I. Les bases d'UML en C++

Ce chapitre contient outre la description des différents diagrammes UML, des exemples concrets d'utilisation de ces diagrammes, ainsi qu'un exemple d'implémentation en C++.

Un nombre important d'exercices est proposé pour permettre de progresser dans l'apprentissage, la compréhension, l'utilisation du langage UML et de sa « traduction » en C++.

Les corrigés des exercices se trouvent en annexe.

Bien que ce ne soit pas obligatoire, il est conseillé de se servir d'un logiciel (voir en annexe pour des suggestions de programmes) pour dessiner les diagrammes UML. Il existe de nombreux programmes (certains sont gratuits) qui permettent de tracer rapidement et efficacement des diagrammes UML. J'ai utilisé personnellement un « vieux » logiciel (Star UML) pour les diagrammes, car celui-ci est rapide, léger et présente toutes les caractéristiques nécessaires. Pour éditer, compiler, tester le code C++, j'ai employé l'environnement Qt dans sa version 4.8. Bien entendu, d'autres outils sont utilisables, que ce soit pour dessiner des diagrammes UML ou pour compiler du C++. Faites simplement attention : j'ai eu recours à la norme 2.x pour UML et à la norme de 1998 pour le C++ ; il ne faut donc pas employer un outil répondant à des normes plus anciennes. Sauf cas très particuliers, les normes plus récentes sont utilisables également.

01. Introduction

a) UML : qu'est-ce que c'est ?

UML (*Unified Modeling Language*) est un langage de modélisation graphique. Défini par l'OMG (*Object Management Group*), il est un standard industriel pour la conception orientée objet des applications.

Il est utilisé principalement pour les phases de conception des programmes ainsi que pour la production de la documentation. Une partie du code peut être générée automatiquement à partir d'un diagramme. Inversement, certains diagrammes peuvent être générés automatiquement à partir du code C++ (rétroconception).

UML n'est pas une méthode : c'est une norme de **représentation**. Il existe différentes méthodes de conception objet qui utilisent UML pour représenter la conception.

Les diagrammes UML se répartissent en trois catégories :

- Les diagrammes de **structure** (statiques). Usuellement les premiers réalisés lors de la conception d'un système, ces diagrammes permettent de définir l'architecture, à des niveaux plus ou moins précis, du système et de chacune de ses parties.
    - Diagramme de paquetages
    - Diagramme de classe
    - Diagramme d'objets
    - Diagramme de composants
    - Diagramme de déploiement
- Les diagrammes de **comportement**. Ils précisent le comportement du système.
    - Diagramme des cas d'utilisation
    - Diagramme états/transitions
    - Diagramme d'activité
- Les diagrammes **d'interactions** (dynamiques). Ils précisent les interactions entre différents acteurs ou différents composants du système.
    - Diagramme de séquence
    - Diagramme de communication
    - Diagramme de temps

UML sert à représenter graphiquement en utilisant une norme précise (que donc plusieurs développeurs peuvent partager), la modélisation d'un système, en particulier lors de la phase de conception. UML n'est pas une méthode, ni un langage de programmation.

Lors de la conception il est parfois inutile de décrire tous les diagrammes UML. De même, dans un diagramme, il est inutile de tout décrire. Le degré de précision est fonction du contexte, de la complexité du système, de l'étape de conception…

Il est possible de commencer l'analyse par le comportement (et donc la définition des diagrammes de comportement, diagramme des cas d'utilisation en premier) ou par la structure (diagramme des *packages*, diagramme des classes), mais également par les deux en parallèle !

Le présent ouvrage ne décrit pas une méthode de conception, mais donne des clés pour utiliser correctement UML lors d'un développement en C++.

Notes

La note est un commentaire, une annotation libre qui peut concerner n'importe quel élément UML.

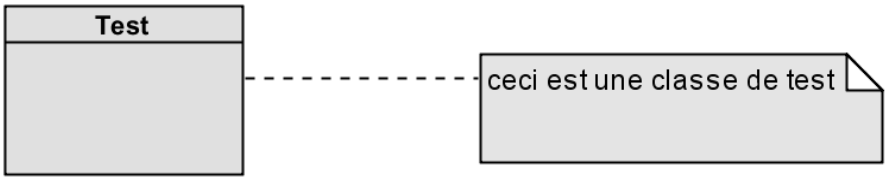

Figure 1 : note attachée à une classe

Stéréotypes

UML ne définissant pas un grand nombre de symboles graphiques, ceux-ci peuvent être étendus par la définition de stéréotype**s** UML. Les stéréotypes permettent d'étendre UML facilement.

Il existe des stéréotypes standards, mais UML n'est pas limité en nombre de stéréotypes.

Un stéréotype se note entre guillemets typographiques : « stéréotype ». Il peut qualifier n'importe quel objet UML.

Dans la figure suivante, la classe `Client` est stéréotypée comme étant « `worker` », c'est-à-dire représentant l'abstraction d'un **acteur** du système. Un stéréotype peut avoir une icône associée. La figure suivante montre la classe `Client` de plusieurs manières (textuelle, en icône, en texte avec décoration) :

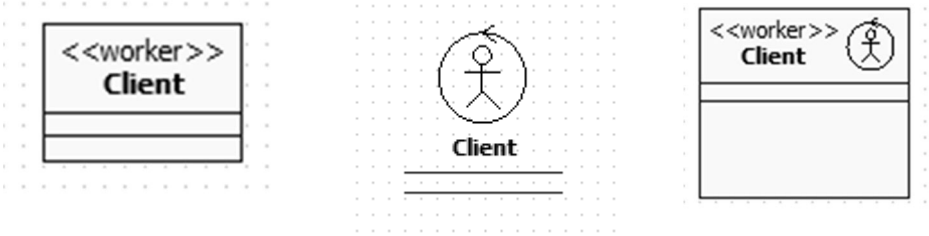

Figure 2 : différentes représentations d'un stéréotype

Comme son nom l'indique, un stéréotype peut symboliser un point commun entre plusieurs objets UML.

b) Le langage C++

Le langage C++ est un langage de programmation, utilisant notamment le paradigme objet. Par conséquent C++ peut implémenter simplement les notions conçues en UML.

UML est souvent associé au langage Java : en effet, Java est « pur objet », assez simple d'utilisation, et né en même temps qu'UML… Néanmoins UML n'est pas fait pour Java, ni Java pour UML !

C++ n'est pas un « pur » langage objet : en effet, C++ est multi-paradigmes (procédural, objet, générique…) mais cela ne veut pas dire qu'il n'est pas « compétent » pour l'objet, bien au contraire ! C++ hérite de *Smalltalk*, qui est le premier des langages objets. La plupart des notions de la programmation par objets (et donc d'UML) viennent de *Smalltalk* et ont été implémentées en C++.

C++, créé au début des années 80, a connu des évolutions :

- Le C++ original (1983) dont l'ancêtre « *C with classes* » empruntait une grosse partie au langage C. On peut considérer que l'ouvrage *The C++ programming language* par B. Stroustrup, créateur du C++, définit la version 1.0 du langage.
- Au début des années 1990, *The annotated C++ reference manual* ajoute des fonctionnalités au langage qui n'est pas encore standardisé.
- En 1998, le comité ANSI et l'ISO normalisent le C++ et sa bibliothèque standard. Cette version du C++ est la plus répandue.
- En 2003, la norme de 98 est révisée.
- En 2011, une nouvelle version du C++ est normalisée. Celle-ci apporte énormément de nouveautés, tout en conservant la compatibilité de C++ avec la version normalisée précédente. Cette version est fréquemment appelée C++ 11.

Bien que la norme actuelle (C++ 11) soit très intéressante, elle n'est pas encore implémentée sur tous les compilateurs C++ : les codes C++ présentés dans ce livre respectent donc la norme précédente (C++ 03, voire C++ 98) et ne tirent donc pas parti des ajouts de la dernière norme.

Le but de ce livre n'étant pas l'apprentissage de C++, si vous n'êtes pas familier avec ce langage, je vous conseille la lecture préalable de mon précédent livre, *La programmation objet en C++*, ou d'un autre ouvrage destiné à enseigner le langage lui-même.

02. Diagramme de paquetages

Ce diagramme décrit les différentes relations entre les paquetages existants dans un système. Un paquetage (en anglais *package*) désigne une couche du logiciel, un ensemble d'entités de niveau inférieur et représente donc une structure assez élevée de celui-ci.

a) Paquetages UML

Un paquetage en UML est symbolisé par un **pictogramme** représentant un classeur. Le nom du paquetage est indiqué dans en haut de celui-ci, ou dans l'onglet en haut à gauche.

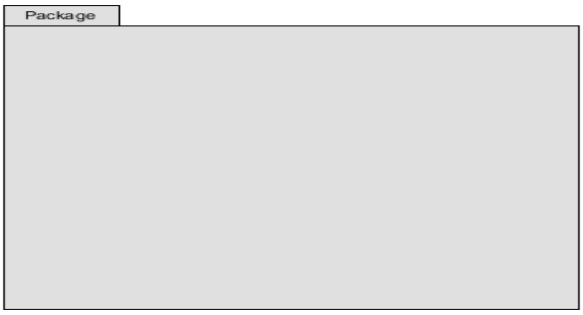

Figure 3 : représentation UML d'un paquetage

Différents paquetages peuvent entretenir trois types de relations entre eux :

- **Dépendance** (ou utilisation) : Un paquetage dépend d'un autre, ou utilise cet autre paquetage.
- **Importation** : Les noms des éléments du paquetage importés sont ajoutés dans l'espace du paquetage important.
- **Fusion** : Les contenus des paquetages sont fusionnés.

Dans les trois cas, le dessin est une flèche simple en pointillé, avec la présence d'un stéréotype UML.

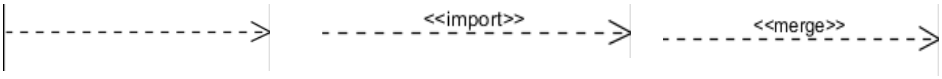

Figure 4 : relations de dépendances, d'importation, de fusion

Un diagramme de paquetages peut aussi servir à représenter l'utilisation

de bibliothèques externes, de frameworks, etc.

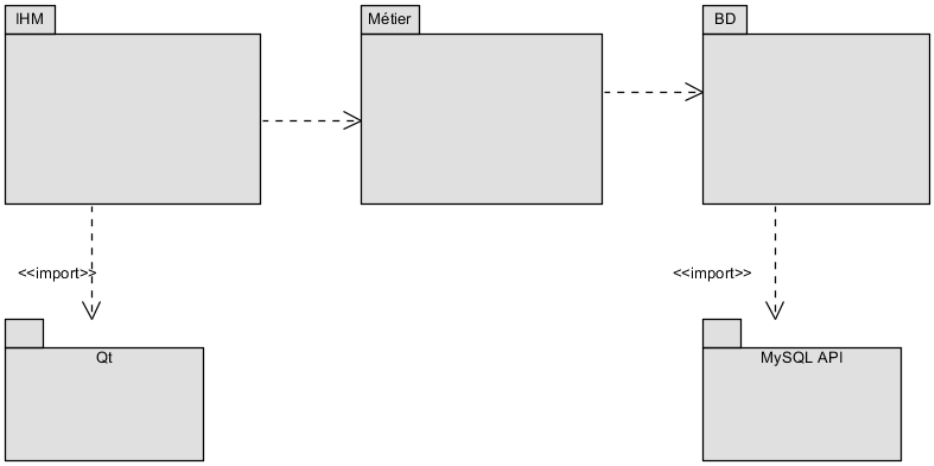

Figure 5 : diagramme de paquetages UML

La Figure 5 indique dans le système la présence de trois grandes sous-parties (trois couches), ainsi que le sens des liens d'utilisation. Ici on voit que la partie « IHM » (l'interface homme/machine) va être amenée à utiliser des classes de la partie « Métier » (mais pas l'inverse), que cette même partie « Métier » va se servir des classes présentes dans le paquetage « BD ». De même, le paquetage « IHM » va importer le paquetage « Qt » (ici représentant le framework Qt) et le paquetage « BD » importe le paquetage « MySQL API » (ici représentant les fonctions de l'API du SGBD MySQL).

Le mode de communication entre les couches (la matérialisation de la dépendance) peut être indiqué soit dans la documentation annexée, soit dans une note liée au diagramme (un commentaire).

b) Traduction en C++

En C++, la notion de paquetage est représentée par la notion d'espace de nom (**`namespace`**) : il est donc facile de faire correspondre le diagramme UML avec du code C++. Il est à noter que le diagramme de paquetages seul ne donne pas le moindre code en C++, mais fournit un canevas général pour l'application. Nous trouverons donc dans notre application autant d'espaces de nom que de paquetages UML. Ci-dessous la définition d'un **espace de**

nom en C++ :

```
namespace IHM
{
}
```

On peut, pour simplifier la structure des fichiers sources du projet, faire correspondre un paquetage (un **namespace**) à un sous-répertoire du projet, contenant les différents fichiers ayant un lien **logique** entre eux (par exemple : la gestion de l'interface homme-machine). Ce n'est pas obligatoire.

Les relations entre paquetages en UML se traduisent de la manière suivante en C++ :

- la dépendance simple est une simple **information** sémantique. Elle peut, en C++, se traduire par une instruction d'inclusion d'un (ou plusieurs) fichier(s) d'en-tête contenant la description (prototypes) du paquetage.
- l'import traduit, en plus, l'ajout des noms du paquetage dans l'espace courant. En C++ cela se transpose par une directive **using namespace**. Par exemple, l'import du paquetage `std` deviendra en C++ :

```
using namespace std ;
```

Exercice 01. **Conception d'un diagramme de paquetages**

Nous devons développer une application assez importante, comportant une partie d'interface graphique, une partie avec des calculs, une partie gérant le stockage des données, une partie gérant les communications réseau. La partie gérant l'interface graphique utilisera le framework `Qt`, la partie avec des calculs utilise une bibliothèque appelée `CPPMath`, le stockage des données se fera dans un fichier `XML`, en utilisant les API `MS-XML`, et la partie de communications réseau utilise une API appelée `CPPSocks`.