Notes sur le module : *UML*

Armel Pitelet

13 décembre 2021

Table des matières

Document compagnon des slides de cours. Le cours présente l'Unified Modeling Language (UML) (qui n'existe que dans le contexte de la Programmation Orientée Objet (POO)) et se focalise sur la création et l'utilisation des Use Case Diagrams (UCDs) et Diagrammes de Classe (DdC).

1 Modélisation des cas d'utilisation

• Slide n°4 : Le but de la modélisation orientée objet est de représenter un objet de la vie courant, en objet "code", que l'on pourra utiliser dans une application. Voir POO.

• Slide n°8 : Un trait simple entre un acteur et une action représente la possibilité pour l'acteur d'effectuer cette action.

• Slide n°9 et 12 : Une flèche à tête complète mais non pleine indique la notion d'héritage. L'objet A pointé est le parent de l'objet B pointé.

• Slide n°10 : Le "include" d'une action A vers une action B indique une obligation de devoir réaliser l'action B pour pouvoir réaliser l'action A. C'est une **obligation**, ici A implique forcément d'avoir réalisé B.

• Slide n°11 : Le "extend" d'une action A vers une action B indique que la réalisation de l'action B peut entrainer (possiblement mais pas obligatoirement) la réalisation de l'action A. Il s'agit d'une relation **d'implication**, ici B implique (peut être) A.

Note :
Dans le doute entre "extend" et "include", une relation d'obligation forcé peut permettre de faire le choix entre les deux.

• Slide n°13-1 : Pour réaliser un UCD il faut se demander
— qui sont les acteurs (utilisateurs),
— quelles sont les taches réalisables par ces acteurs (fonctionnalités, objectifs).

• Slide n°13-2 : Il est également utile de distinguer les cas :
— d'utilisation global : dans lesquels on cherchera à associer les actions et acteurs le plus directement possible,
— d'utilisation par acteur : dans lesquels on peut se permettre plus de souplesse sur les liens actions/acteurs afin de mettre en valeurs les cheminement d'actions impliqué par les fonctionnalités du système.

Note :
Il n'y a pas de solution unique pour représenter un système !

2 Diagramme de classe

Le DdC représente les objets et leurs relations. Il peut être vu comme le squelette d'une application.

Définition : **Classe**,
Description abstraite d'un objet en langage informatique. C'est un *blueprint* permettant l'instantiation d'un objet. Voir POO pour les objets.

- Slide n°17 : En UML on doit donner à un attribut une visibilité, un nom, un type

- Slide n°18 : L'identifiant unique reste un simple attribut d'instance. Il est plus utile dans le code que dans le cadre de la modélisation UML.

- Slide n°19 : En UML on doit donner à une méthode une visibilité, un nom, un type de retour. Les arguments doivent aussi être nommés et typés.

- Slide n°20 : Il manque les "getter" et "setter". Mais en générale on ne les mets pas dans les DdC : leur présence est implicite.

- Slide n°22 : Il est important de spécifier la multiplicité dans les deux sens. Il est également possible de nommer le lien pour être plus précis dans la description. Si l'on nomme le lien du coté de Artist en mettant -Country cela signifie que Artist aura un attribut Country. schéma à faire pour exemple

Note :
Slide 23 à 26 à compléter via StarUML !

Définition : **Association**,
Les deux objets sont indépendants mais partage une relation (souvent par l'intermédiaire d'une troisième classe agrégat ou bien par un simple transfert d'attribut).

Définition : **Multiplicité**,
Nombre d'instance de l'association pour une instance de classe. La multiplicité de A vers B se lie du coté de B. En association simple on sous entend que les deux classes partagent des informations (voir attribut) c'est çà dire qu'elles sont chacune "au courant" de l'existence des instance de l'autre.

Note :
On peut spécifier le transfert d'attribut via association au dessus de la multiplicité associé à chaque instance. Si B est lié à une instance de A on pourra noter au dessus du 1 de la multiplicité le nom de l'attribut de A partagé à B en fournissant également sa visibilité.

Définition : **Association avec navigation**,
Association dans lequel une des classe ne connait pas les instances de l'autre. Elle est représentée par une flèche allant de la classe "sachante" à la classe "ignorante".

Définition : **Classes d'associations**,
Il s'agit d'une classe porteuse d'attributs entre deux autres classes. C (classe d'association met par exemple en relation A et B.)

Note :
Bien que pas réellement codé sous forme de classe (mais plutôt comme des dicos), ces classes d'associations sont conceptuellement importantes dans UML. Elles sont systématiques dans les relations de types "étoile/étoile".

Remarque : *Sur la représentation en ligne*
Représenter "en ligne" on parle d'une classe (simple) et non plus de classe d'association. De plus, il ne faut pas oublier qu'il faut une instance de la classe d'association pour porter chaque instances des classes associées. Une classe d'association n'est pas un tableau regoupant les liens de toutes les

instances de A et B !

Définition : **Agrégat**,
L'objet contenant l'autre est nommé **agrégateur** ou **agrégat**.

Définition : **Agrégé**,
L'objet contenu est nommé **agrégé**.

Définition : **Agrégation**,
(ou agrégation faible) Les deux objets ne sont pas liés : L'agrégé existe indépendamment de l'agrégateur mais en fait partie, ce dernier définit un concept pouvant exister seul et pouvant même faire partie de plusieurs agrégateurs.

Définition : **Composition**,
(ou agrégation forte) Relation asymétrique entre les deux objets : L'un ne peut exister sans l'autre. On parle aussi de composant(agrégé) et composite(agrégat).

Remarque : *Sur la destruction de l'agrégé*
La destruction de l'agrégé peut avoir une influence sur l'agrégateur (ou pas) : Si 0 est un nombre d'agrégé autorisé, il ne se passe rien, sinon l'agrégateur est peut-être à détruire. Ex : Le dernier employé d'une entreprise, le dernier livre d'une bibliothèque.

Définition : **Généralisation**,
Le généralisation décrit le concept d'héritage (ou l'inverse plutôt). Elle permet d'identifier un objet (instance d'une la classe fille) comme un spécialisation d'une classe plus générale (la classe mère). Dit autrement, la généralisation permet d'identifier plusieurs objets comme des sous ensembles d'objets ayant des spécificités communes. Dit dans l'autre sens la classe mère est une généralisation de la classe fille.

3 Divers

Note :
"Symphonie" est un framework PHP.

Définition : **MCD**,
Modèle Conceptuel de Données. Il modélise l'organisation d'une base de donnée (apparemment déprécié par rapport à UML).

Acronymes

DdC Diagramme de Classes. 2, 3, *Glossaire :* DdC

POO Programmation Orientée Objet. 2, *Glossaire :* POO

UCD *Use Case Diagram* (Diagramme de Cas d'utilisation). 2, *Glossaire :* UCD

UML *Unified Modeling Language* (Langage de Modélisation Unifié). 2–4, *Glossaire :* UML

Glossaire

DdC Le Diagramme de classe (DdC) est une représentation graphique qui modélise les différentes classes (entités du systême) et leurs relations (association, agrégation, composition, généralisation)..

POO La Programmation Orientée Objet (POO) est un paradigme de programmation. Représentation numérique de la vie courante. Un objet est composé de ses attributs (caractéristiques) et méthodes (fonctionnalité).

UCD Le *Use Case Diagram* (UCD) est une représentation graphique qui expose les acteurs d'un système (utilisateurs) et les actions (fonctionnalités, objectifs) possibles pour ces acteurs (c'est à dire les interactions avec le système). Cette représentation est axée sur les fonctionnalités du système utilisables par l'utilisateur. L'intérêt est d'identifier les fonctionnalités principale du système et le cheminement d'action nécessaire pour y accéder. Cela permet de structurer au mieux les besoins utilisateurs et les objectifs correspondants d'un système.

UML L'*Unified Modeling Language* (UML) est un langage de modélisation orienté objet. Il s'agit d'un ensemble de représentations graphiques l'analyse et la réflexion pré-développement. Son but est d'aider à la conception d'applications, en spécifiant les différents éléments participant à celle-ci. Il permet de formaliser au mieux le besoin du client à travers un ensemble de représentations graphiques permettant d'expliquer les différentes fonctionnalité du processus que l'on souhaite concevoir ainsi que de gagner du temps. UML est aussi utile pour exposer le projet à d'autres développeurs.