Projet CSC4102 : Gestion des clefs dans un hôtel

Huang ShiHui et Mabileau Paul

Année 2019–2020 — 28 mars 2020

Table des matières

1 Spécification

1.1 Diagrammes de cas d'utilisation

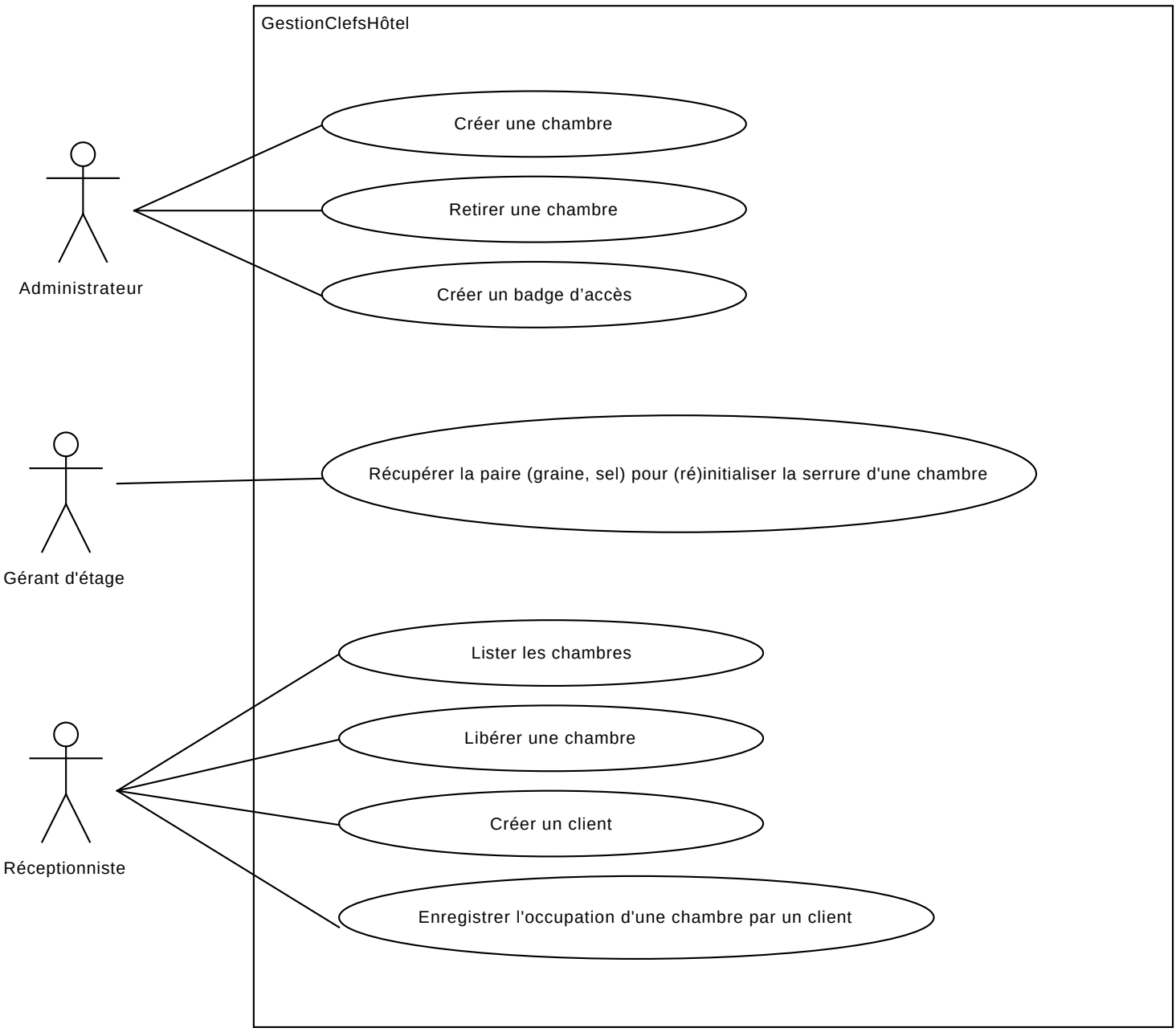

FIGURE 1 – Diagramme de cas d'utilisation du sprint 1

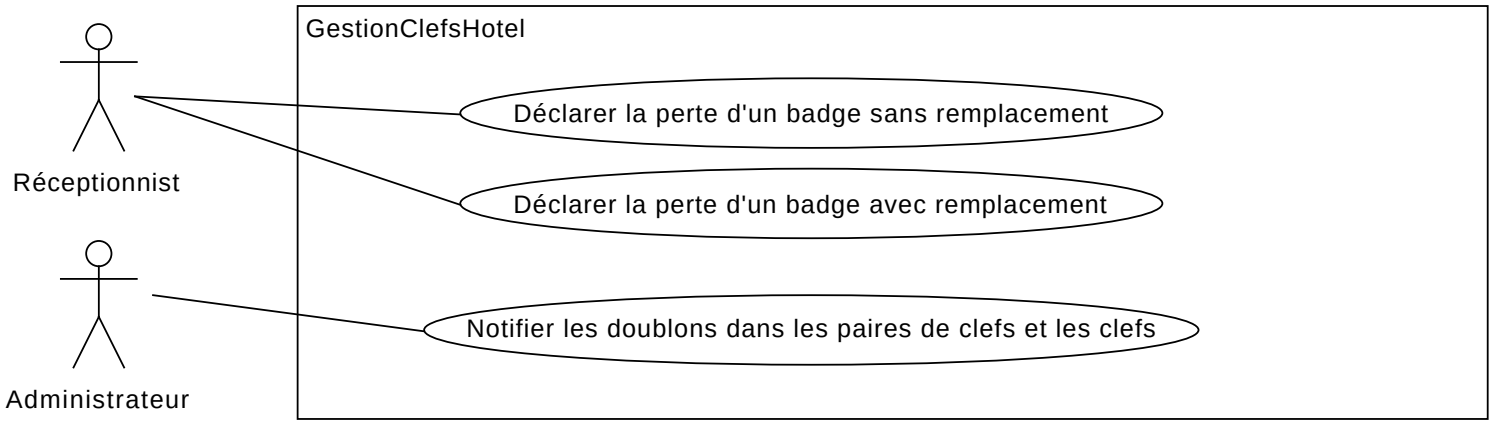

FIGURE 2 – Diagramme de cas d'utilisation du sprint 2

1.2 Priorités, préconditions et postconditions des cas d'utilisation

1.2.1 Sprint 1

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

HAUTE n° 2 — Créer un badge d'access
- précondition :
  ∧ identifiant de le badge bien formé (non null et non vide)
  ∧ identifiant de le badge inexistante
- postcondition :
  chambre avec cet identifiant existante

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 — Enregistrer l'occupation d'une chambre par un client
- précondition :
  ∧ client existante ∧ chambre existante ∧ le badge d'accèss existante ∧ client occupe aucune chambre
  ∧ chambre non occupée
  ∧ badge d'accèss disponible(badge n'associe aucun d'autre clients et chambres, paireClefs sont null) ∧ Dernière paire de clefs de la chambre bien formé (non null et non vide)

— postcondition :
∧ paire de clés du badge d'accès bien formées (non null et non vide)
∧ badge associe avec client et chambre
∧ chambre occupée

HAUTE n° 5 — Libérer une chambre
— précondition :
∧ client existante
∧ badge existante
∧ chambre existante
∧ client occupe une chambre
∧ client occupe la bonne chambre
∧ chambre occupée
— postcondition :
∧ vider le clef du badge
∧ disassocie les relation entre le badge et la chambre, le badge et le client
∧ chambre non occupée

HAUTE n° 6 — (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

basse — Retirer une chambre

Moyenne — Lister les chambres

1.2.2 Sprint 2

HAUTE n° 7 — Déclarer la perte d'un badge sans remplacement
— précondition :
badge avec ce identifiant existant

— postcondition :
(Si le badge était en cours d'utilisation) chambre occupée par le client possédant le badge est libérée
∧ (Si le badge était en cours d'utilisation) badge d'accès retiré du client
∧ badge avec ce identifiant inexistant

HAUTE n° 8 — Déclarer la perte d'un badge avec remplacement
— précondition :
badge avec ce identifiant existant
— postcondition :

(Si le badge était en cours d'utilisation) chambre occupée par le client possédant le badge est libérée

∧ (Si le badge était en cours d'utilisation) badge d'accès retiré du client
∧ badge avec ce identifiant inexistant
∧ enregistrerOccupationChambre avec la meme client, meme chambre et d'autre badges

Moyenne — Notifier les doublons dans les paires de clefs et les clefs

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.

2.1.1 Sprint 1

Numéro de test	1	2	3
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	T
Nombre de jeux de test	2	1	1

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

Numéro de test	1	2	3	4	5	6	7	8
Chambre existante	F	T	T					T
Badge existant		F	T					T
Client existant			F					T
Chambre non occupée				F	T	T	T	T
Client occupe aucune chambre					F	T	T	T
Badge disponible						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	1	1	1	1	1	1	2	1

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

Numéro de test	1	2	3	4	5	6	7
Chambre existante	F	T	T				T
Badge existant		F	T				T
Client existant			F				T
Client occupe une chambre				F	T	T	T
Client occupe la bonne chambre					F	T	T
Chambre occupeé						F	T
Libération acceptée	F	F	F	F	F	F	T
Nombre de jeux de test	1	1	1	1	1	1	2

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

2.1.2 Sprint 2

Numéro de test	1	2
Badge avec ce identifiant existant	F	T
Badge déclaré perdu acceptée	F	T
Nombre de jeux de test	1	1

TABLE 4 – Cas d'utilisation « déclarer la perte d'un badge d'accès »

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
- Clef — clef
- PaireClefs — clef1, clef2
- Util (classe utilitaire déjà programmée) — 'attribut de classe TAILLE_CLEF, méthodes de classe genererUneNouvelleClef et clefToString)

3.2 Diagramme de classes

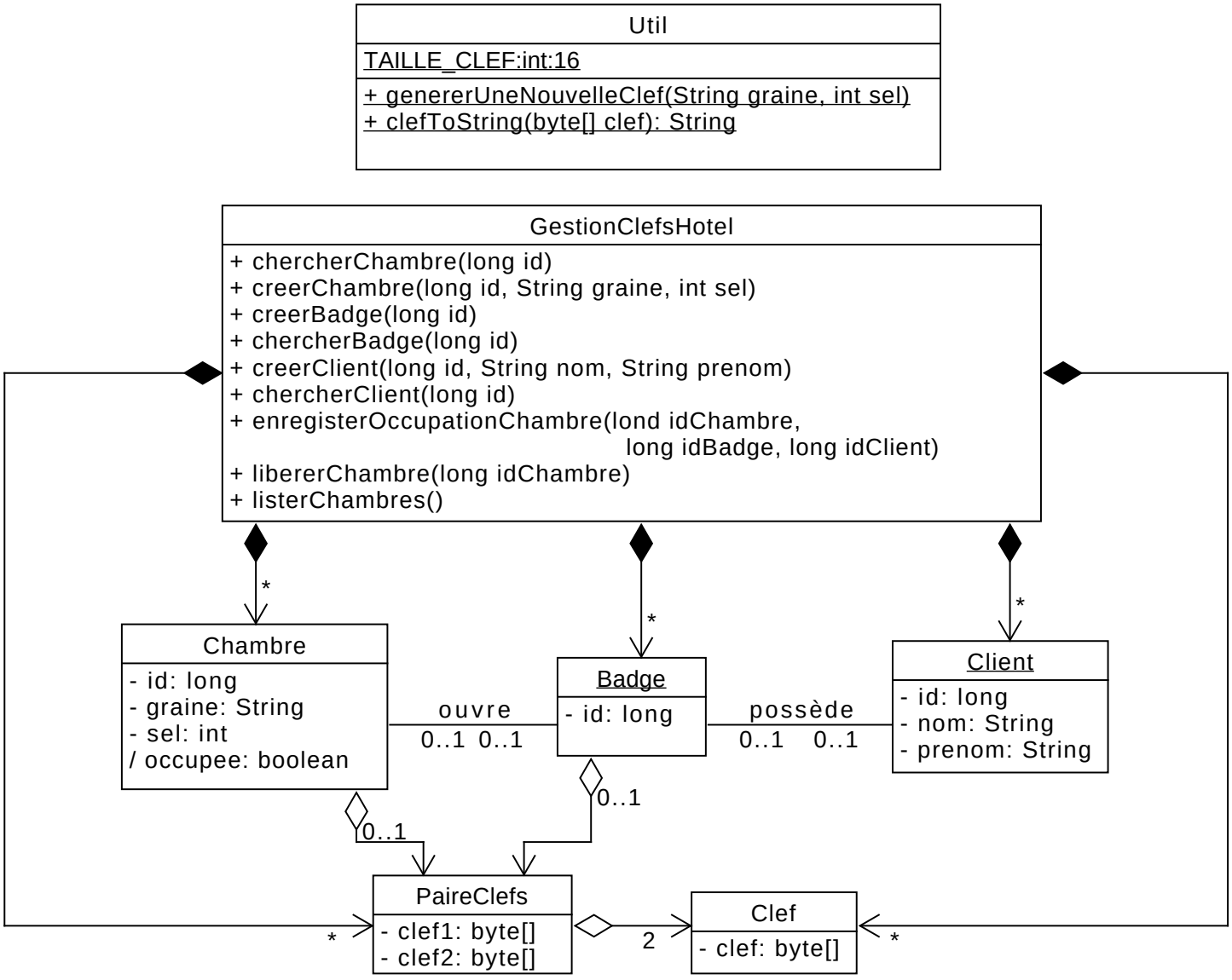

FIGURE 3 – Diagramme de classes

3.3 Diagrammes de séquence

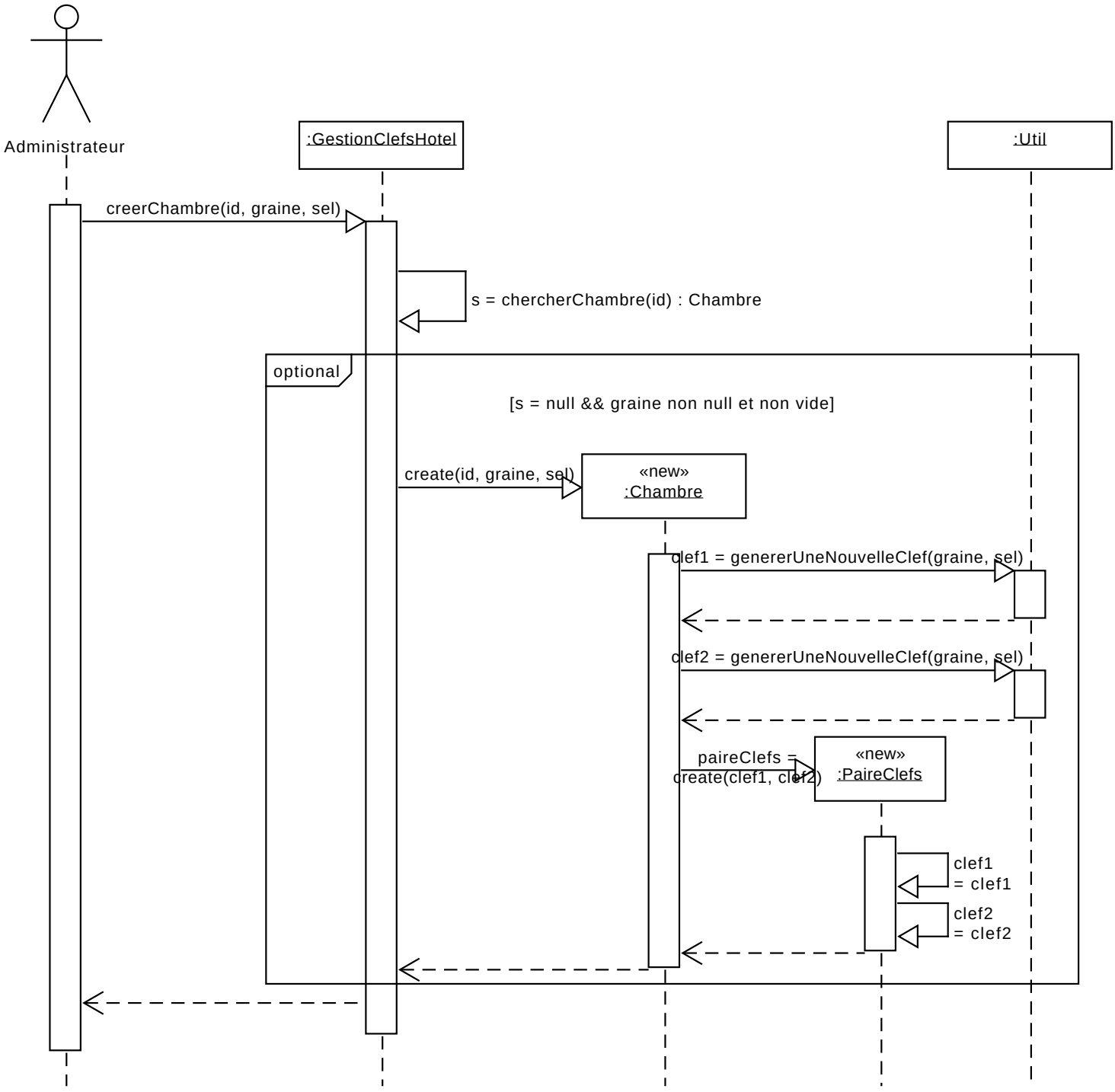

FIGURE 4 – Diagramme de séquence DSUC1 : «Créer une chambre»

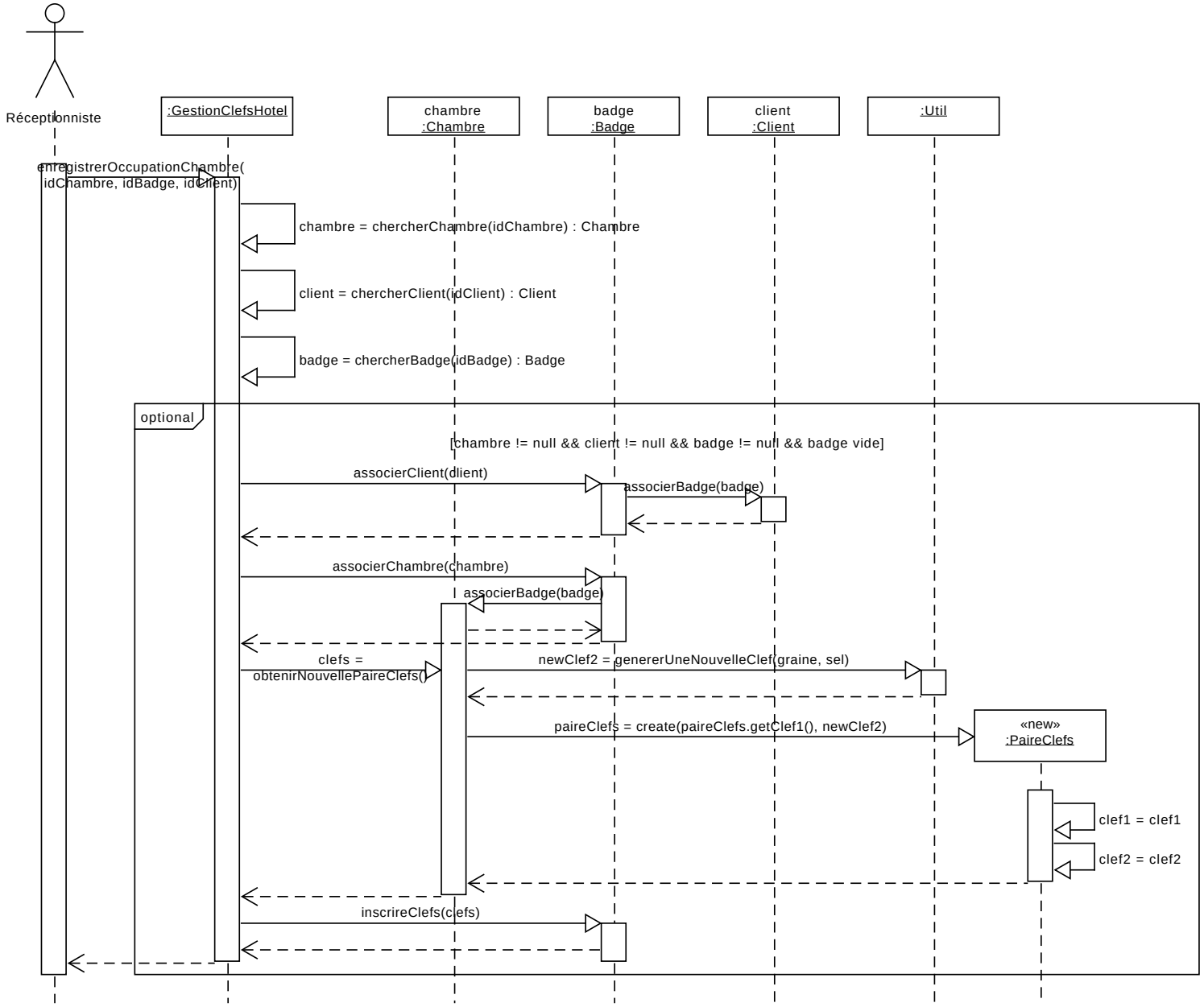

FIGURE 5 – Diagramme de séquence DSUC2 : «Enregistrer l'occupation d'une chambre par un client»

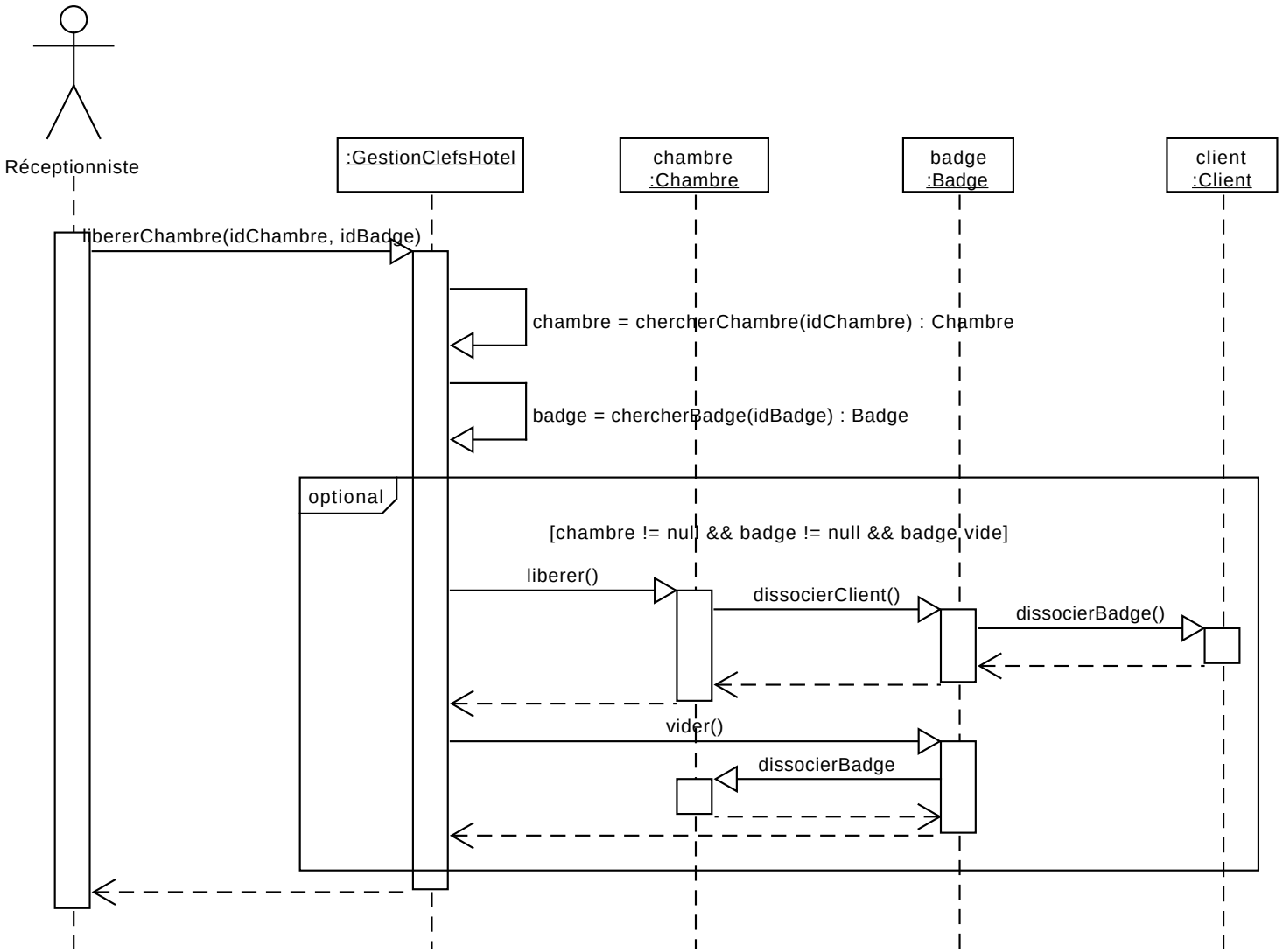

FIGURE 6 – Diagramme de séquence DSUC3 : «Libérer une chambre»

3.4 Diagrammes de séquence

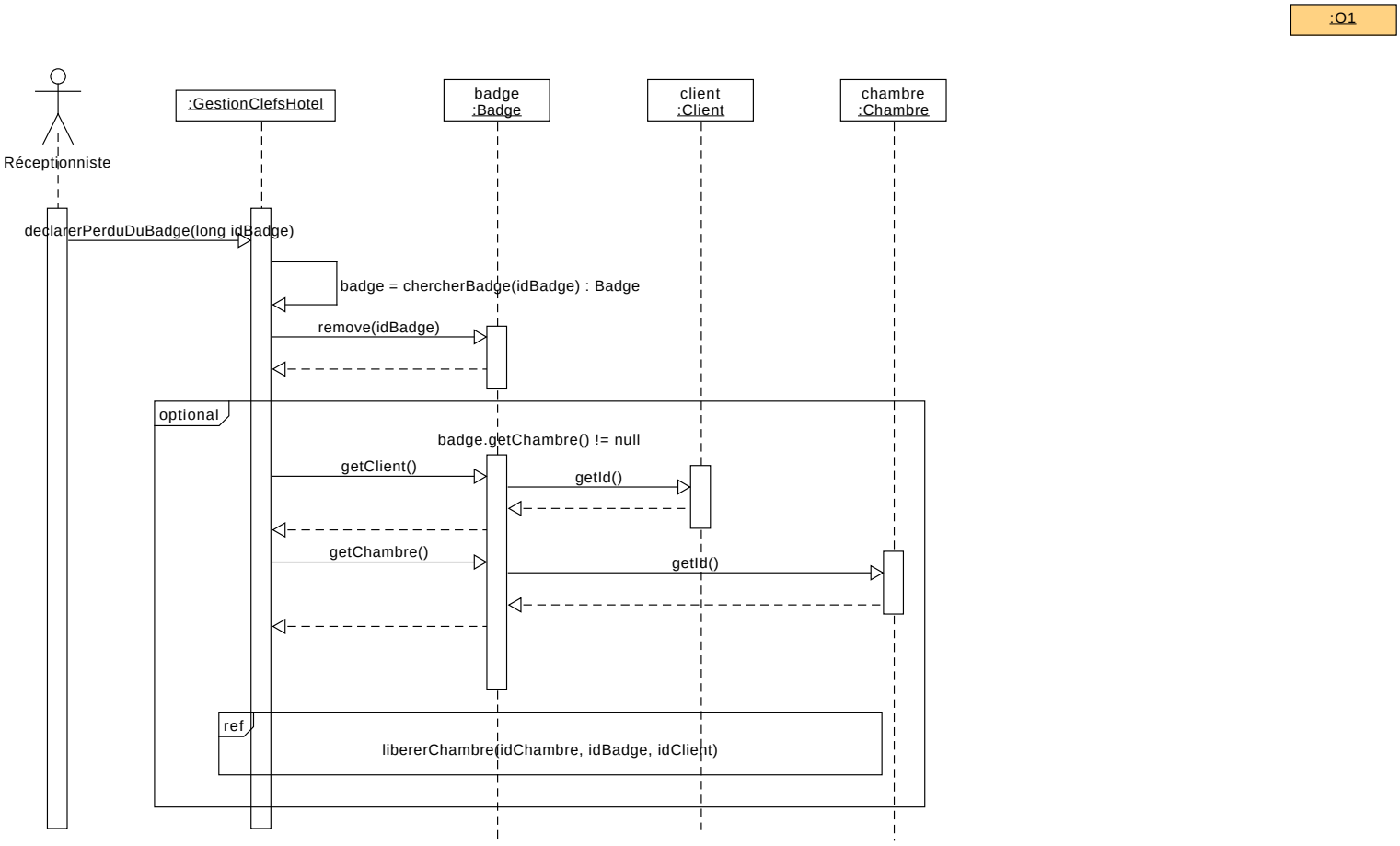

FIGURE 7 – Diagramme de séquence DSUC4 : «Déclarer le perte du badge»

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() + void enregistrerChambre()

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

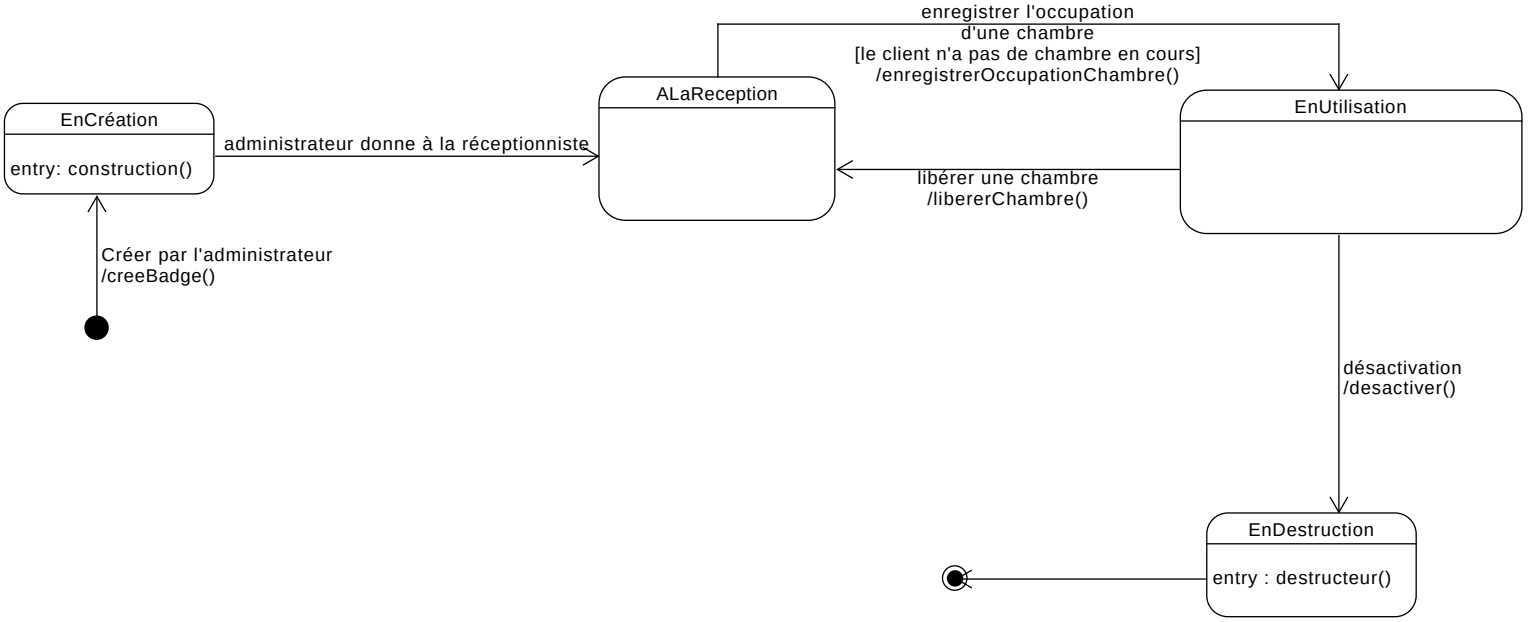

FIGURE 8 – DiagrammesDeMachineAEtats

Invariants :

(un client associe le badge
∧ un chambre associe le badge
∧ les paireClefs de badge ≠ null
∧ le badge du chambre est pareil que ce badge
∧ le badge du client est pareil que ce badge)
∨
(∧ aucun client associe le badge
∧ aucun chambre associe le badge
∧ les paireClefs de badge = null
)

6 Préparation des tests unitaires

	Numéro de test	1
Postcondition	badge $\neq$ null	T
	paireClefs = null	T
	client = null	T
	chambre = null	T
Exception	Levée d'une exception	NON
Effet	Création du badge accepté	T
	Nombre de jeux de test	1

Table 10 – Table de décision du "constructeur" de la classe "Badge"

	Numéro de test	1
Postcondition	client = client	T
	badge.clent = client.badge	T
Exception	Levée d'une exception	NON
Effet	Association Client Bidirectionnelle accepté pour le badge	T
	Nombre de jeux de test	1

Table 11 – Table de décision du "AssociationClientBidirectionnelle" de la classe "Badge"