Ecole Pratique des Hautes Etudes Commerciales
Année Académique 2011-2012

Analyse et intégration de techniques « constraint programming » et « operations research » pour la réalisation d'un outil d'aide à la création d'horaires

Kevin Jacoby *et Xavier* Dubruille

Promoteur: **C. Lambeau**

Travail de fin d'études présenté en vue de l'obtention du grade du diplôme de bachelier en Informatique et Systèmes : finalité Technologie de l'Informatique

Ecole Pratique des Hautes Etudes Commerciales
Technologie de l'Informatique - Avenue du Ciseau 15
B-1348 Louvain-la-Neuve, Belgique TEL : +32 (0)10 47 53 90

Résumé

Un abstract présente en 100-150 mots la substantifique moelle du travail.

l'intérêt de la question
la problématique
quelques mots de méthodologie
les résultats principaux
quelques conclusions et leurs implications

Un abstract :

N'est pas un résumé du travail.

Ne dit pas tout ce que le travail contient.

Ne développe pas toute l'argumentation et l'analyse de la recherche...

Ne dit pas tout mais donne envie de lire.

Mots-clefs : constraint programming, operations research, horaire de cours

Avant-propos

En préambule à ce travail de fin d'études, nous souhaitons adresser nos remerciements les plus sincères aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Nous tenons à remercier chaleureusement :
- Monsieur Lambeau, notre promoteur, pour le temps qu'il a bien voulu nous consacrer et sans qui ce mémoire n'aurait jamais vu le jour

- Monsieur Fauconnier, pour la grande patience dont il a su faire preuve et le temps précieux qu'il nous a accordé

- Madame Bonnave qui a eu la gentillesse de relire et corriger ce travail

- Monsieur Tonon, pour la correction de ce travail.

Table des matières

Introduction

Dans le domaine de l'enseignement, la création des horaires de cours est une tâche ardue pour les secrétariats. En effet, il s'agit de prendre simultanément en compte de très nombreuses contraintes ; les locaux ne sont pas toujours libres, les professeurs ont des *desiderata* particuliers, les élèves ont des parcours différents, etc. Cette somme de critères contraignants fait de la mise en place d'emplois du temps pour chacun des professeurs et, corollairement, des élèves, un travail long et fastidieux qui doit être réitéré à chaque rentrée académique.

Ce type de problème est l'objet de la programmation par contraintes, dite « constraint programming » [1]. Ce paradigme de programmation s'attache à traiter les problèmes où une large combinatoire est nécessaire pour trouver un ensemble de solutions satisfaisant les différents acteurs impliqués. Par exemple, la programmation par contraintes est notamment utilisée pour traiter les problèmes relatifs aux circuits d'attente aériens au-dessus des aéroports. Il s'agit de prendre simultanément en compte la quantité de kérosène restante pour chacun des avions, les pistes occupées, les prédictions météo, les « gates » libres, les ressources disponibles au sol, etc. Mais, la programmation par contraintes se retrouve aussi dans d'autres tâches telles que la gestion de la main d'œuvre, la mise en place de tournois sportifs de grande envergure, etc.

La recherche par contrainte constitue l'un des outils fondamentaux du domaine, plus vaste, de la recherche opérationnelle, dite « operations research » [2]. La recherche opérationnelle se donne pour objectif d'aider la prise de décision pour trouver une solution optimale face à un problème donné. Dans ce contexte, il est naturel que la programmation par contraintes, par sa prise en compte de critères multiples, trouve sa place comme méthode de résolution de problèmes.

C'est dans ce contexte que s'inscrit le présent travail qui vise à détailler la procédure de conception d'un outil dédié à la création d'horaires de cours pour l'EPHEC [3] et reposant sur la programmation par contraintes.

Madame Gillet, Directrice de l'établissement EPHEC à Louvain-la-Neuve, établit l'horaire à chaque nouveau semestre de l'année académique. L'établissement de cet horaire nécessite de considérer un grand nombre de contraintes ; citons notamment les *desiderata* exprimés par les professeurs, la nécessité qu'un cours se donne dans un local avec un matériel adapté au sujet qu'il traite [4] ou, encore, les impératifs liés au programme de cours des différentes promotions.

Toutefois, dans le cadre particulier de l'EPHEC, certaines difficultés supplémentaires sont à examiner. En effet, les professeurs externes, qui ont des charges d'enseignements dans d'autres établissements, doivent se voir attribuer des horaires spécifiques. De plus, certains cours sont dis-

1. Müller, T., Constraint-based Timetabling, Thèse de doctorat, Charles University in Prague, Faculty of Mathematics and Physics, 2005
2. Schärling, A. Décider sur plusieurs critères : panorama de l'aide à la décision multicritère, vol 1. PPUR
3. Ecole Pratique des Hautes Etudes Commerciales
4. Par exemple, certains cours nécessitent du matériel informatique pour leur bonne tenue.

pensés dans des locaux hors des murs de l'EPHEC qui ont des périodes de disponibilité variables qu'il est nécessaire de prendre en compte.

Sans une approche computationnelle, ces contraintes doivent être prises en compte manuellement par la personne établissant l'horaire. Or, cette considération minutieuse des contraintes est un véritable « casse-tête » nécessitant un effort cognitif important pour le résoudre.

La programmation par contraintes permet de résoudre de manière automatique cette problématique et facilite ainsi la tâche qui incombe à la personne en charge de l'élaboration d'un horaire. Ainsi, notre solution s'inscrit dans un cadre applicatif réel et utile face à la tâche chronophage de création d'horaires.

Afin de répondre à cette problématique, différents aspects ont dû être étudiés ; la notion de contrainte que nous décrirons de façon théorique, le paradigme de programmation correspondant et, enfin, les différents outils existants dédiés à ce domaine. Cependant, afin de rendre notre solution utile au plus grand nombre, deux aspects supplémentaires ont dû être travaillés.

Premièrement, il a été décidé d'orienter notre solution dans une visée de service Web. Ainsi, notre solution ne nécessite pas d'installation préalable à son utilisation. Et, en outre, ce choix permet de centraliser les données au sein d'une unique base de données.

Deuxièmement, dans une optique d'ergonomie et de facilité d'emploi, il a été choisi d'effectuer un travail important sur l'interface. En effet, la création d'horaires étant une tâche déjà complexe en soi, il nous a paru inutile d'alourdir la charge cognitive qu'elle nécessite en obligeant l'utilisateur à apprendre à manipuler notre solution.

Ces deux aspects ont apporté un lot de difficultés auquel notre solution a dû répondre. Citons notamment la nécessité d'une approche asynchrone dans la gestion de la base de données afin de garder une solution rapide. En effet, lorsqu'un horaire subit une modification, il est nécessaire de le changer dans la base de données. Cette demande faite à la base de données nécessite une réponse avant de pouvoir continuer. Or, le temps de réponse, au vu des traitements nécessaires, n'est pas toujours assez rapide. En conséquence, il a été nécessaire de trouver des solutions afin de ne pas ralentir l'ensemble du système.

De plus, la mise en place d'un service Web a aussi apporté la nécessité d'utiliser des librairies avancées. Par exemple, nous faisons un grand usage de Google Web Toolkit et Java EE. Ces deux outils proposent des méthodes orientées service Web, cependant, ils nécessitent un temps d'apprentissage plus long comparé à des solutions basiques.

D'autres difficultés ont été rencontrées au cours de notre travail et seront explicitées dans le suite de ce travail.

Afin d'étayer notre propos, ce présent travail se divise en sept sections :

- **La première section** présente la solution dans son ensemble. Y sont notamment décrites l'interface, l'implémentation effective des contraintes, l'utilisation de la mémoire cache, etc.

- **La deuxième section** expose le cadre technologique utilisé pour notre solution.

- **La troisième section** décrit la méthodologie sous-jacente à la conception de notre solution. Y sont notamment examinés l'approche en travail d'équipe et les outils nécessaires à cette approche.

– **La quatrième section** offre une présentation plus poussée des outils utilisés.

– **La cinquième section** montre les différents points importants dans notre code et dans la gestion de la base de données.

– **La sixième section** se destine à la programmation par contraintes et à l'implémentation qui en est faite au sein du code.

– **La septième section** propose quelques réflexions sur notre solution et présente des perspectives d'extension à notre travail.

Enfin, une conclusion achèvera ce travail et seront synthétisés les différents aspects saillants de notre solution.

Chapitre 1

Analyse de la problématique et outils existants

Afin de mieux comprendre la problématique liée à l'élaboration d'un horaire, nous avons tout d'abord pris rendez-vous avec Madame Gillet, directrice de l'EPHEC. Madame Gillet est la personne en charge de l'élaboration des horaires pour la partie Louvain-la-Neuve de l'EPHEC. Nous avons parcouru ensemble les différents problèmes et les différents types de contraintes auxquelles nous devrons faire face. Cette première analyse du problème avait pour but de nous orienter sur le type de contraintes intervenant dans notre travail.

Dans un premier temps, cette présente section, théorisera la notion de contrainte. Dans un deuxième temps, seront présentées les différentes contraintes rencontrées dans l'analyse du problème.

1.1 Une contrainte en théorie

Dans le cadre d'un horaire, nous devons faire face à des contraintes sur des domaines finis. Nous allons illustrer ce concept comme suite :

$$P = (X, D, C)$$

X étant nos variables. Celles-ci seront représentées par les attributions [1] dans lesquelles nous retrouverons les éléments suivants :

- nom du professeur
- le cours
- la classe

Nous pouvons l'écrire sous la forme Vn ou V correspond à la valeur du carton, et n sont numéro.

D correspond au domaine. Ici, il s'agit d'effectuer un horaire. Notre domaine sera le nombre de locaux disponibles multiplié par le nombre de périodes. Dans le cadre du cours du jour à l'EPHEC de Louvain-la-Neuve, nous avons 6 périodes par jour, durant 5 jours. Un local est donc disponible à 6 moments de la journée durant 5 jours/semaine. Il faut donc multiplier le nombre de local par 30. Ce domaine s'écrit donc sous la forme $NlxNp$.

C étant nos contraintes. Il existe différents types de contraintes. Dans la section suivante, nous

1. Les attributions sont les différents cours avec leurs propriétés intrinsèques. Dans le cadre de ce rapport, nous désignerons ces cours par l'appellation de cartons. Cette notion sera explicitée plus loin dans le présent rapport.

en énumérerons certaines pour bien se rendre compte du nombre important de celles-ci et dans quel moment elles interviennent.

Pour régler ce problème de contraintes, il est avantageux de prendre en compte certaines heuristiques ou métaheuristiques. Nous avons défini le problème rencontré à l'EPHEC comme étant de type NP-Complet.

La problématique de la programmation par contrainte est que les algorithmes de résolution tentent à trouver une solution `optimale` au problème. Si une contrainte ne peut être prise en compte lors de cette tentative, nous n'obtiendrons pas de résultat. Pour ce faire, il est nécessaire d'appliquer un filtrage de ce qui ne peut être pris en compte. Ce « mécanisme » s'appelle la `propagation de contraintes`.

La propagation de contraintes a pour objectif d'appliquer un filtrage sur les attributions posant problèmes et de les retirer du pool de cartons [2] pris en charge lors de la résolution de l'horaire. Il est nécessaire de prendre en compte le temps que le filtrage va prendre afin de garantir une meilleure rapidité de la résolution.

1.2 Les types de contraintes

Pour se rendre compte de l'ensemble des contraintes à prendre en compte lors de l'établissement de l'horaire, nous en avons énuméré un maximum dont voici la liste :

Locaux
Matériels : (transparent, audio, Projecteur)
Type de local (sale info, auditoire, TP)
Nombres de places disponibles
Situation géographique
Attribution aux classes d'élèves
Marie Curie Ou Place des Sciences

Professeurs
Desiderata
Statut (externe, plein temps, ...)
Local où se donne le cours
Exigences matérielles
Pas d'enfants dans la classe
Temps Louvain-la-Neuve ou Bruxelles

TABLE I – Contraintes sur les locaux et les professeurs

Cours
Étalement dans le temps
Cours qui se suivent obligatoirement (ex : TP)
Cours qui ne doivent pas se suivre (ex : AN+NDLS)
Groupe ou demi-groupe
Nombre d'heures total
Hebdomadaire ou bimensuel

Attribution (ou carton)
Doit pouvoir ignorer les contraintes
Doit pouvoir ajouter des contraintes

TABLE II – Contraintes sur les cours et les attributions

2. Le pool de cartons présente l'ensemble des cours qu'il est possible de positionner dans le semainier. Cette notion de pool de cartons sera vue dans le prochain chapitre.

Contraintes générales
Éviter les heures de fourches
Si possible, garder les élèves dans un même local et faire déplacer le professeur
Ordre sur les contraintes (poids sur celle-ci) afin de favoriser un externe plutôt qu'un interne
Attribuer les locaux informatiques à un jour donné pour une option
Séparer eu deux semestres
Prendre en compte les jours de congé
Directrice doit définir demi-classe, groupe, etc. pour un cours
Il doit être possible de reporter un cours
Il doit être possible de définir une pause

TABLE III – Contraintes générales

Pour des améliorations futures, d'autres contraintes pourront éventuellement être prises en charge comme :

Étudiants
Répartition en fonction du sexe (sauf TI)
Répartition en fonction de la provenance
Possibilité de changer de classe

Classes
Nombre d'étudiants maximum
Locaux adaptés

TABLE IV – Contraintes sur les étudiants et les classes

En conclusion, nous pouvons voir que le nombre de contraintes à prendre en compte pose un réel problème quand elles doivent être traitées par un humain. Les possibilités, les préférences sur certains types de contraintes devant être absolument prises en compte ou pas, relève d'une réflexion de haut niveau. Les ordinateurs "réfléchissant" de manière plus "mathématique" son en théorie plus apte à répondre de ce genre de problème.

1.3 Analyse des applications existantes

Nous avons fait également une petite étude sur les applications déjà existantes permettant d'élaborer des horaires avec contraintes. Nous pouvons citer `EDT`, programme très repandu dans les écoles, `aSc` Horaire possédant une partie automatisée et manuelle pour la création d'horaire ainsi que `Université time tabling` (UniTime). Cette dernière est une application très prometteuse, mais ne propose que l'établissement d'horaire automatique. Aucune façon manuelle de faire n'est proposée. Il utilise sa propre librairie pour résoudre la problématique des contraintes.

1.4 La programmation par contraintes

Nous avons abordé les contraintes de manière théorique, les avons énumérées et cité les applications déjà existantes, qu'en est-il de l'application de cette théorie ? Pour pouvoir programmer

par contraintes il existe plusieurs libraire. Certaines propriétaires et d'autres open source. Notre travail ayant pour objectif d'être libre et open source, nous nous sommes intéressés à ces dernières et en avons testé certaines :

- GeCode
- Google OR tools
- Python Constraint

Python constraint, constitue une bonne approche de la problématique, le code étant clair et concis. Celui-ci sera le plus parlant pour illustrer la programmation par contraintes de manière plus concrète.

```
from constraint import *
problem = Problem()
problem.addVariable("a", [1,2,3])
problem.addVariable("b", [4,5,6])
problem.getSolutions()
[{'a': 3, 'b': 6}, {'a': 3, 'b': 5}, {'a': 3, 'b': 4},
 {'a': 2, 'b': 6}, {'a': 2, 'b': 5}, {'a': 2, 'b': 4},
 {'a': 1, 'b': 6}, {'a': 1, 'b': 5}, {'a': 1, 'b': 4}]

problem.addConstraint(lambda a, b: a*2 == b,
                      ("a", "b"))
problem.getSolutions()
[{'a': 3, 'b': 6}, {'a': 2, 'b': 4}]

problem = Problem()
problem.addVariables(["a", "b"], [1, 2, 3])
problem.addConstraint(AllDifferentConstraint())
problem.getSolutions()
[{'a': 3, 'b': 2}, {'a': 3, 'b': 1}, {'a': 2, 'b': 3},
 {'a': 2, 'b': 1}, {'a': 1, 'b': 2}, {'a': 1, 'b': 3}]
```

FIGURE I – Code python

Debug : Mettre une explication du code ici

Dans le cadre de notre solution, nous avons utilisé `GeCode` et `Google OR tools`. Ces deux outils sont très puissants et nous ont permis de répondre dans une certaine mesure aux contraintes explicitées.

1.5 Rencontre

Dans le cadre de ce travail, nous avons eu l'occasion de rencontrer monsieur Pierre SCHAUS qui a effectué une thèse de doctorat à l'UCL sur la problématique des contraintes. Notre rencontre nous a permis de mieux cerner la problématique.

Suite à cet entretien, nous avons pu mieux nous rendre compte de la problématique et des

vigilances à prendre lors de la programmation par contraintes. Le solveur ne se contente pas d'être une librairie dans laquelle il suffirait d'appeler des fonctions pour résoudre un problème et avoir un résultat. Les choses doivent se faire petit à petit, car si le solveur ne trouve pas de résultat, il n'est pas possible de savoir pourquoi celui-ci n'a pu en trouver un.

Il faut donc y rentrer nos contraintes pas à pas. Certaines contraintes étant plus importantes que d'autre, il serait également utile de pouvoir leur mettre un poids. Ceci n'étant pas possible [3], il a fallu trouver un autre moyen. Monsieur Pierre SCHAUS nous a alors conseillé d'utiliser une approche par hiérarchisation de nos contraintes. Cette hiérarchie a été exécutée en utilisant un arbre de contraintes. Celles-ci seront ensuite rentrées dans le solveur suivant leurs importances. Monsieur SCHAUS étant entrain d'élaborer ses propres librairies, nous à montrer quelques exemple concret de l'utilisation de celle-ci et de la marche à suivre.

Il nous a ensuite fourni cette librairie, cette dernière n'étant pas encore distribuée actuellement [4]. Cependant, cette dernière n'était pas adaptée à notre problème précis [5]. En conséquence de quoi, nous nous sommes tournés, dans un deuxième temps, vers `GeCode` et `Google OR tools` pour faire quelques tests et élaborer un premier prototype.

1.6 Décision

Cependant, à la fin de l'année, pour le prototype final, nous nous sommes tournés vers la librairie `uniTime` et avons exploré de manière plus approfondie ce que l'application propose. Nous nous sommes basés sur cette dernière pour l'élaboration de notre solveur. En effet, les possibilités que cette librairie offre correspondent parfaitement aux contraintes de type horaire. Nous parlerons plus en détail de cette librairie dans la suite de ce travail.

3. L'implémentation actuelle proposée ne permet pas de mettre un poids d'importance sur les contraintes.
4. Mais elle sera disponible gratuitement et distribuée en open source sous peu.
5. Cette librairie n'est pas adaptée à un problème NP-Complet car il est impossible d'y ajouter des heuristiques.

Chapitre 2

Présentation de l'application

Notre application, surnommée Betty [1], est un outil permettant d'élaborer un horaire de façon plus conviviale et plus rapide comparé à la façon actuelle de procéder à l'Ephec.

Cependant, avant tout développement ultérieur, il est nécessaire de présenter les différents objets composant notre solution. Nous avons choisi de travailler par `projet`. Chaque projet est caractérisé par un ensemble de cours étalés sur deux semestres. Betty permet de gérer plusieurs projets. Dans une optique de confidentialité, l'accès à chaque projet nécessite une `inscription` et une `connexion` de la part de l'utilisateur.

Au sein de chaque projet, chaque cours est représenté par ce que nous appelons un `carton`. Un carton est un cadre décrivant le nom du cours, son sigle et le professeur qui lui est rattaché. Betty propose par un système intuitif de « drag and drop » de positionner ces cartons au sein d'un `semainier` afin de construire un horaire.

Sur cette page présentant le semainier, différents outils sont utilisables afin d'améliorer la création de l'horaire. À chaque cours sont rattachés ses `attributions` spécifiques, c'est-à-dire les différentes classes d'élèves auxquelles se destine le cours. Un système de `notifications` a été mis en place afin d'offrir à l'utilisateur un retour sur les opérations qu'il effectue. Enfin, un système de `filtres` améliore la vision des données.

La programmation par contraintes est utilisée pour définir de nouveaux horaires. Cette utilisation des possibilités s'effectue au travers d'un panneau d'`instances`. Une instance est un état spécifique des cartons, c'est-à-dire un horaire potentiel. Lorsque l'utilisateur l'estime nécessaire, il est possible d'utiliser le `solveur`. Le solveur est le système qui est en charge de calculer un nouvel horaire, c'est-à-dire une nouvelle instance, répondant à certaines contraintes.

Afin de diminuer les temps de chargement, un mécanisme permet d'exploiter la `mémoire cache` des navigateurs côté client. Ce procédé permet de minimiser le nombre de requêtes envoyées au serveur et améliore la rapidité de l'ensemble du système.

Cette présente section décrira ces différents aspects.

1. Brillant Ephec Time Tabling for You

2.1 Connexion et Inscription

La première page de l'application propose à l'utilisateur d'entrer son nom d'utilisateur et son mot de passe. Le mot de passe est crypté en SHA-256 [2], dans la base de données. La page propose également de vous inscrire via la flèche en haut à droite de la fenêtre.

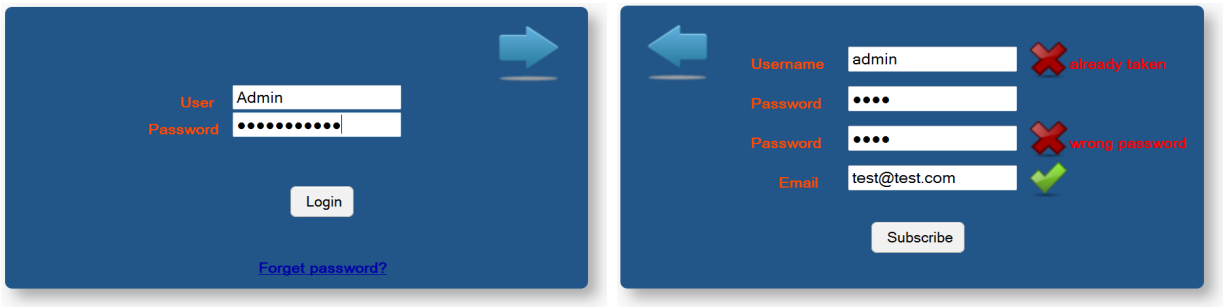

FIGURE II – page de login et d'inscription

Lors de l'inscription, nous invitons l'utilisateur à entrer sont nom d'utilisateur, mot de passe et un email. Les informations entrées sont soumises à des vérifications d'usage, le maximum de ces vérifications étant effectué du côté client afin de réduire l'échange avec le serveur.

Pour cette partie, quelques améliorations peuvent être apportées tel que le changement de mot de passe ou la récupération de celui-ci par envoi de mail.

Debug : capture d'écran

2.2 Page des projets

La page est ordonnée de façon à avoir toujours le dernier projet en date créé en haut de la liste. Un projet est présenté de la manière la plus concise possible afin de ne pas perdre l'utilisateur. Ce dernier à la possibilité de créer un nouveau projet [3], choisir le semestre à élaborer et de supprimer un projet. L'application devrait dans une prochaine mise à jour proposer des options sur celui-ci tels que le partage de projets entre plusieurs utilisateurs.

Lors de la création d'un nouveau projet, celui-ci doit être nommé et contenir les fichiers nécessaires à l'élaboration de l'horaire. Ces fichiers se présentent sous la forme d'un .xls contenant pour le premier, la liste des attributions de chaque cours, ce fichier est le résultat d'une requête SQL et nous a été fourni par l'Ephec. Le deuxième représente la liste des locaux disponibles, ce fichier n'existant pas en tant que tel, à initialement été créé par Madame Vroman, Professeur à l'Ephec, dans le cadre du "projet horaire" du cours de langage avancé de programmation de deuxième année. Nous y avons ajouté des informations pouvant être prises en compte par le solveur, et permettant de facilité l'établissement manuel d'un horaire.

Une fois le projet créé et le quadrimestre sélectionné, l'utilisateur est redirigé vers la page principale de l'application dans laquelle l'horaire pourra être créé.

2. Secure Hash Algorithm 256 bits
3. Option également disponible via le menu

FIGURE III – représentation d'un projet

2.3 Page principale

La page principale se présente comme suit :

- Au nord de la page, nous trouvons les différents filtres et options disponibles tels que :
  1. Card filter
  2. Sélection de la grille à afficher
  3. Un panneau regroupant les différentes instances du projet
- À l'ouest, les cartons créés sur base du fichier des attributions
- Au centre, le semainier où les cartons pourrons venir se glisser
- À l'est, un panneau de notifications permettant d'avoir un suivi des différentes actions faite par l'utilisateur.

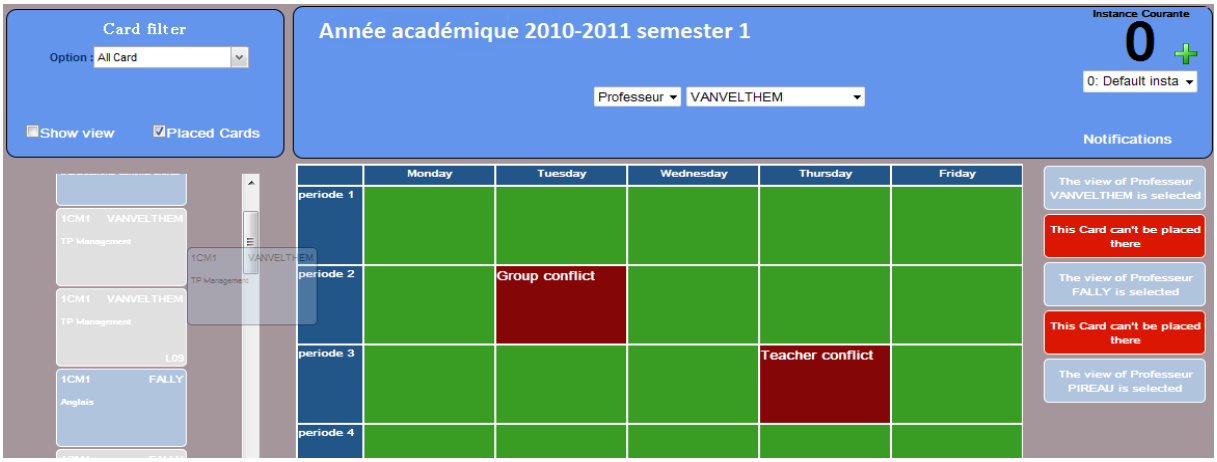

FIGURE IV – page principale

2.3.1 Les attributions

La mise en forme des cartons se base sur le design de ceux actuellement employés à l'Ephec lors de l'établissement manuel de l'horaire. Ceux-ci comportent le/les classe(s) assignée(s) à chaque cours donné par un professeur.

À chaque carton est assigné un ensemble de locaux où pourront se donner les cours. Par exemple, un carton de type informatique sera assigné uniquement aux locaux de type informatique. Lorsque le carton est déposé, la solution lui assigne un local disponible parmi cette liste, et nous pouvons voir apparaître le nom du local en bas à droite du carton.

2.3.2 Semainier

Nous affichons dans le semainier, les informations relatives à la personne, classe ou au local choisis via le filtre prévu à cet effet. La grille se colorie en fonction du carton qui est sélectionné. Un code de couleurs a été mis en place permettant de distinguer si un carton peut être placé ou pas. Par exemple, si l'on se trouve dans la vue d'une classe et que l'on prend un carton d'une autre classe, toute la grille se coloriera en rouge, montrant à l'utilisateur que celui-ci ne peut pas être placé.

Nous distinguons trois groupes de couleurs ; vert, orange et rouge. Ces groupes sont eux-mêmes subdivisés en trois catégories : couleur claire, normale et foncée. Lorsqu'on colorie la grille horaire, il arrive parfois qu'un carton puisse être placé à plusieurs endroits, les uns plus avantageux que d'autres pour la suite de l'établissement de l'horaire, il est donc nécessaire de pouvoir dire à l'utilisateur que le carton peut être placé, mais l'orienter sur un choix plus adéquat. Ce code de couleurs est utilisé dans une version limitée pour le moment.

2.3.3 Notifications

Les notifications sont un support pour l'utilisateur. Lorsque celui-ci effectue une action comme supprimer/ajouter un carton il est nécessaire de savoir si l'action c'est effectuée correctement. À la place d'un popup intrusif, nous avons opté pour ce système, signalant à l'utilisateur de manière plus douce l'état d'actions qui ne peuvent être vues via une interface graphique.

Nous distinguons ici deux couleurs différentes, une couleur se fondant au thème général de l'application lorsque tout s'est effectué correctement et une couleur rouge pour signaler un problème. Ce système est limité et pourra être amélioré dans le futur.

2.3.4 Les filtres

Les différents filtres permettent de faciliter la création de l'horaire de façon manuelle. Si nous voulons créer l'horaire d'un professeur en particulier, avoir les cartons de tous les professeurs rendrait la tâche plus lourde à l'utilisateur. L'utilisation de ce filtre permet d'avoir une meilleure vue sur ce qui doit être placé. Nous parlerons de vue « vue ».

De même, il est possible d'afficher/masquer les cartons déjà placés. Lorsqu'on filtre les cartons d'une classe et que tous ces cartons sont placés, ils ne font théoriquement plus partie de la liste des cartons et donc l'utilisateur n'a pas la possibilité de savoir si ladite classe possède des cartons.

Ce système favorise aussi la vue d'ensemble sur ce qui a déjà ou non été placé.

Une dernière option est de pouvoir automatiquement « switcher » sur la grille horaire correspondant au filtre mis sur les cartons. Si cette option est sélectionnée et que nous filtrons les cartons de la classe *3TL2*, nous supposons ici que l'objectif est de placer les attributions de cette classe. Par conséquent, seule la grille horaire des *3TL2* est affichée.

L'application a donc été pensée afin de minimiser l'effort cognitif de l'utilisateur et de lui offrir les outils adéquats pour se concentrer sur son seul objectif ; la création de l'horaire.

2.3.5 Les instances

Le panneau d'instance permet, au sein d'un même projet, de créer un ensemble de sous-projets. L'objectif principal de ce panneau réside dans l'utilisation du solveur. Celui-ci sera lancé dans une nouvelle instance, permettant à l'utilisateur de continuer son horaire manuellement dans une autre sans être bloqué. Une fois la résolution finie, l'utilisateur peut naviguer entre les différentes instances afin de voir les différents résultats obtenus.

C'est ici que l'implémentation des instances y trouve sa principale utilité. Toutefois, il serait envisageable, comme perspective d'extension à ce travail, de pouvoir comparer deux instances entre elles.

Debug : Capture d'écran

2.3.6 Le solveur

Pour lancer le solveur, nous devons aller dans le menu `project > solveur > solve`. Une fenêtre s'ouvre permettant à l'utilisateur de sélectionner l'instance dans laquelle doit s'effectuer la résolution. Une fois le solveur lancé, une notification apparaît à l'utilisateur lui notifiant que celui-ci est exécuté. À la fin de tentative de résolution, l'utilisateur reçoit une notification lui spécifiant la fin du travail et la bonne ou mauvaise application de ce dernier. Le solveur, dans sa version actuelle, fonctionne correctement, mais ne propose, à l'heure actuelle, aucune option de configuration et fonctionne sur des petits projets.

2.3.7 Mémoire cache

L'application utilise la mémoire cache du navigateur pour stocker les données, ainsi nous minimisons les requêtes vers le serveur. Ceci pourrait être également exploité pour pouvoir travailler sur l'application sans connexion internet. Les données nécessaires à l'établissement de l'horaire étant stockées dans la partie cliente.

Chapitre 3

Justification des choix technologiques

3.1 Client/Serveur

Notre application se base sur une architecture client/serveur pour les multiples avantages que celle-ci apporte.

Premièrement, l'EPHEC étant un établissement possédant plusieurs sites, une application client/serveur permet d'avoir un centre de données commun.

De cette manière, un horaire établi à Louvain-la-Neuve pourra être pris en compte lors de l'établissement d'un horaire à Bruxelles. L'avantage de laisser les charges de travail effectuées sur le serveur [1] permet que l'élaboration de l'horaire puisse se faire sur des machines possédants peu de ressources. Par exemple, il sera possible d'établir l'horaire sur une tablette ou encore sur un smartphone.

Deuxièmement, l'application n'est pas dépendante du système d'exploitation mis en place sur l'ordinateur client. L'horaire peut être débuté sur une machine Windows pour ensuite être continué sur une tablette. Les mises à jour de l'application sont aussi complètement transparentes pour l'utilisateur final. Pas besoin de télécharger et d'installer les mises à jour comme sur un logiciel orienté desktop.

De même, toujours en comparaison avec une application desktop, s'il survient un problème avec la station de travail, il est garanti de pouvoir retrouver ses données dans leur entièreté et l'horaire peut continuer à être établi sur une autre station. En outre, les serveurs offrent de nombreux avantages (duplication des données, séparation sur plusieurs sites,etc.) qu'un ordinateur ne peut pas toujours offrir.

Troisièmement, outre cet aspect de facilité pour la partie cliente, il est aussi très facile d'administrer l'application. Celle-ci étant portable et facile d'installation. Pas besoin de connaissances approfondies ou de configurations spéciales du serveur. Il suffit d'installer un serveur [2] et d'y mettre l'application [3] dans le bon dossier.

De façon analogue, la base de données peut être complètement indépendante de l'application et peut se trouver sur un serveur externe. Ainsi, elle peut, par exemple, être interne à l'EPHEC

1. Solveur, actions lourdes, etc.
2. Un serveur permet de faire tourner une application web Java comme : Tomcat, Jetty, etc.
3. Sous forme de '.WAR'

et l'application peut se trouver sur un serveur public externe. En outre, Le type de SGBD [4] utilisé a peu d'importance. L'application se charge de la créer d'elle-même grâce à l'utilisation de l'outil Hibernate. Dans notre solution Hibernate utilise JDBC [5].

Toutefois, il est tout à fait possible de modifier JDBC par une autre interface en téléchargeant le driver correspondant sur le site d'Orale [6]. Actuellement, Oracle propose 221 drivers différents.

Ainsi, l'application ne nécessite aucune configuration spécifique sur un ordinateur client. De plus, au niveau du serveur, nous avons fait différents tests utilisant notamment `MySQL` ainsi que du `PostgreSQL` et nous n'avons pas rencontré de problèmes de création.

3.2 Java

Le choix du langage Java s'est fait naturellement. Celui-ci est très présent dans le domaine du développement en entreprise. De plus, le langage Java permet de bien structurer son programme ; il fait preuve d'une certaine rigueur, de robustesse du à son typage statique et fort. Nous avons pu tirer avantage de ces derniers points pour élaborer notre application.

Toutefois, il est nécessaire de souligner que, dans nos premiers prototypes, nous avons utilisé le langage Python. Python offre beaucoup de possibilités et est aussi adapté au type d'application que nous voulions élaborer [7]. En outre, il possède une structure favorisant les bonnes pratiques, ce point étant particulièrement important pour nous.

Cependant, au final, le choix du langage Java s'est imposé. Premièrement, le solveur est codé en Java. En effet, bien que nous soyons partis sur l'idée de faire du binding grâce à l'utilisation de SOAP [8] entre le Python et le solveur, dans un souci de clarté du code, nous avons préféré rester dans le même type de langage. Deuxièmement, l'utilisation de Google Web Toolkit nous a aussi confortés dans ce choix. Troisièmement, il existe plusieurs solutions en Java pour la mise en place de serveur. Nous utilisons un serveur Tomcat, écrit lui-même en Java et multiplate-forme.

3.3 Google Web Toolkit

Nous avons choisi de travailler avec Google Web Toolkit (GWT) pour les nombreux avantages [9] que celui-ci apporte.

Premièrement, GWT présente l'avantage de permettre de coder la partie cliente de l'application en Java et de sérialiser ce code en JavaScript (JS) afin de l'exécuter sur le navigateur côté client. Ainsi, le code généré par GWT peut être adapté aux différents navigateurs les plus répandus à ce jour tels que Chrome, Firefox, Internet Explorer, etc.

Deuxièmement, la partie serveur étant réalisée en Java, le choix de GWT, utilisant du Java, permet de présenter une solution cohérente. En effet, GWT utilise du Java EE du coté serveur, langage ayant déjà fait ces preuves et très répandu dans le monde professionnel.

Troisièmement, GWT propose également l'utilisation d'outils tels que Jin and Guice qui feront

4. Système de Gestion de Base de Données
5. Java Data Base Connectivity
6. `http://developers.sun.com/product/jdbc/drivers/`
7. Comme explicité précédemment, la libraire `Python Constraint` a été utilisée dans un premier temps.
8. Simple Object Access Protocol
9. cfr. Chapitre 4.1.1 - GWT

l'objet d'un autre point. En effet, comme son nom l'indique, GWT est une vraie boite à outils qui permet de déployer de façon efficace les bases pour une solution Web.

3.4 HTML5, CSS3

Nous avons utilisé HTML5 et CSS3 en respectant leurs dernières normes en date. GWT permettant d'utiliser ceux-ci, nous avons décidé d'utiliser certaines fonctionnalités intéressantes que ces derniers proposent tels que l'utilisation d'un local storage [10] et qu'un rendu graphique générique et de qualité [11].

3.5 Hibernate

Pour la communication avec la base de données, nous utilisons l'outil Hibernate. Le réel avantage de Hibernate est de pouvoir utiliser les données selon un paradigme objet.

En effet, Hibernat permet de représenter les tables de la base de données en objets et facilite ainsi l'utilisation des données dans notre code en Java [12].

Dans notre solution, Hibernate est une couche supérieure à JDBC. Cependant, il peut être utilisé avec n'importe quel type de SGBD et permet la création automatique des tables.

3.6 Solveur

Après analyse des différentes libraires, nous nous sommes orientés sur les librairies proposées par le projet `Unitime`. Celles-ci étant beaucoup plus adaptées à nos besoins, qui sont de pouvoir gérer des contraintes.

Elle sont écrites en Java et sont orientées pour la problématique de la création d'horaires d'un établissement scolaire avec des contraintes pondérées [13]. En effet, les autres librairies étant plus orientées contraintes pures.

Les librairies `Unitime` proposent des options de configurations correspondant à nos besoins et aux besoins d'un établissement tel que l'EPHEC.

En définitive, nous avons choisi les librairies de `Unitime` pour des raisons de performance et de correspondance face au problème analysé.

3.7 GitHub

La solution proposée dans ce travail repose sur un large travail d'équipe. Cet aspect inhérent à notre travail, nous a conduits à utiliser GitHub afin de permettre une production collaborative, un suivi des modifications, un système de fusion des travaux et, enfin, un mécanisme de suivi des bugs.

10. Le HTML5 offre un local storage permettant de stocker les données directement coté client.
11. Le CSS3 est utilisé pour le rendu graphique de l'application.
12. Java est intrinsèquement lié à un paradigme objet.
13. Par exemple, les desiderata des professeurs ne doivent pas être vues comme des contraintes pures mais comme des contraintes permettant d'orienter le résultat final.

Chapitre 4

Cadre technologique

4.1 Développement du côté client

4.1.1 Google Web Toolkit (GWT)

GWT est un outil open source mis en ligne par Google permettant de développer des applications web avancées. En utilisant cet outil, nous pouvons développer des applications AJAX [1] en langage Java. Lors de l'élaboration d'une application, le cross-compiler de GWT traduit la partie cliente de l'application java en fichiers JS. Ces fichiers peuvent être soumis à des optimisations (obscurcir le code, séparation du JS en plusieurs fichiers, etc.). GWT n'a pas pour objectif d'être "just another library" [2] mais possède sa propre philosophie.

Pour programmer une application web de notre type, il faut maîtriser des outils comme l'AJAX, l'HTML ainsi que le CSS. Le problème principal de ces outils réside dans leur compatibilité entre les différents navigateurs. La façon de mettre en forme un site web n'aura pas toujours le même sur ceux-ci. Il faut prendre en compte aussi la complexité d'utilisation de ces langages de manière avancée (utilisation du DOM en HTML, etc.). Le langage JS est assez complexe d'utilisation, surtout pour l'écriture de grosses applications [3].

Pour pallier à ces problèmes, GWT a été créé. Il a été élaboré dans le but de répondre à un besoin et non pas de proposer une autre librairie standard. Il s'agit d'une vraie boite à outils qui propose des solutions de développement répondant aux besoins du programmeur.

GWT est utilisé par exemple dans des outils web de Google comme Gmail. Nous pouvons aussi citer l'entreprise Rovio à l'origine du jeu à succès Angry Bird ayant développé une version HTML5 du jeux à l'aide de GWT [4].

4.1.1.1 Principe de fonctionnement

GWT possède un plugin Eclipse que nous avons utilisé dans la conception de notre solution. Il est toutefois intéressant de signaler qu'il existe des plugins GWT pour d'autres IDE [5] tels que

1. Asynchronous JavaScript And XML
2. GWT est un ensemble d'outils qui forment une plateforme et non pas simplement une librairie facilitant le travail.
3. Le debuggage de celles-ci étant assez fastidieux puisqu'il s'agit d'un langage interprété.
4. cf. https ://developers.google.com/web-toolkit/casestudies/
5. Integrated Development Environment

NetBeans, JDeveloper, etc. Ce plugin, dans sa version Eclipse, permet d'intégrer les outils GWT à l'IDE et de faciliter son utilisation pour l'utilisateur en automatisant, par exemple, la structure des différents packages java.

Comme nous l'avons expliqué précédemment, le principe de fonctionnement de GWT est de pouvoir créer des applications web en java basées sur un modèle client/serveur et de sérialiser la partie cliente de l'application en JS [6]. Un autre aspect de se fonctionnement réside dans la compilation pouvant être effectuée pour un ou plusieurs navigateurs spécifiques, favorisant la compatibilité et l'homogénéité du logiciel sur chacun d'entre eux. Ces deniers chargent uniquement ce qui les concerne. En plus de ces différents aspects qu'offre GWT, ce dernier permet aussi de préciser quelles sources du code [7] prendre en compte pour un navigateur spécifique.

4.1.1.2 Mode de fonctionnement

Nous distinguons deux types de fonctionnement. Le mode de développement ainsi que le mode de production.

Mode de développement

Le mode de développement consiste à compiler les sources du projet. Celui-ci n'est pas retranscrit en JS mais est directement exécuté en byte code. Ceci afin de permettre le debuggage de l'application. GWT compile tout de même le projet en JS-HTML-CSS afin de pouvoir valider le projet.

Pour pouvoir utiliser le mode de développement, il est nécessaire d'installer au préalable le plugin de développement sur le navigateur web. Ce plugin permet de capturer les événements et actions venant du client pour ensuite les envoyer vers le serveur local lancé au moment de l'exécution du projet.

Mode de production

Le mode de production correspond au code JS généré par le compilateur GWT. Le compilateur crée le JS, HTML et CSS à partir des sources du projet. Ce code correspond à l'application finale qui pourra être déployée sur notre serveur Tomcat.

4.1.1.3 Architecture GWT

L'architecture d'un projet GWT se fait sous la forme client/serveur. Nous allons analyser comment se déroule la communication entre ces deux parties. Nous distinguons deux types de communication dans l'application.

- Client/Client : communication entre les différentes vue de l'application
- Client/Serveur : utilisant le protocole RPC [8]

6. La partie serveur restant elle, en java
7. correspondant au .CLASS généré lors de l'exécution du code Java.
8. Remote Procedure Call

Évènement

Afin de pouvoir gérer la communication entre les différentes vues de l'application, GWT utilise un système d'envoi d'évènements, dits « events ». Ceux-ci permettent aux vues de dialoguer entre elles. Par exemple, une application GWT peut posséder un en-tête. Celle-ci est statique et n'est pas rechargée entre les différentes vues. Lors de l'identification, les informations de connexion peuvent être envoyées à la vue suivante pour spécifier à l'utilisateur le nom du compte sur lequel il est connecté. Les events sont enregistrés auprès de l'eventbus. Cet eventbus peut être écouté pour récupérer les informations voulues.

Actions

Les actions ressemblent aux Events, à l'exception que celles-ci sont envoyées au serveur. Elles permettent, par exemple, de faire des requêtes vers la base de données pour recueillir certaines informations. Pour rester dans le même exemple, lorsqu'un utilisateur se connecte à l'application en spécifiant sont identifiant et son mot de passe, ceux-ci doivent être vérifiés dans la base de données qui va renvoyer la liste des projets qui lui sont assignés. Cette méthode se fait à l'aide du protocole JSON [9]/RPC.

Il est primordial de souligner que les appels se font de manière asynchrone ce qui permet de ne pas bloquer le client lors d'un appel de procédure. Cet aspect a fait l'objet d'un travail transversal tout au long du développement.

4.1.1.4 Avantages de GWT

Les avantages de GWT sont nombreux :

- **Facilité d'utilisation**
  Le premier avantage que nous citerons est la facilité d'installation et d'utilisation de l'outil. Ce dernier ne nécessite pas de configuration fastidieuse. En outre, cet outil offre la possibilité de coder son application à l'aide d'un langage de haut niveau. Cet aspect facilite le développement général.

- **Debbugage**
  GWT permet un debuggage rapide du code, ce dernier étant codé en java et non pas en JS qui est un langage interprété.

- **Optimisation**
  GWT permet notamment l'optimisation et l'obfuscation du code JS. De plus, afin d'améliorer la rapidité des échanges, le code JS est compressé, mis en cache et séparé en différents fichiers.

- **Composants génériques**
  GWT offre à la réutilisation différents widgets prédéfinis. Ainsi, il n'est pas nécessaire de réécrire certaines parties basiques telles qu'un bouton, une boite de dialogue, etc. En outre, GWT offre la possibilité d'adapter ses widgets afin d'en créer de nouveaux.

9. JavaScript Object Notation

– **Sérialisation du code**
Le code Java est traduit en code JS et automatiquement adapté au navigateur de notre choix.

– **JSNI**
GWT présente une JSNI [10], c'est-à-dire une interface nativement implémentée en JavaScript. Cet aspect offre la possibilité d'utiliser directement du code JS au sein même de notre application. Il est donc tout à fait possible d'utiliser des librairies externes entièrement codées en JS telle que Jquery.

– **Utilisation de GinJector and Guice**
GinJector and Guice, n'étant pas un principe propre GWT, permet de créer une indépendance entre les différentes classes du projet. Ainsi, si nous voulons utiliser un objet ou une classe dans une autre, il n' y a pas besoin d'instancier cette nouvelle classe. Nous pouvons simplement injecter.

4.1.1.5 Inconvénients de GWT

Le principal inconvénient que nous avons pu noter suite à notre expérience personnelle est que lorsque l'application atteint un état d'avancement plus prononcé, il devient très lourd et très lent de tester son application en mode développement. La JVM [11] devant traduire le code est très lente. Il nous a fallu en moyenne 5 minutes pour charger une page, et nous avons eu fréquemment des erreurs de type « out of memory » du à la lourdeur de la tâche effectuée par l'ordinateur en mode développement.

Pour certains types de test, nous nous sommes retrouvés dans l'obligation de déployer l'application sur notre serveur Tomcat, car la sérialisation du Java vers le JS étant très lourde, certaines parties devaient être testées directement en mode production (notamment le drag and drop, etc.).

De même, lorsque l'application doit charger une grande quantité d'informations (grand nombre de cartons, professeurs,etc.) la page met beaucoup de temps à s'ouvrir.

Autre inconvénient, le temps de compilation du logiciel. Celui-ci peut être très fastidieux en fonction des paramètres demandés.

Un autre inconvénient réside dans le nombre assez limité de widgets fournis de base dans l'application. Pour certaines fonctionnalités plus avancées nous avons dû avoir recourt à des librairies externes, bien que celle-ci ne soit pas aussi performante (comboBox avec CheckBox, etc.).

4.1.2 GWT-Designer

GWT-Designer est un outil permettant de créer de manière efficace les interfaces graphiques. Ces dernières sont créées via un fichier XML [12] et sont retranscrites en code Java.

Ce code XML est éditable « manuellement » ou peut être créé à l'aide d'une interface de « drag and drop » ou les composants (widgets) peuvent être sélectionnés.

L'avantage d'utiliser un tel outil est double ; il permet de faire une distinction entre les différentes parties du code [13] et il permet d'utiliser le format standard XML.

10. JavaScript Native Interface
11. Java Virtual Machine
12. eXtensible Markup Language
13. Il y a plus de 13 000 lignes de code, fichiers java et xml compris.

4.1.3 GWT - Plateform

GWT - Plateform est un framework MVP [14] permettant de mieux structurer notre projet en respectant une certaine méthode. Ce framework s'incorpore directement à l'IDE et est dépendant de GWT.

Le MVP se base sur le MVC [15]. Celui-ci est un design pattern fournissant une manière d'élaborer des interfaces graphiques. Il est séparé en trois parties ; le modèle de données, les différentes vues de l'application [16] et le présenteur [17].

La partie intéressante de ces deux modèles réside dans l'utilisation d'un contrôleur/présenteur où transite les informations. Ainsi, l'application est soumise à leur contrôle ce qui permet d'avoir une application plus robuste. Dans le MVC, le contrôleur s'occupe de gérer les événements. La logique de contenu (Rendering logic) se passe directement entre le modèle et la vue. En MVP, cette logique est gérée par le présenteur. En définitive, rien ne transite directement entre l'interface et le modèle mais tout est soumis au contrôle du présenteur.

4.1.4 Libraires externes

Pour la création d'une interface plus avancée, nous avons du nous diriger vers des librairies externes.

4.1.4.1 GWT-DND

Nous avons utilisé GWT-DND qui comme son nom l'indique nous a permis d'implémenter le « drag and drop » sur les cartons. Le « drag and drop » fournis par cette librairie nous permet de capturer les Events de la souris mais aussi les Events touch permettant de garder la possibilité qu'offre GWT d'être utilisé sur des appareils mobiles. Ce qui est non négligeable à l'heure actuelle où les smartphones et tablettes ont une place prépondérante sur le marché.

L'outil est simple d'utilisation. Pour pouvoir rendre dragable notre widget, il est nécessaire de créer un "dragControler". Les widgets s'enregistrent auprès de ce contrôleur. Une fois cette opération effectuée, il faut pouvoir définir une zone où ces widgets pourrons être dropés. Pour se faire, nous créons un nouveau "dropControler", celui-ci s'enregistre aussi auprès du dragControler pour signifier qu'il est prêt à recevoir les widgets.

4.1.4.2 Smart-GWT

Smart-GWT est un wrapper [18] de la librairie JS SmartClient. Elle propose un grand nombre de widget venant s'ajouter à ceux fournis par GWT. Puisque Smart-GWT n'est qu'un wrapper de smartClient, elle ne respecte pas l'idée de base de GWT voulant que le code soit écrit totalement en Java et, ensuite, traduit en JS. Nous avons pu noter que les widgets proposés par cette librairie ne sont pas aussi réactifs que ceux de GWT.

Ces deux libraires ont été spécialement conçues pour être utilisées avec GWT. Il ne s'agit pas

14. Model View Presenter
15. Model View Controler
16. Au sein de l'interface du client
17. Le présenteur correspond au contrôleur du MVC
18. Un adaptateur.

de librairie JS comme Jquery, mais bien de librairie [19] orienté GWT. Afin de pouvoir utiliser celle-ci, il est nécessaire de modifier le fichier de configuration du projet en y ajoutant les informations relatives à la librairie.

4.1.5 Les langages web

Le développement des navigateurs, des langages web, des possibilités que ceux-ci permettent et l'évolution de la société visant à s'orienter vers du cloud computing, nous confortent dans l'utilisation des dernières normes de ces langages. Afin de proposer une application web interactive, nous nous sommes orientés sur ces derniers langages web, même si nous n'utilisons pas toute l'étendue de ce qu'ils proposent.

4.1.5.1 HTML5

GWT nous permet l'utilisation du HTML5. L'arrivée du HTML5 permet d'accroître les performances d'une application web, en proposant des fonctionnalités plus avancées telles que l'utilisation de canevas, de balise audio ou vidéo.

En outre, comme évoqué précédemment, une autre fonctionnalité réside dans l'utilisation d'un local storage permettant de stocker une grande quantité de données.

4.1.5.2 CSS3

Le design général de l'application a été basé sur cette dernière norme. Il est donc nécessaire d'avoir un navigateur à jour pour pouvoir profiter de l'avantage qu'offre cette norme.

4.1.5.3 JS

Même si le code JS est généré par le compilateur de GWT, elle est un élément essentiel de l'application.

4.1.6 Local Storage

L'apparition du local storage (LS) avec le HTML5 offre de nouvelles possibilités aux développeurs d'application web. En effet, le local storage permet de stocker une grande quantité de données côté client. Pour bien comprendre le choix de l'utilisation de cette technologie, nous allons le comparer avec le cookie.

4.1.6.1 Local Storage vs Cookies

Tout comme les cookies, les informations contenues dans le local storage ne peuvent pas contenir d'informations sensibles [20]. Il est, cependant, nécessaire de souligner qu'aucun travail de cryptage n'a été fait sur ces données. En effet, nous supposons que les informations contenues dans notre local storage ne sont pas utiles pour les Hackers [21].

L'utilisation du local storage permet, comme les cookies, de partager les données entre les différents onglets du navigateur. Nous aurions pu nous orienter vers une solution plus simple, comme

19. Fournies sous la forme de .JAR.
20. Comme, par exemple, les numéros de carte de crédit.
21. En effet, il nous apparaît inutile de crypter le nom d'un cours.

enregistrer les informations de la partie cliente dans des tableaux ou listes, mais les informations ne seraient plus partagées entre les onglets et fenêtres du navigateur. Ce choix limiterait de manière drastique l'utilisation du programme.

	Local Storage	Cookie
Stockage :	5MB	80KB (4KB/cookie, 20 cookies/domaine)
Bande passante :	Aucune	Envoi de données à chaque requête
Performance :	Data mise en cache du navigateur	Data envoyé répétitivement vers le domaine
Contraintes :	aucune	300 cookies maximum
Rapidité :	très rapide	plus lente

TABLE V – Comparaison Local Storage et Cookie

Le local storage présente donc des avantages. La table V résume les différentes caractéristiques différant entre l'utilisation d'un local storage et l'utilisation de cookies.

De plus, la mise en place d'un local storage, bien qu'étant une partie fastidieuse, permet de faciliter le développement ultérieur de notre solution.

4.1.6.2 Principe

Le principe de ce local storage est donc de fournir un moyen de stocker et d'accéder rapidement aux données tout en limitant le trafic entre le client et le serveur. Nous ne rentrerons pas dans la structure de celui-ci, ce point faisant partie d'un autre chapitre. Les informations sont mises en cache et peuvent être supprimées à tout moment par l'utilisateur.

4.1.6.3 Utilisation dans notre application

Lors de chaque modifications (comme le placement d'un carton), nous enregistrons ces changements dans le local storage ainsi que dans la base de données de manière asynchrone afin de permettre une grande rapidité de l'application.

Dans l'hypothèse où la base de données ne serait plus accessible (ex : perte de connexion) nous avons implémenté une fonctionnalité [22] permettant de synchroniser l'état du local storage avec la base de données lors du retour de connexion.

Cependant, actuellement, une connexion internet est requise lors d'un rechargement de la page afin de se mettre à jour, ceci est indispensable pour offrir une synchronisation cohérente. Cependant une autre fonctionnalité [23] permet, si l'option est activée ou si la base de données n'est pas accessible, de fonctionner sans cette dernière, et lors de la récupération de la connexion, les données sont alors synchronisées.

4.2 Développement du côté serveur

4.2.1 Java EE

La plate-forme Java Entreprise Edition est un ensemble de spécifications portées par un consortium de sociétés internationales qui offrent, ensemble, des solutions pour le développement, le

22. En version alpha au moment de l'écriture de ces lignes.
23. En version alpha au moment de l'écriture de ces lignes.

déploiement et la gestion des applications d'entreprises multiniveaux, basées sur des composants objets.

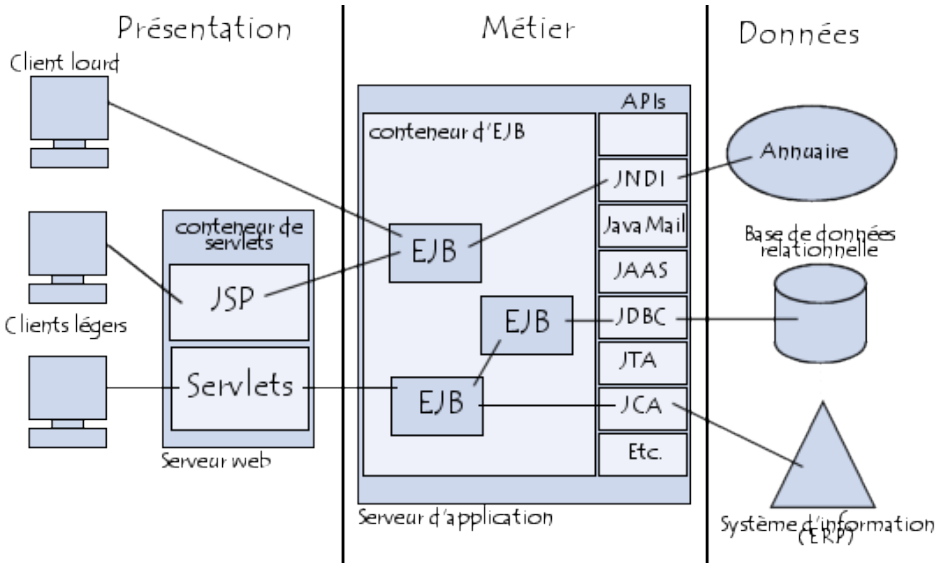

FIGURE V – Diagramme issu de`architecture_JEE.png`

debug : Expliquer l'image
Et dans la figure V[24] ** image** Construit sur la plateforme de Java 2 édition standard (Java SE), la plateforme Java EE ajoute certaines fonctionnalités nécessaires pour fournir une plateforme complète, stable, sécurisée et rapide de Java au niveau entreprise. Dans la mesure où J2EE s'appuie entièrement sur le Java, il bénéficie des avantages et inconvénients de ce langage, en particulier une bonne portabilité et une maintenabilité du code.

L'ensemble de l'infrastructure d'exécution Java EE est donc constitué de services (API) et spécifications tels que :
- HTTP et HTTPS
- Java Transaction API (JTA)
- Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP)
- Java Interface Definition Language (Java IDL)
- Java DataBase Connectivity (JDBC)
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- API JavaMail et JAF (JavaBeans Activation Framework)
- Java API for XML Processing (JAXP)
- Java EE Connector Architecture
- Gestionnaires de ressources
- Entreprise Java Beans (EJB)
- Java Server Pages (JSP)
- Servlet
- Java API for XML Web Services (JAX-WS, anciennement JAX-RPC)

24. Diagramme issu de

- SOAP with Attachments API for Java (SAAJ)
- Java API for XML Registries (JAXR)

Dans le cadre de notre application, seule une sous-partie de ces composants a été utilisée, tels que les servelets, les JSP, JavaMail ou encore JDBC avec Hibernate.

L'architecture J2EE repose sur des composants distincts, interchangeables et distribués, ce qui signifie notamment :

- qu'il est simple d'étendre l'architecture
- qu'un système reposant sur J2EE peut posséder des mécanismes de haute-disponibilité afin de garantir une bonne qualité de service
- que la maintenabilité des applications est facilitée

Ces outils favorisent l'interaction avec, d'une part, le solveur, et d'autre part, la base de données. Cela constitue un avantage majeur de notre application. Java EE regroupe donc nativement ces libraires dans un endroit unique. Ainsi, il n'est plus utile de devoir faire face à des implémentations complexes.

De plus, la cohérence du programme favorise la maintenance de celui-ci. Et ce choix favorise le debuggage de l'application.

Bien qu'ayant déjà utilisé et programmé en Java SE, nous ne nous étions jamais retrouvés confronté au Java Entreprise Edition. La différence entre ceux-ci se trouve surtout dans la complexité de leur utilisation.

Le java SE ayant une approche plus orientée desktop. A contrario, le Java EE propose des outils avancés permettant de faire des applications internet (notamment web, mail, etc.) nécessitant un temps d'apprentissage et d'adaptation plus long. Utiliser du Java EE nous a permis d'en apprendre plus sur ce langage fort demandé en entreprise. En effet, il nous a paru important de connaître ce langage car il constitue un atout majeur pour un développeur.

Pour le log des données côté serveur, nous avons également utilisé log4j qui nous à permis de faciliter le travail de debuggage l'application.

4.2.2 Hibernate

Pour pouvoir communiquer avec notre base de données à partir de notre code java, nous avons, dans un premier temps, utilisé JDBC. Ensuite, dans un paradigme objet, nous avons choisi d'utiliser Hibernate comme une couche supérieure à JDBC. Hibernate permettant de pouvoir transformer les tables en objets, celui-ci offre une bonne cohérence avec l'application.

Hibernate est un framework libre, appellé framework de « mapping objet-relationnel » ou encore de « persistance objet des données » (voir Figure VI).

Cela permet donc à la couche applicative de notre programme de traiter les données venant de la base de données comme des objets, fournissant un mapping entre le relationel et l'orienté objet.

La base de données peut être traitée avec les avantages de l'orienté objet (comme le polymorphisme, l'héritage, etc.). Le lien entre les classes et la source physique des données est défini au sein d'un fichier xml. Ainsi, il s'agit effectivement d'un mapping objet-relationnel.

Cela nous à donc permit de gérer notre base de données dans un paradigme orienté objet comme le reste de l'application.

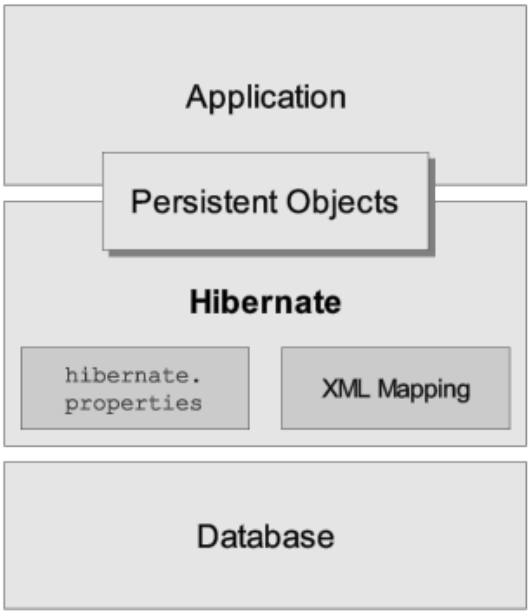

FIGURE VI – Diagramme issu de

Un autre avantage recherché par cette solution réside dans son indépendance par rapport à la base de données qui le rend entièrement portable, et cela sans aucune modification du code. Ainsi, comme évoqué précédemment, notre application peut tourner sur 221 base de données différentes. En effet, notre application, après avoir configuré ses crédentials dans le fichier de configuration de Hibernate, se chargera de créer l'entièreté des tables et la structure de la base de données.

Théoriquement il aurait donc été possible d'envoyer directement un objet "hibernate" (représentant par exemple une table de notre base de données) à GWT donc à l'utilisateur côté client. Cependant, cet aspect reste théorique comme stipulé dans la documentation de GWT [25].

En effet, une SerializationException est levée à chaque fois qu'un type transféré via RPC n'est pas sérialisable. La définition de la sérialisibilité signifie ici que le mécanisme RPC - GWT sait comment sérialiser et désérialiser le type de byte code au format JSON et vice-versa.

Le problème vient du fait que Hibernate modifie les objets afin de les rendre persistants [26]. Au moment du transfert de l'objet, une serialisation est tentée, mais l'objet n'étant pas le même (car modifié par javassist), il ne peut être sérialisé par RPC-GWTP.

Pour répondre à cette difficulté majeure, il a été nécessaire d'ajouter des classes de type DTO [27], qui, étant sérialisables, ont pu servir de communication entre la partie cliente et serveur.

À noter que Hibernate propose son propre langage de requête HQL [28], syntaxiquement proche du SQL, qui lie vision objet et vision relationnelle [29].

25. `http://developers.google.com/web-toolkit/articles/using_gwt_with_hibernate`
26. Pour être exact, c'est la librairie Javassist qui se charge de réécrire le byte code de ces objets pour les rendre persistants.
27. Data Transfer Object
28. Hibernate Query Language
29. `https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html`

Chapitre 5

Méthodologie de conception

Un programme informatique, comme n'importe quel projet, nécessite une bonne gestion pour pouvoir assurer son bon développement. D'une part, il faut faire face aux difficultés liées au travail d'équipe, d'autre part, il faut gérer le projet en tant que tel.

5.1 Git pour le développement collaboratif

La création d'un logiciel en équipe implique la gestion de cette équipe ainsi que le partage et l'échange des informations entre chaque membre de celle-ci.

L'équipe doit travailler de façon coordonnée. En effet, elle doit s'échanger l'état d'avancement du travail ainsi que les résultats obtenus. Dans cette visée, nous nous sommes tournés vers le programme de gestion de versions Git développé par Torvald et, notamment, utilisé dans le cadre du développement du noyau Linux.

Git est un logiciel de gestion de version décentralisé. Il permet de travailler à plusieurs sur un même projet. Il gère lui-même l'évolution du contenu en fusionnant les changements sans perte d'information. En outre, il garde en mémoire toutes les versions du code. De plus, il est libre et adapté aux grands projets.

Afin de permettre un stockage de notre code, nous avons de nous tourner vers un dépôt GitHub [1].

5.2 Logiciel de suivi de problème

En plus de reposer sur Git, GitHub offre un logiciel de suivi de problème, aussi appelé bugtracker. Un logiciel de suivi de problème est en quelque sorte un journal des problèmes classés par type. C'est un outil particulièrement utile pour le développement en équipe. Il donne un aperçu clair des bugs et de leurs états de résolution.

1. GitHub est présenté précédemment.

5.3 Méthode de travail

Pour ce travail, il a été important d'avoir une méthode rigoureuse. Nous avons donc scindé le travail en plusieurs parties, sous forme de blocs à développer.

Pour chacun de ces blocs, nous avons noté les problèmes de conceptions et, ensuite, nous avons analysé comment les résoudre. Par exemple, la difficulté d'implémentation du système d'échanges asynchrones ou, encore, l'absence de support, par GWT, du filtre de cartons ont dû être analysées et ont été l'objet de solution respective.

Nous avons suivi la méthode RUP [2] basée sur UP, définissant un moyen de travailler sur des applications de type orienté objet. Cette dernière fournit de nombreux avantages comme le rôle de chaque acteur du projet, le cycle de vie de l'application, etc. Nous nous sommes donc basés sur cette méthode pour élaborer notre travail dans de bonnes conditions.
Nos besoins étant de trouver une méthode de travail en équipe et d'avoir une bonne gestion de projet, nous ne rentrerons pas dans les détails de cette méthode. Toutefois, nous pouvons dire qu'elle nous à été bénéfique pour l'élaboration de notre application. Certains aspects de cette méthode comme la gestion de projets ont pu être utilisés via GitHub et son bugtracker.

Nous avons conceptualisé notre application dans une optique d'extension. Dans cet objectif, rien n'a été fait statiquement afin de garantir l'évolution future de ce projet. C'est pour ces raisons que certains choix on été effectués, nous rendant la charge de travail plus lourde mais possédant des avantages non négligeables pour la bonne évolution de l'application. L'étendue et le potentiel d'un tel outil-logiciel sont grands, surtout pour un établissement scolaire. Par conséquent, nous nous sommes donc confortés dans cette optique.

5.4 Communication

Une bonne communication interne a dû être assurée afin de pouvoir prendre des décisions sur certaines parties du projet. Bien qu'ayant défini la part de travail de chacun, il est important de bien comprendre ce qui a été fait par l'autre. Il faut aussi pouvoir tirer avantage d'un travail à deux, surtout dans le cadre de réflexion concernant la méthode à utiliser. Par exemple, les choix devant être établis en priorité pour élaborer certaines parties de l'application ou pour régler les bugs, etc.

Nous avons utilisé des logiciels de VOIP et nous nous sommes réunis régulièrement afin de discuter des différents points. Ainsi, il était plus aisé de parler des problèmes rencontrés et de trouver, ensemble, une solution à ces problèmes. Certaines tâches ont donc été effectuées séparément et d'autres conjointement.

2. Rational Unified Process

5.5 Apports périphériques

Nous avons, dans le cadre de nos cours à l'EPHEC, eu l'occasion d'utiliser certaines méthodes permettant de bien clarifier un programme telle que la méthode QQOQCP [3]. Nous avons donc tiré parti de cet apprentissage pour introduire la problématique de notre travail.

Ainsi, en reposant sur les différentes méthodologies existantes, nous avons beaucoup appris sur les problèmes pouvant intervenir dans l'élaboration d'une application (que ce soit web ou desktop). Il est bien entendu impératif de se rendre compte de ces différents points pour pouvoir avoir une cohérence au sein d'une équipe. Il est fréquent pour un développeur de devoir travailler en équipe dans une entreprise. Ainsi, cette approche nous donne l'opportunité d'être mieux préparés au monde professionnel.

3. Quoi ? Qui ? Où ? Quand ? Comment ? Pourquoi ?

Chapitre 6

Structure du code

6.1 Le model View Presenter

MVP est un design pattern [1] afin d'avoir un code clair et structurer d'une application. Le MVP favorise le travail d'équipe et permet aux différents acteurs prenant part au développement de l'application de pouvoir travailler simultanément. Nous allons découper ces différentes parties.

6.1.1 Le modèle

Le modèle englobe les données de l'application. Dans notre cas, les informations relatives aux attributions des cours ainsi que la liste des locaux dans lesquelles devront être placées nos attributions.

6.1.2 La vue

La vue correspond au graphisme de l'application. Pour bien illustrer ce concept, nous pouvons reprendre nos cartons. Ceux-ci contiennent des informations sur les attributions et se voient attribuer, lorsqu'ils sont placés dans un horaire, un local. La vue n'a aucune connaissance de ces informations. En effet, notre carton est uniquement un widget contenant des labels qui sont positionnés d'une certaine façon, qui font un taille de 80 pixels, etc.

6.1.3 Le présenteur

Le présenteur représente la logique de l'application. Nous reprendrons l'exemple du carton pour illustrer cette partie. Le modèle étant les données brutes, la vue étant le moyen de les afficher, il faut pouvoir définir à un endroit, à quel moment les données doivent être mise a vue de l'utilisateur. C'est dans cette partie qu'intervient le présenteur, en mettant dans chaque carton les informations relatives aux attributions.

6.1.4 Le controleur de l'application

GWT introduit un contrôleur d'application. Dans une application GWT, certaines logique qui ne sont pas assignées à un présenteur en particulier. Ce concept est propre à GWT et ne fait pas

1. patron de conception, définissant une façon de conceptualiser un projet

réellement partie de l'architecture du design pattern MVP.

6.2 Les packages java

Nous allons dans cette section présenter la manière don le programme a été conceptualisé. Pour ce faire, nous allons tout d'abord commencer par analyser les différents packages java ainsi que les classes que ceux-ci comportent. Comme nous l'avons vu dans le point précédent, l'application est basée sur MVP. Nous pourrons distinguer ce modèle au travers des différentes classes contenues dans nos packages :

- **be.betty.gwtp**
  S'y trouve le fichier de configuration du projet. Ce fichier comporte l'ensemble des informations utiles au bon fonctionnement du projet.

- **be.betty.gwtp.client**
  Nous y trouvons toutes les classes utilitaires de la partie cliente. Certaines de ces classe ont des méthodes statiques, celles-ci étant nos données qui ne changerons pas.

- **be.betty.gwtp.client.action**
  Toutes les actions envoyées depuis la partie cliente se trouve ici. Elle seront ensuite récupérées dans les packages server du projet.

- **be.betty.gwtp.client.event**
  Les différentes événements envoyés au travers de l'application. Comme déjà expliqué précédement, ces événements sont envoyés sur un busevent, celui-ci peut être écouté afin de récupérer les informations.

- **be.betty.gwtp.client.gin**
  Ce package plus particulier regroupe les informations relatives à GinJector. Celui-ci possède notamment une classe de configuration permettant a Gin de connaître la structure du projet afin d'assurer le bon déroulement des différentes injections effectuées.

- **be.betty.gwtp.client.place**
  Sont contenus les informations relatives aux différentes "page" du projet. Lorsque l'application est lancée, à chaque page est attribuée un token (que nous pouvons retrouver dans l'url), par exemple la page de login correspond à l'url : `http://www.nomdusite.com#login`. Le package placé permet la création de ces liens a travers l'application.

- **be.betty.gwtp.client.presenter**
  A chaque vue est associée un présenteur, les présenteurs de chacune de ces vues est contenu dans ce package.

- **be.betty.gwtp.client.views**
  Ici se trouvent les informations concernant le graphisme général de l'application. Nous distinguons deux types de fichier. Des fichiers classiques Java ainsi que des fichiers XML. Les fichiers XML étant construits à l'aide de GWT-Designer, même si ceux-ci on subit beaucoup

de modifications manuelles.

- **be.betty.gwtp.client.views.ourWidgets**
  Correspond à nos widget. Nous y retrouvons nos cartons ainsi que la modification d'un widget déjà existant offert par GWT. Nous avons dû modifier nous même ce widget pour pallier a certaines limitations de celui-ci.

- **be.betty.gwtp.server**
  Toutes les informations utilisées et les méthodes statiques du coté serveur sont mis dans ce package (comme le package client). S'y retrouvent aussi les différents Handler des actions.

- **be.betty.gwtp.server.bdd**
  La base de données créée par Hibernate se retrouve dans ce package. C'est d'ailleurs, via les classes de ce package, que la base de données est automatiquement créée.

- **be.betty.gwtp.server.guice**
  Équivalent de Jin coté serveur.

- **be.betty.gwtp.server.solver**
  Les différentes classes utilisées par notre solveur. Celles-ci étant basée sur les classes de *Unitime*.

- **be.betty.gwtp.server.shared**
  Contient les méthodes et constantes pouvant être appelées par la partie cliente et serveur de l'application.

  be.betty.gwtp.server.shared.dto
  Les classes de data transfert object, étant des POJO [2] permettant de transférer les données entre le client et le seveur, permettent la bonne communication entre ces deux parties via RPC.

2. Plain Old Java Object

Chapitre 7

La base de données

Avant de rentrer dans les détails de la base de données, il est à noter que la vision portée sur celle-ci doit être considérée selon un point de vue qui n'est pas classique. En effet, l'utilisation l'Hibernate nous a permis de gérer la base de données de manière orientée objet et non plus d'une manière relationnelle. Ainsi, la base de données n'est donc plus vue en tant que tel mais plutôt comme un ensemble de ces objets. Hibernate crée aussi des tables de mapping et dispose de sa propre façon de gérer le relationnel.

La base de données se veut, délibérément, simple et minimale. Le but étant, avant tout, de manipuler les données liées aux attributions et aux autres données requises par la construction d'un horaire (nom des classes, desiderata, etc.). Nous avons donc évité de manipuler les informations superflues en important celles-ci statiquement dans la base de données. Notre application n'ayant pas pour objectif de modifier ces données.

Par exemple, nous ne nous soucions pas de données propres à l'année, la section, l'implantation, etc. Nous nous focalisons uniquement sur les attributions. Les informations qui les composent doivent être séparées en différentes parties (les professeurs, les différents groupes, etc.). Certaines données ne sont pas normalisées, ceci n'étant pas requis.

Nous pouvons noter un objet plus particulier (correspondant donc à une table de la base de données), l'objet ActivityState. Celui-ci contient l'état d'un carton à un instant donné, c'est-à-dire si ce carton est placé, à quel endroit et dans quel ordre le placement a été effectué. Cette manière de faire nous permet d'offrir un historique complet de ce qui a été fait en permettant des retours en arrière ou d'offrir une vision statique de la construction de l'horaire.

Nous privilégions de cette manière, la rapidité et la facilité d'ajout. En effet, cette manière ne nécessite pas de mettre le carton à jour et prévient également les différentes erreurs pouvant survenir comme une écriture simultanée par différents utilisateurs au sein d'un même projet, d'une perte de connexion, etc. Ceci étant des perspectives d'avenir du programme et n'étant pour le moment pas prises en compte.

Afin de stocker et manipuler les contraintes nécessaires à notre application, nous avons décidé de gérer cela d'une manière assez simple et intuitive. Le principe de base est de représenter une contrainte/desiderata par un triplet <source-niveau-destination>.

La source pouvant être des professeurs, locaux, cours, cartons, groupe. Le niveau est simplement un niveau de préférence qui permettra principalement de savoir si cette contrainte est du type "hard" ou "soft". La destination est quant à elle soit une période de la semaine soit un local.

De cette manière, une grande partie des contraintes décrites au chapitre 1 peut être représentée.

Pour ce qui est des contraintes portant sur un ensemble (ex : si on veut que tous les locaux du premier étage soient disponibles le lundi matin, il suffira de mettre une contrainte sur chacun de ses locaux du style <L42 - 3 - Lundi première heure>. En pratique, on a représenté cela par un Objet(table) Préférence sur les différentes sources possibles.

Un autre type de contrainte pouvant être stocké (bien qu'elle n'est pas encore réellement prise en compte) est la notion de dépendance entre deux cartons. Pour des cours qui doivent par exemple être le plus éloignés ou le plus rapprochés possible.

Mais, nous ne permettons actuellement pas [1] les contraintes dépendantes entre elles ou conditionnelles (e.g. si le prof X ne peut pas avoir son jour de congé le lundi, il aimerait avoir ses matinées de libre,etc.).

Pour finir, nous dirons que la base de données n'est pas la base de données complète ; celle-ci est très évolutive, étant données les possibilités et facilités offertes par Hibernate.

1. Et de toute façon, le solveur ne peut pas les gérer dans son implémentation actuelle.

Chapitre 8

Solveur

Pour pouvoir apporter un réel avantage à notre application, comparé à la méthode actuelle de création d'horaire (i.e. manuel), il était nécessaire de lui rajouter une certaine "intelligence". Une partie de cette "intelligence" est prise en charge du coté client, qui comme décrit dans une précédente section, se charge d'assister l'utilisateur lors de la création de son horaire.

Cependant, ce support est minimaliste et une recherche plus poussée est effectué du coté serveur à l'aide d'outil de programmation par contraintes et de "recherche opérationnelle". Ce solveur sera ensuite amené à nourrir le solveur minimaliste du coté client et proposer un certain pilotage de ce solveur au travers de l'application cliente.

Pour arriver à ces résultats, nous nous sommes tournés vers des outils de programmation par contraintes et de recherche opérationnelle. Pour comprendre notre choix de librairie ainsi que son utilisation, il est important de revenir sur les principe fondamentaux de la programmation par contraintes.

8.1 Rappels théorique

La programmation par contrainte un paradigme de programmation [1] ayant pour but la résolution de problèmes combinatoire en descrivant plutot le but recherché que la méthode utilisé, c'est pourquoi la programmation par contrainte es une forme de programmation déclarative. « Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming : the user states the problem, the computer solves it. » — E. Freuder–citation–

dans le cas qui nous concerne, la problème de création d'horraire, est un cas typique de CSP (Constraint Satisfaction Problem), et comme tout csp peut se définir comme suit : *** definition formelle *** Dans notre cas, les variables sont, se qu'on va appeller des "activity", et correspondent à un cours ou à tous type d'activitées pouvant avoir lieu à l'Ephec (il s'agit, dans la méthode traditionnel des cartons physique qui seront manipulé lors de la création d'un horraire.)

Le domaine de valeurs que peuvent prendre ses variables, ses activity, sont un local et une periode de la semaine. Ca fait un domaine à deux dimentions (Lieu X Periode). Dans l'idéal, il faudrait prendre le choix des professeurs comme troisième dimension, mais ce n'est pas le cas dans notre impémentation actuelle.

Les contraintes deviennent alors les relation mathématiques reliant les activity. On a tenté de faire des listes exaustives de ses contraintes, mais seule une sous partie est actuelement pris en

1. paradigme

compte :

8.1.1 Choix de librairies

Le solveur (celui du coté serveur), est indépendant de tout, absolument tout. Il est écrit en java, et repose sur une librérire de csp. Il réside donc du coté serveur, dans le servelet de l'application, c'est donc le meme pour tout utilisateurs se connectant au site. Il ressevra l'information et l'enregistrera xxx

Apprès bpc de recherches, cette librairie est de loins la plus adapté et à jours (la dernière version datant du 20 juin, est d'ailleur celle utilisé par le programme). Ca permetra un dévellopenment du solveur, independament de la bdd ou de l'interface. Les possibilités de cette librairie combiné à nos choix d'infrastrcuture sont énorme, et à notre grande tristèsse, ne sont pas utilisé au maximum de sa puissance.. seule la surface à pu être implémenté, faute de temps.

Definition 1 (*CSP*). A constraint satisfaction problem (*CSP*) is a triple Θ = (V,D,C), where
- V = $\{v_1, v_2, ..., v_n\}$ is a finite set of variables,
- D = $\{Dv_1, Dv_2, ..., Dv_n\}$ is a set of domaines (i.e., Dv_1 is a set of possibles ...
- C = $\{c_1, c_2, ..., c_m\}$ is a finite set of constraints restricting the values that the variables can simultaneously take.

Definition 2 (*assignment*). Let Θ be a CSP, an assignment of the variables from V is $\eta \subseteq \{v/a | v \in \text{V} \ \& \ \text{a} \in \text{Dv}\}$ where $\forall\ v/a,\ w/b \in \eta\ v = \text{w} \Rightarrow a = b$.
An element v/a of $\in$ means that variable v has assigned value a.
An assignement is complete if $|\eta| = |\text{V}|$ (i.e., all variables are assigned).

Definition 3 (*solution to CSP*). A solution to the constraint satisfaction problem Θ is a complete assignement σ of the variables from V that satisfies all the constraints.

Chapitre 9

Discussions

9.1 Choix concernant la performance

La question de performance pour un logiciel est un point très important à prendre en compte. En effet, il faut que le programme puisse fournir une bonne fluidité et que celui-ci soit agréable à utiliser. Pour ce faire, nous avons opté pour une solution permettant d'offrir ces performances. L'architecture de l'application, les fonctionnalités utilisées ainsi que la simplicité des différentes pages de l'application, permet de rendre l'application facile d'utilisation, et d'être performante pour le travail qui lui est demandé de faire.

9.2 Choix concernant la sécurité

Du fait de nos choix de conception, tel que GWT pour générer le JS, Hibernate pour abstraire la base de données ou encore, le choix des RPC pour la communication client-serveur, nous pensons avoir créé une application « de base » très sécurisée, et qu'il serait tout à fait envisageable de la faire tourner sur un serveur externe. La prévention contre des attaques de type XSS [1], CSRF [2], le cryptage du mot de passe en SHA256, ainsi qu'un captcha très basique afin d'éviter les bots favorisent cette sécurité pour le client.

La sécurité du coté serveur à également été prise en compte. L'application tourne sur une Debian stable (Squeeze) ayant très peu de service installé. Tomcat, notre conteneur d'applet, est lui aussi à jour et en version stable. Il ne bénéficie d'aucun droit ROOT, ainsi donc, (contrairement à une autre application tournant sur Windows par exemple), il n'a pas le droit d'émettre sur le port 80. Pour garder l'application safe et performante, une redirection de son port d'origine est effectué par Netfilter [3] sur le port 80.

Nous n'avons pas poussé plus loin la sécurité du coté serveur, étant donnée que celle-ci est destinée à fonctionner sur d'autre machine, mais en situation réel, il serait judicieux de mettre en place un lien HTTPS.

N'étant pas infaillible, la mise à jour (très aisée pour notre type d'application), sera une part importante dans la sécurité du système. Cela étant également accentué par la mise de notre code source sous une licence ouverte de type GPL3.

1. Cross-site Scripting
2. Cross-site Request Forgery
3. framework implémentant un pare-feu dans le noyaux Linux

Chapitre 10

Perspectives

10.1 Possibilités d'amélioration du programme

10.1.1 Mode comparaison

C'était un de nos objectifs principaux ; il était question de pouvoir créer à la voilée les colonnes représentant plusieurs professeurs, locaux ou classe, tout en gardant, pour les lignes, les périodes. Malheureusement le manque de temps à eu raison de nous, ainsi qu'un problème fondamentale, à savoir qu'il n'es pas possible, de rendre une région ne faisant pas partie du DOM (typiquement rajouté en js, donc d'une manière non-statique) comme étant une région "dropable". Cela n'étant certainement pas impossible, la solution idéal nous est pas encore connue ; probablement qui faudrait rendre l'entièreté de la région contenant le tableau comme "dropable" afin de subdifiser celle ci.. car actuelement ce soit les "cases" qui sont dropable..

10.1.2 Solveur

Ce fut également un point assez frustrant, étant donné nos recheches sur le sujet, et surtout la trouvaille de cette librairie "miraculeuse" écrit par Muller, et qui, en plus d'être en gpl et fortement maintenue (dernière version date d'il y a 2 semaines), elle nous paraît très performante et d'une rare adéquation. C'est donc avec beaucoup de tristesse dans l'âme (:p), qu'on n'en utilise qu'une fractions de ses possibilité. C'était tout l'avantage de notre application (et de sont architectur client-server) et ce n'est pas utilisé a son full potentiel. C'est donc la première amérioration que devra subir notre application, car tout est en place pour pouvoir en tirer un avantage certain.

10.1.3 Droits

Un des but de cette architecture était de pouvoir avoir plusieurs compte lié à un projet, avec des droits différents, imaginons des professeurs qui pouraient éditer leur désidérata, ou certaine parsonne qui pourait suivre l'évoltion de la construction de l'horaire sans pour autant avoir les droits de modification.

10.1.4 Mode hors ligne

On se base entièrement sur le local storage, les requètes vers le serveur ne servent qu'a se connecter, charger le projet et enregistrer des modifications. Actuelement, il est possible de conti-

nuer la création d'un horraire chargé en mémoire sans la necessité d'une connections, cependant, pour pouvoir pleinement utiliser cette fonctionnalité, il est necessaire de rajouter certaine xxx, tel qu'une resyncronisation des modifications faite en local avec la bdd lors que la connection est retrouvée ainsi que la possibilité de reprendre un projet en cache.

10.1.5 Upload à partir d'une base de données

le fichier actuellement utilisé pour l'upload des attributions est le résultat d'une requête SQL. Dans un soucis de clareter et de facilité pour l'utilisateur, il est plus évident de devoir rentrer un fichier manuellement étant donné qu'un autre fichier des locaux doit être également fournis. Pour qu'un upload puisse se faire de manière complète, il faudrait que la liste des locaux avec ce que ceux ci comporte soit également enregister dans une base de données afin de ne pas perdre l'utilisateur sur deux méthodes différentes d'envoi et de pourvoir ainsi garder une certaine cohérence du logiciel.

10.1.6 Modifications de données

actuellement il n'est pas possible de modifier toutes les données intervenant dans l'application (notamment la création d'un prof, local, etc.) Il faudrait donc avoir la possibilité d'ajouter certaines informations directement via l'interface du logiciel, afin de prévenir de tout oubli, ou rajout de dernière minute sur le nombre de classes, etc.

10.1.7 Upload de contraintes

L'idéal serait de permettre à l'utilisateur d'uploader sont fichier de contraintes (sous forme de fichier XML par exemple), ou de pouvoir les entrée manuellement via l'interface. N'étant pas l'objectif de base de notre travail de fin d'étude, celui-ci étant plus orienté sur la programmation par contraintes à proprement parlé et l'intégration de cette programmation dans un outils d'aide à la création d'horaire, et donc l'élaboration de ce logiciel, ceci limite l'utilisateur de l'application aux uniques configuration que nous aurions choisi au préalable. Il faudrait donc dans l'idéal permettre à l'utilisateur de pouvoir manipuler lui même les contraintes.

Conclusion

Dans le présent rapport, nous avons vu comment répondre au problème réel et récurrent de la création d'horaires dans le contexte académique. Pour répondre à cette problématique, nous avons dû effectuer un large travail sur trois quatre aspects en particuliers ; la programmation par contraintes, une solution client/serveur, et, enfin, un travail collaboratif en équipe.

Dans le cadre de la programmation par contraintes, un travail a été fait sur la théorie sous-jacente aux contraintes et au paradigme de programmation correspondant. Cet aspect a été travaillé tant dans la littérature sur le sujet que lors de notre rencontre avec Monsieur Schaus, docteur dans le sujet. Cette rencontre a notamment souligné l'importance du choix des bons outils. En effet, il existe de nombreuses librairies dédiées à la programmation par contraintes, cependant peu sont adaptées à notre problème et permettent de prendre en compte la pondération des contraintes. En effet, comme évoqué précédemment, un *desiderata* exprimé par un professeur n'est pas vraiment une contrainte dure et doit être l'objet d'un traitement à l'aide de règles heuristiques et métaheuristiques.

Notre travail sur la programmation par contraintes nous a permis de théorisé plus facilement le problème posé par la création d'horaires à l'Ephec. Rappelons que ce problème a été analysé notamment grâce à notre entretien avec Madame Gillet, directrice de l'établissement sur le site de Louvain-la-Neuve. En définitive, nous avons utilisé les libraires offertes par *UniTime* afin d'implémenter un solveur permettant de résoudre les problèmes posés par la création d'horaire.

Cependant, afin de rendre notre solution utile au plus grand nombre, il a été choisi de prendre une architecture client/serveur. Cette architecture permet notamment de minimiser les problèmes classiques côté client, de centraliser les données sur un seul serveur et d'utiliser des technologies fiables. En outre, en utilisant GWT, nous avons permis de scinder l'exécution du code entre le serveur et le client. En effet, comme expliqué précédemment, GWT permet de transposer du Java en JavaScript afin de faire exécuter le code par le navigateur côté client.

Enfin, nous avons privilégié les derniers standards, tels que HTML5 et CSS3, afin d'assurer une compatibilité ascendante dans le développement futur de l'application. Ainsi, cet aspect client/-serveur permet d'élaborer une interface Web afin de permettre à l'utilisateur de se concentrer sur l'aspect important de sa tâche, à savoir la création d'horaire.

Nous avons réalisé cette solution en équipe. Cet aspect a nécessité d'utiliser certaines méthodes et certains outils. Ces derniers sont surtout git et les systèmes de téléphonie VIP. Ce travail en équipe nous a permis d'apprendre à travailler en équipe et de mettre au point des normes de codage.

Plus personnellement, ce travail nous a particulièrement tenu à cœur. En effet, beaucoup de travail a été fourni pour utiliser des technologies modernes tels que Hibernate, Java EE que nous

pensons être des atouts dans le monde professionnel. Toutefois, il est nécessaire de souligner que nous avons été surpris par la longueur d'apprentissage et d'implémentation de ces technologies. Ces difficultés ont dû être analysées et réglées, notamment au niveau problématique de la gestion asynchrone des échanges client et serveur.

En effet, bien que la solution que nous proposons soit encore perfectible en de nombreux points, notamment au niveau des résolutions de contraintes complexes qui étaient l'objectif premier de ce travail, il est nécessaire de souligner que la méthodologie ayant conduit à son élaboration et les choix technologiques pris permettent, malgré la grande quantité de code, une maintenabilité et une robustesse relatives.

De plus, la méthodologie suivie, qui consistait à découper les parties du projets en blocs, aurait permis de doubler l'équipe sans que les différents développeurs aient à se marcher sur les pieds. De plus, les moyens mis en place pour développer la solution de façon collaborative auraient permis la gestion d'une équipe plus grande.

En conclusion, nous avons proposés une méthodologie ainsi qu'une solution pour l'assistance à la création d'horaire. Cet aspect rejoint le domaine plus large de la recherche opérationnelle dans laquelle s'inscrit ce présent projet.

Bibliographie

Müller, T. (2005). *Constraint-based Timetabling*. Thèse de doctorat, Charles University in Prague, Faculté of Mathematics and Physics.

Schärlig, A. (1985). *Décider sur plusieurs critères : panorama de l'aide à la décision multicritère*, volume 1. PPUR.

Annexes

A Formats de sérialisation RDF

B Exemple de code

```
 from constraint import *
 problem = Problem()
 problem.addVariable("a", [1,2,3])
 problem.addVariable("b", [4,5,6])
 problem.getSolutions()
[{'a': 3, 'b': 6}, {'a': 3, 'b': 5}, {'a': 3, 'b': 4},
 {'a': 2, 'b': 6}, {'a': 2, 'b': 5}, {'a': 2, 'b': 4},
 {'a': 1, 'b': 6}, {'a': 1, 'b': 5}, {'a': 1, 'b': 4}]

 problem.addConstraint(lambda a, b: a*2 == b,
                       ("a", "b"))
 problem.getSolutions()
[{'a': 3, 'b': 6}, {'a': 2, 'b': 4}]

 problem = Problem()
 problem.addVariables(["a", "b"], [1, 2, 3])
 problem.addConstraint(AllDifferentConstraint())
 problem.getSolutions()
[{'a': 3, 'b': 2}, {'a': 3, 'b': 1}, {'a': 2, 'b': 3},
 {'a': 2, 'b': 1}, {'a': 1, 'b': 2}, {'a': 1, 'b': 3}]
```

FIGURE VII – Code python