

La conception orientée objet (COO)

1. Relation entre la POO et la COO

La conception d'un programme orienté objet en C++ est une étape qui demande une certaine maturation. L'essence d'un tel programme ne tient pas seulement à l'implémentation des algorithmes qu'il contient, mais bien à la structure des classes et aux relations qui les réunissent. Nous nous proposons dans cette section de décrire les approches de méthodes liées à la conception de programme C++.

a. L'approche initiale de C++

Avant tout, rappelons-nous que C++ a été créé pour réaliser des applications à destination du domaine des télécommunications. Ce langage n'est donc pas seulement un travail de recherche, mais c'est aussi le fruit d'un travail d'ingénierie informatique. Bien entendu, son concepteur Bjarne Stroustrup s'est assuré qu'il était suffisamment général pour être adapté à d'autres situations. Pour atteindre cet objectif, il a conçu trois éléments essentiels :

- ˆ le langage C++ en lui-même ;
- ˆ la bibliothèque standard STL ;
- ˆ une méthode de conception orientée objet pour C++.

La méthode proposée par Bjarne Stroustrup repose évidemment sur son expérience de conception d'applications antérieure à C++. Elle se fonde sur un certain nombre de thèmes forts, dont certains sont des classiques des méthodes de conception logicielle, orientée objet ou pas. Citons parmi ces thèmes la délimitation du système, la complexité des applications par rapport au degré d'abstraction des outils, ou encore les cycles de conception et de programmation perçus comme des activités itératives.

Le processus de développement minimal est constitué de trois étapes :

- ˆ analyse ;
- ˆ conception ;

- ˘ implémentation.

L'étape qui nous préoccupe est justement celle de la conception. L'auteur de la méthode organise des sous-processus à partir du découpage suivant :

- ˘ identification des classes, des concepts et de leurs relations les plus fondamentales ;
- ˘ spécification des opérations, c'est-à-dire des méthodes ;
- ˘ spécification des dépendances (des cardinalités dirait-on en UML) ;
- ˘ identification des interfaces pour les classes les plus importantes ;
- ˘ réorganisation de la hiérarchie des classes ;
- ˘ utilisation de modèles pour affiner le diagramme.

En conclusion, Bjarne Stroustrup a proposé une méthode générale, adaptée à la conception de programmes C++. Il est vraisemblable que le langage lui-même a été influencé par la méthode de conception orientée objet.

b. UML et C++

Un des intérêts de l'approche décrite ci-dessus vient du fait qu'elle est compatible avec un système de notation comme UML. De fait, UML a été créé à une époque où C++ était l'un des rares langages de programmation orientée objet disponibles pour l'industrie.

UML est l'acronyme d'*Unified Modeling Language*. Il s'agit d'un formalisme qui unifie les travaux en matière de conception orientée objet développés pour la plupart au cours des années 70 et 80. Ce formalisme est à l'initiative d'une société, Rational, qui a ensuite proposé le logiciel de modélisation Rose, ainsi qu'une méthode à part entière appelée RUP (*Rational Unified Process*).

Comme C++ était l'un des principaux langages du marché, il a fortement influencé l'élaboration d'UML. Aussi est-il fréquent de trouver des logiciels de modélisation UML capables de transformer automatiquement un diagramme de classes en programme C++, sans toutefois pouvoir fournir l'implémentation correspondante !

Le formalisme UML est constitué de neuf diagrammes. Il n'est souvent pas nécessaire d'exécuter l'ensemble des diagrammes pour parvenir à une modélisation conforme. Par contre, certains diagrammes nécessitent de s'y prendre en plusieurs fois.

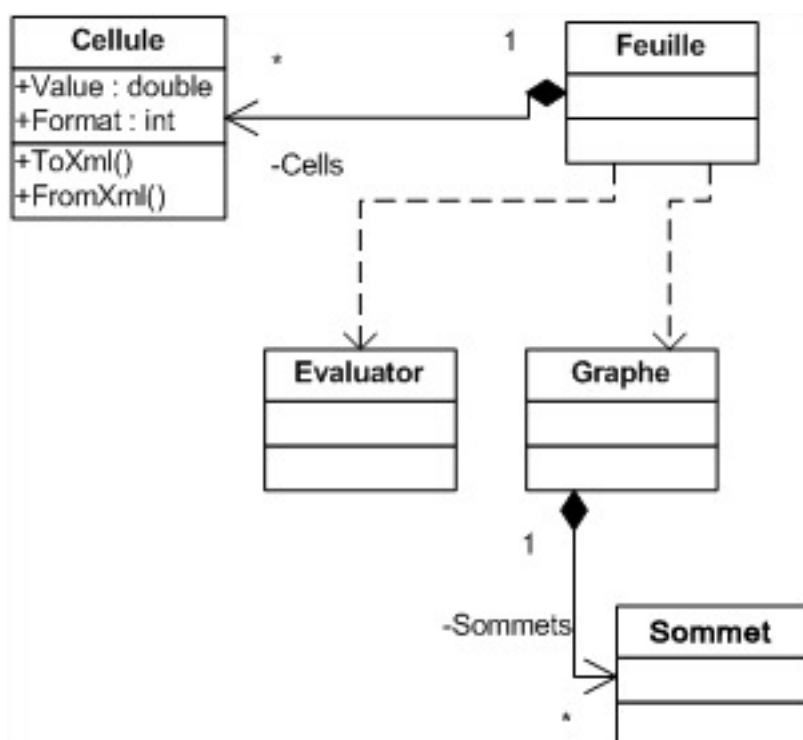
On parle alors d'un processus de modélisation itératif.

Voici la liste des diagrammes constitutifs du formalisme UML 1 :

Diagramme des cas d'utilisation	Vision fonctionnelle	Pose les limites du système en listant les acteurs et leurs intentions, sans s'attarder sur le "comment".
Diagramme de collaboration	Vision fonctionnelle	Les cas d'utilisations sont décrits sous la forme de scénarios textuels avant d'être formalisés en diagrammes de collaboration. C'est le début du comment.
Diagramme d'activité	Vision fonctionnelle	C'est une version "workflow" du diagramme de collaboration plus adaptée à la description de certains scénarios.
Diagramme état-transition	Vision dynamique	C'est une version technique d'un workflow ; on considère des changements d'état au sein de classes qui prennent progressivement forme.
Diagramme de séquence	Vision dynamique	Semblable au diagramme d'état transition, le diagramme de séquence étudie la constitution de classe sous l'angle du temps, en faisant progressivement ressortir des méthodes.
Diagramme du domaine	Vision "métier"	Ce diagramme ne fait pas partie d'UML mais il est indispensable. Les classes métier sont souvent structurées à partir des tables SQL.
Diagramme de classes	Vision statique	Ce diagramme est un peu la version affinée, superposée des précédents diagrammes.
Diagramme de composants	Vision système	Lorsque les classes sont stéréotypées, pourvues d'interfaces techniques, elles sont appelées composants.
Diagramme de déploiement	Vision système	Ce diagramme précise sur quels nœuds les composants doivent être déployés.

Des outils tels que Visio Architect ou Enterprise Architect sont capables de reproduire le diagramme de classes d'un programme, ou inversement de générer du C++ à partir d'un diagramme de classes. Comme UML et C++ sont deux langages très généraux, ils s'entendent parfaitement et toute la panoplie du C++ peut être écrite en UML.

Le diagramme de classes suivant montre comment modéliser une partie du tableau `CLIMulticube` étudié au chapitre Les univers de C++. C'est bien la preuve que le formalisme UML n'est aucunement réservé aux langages C# ou Java.



2. Les design patterns

Les design patterns sont des façons standards d'aborder des problèmes logiciels. Il s'agit de constructions de classes répertoriées que l'on adapte (développe) au gré des besoins.

Nous avons vu au chapitre Les univers de C++ que les MFC implémentaient le pattern MVC (Modèle Vue Contrôleur) à travers l'architecture document vue. Il existe de très nombreux modèles et certains générateurs de code peuvent constituer l'ossature d'une application en paramétrant des patterns pour un langage donné.

Pour illustrer les patterns, nous développons l'exemple du Singleton. Il s'agit de veiller à ce qu'une classe ne puisse être instanciée qu'une et une seule fois. Un exemple habituel d'utilisation de ce pattern concerne les fichiers de configuration qui ne doivent pas exister en plusieurs exemplaires au sein d'une application.

```
class Configuration
{
private:
    static Configuration* instance;
    Configuration()
    {
        // Le constructeur privé empêche toute instanciation
        // externe à la classe
    }

public:
    static Configuration* getInstance()
    {
        if(instance==NULL)
        {
            printf("Instanciation de Configuration\n");
            instance = new Configuration();
        }

        return instance;
    }

public:
    bool param1,param2;
};

Configuration* Configuration::instance=NULL;

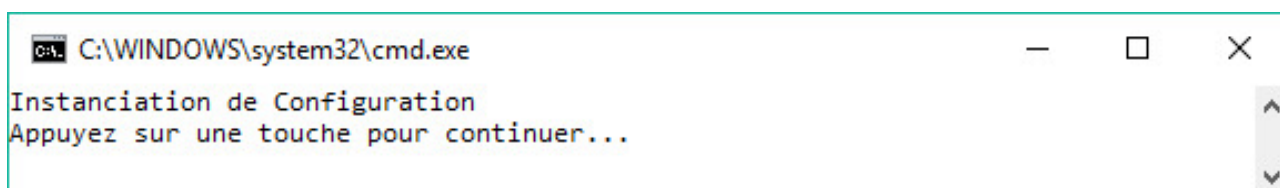
int _tmain(int argc, _TCHAR* argv[])
{
    // utilisation du singleton
    Configuration::getInstance()->param1 = true;
    Configuration::getInstance()->param2 = true;

    return 0;
}
```

```
}
```

Comme le constructeur est privé, il est impossible d'instancier la classe `Configuration` en dehors de celle-ci. Cependant, nous pouvons confier cette instanciation à une méthode publique et statique, `getInstance()`. Cette dernière s'appuie sur un champ statique, `instance`, pour contrôler l'unique instanciation de la classe. Au premier appel de `getInstance()`, le champ `instance` égale `NULL` et la classe est instanciée. Tous les appels successifs travailleront à partir de cet unique objet. De cette façon, les paramètres de la configuration `param1` et `param2` n'existeront qu'en un seul exemplaire au sein du programme.

La copie d'écran ci-dessous montre que l'instanciation n'a eu lieu qu'une seule fois :



```
C:\WINDOWS\system32\cmd.exe
Instanciation de Configuration
Appuyez sur une touche pour continuer...
```