Chapitre 2

Langage de modélisation UML

2.1 Introduction

UML[1] est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la « conception orientée objet ».

UML est l'accomplissement de la fusion de précédents langages de modélisation objet : *Booch*, OMT, OOSE. Principalement issu des travaux de Grady Booch, James Rumbaugh et Ivar Jacobson, UML est à présent un standard défini par l'OMG[2].

2.2 Principes

UML est :

— un norme ;

— un langage de modélisation objet ;

— un support de communication ;

— un cadre méthodologique pour une analyse objet.

UML n'est pas :

— une méthode d'analyse objet.

UML définit 14 types de diagrammes qui vont permettre de modéliser un projet tout au long de son cycle de vie :

1. *Unified Modeling Language* : méthode de représentation de la modélisation orientée objet.
2. *Object Management Group* : association américaine de promotion du modèle objet.

- diagrammes statiques :
  1. diagramme de classes ;
  2. diagramme d'objets ;
  3. diagramme de composants ;
  4. diagramme de déploiement ;
  5. diagramme de paquetages ;
  6. diagramme de structure composite ;
  7. diagramme de profils.
- diagrammes comportementaux :
  1. diagramme de cas d'utilisation ;
  2. diagramme d'états-transitions ;
  3. diagramme d'activités.
- diagrammes dynamiques (ou d'interaction) :
  1. diagramme de séquences ;
  2. diagramme de communication ;
  3. diagramme global d'interaction ;
  4. diagramme de temps.

Dans le cadre de ce cours, nous étudierons uniquement les « diagrammes de classes » afin d'être capable de lire et décoder les diagrammes fournis dans les exercices ou à l'examen. Pour un cours complet en français, consulter [UML 12].

2.3 Diagramme de classes

Avertissement

La présentation du diagramme de classes est simplifiée en omettant volontairement certains détails inutiles pour une première approche.

2.3.1 Classe

Rappel : une classe est une description abstraite (condensée) d'une famille d'objets du domaine de l'application. Elle définit leur structure, leur comportement et leurs relations.

Les classes sont représentées par des rectangles compartimentés (voir Fig. 2.1) :

— le 1er compartiment représente le nom de la classe ;

— le 2e compartiment représente les champs de la classe ;

— le 3e compartiment représente les opérations (méthodes) de la classe.

Légende :

— le symbole « - » indique que le champ ou la méthode est **privé** (`private`).

— le symbole « + » indique que le champ ou la méthode est **public** (`public`).

— un champ souligné indique que le champ est statique (mot-clé `static` dans C#)

— le mot-clé `«property»` préfixe les propriétés

— le mot-clé `«ctor»` préfixe les constructeurs

— le mot-clé `{read-only}` indique que le champ est en lecture-seulement (mot-clé `readonly` dans C#)

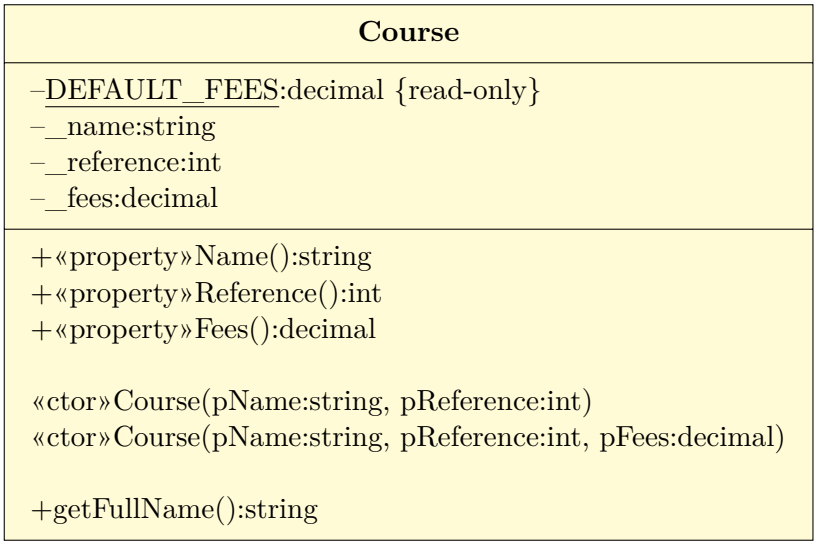

FIGURE 2.1 – Représentation d'une classe en UML

2.3.2 Liaison entre les classes

En « conception orientée objet », on distingue couramment 4 variétés de relations entre les classes (voir Fig. 2.2) :

la dépendance (relation `USES`), qui indique qu'un composant utilise un autre composant par exemple, une maison utilise de l'électricité ;

la composition (relation `OWNS_A`), qui indique qu'un composant est constitué d'un autre - par exemple, une maison comporte une (ou plusieurs) salles de bain ;

l'agrégation (relation `HAS_A`), qui indique qu'un composant possède un autre - par exemple, une maison possède un (ou plusieurs) occupants **mais** les occupants vont **survivre** à la maison ;

l'héritage (relation `IS_A`), qui indique qu'un composant est une spécialisation d'un autre - par exemple, une maison individuelle est une forme de bâtiment.

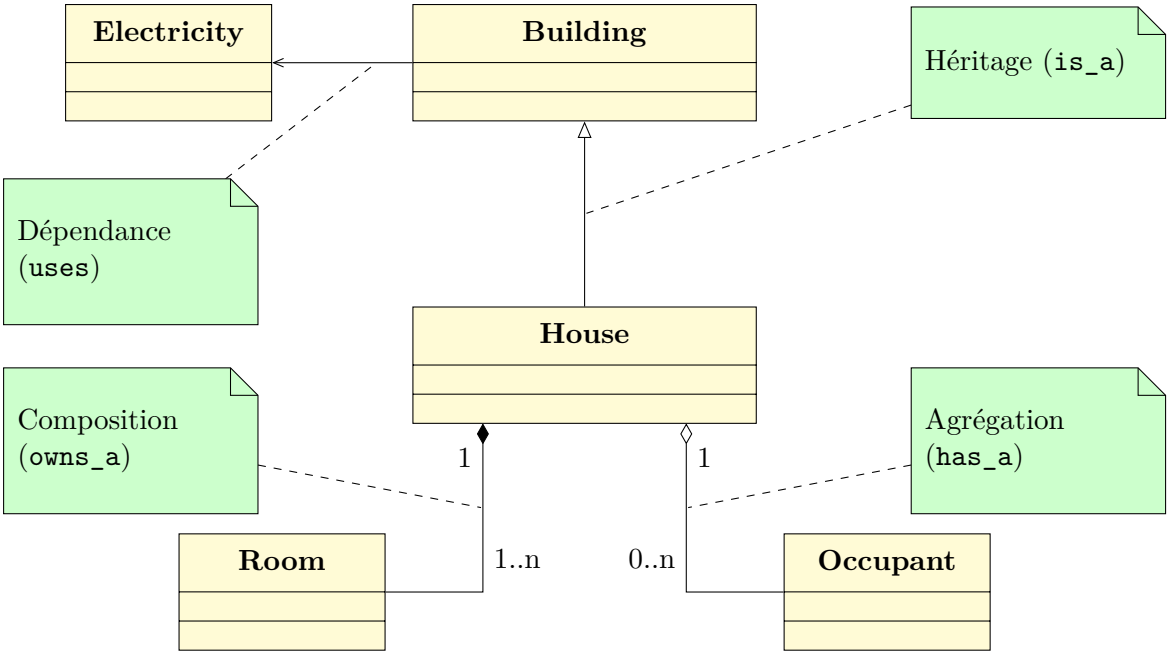

FIGURE 2.2 – Symboles de relations

2.3.2.a Dépendance (*dependency*)

La dépendance entre `ClassA` et `ClassB` indique que :

— `ClassA` possède au moins une méthode ayant un paramètre de type `ClassB` ;

— et/ou `ClassA` possède au moins une méthode retournant un type `ClassB`.

Voir Fig.2.3

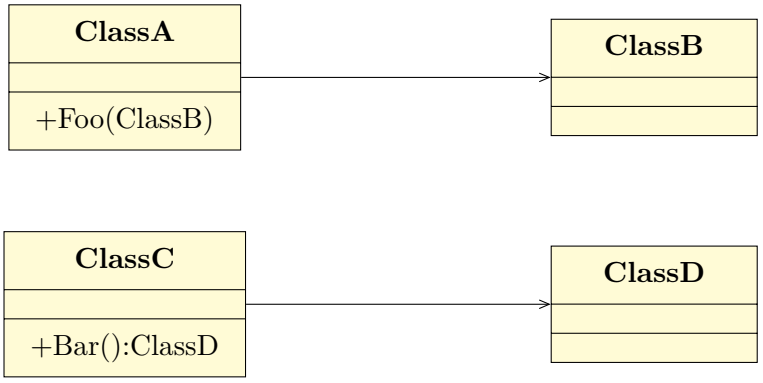

FIGURE 2.3 – UML : dépendance (*dependency*)

2.3.2.b Agrégation (*aggregation*)

Agrégation = association partagée, *shared association*

L'agrégation entre `Car` et la `Engine` montre que :

- il y a une association forte entre `Car` et `Engine` ;
- `Engine` possède d'autres associations fortes (d'où le nom *shared association*) ;
- il existe des *getter/setter* ou des méthodes *add/remove* publics pour les objets de type `Engine` ou `Wheel`.

Remarques :

- l'association peut remplacer l'aggrégation dans tous les cas ;
- l'aggrégation ne peut pas remplacer la dépendance.

> *Martin Fowler suggest that the aggregation link should not be used at all because it has no added value and it disturb consistency, Quoting Jim Rumbaugh « Think of it as a modeling placebo », [AVI 12].*

Voir Fig.2.4

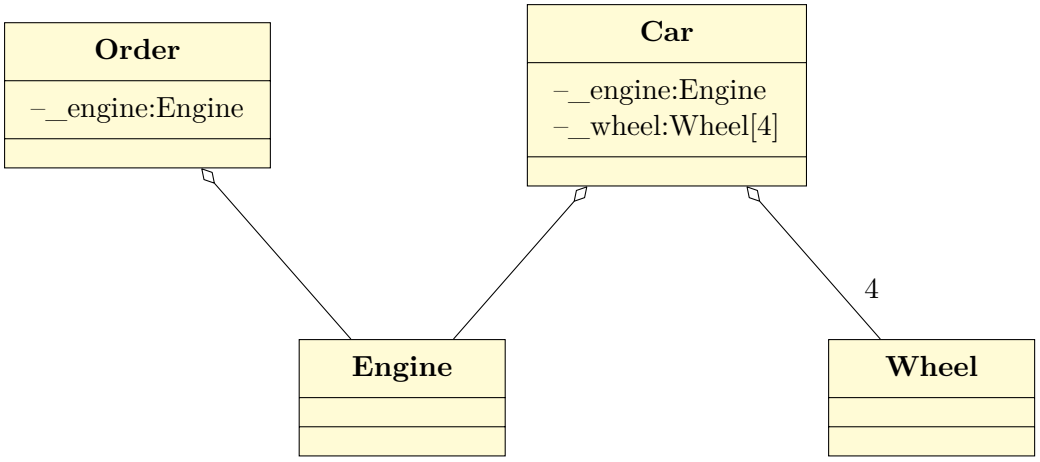

FIGURE 2.4 – UML : agrégation (*aggregation*)

2.3.2.c Composition (*composition*)

Composition = association non partagée, *not-shared association*

La composition entre `Person` et `Hand` montre que :

- `Person` possède au moins un champ de type `Hand` ;
- la destruction d'un objet de type `Person` entrainera la destruction des objets de type `Hand` ;
- il n'existe pas de *getter/setter* publics pour l'objet de type `Hand`.

Voir Fig.2.5

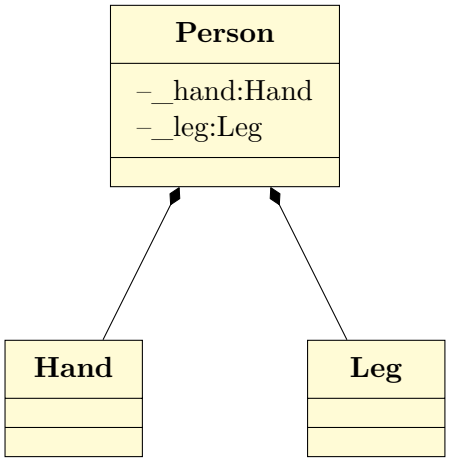

FIGURE 2.5 – UML : composition (*composition*)

2.3.2.d Héritage (*inheritance*)

`House` hérite de `Building` indique que :

- `House` possède les mêmes champs que `Building` et peut en posséder d'autres ;
- `House` possède les mêmes méthodes que `Building` et peut en posséder d'autres.

Voir Fig.2.6

2.3.2.e Récapitulatif

Le tableau 2.1 résume les différentes liaisons entre les classes.

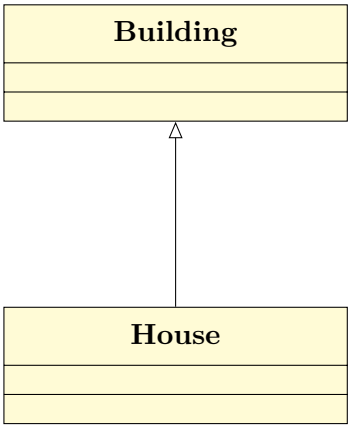

FIGURE 2.6 – UML : Héritage (*inheritance*)

Liaison	Link	*Symbole*	*Exemple*
dépendance	*dependency*	⟶	A House `USES` Electricity
agrégation	*agregation*	⟶◇	A House `HAS_A` Occupant.
composition	*composition*	⟶◆	A House `OWNS_A` Room.
héritage	*inheritance*	⟶▷	A House `IS_A` Building.

TABLE 2.1 – UML : différentes liaisons entre les classes

2.4 Éditeurs de diagrammes UML

2.4.1 Introduction

Il existe de nombreux logiciels pour réaliser des diagrammes UML, voir :

— `http://fr.wikipedia.org/wiki/Comparaison_des_logiciels_d'UML`.

— `http://stackoverflow.com/questions/15376/whats-the-best-uml-diagramming-tool`

2.4.2 Logiciel *ArgoUML*

ArgoUML est multi-plateforme, *open-source* et permet de générer le code des classes pour les langages : C++, *C#*, Java, PHP4, PHP5, SQL (`http://argouml.tigris.org`).

2.4.2.a Paramètres des méthodes

L'onglet « *Propriétés* » d'un paramètre de méthode permet de préciser :

— `in` : paramètre en entrée uniquement ;

— `out` : paramètre en sortie uniquement ;

— `inout` : paramètre en entrée/sortie ;

— `return` : valeur retournée par la méthode.

2.4.2.b Constructeur, destructeur

L'onglet « *Stereotype* » permet d'indiquer qu'une méthode est un constructeur ou un destructeur.

2.4.2.c Liaison entre les classe

Représentation sur un diagramme de classe (avec « *Argo UML* », voir Fig. 2.7) :
l'héritage (relation `IS_A`) : outil « *New Generalization* »
la composition (relations `HAS_A`) : outil « *New Composition* »
l'agrégation (relations `HAS_AN`) s : outil « *New Aggregation* »

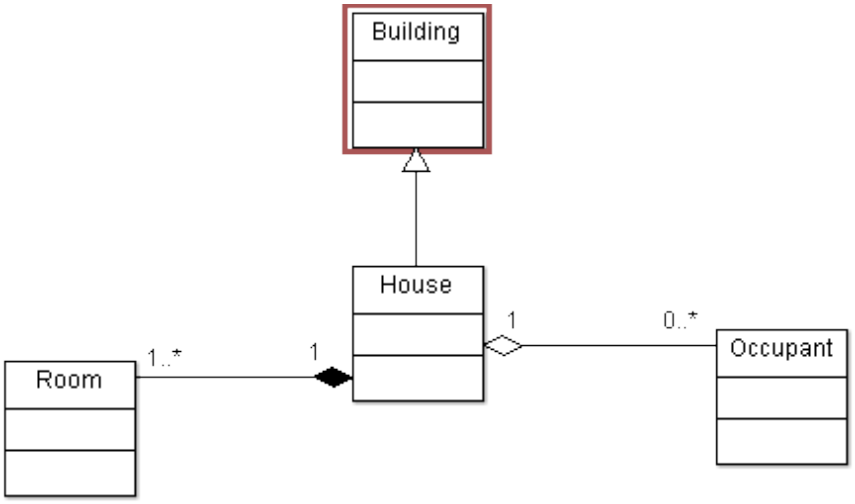

FIGURE 2.7 – Exemple de diagramme de classes avec *ArgoUML*

2.4.3 *TopCoder UML Tool*

TopCoder UML Tool est multi-plateforme, *libre* et permet de générer le code des classes pour les langages : *C#* et Java (`http://apps.topcoder.com/wiki/display/tc/TopCoder+UML+Tool`).

2.4.4 *Violet UML Editor*

Violet UML Editor est multi-plateforme, *libre* et ne permet pas de générer le code des classes (`http://sourceforge.net/projects/violet`).

2.4.5 *Visual Paradigm for UML*

Visual Paradigm for UML est multi-plateforme, commercial (mais avec une version gratuite pour une utilisation non commerciale (*Community edition*)) et ne permet pas de générer le code des classes en version *Community edition*.
(`http://www.visual-paradigm.com`)

2.4.6 Logiciel *UMLet*

UMLet est multi-plateforme, *open-source* et très simple d'utilisation (c'est plutôt un logiciel de dessin orienté UML).

- `http://www.umlet.com/`
- `http://www.umlet.com/umletino/` : version (limitée) en ligne
- `http://jl2tho.blogspot.fr/2012/05/test-dumlet-un-plug-in-eclipse-pour-uml.html` : *plugin* pour *Eclipse*
- tutoriel : `http://la-vache-libre.org/umlet-creez-facilement-vos-diagrames-uml-sous-gnulinux/`

2.4.7 Logiciel *Enterprise Architect*

Enterprise Architect est disponible uniquement sur *Windows*, commercial. (`http://www.sparxsystems.com/`)