Projet CSC4102 : Gestion des clefs dans un hôtel

Huang ShiHui et Mabileau Paul

Année 2019–2020 — 4 mars 2020

Table des matières

1 Spécification

1.1 Diagrammes de cas d'utilisation

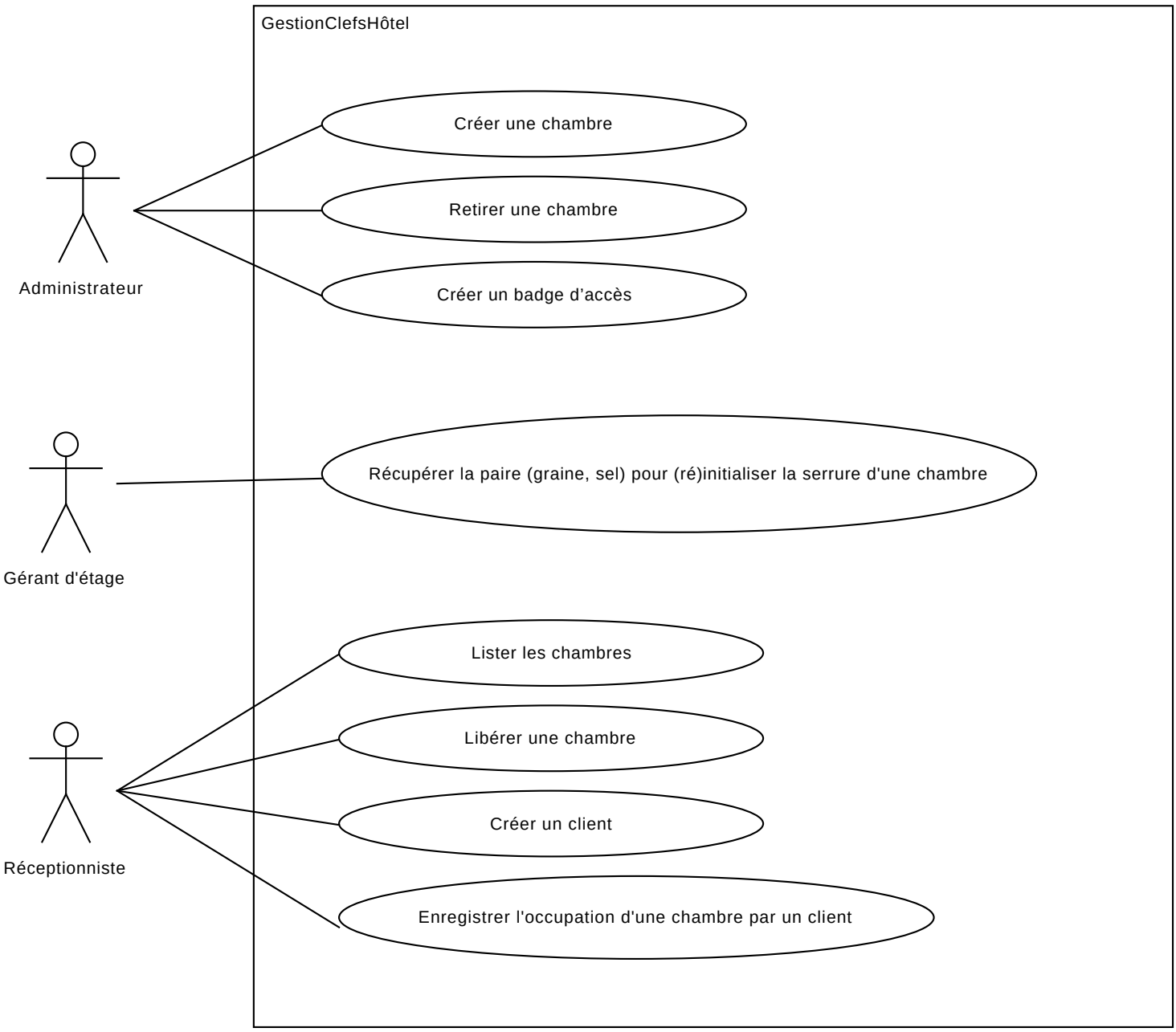

FIGURE 1 – Diagramme de cas d'utilisation

1.2 Priorités, préconditions et postconditions des cas d'utilisation

HAUTE n° 1
- Créer une chambre
  - précondition : identifiant/code de la chambre bien formé (non null et non vide)
    ∧ chambre avec ce code inexistante
    ∧ graine pour la génération des clefs bien formée (non null et non vide)
  - postcondition : chambre avec cet identifiant existante

HAUTE n° 2
- Créer un badge d'access
  - précondition :
  - postcondition :
    badge vierge

HAUTE n° 3
- Créer un client
  - précondition :
    nom prénom et identifiant du client bien formées (non null et non vide)
    ∧ client non existant dans le système
  - postcondition :
    client enregistré dans le système

HAUTE n° 4
- (Re)Initialiser la serrure d'une chambre
  - précondition :
    identifiant de la serrure bien formé (non null et non vide)
    ∧ graine et sel pour la génération des clefs bien formée (non null et non vide)
  - postcondition :
    serrure initialisé

HAUTE n° 5
- Enregistrer l'occupation d'une chambre par un client
  - précondition :
    nom et prénom du client bien formés (non null et non vide)
    ∧ client existe
    ∧ client occupe aucune chambre
    ∧ identifiant/code de la chambre bien formé (non null et non vide)
    ∧ chambre avec ce code existante
    ∧ chambre non occupée
    ∧ badge d'accèss disponible
  - postcondition :
    badge d'accès initialisé
    ∧ paire de clés du badge d'accès bien formées (non null et non vide)
    ∧ +1 sur nombre de chambre occupée en cours du client
    ∧ chambre occupée

HAUTE n° 6
- Libérer une chambre
  - précondition :
    ∧ client existe
    ∧ client occupe une chambre
    ∧ identifiant/code de la chambre bien formé (non null et non vide)
    ∧ chambre occupée

— postcondition :
∧ -1 sur nombre de chambre occupée en cous du client
∧ chambre non occupée

basse — Retirer une chambre

Moyenne — Lister les chambres

2 Préparation des tests de validation

2.1 Tables de décision des tests de validation

La fiche programme du module CSC4102 ne permettant pas de développer des tests de validation couvrant l'ensemble des cas d'utilisation de l'application, les cas d'utilisation choisis sont de priorité HAUTE.

Numéro de test	1	2	3	4
Identifiant/code de la chambre bien formé (non null et non vide)	F	T	T	T
Graine pour la génération des clefs bien formée ($\neq$ null $\wedge$ $\neq$ vide)		F	T	T
Chambre inexistante avec ce code			F	T
Création acceptée	F	F	F	T
Nombre de jeux de test	2	2	1	1

TABLE 1 – Cas d'utilisation « créer une chambre »

Numéro de test	1	2	3	4	5	6	7	8
Nom prénom et identifiant du client bien formés (non null et non vide)	F	T	T					T
Client existe		F	T					T
Client occupe aucune chambre			F					T
Identifiant/code de la chambre bien formé ($\neq$ null $\wedge$ $\neq$ vide)				F	T	T	T	T
Chambre avec ce code existante					F	T	T	T
Chambre non occupée						F	T	T
Dernière paire de clefs de la chambre bien formé ($\neq$ null $\wedge$ $\neq$ vide)							F	T
Enregistrement accepté	F	F	F	F	F	F	F	T
Nombre de jeux de test	2	1	1	2	1	1	2	2

TABLE 2 – Cas d'utilisation « enregistrer l'occupation d'une chambre par un client »

Numéro de test	1	2	3	4	5	6	7
Client existe	F	T					T
Client occupe une chambre		F					T
Identifiant/code de la chambre bien formé ($\neq$ null $\wedge \neq$ vide)			F	T	T	T	T
Chambre occupée					F	T	T
Paire de clés du badge d'accès bien formées ($\neq$ null $\wedge \neq$ vide)						F	T
Libération acceptée	F	F	F	F	F	F	T
Nombre de jeux de test	1	1	2	1	1	2	2

TABLE 3 – Cas d'utilisation « libérer une chambre »

3 Conception

3.1 Liste des classes

À la suite d'un parcours des diagrammes de cas d'utilisation et d'une relecture de l'étude de cas, voici la liste de classes avec quelques attributs :

- GestionClefsHotel (la façade)
- Chambre — identifiant, graine, sel
- Client — identifiant, nom, prénom (ces deux derniers sont ajoutés ici mais ne sont pas essentiels au fonctionnement du système GestionClefsHotel)
- Badge — identifiant
- PaireClefs — clef1, clef2
- Util (classe utilitaire déjà programmée) — 'attribut de classe TAILLE_CLEF, méthodes de classe genererUneNouvelleClef et clefToString)

3.2 Diagramme de classes

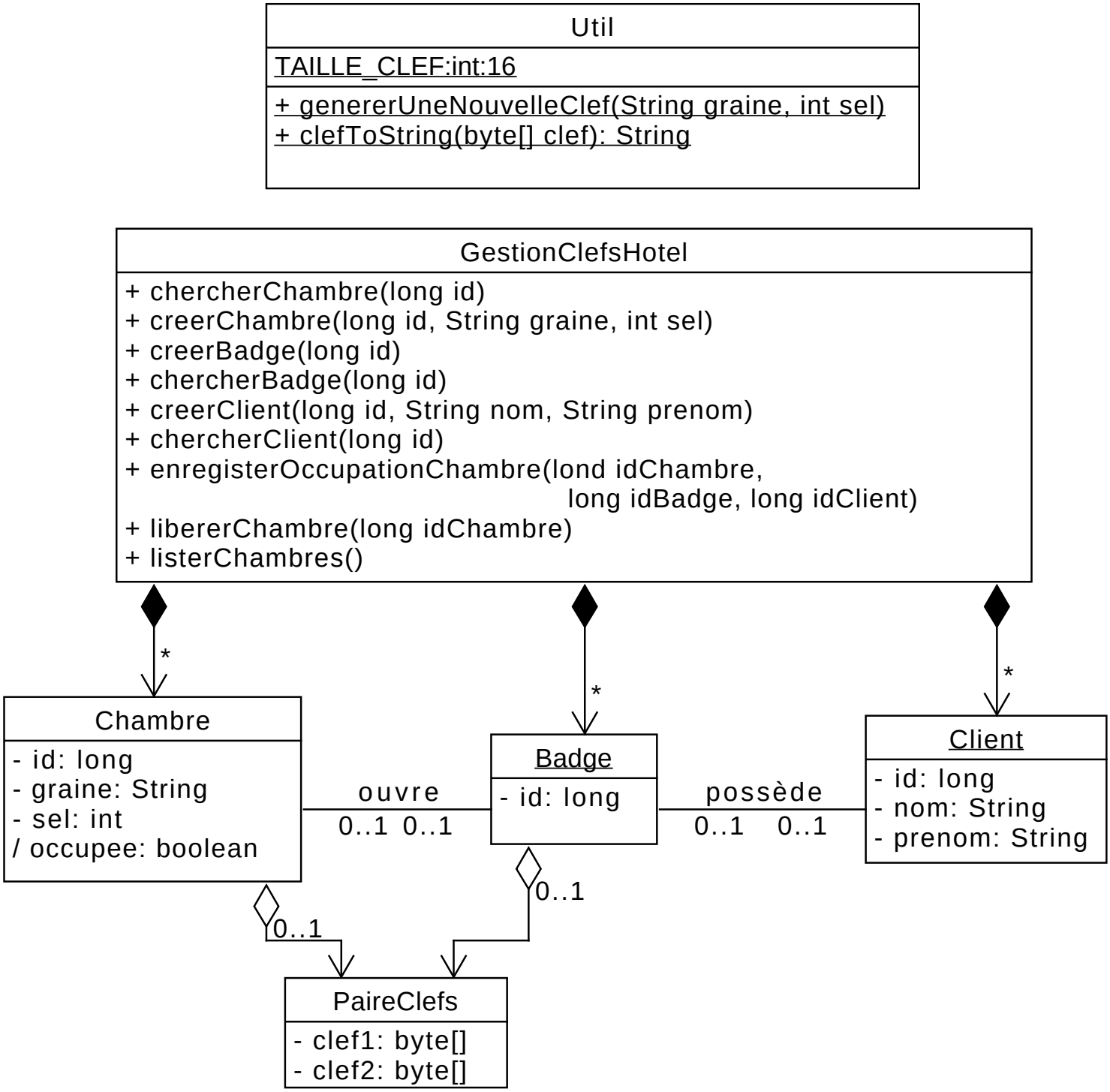

FIGURE 2 – Diagramme de classes

3.3 Diagrammes de séquence

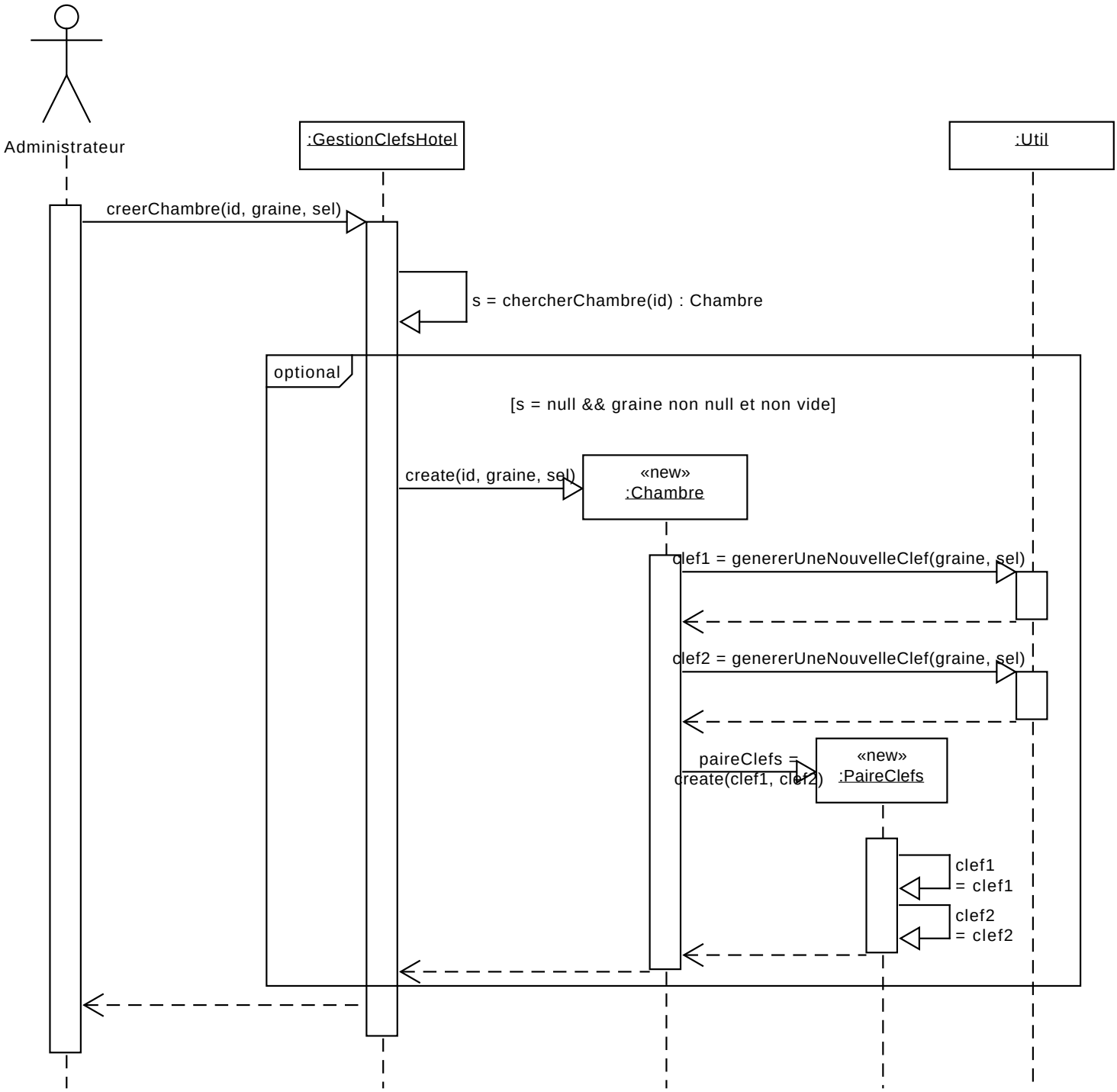

FIGURE 3 – Diagramme de séquence DSUC1 : «Créer une chambre»

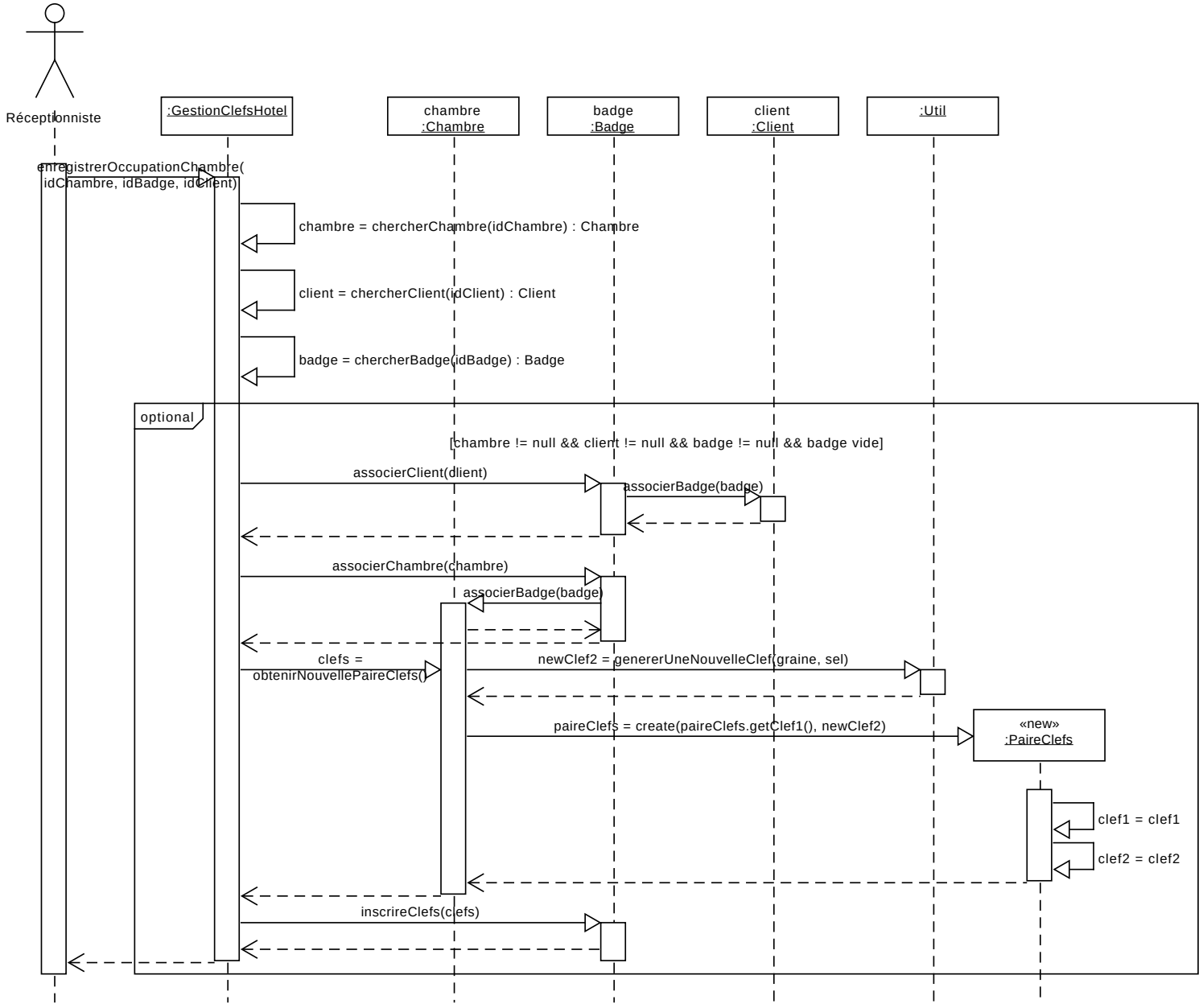

FIGURE 4 – Diagramme de séquence DSUC2 : «Enregistrer l'occupation d'une chambre par un client»

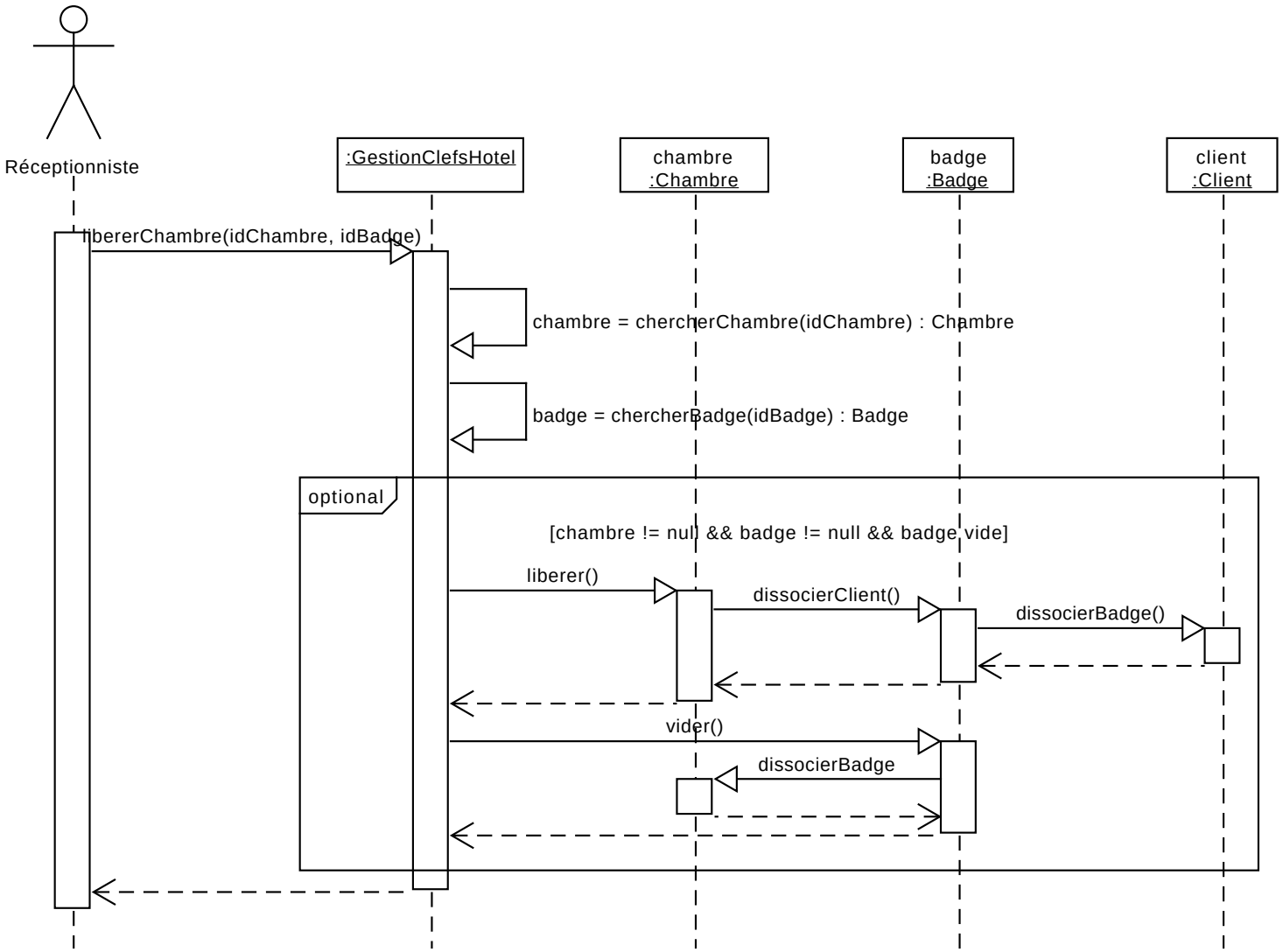

FIGURE 5 – Diagramme de séquence DSUC3 : «Libérer une chambre»

4 Fiche des classes

4.1 Classe GestionClefsHotel

GestionClefsHotel
<– **attributs « association »** –> – chambres : [Chambre] – badges : [Badge] – clients : [Client]
<– **constructeur** –> + GestionClefsHotel() + boolean invariant() <– **operations « cas d'utilisation »** –> + Chambre creerChambre(long id, String graine, int sel) + Badge creerBadge(long id) + Client creerClient(long id, String nom, String prenom) + void enregistrerOccupationChambre(long idChambre, long idBadge, long idClient) + [Chambre] listerChambres() + void libererChambre(long id) <– **opérations de recherche** –> + Badge chercherBadge(long id) + Chambre chercherChambre(long id) + Client chercherClient(long id)

4.2 Classe Chambre

Chambre
<– **attributs** –> – id : long – graine : String – sel : int – occupee : boolean <– **attributs « association »** –> – paireClefs : PaireClefs – badge : Badge
<– **constructeur** –> + Chambre(long id, String graine, int sel) + boolean invariant() <– **operations « cas d'utilisation »** –> + void inscrireClefs(PaireClefs paireClefs) + void liberer()

4.3 Classe Badge

Badge
<– **attributs** –> – id : long <– **attributs « association »** –> – paireClefs : PaireClefs – chambre : Chambre – client : Client
<– **constructeur** –> + Badge(long id) + boolean invariant() <– **operations « cas d'utilisation »** –> + void inscrireClefs(PaireClefs paireClefs) + void vider() + void associerClient(Client client) + void associerChambre(Chambre chambre) + void dissocierClient()

4.4 Classe Client

Client
<– **attributs** –> – id : long – nom : String – prenom : String <– **attributs « association »** –> – badge : Badge
<– **constructeur** –> + Client(long id, String nom, String prenom) + boolean invariant()

4.5 Classe PaireClefs

PaireClefs
<– **attributs** –> – clef1 : byte[] – clef2 : byte[]
<– **constructeur** –> + PaireClefs(byte[] clef1, byte[] clef2) + boolean invariant()

5 Diagrammes de machine à états et invariants

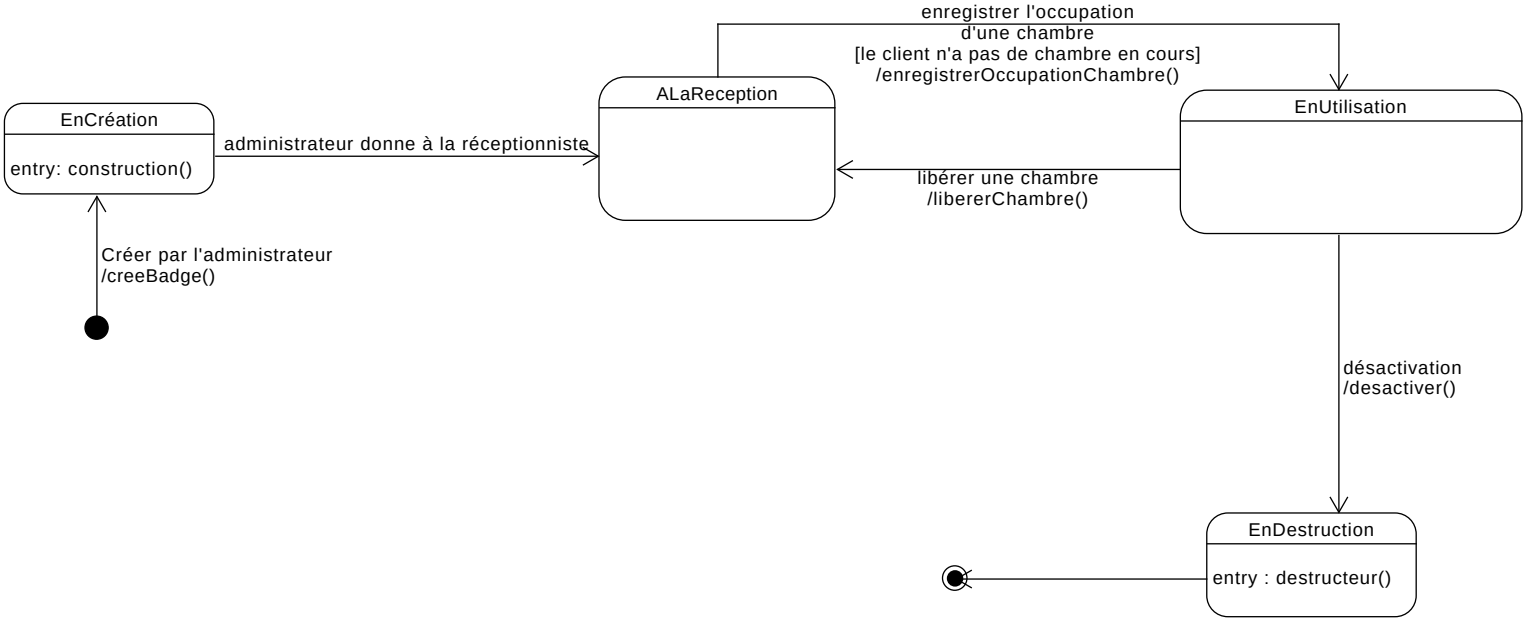

FIGURE 6 – DiagrammesDeMachineAEtats

Invariants : identifiant de badge non null

6 Préparation des tests unitaires

La section est à compléter avec les tables de décision de certaines méthodes des classes les plus importantes.