UNIVERSITÉ DE SHERBROOKE

UCAC | icam

PROGRAMMATION CONCURRENT

PROJET PROGRAMMTION SYSTEME

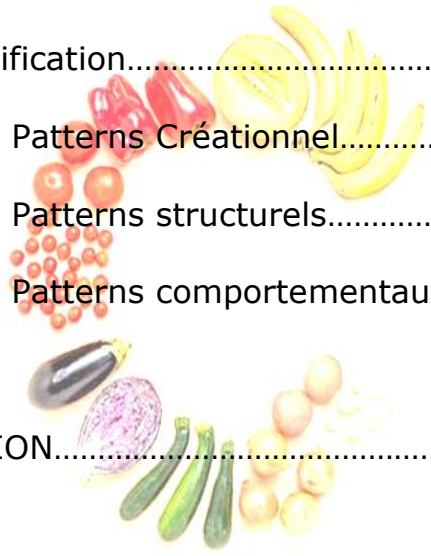
« DESIGN PATTERN»

Depuis quelque temps, dans un restaurant, les incidents regrettables se multiplient : file d'attente trop importante, désorganisation du service entre la salle et la cuisine, manque de matériel pour servir ou cuisiner entrainant de très longues attentes en salle pour les clients… En résumé, les clients partaient très insatisfaits. Aujourd'hui on sait qu'un client insatisfait prend une des applications pour noter le restaurant et saisit un avis négatif, ce qui a pour conséquence de baisser la réputation de la maison.

Pour cette raison, le directeur avec sa grande chaîne internationale de restaurants souhaite s'équiper d'une nouvelle application informatique pour améliorer l'accueil du public, le remplissage des salles, la gestion des réservations et l'organisation du travail en cuisine, décidé d'investir dans le développement d'une application de gestion et supervision du fonctionnement de son restaurant (salle de restauration et cuisine). Il fait donc appel à vous, étudiants ingénieurs CESI, Ucac-Icam, en X3, pour l'aider sur ce projet.

Vous avez donc pour mission, d'analyser les résultats de la simulation et faire des propositions d'amélioration. Vos propositions seront dans l'obligation de reposer sur des éléments mesurables pour mettre en évidence, sans ambiguïté, les gains que vos propositions suggèrent et donc doivent être exprimées en unité de mesure (temps, pourcentage, argent, etc…).

TABLE DE MATIERES



1

1.1. Définition

Un **design pattern** est un élément de logiciel orienté objet réutilisable présentant une solution générique à des problèmes récurrents.
A la différence d'un Framework, le design pattern n'apporte aucune solution globale à un problème général. Il se cantonne à des problèmes de plus petites tailles plus concrets.
Il s'adresse donc aux designers afin de leur permettre de modulariser, réutiliser, et structurer leur conception.

1.2. Avantages des design patterns

- Les design pattern permettent donc d'apporter une solution éprouvée, efficace et réutilisable et forcent les concepteurs à proposer une solution modulaire.

- Ils permettent un gain de temps considérable ; Par exemple, il n'est pas nécessaire de gaspiller son énergie à savoir si telle ou telle solution est la meilleure alors que le problème a déjà été résolu.

- Il en résulte une meilleure compréhension du design complet du problème, rendant les discussions entre concepteurs plus aisées.

1.3. Inconvénients des design patterns

Il est cependant vrai que tout n'est pas facile pour autant : un pattern de son choix ne peut en effet être implanté aveuglément dans son design. Les patterns sont pour la plupart volontairement abstraits, donc très peu formalisés, ce qui leur permet par ailleurs d'être au maximum ouvert à tous les types de problèmes.

Un effort supplémentaire est donc nécessaire pour intégrer au mieux notre pattern au problème rencontré. Ce qui suppose une très bonne connaissance de ce qui est utilisé.

Les problèmes rencontrés sont très hétéroclites, au même titre que les design pattern. Ils peuvent être très formalisables, alors que d'autres le sont beaucoup moins. Inversement, des formalismes sont adaptés à des types de pattern correspondants.

2

2.1. Objectifs du projet

L'analyse des différentes caractéristiques (qualités et défauts) des design pattern met en évidence un net manque de formalisation. Il convient néanmoins de veiller à ne pas perdre l'essence du pattern afin de ne pas dévier vers des solutions non adaptables.

Ce projet s'attachera donc à proposer des solutions concrètes pour permettre aux concepteurs d'utiliser au mieux les capacités des patterns tout en limitant le temps d'intégration.

Objectifs :

- Apporter une amorce de solution aux défauts rencontrés dans l'utilisation des design patterns à travers un formalisme qui décrit aussi bien la structure que le comportement.
- Trouver une bonne formalisation tout en restant le plus proche possible de la forme originale du pattern

2.2. Classifications

« Design Patterns, Elements of reusable Object-Oriented software » est la référence en ce qui concerne les software patterns, il contient un catalogue détaillé de 23 solutions différentes groupées en 3 catégories (design créationnel, structurel et comportemental).

Nous pouvons classifier les patterns selon 2 échelles :

1. **Leurs types :**
   - Patterns créationnel
   - Patterns structurels
   - Patterns comportementaux

2. **Leurs qualités d'interprétations :**
   - Précise Alternative possible
   - Généralisation précise
   - Ouvert à certaines variations
   - Informel, description théologique
   - Délibérément sans détails

Dans le cadre de notre Project et pour être en accord avec les objectifs mentionnes plus haut, nous avons opter et élaborons plus sur les **patterns selon leurs types** de notre projet comme cite plus haut.

2.2.1. Patterns créationnel :

1. Factory

Factory est un des modèles de conception les plus utilisés. Ce patron permet de mettre en place une classe qui va se charger d'instancier les classes nécessaires au fonctionnement de votre application. Cela permet notamment au code source de se libérer de toute responsabilité concernant l'implémentation, la configuration et l'implémentation.

Il est particulièrement utile lorsque vous devez instancier des objets qui implémentent une même interface ou classe abstraite mais dans des contextes différents. Il peut s'agir par exemple d'une classe de gestion de bases de données qui permet de manipuler plusieurs types de bases de données.

2. Singleton :

Le modèle de conception Singleton est l'un des modèles de conception les plus simples. Ce modèle garantit que la classe n'a qu'une seule instance et fournit un point d'accès global à celle-ci. Le modèle garantit qu'un seul objet d'une classe spécifique est jamais créé. Toutes les autres références aux objets de la classe singleton font référence à la même instance sous-jacente.

Ce DP est intéressant au vu des caractéristiques humaines de notre projet, entre autres le personnel de salle et de cuisine.

Application :

Le singleton s'utilisera dans la partie « Model » dans le MVC puisqu'une seule instance est nécessaire pour les donnes à afficher.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SingetonDesignPattern
{
    public class Singleton
    {
        private static Singleton instance = null;
        private Singleton()
        {
        }
        private static object lockThis = new object();

        public static Singleton GetInstance
        {
            get
            {
                lock (lockThis)
                {
                    if (instance == null)
                        instance = new Singleton();

                    return instance;
                }
            }
        }
    }
}
```

Figure 1: Initialisation thread-safe (verrouillage à double vérification) du modèle singleton

2.2.2. Patterns structurels :

1. Decorator

Le Décorateur est un patron de conception structurel qui permet d'ajouter dynamiquement de nouveaux comportements à des objets en les plaçant à l'intérieur d'objets spéciaux appelés emballeurs (wrappers). En utilisant ce modèle, vous pouvez étendre le comportement d'un seul objet, plutôt que d'étendre le comportement d'une classe dans son ensemble (en utilisant par exemple des sous-classes).

2. Façade

La Façade est un patron de conception structurel qui fournit une interface simplifiée

(mais limitée) à un système complexe de classes, bibliothèques ou Framework (améliorer l'utilisation d'un sous-système).

Application :

La façade s'utilisera pour interagir d'une façon moins complexe avec la classe de la salle de restauration et de la cuisine et simplifier l'utilisation de leurs méthodes.

3. MVC (Model – View – Controller)

Le modèle-vue-contrôleur (MVC) est un modèle architectural qui sépare une application en trois composants logiques principaux : le modèle, la vue et le contrôleur. Chacun de ces composants est conçu pour gérer des aspects de développement spécifiques d'une application. MVC est l'un des Framework de développement Web standard les plus fréquemment utilisés pour créer des projets évolutifs et extensibles.

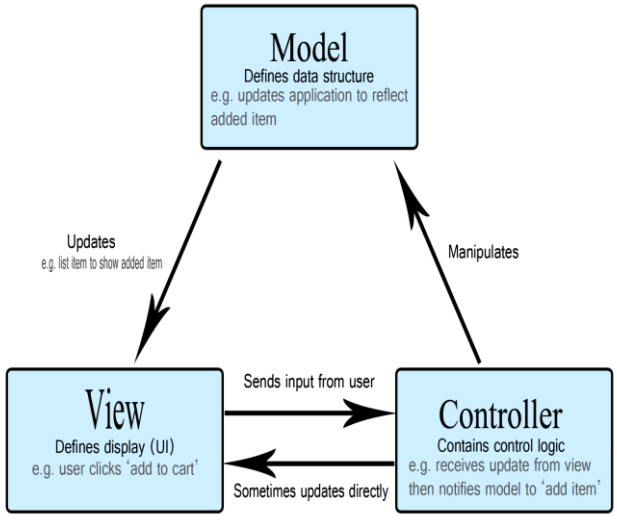


Figure 2 : Modèle MVC

2.2.3. Patterns comportementaux

3. Strategy

La Stratégie est un patron de conception comportemental qui transforme un ensemble de comportements en objets, et les rend interchangeables à l'intérieur de l'objet du contexte original et permettre à chaque algorithme de varier indépendamment du client qui l'utilise.

Application :

Le strategy s'utilisera dans le menu, puisqu'il y a plusieurs des recettes pour les entrées, les plats et les desserts, le strategy permettra choisir les recettes en fonctionne des ingrédients et d'autres facteurs.

4. Observer

Lorsqu'un événement se produit dans une application, il peut être nécessaire de produire une ou plusieurs actions en cascade. La première solution est d'écrire cette suite d'action dans une méthode. Le problème : cette classe devient donc dépendante de toutes les autres qui exécutent ces actions.

Observer permet de séparer les différentes actions, de limiter les dépendances inter-objets.

Application :

L'observer s'utilisera dans la partie « vue » dans le MVC et chaque fois qu'une modification est présentée sera notifiée et une mise à jour sera effectuée dans le système.

3

CONCLUSION

Lors de l'élaboration de notre projet, nous avons pu apprécier le formidable engouement suscité par les designs patterns. Nous constatons également la multitude et la diversité des patterns proposés. Est-ce pour autant une solution miracle a tous les problèmes de conception ? Ou plutôt une ressource supplémentaire d'information ? Une chose apparaît certaine, les patterns permettent d'unifier la discussion entre concepteurs ; A ce titre, il est opportun de ne pas les négliger.

Nous avons par ailleurs pu distinguer deux grands types de patterns ; ceux plutôt comportementaux, et ceux structurels. Chaque pattern possédant des exigences diverses en matière de modélisation, nous avons donc opté pour une solution mixte, mettant en jeu aussi bien une description des comportements qu'une modélisation de la structure de chaque acteur.

Cette théorie semble malgré tout inutile si elle ne permet pas des applications concrètes ; par exemple dans l'industrie de la production logicielle, un outil de développement capable d'intégrer les designs patterns, leurs structures, leurs comportements et leurs caractéristiques. Un outil qui serait non seulement capable d'intégrer une solution dans une conception existante, mais également de reconnaître un pattern, de le combiner avec d'autres et de le synthétiser sans limite.

4

REFERENCES

1. (27 12 2021 p.) https://tkt.paris/comment-mettre-en-place-un-design-pattern/

2. A. Le Guennec, G. S. *Design Patterns Application in UML.*

3. E.Gama, R. H. *Design Patterns : Elements of Object-Oriented Software.*

4. Schmidt., M. F. *Object-Oriented Application Frameworks.*

5. systémique, L. -L. (27 12 2021 p.) *https://www.google.com/*. https://www.google.com/: https://os.zhdk.cloud.switch.ch/tind-tmp-epfl/21bb0b60-978e-4585-9c86-f9290b4ae03d?response-content-disposition=attachment%3B%20filename%2A%3DUTF-8%27%27IC_TECH_REPORT_200239.pdf&response-content-type=application%2Fpdf&AWSAccessKeyId=ded3589a13b4450889b2f