Contrôle continu 2 - Étude et utilisation d'un patron de conception

Vous aurez, dans cet exercice, à produire à la fois des modèles et du code java (mais si peu...).

Une entreprise en cours de création cherche à construire une application de domotique qui permette à un utilisateur de contrôler différents services pour sa maison, son appartement ou un bâtiment quelconque. Par exemple, il pourra, à partir de cette application, contrôler les lumières de la maison, la chaîne HiFi, la porte du garage, l'alarme, le thermostat, etc. L'application à concevoir doit être sous la forme d'une interface graphique caractérisée par :

— un certain nombre de couples de boutons On/Off qui permettent de mettre en marche ou d'arrêter un service ;

— la possibilité pour l'utilisateur de choisir quel service associer à quel couple de boutons.

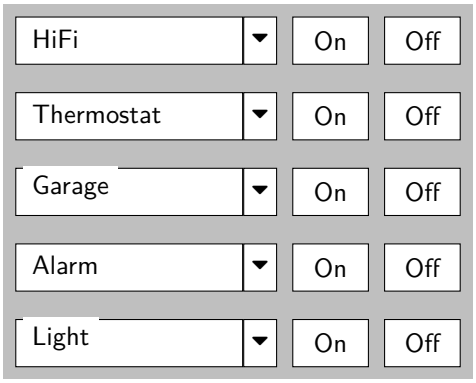


Figure 1 – Exemple d'interface utilisateur

La Figure 1 donne une idée de ce que peut être une telle interface. Son aspect rudimentaire est justifiée par le fait que l'entreprise propose un concept dans lequel elle compte vendre à la fois le logiciel permettant de faire fonctionner la télécommande et un kit matériel (avec plan) pour construire soi-même simplement la télécommande physique associée. Le client pourra ainsi y mettre le nombre de services désiré.

Ce concept est fondé sur le fait qu'il existe un certain nombre de classes permettant de contrôler différents services. Ces classes ont été développées par différents fournisseurs ; quelques-unes d'entre elles sont représentées sur la Figure 2. On suppose également que l'on pourra ajouter d'autres classes provenant d'autres fournisseurs dans le futur.

L'université Paris-Est Créteil s'est tout de suite positionnée en tant que client potentiel de cette entreprise. Elle y voit un moyen excellent et pas cher (les boitiers peuvent être construits par le personnel technique des services de logistique des sites) de faciliter le travail des gardiens dans leurs rondes quotidiennes d'ouverture et fermeture des bâtiments, ainsi que la réponse aux demandes d'accès à ces bâtiments. C'est pourquoi elle vous demande d'aider l'entreprise de domotique à élaborer son concept. Vous êtes en charge de produire les modèles qui permettront de programmer la télécommande, en donnant notamment la possibilité à son utilisateur d'associer à chaque couple de boutons On/Off le service qu'il souhaite. Vous devez pour cela répondre aux questions ci-dessous.

Question 1. Proposez une première version d'un diagramme de classes (DC) modélisant la télécommande, dans laquelle le problème de l'affectation d'un service à un couple de boutons ne sera pas traité.

Question 2. Si les classes fournies par les fournisseurs de service étaient modifiables, quelle solution simple proposeriez-vous pour résoudre le problème de l'affectation des services aux boutons ?

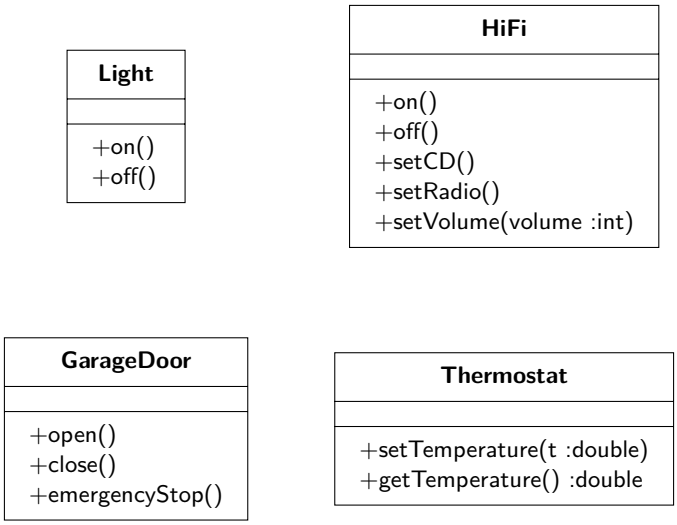


Figure 2 – Quelques classes de contrôle

Question 3. Malheureusement, les classes fournies pour les services correspondent à du code compilé et ne sont donc pas modifiables. Il nous faut de fait trouver un moyen de pouvoir assigner des actions à chaque couple de boutons de l'interface graphique. Pour cela, nous allons avoir recours au patron « Commande ». Une description de ce patron et de son fonctionnement est fournie en annexe : merci d'en prendre connaissance avant de répondre aux questions qui suivent.

a - Ajouter le patron « Commande » au DC que vous avez produit à la question en l'adaptant pour une commande qui permet d'allumer la lampe (nommée `LightOnCommand`). Qui joue le rôle d'invoker ? Qui joue celui de receiver ?

b - Écrire en Java l'interface `Command` et la classe `LightOnCommand`.

c - Comment faudrait-il étendre le diagramme de classes obtenu en pour avoir un modèle de l'ensemble de la télécommande ? On supposera le nombre de services activables depuis cette télécommande fixe et connu.

d - La classe `Thermostat` dispose des 2 méthodes `setTemperature` et `getTempérature`. Cela n'est pas directement compatible avec des boutons On/Off qui piloteraient la température dans le bâtiment. Quelle proposition formulez-vous pour faire coïncider méthodes et boutons ? Autrement dit, quelles sont les commandes (du patron « Commande ») qu'il faudrait implanter, avec quel code pour la méthode `execute()` ?

e - On souhaite pouvoir créer des séquences de commandes – un genre de « macros » - pour les associer à un couple de boutons sur la télécommande. Par exemple, la macro « Nuit de l'info » permettrait, en un seul clic, d'allumer les lumières et la chaine HiFI (on va supposer que les bâtiments en sont dotés...), de désactiver l'alarme et de basculer en mode « VPN filaire » l'ensemble des connexions réseau d'un bâtiment. Modifiez votre diagramme de classes pour pouvoir modéliser et implanter de telles macros, en utilisant un patron de conception (que vous n'oublierez pas de mentionner).

Description du patron « Commande »

Le patron « Commande » découple un objet émettant une requête de celui qui sait l'exécuter. Il permet d'encapsuler dans un objet invoker une requête sur un objet récepteur (receiver), c'est-à-dire un ou plusieurs appels de méthodes sur cet objet récepteur, et d'appeler ensuite cette requête au moment opportun. L'idée du patron est la suivante : on fait porter par un invoker une requête sur un objet dit receiver. L'invoker n'est pas obligé de connaître le receiver réel qu'il va utiliser. C'est un client qui est en charge de la création de la relation entre l'invoker et la requête ; ce n'est pas forcément lui qui l'activera via l'invoker.

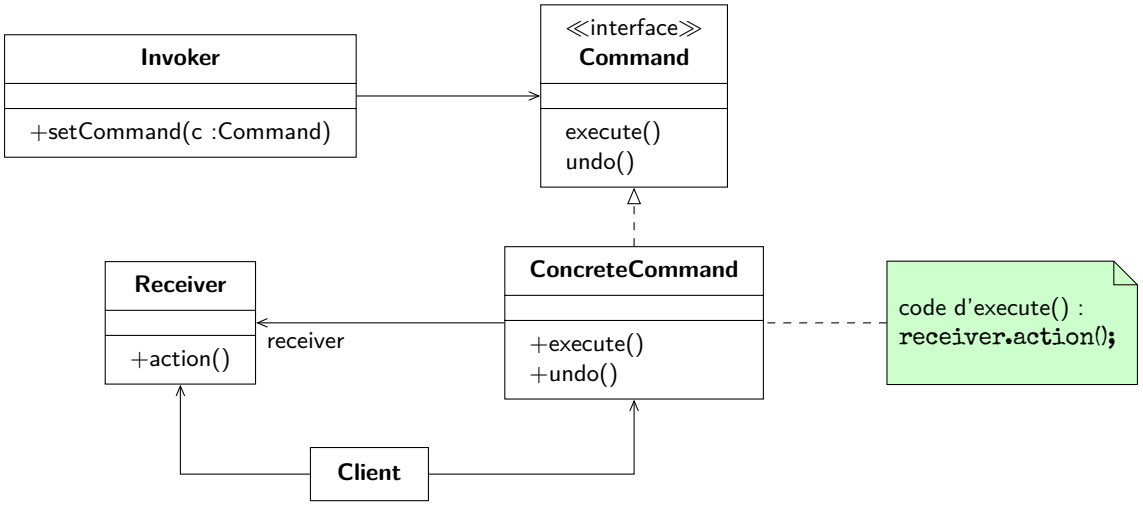


Figure 3 – Patron de conception « Commande »

Le patron fonctionne en 2 temps (cf Figure 4) :

- Temps 1 : le client crée un objet de type `Command` qui utilise une instance particulière de `Receiver`. L'objet de type `Command` contient toutes les informations pour exécuter un certain nombre d'actions sur l'instance de `Receiver`. Le client appelle alors `setCommand` sur une instance d'`Invoker` pour lier cette dernière à la commande créée.
- Temps 2 : au moment opportun, l'instance d'`Invoker` appelle la méthode execute de la commande. Ce moment peut être déterminé par l'instance d'`Invoker` elle-même ou par un appel extérieur à une méthode de cette même instance, comme par exemple l'appui sur un bouton d'une interface graphique...

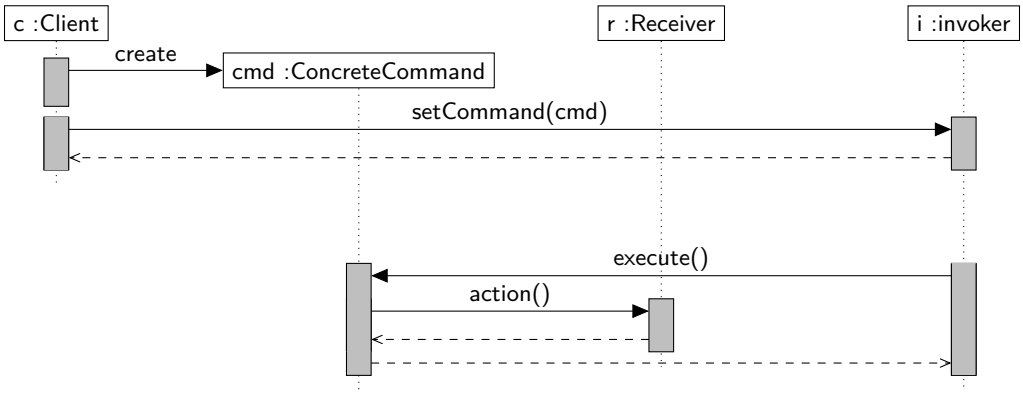
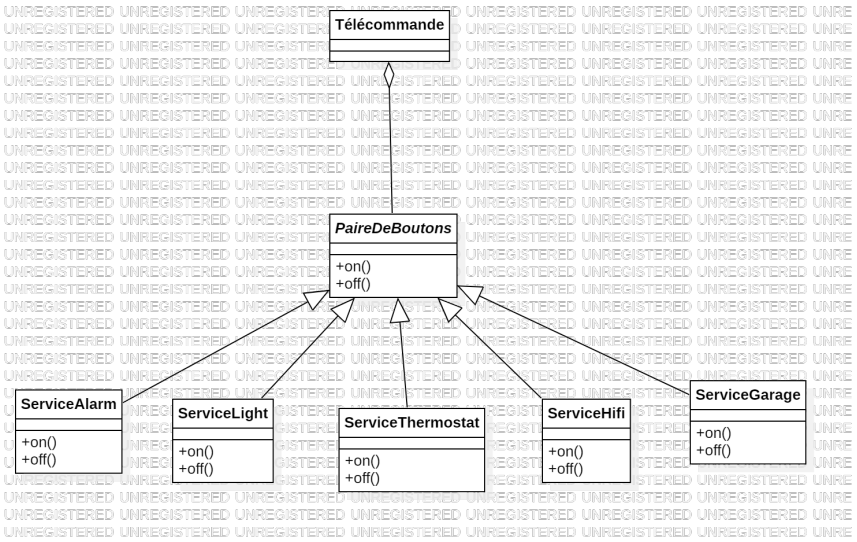


Figure 4 – Diagramme de séquence

Correction

Question 1.

La télécommande contient plusieurs paires de boutons. Chaque paire contient une méthode `on()` et une méthode `off()`. Une paire de bouton est instanciée par un service, ayant donc une méthode `on()` et une méthode `off()`.

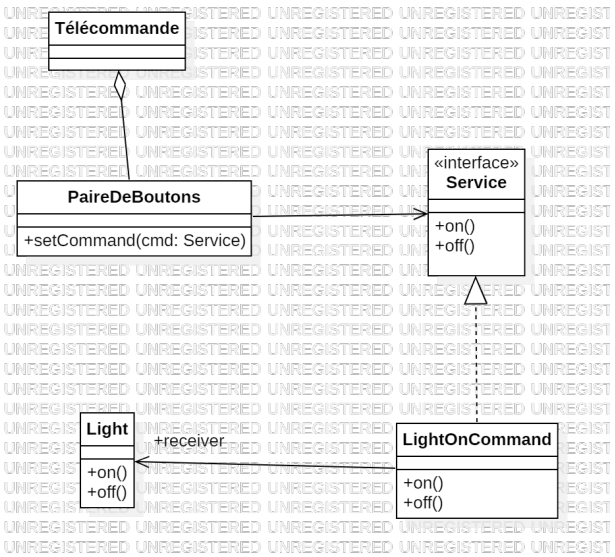


Question 2.

Si on pouvait modifier le code des classes réalisant les services souhaités, il suffirait de les dotter chacune d'un méthode `on()` et d'une méthode `off()`. On pourrait ensuite les déclarer comme sous classe de `PaireDeBoutons`, avec `extends PaireDeBouton` dans leur définition.

Question 3.

a - `PaireDeBoutons`, maintenant une classe concrète, va jouer le rôle d'"Invoker". L'interface `Service` joue le rôle de "Command", et `LightOnCommand` est naturellement une commande concrète. Light est le "receiver" dans ce modèle.

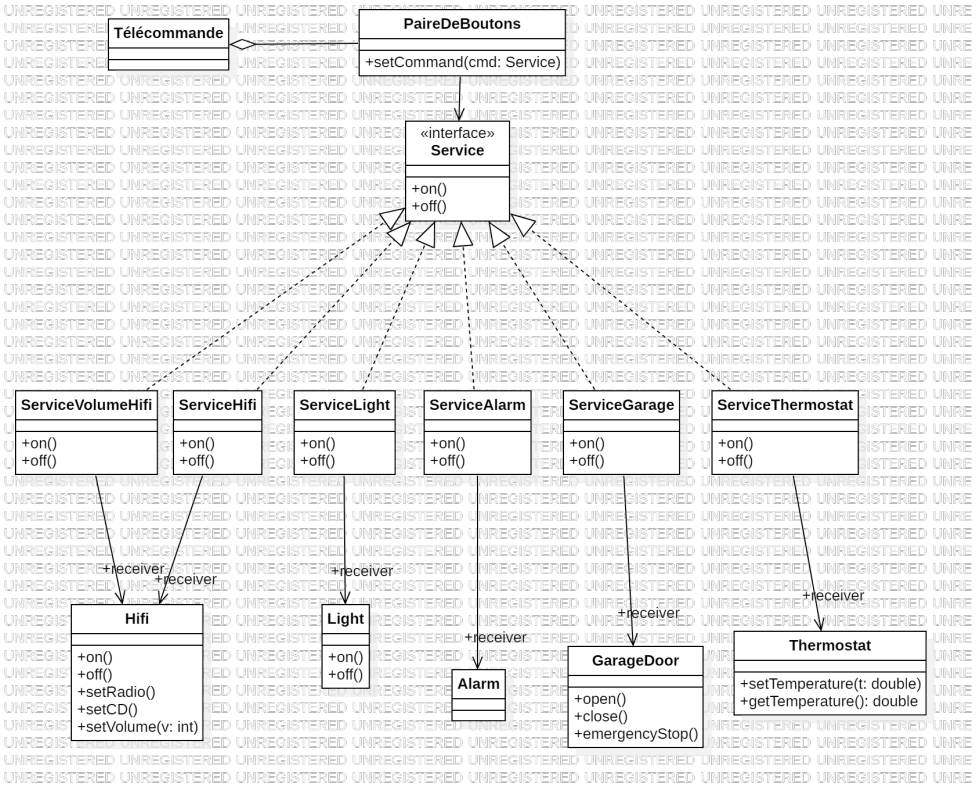


b - Dans notre modèle, l'interface commande s'appelle `Service`.

```
public interface Service{
  public void on()
  public void off()
}
public class LightOnCommand implements Service{
  private receiver Light;

  void on(){
    this.receiver.on()
  }
  void off(){
    this.receiver.off()
  }
}
```

c - On ajoute un service par comportement que l'on affecter à une paire de boutons. Chaque service va ensuite utiliser son "receiver" pour accomplir ses opérations.



Notez que plusieurs services peuvent utiliser le même "receiver". C'est le cas par exemple des deux services afférents au système HiFi, i.e. `ServiceHiFi` et `ServiceVolume`.

d - Le comportement naturel est d'incrémenter la température d'un degré Kelvin avec `on`, et de la décrémenter avec `off`.

```
public class ServiceTemperature implements Service{
  private receiver Thermostat;

  void on(){
    // le bouton on augmente la température d'un degré
    this.receiver.setTemperature(this.receiver.getTemperature() + 1)
  }
  void off(){
    // le bouton off diminue la température d'un degré
```

```
    this.receiver.setTemperature(this.receiver.getTemperature() - 1)
  }
}
```

e - On peut utiliser le patron Decorateur. On aura alors un service "neutre", qui ne réalise aucune action. Pour ajouter des comportements, on utilisera des services decorateurs. Ces services réaliseront une action spécifique (allumer la lumière), avant d'appeler la méthode idoine du service qu'ils contiennent. Par exemple, pour réaliser un bouton qui gère simultanément la Hifi (sous/hors tension) et la lumière (allumée/éteinte) on pourra utiliser une déclaration :

```
ServicePower = new ServiceNeutre()
ServicePower = new ServiceLight(ServicePower)
ServicePower = new ServiceHifi(ServicePower)
```

Avec ce nouveau service, lorsequ'on invoke la méthode `on()`, on va :

1. appeler Hifi.on()
2. appeler Light.on()
3. appeler ServiceNeutre.on(), qui ne fait rien.