RAPPORT DESIGN PATTERN

Cas du MVC, signaux et slots

UCAC-ICAM

PROJET PROGRAMMATION SYSTEME

Dans le cadre de votre projet en C++ et Qt portant sur la simulation du service d'un restaurant, deux design patterns majeurs ont été appliqués : le pattern \*\*Observer\*\* via le mécanisme Qt des \*\*Signals et Slots\*\*, et le pattern \*\*MVC (Model-View-Controller)\*\*. Ce rapport détaille ces patterns, leur fonctionnement, leurs avantages, et leurs cas d'utilisation dans le projet.

**1. Design Pattern Observer via Qt Signals et Slots**

**Description**

Le **design pattern Observer** est un patron de conception comportemental permettant à un objet (sujet) de notifier un ensemble d'autres objets (observateurs) lorsque son état change. Qt implémente ce pattern à travers son système de Signals et Slots, qui est un mécanisme de communication entre objets.

- **Signal** : Un signal est une notification émise par un objet pour indiquer qu'un événement a eu lieu.

- **Slot** : Un slot est une méthode qui réagit à un signal.

- **Connexion** : Les signaux et slots sont connectés dynamiquement à l'exécution. Quand un signal est émis, tous les slots connectés sont invoqués automatiquement.

**Mise en œuvre dans le Projet**

Dans la simulation de service d'un restaurant, le pattern Observer via Signals et Slots est utilisé pour gérer les interactions dynamiques entre les composants. Par exemple :

1. **Notifications des Commandes :**

- **Signal** : Quand un serveur prend une commande, un signal comme `commandeAjoutee()` est émis.

- **Slot** : Les composants responsables de l'affichage ou du traitement des commandes, comme une file d'attente de commandes ou une interface utilisateur, sont connectés au signal pour mettre à jour leur état.

**2. Mise à Jour des Interfaces :**

- Lorsqu'un état (par ex. "Plat prêt") change, l'interface utilisateur est automatiquement mise à jour pour refléter les nouvelles informations sans intervention manuelle.

**Avantages**

- **Découplage** : Les objets émetteurs et récepteurs sont faiblement couplés. Ils n'ont pas besoin de connaître les détails les uns des autres.

- **Extensibilité** : Permet d'ajouter facilement de nouveaux observateurs sans modifier les sujets.

- **Réactivité** : Favorise une réponse dynamique aux événements, utile pour les systèmes interactifs.

**Limites**

- Une connexion excessive de signaux et slots peut rendre le débogage complexe.

- Peut augmenter le risque de fuites mémoires si les connexions ne sont pas gérées correctement.

---

**2. Design Pattern MVC (Model-View-Controller)**

**Description**

Le pattern \*\*Model-View-Controller (MVC)\*\* est une architecture logicielle qui sépare les responsabilités d'une application en trois composants principaux :

**1. Modèle (Model) :**

- Contient les données et la logique métier de l'application.

- Notifie les vues des changements via des signaux.

**2. Vue (View) :**

- Représente l'interface utilisateur et affiche les données du modèle.

- Ne contient aucune logique métier.

**3. Contrôleur (Controller) :**

- Gère les entrées utilisateur et les traduit en actions sur le modèle ou les vues.

**Mise en œuvre dans le Projet**

Dans la simulation de service d'un restaurant, le pattern MVC permet de structurer l'application pour une meilleure lisibilité et maintenabilité.

**1. Modèle (Model) :**

- Gère les entités principales comme les tables, les commandes, les plats, et leur état (par ex. "Servi", "En préparation").

- Exemple : Une classe `Commande` stocke les informations sur les commandes et notifie les vues des changements via des signals.

**2. Vue (View) :**

- Comprend les éléments graphiques pour afficher les tables, la liste des commandes, et l'état de la cuisine.

- Exemple : Une `QTableView` affiche dynamiquement les commandes issues du modèle.

**3. Contrôleur (Controller) :**

- Capture les actions utilisateur, comme l'ajout d'une commande ou le marquage d'une commande comme terminée.

- Exemple : Un bouton "Ajouter Commande" est relié à un slot dans le contrôleur qui crée une nouvelle commande dans le modèle.

**Avantages**

-**Séparation des responsabilités** : Permet de développer, tester et maintenir chaque composant indépendamment.

- **Réutilisation** : Les vues et modèles peuvent être réutilisés dans d'autres contextes.

- **Scalabilité** : Facile à étendre avec de nouvelles fonctionnalités ou interfaces utilisateur.

**Cas d'Utilisation**

**- Gestion des Commandes** : Le modèle contient les données des commandes, la vue affiche les commandes dans une liste, et le contrôleur gère les actions comme "Marquer comme servi".

- **Affichage Dynamique des Tables**: Les vues se mettent à jour automatiquement lorsque l'état des tables change (libre/occupé).

**Limites**

- Peut introduire une complexité initiale à cause du besoin de structurer strictement les composants.

- Peut être excessivement fragmenté pour des applications très simples.

---

**Conclusion**

Les design patterns Observer et MVC jouent des rôles complémentaires dans votre projet :

- Le **pattern Observer (Signals et Slots)** permet une communication efficace et réactive entre les composants, idéal pour une application interactive comme la simulation de service d'un restaurant.

- Le **pattern MVC** structure l'application, en séparant les responsabilités, ce qui rend le code plus clair, modulable et maintenable.

En combinant ces deux approches, votre projet bénéficie d'une architecture robuste, favorisant une expérience utilisateur fluide et une facilité d'évolution du logiciel.