Avenue Paul Langevin

59655 Villeneuve d'Ascq cedex

PROJET DE PPO
AGENDAS DE TACHES

Projet réalisé par:

Yizhou LIN

Aimée UMUHOZA

Tuteur:

Walter RUDAMETKIN

Département Génie informatique et statistique 4A

Année : 2015-2016

Sommaire

Introduction 2

I. Schéma de conception UML 3

II. Analyse et conception 4

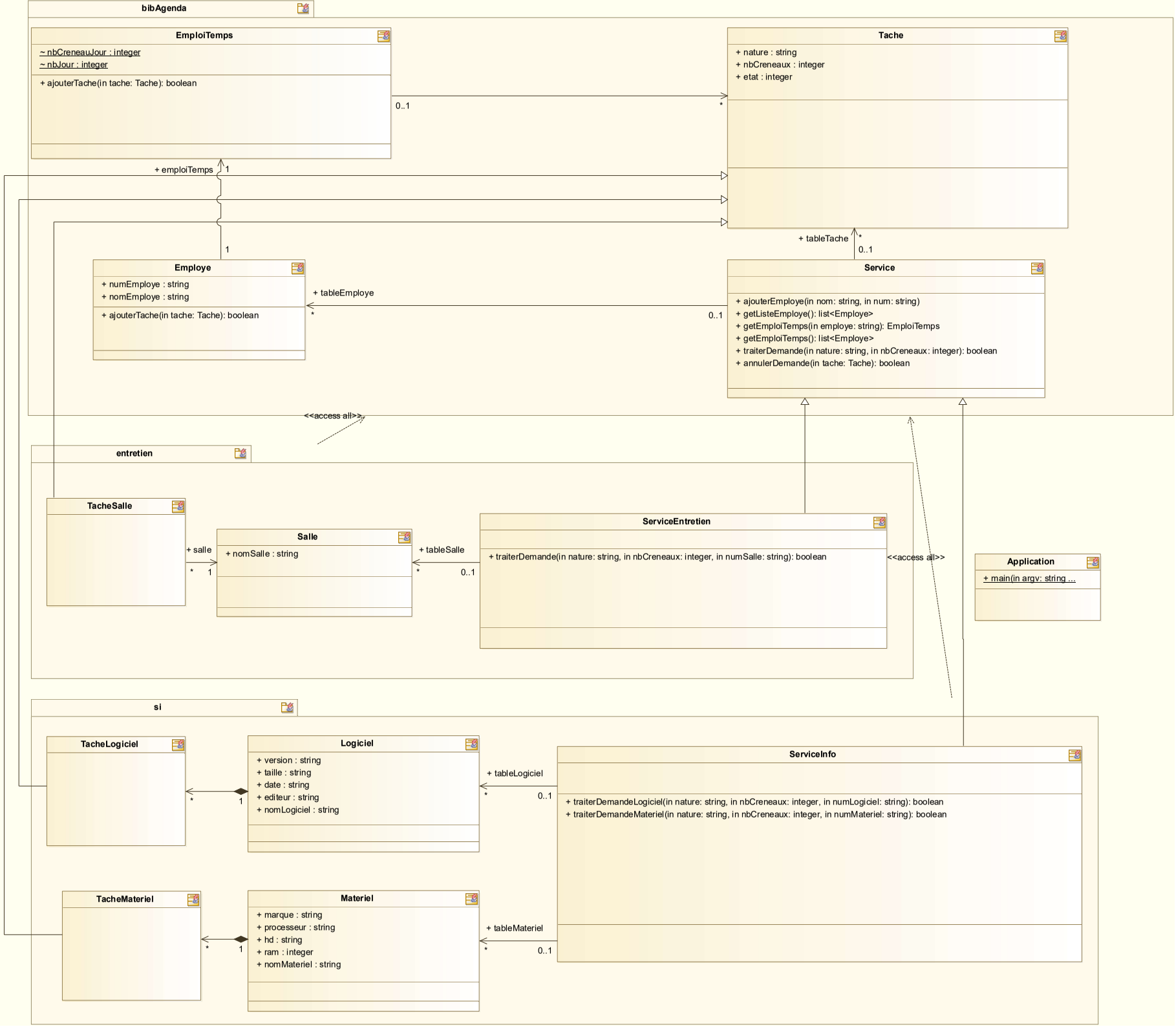
- Service 4
- EmploiTemps 5

III. Méthodes essentielles 6

- Traitement de demande 6
- Annulation d'une demande 7

Introduction

Au cours de ce projet, nous travaillons à la conception de logiciels d'agendas destinés à des services devant répondre à des demandes de travaux. L'objectif de ce projet est de mettre en place un logiciel résolvant les problèmes de gestion d'agendas de tâches. Pour résoudre ce projet, nous avons conçu une bibliothèque de classes générale ainsi que ses différentes applications dans un schéma UML. Dans ce rapport nous allons donc décrire les principales fonctionnalités de nos méthodes utilisées par rapport au classées concernées.


bibAgenda
EmploiTemps
~ nbCreneauJour : integer
~ nbJour : integer
+ ajouterTache(in tache: Tache): boolean
0..1
*
Tache
+ nature : string
+ nbCreneaux : integer
+ etat : integer
+ emploiTemps
1
1
+ tableTache
*
0..1
Employe
+ numEmploye : string
+ nomEmploye : string
+ ajouterTache(in tache: Tache): boolean
+ tableEmploye
*
0..1
Service
+ ajouterEmploye(in nom: string, in num: string)
+ getListeEmploye(): list<Employe>
+ getEmploiTemps(in employe: string): EmploiTemps
+ getEmploiTemps(): list<Employe>
+ traiterDemande(in nature: string, in nbCreneaux: integer): boolean
+ annulerDemande(in tache: Tache): boolean
<<access all>>
entretien
TacheSalle
+ salle
*
1
Salle
+ nomSalle : string
+ tableSalle
*
0..1
ServiceEntretien
+ traiterDemande(in nature: string, in nbCreneaux: integer, in numSalle: string): boolean
<<access all>>
Application
+ main(in argv: string ...
si
TacheLogiciel
*
1
Logiciel
+ version : string
+ taille : string
+ date : string
+ editeur : string
+ nomLogiciel : string
+ tableLogiciel
*
0..1
ServiceInfo
+ traiterDemandeLogiciel(in nature: string, in nbCreneaux: integer, in numLogiciel: string): boolean
+ traiterDemandeMateriel(in nature: string, in nbCreneaux: integer, in numMateriel: string): boolean
TacheMateriel
*
1
Materiel
+ marque : string
+ processeur : string
+ hd : string
+ ram : integer
+ nomMateriel : string
+ tableMateriel
*
0..1

II. Analyse et conception

Il nous a été demandé de répartir des tâches aux employés suivant leur disponibilité sachant qu'une tâche doit être traitée pendant n créneaux consécutif.

Nous avons donc choisi de créer quatre classes « `Service` », « `Tache` », « `Employé` » et « `emploiTemps` » dans une bibliothèque nommé **`bibAgenda`**, classes « `ServiceInfo` », « `Logiciel` », « `Matériel` », « `TacheLogiciel` » et « `TacheMateriel` » qui se trouvent dans le package **si** et trois classes « `ServiceEntretien` », « `TacheSalle` » et « `Salle` » dans le package **`entretien`**.

● Service

Service est une classe générique ayant comme sous-classes `ServiceEntretien` qui s'occupe des salles, et `ServiceInfo` qui s'occupe des tâches logicielles ainsi que des tâches matérielles. Elle contient les variables et méthodes partagent par se sous-classes.

Dans cette classe, nous avons huit méthodes :

- Nous avons une méthode **« `ajouterEmploye` »** dans la classe service, elle prend en entrée le nom et le numéro de l'employé. Cette méthode nous permet d'ajouter les employés au service.

- **« `getListeEmployes` »** ne prend rien en paramètre et nous permet d'afficher tous les employés. Donc elle retourne une liste d'employés.

- **« `getEmployeTemps` »** prend en entrée le nom de l'employé, ensuite elle affiche l'emploi du temps de cet employé.

- **« `getEmployeTemps` »** : ne prend rien en paramètre et nous permet d'afficher les emplois du temps de tous les employés.

- **« `traiterDemande` »** cette méthode nous permet de traiter une demande de tâche quelconque car elle prend en entrée la nature de la tâche à traiter ainsi que le nombre de créneaux nécessaire pour traiter cette demande et elle fait appel

à la méthode ajouterTache, celle-ci permet de créer une tache ensuite l'ajouter dans un emploi du temps d'un employé. Elle retourne vrai si la demande a été traitée c'est-à-dire la tâche a été créée faux sinon.

➢ **« annulerDemande »** nous permet d'annuler une demande donnée, elle prend en paramètre la tâche a annulé ensuite elle fait appel à la fonction « annuler » définie dans la classe tache. Cette fonction retourne vrai si la demande a été annulée faux sinon.

- ## EmploiTemps

➢ Avant de traiter une tâche, il faut que cette dernière soit créée. Donc dans cette classe nous avons un constructeur « `EmploiTemps` » pour de créer un nouveaux emploi du temps. La matrice de tâche va être initialisée à `null`. Les tâches créées par les services des types différents, soit une tâche d'entretien, soit une tâche de service informatique, soit une tâche générique, va être insérés dans cette matrice.

➢ Nous avons également une méthode **« `ajouterTache` »** cette méthode est paramétrée par une tâche. Elle va essayer d'ajouter une tâche dans l'emploi du temps. Si cette opération est réussie la méthode va retourner vrai sinon elle va retourner faux.

III. Méthodes essentielles

• Traitement de demande

Nous allons d'abord créer une tâche par rapport à sont type, soit une tâche d'entretien, soit une tâche de service informatique, soit une tâche générique.

Nous parcourons le tableau des employés pour chercher un employé disponible pour le nombre de créneaux donnés. Pour ce faire, nous allons faire l'appel à la méthode « `ajouterTache` » dans l'emploi du temps de tous les objets de types Employé. Elle est retourne une variable booléenne pour confirmer si la tâche a été bien ajoutée avec succès. Ainsi la tâche sera ajoutée dans un tableau de tâche

Les tâches non traitées sont annulées sauf les tâches matérielles qui seront mise en attente en attendant qu'il y ait un créneau qui se libère.

Pseudo-code de la méthode `ajouterTache(Tache)` utilisée :

`Classe :EmploiTemps`
`Méthode :ajouterTache(Tache) :booléen`

`Donne de la classe :`
`nbCreneauJour :int //nombre de créneaux par jour`
`nbJour : int //nombre de jours dans une semaine`
`Donner : tache : Tache //la tâche a ajouter`
`Local :`
`nbLibre : int //nombre de creneaux livre déjà trouvé`
`ptJour, ptCre : int //variable de boucle`
`flagTrouve : booleen initialisé à false,`
`changer à vrai si la tâche est traitée`

`Tantque flagTrouve == false et ptJour < nbJour faire`
`nbLibre = 0; // nombre de creneaux libre`
`ptCre = 0;`
`Tantque flagTrouve == false et ptCre<nbCreneauJour faire`
`Si un créneaux est libre ou la tache a ete annulée alors`
`nbLibre = nbLibre+1`
`Si nbLibre >= nombre de créneau de tache`

```
      et pour tous les ressources dans
              la tableau de ressources de tache sont libre
      alors
        tableTache[ptJour][ptCre-tache.nbCreneaux+1] = tache;
        tache.etat = 1;
        flagTrouve = true;
      Fin Si
      ptCre = ptCre + 1
    Sinon
      nbLibre = 0;
      ptCre = ptCre + tableTache[ptJour][ptCre].nbCreneaux;
    Fin Si
  Fait
  ptJour = ptJour+1;
Fait
return flagTrouve;
```

Annulation d'une demande

Pour traiter cette partie, nous avons créé une méthode **« annuler »** ne prend rien en paramètre et elle nous permet d'annuler n'importe quelle tâche soit une tâche logicielle, soit une tâche salle, soit une tâche matérielle. Dans la classe « Tâche » nous avons une variable **< état >** qui va nous permettre de savoir si une tâche est en attente d'être traitée (par exemple une tâche matérielle), ou si elle a été annulée ou si elle a été traitée.

Voici un tableau qui résume les valeurs prises par la variable état :

Valeur	Sens
0	Tâche initialisée, en attente d'être traitée
1	Tâche traitée
-1	Tâche annulée

Cette méthode utilise le pseudo langage suivant :

Classe :Tache
annuler () : booléen
Local : resultat : booléen

boolean resultat = (etat==0);
//Pour vérifier si la tâche est annulée
//si la tâche est déjà annulée la méthode retourne faux
etat = -1;
return resultat;