



UNIVERSITÉ DE MONTPELLIER

PROJET TER

Outil de Gestion d'un Emploi du Temps

Thomas LEMAÎTRE
Bérénice LEMOINE
Julien LESINSKI
Olivier MONTEL

Supervisé par
Mme Hinde BOUZIANE

5 mai 2017

Sommaire

1	Introduction	2
2	Analyse des Problèmes et Choix de Conception	2
2.1	Modélisation	2
2.2	Gestion des Données et des Contraintes	4
2.3	Interface Homme-Machine	5
2.4	Schéma Récapitulatif des Choix de Conception	6
3	Choix Technologiques et Implémentation	6
3.1	Choix et relations des fonctionnalités	7
3.2	Gestion des Données et des Contraintes	7
3.3	Interface Homme-Machine	8
4	Avancement	10
4.1	Perspectives	11
5	Conclusion	12
6	Remerciements	13
7	Annexes	14
7.1	Choco	14
7.2	Liste des contraintes	14
7.3	Représentation du fichier de sauvegarde	15
7.4	Diagramme de Classe	16

1 Introduction

Le TER¹ de 2^e année de licence en informatique est un projet important permettant de découvrir des domaines que nous ne maîtrisons pas encore, ainsi que d'utiliser de manière concrète des connaissances déjà acquises.

Ce projet informatique avait pour but de mettre en place un outil de gestion d'emploi du temps, c'est-à-dire, que nous devions créer un emploi du temps sur une semaine type d'un semestre. Le projet Octempos consiste à développer un outil pour créer et afficher un emploi du temps respectant un maximum de contraintes saisies par l'utilisateur. Pour mener à bien ce projet, il a fallu être en mesure de gérer de multiples contraintes mais aussi être apte à travailler en groupe et donc à se coordonner sur la modélisation et l'implémentation du projet. Nous avons mis en place un programme en Java avec une interface graphique (réalisée grâce à Java Swing/Awt) permettant la saisie des différentes UEs² ainsi que des contraintes souhaitées, et réalisant un emploi du temps qui respecte au mieux les attentes de l'utilisateur ; ceci à l'aide d'un solveur appartenant à une librairie de Java appelée « Choco ». Si le solveur trouve une solution, le programme crée une page web affichant les cours des différents groupes pour une semaine.

Lors des premières séances de travail en groupe nous avons surtout réfléchi à la modélisation du projet, c'est-à-dire, aux façons de mettre en œuvre les différentes problématiques, mais aussi à la répartition des tâches. Apprenant le Java depuis le début du semestre 2, nous avons tous dû apprendre à utiliser un langage ou un concept inconnu, nous nous sommes ainsi mis d'accord assez facilement sur la répartition des tâches, chacun choisissant la partie qui l'intéressait le plus. Toutefois, les réflexions importantes sur la répartition des différentes directions à prendre se sont faites en groupe. Afin de faciliter le partage de fichiers et le travail collaboratif, nous avons utilisé le serveur « Gît » de l'université : « GitLab ». Il est possible d'avoir accès à notre dépôt Git³. Pour le partage des fichiers, nous avons fait le choix d'utiliser l'IDE (Integrated Development Environment) « Eclipse Neon » (une version adaptée à Java) pour sa simplicité d'utilisation mais aussi pour sa bonne cohabitation avec Git.

Ce rapport abordera d'abord le problème précis sur lequel nous avons travaillé, ainsi que les choix de conception. Ensuite, il traitera les choix de modélisation puis les outils utilisés. Finalement, il exposera l'avancement du projet et conclura sur un bref résumé du projet et des compétences que nous avons acquises.

2 Analyse des Problèmes et Choix de Conception

Cette partie va introduire les différentes problématiques qui ont aiguillé le projet Octempos et la façon dont nous avons choisi de répondre à celles-ci.

2.1 Modélisation

La modélisation du problème a été une partie importante de notre projet. Celle-ci a mené chacun d'entre nous à exposer sa vision des choses pour ensuite trouver, ensemble, une solution qui nous semblait être la meilleure mais aussi celle que nous étions capable de réaliser. Le point sur lequel nous avons tous été d'accord était le suivant : il fallait différents objets, un objet représentant les UEs, un pour les contraintes et un autre pour les groupes. Ensuite, nous

1. Travaux d'Etude et de Recherche
2. Unités d'Enseignement

3. Si vous souhaitez avoir accès au dépôt « Git », contactez l'un de nous quatre (Lemaître Thomas, Lemoine Bérénice, Lesinski Julien, Montet Olivier) à l'adresse mail : prénom.nom@etu.umontpellier.fr

avons réfléchi aux informations nécessaires pour chacun des objets. Ainsi, il a tout d'abord fallu déterminer différents types nécessaires pour l'objet **Contrainte** (voir Table 1).

<i>Type de contrainte</i>	<i>Description</i>
Voulu	Elle permet d'imposer un horaire (Jour et heure) pour une séance (TD, TP ou CM) choisi par l'utilisateur.
NonVoulu	Elle permet d'interdire une séance d'être à un horaire particulier (choisi par l'utilisateur).
Diff	Elle permet d'obliger deux séances (ou plus) à n'avoir pas lieu au même moment.
Equi	Elle permet de contraindre deux séances à avoir lieu au même moment.
Suivi	Elle permet de faire en sorte qu'exactement deux séances se trouvent à la suite l'une de l'autre.

TABLE 1 – Les différents types de contrainte

Pour l'objet UE, il a fallu mettre beaucoup d'attributs, tout d'abord, le nom du responsable, celui de l'UE, les nombres des différents types de cours, mais aussi les différents groupes qui composent cette unité d'enseignement, etc. La liste des éléments essentiels pour chacun des objets a ainsi été mise en place (pour plus de détails vous pouvez trouver en annexe le premier diagramme de classes que nous avons réalisé et le diagramme UML⁴ final, vous permettant de voir tous les attributs des différents objets et de voir l'évolution de notre modèle de conception).

Par la suite, il nous a fallu établir un modèle pour exploiter ces différents objets. Un modèle est axé sur trois points. Premièrement, des variables, ensuite des contraintes sur les variables et enfin un solveur qui va résoudre toutes les contraintes en assignant des valeurs aux variables pour trouver une solution à notre problème. La variable ici choisie pour le programme Octempos est une matrice d'entiers (l'ensemble de valeurs que peuvent prendre ces entiers est compris entre 0 et le nombre de séances total). Elle est composée de trente-cinq colonnes et son nombre de lignes est égal au nombre total de sous-groupes (les sous-groupes sont des groupes indivisibles, par exemple, le groupe A et les sous-groupes A1 et A2). Le but de ce modèle est, pour chaque sous-groupe, de lui assigner un tableau de trente-cinq cases pour représenter tous les créneaux de 1h30 qu'il y a dans une semaine (5 jours × 7 créneaux).

Sous-Groupes \ Créneaux	Lundi (8h–9h30)	Lundi (9h45–11h15)	Lundi (11h30–13h)	...
A1	0	1	2	...
A2	3	1	4	...

TABLE 2 – La variable *matrice*

Par exemple, la figure ci-dessus représente une matrice avec deux sous-groupes, et les entiers de 0 à 4 sont des identifiants (ID) de séances. L'ID 1 est un cours commun entre les deux sous-groupes, la valeur 0, quant à elle, ne représente aucune séance, elle est utilisée pour un horaire sans cours pour le sous-groupe.

Nous verrons, par la suite, comment les objets sont instanciés et utilisés par le modèle pour générer un emploi du temps.

4. Unified Modeling Language

2.2 Gestion des Données et des Contraintes

La création d'un outil de gestion d'emploi du temps nécessite de manipuler de nombreuses données (que nous appellerons ici objet) telles que les unités d'enseignement, les contraintes à respecter ou encore les groupes qu'il faudra créer selon les paramètres des UEs.

2.2.1 Sauvegarde des données

Une sauvegarde des données s'est avérée nécessaire afin de faciliter la saisie de celles-ci. Cette sauvegarde permet, une fois que l'utilisateur a saisi des informations, de les sauvegarder. S'il le décide, il sera en mesure de les réutiliser et de modifier ces données sans avoir à tout ressaisir. Pour mettre en place cette sauvegarde, il a fallu réfléchir à la façon de récupérer les contraintes, les unités d'enseignement et les informations sur le responsable, le nombre d'heures, etc.

Nous avons opté pour une interface permettant de saisir de nouvelles données et d'importer des données existantes. Il est possible aussi de modifier ces données si elles existent déjà. Ce qui permettra, par exemple, d'alléger les contraintes lorsque le programme n'arrive pas à fournir une solution. Afin d'enregistrer les données, nous avions le choix entre faire une base de données ou créer un fichier (semblable à un fichier de configuration). Possédant peu de connaissances sur les bases de données, nous avons préféré apprendre à gérer les flux en Java, et avons choisi de créer un fichier (vous pouvez trouver ce fichier sous le nom de « save.txt »).

Le fichier a une syntaxe que nous avons définie : tout d'abord les groupes qui ont été entrés lors de la saisie avec chacun de ses sous-groupes. Ensuite, il y a le nom d'une UE, avec son responsable, les différentes séances de celle-ci et les différentes contraintes pour chacune des séances, etc.

2.2.2 Les Contraintes

Une fois les données sauvegardées et récupérées, il faut les soumettre à un ensemble de contraintes pour le solveur puisse fournir une solution.

Il est très rare de réussir à organiser un emploi du temps qui respecte toutes les contraintes, c'est pour cette raison qu'il faut en modéliser deux sortes. Les contraintes dures qui doivent toutes être satisfaites par le solveur et les contraintes molles dont le solveur essaie d'en satisfaire un maximum. En voici un descriptif dans la table 3.

<i>Contraintes Dures</i>	<i>Contraintes Molles</i>
<ul style="list-style-type: none">• Si l'une d'elles n'est pas satisfaite, aucune solution n'est trouvée par le solveur.• Représente des attentes, l'utilisateur veut toutes les voir apparaître dans l'emploi du temps.• Contraintes utilisées par défaut dans Octemos.	<ul style="list-style-type: none">• Si l'une d'elles n'est pas satisfaite, le solveur cherche une autre solution sans cette contrainte.• Représente des préférences, l'utilisateur veut en avoir un maximum représentée dans l'emploi du temps.

TABLE 3 – Deux sortes de contrainte

Citons quelques exemples pour comprendre :

- On attend du programme qu'un groupe ait une pause pour manger, concrètement, il ne doit pas avoir un cours à 11h30 et à 13h15. Cette contrainte par défaut est une contrainte dure, elle ne peut pas être retirée par l'utilisateur et doit être satisfaite dans tous les cas.
- Interdire un horaire à une séance est une contrainte dure, car si l'utilisateur l'a souhaitée, elle doit être satisfaite.
- Une séance qui doit être suivie dans le temps par une autre séance est une contrainte molle : si l'utilisateur l'a entrée, mais qu'elle ne peut être satisfaite, alors elle n'apparaîtra pas dans la solution.

Vous trouverez en annexe la liste de toutes les contraintes gérées par Octempos.

2.3 Interface Homme-Machine

L'objectif de l'IHM⁵ est de simplifier l'utilisation de l'outil pour les utilisateurs. Nous avons pour cela décidé de créer un système de fenêtre.

2.3.1 Visualisation et Manipulation des Données

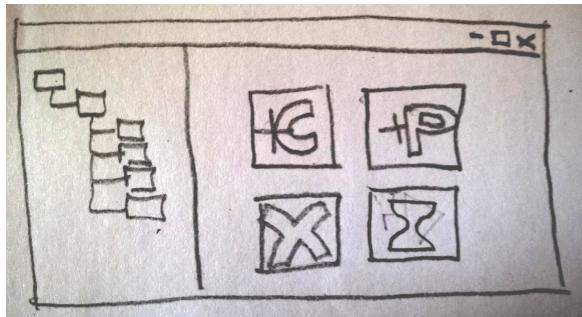


FIGURE 1 – Un schéma de la fenêtre principale

images plutôt que du texte. Cela peut paraître déroutant lors de la première utilisation mais fluidifie la navigation lors des utilisations suivantes. Ces images permettent ainsi à l'utilisateur de moins chercher où se trouve le bouton dont il a besoin, néanmoins, en passant sur le bouton une infobulle doit pouvoir informer de ce que fait le bouton concrètement.

Pour informer l'utilisateur des étapes de la génération de l'emploi du temps, nous avons décidé de créer une fenêtre contenant les messages affichés normalement dans la console.

En ce qui concerne le système de fenêtre, il doit permettre notamment de modifier les données inhérentes au programme. En effet, les UEs et les contraintes (ou préférences) doivent être modifiables et supprimables via des boutons dédiés de la fenêtre principale après avoir sélectionné les éléments voulus. Les séances, en revanche, ne devaient pas pouvoir être modifiables directement, car nous avons décidé de les générer selon les paramètres de l'UE. Une autre fenêtre doit aussi permettre de modifier les groupes de base.

Nous avons choisi pour permettre à l'utilisateur de sauvegarder, de créer un bouton dans la fenêtre principale qui serait disponible uniquement quand il y aurait des données à sauvegarder.

Le but était d'avoir une synthétisation des données de l'outil dans la fenêtre, représentée à gauche sur le schéma.

Pour afficher les groupes de base, nous avons décidé de les lister dans une autre fenêtre : la fenêtre des groupes. Les sous-groupes, en revanche, ne doivent pas être listés car ceux-ci sont gérés par l'outil uniquement. L'utilisateur doit aussi pouvoir accéder au mode d'emploi d'Octempos, nous avons décidé, pour cela, de créer un bouton dans la fenêtre principale. Nous avons choisi que tous les boutons de la fenêtre principale soient des

5. Interface Homme-Machine

2.3.2 Affichage de la Solution

Une fois qu'une solution est trouvée, il fallait un moyen pour l'afficher sous la forme d'un emploi du temps. Ce dernier représente les cinq jours de travail de la semaine et affiche l'emploi du temps de tous les groupes créés. Nous avons fait le choix de créer une solution modifiable par l'utilisateur pour permettre des modifications manuelles. Pour cela, il a été décidé que l'emploi du temps serait renvoyé sous forme de page web contenant cinq tableaux, un par jour de travail de la semaine. Ces tableaux sont des tableaux à double entrée (semblable à la Figure 2), les lignes représentent les groupes que l'on doit afficher et les colonnes correspondent aux horaires possibles de travail.

2016-2017		Faculté des Sciences		Emploi du Temps		I3 INFORMATIQUE		Semestre 6		Responsable Formation : Hinda Bouziane hinda_bouziane@lille1.fr	
STD = salle de TD (40 places); STD+P= salle de TD informatisée du Bât 5; TP B6 = TP au bâti 6 réservé pour nos soins; Amphi cours = 120 places											
LUNDI 13h				8H00 - 9H30	9H45 - 11H15	11H30 - 13H	13H15 - 14H45	15H - 16H30	16H45 - 18H15		
	A				TP HLN604 - STD/TP bâti 5		CM HLN604 - STD avec vidéo	TD HLN604 - STD			
	B				TP HLN604 (+ Math-Info) - TP bâti 5		TP HLN604 (+ Math-Info) - STD/TP bâti 5	CM HLN606 (+ Math-Info) - S.C 60 places	TD HLN606 (+ Math-Info) - S.C 60 places		
	C				TP HLN604		CM HLN604	TD HLN604			
	CM1 - Gx A						TP HLN605	CM HLN605			
	CM1 - Gx B										
	CM1 - Gx C										
					TP HLN608						
MARDI 13h											
	A				TD HLN612 - STD		TD HLN601 *				
	B				TD HLN612 (1/2 Gr) - STD		TD HLN602 - STD				
	C				TD HLN602 (1/3 du Gr) + Math-Info - STD		TD HLN601 (pour 1/3 du Gr) * / TD HLN602 (pour autre 1/3 du Gr)				
	CM1 - Gx A				TD HLN602		TD HLN601 *				
	CM1 - Gx B				TD HLN602		TD HLN601				
	CM1 - Gx C				TD HLN602		TD HLN601 *				
MERCRIDI 13h											
	A				TD HLN603 - STD/TP bâti 5	TP HLN603 - STD/TP bâti 5					
	B				TD HLN603 - STD/TP bâti 5	TP HLN603 - STD/TP bâti 5					
	C				TD HLN603 - STD/TP bâti 5	TP HLN603 - STD/TP bâti 5					
	CM1 - Gx A				TD HLN603	TP HLN603					
	CM1 - Gx B				TD HLN603	TP HLN603					
	CM1 - Gx C				TD HLN603	TP HLN603					
JEUDI 13h											
	A				TP HLN611 - TP bâti 8	TD HLN611 - STD/TP bâti 8					
	B				TP HLN611 - TP bâti 8	TD HLN611 - STD/TP bâti 8					
	C				TP HLN611 - TP bâti 8	TD HLN611 - STD/TP bâti 8					
	CM1 - Gx A				TP HLN611	TD HLN611					
	CM1 - Gx B				TP HLN611	TD HLN611					
	CM1 - Gx C				TP HLN611	TD HLN611					
VENDREDI 13h											
	A				TD HLN612 (1/2 Gr) - STD		TD HLN602 - STD				
	B				TD HLN612 - STD		TD HLN602 *				
	C				TD HLN612		TD HLN602 (2/3 du Gr) * / TD HLN602 (pour 1/3 restant du Gr)				
	CM1 - Gx A				TD HLN612						
	CM1 - Gx B				TD HLN612						
	CM1 - Gx C				TD HLN612						

FIGURE 2 – Exemple d'emploi du temps

2.4 Schéma Récapitulatif des Choix de Conception

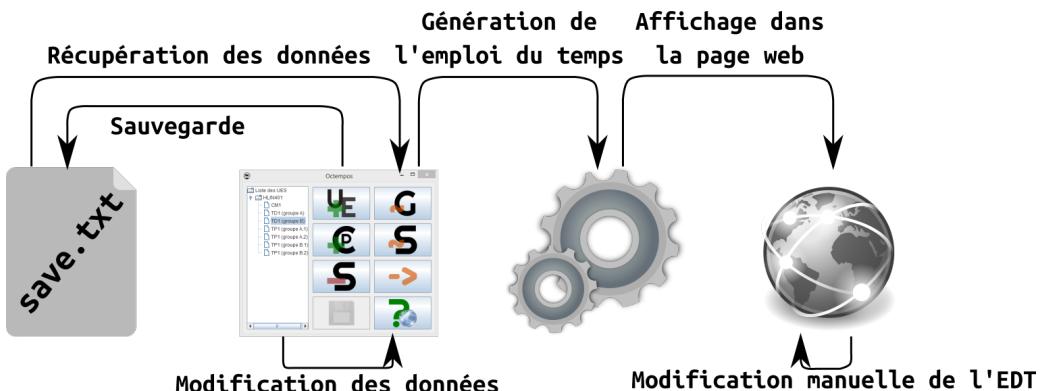


FIGURE 3 – Représentation de l'architecture voulue

3 Choix Technologiques et Implémentation

Cette partie du rapport va aborder les points techniques du projet. En passant par les choix des langages et outils utilisés mais aussi par les choix d'implémentation.

3.1 Choix et relations des fonctionnalités

Les données ont été introduites sous forme d'objet, il était donc naturel de les implémenter dans un langage orienté objet. Connaissant le C++ et débutant en Java, notre choix s'est donc porté sur l'un d'entre eux. L'utilisation de « Choco » comme solveur de contraintes nous a dirigé sur le Java (cf. 3.2 Gestion des Données et des Contraintes), plutôt qu'utiliser le C++ avec son solveur « GECODE » qui nous paraissait moins abordable. Les principales bibliothèques Java qui ont été utilisées sont « Choco », pour les contraintes, et « Java Swing » pour l'interface graphique. Également, les classes (préexistantes) telles que `StringTokenizer`, `BufferedReader`, `BufferedWriter`, `FileReader` et `FileWriter` ont été essentielles à la sauvegarde et la lecture des données inscrites dans le fichier « save.txt ». La Figure 3 présente les points sur lesquels est centré le projet. Elle en présente quatre, mais lors de l'utilisation de l'outil, seules l'interface graphique et la page web sont visibles. Le programme affiche l'interface laissant saisir les UEs ainsi que les contraintes désirées. L'utilisateur fait ensuite le choix de sauvegarder ou non ces informations. S'il le fait, le programme gérant l'interface sauvegarde les valeurs dans un fichier « save.txt », ainsi lors de la prochaine utilisation du programme, si l'utilisateur demande de récupérer la sauvegarde, le programme ira lire le fichier afin d'enregistrer les données dans le programme. Sinon, il devra ressaisir toutes les informations à la prochaine réutilisation. Une fois les données saisies ou récupérées, le programme fait appel au solveur pour générer un emploi du temps respectant un maximum de contraintes. Puis, le programme affiche l'emploi du temps dans une page web.

3.2 Gestion des Données et des Contraintes

3.2.1 Implémentation de la sauvegarde des Données

Cette partie est celle qui a été le plus souvent modifiée. Nous voulions d'abord nous en occuper dans le `main`, mais cela rendait notre code abscons. Nous avons donc choisi de créer une classe `Configuration.java`, qui gère l'écriture des données, de l'interface dans le fichier de sauvegarde (en utilisant la méthode `void LoadConfig(String nomFichier)`). Cette classe gère aussi la lecture du fichier (à l'aide de la classe `StringTokenizer` en Java) et l'enregistrement des données dans le programme (à l'aide de la méthode `boolean saveConfig(String nomFichier)`). La classe `Configuration.java` contient de nombreux accesseurs qui ont facilité l'implémentation des autres parties du projet.

3.2.2 Modèle de Contraintes

Nous avons précédemment souligné l'importance d'établir un modèle pour trouver une solution au problème de gestion. Pour l'implémenter, notre choix s'est porté sur la librairie Choco. C'est une librairie Java qui fournit un grand nombre de méthodes permettant de créer des contraintes (voir annexe sous-section 7.1)

En premier lieu, il faut implémenter la variable vue dans la partie 2.1.

- `Model model = new Model("Emploi Du Temps");`

Avant tout, on instancie un nouveau `Model` nommé « Emploi Du Temps » (`Model` étant un objet de Choco).

- `IntVar[][] matriceDuTemps = model.intVarMatrix("", nbG, 35, 0, n);`

La variable est donc une matrice de `IntVar` (une variable spécifique à Choco qui s'assimile à l'entier (`int`) de Java) créée grâce à la méthode :

```
intVarMatrix(String nom,int nbLisgnes,int nbColonnes,int valMin,int valMax).
```

Ces `IntVars` vont pouvoir, lors de l'exécution, prendre un entier entre 0 et n où n est le nombre de séances total.

En second lieu, il faut établir un ensemble de contraintes sur la variable, vous pouvez retrouverz l'ensemble des contraintes en annexe. Étudions quelques contraintes pour comprendre :

- `model.count(0, onze-treize, model.intVar(1,2)).post();`

Cette contrainte permet à un groupe d'avoir une pause pour manger, concrètement, il ne doit avoir qu'un seul cours de 11h30 à 15h. La méthode `count(int valeur, IntVar[] Tableau, IntVar limite)` permet par ailleurs de tester si l'entier 0 (aucun cours), se trouve entre 1 à 2 fois dans le tableau `onze-treize` (qui possède les deux créneaux possibles de repas d'un groupe pour un jour).

- `model.arithm(matriceDuTemps[IdsousGroupe][Obli], "=", seance.getId()).post();`

Ici, on impose à un groupe d'avoir une séance donnée à un horaire précis.

La méthode `arithm(IntVar horaire, String opérateur, int identifiant)` oblige le créneau d'un groupe (position dans la matrice [ID du groupe] [horaire de l'obligation]) à posséder cette séance (donc que le `IntVar` soit égal à l'identifiant du groupe).

- La dernière contrainte est plus compliquée, elle doit faire en sorte que deux séances se suivent dans le temps.

Cependant, c'est une contrainte molle, donc il ne faut pas seulement demander au solveur de la résoudre, mais aussi de vérifier qu'elle est satisfiable. À la place du `.post()` qu'on utilise pour les contraintes dures, on utilise la méthode `.reify()`. Cette méthode a la particularité de renvoyer `true` si la contrainte est satisfiable, sinon elle renvoie `false` et demande au solveur de l'oublier. Pour satisfaire le plus grand nombre de préférences, il suffit de récupérer le nombre de préférences satisfiables et de le maximiser.

Le solveur va par la suite résoudre toutes ces contraintes pour assigner une valeur à chaque `IntVar` de la matrice. S'il existe une solution, il envoie une matrice d'entiers à une fonction d'affichage.

3.3 Interface Homme-Machine

Pour créer cette interface, nous avons utilisé majoritairement la bibliothèque Java Swing et un peu Java Awt (Swing s'appuyant sur Awt) en utilisant donc, ses modules graphiques mais aussi ceux qui permettent de répondre aux actions de l'utilisateur, soit ceux qui impliquent de la programmation événementielle.

3.3.1 Visualisation des Données

Pour créer et afficher la liste des UEs présentes dans le programme, la fenêtre principale prend l'instance de la classe `Configuration` qui contient toutes les informations du programme et explore chaque UE puis les séances à l'intérieur de celles-ci pour finir avec les contraintes portant sur les séances en créant, à chaque fois, un noeud dans un `JTree`⁶. Cet arbre est particulièrement adapté pour conserver la hiérarchie des données du programme (en jouant sur la profondeur des noeuds). La fenêtre de groupe suit le même procédé pour afficher la liste des groupes de base.

La fenêtre qui apparaît pendant la génération de l'emploi du temps est actualisée grâce à la génération de l'emploi du temps elle-même qui définit le texte qu'elle contient grâce aux méthodes `void addTextLn()` et `void clearText()` et définit aussi la progression ce qui change

6. widget représentant un arbre dans Java.Swing

l'état de la barre de progression (`JProgressBar`) grâce à la méthode void `setProgression(int percentage)`. Le manuel est ouvert par une simple fonction de la bibliothèque Java qui permet d'ouvrir un document externe avec le logiciel par défaut de l'utilisateur.

3.3.2 Manipulation des Données via l'IHM

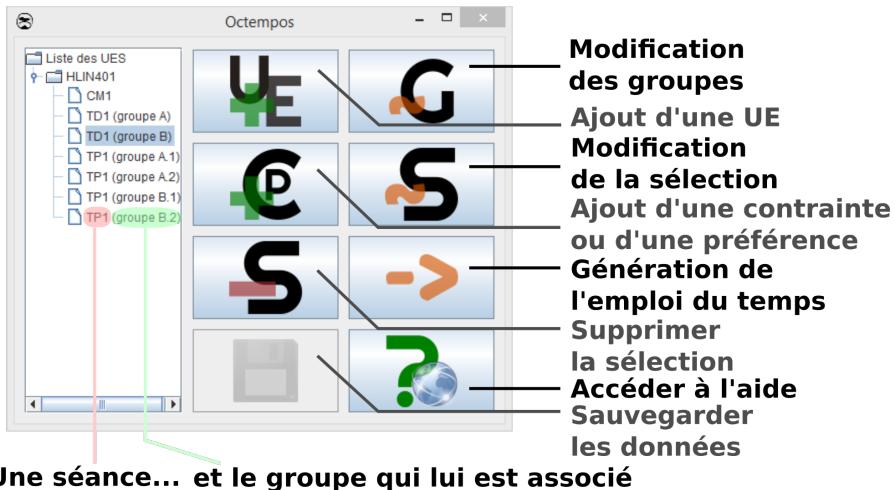


FIGURE 4 – Fenêtre d'accueil

Le système de fenêtre effectue à chaque modification un ensemble de contrôle permettant de vérifier la cohérence des informations entrées par l'utilisateur (vérification que les champs ne sont pas vides, pour une UE : qu'il y ait au moins un groupe de sélectionné, etc).

Les boutons de modification et de suppression récupèrent le niveau de l'élément sélectionné dans le `JTree` pour déterminer la nature de celui-ci et modifier les données en conséquence.

Le bouton de sauvegarde est grisé quand il n'y a pas eu d'appel à la méthode void `dataChanged()` (pas de données modifiées depuis la dernière sauvegarde). Quand l'utilisateur clique sur ce bouton (et qu'il n'est pas grisé), les données sont sauvegardées sur le disque dur de l'utilisateur et plus précisément sur le fichier « save.txt ».

3.3.3 Affichage de la Solution

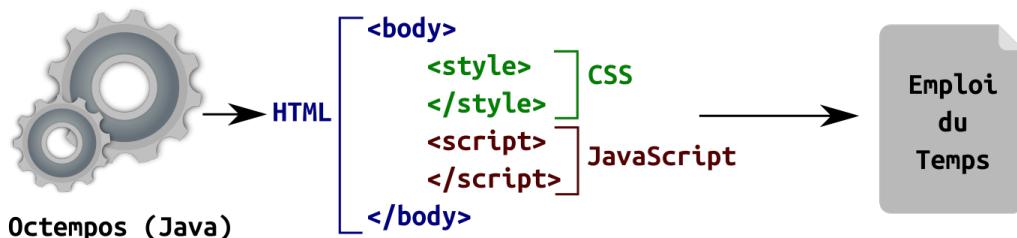


FIGURE 5 – Schéma d'écriture du fichier html

Pour créer notre page web, nous avons écrit un fichier HTML contenant notre emploi du temps par l'intermédiaire d'un programme en Java, en écrivant les balises `<style>` et `<script>` ainsi que le corps de notre emploi du temps. Voici un bref aperçu du codage d'une séance dans un tableau.

```

<table>
  <tr>
    ...
    <td class="HLIN403">HLIN403-CM</td>
    ...
  </tr>
  ...
</table>

```

Nous avons également deux boutons qui sont utilisés pour la modification des tableaux. Un bouton « Supprimer » et un bouton « Modifier/Ajouter ». Les CMI étant un cas particulier, nous avons décidé de rajouter dans chaque tableau trois lignes spécialement dédiées aux CMI pour que l'utilisateur puisse plus facilement gérer leurs séances.

	Groupe	8h-9h30	9h45-11h15	11h30-13h	13h15-14h45	15h-16h30	16h45-18h15	18h30-20H
Lundi	Groupe A.1		HLIN403-CM			HLIN401-TP		HLIN401-TD
	Groupe A.2		HLIN403-CM		HLIN401-TP			HLIN401-TD
	Groupe B.1		HLIN403-CM	HLIN401-TD				
	Groupe B.2		HLIN403-CM	HLIN401-TD				
	Groupe CMI 1							
	Groupe CMI 2							
	Groupe CMI 3							

	Groupe	8h-9h30	9h45-11h15	11h30-13h	13h15-14h45	15h-16h30	16h45-18h15	18h30-20H
Mardi	Groupe A.1				HLIN403-TD			
	Groupe A.2				HLIN403-TD			HLIN403-TP
	Groupe B.1		HLIN401-TP					
	Groupe B.2							
	Groupe CMI 1							
	Groupe CMI 2							
	Groupe CMI 3							

FIGURE 6 – Emploi du temps final

4 Avancement

L'utilisation de Java nous a permis de créer facilement des classes représentant les objets dont nous avions besoin grâce aux nombreuses propriétés que l'on peut fournir à un objet (attributs et méthodes). Cela nous a aussi permis de sauvegarder les données de manière cohérente et pratique. Aussi, nous avons créé des accesseurs permettant l'accès rapide à chacune des données nécessaires au solveur de contraintes et à l'interface.

De plus, grâce à la bibliothèque graphique Java Swing, nous avons été en mesure de produire une interface cohérente permettant à l'utilisateur de modifier les données du programme afin de construire un emploi du temps adapté à ses besoins. En plus de cela, le système vérifie la cohérence des données entrées afin de permettre au programme de fonctionner correctement. Ensuite, grâce à l'organisation des informations, leur récupération pour les exploiter afin d'établir le modèle de contraintes n'a pas été un souci. La mise en place d'un modèle qui permet de gérer des contraintes entre les séances mais aussi celles qui sont globales pour l'emploi du temps

a été réalisée avec succès. La partie qui demanda le plus de temps est d'apprendre et d'utiliser une nouvelle forme de programmation, la programmation par contraintes. Mais, en définitive, toutes les contraintes nécessaires (voir annexe sous-section 7.2) ont été codées et fonctionnent.

Finalement, nous obtenons un emploi du temps scindé en cinq tableaux, un par jour de la semaine. Ce dernier nous affiche dans chaque ligne l'emploi du temps pour un groupe. Nous avons aussi mis à disposition une aide qui explique comment manipuler cet emploi du temps et deux boutons servant à le modifier.

4.1 Perspectives

Tout d'abord, nous gérons les données de manière fonctionnelle, mais nous nous sommes rendus compte après le cours sur l'héritage en Java (pendant l'UE « Modélisation et programmation par objet 1 ») que nous aurions pu améliorer la gestion des différentes contraintes en utilisant l'héritage. Ainsi, il aurait fallu faire une classe abstraite `Contrainte.java` puis deux sous-classes : une pour les contraintes de type `Voulu` et `NonVoulu` qui possède exactement les mêmes attributs et une seconde pour les autres types qui possède une liste de séances en plus mais aucun créneau horaire.

Aussi, l'interface est fonctionnelle mais pourrait être améliorée notamment en modifiant l'aspect visuelle des fenêtres « Contrainte » et « UE ». Par manque de temps, nous n'avons pas pu vérifier comment faire pour empêcher le programme d'être lancé plusieurs fois ce qui pourrait être gênant pour la sauvegarde qui serait écrasée par chaque instance du programme. Nous pourrions également proposer à l'utilisateur de choisir l'emplacement de l'enregistrement du fichier de sauvegarde ou encore quelle sauvegarde il souhaite utiliser. Il faudrait pour cela apporter une amélioration à notre programme qui serait la vérification de l'intégrité du fichier de sauvegarde.

En ce qui concerne la gestion des contraintes, les perspectives sont d'optimiser le système mis en place. Effectivement, les méthodes utilisées sont souvent mal choisies car nous n'avons suivi aucun cours officiel de programmation par contraintes, et cela engendre parfois une grande complexité dans l'algorithme qui ralentit fortement la résolution.

Finalement, pour la sortie de l'emploi du temps, nous pourrions améliorer son rendu général, ainsi que créer un bouton « Ajout Ligne » qui rajoutera une ligne à chaque emploi du temps, et qui remplacerait les trois lignes réservées aux CMI qui ne sont pas forcément nécessaires.

5 Conclusion

Nous avons eu du 16 janvier 2017 au 28 avril 2017 pour imaginer et concevoir un gestionnaire d'emploi du temps d'une semaine type à la Faculté des Sciences de Montpellier. Nous avons développé une interface intuitive pour exploiter au mieux toutes les fonctionnalités créées dans le but de produire automatiquement un emploi du temps sur mesure. Une grande partie de notre travail s'est focalisé sur l'organisation des informations, notamment à partir des diagrammes UML. Ce travail nous a donné la possibilité de sauvegarder les attentes de l'utilisateur, et par la suite de les récupérer pour pouvoir les traduire en un modèle de contraintes. La découverte et l'utilisation de la librairie Choco nous a, par ailleurs, été d'une grande aide pour pouvoir optimiser la génération d'un emploi du temps. Outre cette faculté du programme, nous avons également combiné la génération automatique à la modification manuelle via une page HTML. Aujourd'hui, ce gestionnaire offre une grande diversité d'options dans le cadre de l'organisation d'un emploi du temps universitaire, et permet de gérer au mieux les attentes des responsables d'UE, comme celle des étudiants.

Ce premier projet nous a permis de mieux appréhender l'univers de la gestion de projet. En consacrant exclusivement le premier mois de notre projet à la recherche et au travail en commun, nous avons pu rapidement l'organiser en quatre points, et répartir ainsi les tâches facilement entre les membres du groupe. L'étude du Java et de l'UML durant le semestre puis l'approfondissement de ceux-ci dans le cadre du projet ont considérablement élargi nos connaissances dans ce domaine. À plusieurs reprises, nous avons modifié notre travail grâce aux nouvelles connaissances acquises en cours afin de le parfaire. C'était un travail intéressant et nous ressentions confiants de cette expérience. Pour finir, la découverte de nouveaux langages tels que la programmation par contraintes, que nous étudierons en Master, ou encore la programmation évènementielle (pour l'IHM) nous a offert des connaissances personnelles non négligeables.

6 Remerciements

Nous tenons à remercier Mme Bouziane pour l'aide qu'elle nous a apportée tout au long du projet, ainsi que M. Bourreau pour son temps et ses explications sur la programmation par contraintes qui nous ont été d'une grande aide.

7 Annexes

7.1 Choco

Choco est une bibliothèque Java Open Source dédiée à la programmation par contraintes. Il permet à l'utilisateur de modéliser son problème. Pour la résolutions d'un problème, choco s'appuie sur une stratégie de filtrage. Il étudie toutes les contraintes pour assigner des valeurs aux variables dans le but d'optimiser l'objectif qui lui est fixé.

Vous pouvez trouver le lien vers le site de choco ici : <http://www.Choco-solver.org/>

Trouvez ici le lien vers la Javadoc de Choco et toutes leur méthodes : <http://www.Choco-solver.org/apidocs/index.html>

7.2 Liste des contraintes

→ Par défaut dans Octempos

- Chaque créneau de l'emploi du temps possède ou non une seule séance.
- Un même groupe ne peut avoir plusieurs fois la même séance dans son emploi du temps.
- Toutes les séances contenant au moins un groupe seront représentées dans l'emploi du temps.
- Une séance suivie par plusieurs groupes est au même créneau pour tous les groupes.
- Un groupe ne peut avoir cours à 8h et 18h30 dans une même journée.
- Un groupe ne peut avoir cours à 11h30 et 13h15 dans une même journée.
- Un groupe ne peut avoir un trou de plus de 3 heures (deux séances vides).

→ Pour les attentes de l'utilisateur

- Une séance respecte les obligations des créneaux indiqués.
- Une séance respecte les interdictions des créneaux indiqués.
- Une séance ne peut être au même créneau que d'autres séances si elle est indiquée par l'utilisateur, contrainte **Diff**.
- Deux séances qui doivent se suivre (fait indiqué par l'utilisateur) ne doivent pas être sur deux jours différents (donc : séance1 à 18h30 et séance2 à 8h le lendemain).

→ Pour les préférences

- Un groupe ne peut avoir un seul cours au cours d'une journée (par défaut).
- Une séance est au même créneau que d'autres séances si elle est indiquée par l'utilisateur, contrainte **Eq**.
- Une séance doit être directement suivie d'une autre séance si l'utilisateur l'a voulu, contrainte **Suivi**.

7.3 Représentation du fichier de sauvegarde

Voici un extrait du fichier de sauvegarde montrant la représentation d'une UE avec deux séances :

```
Groupe=A
  Sous-groupe=A.1
  Sous-groupe=A.2
Groupe=B
  Sous-groupe=B.1
  Sous-groupe=B.2
UE=HLIN401
  Nom=Montet
  Prenom=Olivier
  Total=0
  NombreCM=1
  NombreTD=1
  NombreTP=1
  Foreground=-13434880
  Background=-16711681
  GroupeUE=A
  GroupeUE=B
  PartageCM=TOUS
  PartageTD=GROUPE
  PartageTP=DEMIGROUPE
  Seance=CM1
    TypeSeance=CM
    GroupeSeance=A
    GroupeSeance=B
    Voulu=Vendredi:H2
    Equi=TD1
  Seance=TD1
  ...
  
```

7.4 Diagramme de Classe

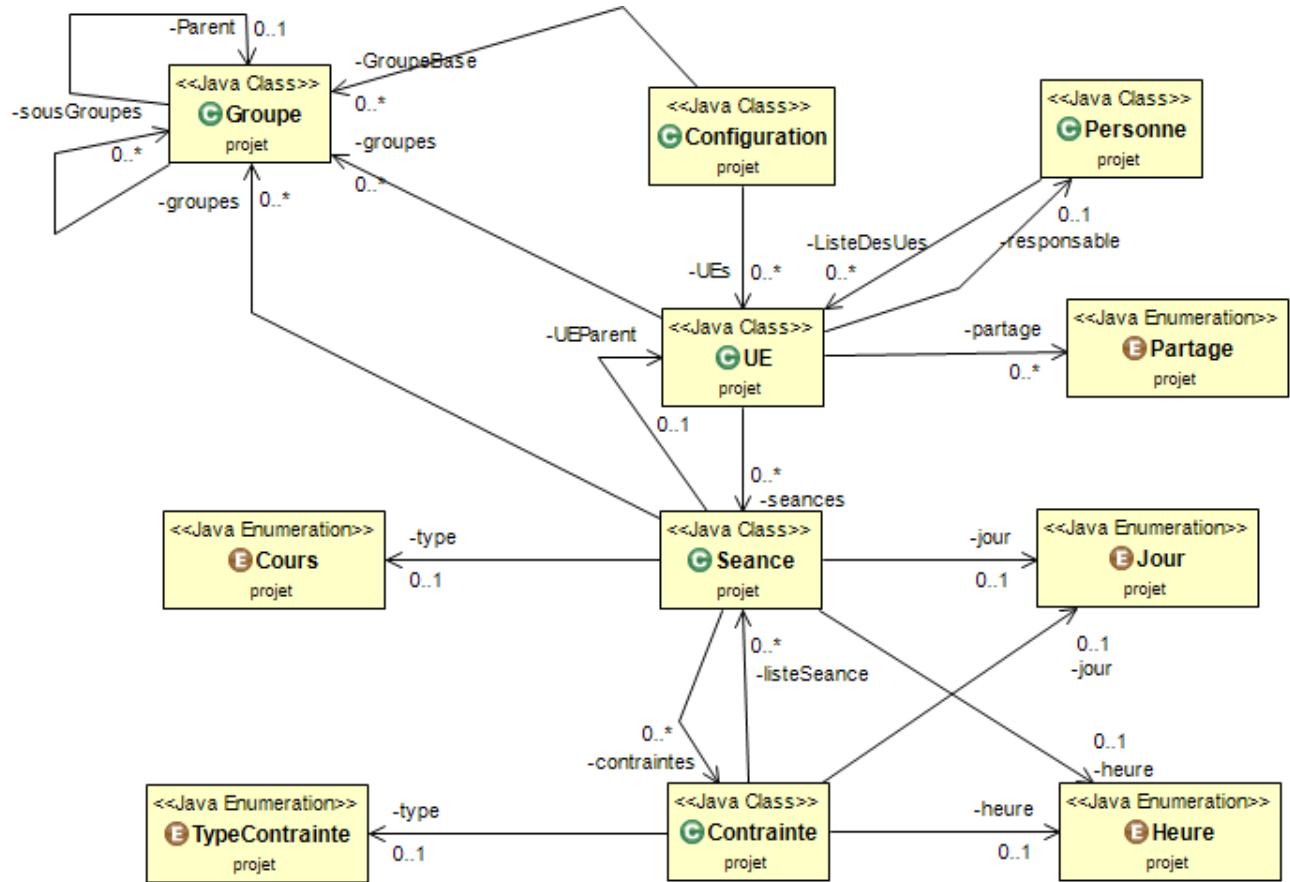


FIGURE 7 – Le diagramme de classe simplifié de Octempos