Table des matières

[CHAPTER 2: UNIFIED MODELING LANGUAGE(UML) 2](#_Toc182002646)

[INTRODUCTION 2](#_Toc182002647)

[I. SCENARIO 2](#_Toc182002648)

[I.1 MODELING LANGUAGES 2](#_Toc182002649)

[I.2 OBJECT ORIENTED 3](#_Toc182002650)

[I.3 MODELS 3](#_Toc182002651)

[I.4 LIFE CYCLE OF A SOFTWARE 3](#_Toc182002652)

[I.5 HISTORY OF UML 4](#_Toc182002653)

[I.6 OBJECT APPROACH 4](#_Toc182002654)

[II. THE UP (Unified Process) 5](#_Toc182002655)

[II.1 CHARACTERISTICS OF THE UP 5](#_Toc182002656)

[II.2 UP LIFE CYCLE 6](#_Toc182002657)

[II.3 EXAMPLE OF A SOFTWARE DEVELOPMENT PROCESS THAT IMPLEMENTS THE UP 8](#_Toc182002658)

[III. THE VIEWS 9](#_Toc182002659)

[IV. DIAGRAMS 10](#_Toc182002660)

[IV.1 BEHAVIOURAL DIAGRAMS 11](#_Toc182002661)

[IV.1.1 USE CASE DIAGRAMS 11](#_Toc182002662)

# CHAPTER 2: UNIFIED MODELING LANGUAGE(UML)

## INTRODUCTION

It is the best consolidation unified modelling practices that has being established over the years in the use of modelling languages. UML enables us to present the varying widing aspects of a software system (e.g. requirements, data structure, data flows and information flows) within a single framework using object oriented concepts.

# SCENARIO

Imagine that you want to develop a software system that a customer has ordered from you. One of the first challenges you are confronted with is clarifying what the customers actually wants and whether you have understood the customers exact requirements for the prospective system. This first step is already critical for the success or failure of your project. The question is how do you communicate with your customer. Natural language is not necessarily a good choice as it imprecise and ambiguous. Misunderstandings can easily arise and there is a very great risk that people with different backgrounds (e.g. computer scientist and business managers) will talk at cross-purposes which can have serious consequences. What you need is to able to create a model for your software. This model highlights the important aspects of the software in a clear form of notation that is a s simple as possible but abstract from irrelevant details just like models in architecture e.g. construction plans.

A construction plan for a building contains information such as the floor plan. Construction materials to be used are not specified at this point in time, they are irrelevant and will make the plan more complicated than necessary. A separate plan is created for this aspect to avoid presenting too much information at once.

## MODELING LANGUAGES

Modelling languages are precisely used for this scenario and clearly define rules for a structured description of a system. These languages can be textual: C++, JAVA, KOTLIN, RUBY, PYTHON, SWIFT, PHP. Or visual (i.e. languages that provide visuals for symbols for transistors,diodes,etc). The language UML is a general purpose modelling language, we will use UML to get to know the most important concepts of object oriented modelling

## OBJECT ORIENTED

Object oriented modelling is a form of modelling that obeys the modelling object oriented paradigm. We will look briefly at the notion of a model and the main concepts of object orientation

## MODELS

Models allows us to describe system efficiently and elegantly. A system is an intergrated whole made up components that are related to one another and influence each other in such a way that they can be perceived as a single, task-based or purposed-based unit. In this regard, they separate themselves from the surrounding environment. Examples of system are material things such as cars, air planes, ecological environment such as lakes and forests but also organisational units such as a university or a company.

It is also a good way to control its complexity and ensure its consistency, the purpose of computer modelling is to be able bring out quality software that meets the criteria.

* Validity: It is the ability of a software product to perform exactly its functions defined by the specification of the user.
* Reliability: It is the ability of a software product to operate regardless of the operating conditions (Component failure, load increase)
* Extensibility: It is the ease of making correction or addition functions to a software.
* Reusable: It is the ability of the software to be reused in row or path in new applications.
* Compatibility: It is the ease with which a software can be combine with other softwares.
* Portability: It is the ability of a software to operate on many platforms.
* Integrity: It is the ability of a software to protect its own data against unauthorised access.
* Easy to learn, use , prepare data, interprete errors and recover from user errors

We model a system to visually and graphically describe the needs and the functional and technical solutions of a software project.

## LIFE CYCLE OF A SOFTWARE

In order to solve the problem that caused the software crisis in the 1970s mainly due to increased cost, maintenance and development difficulties, unreliability, non-respects of the specifications, non-respect of the deadlines. Softwares proposes a set of steps which if respected leads to quality software called the software lifecycle. The life cycle designs all the stages of the software development from its conception to its disappearance.

The stages of the software life cycle are as follows:

1. Preliminary Studies
2. The Elaboration of Specification or Expression of it
3. Analysis
4. Design
5. Coding
6. Test
7. Maintenance

## HISTORY OF UML

From the latest centuries many languages for representing OOP were developed and implemented , to unify these languages, 3 developers James Rum Bourgh, Grady Brooch, IYAR Jacobson decided to merge several existing languages (The BROOCH METHOD, the object modelling technique (OOT) and the object oriented software engineering method (OOSE) ) into a language which is common and standardized called UML

UML is a graphical modelling language based on schema called diagrams designed to provide a standard visualization when designing a software. It was in 1997 that the OMG(Object Management Group which is a consortium and which goal is to promote and standardize object oriented) developed “UML 1.0” due to dissatisfaction, the developers settled up a working group to improve the language on several versions. A major religion was then carried out and we left from UML **1.0** to UML **1.5** which include **9** diagrams. Today, the latest version of the specification validated by the OMG in 2017 is UML **2.5.1** (Which bring radical innovation and greatly extend the scope of UML and which include **14** diagrams).

NB: UML does not recommend any approach so it is not a method. Every one is free to use the diagram type they want in the other they want. It is enough if the diagram produced are consistent with each other before moving on the realization of a software.

## OBJECT APPROACH

UML is an object oriented language i.e. it considers the software as a collection of dissociated and identified objects with characteristics or attributes and method. The particularity of Object Oriented is that it brings data and treatment closer.

This approach is organised around 3 main functions

* It is iterative and incremental: It consist of going back and forth between the initial plan and the modification made by a project stakeholder.
* Driven by a customer and user name: This need to represent the customer requirement that will have to be implemented and that will serve as a common trait through out the development cycle
* Centred on the software architecture: The software architecture describes the strategic choices which largely determine the quality of the software (performance, reliability, adaptability, etc ).This involves asking basic questions like what will be the different software components to use? Which hardware will each of the component be installed on?

Some important notions are specific to the object approach and contribute to the implementation of quality software

1. The class: It is an abstract datatype which specifies the characteristics common to a whole family of objects and which make possible to create objects having these characteristics. In order words, a class declare properties common to a set of objects and it is a moule from which it is possible to create objects . The class animals has as attributes the specie, the sex and as methods shout,move.
2. Encapsulation: It consists of hiding an implementation of an object by defining an interface, this means that the user of a class doesn’t necessarily have to known how the data has being structured. It guarantees the integrity of the data because it makes possible to restrict the direct access to the attribute of the object.It defines the level of visibility(access right on data). We have 3 main level of visibility:

* Public: The function of all classes can access data or method of a class defined with the public visibility level. This is the lowest level of data protection
* Protected: Here access to data is reserved for the function of the inherited classes i.e. by member function of the class as well as the derived classes.
* Private: Here access to data is limited to the methods of the class itself. This is the highest level of data protection.

1. Inheritance: It is the mechanism for transmitting the characteristics of a class (attribute or method) to a subclass. A class can be specialized in order to add specific characteristics or to adapt some of them. Several classes can be generalized into a class in order to group with common characteristics of a set of classes. Thus, specialization and generalization make it possible to build class hierarchy
2. Polymorphism: It represents the ability of a method to apply to objects of different classes.

UML is composed of several parts

1. The views: They are the observable parts of the system. They describes from a given point of view i.e. organisational, temporal, dynamic, architectural, geographical, logical when combining all these views it is possible to define a complete system
2. The Diagrams: They are graphical elements which describes the content of views, they are abstract notion

# THE UP (Unified Process)

The mastery of development processes involves an organisation and follow up of activities. UP is a generic software development method. Generic means that it is necessary to adapt the concept of the project, team, domain and slash or organisation

## CHARACTERISTICS OF THE UP

The unified system is a software development process. It brings together the activities to be carried out. To transform the needs of the user into a software system. The essential characteristics of the unified process are:

* It is component based
* It uses the UML Language (set of tools and diagrams)
* It is driven by the used cases
* It is centred in architecture
* It is iterative and incremental

1. UP is driven by used cases

The main purpose of a software system is to serve its users; It is therefore important to understand the wants and needs of future users. The term user does not only refer to human users but also to other systems. Used cases are not just a tool to specify system requirements. They will completely guide the development process through the use of models based on the use of UML language.

* From the used case model, developers will create a series of design and implementation models that will realize the used case
* Each successive model will then be revised to control the conformity of the used cases.
* Lastly, the testers will test the implementation to assure that the components of the implementation model corresponds to the used cases

1. UP is centred on architecture

From the start of the process, we will have a view of the architecture to be put in place, the architecture of a software system could be described as the different views of the system that needs to be built. The software architecture is equivalent to the most significant static and dynamic aspects of the system. Architecture emerge from business needs as expressed by users and other stakeholders and they are reflected in used cases. It is also influenced by other factors:

* The platform on which the system will have to run
* The reusable building block available for development
* Deployment consideration, existing systems and non functional needs

(performance, reliability, etc)

1. UP is iterative and incremental

Developing a software for commercialization is a huge undertaking that can take several months. It is impossible to develop several things one time. We can cut the work into several parts which are as many small projects. Each of them representing an iteration that results in an increment. An iteration means the succession of steps in the sequence of activities while an increment corresponds to progress in the different stages of development. The choice of what to implement during an iteration is based on 2 factors:

* An iteration takes into account a certain number of used cases which together improve the usability of the product at a certain stage of development.
* The iteration treats in priority the major risks

An increment is often an additive. At each iteration, the developers identify and specify the relevant used cases, create a design guided by the chosen architecture, implement this design in the form of components and verify that this complies with the used cases. As soon as an iteration meets the set objectives, development moves to the next iteration.

## UP LIFE CYCLE

UP LIFE CYCLE

Communication

inception

Software increment

Deployment

Planning

transition

elaboration

construction

Construction

Modeling

Phase 1: inception (Initiation)

1. Define project scope and goals.
2. Identify stakeholders and their roles
3. Develop business case and project plan
4. Create initial use cases and project schedule.

Phase 2: elaboration (analysis)

1. Refine use cases and requirements.
2. Develop architecture and design
3. Create prototype or proof-of-concept
4. Define test plan and quality metrics

Phase 3: construction (implementation)

1. Develop software components
2. Integrate components into working system
3. Conduct unit testing and integration testing
4. Refine and optimize

Phase 4: transition (deployment)

1. Plan deployment and rollout
2. Conduct system testing and acceptance testing
3. Train users and provide support
4. Monitor and evaluate system performance

Types of unified processes

1. UNIFIED PROCESS (UP)

* Developed by Rational Software (now IBM)
* Focus: iterative and incremental development
* Phases: Inception, elaboration, construction, transition.
* Emphasis: use cases, architecture, and iterative development

1. RATIONAL UNIFIED SYSTEM (RUP)

* Extensio n of UP developed by Rational Software (now IBM)
* Focus: structured and disciplined approach
* Phases: inception, elaboration, construction, transition
* Emphasis: use cases, architecture, and iterative development

1. AGILE UNIFIED PROCESS (AUP)

* Developed by Scott Ambler
* Focus: agile and iterative development
* Phases: inception, construction, transition
* Emphasis: iterative development, testing and continuous improvement

1. ENTERPRISE UNIFIED PROCESS (EUP)

* Developed by Scott Ambler
* Focus: enterprise-level software development
* Phases: Inception, elaboration, construction, transition
* Emphasis: scalability, flexibility, and enterprise architecture

1. OPEN UNIFIED PROCESS (OUP)

* Developed by Eclipse Foundation
* Focus: Open-source and collaborative development
* Phases: inception, elaboration, construction, transition
* Emphasis: flexibility, scalability, and community involvement

DIFFERENCES

1. level of formality: RUP is more formal, while AUP and OUP are more flexible
2. iterative development: UP, RUP, and AUP emphasise iterative development, while EUP focuses on scalability.
3. Agile Principles: AUP incorporate agile principles, while RUP and UP are more traditional.
4. Enterprise focus: EUP is designed for enterprise-level development, while UP and RUP are more general-purpose.
5. Open-source: OUP is open-source, while RUP and UP are proprietary.

## EXAMPLE OF A SOFTWARE DEVELOPMENT PROCESS THAT IMPLEMENTS THE UP

* 1. 2TUP (Two Tracks Unified Process)

In the 2TUPprocess, development activities are organised according to 5 work flows:

* Capturing needs
* Analysis
* Design
* Implementation
* Test

The 2TUP process is a frame of the bast development practices. It should be used as a guide to carry out a project and not as the ultimate and universal method of development

1. The 2TUP Development Life Cycle:

Functional

Technical

Preliminary Design

Detail design

Coding and testing

Recipe

Realization

Prototype

Capture of Technical needs

Analysis

Capture of Business needs

The provides a Y-shaped development cycle which separates the technical aspects from the functional aspects. It begins with a preliminary study which essentially consists in identifying the actors which will interact with the system to be built, the messages that the actors and the system exchange then to produce the specifications and to model the context. The process revolves around 3 essential phases:

* Functional branch:

The functional branch capitalizes on the knowledge of the business or the company.

* + - The capture of functional needs which produces a model of needs focused on the business of the user.
    - The analysis , this corresponds to studying the functional specification in order to obtain an idea of what the system will achieve
* Technical branch:

The technical branch capitalizes on technical know how or technical constraints. The techniques developed for the system are independent of the functions to be performed. This right branch include:

* The capture of technical needs which list all the constraints and choices determining the design of the system. The tools and material selected as well as the consideration of integration constraints with the existing system generally condition the technical architecture prerequisites.
* The generic design or prototype which defines the components necessary for the construction of the technical architecture. This design is completely independent of the functional aspects. Its objective is to standardize and reuse the same mechanisms for an entire system. The technical architecture builds the backbone of the computer system and eliminates most of the technical level risks. The importance of its success is such that it is advisable to make a prototype to ensure its validity.
* Realization Branch:

The realization branch consists of bringing the 2 branches together allowing application design to be carried out and finally the delivery of solution adapted to the needs. This middle branch includes:

* The preliminary design which represents the delicate state because it implements the analysis model into the technical architecture so as to draw the map of the components of the system to be developed
* The Detail Design which studies how to make each component
* The Coding and Testing stage which produces the components and progressively test the code units produced
* The Acceptance or Recipe Stage which finally consists in validating the functions of the developed system

# THE VIEWS

A way of implementing UML is to consider the different views that can be grouped together in order to collaborate and define a system.

1. The used case view: It is the description of the model seen by the actors of the system. It corresponds to the needs expected by each actor (package diagram, used case diagram).
2. The logical view: It’s the definition of the system seen internally, it explains how the need of the actors can be satisfied. It describes the system in terms of classes, objects and communication. It identifies the elements of the domain, the relations and the interactions between these elements (class diagrams and object diagrams).
3. The component view: It’s highly the different parts that will make up the future system (source files, libraries, databases, executables, etc). This view includes 2 diagrams a composite structure diagram and a component diagram.
4. The process view: It describes the interaction between process synchronisation and communication of parallel activities. It includes the sequence diagram, the activity diagram, the collaboration or communication diagram, the state machine diagram, the interaction overview diagram and timing diagram.
5. Deployment view: This view describes the physical architecture of each element of a system and the distribution of the past of the software of these elements. It includes the deployment diagram.

# DIAGRAMS

DIAGRAMS

class

Package

Profile

object

Component

Composition Structure

Deployment

activity

State machine

Interaction Diagram

Used case

Structural or Static

Behavioural or Dynamic

Sequence

Communication

Interaction Overview

Timing

## BEHAVIOURAL DIAGRAMS

Behavioural diagrams does not deal with static structures. It rather shows the existing process flow as expected for the use of the program or software. Behavioural diagram describes a system in terms of interaction on 2 plans

* The system seen as a black box: It shows who interact with the system and for what purpose (used case), How does the system interact with its environment( sequence and activity)
* System seen as a white box: It shows how the object of a system interacts (communication diagrams or collaboration diagrams and sequence diagrams). It describes the evolution of the system over time, how the state of the object evolve (state machine diagram).

### USE CASE DIAGRAMS

They describe the high level diagrams and scope of a system. This diagrams also identifies the interaction between the systems and its actors. The use cases and actors in use case diagrams describes what the system does and how the actors use it but not how the system operate internally especially we can employ a used case diagram to answer the following questions

* What are we describing? The system
* Who interacts with the system? The actors
* What can the actors do? The actors are doing used cases
  1. An actor

It is a model element that interacts with a system. It can be an abstract person (e.g. customer) or an external system. The role of actor can be attributed to a service, a society or an API. A person can be represented by several actors depending on the roles they play (e.g. the administrator and director roles can be assigned to the same person). To identify the actors of the system it is necessary to focus on the roles played by the entities out of the scope.

NB: In UML there is no notion of internal or external actors. By definition and actor is out of the scope whether he belongs or not to the company.

There are 2 main category of actors:

* The principal, primary or main actors: These are the people who use the main function of the system
* The secondary actors: These are entities that provide services to the system.

actor

Primary actors

Secondary actors

* 1. The used case

A used case corresponds to the objective of a system motivated by the need of the actors. The set of used cases describes the objectives or goals of a system

tiktok

client

The link between an actor and its used case is called an association

A used case a verb in its infinitve form plus the nominal case.

* 1. Generalization or Specialization:

In a general relation between use cases, the child use case is a specialization of the parent use case. This relationship is useful for showing that one used case is a special type of another and the more specialized used case differs somehow from the original. Thus, all the steps of the original use case must be executed

client

An actor can also participate in generalization ship with other actors, the child actor will then be able to communicate with the same use cases of the parent actor by inheritance.

For example: Here the administrator inherits from the client and hence inheriting all its use cases

client

administrator

* 1. Extension Relationship

It indicates that the source use case adds its behaviour to the destination used case. The used case is optional and comes after the base used case. It is represented by the dash arrow in the direction of the use case with the notation <<extend>>.They can have several extension points

client

<<extend>>

* 1. Inclusion

The used case is part of the base use case. It is represented by a dash arrow in the direction of the use case with the notation include

client

<<included>>

* 1. The System

It shows the formalism in which a used case is related. An actor interacts with the system by using the system as an active actor meaning that the actor initiate the execution of used cases. If the interaction involves the actor being used by the system meaning that the actor is a passive actor providing functionalities for the execution of the used cases. The secondary actor does not necessarily have to be passive

TEXTUAL DESCRIPTION OF USED CASES

The description of a used case is done by scenarios that describes the logical sequence of action that constitutes this used case. These descriptions specifies what the actors does and what the system does. It clarifies the progress of the functionality by describing the chronology of the action which will have to carried out.It has 3 main parts

Part 1: Identification

* Title: It is the name of the used case
* Summary: It is the description of the used case
* Actor
* Date
* Stakeholder: involve with the analysis

Part 2: Description of Scenarios

* Preconditions: It is the state of the system before the used case can be triggered
* Triggers: This is the event that causes the used case to be initiated
* Scenarios: This are several types of scenarios

**Nominal Scenario** which corresponds to a normal development of a used case.

**The Alternative Scenario** which is variant of the nominal scenario. This is the class for steps linked to conditions

**The Exception Scenario** we talk of exception scenario when a step in the process could be disrupt due an abnormal event.

**Post Condition Scenario:** They describe the state of the system after the end of each scenario

Part 3: Nonfunctional Requirements

NB: A textual description is used to clearly narrate the expected system response and outcomes. It shows what the system will use for the development and also the impotence for testing them.

Example: Textual Description of Authentication

Title: Description of Authentication

Summary: We are authenticating to have access to the platform

The actors: all the users

Date: 12 Nov 2024

Stakeholder: Sohlaurie

Precondition: Is to already have an account

Trigger: a click on a button OR perform an action in system(e.g. payment)

Scenarios:

Nominal scenario

1. Click on the authentication button/enter the url
2. Display the login form
3. Enter the credentials and the password
4. Verifies the conformity of the input fields
5. Send query verification (compares if credentials are existent in the DB)
6. DBMS sends the answer or reply
7. Display successful message

Alternative scenario:

3a. At step 3 of the nominal scenario the user can enter wrong credentials

4a. The system can display an error message then return to step 2

6a. If the DBMS verifies and there is no correspondence it sens an error message and brings you back to the 2nd nominal case

Exeception Scenario

3b. If user enters the wrong password 3 TIMES, the system goes to stanby for atleast 3mins

Post Condition of Success: The system displays the workspace

Post Condition of Failure: You cannot have access to the platform

Non functional requirement: The password visibility

On a e-commerce website a user can select products, possibly after reading the data sheet of each product. He can then view his basket. If he decides to validate his basket, he must give his contact details and he can either choose to pay online or pay at the reception. Draw the corresponding use case diagram of this system. www

<<included>>

<<included>>

<<included>>

E-Commerce platform

client

General Use case diagram of E-Commerce Website

client

<<extended>>

<<extended>>

<<included>>

PPayment API

Specific Used case diagram for Payment

Exercise: At UBA, the management process of obtaining a visa card must do it as follows;

A client wishing to obtain a visa card must request it from an account manager

The card is not granted if the applicant is not a customer of the bank

Everyday the credit card office receives request, identify them, processes them, classify them and transmit to the card manager service for verification of the conformity of information sent by the customers

At the end of the month he is responsible for ensuring that the list of applicants are sent to the international bank card center for establishment

As soon as the bank has received all the cards from the center, the card office sends the customers via its manager, a notice of availability and a notice of deduction of annual fee

The customer then comes to select its card and sign and acknowledgement receipt.

If after 2 months the card has not been withdrawn it is cancelled

ICBC

CCO

CMS

Abstract actors

customer

Account Manager

<<included>>

<<included>>

<<extended>>

<<extended>>

<<extended>>

<<extended>>

<<extended>>

<<included>>

<<extended>>

<<included>>

<<included>>

<<included>>

Mr Messio our favourite TEACHER always faces a problem going to canal Olympia. By the way, he lives far from Canal Olympia. He is never coming early and it is difficult to buy a ticket on time. He then ask to his favourite level 2 class to create a web app that will permit user to look the different movies available, the future programs, reserve a ticket of the movie, reserve a seat, reserve popcorn. The condition to reserve popcorn is to buy 2 tickets. The aim is to be able to generate an electronic card ticket. Give the analysis of the system.

ICBC

CCO

CMS

Abstract actors

customer

Account Manager

<<included>>

<<included>>

<<extended>>

<<extended>>

<<extended>>

<<extended>>

<<extended>>

<<included>>

<<extended>>

<<included>>

<<included>>

<<included>>

è

### ACTIVITY DIAGRAM

An activity diagram provides a behaviour view of the system by describing the sequence of action in a process or activity. The activity diagram makes it possible to emphasize the processing and to graphically formalize the sequence of action carried out in the use case.

ELEMENTS OF AN ACTIVIVTY DIAGRAM

1. Action: It is an elementary and instantaneous operation. It is the discrete step from which a behaviour is built.

**Enter password**

NB: It should be noted that there are several types of actions. If an action of a used case corresponds to the called of an internal use case it is represented by an action containing a special site that is 2 circles connected by a line

**Place order**

1. Activity: It defines a behaviour described by an organised series of units whose simple elements are the actions. An activity is a complex operation that can be broken down into actions.
2. Transition: It represents the package from one action to another. It is represented by an arrow.

**Enter password**

**Display form**

1. Decision Node: It represents a decision and always have at least 2 parts branching out with condition texts to allow users to view options.

Information verification

Enter password

Display error message

Display success message

Display interface

Decision

Not OK

OK

NB: ONLY THE SYSTEM IS TAKING THE DECISION

1. Initial Node: It represents the beginning of a process or work flow in an activity diagram

Fig: Representation of the initial node

1. Final Node: It marks the end state of an activity and represent the completion of all flow of the process. There are 2 representation of final node:

Represents the END OF THE ENTIRE activity

WHILE

Represents the end of a specific work flow

1. Partition: They are used to organise the activity nodes arranged in an activity diagram through groupings. They can take form of a table and the activity nodes must belong to a single partition and the transition can pass across partition boundaries.

**Print Instructions**

**Create exam**

**Take Exam**

**Correct exam**

**Publish Result**

Student

Assistant

Teacher

SEQUENCE DIAGRAM

A sequence diagram is an UML diagram that illustrates a sequence of messages between objects in an interaction. It consists of a group of objects that are represented in life line and the message they exchange overtime during the interaction. Interaction is arranged from top to bottom, following their other occurance. The goal of a sequence diagram is to:

* Represent the details of the use case
* Model the logical flow of a complex procedure, function or operation
* Model the high level interaction between acting objects in the system
* Either model generic interaction (showing all possible path through the interaction) or specific instances of an interaction (Showing just one path through the interaction)

1. Components of a Sequence Diagram

Lifeline: It is a dotted vertical line that shows the sequential event affecting an object during the process.

Lifeline

Objects:

: Object

Actor: he communicates with objects in a system

Activation: It represents the time required for an object or an actor to complete a task, it indicates when the object perform an action. The longer the task requires the longer the activation bulbs is

Message: A message is model by a horizontal arrow between activation. It indicates the communication between objects. There are several types of messages in a sequence diagram;

* The Synchronous message: The reception of a synchronous message must cause the recipient to launch one of its operation or methods. The sender of the message remains blocked during the entire execution of the method and therefore wait for the end of the method before being able to launch a new message
* The Asynchronous message: They do not require a response from the recipient before the sender continues. It is represented by an empty head arrow.

* The reponse: It represent a response to the message

1. Structure of a Sequence Diagram

They combine interaction fragments: They represent an articulation of operations. It is defined by an operand or operator. The operator conditions the meaning of the fragment. There are 12 operators in UML 2.0 notation. Combine fragments make it possible to define sequence diagram in a compact way.A combine fragment is represented by a rectangle whose upper left corner contains a pentagon. In the pentagon, there is a type of combination called the interaction operator. There are different types of operators for combine fragments.

|  |  |  |
| --- | --- | --- |
|  | Operator | Purpose |
| Branches and loop | Alt | Alternative interaction |
| Opt | Optional interaction |
| Loop | Iterative interaction |
| Break | Exception interaction |
| Concurrency and order | Seq | Weak order |
| Strict | Strict order |
| Par | Concurrent interaction |
| Critical | Atomic interaction |

Branches of loops

1. An alternative interaction operator:

It has atleast 2 operands. Each operand represent an alternative path in the execution which corresponds approximately to multiple cases in programming languages. E.g. the switch statement in JAVA.

1. Choice and loop operator:

Alternative options break and loop. The alt operator is an alternative it represents 2 possible behaviours; It is an equivalent of the if then and if not. Therefore only one of the 2 branches will be performed in the given scenario. The execution condition of one of the 2 branches(can be explicit or implicit). The use of them allows to indicate that the branch is executed if the condition

The opt operator: The opt operator denotes an optional combined fragment. It represents a behaviour that may or many not happen. AN optional fragment is equivqlent to alt fragment

Example of an alt and opt channel.

When a student wants to regixter for the exam the following case might occur:

* There are still places available and the student can register.
* There is a place available on the waiting list then student has to decide whether to go on the waiting list.
* If there is no place available for the exam or on the waiting list, the student receive an error message and he cannot register to the course

Student

: Student Admin System

: DBMS

Register (MatNo, exam)

Enter(MatNo, exam)

Status == enter.Status

alt

(Status ==OK)

(Status ==Waiting list free)

Register “WI”

(Status on WL==true)

Register (MatNo, exam)

Enter WL(MatNo, exam)

Register “OK”

(else)

Register= “error”

EnterWL. “OK”

Register “OK”

Bifurcation Node:

It is a control node that a separate a stream on several competing system. It therefore has an incoming arc and several outgoing arcs. It is represented by a thick line. If an execution path splits into multiple simultaneously active execution paths later on. You can realize this using a parallelization node. A parallelization node is a pit with a black bar with one incoming edge and multiple outgoing edges.

You can merge concurrent sub-path using a synchronisation node. This node is the counterpath to the parallelization node

Parallelization node

Synchronisation node

Participate in assignment

Attend lecture

Write Exam

Register

**Request login form**

Display error message

Execution query

Send query result

Process result

Send authenticate query

Display user dashboard

Display error message

Fills form and submit

Provide login form

Check Validation

Check form conformity

Activity Diagram of authentication

actor

2.3: Send authentication query ( )

2.1: Fill and Submit form ( )

1:Request form ( )

: Student Admin System

: DBMS

alt

1.1: Display form ( )

2.1: Check form ( )

2.2: Send error message ( )

2.5: Display login failed message ( )

3: Display dashboard( )

2.3.2 Send result

[Conformity not ok]

[Conformity OK]

2.4 Check Validity

(Valid)

(InValid)

2.3.1 Execute query

alt

Sequence Diagram of Authentication

COMMUNICATION DIAGRAMS

Similar to sequence diagrams, communication diagrams models message forwarding using life lines by numbering the sequences. The number represents the order in which messagesare sent. On communication diagrams objects are presented with connectors association between them.

* + - 1. The life line: They are represented by rectangle containing the name and type of the proceeding.
      2. Connectors: The relationship between life lines are called connectors
      3. Message: refer to a particular communication between life line and are usually ordered in ascending order of re

1. Login

2. getcourse

: Student

: E-learning system

:DBMS

Example of communication diagram

The communication diagram shows directly who communicates with whom. The relationships are the result of the exchange of messages.

1: Request form()

2: Fill and submit form()

: DBMS

: E-learning system

2.3.1: Invalid user()

2.3.2: Valid User()

2.3.1: Check user availability()

2.3.2.1: Display login failed message()

2.3.3.1: Display dashboard()

2.1: Check form()

2.3: transfer form values

1.1: display form()

2.2 send error message()

actor

COMMUNICATION DIAGRAM OF AUTHENTICATION

STATE MACHINE DIAGRAM

UML State Machine diagram describes the internal behaviour of an object. They present the possible sequences of state and actions that a class instance can process during its life time. A state represents a period during the life of an object during which it waits for an event or perform an activity. A transition status diagram basically makes it possible to represent and describe the state changes of an object related to an event in the system. To build a state machine diagram the following questions can be asked.

What happens when an object is in a specific state?

Which event must occur for its behaviour to be modified?

What are the triggering events?

What properties must the object have to be able to change its state?

1. State: An object can go through a series of state during its life time. A state represent a period in the life of an object during which it waits for an event or performs an activity by satisfying a number of condition.There are several states:

* Simple state: This is an elementary state that does not contain an internal transition or sub-state

Simple state

* Composite state: It enclose different state

Composite state

1. Initial node: It defines the initial state or beginning of a system
2. Final node: It represents the final state and end of a system
3. Transition: It represents the transition from one state to another

Event [condition]

Target state

Source state

Event [condition]

Fig: Representation of transition

Example of state machine diagram

More than 60 years

Working

Fired

Retired

Hired

More than 60 years

Unemployed

1. Decision node: You can use it to model alternative transitions

Decision

OK

Not OK

1. The parallelization node:

1. Synchronisation node:
2. Teminate node: If a terminate node is reached in a flow the state machine terminates and the model object ends to exists

registered

Not graded

Positive

Negative

Decision

[grade >5]

[grade <=5]

performance

withdraw

graded

Correction [new grade >5]

Correction [new grade <=5]

grade

Assignment: Draw the State machine diagram of

* + - 1. Water
      2. Student in AICS

NB: The state machine diagram ……………….

TIMING DIAGRAM

It explicitly shows the state changes of the interaction partners that can occur due to time events or as a result of the exchange of messages. That is it mostly represent the state machine and the sequence diagram.

INTERACTION OVERVIEW DIAGRAM

It models the connection between several interaction processes by setting individual interaction diagrams (i.e sequence , communication, timing and other interaction overview diagram.) in a time based and coxial system. It also specifies conditions under which interaction process are . It is the type of activity diagram that focuses on modelling the flow or control between multiple interactions. It provides a high level vue of how different interactions are related and how the work together in a system

They describe the structure of the system in terms of system components (objects,class, packages, components ) and the relation between these components (Specialization, association, dependencies, etc)

Class diagram: The class diagram expresses the static structure of the system in terms of classes and the relationship between those classes. The purpose of the class diagram is to model the feature of the information system. It is used to represent all the information that is managed by the domain by describing what the attributes and behaviour which it has rather detailing methods to achieve operations. This information is structured i.e. it has been grouped into classes. The diagram highlights possible relationships between these classes. The class diagram has 6 concepts: class, attributes, relations, operations, identifier, generalisation/specialisation.

1. Class: A class is an abstract description of a set of objects of an application domain. It defines the structure the behaviour and the relationship of these objects.

|  |
| --- |
| Class name |
| Attribute1: type  Attribute2: type |
| Operation1()  Operation2() |

1. Attributes: An attribute is an elementary piece of information represented by its name and its format that defines a class.

UML defines 3 levels of visibility for attributes;

* Public (+): It means that the element of the class is visible to all client of the class
* Private (-): The element is only visible to object of the class in which it is declared.
* Protected (#): The element is visible to sub-classes of the class

|  |
| --- |
| Class name |
| + public Attribute1 : type  # protected Attribute2 : type  - private Attribute3 :type |
| Operation1()  Operation2() |

1. Identifier: An identifier is a particular attribute which makes it possible to uniquely identify each object or instance of a class.
2. Operation: An operation represents an element of object behaviour defined in a class. An operation is functionally provided by a class. The description of the operation can specify the input and the output parameters as well as the elementary actions to be performed.
3. Relation: These are several types of relationships between classes; association, generalization/specialization and dependency.
4. Association: It’s a relationship between 2 or more classes. An association thus indicates that there can be a link between instances of associated instances. An association is a relationship between 2 classes(binary association) or more(N-ary association) which describes the structural connection between their instances.

* Binary association: It is represented by a solid line between classes associated with. It can be adorned with a name possibly with a clarification of the reading direction. When the 2 end of the association points towards the same class, the association is said to be reflexive.
* N-ary association: It links more than 2 classes. It is represented by a large diamond with a path leading to each participating classes. If the name of the association exists it is written either inside the diamond or beside it. In UML, the number of instances of the association for an instance is called multiplicity. Always given as an interval in the form minimum … maximum. They specify the number of objects that may be associated with exactly one object of the opposite sign. The values that the maximum or minimum may adopt are natural numbers and asterix. If minimum and maximum are identical, one value and the dot can be omitted

|  |  |
| --- | --- |
| Multiplicity | Definition |
| 1 | One and only one |
| 0..1 | Zero or one |
| N or \* | Natural numbers |
| 0…\* | Zero to many |
| 1…\* | One to many |

|  |
| --- |
| Lecturer |
|  |
|  |

|  |
| --- |
| Assignments |
|  |
|  |

issues

1

1

|  |
| --- |
| Lecturer |
|  |
|  |

|  |
| --- |
| Assignments |
|  |
|  |

1..\*

gives

1..\*

|  |
| --- |
| Person |
|  |
|  |

examiner

examines

examinees

The one to one multiplicity

|  |
| --- |
| Car |
|  |

|  |
| --- |
| Driver |
|  |

1

1

drives

The one to many multiplicity

|  |
| --- |
| Newspaper |
|  |

|  |
| --- |
| Article |
|  |

contain

0..\*

1..\*

The many to many multiplicity

1..\*

compose

1..\*

|  |
| --- |
| Musicians |
|  |

|  |
| --- |
| Group |
|  |

|  |
| --- |
| Class |
|  |

|  |
| --- |
| Teacher |
|  |

|  |
| --- |
| Student |
|  |

|  |
| --- |
| Course |
| Dav  Short\_time  Duration |

\*

\*

\*

It is a class formed between 2 or more classes. Its identifiers functionally depend on the identifiers of the classes from which it is formed but sometimes an attribute can depend functionally on 2 identifiers belonging to 2 different classes. An association class can also have association between other classes.

Exam meeting

Exam

Student

\*

\*

1. Aggregation: In class diagram we often use associations that expresses relationships of containers/content or set/elements. This relationship is represented by a small empty diamond on the side of the aggregate. UML differentiates between 2 types: shared aggregation and strong aggregation/composition. Both are represented by a diamond at the association end of the class that stands for whole. The differentiation between composition and shared aggregation is indicated by a solid diamond for composition and a hollow diamond for the shared aggregation. Both are transitive and asymmetric association.
2. Transitivity
3. Asymmetry expresses that it is not possible for A to be part of B and B to be part of A
4. Shared Aggregation: A shared aggregation expresses a weak belonging of the part of the whole meaning that parts also exists independent of the whole. The multiplicity of the aggregate end may be greater than one meaning that an element can be part of multiple other elements simultaneously.

NB: An aggregated element can be linked to other classes.

* Deleting the set does not delete the element.
* An aggregate can be multiple.
* Aggregation implies a relationship where the element can exist regardless of the aggregate.
* Aggregation is considered a week association

|  |
| --- |
| Type |
|  |
|  |

|  |
| --- |
| Car |
|  |
|  |

1..1

0..\*

Fig: Shared Aggregation

1. Composition Aggregation. It is special case of aggregation in which the life of the component is linked to that of the aggregate. It oftens refers to a physical capacity. In the composition association the aggregate can’t be multiple. In addition to the aggregation, the composition implies a coincidence of the life time of the component. The destruction of the aggregate (container), automatically implies the destruction of all the linked components. The composition is represented by a small black diamond beside the aggregate

|  |
| --- |
| Building |

|  |
| --- |
| Beamer |

|  |
| --- |
| Lecture Hall |

1

0..1

1

1

Fig: Composition Aggregation

Example: A University can have zero or many departments and a department belongs to one university. A department can have zero or many professors while a professor can belong to zero or many department. The composition diamond means that if we eliminate university the department will cease to exist whereas the hollow diamond shows that if we remove department professor entity will still exists

|  |
| --- |
| University  1  0..\*  0..\*  0..\* |
|  |
|  |

|  |
| --- |
| Professor |
|  |
|  |

|  |
| --- |
| Department |
|  |
|  |

Shared

Composition

1. Generalization: The principle of generalization makes it to identify among the objects of a class, subset of objects with specific definition. The more specific class (daughter class) is more consistent than the main class (mother class) i.e. it inherits all attributes, members, relationships from the main class and can have ither relationships. The generalization relationship is indicated by an arrow pointing the parent class

|  |
| --- |
| Person |
| - Idpers:int[1]  - Fname: string[1]  - Sname: string[1]  - Email: string[1] |
|  |

|  |
| --- |
| Client |
| enterprisename:string |
|  |

|  |
| --- |
| Employee |
| - Salary:double[1] |

OBJECT DIAGRAMS

The Object diagram represents the specific instance of a class at a specific moment. It allows you to express the class diagram through examples in highlighting the imperceptible details in the class diagram. In other words, the class diagram models the rules and the object diagram models the fact. One object diagram is materialised as a class diagram with some little differences

|  |
| --- |
| Account |
| - Id:int  - Currency: device  - Balance: float |
| Deposit(amount:float)  Withdraw(amount:float)  Balance():float |

|  |
| --- |
| Current:Account |
| Id=13  Currency=XAF  Balance=100,000.000 |
| Deposit(amount:float)  Withdraw(amount:float)  Balance():float |

|  |
| --- |
| Saving:Account |
| Id=13  Currency=XAF  Balance=100,000.000 |
| Deposit(amount:float)  Withdraw(amount:float)  Balance():float |

Package diagram:

The package is on the tab or inside the rectangle.The dotted arrows are dependencies, one package depends on another if changes in the order can force changes in the format. Depeding on where a package is used, packages can participate in different types on relationships. For example in a class diagram packages represents groupings of classes. Therefore aggregation and association relationships are possible.

1. Package: It is used to group together logically related elements within a system. Each element contained within the package should be packable elements and have a unique name
2. Dependencies: It is a visual representation of how an element or a set of elements can influence. Dependencies are divided into 2 groups: The access and the import dependencies

Access Dependency: It indicates that the packet

access

Import Dependency: It indicates that the dependency has been jiqpd

import

1. Component
2. Dependecies/ It is used to show the relationship between 2 components. It is represented by a dotted arrow pointing from a given component to the component on wgich is depends