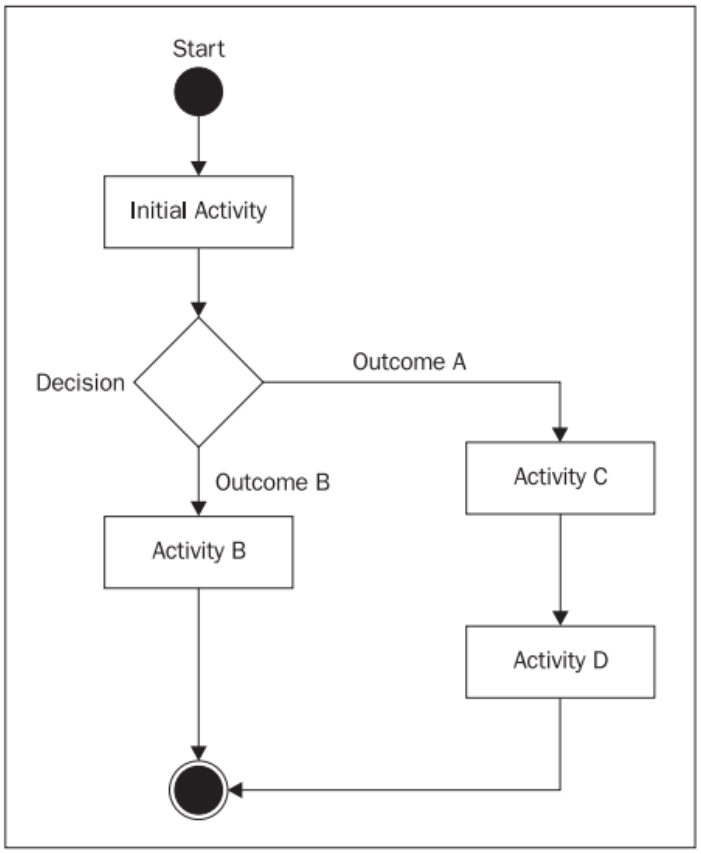
Collaboration diagrams are also built from the use cases - but this time the emphasis is on the spatial distribution of the objects involved. This is not to say that there are no temporal elements in collaboration diagrams, since the sequence of events is mapped using numbers. There are times when collaboration diagrams can make good sense - especially if we find that we want to emphasize a set of objects themselves rather than any sequence of events between them.



5. Activity diagrams

Activity diagrams take the information available from the collaboration and sequence diagrams that we've just looked at, and present that information in a more detailed fashion. The purpose of activity diagrams is now to show the inner workings for a particular object.

Activity diagrams can map out a method or property showing what that method or property has to do in a step-by-step manner. Activity diagrams are similar to flowcharts in algorithms. This detailed map can then be used to explore the best method of coding a method or property, check for missing or unnecessary sections, and as a guide to writing the code. Here is a sample activity diagram. This activity diagram simply specifies what Activity is to be initiated at a certain Decision point, depending on the outcome of that Decision.



Other UML diagrams are:

6. Component Diagram

A component diagram provides a physical view of the system. Its purpose is to show the dependencies that the software has on the other software components (e.g., software libraries) in the system.

7. Deployment Diagram

The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other. Since the diagram models the physical runtime, a system's production staff will make considerable use of this diagram.

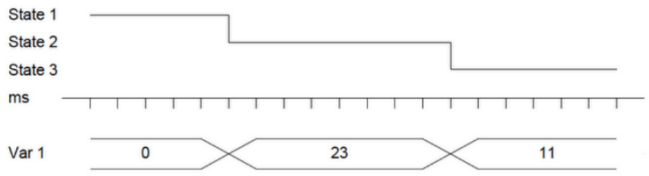
8. Object Diagram

Object Diagram Shows a complete or partial view of the structure of an example modeled system at a specific time. It is created from the class diagram.

9. Package Diagram

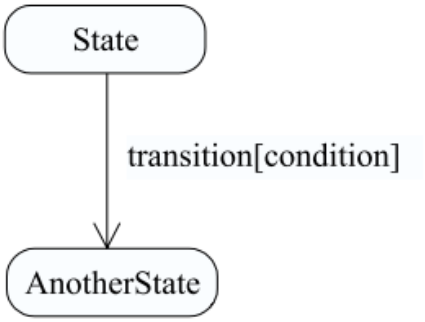
Describes how a system is split-up into logical groupings by showing the dependencies among these groupings.

10. Timing Diagram



A specific type of interaction diagram where the focus is on timing constraints. Timing diagrams model sequence of events and their effects on states and property values. Time flows along a horizontal axis from left to right. They can be used to show method execution profiling or concurrency scenarios.

11. State Machine Diagram



states of objects are represented as rectangles with rounded corners. The transition between different states is represented as an arrow between states, and a condition of that transition occurring may be added between square braces. The condition is called guard. State machine diagrams describes the states and state transitions of the system.

References

Unified Modeling Language

http://en.wikipedia.org/wiki/Unified_Modeling_Language

UML basics: The component diagram

http://www.ibm.com/developerworks/rational/library/dec04/bell/

Practical UML: A Hands-On Introduction for Developers

http://wwwis.win.tue.nl/2R690/together/

OO – UML Behavior Diagrams

http://giuliozambon.blogspot.com/2010/09/oo-uml-behavior-diagrams.html

UML 2 Tutorial

http://www.sparxsystems.com/resources/uml2_tutorial/

The official UML Web site.

http://www.uml.org