Travaux Dirigés No5 : Test MBT

Frédéric Peschanski

22 février 2017

Dans ce TD nous nous mettons en pratique le test basé sur les modèles ou *Model Based Testing* (MBT) sur un modèle simplifié de programme de contrôle pour ascenseur (cf. annexe).

Remarque : la spécification du service Lift dépend d'un service Commands qui enregistre les commandes utilisateurs. Ce service n'est pas spécifié explicitement (la spécification est donc partielle).

Exercice 1 : Couverture des préconditions

Lister l'ensemble des objectifs de test à atteindre pour le critère : **couverture des préconditions** Indiquer pour chaque objectif si il est atteignable ou dans le cas contraire justifier. Proposer des cas de test pour une sélection d'objectifs atteignables. Déterminer le taux de couverture obtenu ainsi que le taux de couverture prévu.

Exercice 2 : Couverture en termes d'automates

Question 1 : couverture des transitions

Lister les objectifs de tests pour le critère de couverture des **transitions**.

Question 2 : couverture d'état(s) remarquable(s)

Identifier un **état remarquable** de service Lift et répondez à l'objectif de test correspondant.

Question 3 : couverture de paires de transitions

Lister les objectifs de test correspondant aux **paires de transitions** possibles du service.

Question 4 : couverture de scénario(s) utilisateur

Identifier un **scénario utilisateur** complexe et proposer un objectif de test associé. Si ce dernier est atteignable proposer un cas de test.

Exercice 3 : Couverture des données

Question :

Pour l'objectif de test — franchissement de la transition selectLevel — proposer :

- un cas de test **dans les bornes**
- trois cas de test **aux bornes**
- deux cas de test **hors borne**

Annexe : spécification du contrôleur d'ascenseur

```
types   TLiftStatus = enum {   IDLE          // En attente
                               STANDBY_UP    // Prêt déplacement vers le haut
                               STANDBY_DOWN  // Prêt déplacement vers le bas
                               MOVING_UP     // Déplacement vers le haut
                               MOVING_DOWN   // Déplacement vers le bas
                               STOP_UP       // Din déplacement vers le haut
                               STOP_DOWN };  // Fin déplacement vers le bas

        TDoorStatus = enum {   OPENED        // Porte ouverte
                               CLOSED        // Porte fermée
                               OPENING       // Porte en cours d'ouverturte
                               CLOSING }     // Porte en cours de fermeture

Service : Lift
Use : Commands
Observators :
    const minLevel : [Lift] → int
    const maxLevel : [Lift] → int
    level : [Lift] → int // étage courant
    doorStatus : [Lift] → TDoorStatus
    liftStatus : [Lift] → TLiftStatus
    commands : [Lift] → Commands // Commandes de contrôle
Constructors :
    init : int × int → [Lift]
        pre init(min,max) require 0 ≤ min < max
Operators :
    beginMoveUp : [Lift] → [Lift]
        pre beginMoveUp(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = STANDBY_UP
                                   ∧ level(L) < Commands::getNextUpCommand(commands(L))
    stepMoveUp : [Lift] → [Lift]
        pre stepMoveUp(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = MOVING_UP
                                  ∧ level(L) < Commands::getNextUpCommand(commands(L))
    endMoveUp : [Lift] → [Lift]
        pre endMoveUp(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = MOVING_UP
                                 ∧ level(L) = Commands::getNextUpCommand(commands(L))
    beginMoveDown : [Lift] → [Lift]
        pre beginMoveDown(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = STANDBY_DOWN
                                     ∧ level(L) > Commands::getNextDownCommand(commands(L))
    stepMoveDown : [Lift] → [Lift]
        pre stepMoveUp(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = MOVING_DOWN
                                  ∧ getLevel(L) > Commands::getNextDownCommand(commands(L))
    endMoveDown : [Lift] → [Lift]
        pre endMoveDown(L) require   doorStatus(L) = CLOSED ∧ liftStatus(L) = MOVING_DOWN
                                   ∧ getLevel(L) = Commands::getNextDownCommand(commands(L))
    openDoor : [Lift] → [Lift]
        pre openDoor(L) require doorStatus(L) = CLOSED ∧ liftStatus(L) ∈ { IDLE, STOP_UP, STOP_DOWN }
    closeDoor : [Lift] → [Lift]
        pre closeDoor(L) require doorStatus(L) = OPENED ∧ liftStatus(L) ∈ {IDLE, STANDBY_UP, STANDBY_DOWN }
    doorAck : [Lift] → [Lift]
        pre doorAck(L) require doorStatus(L) ∈ { OPENING, CLOSING }
    selectLevel : [Lift] × int → [Lift]
        pre selectLevel(L,n) require   liftStatus(L) ∈ { IDLE, STANDBY_UP, STANDBY_DOWN }
                                     ∧ minLevel(L) ≤ n ≤ maxLevel(L)
```

⇒ suite page suivante ...

Annexe : spécification du contrôleur d'ascenseur (observations)

Observations :
[invariants]
- $\mathsf{minLevel(L)} \leq \mathsf{level(L)} \leq \mathsf{maxLevel(L)}$
- $\mathsf{liftStatus(L)} \in \{\mathsf{MOVING\_UP}, \mathsf{MOVING\_DOWN}\}$
  $\Longrightarrow$ $\mathsf{doorStatus(L)} = \mathsf{CLOSED}$
- $\mathsf{liftStatus(L)} = \mathsf{IDLE}$
  $\Longrightarrow$ $\mathsf{doorStatus(L)} = \mathsf{OPENED}$

[init]
$\mathsf{minLevel(init(min,max))} = \mathsf{min}$
$\mathsf{maxLevel(init(min,max))} = \mathsf{max}$
$\mathsf{level(init(min,max))} = \mathsf{min}$
$\mathsf{liftStatus(init(min,max))} = \mathsf{IDLE}$
$\mathsf{doorStatus(init(min,max))} = \mathsf{OPENED}$
$\mathsf{commands(init(min,max))} = \mathsf{Commands::init()}$

[beginMoveUp]
$\mathsf{liftStatus(beginMoveUp(L))} = \mathsf{MOVING\_UP}$

[stepMoveUp]
$\mathsf{level(stepMoveUp(L))} = \mathsf{level(L)} + 1$

[endMoveUp]
$\mathsf{liftStatus(endMoveUp(L))} = \mathsf{STOP\_UP}$
$\mathsf{commands(endMoveUp(L))} = \mathsf{Commands::endUpCommand()}$

[beginMoveDown]
$\mathsf{liftStatus(beginMoveDown(L))} = \mathsf{MOVING\_DOWN}$

[stepMoveDown]
$\mathsf{level(stepMoveDown(L))} = \mathsf{level(L)} - 1$

[endMoveDown]
$\mathsf{liftStatus(endMoveDown(L))} = \mathsf{STOP\_DOWN}$
$\mathsf{commands(endMoveDown(L))} = \mathsf{Commands::endDownCommand()}$

[openDoor]
$\mathsf{doorStatus(openDoor(L))} = \mathsf{OPENING}$

[closeDoor]
$\mathsf{doorStatus(closeDoor(L))} = \mathsf{CLOSING}$

[doorAck]
- $\mathsf{doorStatus(L)} = \mathsf{OPENING} \Longrightarrow \mathsf{doorStatus(doorAck(L))} = \mathsf{OPENED}$
- $\mathsf{doorStatus(L)} = \mathsf{CLOSING} \Longrightarrow \mathsf{doorStatus(doorAck(L))} = \mathsf{CLOSED}$
- $\mathsf{liftStatus(L)} = \mathsf{IDLE} \wedge \mathsf{Commands::getNbDownCommands(commands(doorAck(L)))} > 0$
  $\Longrightarrow$ $\mathsf{liftStatus(doorAck(L))} = \mathsf{STANDBY\_DOWN}$
- $\mathsf{liftStatus(L)} = \mathsf{IDLE} \wedge \mathsf{Commands::getNbUpCommands(commands(doorAck(L)))} > 0$
  $\Longrightarrow$ $\mathsf{liftStatus(doorAck(L))} = \mathsf{STANDBY\_UP}$
- $\mathsf{liftStatus(L)} = \mathsf{IDLE} \wedge \mathsf{Commands::getNbDownCommands(commands(doorAck(L)))} = 0$
  $\wedge\ \mathsf{Commands::getNbUpCommands(commands(doorAck(L)))} = 0$
  $\Longrightarrow$ $\mathsf{liftStatus(doorAck(L))} = \mathsf{IDLE}$
- $\mathsf{liftStatus(L)} \neq \mathsf{IDLE} \Longrightarrow \mathsf{liftStatus(doorAck(L))} = \mathsf{IDLE}$

[selectLevel]
$\mathsf{level} = \mathsf{level(L)} \Longrightarrow \mathsf{commands(selectLevel(L,level))} = \mathsf{commands(L)}$
$\mathsf{level} > \mathsf{level(L)} \Longrightarrow \mathsf{commands(selectLevel(L,level))} = \mathsf{Commands::addUpCommand(commands(L),level)}$
$\mathsf{level} < \mathsf{level(L)} \Longrightarrow \mathsf{commands(selectLevel(L,level))} = \mathsf{Commands::addDownCommand(commands(L),level)}$