Devoir sur Table
Contrôle des accès *

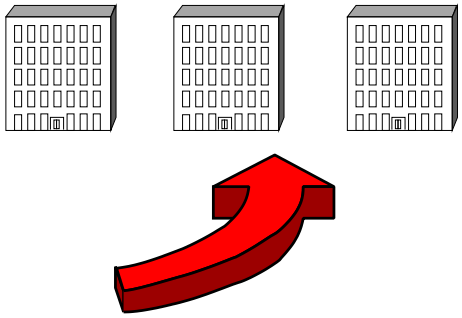
Frédéric Peschanski

Mars 2008

Durée : 2 heures

Document autorisé : polycopié de cours

Description du problème



Dans ce devoir nous étudions les spécifications d'un système de contrôle des accès à des bâtiments sécurisés.

Informellement, le système contrôle les accès de personnes à des bâtiments par l'intermédiaire de portiques automatiques. Chaque personne possède une carte d'accès électronique avec un identifiant unique. Pour entrer dans un bâtiment une personne doit introduire sa carte d'accès dans un portique. Ce dernier s'ouvrira uniquement si la personne possède les droits d'accès au bâtiment.

Les autorisations d'accès se font par zones (de bâtiment(s)). Ainsi, une personne autorisée pour la zone A pourra effectivement entrer et/ou sortir dans tout bâtiment de la zone A.

A la suite d'une analyse préliminaire, il est proposé de spécifier trois modules du systèmes :

1. un module de gestion de base de données des personnes et de leur(s) autorisation(s)
2. un module de contrôle de fonctionnement local de portique
3. un module de contrôle global des accès centralisé

*Cette étude de cas est inspirée d'une étude similaire dans le cadre de la méthode de développement B, cf. `http://www-lsr.imag.fr/B/Documents/ClearSy-CaseStudies/PORTES/SourcesFR/main.html`

1 Exercice 1 : Gestion de la base de données des personnes et autorisations

Les données associées aux personnes sont spécifiées de la façon suivante :

Service : Person
Types : String, Set<T>, int
Observators :
    const getName : [Person] → String
    const getId : [Person] → int
    getAccesses : [Person] → Set<String>
    hasAccess : [Person] * String → boolean
    equals : [Person] * Person → boolean
Constructors :
    init : String * int → [Person]
        precondition : init(name,id) **require** name≠null ∧ id > 0
Operators :
    grant : [Person] * String → [Person]
        precondition : grant(P,area) **require** area ∉ getAccesses(P)
    deny : [Person] * String → [Person]
        precondition : deny(P,area) **require** s ∈ getAccesses(P)

Question 1.1

Ajouter au service Person la section des **Observations**. On considérera que deux personnes sont « égales » si leurs identifiant numérique est le même.

Question 1.2

Pour garantir globalement l'unicité des identifiant numériques de personnes, un service dédié est spécifié ci-dessous.

Service : IdManager
Types : int, boolean, Set<T>
Observators :
    isKnown : [IdManager] * int → boolean
    getAllIds : [IdManager] → Set<int>
    getLast : [IdManager] → int
Constructors :
    init : → [IdManager]
Operators :
    generate : [IdManager] → [IdManager]
Observations :
[invariants]
    isKnown(IM,id) = id ∈ getAllIds(IM)
    ∀ id ∈ getAllIds(IM), id > 0
    getAllIds(IM)=null ⟹ getLast(IM)=null
    getAllIds(IM) ≠ null ⟹ getLast(IM) ∈ getAllIds(IM)
[init]
    getAllIds(init()) = ∅
    getLast(init()) = null
[generate]
    size(getAllIds(generate(IM))) = size(getAllIds(IM))+1
    getLast(generate(IM)) ∈ getAllIds(IM)

Selon-vous la spécification proposée est-elle cohérente ? Dans le cas contraire, corrigez la. La spécification est-elle complète ? Justifiez votre réponse.

Question 1.3

On propose de réaliser la spécification du service PersonDB de gestion de la base de données des personnes et des autorisations.

Les observateurs à spécifier sont les suivants :
- isMember permet de tester si une personne est enregistrée dans la base, à partir de son identifiant numérique
- getNbMembers retourne le nombre de personnes enregistrées dans la base
- findByName permet de retrouver des personnes dans la base, à partir de leur nom
- findByArea idem mais à partir d'une zone d'accès
- findById idem mais à partir d'un identifiant numérique (**attention** : une seule personne possible)
- getAreas retourne l'ensemble des zones d'accès enregistrées (chaînes de caractères)
- getIdManager permet d'accéder au service de gestion des identifiant

Les opérateurs, en plus du constructeur sans argument, sont les suivants :
- addPerson permet d'ajouter une personne dans la base
- removePerson effectue l'opération inverse

On adoptera une approche défensive pour la spécification. Si nécessaire (pour par exemple assurer la complétude de la spécification), on pourra ajouter des observateurs et/ou opérateurs.

Exercice 2 : Contrôle des accès

Les services suivant décrivent les cartes d'accès associées aux personnes et les portiques (*turnstile*) déployés dans les bâtiments.

Service : AccessCard
Types : int
Observators :
    const getId : [AccessCard] → int
Constructors :
    init : int → [AccessCard]
Observations :
[init]
    getId(init(id)) = id

Service : TurnStile
Types : boolean, enum TSMode = { ENTER, EXIT }
Require : AccessCard , boolean
Observators :
    cardInserted : [TurnStile] → boolean
    getCard : [TurnStile] → AccessCard
        precondition : getCard(TS) **require** cardInserted(TS)
    isOpened : [TurnStile] → boolean
    getMode : [TurnStile] → TSMode
        precondition : getMode(TS) **require** cardInserted(TS)
Constructors :
    init : → [TurnStile]
Operators :
    insertIn : [TurnStile] * AccessCard → [TurnStile]
        precondition : insertIn(TS,card) **require** ¬ cardInserted(TS) ∧ ¬isOpened(TS)
    insertOut : [TurnStile] * AccessCard → [TurnStile]
        precondition : insertOut(TS,card) **require** ¬cardInserted(TS) ∧ ¬ isOpened(TS)
    eject : [TurnStile] → [TurnStile]
        precondition : eject(TS) **require** cardInserted(TS)
    open : [TurnStile] → [TurnStile]
        precondition : open(TS) **require** ¬isOpened(TS) ∧ ¬cardInserted(TS)
    close : [TurnStile] → [TurnStile]
        precondition : close(TS) **require** isOpened(TS)

```
    timeout : [TurnStile] → [TurnStile]
Observations :
[invariants]
    cardInserted(TS) = getCard(TS)≠null
[init]
    getCard(init()) = null
    isOpened(init())= false
[insertIn]
    getCard(insertIn(TS,card)) = card
    getMode(insertIn(TS,card)) = TSMode.ENTER
    isOpened(insertIn(TS,card)) = false
[insertOut]
    getCard(insertOut(TS,card)) = card
    getMode(insertOut(TS,card)) = TSMode.EXIT
    isOpened(insertOut(TS,card)) = false
[eject]
    getCard(eject(TS,card)) = null
    isOpened(eject(TS,card)) = isOpened(TS)
[open]
    getCard(open(TS)) = null
    isOpened(open(TS)) = true
[close]
    getCard(close(TS)) = null
    isOpened(close(TS)) = false
[timeout]
    getCard(timeout(TS)) = null
    isOpened(timeout(TS)) = false
```

Question 2.1

Expliquez le mode de fonctionnement d'un portique (*turnstile*). Dessinez un diagramme d'états/transitions avec tous les états intéressants, pas seulement le TSMode (diagramme UML d'état ou automate fini).

Question 2.2

L'objectif de cet exercice est de compléter la spécification du contrôleur central des accès. La spécification partielle est ci-dessous :

```
Service : AccessControl
Require : PersonDB, TurnStile
Types : int, Set<T>, String
Observators :
    getDB : [AccessControl] → PersonDB
    getSiteMap : [AccessControl] → Map<String,Set<TurnStile>> // tous les portiques
    getDoorsInArea : [AccessControl] * String → Set<TurnStile> // portiques d'une zone
    isPresent : [AccessControl] * int → boolean // présence d'une personne
    getPersonsInArea : [AccessControl] * String → Set<Person> // personnes d'une zone
    getPersonsOnSite : [AccessControl] → Set<Person> // toutes les personnes sur site
    getNbInArea : [AccessControl] * String → int // nb de personnes dans une zone
    getNbOnSite : [AccessControl] → int // nb de personnes sur site
Constructors :
    init : PersonDB * Map<String,Set<TurnStile>> → [AccessControl]
Operators :
    access : [AccessControl] * int * String → [AccessControl]
        precondition : A compléter
    exit : [AccessControl] * int * String → [AccessControl]
        precondition : A compléter
Observations :
A compléter
```

L'opérateur d'accès access prend en paramètre l'identifiant unique d'une personne et le nom de la zone d'accès. Les préconditions à spécifier sont les suivantes :

– la personne doit être enregistrées avec accès autorisé à la zone spécifiée
– la carte correspondant à cette personne doit être insérée dans un des portiques de la zone (et pas un autre, détection de copie illégale de carte)
– le portique considéré doit être en mode d'entrée

Pour la sortie les préconditions sont similaire mais le portique doit être en mode de sortie.

Exercice 3

On aborde dans cet exercice la conception logicielle et quelques aspects du développement logiciel de notre système de contrôle des accès.

Question 3.1

Proposer un diagramme de composant ainsi qu'un diagramme de classes pour décrire respectivement l'architecture du système et l'architecture de son implémentation. On adopte dans cet exercice une approche orientée conception par contrat. Dans le diagramme de classe, on intégrera donc le support logiciel nécessaire pour la vérification embarquées des assertions logicielles.

Question 3.2

Donner le code de la classe dont la responsabilité est de vérifier à l'exécution si le service AccessControl est correctement implémenté.