Avenue Paul Langevin

59655 Villeneuve d'Ascq cedex

PROJET DE PPO

AGENDAS DE TACHES

Projet réalisé par:

Yizhou LIN

Aimée UMUHOZA

Tuteur:

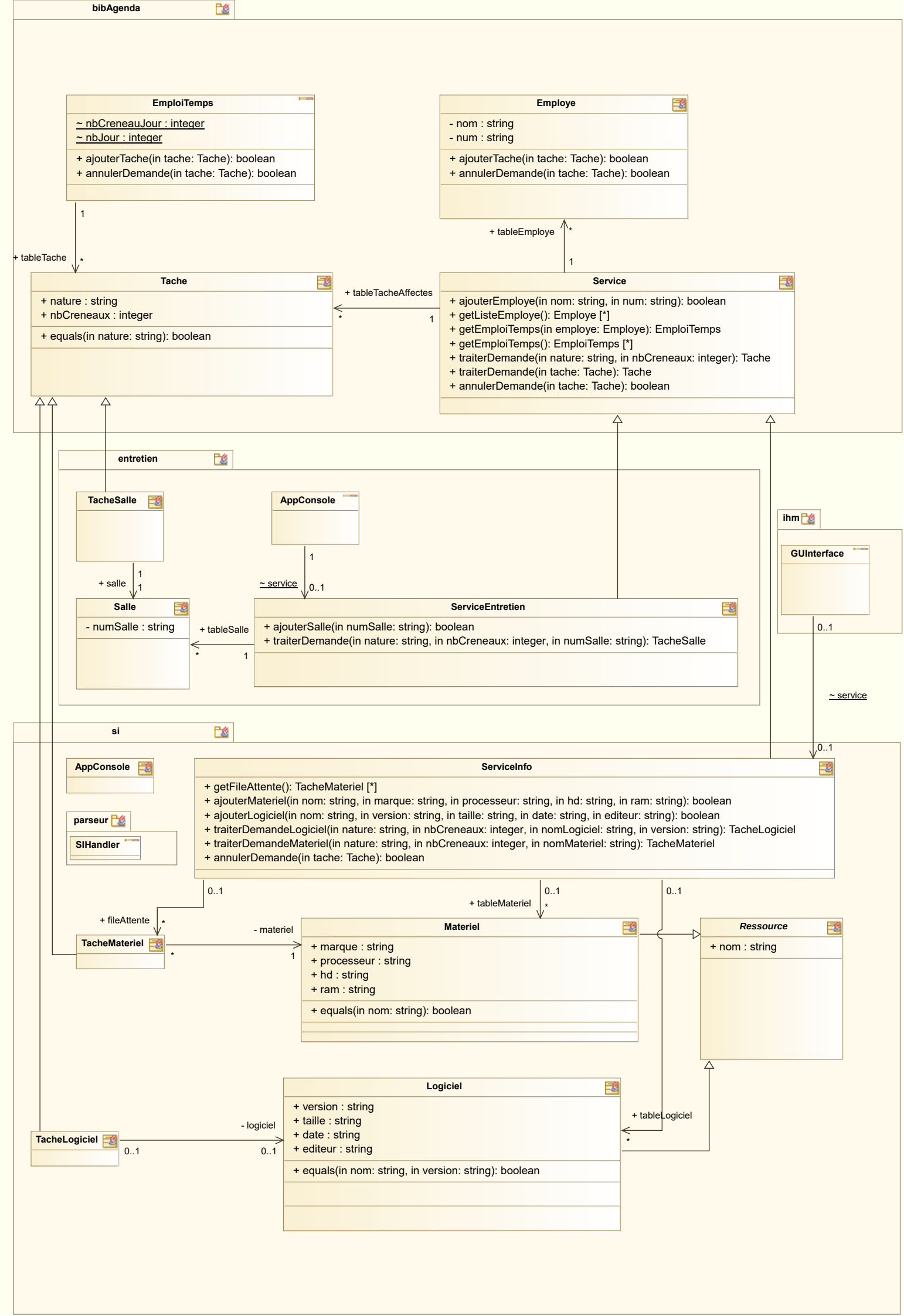
Walter RUDAMETKIN

Département Génie informatique et statistique Année : 2015-2016

Sommaire

Introduction

Après avoir réalisé un premier travail d'analyse et de conception, des modifications ont été apportées quant aux choix de nos structures de données, des méthodes utilisées ainsi qu'au niveau du schéma de conception UML. Dans ce rapport ci nous allons donc détailler les changements apportés ainsi que la démarche adoptée pour mener à bien le projet. Dans une moindre mesure nous évoquerons certaines difficultés rencontrées et les améliorations possibles. Ainsi ce rapport se veut être un complément au rapport d'analyse et de conception


bibAgenda
EmploiTemps
~ nbCreneauJour : integer
~ nbJour : integer
+ ajouterTache(in tache: Tache): boolean
+ annulerDemande(in tache: Tache): boolean
Employe
- nom : string
- num : string
+ ajouterTache(in tache: Tache): boolean
+ annulerDemande(in tache: Tache): boolean
1
+ tableTache
*
+ tableEmploye
*
1
Tache
+ nature : string
+ nbCreneaux : integer
+ equals(in nature: string): boolean
+ tableTacheAffectes
*
1
Service
+ ajouterEmploye(in nom: string, in num: string): boolean
+ getListeEmploye(): Employe [*]
+ getEmploiTemps(in employe: Employe): EmploiTemps
+ getEmploiTemps(): EmploiTemps [*]
+ traiterDemande(in nature: string, in nbCreneaux: integer): Tache
+ traiterDemande(in tache: Tache): Tache
+ annulerDemande(in tache: Tache): boolean
entretien
TacheSalle
AppConsole
1
+ salle
1
1
~ service
0..1
Salle
- numSalle : string
+ tableSalle
*
1
ServiceEntretien
+ ajouterSalle(in numSalle: string): boolean
+ traiterDemande(in nature: string, in nbCreneaux: integer, in numSalle: string): TacheSalle
ihm
GUInterface
0..1
~ service
0..1
si
AppConsole
parseur
SIHandler
ServiceInfo
+ getFileAttente(): TacheMateriel [*]
+ ajouterMateriel(in nom: string, in marque: string, in processeur: string, in hd: string, in ram: string): boolean
+ ajouterLogiciel(in nom: string, in version: string, in taille: string, in date: string, in editeur: string): boolean
+ traiterDemandeLogiciel(in nature: string, in nbCreneaux: integer, in nomLogiciel: string, in version: string): TacheLogiciel
+ traiterDemandeMateriel(in nature: string, in nbCreneaux: integer, in nomMateriel: string): TacheMateriel
+ annulerDemande(in tache: Tache): boolean
0..1
+ fileAttente
*
0..1
+ tableMateriel
*
0..1
TacheMateriel
- materiel
*
1
Materiel
+ marque : string
+ processeur : string
+ hd : string
+ ram : string
+ equals(in nom: string): boolean
Ressource
+ nom : string
Logiciel
+ version : string
+ taille : string
+ date : string
+ editeur : string
+ equals(in nom: string, in version: string): boolean
+ tableLogiciel
*
TacheLogiciel
- logiciel
0..1
0..1

II. Structure de données

Pour notre projet, nous avons choisis d'utiliser comme collection le List car on peut stocker autant d'éléments qu'on souhaite sans craindre de dépasser la taille du tableau et il nous permet d'accéder à chaque élément de la liste via son indice.

Cependant, nous avons besoin de stocker les employés, les matériels, les logiciels ainsi que les tâches qui sont en attente.

De plus dans le List les doublons sont autorisés car on peut avoir deux ou plusieurs employés possédant le même nom ou des tâches possédant la même nature.

Nous avons trois types d'objets list Vector, LinkedList et ArrayList, nous avons donc choisi les ArrayList car ils sont plus rapides et plus simple en lecture contrairement aux deux autres.

Nous avons également pensé utiliser la collection Map, et choisir le HashMap car pour chercher un employé par son nom, la méthode get() de HashMap a une complexité en O(1) c'est à dire constante. Cependant nous n'avons pas eu assez de temps.

- `tableEmploye = new ArrayList<Employe>() : Permet de stocker tous les employés`
- `tableTacheAffectes = new ArrayList<Tache>() : Permet de stocker les tâches affectées`
- `tableSalle = new ArrayList<Salle>() : permet de stocker toutes les salles`
- `tableLogiciel = new ArrayList<Logiciel>() : permet de stocker tous les logiciels`
- `tableMateriel = new ArrayList<Materiel>() : permet de stocker tous les matériels`
- `fileAttente = new ArrayList<TacheMateriel>() : permet de stocker les tâches matérielles qui sont en attente.`

Nous avons également une matrice `Tache[][] tableTache` qui contient l'emploi du temps d'un employé.

III. Méthodes essentielles

a. Traitement d'une demande

○ Service :

Pour traiter une demande par sa nature et ses nombre de créneaux on implémente la méthode `Tache traiterDemande(String nature,int nbCreneaux)` qui prend en entrée la nature de l'intervention et le nombre de créneaux nécessaire pour une tâche quelconque.

```
public Tache traiterDemande(String nature,int nbCreneaux){
            Tache tache = new Tache(nature,nbCreneaux);
            return traiterDemande(tache);}
```

Cette méthode fait appel à la méthode `traiterDemande(tache)` qui prend en paramétrée la tache créée.

```
      public Tache traiterDemande(Tache tache){
            for (Employe E : tableEmploye){
                  if(E.ajouterTache(tache)){
                        tableTacheAffectes.add(tache);
                        return tache;}
            }return null;
```

Ensuite elle parcourt la liste des employés en cherchant un employé disponible pour la tâche donnée. Pour chaque employé, on vérifie si c'est possible d'ajouter cette tâche dans son emploi du temps grâce à la méthode `ajouterTache(tache)` implémente dans la classe **EmploiTemps**, elle prend en entrée la tâche a affecté et retourne une variable booléenne pour confirmer l'ajout ou non de cette tâche.

On cherche un employé et un créneau disponible, Des que l'employé est trouvé la recherche s'arrête ainsi la tâche est ajoutée dans `tableTacheAffectes,` sinon la tâche est rejetée sauf si c'est **une tâche matérielle**, celle-ci est mise dans la liste d'attente en attendant un créneau qui se libéré.

○ Service d'entretien :

Pour traiter une tâche salle, nous avons implémenté la méthode `TacheSalle traiterDemande(String nature,int nbCreneaux,String numSalle)` qui prend en entrée

la nature de l'intervention, le nombre de créneaux nécessaire pour cette tâche et le numéro de la salle dans laquelle il faut intervenir.

```
public TacheSalle traiterDemande(String nature,int nbCreneaux,String
numSalle)throws SalleNotFoundException{
            Salle salle = null;
            for(Salle s : tableSalle){
                  if(s.getNumSalle().equals(numSalle)){salle = s;}
            }
            if(salle == null){throw new SalleNotFoundException();
            }TacheSalle tache = new TacheSalle(nature,nbCreneaux,salle);
            return (TacheSalle) traiterDemande(tache);
      }}
```

On vérifie d'abord si cette salle existe, ensuite une tâche est créée, elle sera traitée de la même manière que pour la partie service, car cette méthode fait appel à la méthode `traiterDemande(tache)`, elle-même fait appel à la méthode `ajouterTache(tache)` implémente dans la classe **EmploiTemps** qui nous permet de chercher un employé et un créneau disponible pour une tâche donnée. De plus elle retourne une tâche salle.

○ Service informatique :

Pour traiter une demande de travail concernant **une tâche logicielle**, on implémente la méthode `TacheLogiciel traiterDemandeLogiciel(String nature,int nbCreneaux,String nomLogiciel,String version)`, elle prend en entrée la nature de l'intervention, le nombre de créneaux nécessaire pour traiter cette tâche, ensuite le nom du logiciel sur lequel il faut intervenir et son numéro de version.

```
public TacheLogiciel traiterDemandeLogiciel(String nature,int
nbCreneaux,String nomLogiciel,String version) throws
LogicielNotFoundException{
            Logiciel logiciel = null;
            for(Logiciel l : tableLogiciel){
                  if(l.equals(nomLogiciel,version)){logiciel = l;break;}
            }if(logiciel == null){throw new LogicielNotFoundException();}
TacheLogiciel tache = new TacheLogiciel(nature, nbCreneaux,
logiciel);return (TacheLogiciel) traiterDemande(tache);}
```

On parcourt la liste des logiciels pour vérifier si le nom et la version du logiciel donné existe, si oui on affecte ce logiciel à la variable logicielle ensuite une tâche logicielle est créée et elle sera traitée de la même manière qu'une tâche salle car cette méthode fait appel à la

méthode `traiterDemande(tache)` qui retourne une tâche logicielle si celle-là a été traitée. Sinon la tâche est rejetée.

Pour traiter une demande de travail concernant **une tâche matérielle**, on implémente la méthode `TacheMateriel traiterDemandeMateriel(String nature,int nbCreneaux,String nomMateriel)` elle prend en entrée la nature de l'intervention, le nombre de créneaux nécessaire pour traiter cette tâche et le nom du matériel c'est-à-dire la machine sur laquelle il faut intervenir.

```
public TacheMateriel traiterDemandeMateriel(String nature,int nbCreneaux,String nomMateriel) throws MaterielNotFoundException{
            Materiel materiel = null;
            for(Materiel m : tableMateriel){if(m.equals(nomMateriel) ){materiel = m;break;}}
            if(materiel == null){throw new MaterielNotFoundException();}
TacheMateriel tache = new TacheMateriel(nature, nbCreneaux, materiel);
TacheMateriel tacheTraite =  (TacheMateriel) traiterDemande(tache);
            if(tacheTraite==null){this.fileAttente.add(tache);}
            return tacheTraite;}
```

On parcourt la liste des matériels pour vérifier si le matériel donné existe, si oui une tâche matérielle est créée, ensuite elle sera traitée de la même manière qu'une tâche salle ou logicielle. Contrairement aux autres tâches une tâche matérielle non traitée est enregistrée dans une file d'attente `fileAttente`. Lors d'une annulation de demande de travail, on essaie de traiter des tâches matérielles mise en attente.

b. Annulation d'une demande

Service

Dans cette nous avons implémenté la méthode **`annulerDemande(Tache tache)`** dans la classe **Service,** elle prend en entrée la tâche a annulé.

```
public boolean annulerDemande(Tache tache){
            for (Employe E : tableEmploye){
                  if(E.annulerDemande(tache)){
                        tableTacheAffectes.remove(tache);
                        return true;}}
            return false;}}
```

Elle fait appel à la méthode **annulerDemande(Tache tache)** implémentée dans la classe **EmploiTemps** qui va annuler la tâche donnée dans l'emploi du temps de l'employé (la matrice des tâches) affecté à cette tâche.Ensuite la tâche est également supprimé dans la liste des tâches affectées.

```
public boolean annulerDemande(Tache tache) {
      for (int ptJour = 0; ptJour < nbJour; ptJour++) {
      for (int ptCre = 0; ptCre < nbCreneauJour; ptCre++) {
            if (tache.equals(tableTache[ptJour][ptCre])) {
                  tableTache[ptJour][ptCre] = null;
                  return true;}}}
return false;}}
```

Ce deux méthode retourne vrai si la tâche est annulée, faux sinon.

○ Service d'entretien

Ici cela se fait de la même manière que dans service car nous utilisons les méthodes d'annulations se trouvant dans la sur - classe Service, nous allons annuler la tâche donnée dans la matrice des tâches dans la classe **EmploiTemps**, ensuite dans l'emploi du temps de l'employé affecté à la tâche, puis dans la liste des tâches affectées.

○ Service informatique

Dans ce service, nous avons deux types d'annulations de travail de demande suivantes :

- Annulation d'une tâche logicielle qui se fait de la même manière que l'annulation d'une tâche salle.
- Annulation d'une tâche matérielle se fait également de la même manière que les autres tâches, sauf au cas ou la tâche se trouve seulement dans la file d'attente, celle-ci est annulée seulement dans la file d'attente.

Ensuite, on parcourt la liste d'attente des tâches matérielles et on essaie de traiter une demande déjà en attente. Si la tâche est traitée, celle-ci est effacée dans la liste d'attente.

```
public boolean annulerDemande(Tache tache){
      if(!super.annulerDemande(tache)){
      if(tache instanceof TacheMateriel)fileAttente.remove(tache);}}
List<TacheMateriel> tacheTraite = new ArrayList<TacheMateriel>();
      for(TacheMateriel tm:fileAttente){
            if(traiterDemande(tm)!=null){tacheTraite.add(tm);}}
```

```
for(TacheMateriel tm:tacheTraite){
        fileAttente.remove(tm);}return true;}}
```

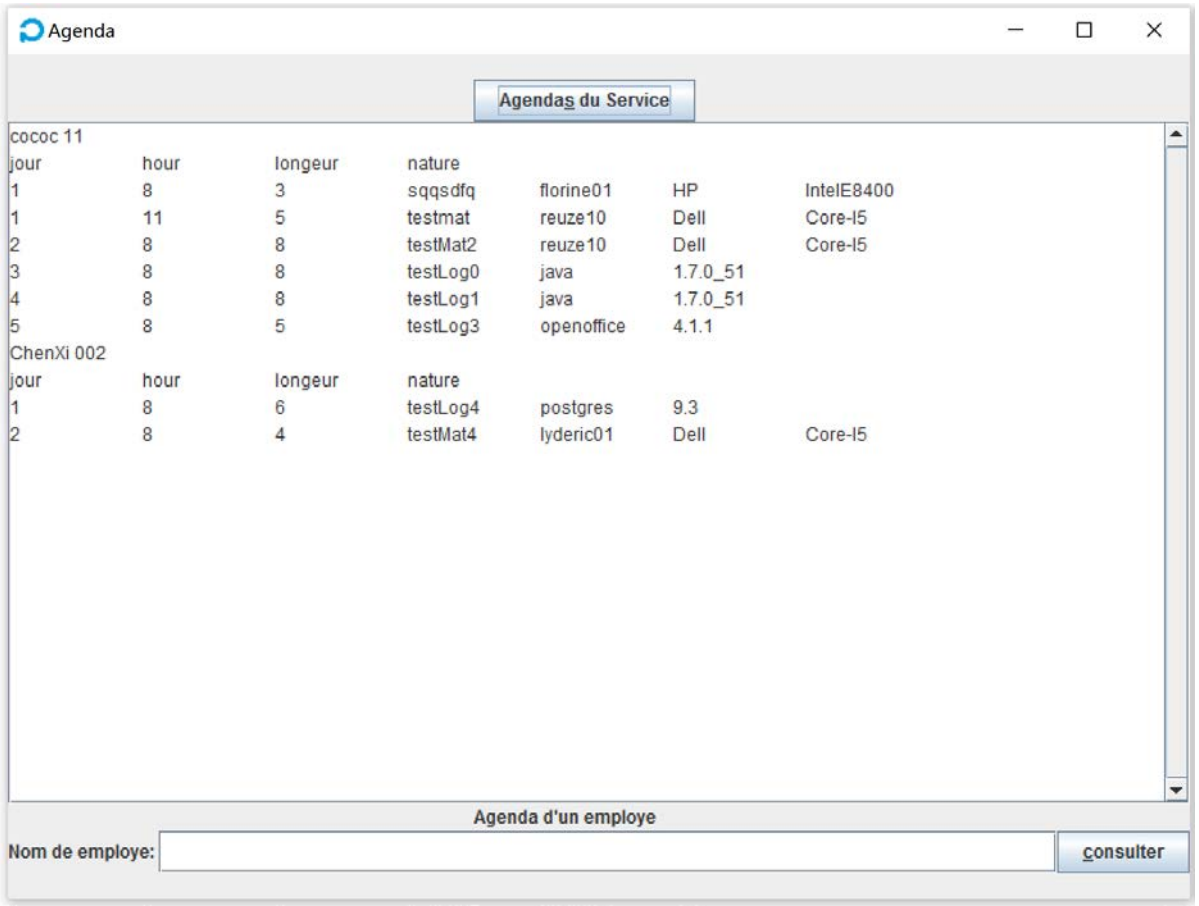
Si on a une tâche annulée seulement dans la liste d'attente, ce n'est pas la peine de traiter la file d'attente étant donné que l'emploi du temps n'est pas modifié, cependant trouvera pas un employé disponible. Malheureusement nous avons oublié de traiter cette partie !

c. Sérialisation

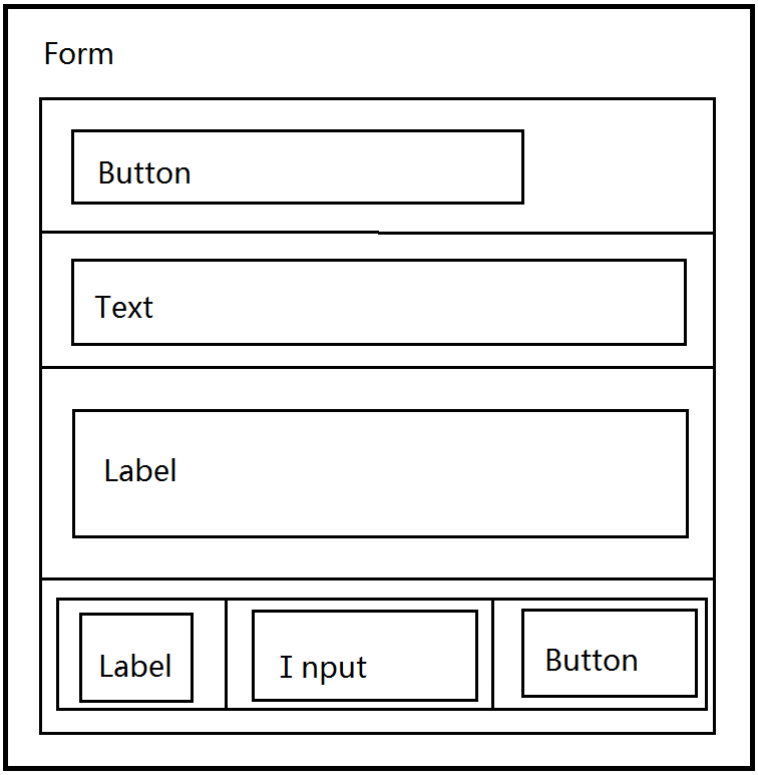
La sauvegarde des données de la classe service informatique est réalisée grâce aux méthodes save(fichier) et load(fichier) programmées dans l'application console du service informatique. La méthode save(fichier) permettra d'enregistrer tout le contenu du service informatique dans un fichier de type binaire « data.bin » ; et la méthode load(fichier) permettra de recharger les données sauvegardées afin de les utiliser dans l'interface graphique.

d. Interface graphique

Nous avons donc mis en place une interface graphique suivant :



La structure de l’interface graphique est



Pour afficher afficher les emplois du temps de tous les employés, il suffit de cliquer sur le bouton Agendas du service sinon pour un employé, il faut taper son nom dans la partie nom de l'employé.

IV. Structure du logiciel

```
Agenda
|    build.xml (Fichier contrôlant la procédure de compilation)
|    data.bin (Contient les données sauvegardent)
|    parc.xml (Fichier donné)
|    polytech_128.png
|---lib
|        xerces.jar
|
|---src (Contient les codes sources)
     |---bibAgenda (Package bibAgenda)
     |        EmploiTemps.java
     |        Employe.java
     |        Service.java
     |        Tache.java
     |
     |---entretien (Package entretien)
     |        AppConsole.java
     |        Salle.java
     |        SalleNotFoundException.java
     |        ServiceEntretien.java
     |        TacheSalle.java
     |
     |---ihm (Package ihm contenant l'interface graphique)
     |        GUInterface.java
     |        (Cette classe est utilisée pour l'interface graphique)
     |
     |---si (Package si)
          |    AppConsole.java
          |    (Contient le code de lecture du xml et de la sérialisation)
          |    Logiciel.java
          |    LogicielNotFoundException.java
          |    Materiel.java
          |    MaterielNotFoundException.java
          |    Ressource.java
          |    ServiceInfo.java
          |    TacheLogiciel.java
          |    TacheMateriel.java
          |
          |---parseur
                   HandlerLogiciels.class
                   SIHandler.java
```

v. Mode d'emploi

Sous linux :

Pour compiler il suffit de se placer dans le dossier Agenda, ensuite taper la commande **ant**.

Pour supprimer tous les fichiers compilent, il faut taper la commande **ant clean**.

Pour exécuter l'application console(AppConsole.java) dans entretient, il faut taper la commande **source appEnt.sh**.

Pour exécuter l'application console(AppConsole.java) dans service informatique si, il faut taper la commande **source appSI.sh**.

Pour exécuter l'interface graphique(GUInteface.java), il faut taper la commande **source appG.sh**.

BILAN ET CONCLUSION

A la fin de ce projet, on a pu réaliser la conception d'une bibliothèque de classes générales utilisable par différents service comme service entretient et service informatique. Ensuite nous avons sérialisé le service informatique, une fois sérialisé a pu générer un fichier binaire qu'on a utilisé pour recharger le contenu de ce service et l'afficher sous une interface graphique.

Une amélioration possible consisterai à réduire la complexité pour la recherche d'un employé par son nom. Pour ce faire il faudrait utiliser un HashMap au lieu d'utiliser un ArrayList.

D'une autre part, on pourra aussi contrôler tous les conflits par exemple deux tâches ne peuvent pas effectuer dans la même salle..... comme nous avons prévu de le faire dans la première partie du rapport. On aura pu également partager les tâches à tous les employés au lieu de les affectées d'abord au premier employé trouvé.

Même si notre structure de données n'est pas optimale, elle est restée simple et plus facile à comprendre.

En conclusion ce projet fut d’un apport très considérable, il nous a permis de bien évaluer la puissance et l’efficacité du langage de programmation orienté objet Java sur un travail d’une assez importante envergure. Par ailleurs nos compétences ont bien été développées notamment dans l’utilisation de l’IDE ECLIPSE qui est pourra nous servir tout au long de notre cursus et carrière.