Projet GELO : Étude de cas l' « Gestion de bureaux en situation de bilocalisation »

Bilal KEFIF et Soni DIEDHIOU

Année 2025–2026 — December 1, 2025

Contents

1 Spécification

1.1 Diagrammes de cas d'utilisation

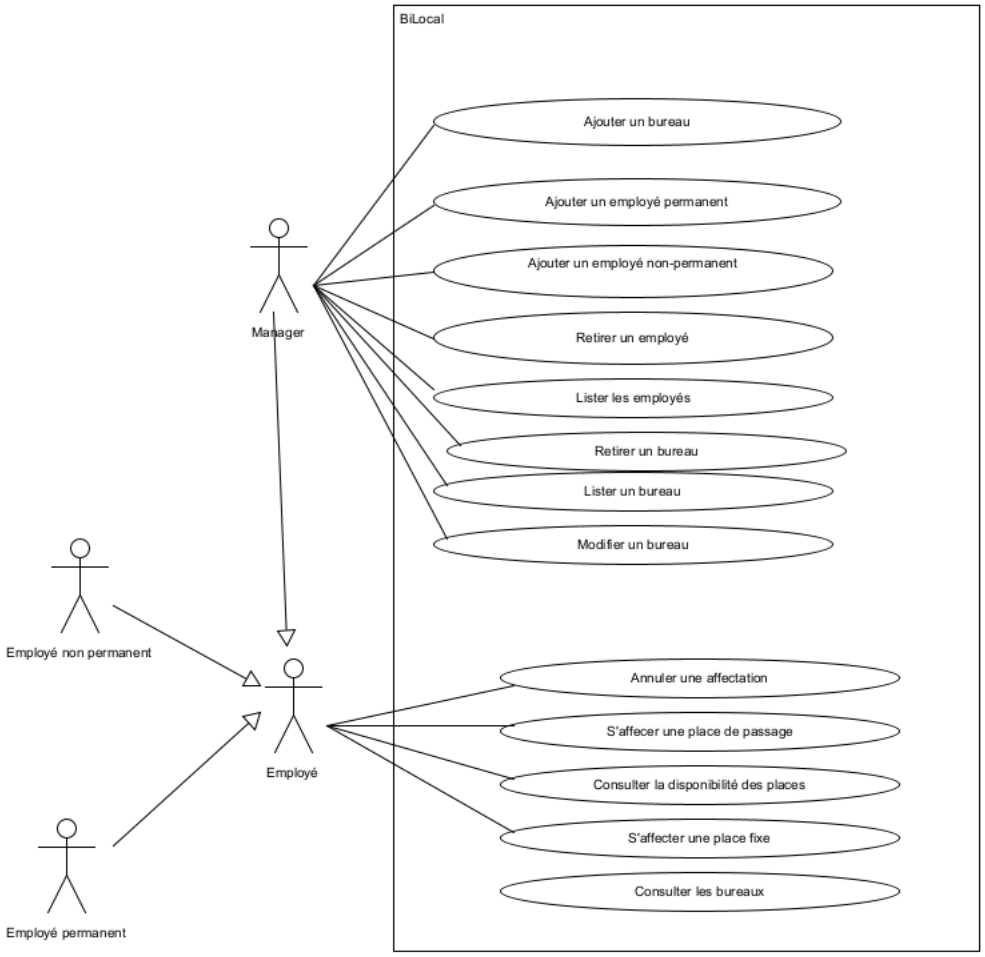

Figure 1: Diagramme de cas d'utilisation

1.2 Priorités, préconditions et postconditions des cas d'utilisation

Les priorités des cas d'utilisation pour le `sprint 1` sont choisies avec les règles de bon sens suivantes:

- pour retirer une entité du système, elle doit y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du retrait;
- pour lister les entités d'un type donné, elles doivent y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du listage;
- il est *a priori* possible, c.-à-d. sans raison contraire, de démontrer la mise en œuvre d'un sous-ensemble des fonctionnalités du système, et plus particulièrement la prise en compte des principales règles de gestion, sans les retraits ou les listages.
- la possibilité de lister aide au déverminage de l'application pendant les activités d'exécution des tests de validation.

Par conséquent, les cas d'utilisation d'ajout sont *a priori* de priorité « haute », ceux de listage de priorité « moyenne », et ceux de retrait de priorité « basse ».

Dans la suite, nous donnons les préconditions et postconditions pour les cas d'utilisation de priorité « Haute ». Pour les autres, nous indiquons uniquement leur niveau de priorité.

1.3 Priorités, préconditions et postconditions des cas d'utilisation

Les priorités des cas d'utilisation pour le `sprint 1` sont choisies avec les règles de bon sens suivantes:

- pour retirer une entité du système, elle doit y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du retrait;
- pour lister les entités d'un type donné, elles doivent y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du listage;
- il est *a priori* possible, c.-à-d. sans raison contraire, de démontrer la mise en œuvre d'un sous-ensemble des fonctionnalités du système, et plus particulièrement la prise en compte des principales règles de gestion, sans les retraits ou les listages;
- la possibilité de lister aide au déverminage de l'application pendant les activités d'exécution des tests de validation.

Par conséquent, les cas d'utilisation d'ajout sont *a priori* de priorité « haute », ceux de listage de priorité « moyenne », et ceux de retrait de priorité « basse ».

Dans la suite, nous donnons les préconditions et postconditions pour les cas d'utilisation de priorité « Haute ». Pour les autres, nous indiquons uniquement leur niveau de priorité.

HAUTE n° 1

- Ajouter un employé permanent
  - précondition : identifiant de l'employé bien formé (non null et non vide)
    ∧ nom bien formé (non null et non vide)
    ∧ prénom bien formé (non null et non vide)
    ∧ date d'embauche non null
    ∧ fonction du permanent bien formée (non null et non vide)
    ∧ fonction du permanent ∈ {direction département, direction adjointe département, assistance gestion, enseignement recherche}
    ∧ employé avec cet identifiant inexistant

– postcondition : un employé permanent avec cet identifiant existe dans le système

HAUTE n° 2

- Ajouter un employé non-permanent
  - précondition : identifiant de l'employé bien formé (non null et non vide)
    ∧ nom bien formé (non null et non vide)
    ∧ prénom bien formé (non null et non vide)
    ∧ date d'embauche non null
    ∧ date de fin de contrat non null
    ∧ date de fin de contrat ≥ date d'embauche
    ∧ fonction du non-permanent bien formée (non null et non vide)
    ∧ fonction du non-permanent ∈ {doctorat, post-doctorat, ingénierie recherche, stage}
    ∧ employé avec cet identifiant inexistant
  - postcondition : un employé non-permanent avec cet identifiant existe dans le système

HAUTE n° 3

- Ajouter un bureau
  - précondition : identifiant du bureau bien formé (non null et non vide)
    ∧ site cible existant dans le système
    ∧ type de bureau bien formé (non null et non vide)
    ∧ type de bureau ∈ {PERMANENTS, NON_PERMANENTS}
    ∧ nombres de places fixes et de passage bien formés (entiers ≥ 0)
    ∧ nbPlacesTotales = nbPlacesFixes + nbPlacesPassage
    ∧ si typeBureau = PERMANENTS alors nbPlacesTotales ∈ {1, 2} et nbPlacesFixes = 1
    ∧ si typeBureau = NON_PERMANENTS alors nbPlacesTotales ≤ 6
    ∧ aucun bureau avec cet identifiant inexistant sur ce site
  - postcondition : un bureau avec cet identifiant existe dans le système,
    rattaché au site donné, avec le type indiqué,
    et les nombres de places fixes et de passage respectant les contraintes

HAUTE n° 4

- S'affecter une place fixe
  - précondition : employé connecté au système et identifié
    ∧ employé existant dans le système
    ∧ date d'occupation demandée non null
    ∧ existence, sur le site choisi, d'au moins une place de type FIXE disponible à cette date
    ∧ l'employé ne dispose d'aucune autre affectation de place fixe active (place fixe: max 1 par employé, sur un seul site)
    ∧ la granularité 1 place/jour/site n'est pas violée pour cet employé
  - postcondition : une nouvelle affectation de place FIXE est créée pour cet employé,
    sur le site et à la date demandés, avec `dateFin = null`,
    et les contraintes de gestion (au plus une place fixe par employé,
    1 place/jour/site) restent satisfaites

HAUTE n° 5

- S'affecter une place de passage
  - précondition : employé connecté au système et identifié
    ∧ employé existant dans le système
    ∧ site ciblé existant dans le système
    ∧ dates de début et de fin de réservation bien formées (non null et cohérentes)

∧ dateFin ≥ dateDebut
∧ existence d'au moins une place de type PASSAGE disponible sur ce site pour toute la période demandée
∧ pour cet employé, les nouvelles périodes de passage ne se chevauchent pas avec ses affectations de passage existantes
∧ pour cet employé, la granularité 1 place/jour/site est respectée

– postcondition : une nouvelle affectation de place PASSAGE est créée pour cet employé,
sur le site choisi et pour la période `[dateDebut, dateFin]`,
en respectant la granularité 1 place/jour/site et la non-chevauchement des périodes

Moyenne	• Lister les employés
Moyenne	• Lister un bureau
Moyenne	• Consulter la disponibilité des places
Moyenne	• Consulter les bureaux
Moyenne	• Modifier un bureau
Moyenne	• Retirer un employé
basse	• Retirer un bureau
basse	• Annuler une affectation
basse	

2 Préparation des tests de validation

2.1 Tables de décision des tests de validation

La fiche programme du module GELO 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	5	6	7	8
Identifiant de l'employé bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T
Nom bien formé ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T
Prénom bien formé ($\neq$ null $\wedge \neq$ vide)			F	T	T	T	T	T
Date d'embauche $\neq$ null				F	T	T	T	T
Fonction du permanent bien formée ($\neq$ null $\wedge \neq$ vide)					F	T	T	T
Fonction du permanent $\in$ {direction département, direction adjointe département, assistance gestion, enseignement recherche}						F	T	T
Employé avec cet identifiant non existant							F	T
Création acceptée	F	F	F	F	F	F	F	T
Nombre de jeux de test	2	2	2	1	2	5	1	1

Table 1: Cas d'utilisation « ajouter un employé permanent ». Le test 6 possède 5 jeux de test pour les 4 fonctions de non-permanents et pour une fonction inconnue.

Numéro de test	1	2	3	4	5	6	7	8
Identifiant bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T
Nom bien formé ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T
Prénom bien formé ($\neq$ null $\wedge \neq$ vide)			F	T	T	T	T	T
Date d'embauche $\neq$ null				F	T	T	T	T
Date fin de contrat $\neq$ null					F	T	T	T
Date fin $\geq$ date embauche						F	T	T
Fonction non-permanent bien formée ($\neq$ null $\wedge \neq$ vide)							F	T
Fonction $\in$ {doctorat, post-doctorat, ingénierie recherche, stage}								F
Création acceptée	F	F	F	F	F	F	F	T
Nombre de jeux de test	2	2	2	1	1	2	1	1

Table 2: Cas d'utilisation « ajouter un employé non-permanent ».

Numéro de test	1	2	3	4	5	6	7	8	9	10	11	12
Identifiant du site bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T	T	T	T	T
Identifiant de bureau bien formé ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T	T	T	T	T
Site cible existant dans le système			F	T	T	T	T	T	T	T	T	T
Aucun bureau avec cet id sur ce site				F	T	T	T	T	T	T	T	T
Nombres de places ≥ 0					F	T	T	T	T	T	T	T
Nombre total de places ≥ 1						F	T	T	T	T	T	T
Nombre total de places ≤ 6 (NON_PERMANENTS)							F	T	T	T	T	T
Création acceptée	F	F	F	F	F	F	F	T	T	T	T	T
Jeux de test (tests 8–12: répartitions valides)	2	2	1	1	2	1	3	1	1	1	1	1

Table 3: Cas d'utilisation « ajouter un bureau non-permanent » (21 tests).

Numéro de test	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Id employé bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T	T	T	T	T	T	T
Id place bien formé ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T	T	T	T	T	T	T
Employé existant dans le système			F	T	T	T	T	T	T	T	T	T	T	T
Place existante dans le système				F	T	T	T	T	T	T	T	T	T	T
Compatibilité permanent/non-permanent					F	T	T	T	T	T	T	T	T	T
Place de type FIXE (pas PASSAGE)						F	T	T	T	T	T	T	T	T
Quota places fixes respecté (normal: max 1, manager: max 1/site)							F	T	T	T	T	T	T	T
Place disponible (non occupée)								F	T	T	T	T	T	T
Affectation acceptée	F	F	F	F	F	F	F	F	T	T	T	T	T	T
Jeux de test (tests 9–14: cas valides)	2	2	1	1	2	1	2	1	1	1	1	1	1	1

Table 4: Cas d'utilisation « affecter une place fixe » (18 tests).

Numéro de test	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Id employé bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T	T	T	T	T	T	T
Id place bien formé ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T	T	T	T	T	T	T
Date de début $\neq$ null			F	T	T	T	T	T	T	T	T	T	T	T
Employé existant dans le système				F	T	T	T	T	T	T	T	T	T	T
Place existante dans le système					F	T	T	T	T	T	T	T	T	T
Compatibilité permanent/non-permanent						F	T	T	T	T	T	T	T	T
Place de type PASSAGE (pas FIXE)							F	T	T	T	T	T	T	T
Date début pas dans le passé								F	T	T	T	T	T	T
Date fin $\geq$ date début (si définie)									F	T	T	T	T	T
Place disponible pour la période										F	T	T	T	T
Affectation acceptée	F	F	F	F	F	F	F	F	F	F	T	T	T	T
Jeux de test (tests 11–14: cas valides)	2	2	1	1	1	2	1	1	1	2	1	1	1	1

Table 5: Cas d'utilisation « affecter une place de passage » (20 tests).

3 Conception

3.1 Liste des classes

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

- BiLOCAL (façade du système) — frontière, pas d'attribut métier
- Employé — id:String, nom:String, prénom:String, dateEmbauche:LocalDate, dateFinContrat:LocalDate?, fonction:Fonction
- Site — id:String, nom:String, adresse:String
- Bureau — id:String, site:Site, places:List<Place>, typeBureau:TypeBureau, nbPlacesFixes:int, nbPlacesPassage:int
- Place — id:String, typePlace:TypePlace, bureau:Bureau affectations:List<Affectation>
- Affectation — id:String, employe: Employe, place:Place, dateDebut:LocalDate, dateFin:LocalDate?, dateAffectation:LocalDate
- «enum» Fonction — {DIRECTION_DEPARTEMENT, DIRECTION_ADJOINTE_DEPARTEMENT, ASSISTANCE_GESTION, ENSEIGNEMENT_RECHERCHE, DOCTORAT, POST_DOCTORAT, INGENIERIE_RECHERCHE, STAGE}
- «enum» TypeFonction — {PERMANENT, NON_PERMANENT}
- «enum» TypePlace — {FIXE, PASSAGE}
- «enum» TypeBureau — {PERMANENTS, NON_PERMANENTS}

3.2 Diagramme de classes

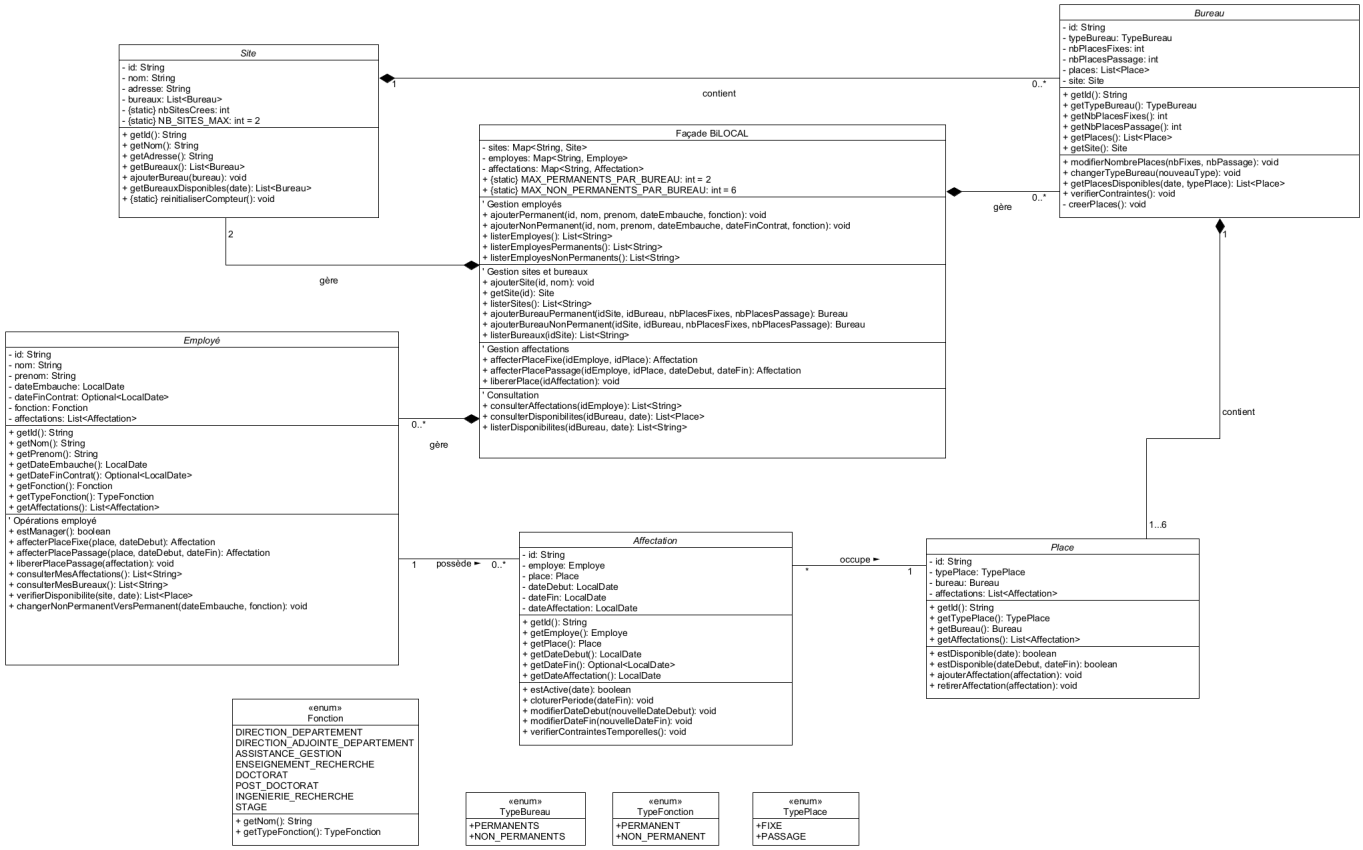

Figure 2: Diagramme de classes de la conception préliminaire

3.3 Diagrammes de séquence

Voici la description textuelle du cas d'utilisation « ajouter un employé permanent » :

- **arguments en entrée** : identifiant de l'employé, nom de l'employé, prénom de l'employé, date d'embauche, fonction ;
- **rappel de la précondition** : identifiant non null et non vide $\wedge$ nom non null et non vide $\wedge$ prénom non null et non vide $\wedge$ date d'embauche non null $\wedge$ fonction non null $\wedge$ fonction du permanent $\in$ {direction département, direction adjointe département, assistance gestion, enseignement recherche} $\wedge$ employé avec cet identifiant inexistant ;
- **algorithme** :
  1. vérifier que tous les arguments sont bien formés (non null, non vides, dates valides) ;
  2. vérifier que la fonction correspond à un *type permanent* : fonction.getTypeFonction() = PERMANENT ;
  3. chercher un employé avec cet identifiant ;
  4. vérifier que l'employé est inexistant ;
  5. instancier l'employé permanent (dateFinContrat = null) ;
  6. ajouter l'employé dans la collection des employés.

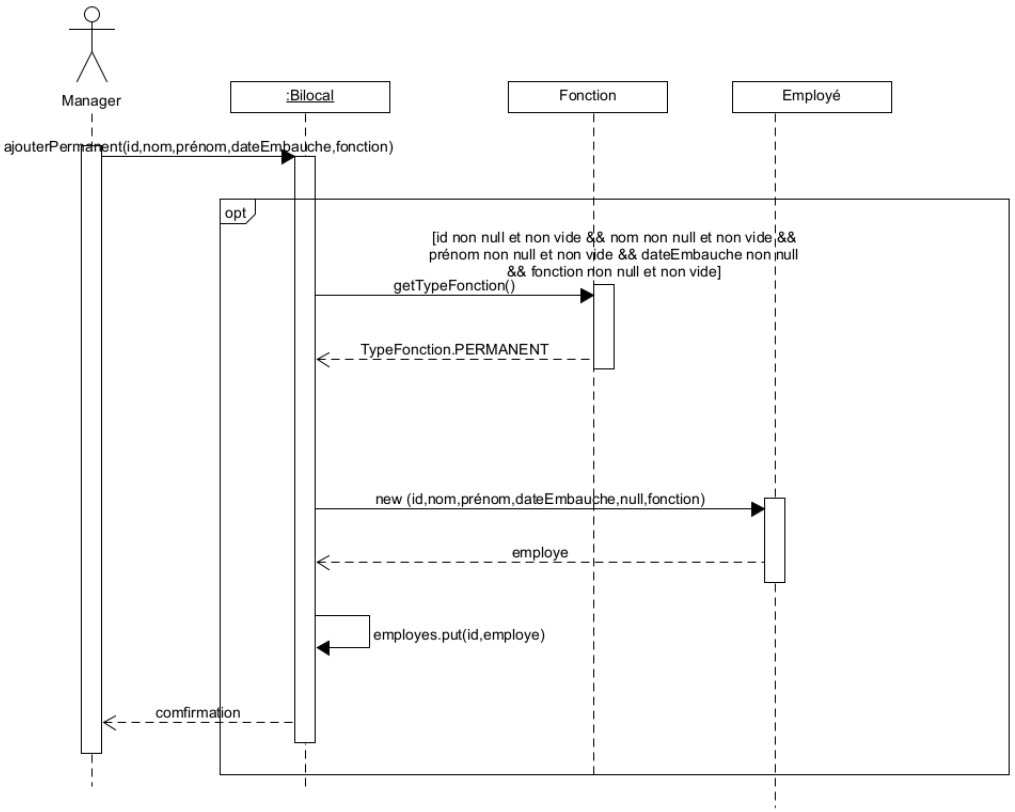

Figure 3: Diagramme de séquence du cas d'utilisation « ajouter un employé permanent »

Voici la description textuelle du cas d'utilisation « ajouter un employé non permanent » :

- **arguments en entrée** : identifiant de l'employé, nom de l'employé, prénom de l'employé, date d'embauche, date de fin de contrat, fonction ;
- **rappel de la précondition** : identifiant non null et non vide $\wedge$ nom non null et non vide $\wedge$ prénom non null et non vide $\wedge$ date d'embauche non null $\wedge$ date de fin de contrat non null $\wedge$ fonction non null $\wedge$ fonction du non permanent $\in$ {doctorat, post-doctorat, ingénierie recherche, stage} $\wedge$ employé avec cet identifiant inexistant ;
- **algorithme** :
  1. vérifier que tous les arguments sont bien formés (non null, non vides, dates valides) ;
  2. vérifier que la fonction correspond à un *type non permanent* : fonction.getTypeFonction() = NON_PERMANENT ;
  3. chercher un employé avec cet identifiant ;
  4. vérifier que l'employé est inexistant ;
  5. instancier l'employé non permanent (avec dateFinContrat) ;
  6. ajouter l'employé dans la collection interne des employés.

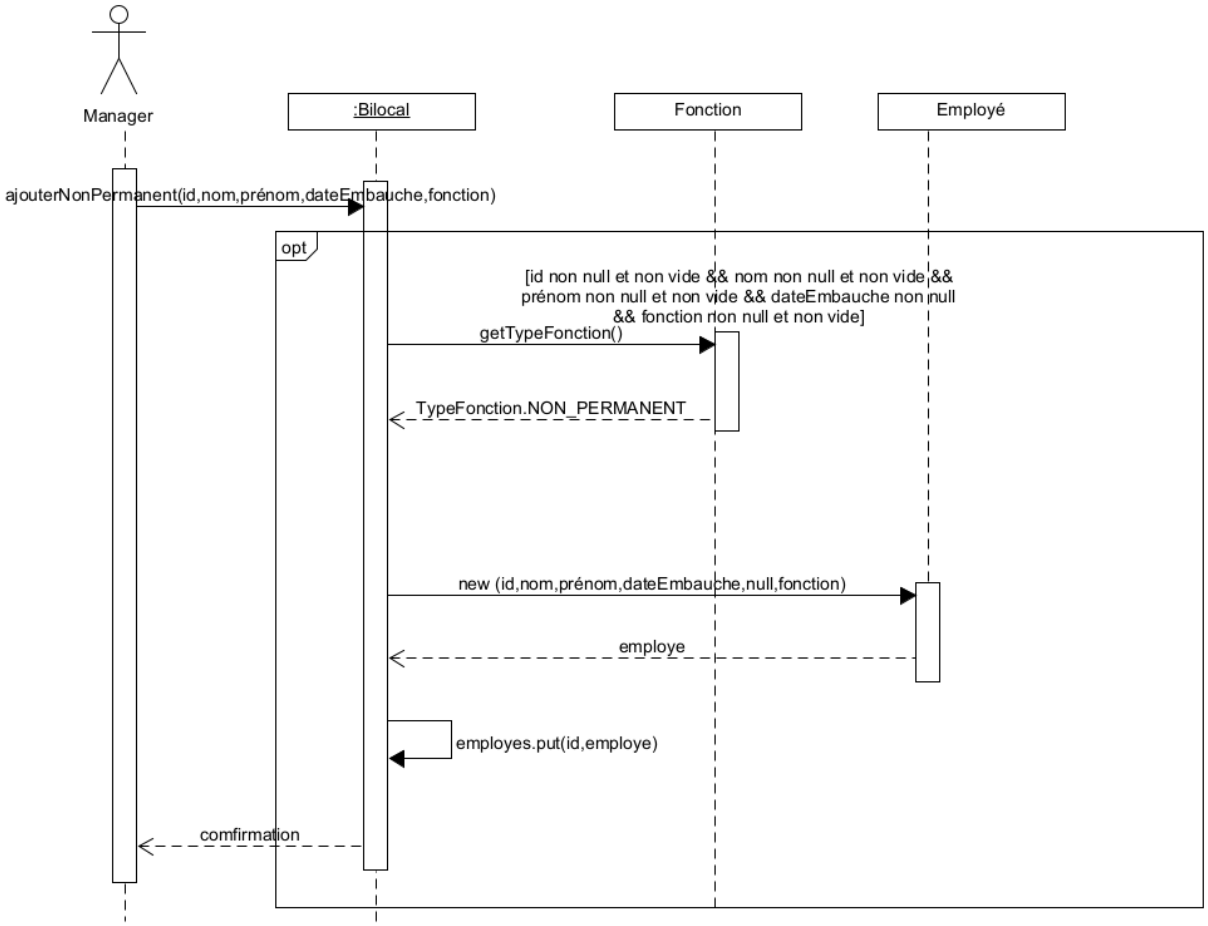

Figure 4: Diagramme de séquence du cas d'utilisation « ajouter un employé non-permanent »

Voici la description textuelle du cas d'utilisation « ajouter un bureau » : **Arguments en entrée** : identifiant du bureau, identifiant du site, type de bureau (permanents ou non permanents), nombre de places fixes, nombre de places de passage.

Préconditions :

- identifiant du bureau bien formé (non null et non vide)
- identifiant du site existant dans le système
- typeBureau non null et $\in$ {PERMANENTS, NON_PERMANENTS}
- $nbPlacesFixes \geq 0$
- $nbPlacesPassage \geq 0$
- **contraintes structurelles** :
  - si typeBureau = PERMANENTS :
    - $nbPlacesFixes = 1$

        * $nbPlacesFixes + nbPlacesPassage \in \{1, 2\}$
    - si typeBureau = NON_PERMANENTS :
        * $nbPlacesFixes + nbPlacesPassage \leq 6$

- identifiant de bureau unique

Algorithme :

1. vérifier la validité de tous les arguments
2. vérifier que le site existe dans la collection des sites
3. vérifier qu'aucun bureau n'existe déjà avec le même identifiant
4. vérifier les contraintes liées au type du bureau :
    - si PERMANENTS : une place fixe obligatoire, total de places égal à 1 ou 2
    - si NON_PERMANENTS : nombre total de places inférieur ou égal à 6
5. rattacher le bureau au site via :

    ```
    site.ajouterBureau(bureau)
    ```

6. ajouter le bureau dans la collection interne des bureaux de `BiLOCAL`
7. créer $nbPlacesFixes$ objets `Place` de type FIXE et les ajouter au bureau
8. créer $nbPlacesPassage$ objets `Place` de type PASSAGE et les ajouter au bureau

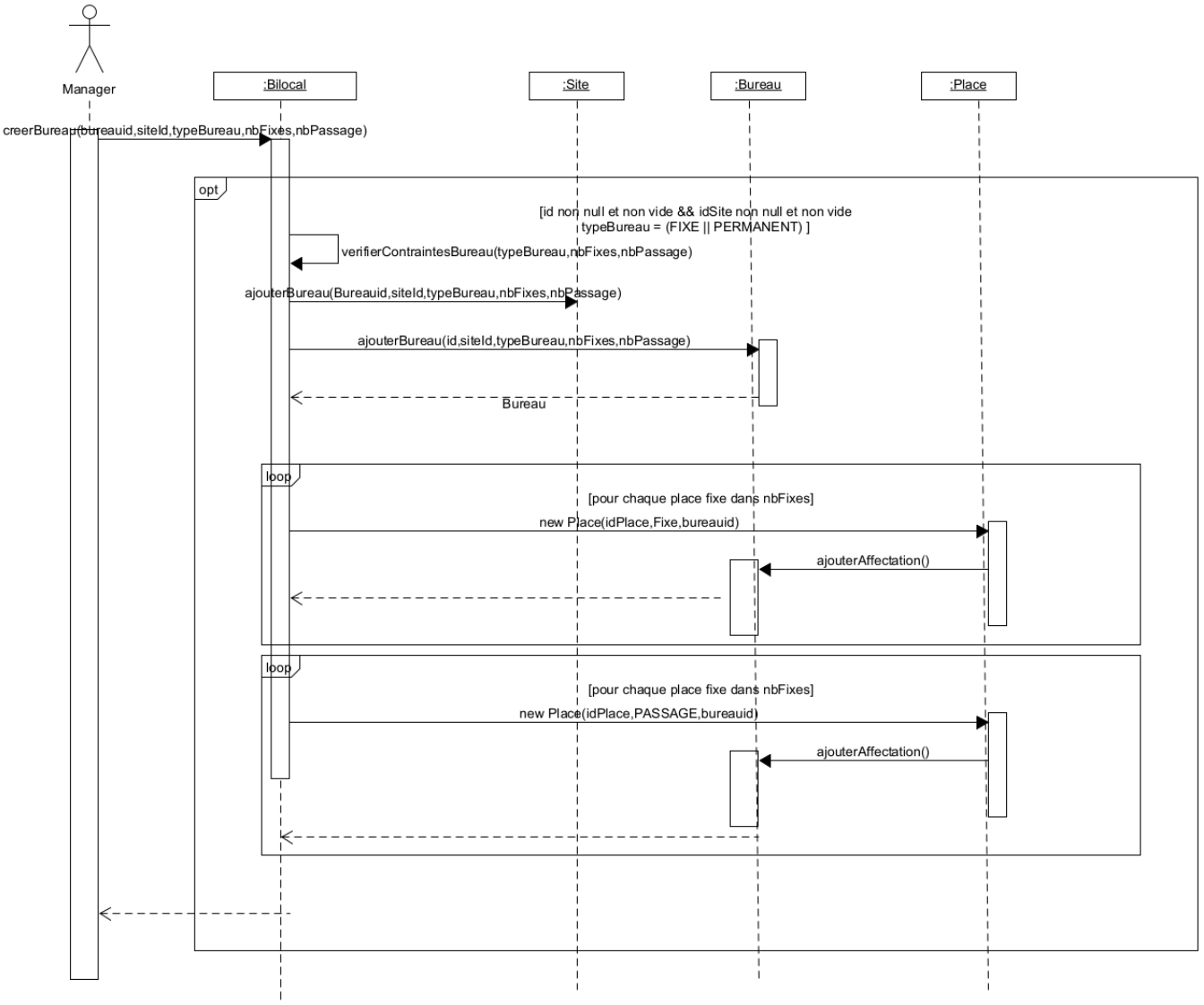

Figure 5: Diagramme de séquence du cas d'utilisation « ajouter un bureau »

Voici la description textuelle du cas d'utilisation « s'affecter une place fixe » : **Arguments en entrée** : place à affecter, date d'affectation. **Rappel de la précondition** : place non null $\wedge$ place de type FIXE $\wedge$ date d'affectation non null $\wedge$ place disponible à cette date (pas déjà affectée) **Algorithme** :

1. vérifier que la place est non null et que son type est FIXE
2. vérifier que la date est bien formée (non null)
3. vérifier via `place.estDisponible(date)` que la place est libre
4. construire une nouvelle affectation avec : id unique, employé courant, place, dateDébut = date, dateFin = null, dateAffectation = aujourd'hui

5. ajouter cette affectation à la place (`place.ajouterAffectation(affectation)`)
6. ajouter cette affectation à l'employé (`employe.affectations.add(affectation)`)

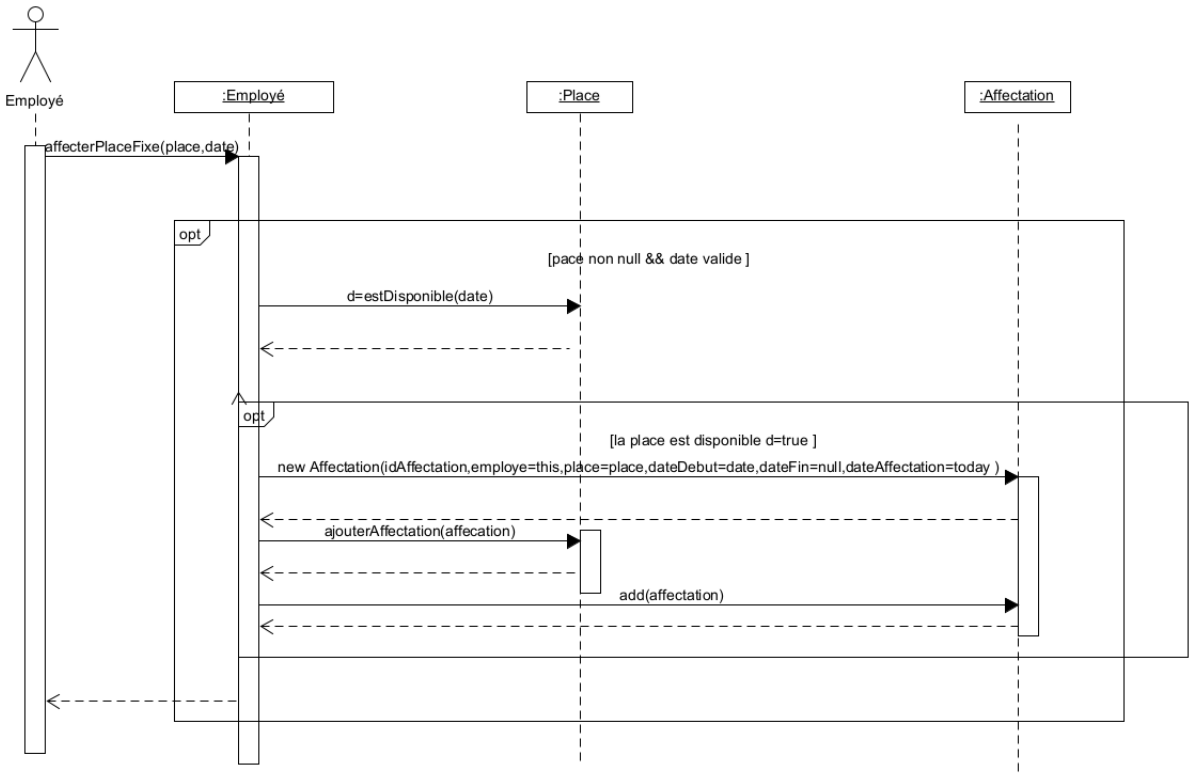

Figure 6: Diagramme de séquence du cas d'utilisation « affecter place fixe »

Voici la description textuelle du cas d'utilisation « s'affecter une place de passage » : **Arguments en entrée** : place à affecter, date de début, date de fin. **Rappel de la précondition** : place non null $\wedge$ place de type PASSAGE $\wedge$ dates non nulles $\wedge$ dateFin $\geq$ dateDebut $\wedge$ place disponible sur la periode demandée **Algorithme** :

1. vérifier que la place est non null et de type PASSAGE
2. vérifier que dateDebut et dateFin sont non nulles
3. vérifier que dateFin $\geq$ dateDebut
4. vérifier via `place.estDisponible(dateDebut, dateFin)` que la période est libre
5. instancier une nouvelle affectation avec : id unique, employé courant, place, dateDébut = dateDebut, dateFin = dateFin, dateAffectation = aujourd'hui
6. ajouter cette affectation à la place (`place.ajouterAffectation(affectation)`)
7. ajouter cette affectation à l'employé (`employe.affectations.add(affectation)`)

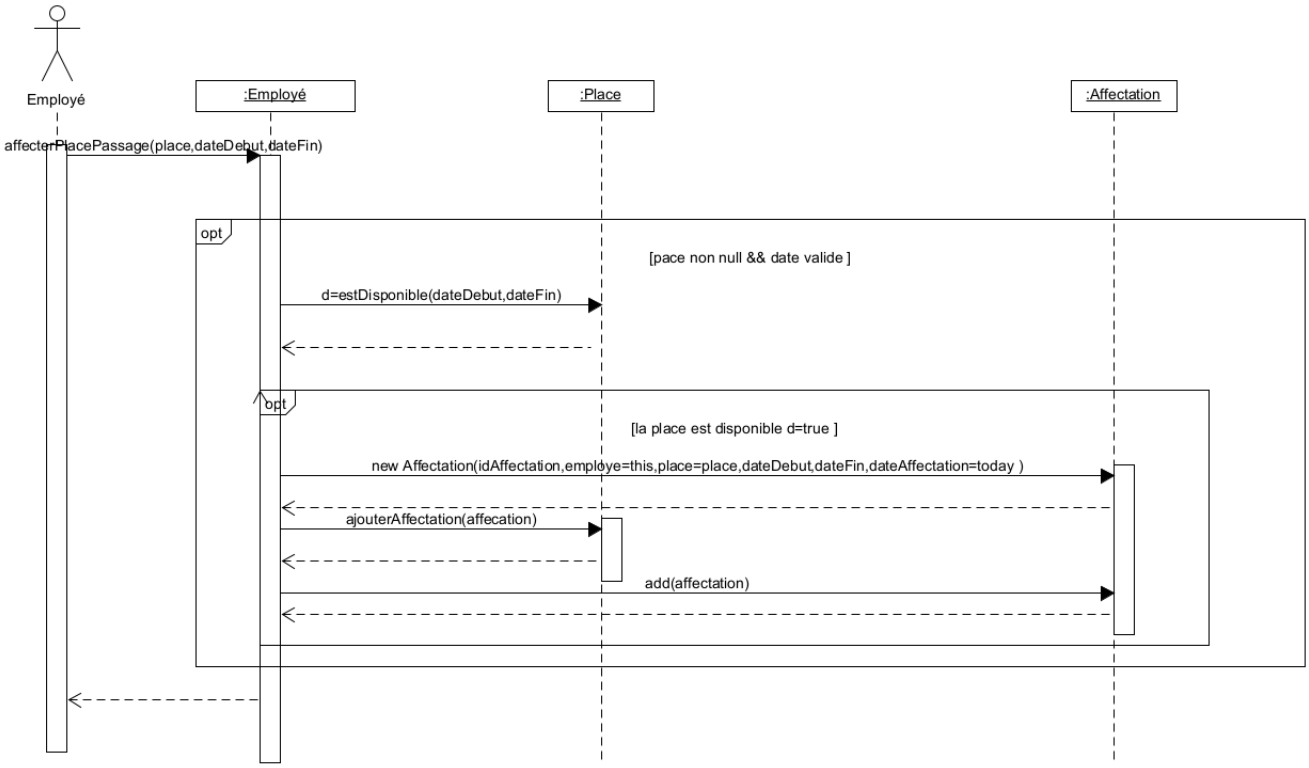

Figure 7: Diagramme de séquence du cas d'utilisation « ajouter place de passage »

4 Fiche des classes

4.1 Classe BiLOCAL

BiLOCAL
<– constantes de classe –> + MAX_PERMANENTS_PAR_BUREAU : int = 2 + MAX_NON_PERMANENTS_PAR_BUREAU : int = 6
<– attributs –> – sites : Map<String, Site> – employes : Map<String, Employe> – affectations : Map<String, Affectation>
<– constructeur –> + BiLOCAL()
<– operations – gestion des sites –> + ajouterSite(String id, String nom) : void + getSite(String id) : Site + listerSites() : List<String>
<– operations – gestion des bureaux –> + ajouterBureauPermanent(String idSite, String idBureau, int nbPlacesFixes, int nbPlacesPassage) : Bureau + ajouterBureauNonPermanent(String idSite, String idBureau, int nbPlacesFixes, int nbPlacesPassage) : Bureau + listerBureaux(String idSite) : List<String>
<– operations – gestion des employés –> + ajouterPermanent(String id, String nom, String prenom, LocalDate dateEmbauche, String fonction) : void + ajouterNonPermanent(String id, String nom, String prenom, LocalDate dateEmbauche, LocalDate dateFinContrat, String fonction) : void + listerEmployes() : List<String> + listerEmployesPermanents() : List<String> + listerEmployesNonPermanents() : List<String>
<– operations – gestion des affectations –> + affecterPlaceFixe(String idEmploye, String idPlace) : Affectation + affecterPlacePassage(String idEmploye, String idPlace, LocalDate dateDebut, LocalDate dateFin) : Affectation + libererPlace(String idAffectation) : void + consulterAffectations(String idEmploye) : List<String> + consulterDisponibilites(String idBureau, LocalDate date) : List<Place> + listerDisponibilites(String idBureau, LocalDate date) : List<String>
<– invariant –> + invariant() : boolean

4.2 Classe Employé

Employé
<– **attributs** –> – id : String (final) – nom : String – prenom : String – dateEmbauche : LocalDate – dateFinContrat : LocalDate (null pour permanent) – fonction : Fonction – affectations : List<Affectation>

<– **constructeurs** –>
+ Employe(String id, String nom, String prenom, LocalDate dateEmbauche, Fonction fonction) // *permanent*
+ Employe(String id, String nom, String prenom, LocalDate dateEmbauche, LocalDate dateFinContrat, Fonction fonction) // *non-permanent*

<– **accesseurs** –> + getId() : String + getNom() : String + getPrenom() : String + getDateEmbauche() : LocalDate + getDateFinContrat() : Optional<LocalDate> + getFonction() : Fonction + getTypeFonction() : TypeFonction + getAffectations() : List<Affectation> + estManager() : boolean
<– **operations – affectation de places** –> + affecterPlaceFixe(Place p, LocalDate date) : Affectation + affecterPlacePassage(Place p, LocalDate dateDebut, LocalDate dateFin) : Affectation + libererPlacePassage(Affectation a) : void
<– **operations – consultation** –> + consulterMesAffectations() : List<String> + consulterMesBureaux() : List<String> + verifierDisponibilite(Site site, LocalDate date) : List<Place>
<– **operations – changement statut** –> + changerNonPermanentVersPermanent(LocalDate dateEmbauche, Fonction fonction) : void
<– **invariant** –> + invariant() : boolean

4.3 Classe Bureau

Bureau
<– **attributs** –> – id : String (final) – typeBureau : TypeBureau

− nbPlacesFixes : int − nbPlacesPassage : int − places : List<Place> − site : Site
<– constructeur –> + Bureau(String id, TypeBureau type, int nbFixes, int nbPassage, Site site)
<– accesseurs –> + getId() : String + getTypeBureau() : TypeBureau + getNbPlacesFixes() : int + getNbPlacesPassage() : int + getPlaces() : List<Place> + getSite() : Site
<– operations – modification –> + modifierNombrePlaces(int nbFixes, int nbPassage) : void + changerTypeBureau(TypeBureau type) : void
<– operations – disponibilités –> + getPlacesDisponibles(LocalDate date, TypePlace typePlace) : List<Place>
<– validation –> + verifierContraintes() : void + invariant() : boolean

4.4 Classe Place

Place
<– attributs –> − id : String (final) − typePlace : TypePlace (final) − bureau : Bureau (final) − affectations : List<Affectation>
<– constructeur –> + Place(String id, TypePlace type, Bureau bureau)
<– accesseurs –> + getId() : String + getTypePlace() : TypePlace + getBureau() : Bureau + getAffectations() : List<Affectation>
<– operations – disponibilités –> + estDisponible(LocalDate date) : boolean + estDisponible(LocalDate dateDebut, LocalDate dateFin) : boolean
<– operations – gestion affectations –> + ajouterAffectation(Affectation a) : void + retirerAffectation(Affectation a) : void
<– invariant –>

+ invariant() : boolean

4.5 Classe Affectation

Affectation
<– attributs –> – id : String (final) – employe : Employe (final) – place : Place (final) – dateDebut : LocalDate – dateFin : LocalDate (null = période ouverte) – dateAffectation : LocalDate (final)

<– constructeurs –>
+ Affectation(String id, Employe emp, Place place, LocalDate debut) // *place fixe*
+ Affectation(String id, Employe emp, Place place, LocalDate debut, LocalDate fin) // *place passage*

<– accesseurs –> + getId() : String + getEmploye() : Employe + getPlace() : Place + getDateDebut() : LocalDate + getDateFin() : Optional<LocalDate> + getDateAffectation() : LocalDate
<– operations – état –> + estActive(LocalDate date) : boolean
<– operations – modification période –> + cloturerPeriode(LocalDate dateFin) : void + modifierDateDebut(LocalDate nouvelleDateDebut) : void + modifierDateFin(LocalDate nouvelleDateFin) : void
<– validation –> + verifierContraintesTemporelles() : void + invariant() : boolean

4.6 Classe Site

Site
<– constantes de classe –> – NB_SITES_MAX : int = 2
<– attributs de classe –> – nbSitesCrees : int // *compteur partagé*
<– attributs –> – id : String (final)

− nom : String − adresse : String − bureaux : List<Bureau>
<– constructeur –> + Site(String id, String nom, String adresse)
<– méthodes de classe –> + reinitialiserCompteur() : void *// pour tests*
<– accesseurs –> + getId() : String + getNom() : String + getAdresse() : String + getBureaux() : List<Bureau>
<– operations – gestion bureaux –> + ajouterBureau(Bureau b) : void + getBureauxDisponibles(LocalDate date) : List<Bureau>
<– invariant –> + invariant() : boolean

5 Diagrammes de machine à états et invariants

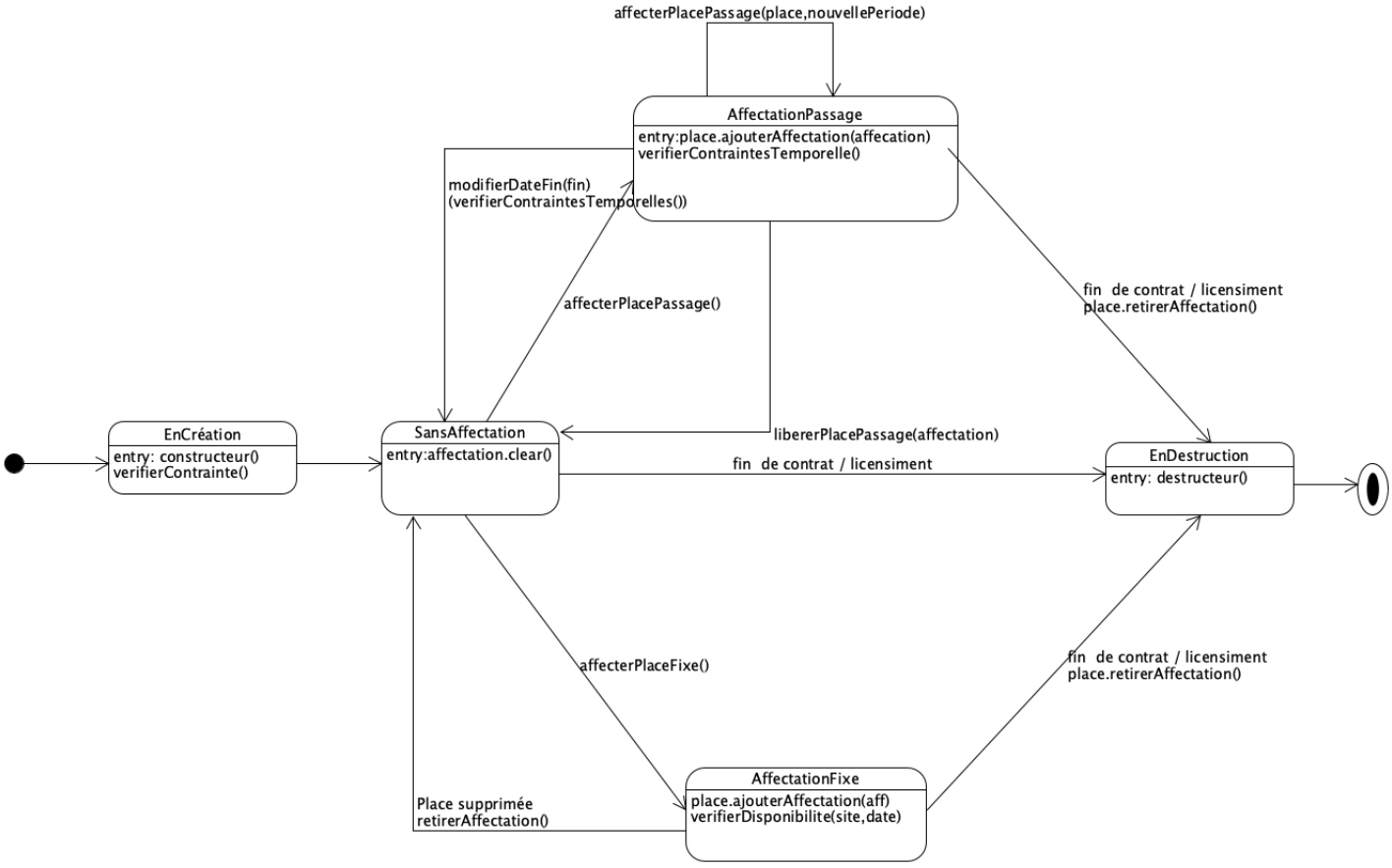

Figure 8: Diagramme de machine à état pour la classe Employé

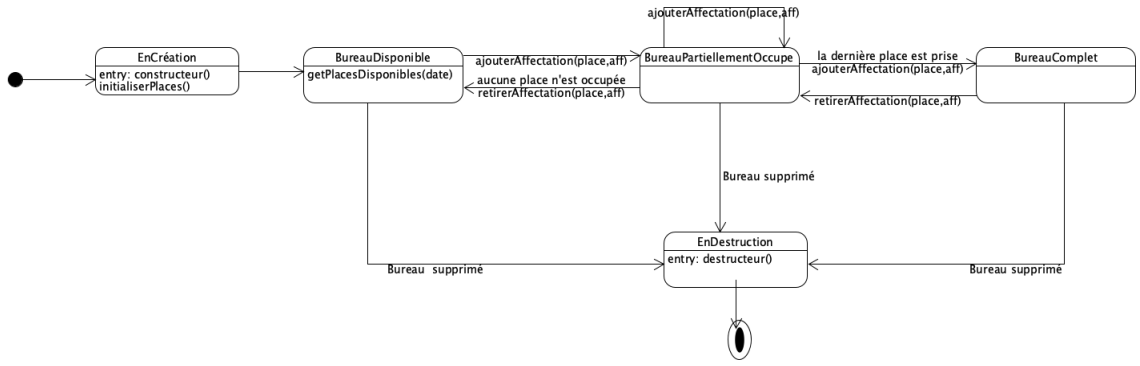

Figure 9: Diagramme de machine à état pour la classe Bureau

6 Préparation des tests unitaires

6.1 Classe Employe

6.1.1 Constructeur permanent

Numéro de test	1	2	3	4	5	6	7	8
id valide ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T
nom valide ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T
prenom valide ($\neq$ null $\wedge \neq$ vide)			F	T	T	T	T	T
dateEmbauche $\neq$ null				F	T	T	T	T
fonction $\neq$ null					F	T	T	T
fonction.getTypeFonction() = PERMANENT						F	T	T
Création réussie	F	F	F	F	F	F	T	T
Nombre de jeux de test	2	2	2	1	1	4	4	-

Table 12: Table de décision pour le constructeur permanent de Employe

6.1.2 Constructeur non-permanent

Numéro de test	1	2	3	4	5	6	7	8	9
id valide ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T	T
nom valide ($\neq$ null $\wedge \neq$ vide)		F	T	T	T	T	T	T	T
prenom valide ($\neq$ null $\wedge \neq$ vide)			F	T	T	T	T	T	T
dateEmbauche $\neq$ null				F	T	T	T	T	T
dateFinContrat $\neq$ null					F	T	T	T	T
dateFinContrat $\geq$ dateEmbauche						F	T	T	T
fonction $\neq$ null							F	T	T
fonction.getTypeFonction() = NON_PERMANENT								F	T
Création réussie	F	F	F	F	F	F	F	F	T
Nombre de jeux de test	2	2	2	1	1	1	1	4	4

Table 13: Table de décision pour le constructeur non-permanent de Employe

6.1.3 Méthode affecterPlaceFixe

Numéro de test	1	2	3	4	5	6
place $\neq$ null	F	T	T	T	T	T
dateDebut $\neq$ null		F	T	T	T	T
place.getTypePlace() = FIXE			F	T	T	T
Employé normal : pas déjà une place fixe				F	T	-
Manager : pas de place fixe sur ce site					F	T
Affectation réussie	F	F	F	F	F	T
Nombre de jeux de test	1	1	1	1	2	2

Table 14: Table de décision pour la méthode affecterPlaceFixe de Employe

6.2 Classe Bureau

6.2.1 Constructeur

Numéro de test	1	2	3	4	5	6	7	8	9
id valide ($\neq$ null $\wedge \neq$ vide)	F	T	T	T	T	T	T	T	T
typeBureau $\neq$ null		F	T	T	T	T	T	T	T
site $\neq$ null			F	T	T	T	T	T	T
nbPlacesFixes ≥ 0				F	T	T	T	T	T
nbPlacesPassage ≥ 0					F	T	T	T	T
Si PERMANENTS : nbTotal $\in \{1, 2\}$ et nbFixes = 1						F	T	-	-
Si NON_PERMANENTS : nbTotal $\leq$ 6 et nbTotal ≥ 1							-	F	T
Création réussie	F	F	F	F	F	F	T	F	T
Nombre de jeux de test	2	1	1	1	1	2	2	2	5

Table 15: Table de décision pour le constructeur de Bureau