POLYTECH ANGERS

TP 5A - Génie Logiciel
Patrons de conception: State

Nicolas Delanoue

(1 séance de 1h20)

Objectif : Se familiariser avec le patron de conception (Design Pattern) *State*. Améliorer un code existant (notion de *refactorying*).

1 Cahier des charges et prototype

L'objectif est de concevoir un programme modélisant, de manière très simplifiée, l'utilisation d'un téléphone.

Il s'agit en particulier de gérer le comportement de certaines méthodes en fonction de l'état dans lequel se trouve un téléphone.

Pour simplifier l'exercice, on considérera 3 états :

— `DOWN`, état raccroché,
— `UP`, état décroché,
— `TALKING`, état en conversation (talking).

ainsi que 3 actions :

— décrocher (pick up),
— raccrocher (put down),
— converser (talk).

On ignore, pour des raisons de simplicité, les actions telles que la composition d'un numéro. Le déroulement de ces actions dépend de l'état et certains actions peuvent modifier l'état. Par exemple, on peut converser que si la méthode `decrocher()` a été préalablement invoqué, et que l'état du téléphone est donc décroché.

Le projet `TPState_eclipse_sujet` implémente le comportement des différentes méthodes (actions sur le téléphone) que l'on peut invoquer (voir illustration 1 pour la modélisation UML), au moyen d'une interface graphique (`Gui`) fournie.

2 Travail à réaliser

A partir du code fourni, proposez une machine à états décrivant le comportement de la classe `Phone`.

Une limitation du programme fourni est la gestion des comportements en fonction de l'état du téléphone. Cette gestion est implémentée au moyen d'une série de tests effectués sur l'état du téléphone. Par ailleurs, pour une méthode donnée de la classe `Phone`, les différents comportements sont implémentés au sein d'une même méthode de la classe `Phone`. Ceci n'apparaît pas forcément critique dans ce cas trivial, mais pourrait le devenir si la complexité des différents comportements venait à croître (i.e. nombre de lignes de code associé à un état).

Pour résoudre ce problème, on souhaite modifier ce code en intégrant le design pattern `State` (voir figure 1) : à chaque état sera associé une classe implémentant le comportement des méthodes de la classe `Phone` (transition). Chacune de ces classes aura également la possibilité de modifier l'état actif.

Pour résumer, le travail d'intégration du `State` à réaliser consiste à :

— Proposer un diagramme de classe UML adapté à notre contexte,

— ajuster le code de la solution précédent en conséquence, afin d'obtenir le même comportement.

Remarque : la classe `Gui` n'aura pas être modifiée.

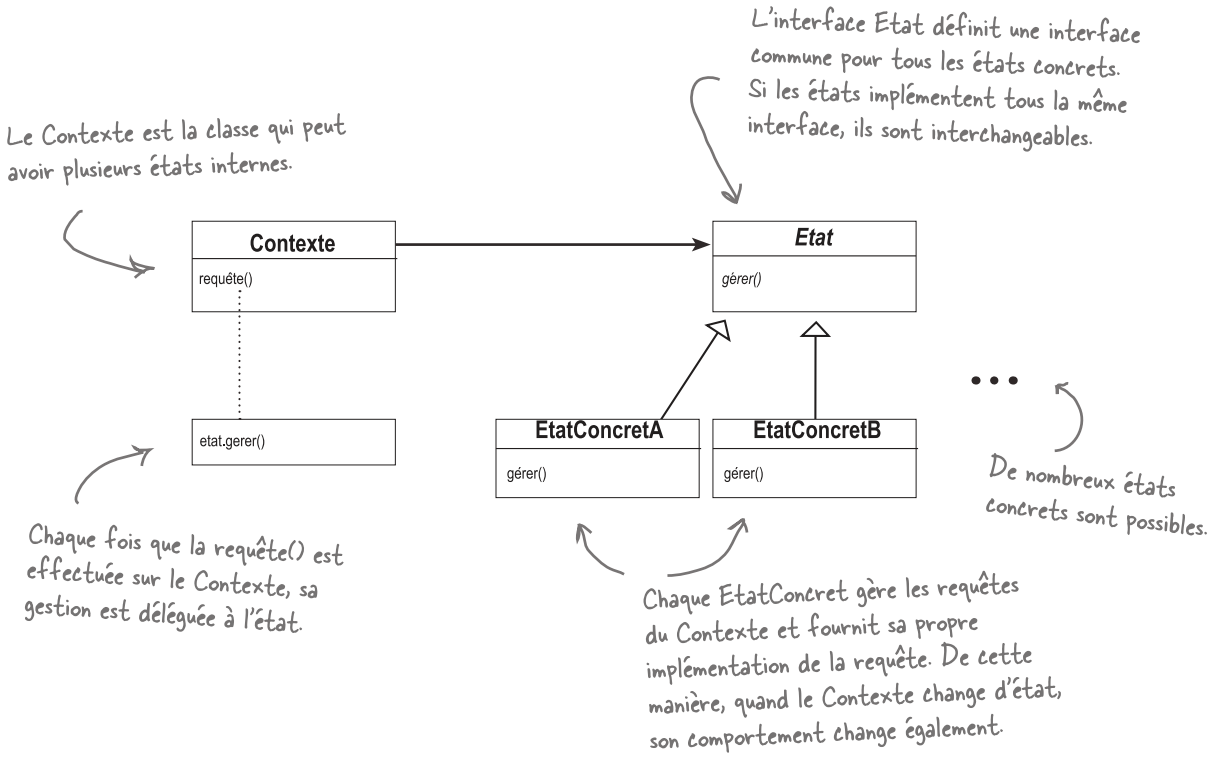

FIGURE 1 – Modèle UML du patron de conception *State* issue du livre Design patterns : tête la première, Freeman, E. and Freeman, E. and Baland, M.C. and Sierra, K., O'Reilly.