Architecture Système de contrôle d'ascenseur

1/ Package ui

Ce package contient la fonction main, on lancera donc le programme à partir de lui.

Son rôle est d'instancier les objets de la simulation, Flow et Elevator. Il instancie ces objets grâce à leurs fabriques qu'il faudra instancier.

On crée l'elevator grâce à la méthode buildElevator de le classe ElevatorFactory qui prend le fichier de configuration de l'ascenseur en paramètre. Cette méthode renvoie le nombre d'étages spécifié dans le fichier.

On crée ensuite le flow avec la méthode buildFlow de FlowFactory qui prend le fichier de configuration du flow en paramètre. Cette méthode renvoie un objet d'une classe implémentant IFlow (classe implémentée dans le package flow.imp). On se servira des méthode de cette interface pour communiquer entre l'UI (le main) et le package flow. buildFlow initialise également le séquenceur (fonction create) en fonction des paramètres lus dans le fichier.

Il faudra que getMaxLevel de IFlow retourne un nombre inférieur ou égal à celui retourné par buildElevator pour pouvoir lancer la simulation.

2/ Package elevator

2.1 – Classe ElevatorFactory

buildElevator va instancier les classes nécessaires au fonctionnement de l'ascenseur dans le package elevator.imp. La classe Elevator devra posséder un objet de type ElevatorCommand implémentant IElevatorCommand ainsi qu'un champ de type IElevatorNotifier non défini au départ.

On créera également une classe implémentant Event et on l'ajoutera au Sequencer avec addProcess pour pouvoir déplacer l'ascenseur lors de la simulation.

Au final on passe l'elevator à l'ElevatorCommandFactory via la méthode setElevator.

2.2 – L'interface IElevatorCommand

Cette interface permet de communiquer entre les packages elevator et controlsystem. Elle permet au système de contrôle de donner des ordres à l'ascenseur.

Cette interface est implémentée (les méthodes sont définies) par le package elevator et controlsystem l'utilise (en appelle les métodes).

La méthode stopAtLevel dit simplement à l'ascenseur de s'arrêter au niveau suivant. On utilise donc cette commande lorsque l'ascenseur est déjà en mouvement.

La méthode move prend une direction en paramètre et ordonne à l'ascenseur de se déplacer vers le haut ou vers le bas. On n'utilise cette commande que lorsque l'ascenseur est à l'arrêt.

2.3 – Classe ElevatorCommandFactory

Cette classe n'est pas instanciée, elle ne possède que des attributs et méthodes statiques.

Elle établit la communication entre elevator et controlsystem avec les interfaces IElevatorCommand et IElevatorNotifer.

Elle contient un attribut *elevator* pour pouvoir y récupérer l'elevatorCommand pour le passer au controlsystem et pour pouvoir y définir l'elevatorNotifier passé par le controlsystem. Cet attribut est défini par l'ElevatorFactory grâce à la méthode setElevator.

- La méthode getElevatorCommand prend un objet de classe implémentant IElevatorNotifier. Elle ajoute ce notifier à son *elevator* pour notifier le controlsystem des changements d'état de l'ascenseur. Elle retourne ensuite l'elevatorCommand de son *elevator*. Cette méthode n'est appelée qu'une fois, elle permet de lier un ascenseur à un système de contrôle.

3 / Package flow

Le flow est créé par FlowFactory.

Lors de sa création il initialise le séquenceur avec les paramètres temporels origine et durée de simulation en fonction du fichier de configuration lu. Il sera ensuite déclenché par le séquenceur pour envoyer les requêtes des utilisateurs à l'interface de l'ascenseur (UIElevator). Il faut donc qu'une classe implémente Event et qu'une instanciation en soit ajoutée au séquenceur via addProcess.

3.1 – FlowFactory

Cette fabrique doit être instanciée.

- La méthode buildFlow prend en paramètre le fichier de configuration du flow contenant la liste des personnes utilisant l'ascenseur avec leur heure d'arrivée, l'étage d'arrivée, etc. Elle crée le flow et stocke donc ces utilisateurs afin de pouvoir envoyer les requêtes associés lorsque le flow sera déclenché par le séquenceur. Elle renvoie finalement un objet de classe implémentant IFlow.

3.2 – L'interface IFlow

Cette interface permet la communication entre les packages ui et flow

- La méthode getMaxLevel retourne l'étage le plus haut trouvé dans le fichier flow, est utilisé pour vérifier que les étages demandés dans le fichier ne dépassent pas le nombre d'étages de l'ascenseur.

- La méthode start lance la simulation. Lorsque la simulation se termine les résultats sont écrits dans les fichiers "completedFlow" et "measures".

3.3 – L'interface IUser

Cette interface permet la communication entre les packages flow et elevatorui.

Elle est implémentée par le package flow et utilisée (les méthodes sont appelées) par le package elevatorui.

Ses méthodes permettent à l'elevatorui de signaler au flow que ses requêtes ont bien été traitées :

- notifyCall sera appelée lorsque l'ascenseur arrivera au niveau demandé après un appel de celui-ci (lorsque l'utilisateur est à l'extérieur de l'ascenseur). Lorsque le flow recevra un notifyCall il déclenchera alors une requête de déplacement sur uielevator via la méthode createMove de UIElevatorRequestFactory.

- notifyMove est appelée lorsque l'ascenseur arrive au niveau demandé après une demande de déplacement (lorsque l'utilisateur est à l'intérieur de l'ascenseur).

4 / Package uielevator

Ce package permet de passer des requêtes du flow vers le controlsystem et de prévenir le flow du bon traitement de ces requêtes via une notification du controlsystem.

4.1 – UIElevatorRequestFactory

Cette fabrique doit être instanciée (par flow). Son instanciation entraîne la création de son attribut de type *UIElevator* auquel elle passera les requêtes du flow. Elle permet la communication entre flow et uielevator.

Ses méthodes sont appelées par le flow lorsque le séquenceur le déclenche et qu'il y'a une requête à envoyer (appel ou déplacement d'ascenseur).

- createCall prend en arguments l'étage à partir duquel l'ascenseur est appelé, une direction indiquée par l'utilisateur (qui indique s'il voudra monter ou descendre) et l'utilisateur lui-même (objet de classe implémentant IUser) pour pouvoir notifier de la fin du traitement de la requête (arrivée de l'ascenseur et ouverture des portes signifiant que l'utilisateur peut entrer dans l'ascenseur).

- createMove prend en arguments l'étage auquel l'utilisateur veut se déplacer et l'utilisateur lui-même (objet de classe implémentant IUser) pour pouvoir notifier de la fin du traitement de la requête (ascenseur arrivé à l'étage voulu et portes ouvertes, l'utilisateur peut sortir).

Ces méthodes instancient un objet implémentant l'interface IUIrequest. Dans cet objet seront stockées toutes les informations à propos de la requête : étage demandé, direction (nulle si la requête est une demande de déplacement) et utilisateur (IUser). Une fois l'objet créé il sera passé à CSRequestFactory du package controlsystem via la méthode stopRequest pour le transformer en requête de controlsystem et pour l'ajouter à ce dernier.

4.2 – L'interface IUIRequest

Cette interface permet la communication entre uielevator et controlsystem.

Elle est implémentée par le package elevatorui et utilisée (ses méthodes sont appelées) par le package controlsystem.

- Sa seule méthode notifyStop est appelée par le controlsystem pour signaler que l'ascenseur est arrivé à un des étages demandés et donc que la ou les requêtes associées ont été traitées. Elle prend en argument la date (temps de simulation) à laquelle l'ascenseur est arrivé et a ouvert ses portes.

La classe qui implémente cette interface devra, lorsqu'elle reçoit un notifyStop, faire remonter cette notification au flow grâce à l'attribut *user* qu'elle a précédemment stockée.

Cette notification est remontée grâce aux méthodes notifyCall ou notifyMove (selon le type de requête) de l'interface IUser.

5 / Package controlsystem

Ce package reçoit des requêtes de uielevator, les stocke, optimise les déplacements de l'ascenseur en fonction de ces requêtes (c'est-à-dire détermine dans quel ordre ces requêtes doivent être traitées) et commande l'ascenseur (mise en mouvement/arrêt). Lorsque l'ascenseur arrive à un des étages demandés par une requête le système de contrôle en notifie uielevator via l'interface IUIRequest.

5.1 – L'interface IElevatorNotifier

Cette interface permet la communication entre les packages controlsystem et elevator.

Elle est implémentée par le package controlsystem et utilisée (ses méthodes sont appelées) par le package elevator. Elle est donnée à elevator en argument de la méthode getElevator de ElevatorCommandFactory.

- La méthode notifyLevel est appelée par elevator à chaque fois que l'ascenseur change d'étage. Elle prend en argument le niveau auquel l'ascenseur est arrivé. Lorsque le controlsystem reçoit cette notification il va vérifier si l'ascenseur doit s'arrêter au prochain étage ou non : si oui il envoie une commande stopAtLevel via l'interface IElevatorCommand , sinon il attend le prochain notifyLevel.

- La méthode notifyState est appelée par elevator à chaque changement d'état de l'ascenseur. Les états possibles pour l'ascenseur sont définis par le type ElevatorState dans le package commontypes et sont les suivants : "ascenseur arrêté et portes ouvertes", "ascenseur arrêté et portes fermées" et "ascenseur en mouvement". Lorsque que les portes s'ouvrent controlsystem prévient le uielevator qu'une de ses requêtes a été traitée. Lorsqu'elles se ferment il peut alors demander à l'ascenseur de se remettre en mouvement. Si l'ascenseur est en mouvement il peut lui demander de s'arrêter au prochain niveau par lequel l'ascenseur passera.

5.2 – CSRequestFactory

Cette classe permet la communication entre uielevator et controlsystem.

Cette fabrique doit être instanciée et possède en attribut un objet du package controlsystem pour pouvoir lui passer les requêtes.

- Son unique méthode stopRequest est appelée par l'uielevator. Elle crée une requête pour le controlsystem à partir d'un étage, d'une direction et d'un objet d'une classe implémentant IUIRequest qui servira à notifier de la fin du traitement de la requête. La requête instanciée par cette méthode est ajoutée au controlsystem qui va la ranger avec les requêtes en cours de traitement selon l'ordre de priorité calculé.