Ecole Supérieur des
Technologies de l'Information

Classe : **L3 IDEV**

Année académique : **2019-2020**

Mois : **Juillet**

COURS UML ANALYSE ET CONCEPTION

PRESENTATION GENERALE DE UML ET DU CONCEPT ORIENTE OBJET

I- Historiques d'UML

Voici quelques mots sur l'historique d'UML, pour vos recherches personnelles, je vous invite à enrichir vos connaissances sur les origines d'UML.

UML est né de la fusion des trois méthodes qui ont influencé la modélisation objet au milieu des années 90 : OMT, Booch et OOSE. Il s'agit d'un compromis qui a été trouvé par une équipe d'experts : Grady Booch, James Rumbaugh et Ivar Jacobson. UML est à présent un standard défini par l'Object Management Group (OMG). De très nombreuses entreprises de renom ont adopté UML et participent encore aujourd'hui à son développement.

UML est surtout utilisé lorsqu'on prévoit de développer des applications avec une démarche objet (développement en Java, en C++, etc.).

On est d'avis que l'on peut tout à fait s'en servir pour décrire de futures applications, sans pour autant déjà être fixé sur le type de développement.

II- Définitions

UML, c'est l'acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un langage visuel constitué d'un ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour représenter le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par le logiciel, etc.
Réaliser ces diagrammes revient donc à modéliser les besoins du logiciel à développer.

Comment faire pour réaliser ces diagrammes ?

Vous pouvez soit reprendre les normes de ces diagrammes que nous verrons au fur et à mesure de ce cours et les dessiner à la main, soit utiliser des logiciels gratuits ou payants pour les réaliser à savoir StarUml, ArgoUml, BoUml, PowerDesigner, etc...

III- La modélisation avec UML

Pourquoi Modéliser ?

Modéliser, c'est décrire de manière visuelle et graphique les besoins et, les solutions fonctionnelles et techniques de votre projet logiciel.

Dans la pratique, l'utilisation de schémas et d'illustrations rend quelque chose de complexe plus compréhensible.

Disons que c'est intéressant mais quel est le lien avec UML alors ?
C'est exactement ce à quoi UML sert dans des projets de réalisation de logiciels.

Un document de texte décrivant de façon précise ce qui doit être réalisé contiendrait plusieurs dizaines de pages. En général, peu de personnes ont envie de lire ce genre de document. De plus, un long texte de plusieurs pages est source d'interprétations et d'incompréhension.

UML nous aide à faire cette description de façon graphique et devient alors un excellent moyen pour « ***visualiser*** » **le(s) futur(s) logiciel(s).**

Un logiciel qui a été réalisé sans analyse et sans conception (étapes où l'on modélise le futur logiciel) risque lui aussi de ne pas répondre aux besoins, de comporter des anomalies et d'être très difficile à maintenir.

IV- **Phase d'Analyse et Conception**

Comme n'importe quel type de projet, un projet informatique nécessite une phase d'analyse, suivi d'une étape de conception. Dans la phase d'analyse, on cherche d'abord à bien comprendre et à décrire de façon précise les besoins des utilisateurs ou des clients.

- *Certaines questions doivent être posé davantage :*

Que souhaitent-ils faire avec le logiciel ?
Quelles fonctionnalités veulent-ils ?
Pour quel usage ?
Comment l'action devrait-elle fonctionner ?

C'est ce qu'on appelle « ***l'analyse des besoins*** ».

Après validation de notre compréhension du besoin, nous imaginons la solution.
C'est la partie **analyse de la solution**.

Dans la **phase de conception**, on apporte plus de détails à la solution et on cherche à clarifier des aspects techniques, tels que l'installation des différentes parties logicielles à installer sur du matériel.

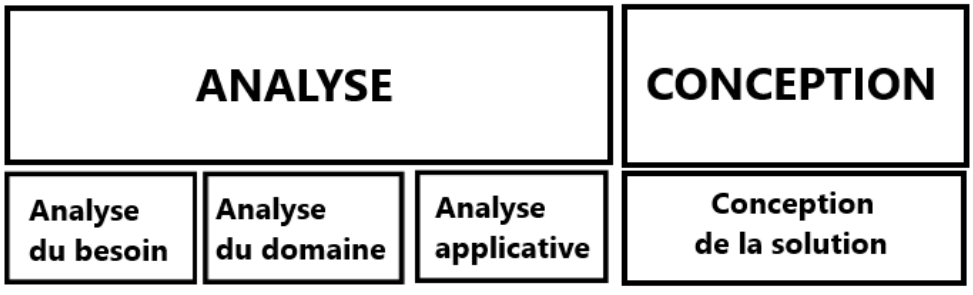

Figure 1 — Phase d'analyse et conception

Pour réaliser ces deux phases dans un projet informatique, nous utilisons des méthodes, des conventions et des notations. UML fait partie des notations les plus utilisées aujourd'hui.

Nous allons voir l'utilité du langage UML et ses outils principaux : ***les diagrammes.***

Nous verrons comment ce langage peut contribuer à la phase d'analyse des besoins et du domaine d'un projet informatique.

V- **Règles générales -UML**

Afin d'assurer un bon niveau de cohérence et d'homogénéité sur l'ensemble des modèles, UML propose d'une part un certain nombre de règles d'écriture ou de représentations graphiques normalisées et d'autre part des mécanismes ou des concepts communs applicables à l'ensemble des diagrammes. Certains éléments, comme les stéréotypes, sont spécifiquement prévus pour assurer une réelle capacité d'adaptation et d'évolution de la notation notamment pour prendre en compte les particularités des différentes situations à modéliser. Les principaux éléments généraux d'UML que nous présentons sont : le stéréotype, la valeur marquée, la note, la contrainte, et la relation de dépendance.

En outre UML propose un méta-modèle de tous les concepts et notations associées utilisés dans les treize diagrammes du langage de modélisation.

Méta-modèle

Le langage de modélisation UML respecte un certain nombre de règles sur les concepts manipulés (classes, attributs, opérations, paquetages…) ainsi que sur la syntaxe d'écriture et le formalisme de représentation graphique. L'ensemble de ces règles constitue en soi un langage de modélisation qui a fait l'objet d'un méta-modèle UML.

L'intérêt de disposer d'un méta-modèle UML permet de bien maîtriser la structure d'UML et de faciliter son évolution.

Le méta-modèle d'UML est complètement décrit dans la norme.

Stéréotype

Un stéréotype constitue un moyen de classer les éléments de la modélisation. Un certain nombre de stéréotypes sont déjà définis dans UML, mais d'autres valeurs de stéréotypes peuvent être ajoutées si cela est nécessaire soit à l'évolution générale d'UML, soit à la prise en compte de situations particulières propres aux entreprises.

Les stéréotypes peuvent s'appliquer à n'importe quel concept d'UML. Nous nous intéresserons dans ce cours à un certain nombre d'entre eux que nous présenterons au niveau des diagrammes lorsque leur utilisation nous paraîtra pertinente.

En particulier, dans le diagramme de classe, le stéréotype permet de considérer de nouveaux types de classe. Cette possibilité d'extension pour les classes, se définit donc au niveau méta-classe.

Formalisme et exemple

Le nom du stéréotype est indiqué entre guillemets. Un acteur peut être vu comme un stéréotype particulier d'une classe appelée acteur. L'exemple (fig. 2) montre une classe Client stéréotypée comme « acteur ».

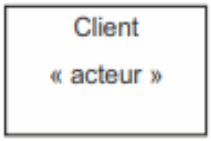

Figure 2 — Exemple d'une classe stéréotypée

Valeur marquée :

UML permet d'indiquer des valeurs particulières au niveau des éléments de modélisation et en particulier pour les attributs de classe. Une valeur marquée se définit au niveau méta-attribut. Formalisme et exemple La valeur marquée est mise entre accolades avec indication du nom et de la valeur : {persistance : string} si l'on veut ajouter ce type d'attribut dans une classe.

Profil :

Afin de donner la possibilité de spécialiser chaque application d'UML à son propre contexte, UML propose de définir un profil d'utilisation caractérisé principalement par la liste des stéréotypes, la liste des valeurs marquées et les contraintes spécifiées pour un projet donné.

Note :

Une note correspond à un commentaire explicatif d'un élément d'UML. Formalisme et exemple La figure 3 montre le formalisme et un exemple de la représentation d'une note.

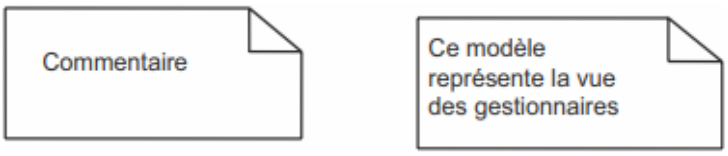

Figure 3 — Formalisme et exemple d'utilisation d'une **note**

Contrainte :

Une contrainte est une note ayant une valeur sémantique particulière pour un élément de la modélisation. Une contrainte s'écrit entre accolades {}. Dans le cas où la contrainte concerne deux classes ou plus, celle-ci s'inscrit à l'intérieur d'une note.

Formalisme et exemple

• Première forme d'écriture d'une contrainte : {ceci est une contrainte}.

• Deuxième forme d'écriture : à l'intérieur d'une note (fig. 4).

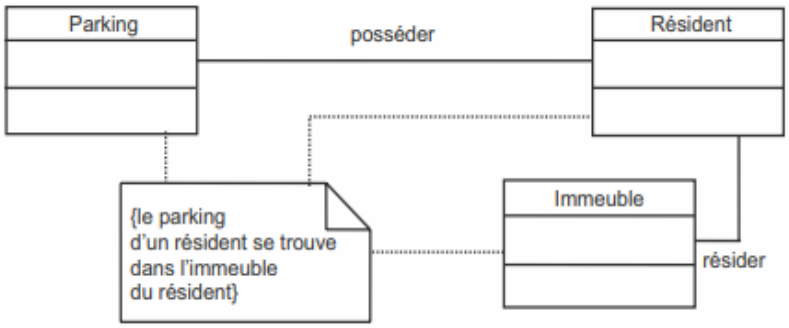

Figure 4 — Exemple d'utilisation d'une **contrainte** (sans représentation des multiplicités)

VI- Concepts de l'approche objet

Dans la gestion de projet, nous pouvons citer deux approches permettant de définir les besoins :

La décomposition fonctionnelle (ou l'approche procédurale)

L'approche objet (sur laquelle est basée UML)

La décomposition fonctionnelle :

Avant l'apparition de l'approche objet dans les années 80, une autre démarche était largement utilisée.

Pendant longtemps, de nombreux logiciels étaient conçus par l'approche fonctionnelle descendante, appelée également la décomposition fonctionnelle.

L'approche par décomposition fonctionnelle considère que le logiciel est composé d'une hiérarchie de fonctions et de données.

Les fonctions fournissent les services désirés et les données représentent les informations manipulées. La démarche est logique, cohérente et intuitive (fig. 5).

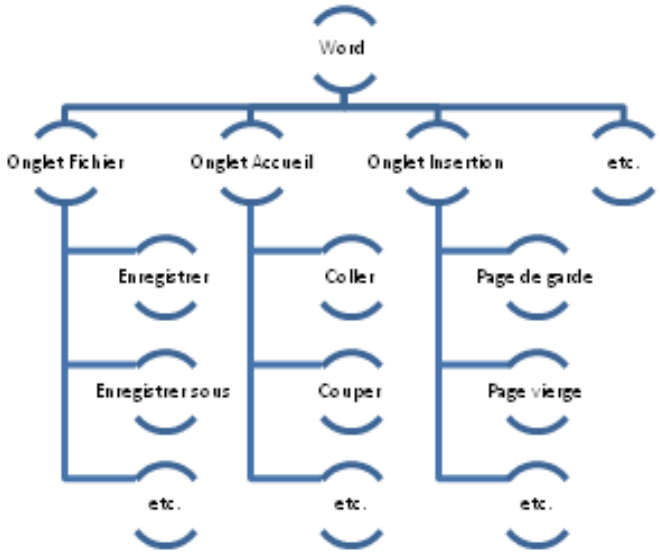

Figure 5 — Exemple de décomposition fonctionnelle pour le logiciel Word

Une fonction ou fonctionnalité est une option mise à disposition des utilisateurs. Lorsque vous utilisez le progiciel Word, vous disposez de plusieurs fonctionnalités qui sont regroupées dans des menus et des onglets. Par exemple, dans l'onglet « Accueil », vous trouverez une fonctionnalité qui permet de modifier la couleur du texte, une autre pour rechercher un mot dans le texte, etc.

Pour réaliser une fonction du logiciel, on peut utiliser un ensemble d'autres fonctions, déjà disponibles, à condition qu'on rende ces dernières suffisamment génériques.

Comme vous pouvez le voir dans l'exemple précédent, le progiciel Word contient la fonction « **Enregistrer sous** », qui utilise très probablement la fonction « **Enregistre**r ». Cette fonction se sert de l'emplacement du nom du fichier connu à l'ouverture.

Pour la fonction « **Enregistrer sous** », l'utilisateur doit préciser un endroit où enregistrer tout en donnant éventuellement un autre nom au fichier. L'enregistrement se fait alors exactement de la même manière que dans la fonction « **Enregistrer** ».

Le constat

Ce découpage n'a pas que des avantages. Les fonctions sont alors interdépendantes : une simple mise à jour du logiciel à un point donné, peut impacter en cascade d'autres fonctions. On peut éviter cela en faisant attention à créer des fonctions très génériques. Mais respecter cette contrainte rend l'écriture du logiciel et sa maintenance plus complexe.

Par exemple, si les développeurs de la société Microsoft décident de modifier le déroulement de la fonction « Enregistrer », cela impactera forcément la fonction « Enregistrer sous ». On peut aisément imaginer qu'une modification dans une fonction qui est utilisée par un grand nombre d'autres fonctions sème un peu la pagaille.

Ayant constaté les problèmes inhérents à la décomposition fonctionnelle, de nombreux experts se sont penchés sur une approche différente. De cette réflexion est née **l'approche objet**, dont UML s'inspire largement.

L'approche objet est une démarche qui s'organise autour de 4 principes fondamentaux. C'est une démarche :

itérative et incrémentale ;
guidée par les besoins du client et des utilisateurs ;
centrée sur l'architecture du logiciel ;
qui décrit les actions et les informations dans une seule entité.

- L'approche objet utilise une démarche itérative et incrémentale.

Exemple :

Prenons l'exemple de construction d'une maison. L'architecte réalise toujours plusieurs versions des plans avant d'avoir la version définitive (c'est-à-dire celui accepté par le client). Après un premier entretien avec son client, il réalise les plans pour la vision extérieure de la maison et les plans de surface. Ces plans sont soumis au client et donnent généralement lieu à des discussions. Cela permet à l'architecte de mieux cerner le besoin de son client. Il apporte alors les modifications qui s'imposent aux plans qu'il avait réalisés. Il s'agit là d'une ***première itération*** sur les plans de la vision extérieure et les plans de surface.

Par la suite, l'architecte réalisera les plans techniques qui seront nécessaires pour les différents corps de métier (les maçons, les électriciens, les plombiers, etc.). Il arrive parfois que ces plans mettent en évidence certaines contraintes qui obligent l'architecte à revenir sur les plans de surface (par exemple pour ajouter l'emplacement d'une colonne d'aération qui n'avait pas été prévue au départ). Il y a donc une ***deuxième itération*** sur les plans de surface.

- ✓ Une **démarche itérative** c'est donc faire des allers-retours entre le plan initial et les modifications apportées par les acteurs du projet.

De la même façon, la modélisation d'un logiciel se fera en plusieurs fois. Les diagrammes ne seront pas réalisés de façon complètement satisfaisante dès la première fois. Il nous faudra plusieurs versions d'un même diagramme pour obtenir la version finale. Chaque version est le fruit d'une itération (un nouveau passage sur les diagrammes). Une itération permet donc d'affiner la compréhension du futur logiciel.

La démarche est guidée par les besoins du client et des utilisateurs

Les besoins d'un client, dans un projet logiciel sont les exigences exprimées par le client au niveau des fonctionnalités, du rendu et des actions d'un logiciel. Par exemple, lors de la réalisation du progiciel Word, les clients ont probablement souhaité un logiciel qui leur permettrait d'écrire du texte, de l'effacer, d'intégrer des images, enregistrer le contenu, le supprimer, etc.

Les besoins des utilisateurs servent de fil rouge, tout au long du cycle de développement :

Lors de la ***phase d'analyse*** pour la clarification, l'affinage et la validation des besoins des utilisateurs ;

Lors de la ***phase de conception*** et de ***réalisation*** pour la vérification de la prise en compte des besoins des utilisateurs ;

Lors de la phase de test afin de garantir la satisfaction des besoins.

- ❖ L'approche objet décrit aussi bien les actions que les informations dans une même entité.

Les différents diagrammes utilisés en UML donnent tous une vision particulière du logiciel à développer. Parmi ces diagrammes, il en est un qui représente particulièrement bien ce qui est à développer si on opte pour un développement objet. Il s'agit du diagramme de classes.

L'approche objet est centrée sur le diagramme de classes qui décrit aussi bien des actions que des informations dans une même entité. Les autres diagrammes nous aident à voir clair dans les besoins et dans la solution qui est à développer. Ils permettent de compléter le diagramme de classes.

Jusque-là, on n'a pas encore défini les diagrammes que l'on peut utiliser. Pour l'instant, nous pouvons retenir que le diagramme de classes donnera une vision assez claire des informations qui seront utilisées par le logiciel, mais également des fonctions (ou opérations) qui devront s'appuyer sur ces informations. L'approche objet mélange donc habilement l'analyse des informations et des actions au lieu de les analyser séparément.

EXERCICES :

1- Essayer de définir la démarche incrémentale.
2- Quelles sont les actions de la phase d'analyse ?
3- Définir une architecture logique, quels sont ses objectifs ?
   Donnez un exemple d'architecture logique.
4- Un modèle est un objet conçu et construit, mais qu'est-ce qu'un langage de modélisation ?
5- Selon vous, quels sont les avantages de concevoir et de programmer avec *l'**approche-objet*** ?
6- Citez les étapes logiques du développement d'une application ou d'un logiciel.
   Décrivez chaque étape en quelques lignes.