

UML - Unified Modeling Language

Ali ZAINOUL

for Evogue
December 4, 2022



Table des matières

- 1 Introduction et annotations
- 2 Utilité du langage UML et domaines d'utilisation
- 3 Les diagrammes en UML
- 4 Modèle des cas d'utilisation
- 5 Principes fondamentaux de la OOP - rappels

Introduction

- **L'analyse fonctionnelle:** approche structurante au service de l'équipe en charge de la création ou de l'amélioration d'un produit. L'objectif est d'identifier le besoin client, afin de déterminer les fonctions du produit, avant de rechercher les solutions techniques et technologiques à mettre en œuvre.
- l'UML est destiné à faciliter la conception des documents nécessaires au développement d'un logiciel orienté objet, comme standard de modélisation de l'architecture logicielle.

Annotations

- MOA (Maître d'Ouvrage/Chef de projet): celui qui décide ce qui doit être fait.
- MOE (Maître d'œuvre): celui qui décide comment le faire.
- AMOA (Assistant à Maîtrise d'Ouvrage): celui qui aide à le faire.
- CdC (Cahier de Charges): document formalisant un besoin, en détaillant les fonctionnalités attendues d'un système, d'un produit ou d'un service ainsi que les contraintes auxquelles il est soumis.

UML et domaines d'utilisation

- Développement/ programmation
- Génie logiciel
- Modélisation du comportement d'une BDD

Les diagrammes en UML

■ On note trois types de diagrammes UML:

- Diagrammes de structure ou diagrammes statiques,
- Diagrammes de comportement,
- Diagrammes d'interaction ou diagrammes dynamiques.

Les diagrammes en UML

■ Les Diagrammes de structure ou diagrammes statiques:

- **Diagramme de classes** (class diagram) : représentation des classes qui interviennent dans le système.
- **Diagramme d'objets** (object diagram) : représentation des instances de classes (objets) utilisées dans le système
- **Diagramme de composants** (component diagram) : représentation des composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)
- **Diagramme de déploiement** (deployment diagram) : représentation des éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.

Les diagrammes en UML

■ Les Diagrammes de structure ou diagrammes statiques:

- **Diagramme des paquets** (package diagram) : représentation des dépendances entre les paquets (un paquet étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML), c'est-à-dire entre les ensembles de définitions.
- **Diagramme de structure composite** (composite structure diagram) : représentation sous forme de boîte blanche des relations entre composants d'une classe (depuis UML 2.x).
- **Diagramme de profils** (profile diagram) : spécialisation et personnalisation pour un domaine particulier d'un meta-modèle de référence d'UML (depuis UML 2.2).

Les diagrammes en UML

■ Les Diagrammes de comportement:

- **Diagramme des cas d'utilisation** (use-case diagram) : représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire de toutes les fonctionnalités que doit fournir le système.
- **Diagramme états-transitions** (state machine diagram) : représentation sous forme de machine à états finis du comportement du système ou de ses composants.
- **Diagramme d'activité** (activity diagram) : représentation sous forme de flux ou d'enchaînement d'activités du comportement du système ou de ses composants.

Les diagrammes en UML

■ Les Diagrammes d'interaction ou diagrammes dynamiques:

- **Diagramme de séquence** (sequence diagram) : représentation de façon séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- **Diagramme de communication** (communication diagram) : représentation de façon simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets (depuis UML 2.x).
- **Diagramme global d'interaction** (interaction overview diagram) : représentation des enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité) (depuis UML 2.x).
- **Diagramme de temps** (timing diagram) : représentation des variations d'une donnée au cours du temps (depuis UML 2.3).

Modèle des cas d'utilisation

- En langage UML, les diagrammes de cas d'utilisation modélisent le comportement d'un système et permettent de capturer les exigences du système. Les diagrammes de cas d'utilisation décrivent les fonctions générales et la portée d'un système.

Principes fondamentaux de la OOP - rappels

- La Programmation Orientée Objet (POO) est un des principes fondamentaux en programmation informatique, ses origines remontent aux années 1970 avec les langages Simula et Smalltalk, mais le principe a rapidement pris son envol grâce à la création du langage C++ qui est l'extension du langage C, avec en effet, cette quête de rendre les logiciels plus robustes.

Principes fondamentaux de la OOP - rappels

- Une mnémotechnique utile regroupant les six concepts fondamentaux de la POO est: **ACOPIE**
 - Abstraction
 - Class
 - Object
 - Polymorphism
 - Inheritance
 - Encapsulation

Object

- En informatique, un objet est un conteneur (container) symbolique et autonome contenant des informations et des fonctions/méthodes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel.

Class

- La classe est une structure informatique particulière dans le langage objet.
- Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type.
- Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe. C'est pourquoi les attributs d'un objet sont aussi appelés variables d'instance et les messages opérations d'instance ou encore méthodes d'instance.

Class

- L'interface de la classe (l'ensemble des opérations visibles) forme les types des objets.
- Selon le langage de programmation, une classe est soit considérée comme une structure particulière du langage, soit elle-même comme un objet (objet non-terminal). Dans le premier cas, la classe est définie dans le runtime ; dans l'autre, la classe a besoin elle aussi d'être créée et définie par une classe : ce sont les méta-classes. L'introspection des objets (ou « méta-programmation ») est définie dans ces méta-classes.
- La classe peut être décrite par des attributs et des messages. Ces derniers sont alors appelés, par opposition aux attributs et messages d'un objet, variables de classe et opérations de classe ou méthodes de classe. Parmi les langages à classes on retrouve Smalltalk, C++, C#, Java, etc.

Encapsulation

- L'encapsulation est le fait de regrouper les données et les méthodes qui les manipulent en une seule entité appelée *capsule*.
- Elle permet d'avoir un code organisé, restreindre l'accès à certaines portions depuis l'extérieur de la capsule et avoir un code robuste.

Abstraction

- L'abstraction est l'un des concepts clés dans les langages de programmation orientée objet (POO). Son objectif principal est de gérer la complexité (comprendre difficulté ici) en masquant les détails inutiles à l'utilisateur
- L'abstraction est le processus consistant à représenter un objet dans la vie réelle en tant que modèle informatique.
- Cela consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l'on souhaite manipuler, et à les placer dans un modèle informatique convenable.

Inheritance

- C'est un mécanisme pour transmettre toutes les méthodes d'une classe dite "mère" vers une autre dite "fille" et ainsi de suite.

Polymorphism

- Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

Les bases de l'UML

On va illustrer ici les concepts les plus utilisés en UML:

- **Notion Orientation Objet:** principes d'analyse et de conception d'applications orientées objet: notions de programmation orientée objet (POO) et ses six principes.
- **Notion de Structure:** notion d'acteur, d'artéfact, attribut, classe, composant, interface, objet, package et propriété.
- **Notion de comportement:** activité, évènement, message, méthode, état et cas d'utilisation.
- **Notion de relation:** notion d'agrégation, d'association, de composition, de dépendance, de généralisation et d'héritage.
- **Notion de diagrammes:** cf sections précédentes.

Les bases de l'UML

- **Notion Orientation Objet:** déjà vus dans les sections précédentes.
- **Notion de Structure:** notion d'acteur, d'artéfact, attribut, classe, composant, interface, objet, package et propriété.
 - **Notion d'acteur:** En UML, un acteur est défini en tant qu'entité jouant le rôle d'un utilisateur ou par un système interagissant avec le système modélisé. Les acteurs apparaissent souvent dans les diagrammes de cas d'utilisation.



Figure: User

Les bases de l'UML

■ Suite:

- **Notion d'artéfact:** Un artefact est un élément concret du monde réel (un document, un exécutable/script/fichier, une table de BDD etc.). C'est un classeur représenté par un rectangle contenant le mot-clef « artifact » suivi du nom de l'artefact. Ceci ne nous concerne pas dans le cadre de ce cours.
- **Les notions d'attribut (= membre), classe, interface, objet et package ont été vus précédemment.**
- **Notion de composant:** un composant décrit l'organisation du système du point de vue des éléments logiciels: modules, données ou encore composants de configuration.

Concepts objets, et diagrammes de classes - lien avec la POO

- On a vu qu'un diagramme de classes a généralement trois compartiments:
 - Le premier concerne le nom de la classe
 - Le second concerne les membres/attributs de la classe.
 - Tandis que le troisième, concerne les méthodes/opérations de la classe.

Concepts objets, et diagrammes de classes - lien avec la POO

- La figure ci-dessous explicite la signature d'un diagramme de classes d'un point de vue global:

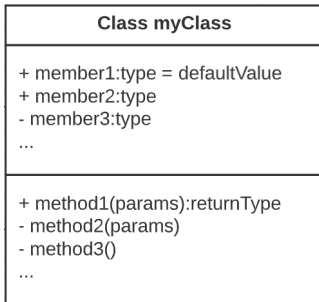


Figure: classDiag

Concepts objets, et diagrammes de classes - lien avec la POO

- La figure ci-dessous explicite les diverses notions de relation existantes:







Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Figure: relationships

Diagramme d'activité

- Les diagrammes d'activités se focalisent sur les traitements de données. Ils modélisent le cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter visuellement le comportement d'une méthode ou le déroulement d'un cas d'utilisation précis.
- Les diagrammes d'activités explicitent le flot de contrôle d'une activité à l'autre. Ils sont particulièrement adaptés à la description des cas d'utilisation.

Diagramme d'activité

- Une **action** est le plus petit traitement qui puisse être exprimé en UML. Voici une liste non exhaustive des actions les plus utilisées:
 - **Action appeler (call operation):** L'action appeler correspond à l'invocation d'une méthode sur un objet de manière synchrone ou asynchrone.
 - **Action comportement (call behavior):** L'action call behavior est une variante de l'action call operation car elle invoque directement une activité (ensemble de méthodes et de traitements) plutôt qu'une opération.
 - **Action envoyer (send):** Cette action crée un message et le transmet à un objet cible. (envoi d'email, de signal..)
 - **Action accepter événement(accept event)** Utilisée souvent dans le cadre d'appels asynchrones. (attente d'une réponse serveur, threads, mutex, appels asynchrones)

Diagramme d'activité

■ Suite:

- **Action accepter appel (accept call):** Variante de l'action accept event pour les appels synchrones.
- **Action répondre (reply):** Cela permet de transmettre un message en réponse à la réception d'une action de type accept call.
- **Action créer (create):** Cette action permet d'instancier un objet (i.e: instantiation d'un objet).
- **Action détruire (destroy):** Cette action permet de détruire un objet. On retrouve souvent cela dans le cadre de la POO en C++ (notion de destructeurs) ou en BDD (notion de Delete)
- **Action lever exception (raise exception):** Cette action permet de lever explicitement une exception. (Une division par zéro à titre d'exemple).

Diagramme de séquences et de temps

- Un diagramme de séquences est la représentation visuelle d'interactions entre divers acteurs et le système selon un ordre chronologique dans la formulation UML.
- En UML, un diagramme de temps est une forme particulière de diagramme de séquences où les axes ont été inversés.