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 mémoire, 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.

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 *desideratas* particuliers, les élèves ont des parcours différents, etc. Cette somme de critères contraignants font 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 aérien 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 météos, 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 *desideratas* 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

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.

établissements, doivent se voir attribuer des horaires spécifiques. De plus, certains cours sont dispensés 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 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 du 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 apportés 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 le base de donnée. 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 d'utiliser des fonctions de rappel, dit « callback », 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 décrits 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 poussées 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 présentés 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 suivant :

- 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. C'est-à-dire, les différents cours avec leur propriétés intrinsèques. Dans le cadre de ce rapport, nous désignerons ces cours par l'appellation de cartons. Cette notion sera explicitée dans le chapitre suivant.

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 méta-heuristiques. 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 énumérées 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
Desideratas
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 congés
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 prisent 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 posent 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 rependu et beaucoup utilisé dans les écoles, `aSc` Horaire possédant une partie automatisée et manuel 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, quant 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 somme 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 contrainte 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 – Script SQL pour la création de la table *band*

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 importante 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 chapitre lui étant concernée.

3. L'implémentation actuelles des solveurs proposés 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 rattachées 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.

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 effectuées du coté 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 tel 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 fournis 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.

2.3 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

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

– 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 suivit des différentes actions faite par l'utilisateur.

Debug : Capture d'écran

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 manuelle 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.

Debug : Capture d'écran

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-même 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 limité pour le moment. **Debug : Capture d'écran**

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 c'est effectué correctement, et une couleur rouge pour signaler un problème. Ce systèmes est limité et pourra être amélioré dans le futur.

Debug : Capture d'écran

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 la dite 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 peut permettre 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,...) 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 drivers 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 et offre la possibilité d'utiliser des variables statiques typées. 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 conforté 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.

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

Troisièmement, GWT propose également l'utilisation d'outils tels que Jin and Guice qui feront l'objet d'un autre point. En effet, comme sont nom l'indique, GWT est une vrai boite à outils qui permettent 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 conduit 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. Qui, rappelons-le, est intrinsèquement lié à un paradigme objet.
13. Par exemple, les desideratas 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)

4.1.1.1 Présentation

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 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" mais possède sa propre philosophie.

Pour programmer une application web de notre type, il faut maîtriser des outils comme l'AJAX[1], l'HTML ainsi que le CSS. Le problème principal de ces outils est la compatibilité des navigateurs. La façon de mettre en forme un site web n'aura pas toujours le même rendu sur les différents navigateur. Il faut prendre en compte aussi la difficulté d'utilisation de ces outils de manière avancée (utilisation du DOM en HTML,etc.). Le langage JS est assez complexe d'utilisation, surtout pour l'écriture de grosses applications, le debuggage de celles-ci étant assez fastidieux puisqu'il s'agit d'un langage interprété. Pour pallier à ses problèmes, GWT à 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 vrai boite à outils, et propose des solutions de développement répondant aux besoins du programmeur.

4.1.1.2 Principe de fonctionnement

GWT possède un plugin Eclipse (et pour d'autre IDE[2] comme NetBeans, JDeveloper, etc.), Eclipse étant l'IDE que nous avons choisi. 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 paquets java.

Comme nous l'avons expliqué précédemment, le principe de fonctionnement de GWT est de pouvoir créer des applications web basé en langage Java, celles-ci étant basées sur le modèle

1. Asynchronous JavaScript And XML
2. Integrated Development Environment

client/serveur, et de convertir la partie cliente de l'application en JS, la partie serveur reste en java. L'application peut être compilé pour un ou plusieurs navigateurs. favorisant la compatibilité et l'homogénéité du logiciel sur chacun d'entre eux. Ces deniers chargent uniquement ce qui les concernes. En plus de ces différents aspects qu'offre GWT, il permet aussi de préciser quelles sources du code [3] prendre en compte pour un navigateur spécifique.

4.1.1.3 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 en byte code. Ceci afin de permettre le debuggage de l'application. GWT compile tout de même le projet en JS html et 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 créer le JS, HTML et CSS a partir de sources du projet. Ce code correspond à l'application final qui pourra être déployer sur notre serveur Tomcat.

4.1.1.4 Architecture GWT

L'architecture d'un projet GWT se fait sous la forme de client/serveur. 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 [4]

Évènement

Afin de pouvoir la communication entre les différentes vues de l'application, GWT utilise un système d'envoi d'évènements (appelé Events). Ceux-ci permetent aux vues de dialoguer entre elle. Par exemple, une application GWT peu posséder une en-tête. Celle-ci est statique et n'est pas recharger 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 enregistré auprès de l'eventbus. Cet eventbus peut être écouté pour récupéré les informations voulues.

3. correspondant au .CLASS généré lors de l'exécution du code java
4. Remote Procedure Call

Actions

Les actions ressemble au Event, à l'exception que celles-ci sont envoyées au serveur. Elle permette 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 vérifié dans la base de données qui va renvoyer, dans le cas ou l'utilisateur existe, la liste des projets qui lui sont assignés. Cette méthode ce fait à l'aide du protocole JSON[5]/RPC. Les appels se font de manière asynchrone permettant de ne pas bloquer le client lors d'un appel de procédure.

4.1.1.5 Avantages

Facilité d'utilisation

Le premier avantage que nous citerons est la facilité d'installation et d'utilisation de l'outil. Pas besoin de configuration fastidieuse. Installer l'IDE, Installer le Plugin, et ça fonctionne. La possibilité de pouvoir coder sont application à l'aide d'un langage de haut niveau facilite le développement.

Debbugage

Il permet un debuggage rapide du code, celui-ci étant codé en java et non pas en JS qui est un langage interprété.

Optimisation

Optimisation du code, obfuscation de celui-ci, compression du JS, mise en cache, séparation du JS en différent fichier, etc.

prédéfini de composant

Différent widget prédéfini sont disponible, ainsi, pas besoin de codage fastidieux dans le designer d'un bouton, une boite de dialogue, etc. avec la possibilité de créer ses propres widget.

Code adapter en fonction du navigateur

Le code java est traduit en code JS et automatiquement adapté au navigateur de notre choix.

JSNI (JavaScript Native Interface)

GWT 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 codée en JS comme Jquery.

Utilisation de GinJector

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

5. JavaScript Object Notation

4.1.1.6 Inconvénients

Le principal inconvénient que nous avons pu noter suite à notre expérience personnelle est lorsque l'application atteint un état d'avancement plus avancé, il devient très lourd et très lent de tester son application en mode développement. La JVM [6] devant traduire le code est très lente. Il nous a fallu en moyenne 5 minutes pour charger une page, et nous avons eu frequement des erreurs de type "out of memory" du à la lourdeur de la tache effectuer par l'ordinateur. Pour certains types de test, nous nous somme retrouver dans l'obligation de déployer l'application sur notre serveur tomcat, car la traduction du java vers le JS étant très lourde, certaines partie devait être tester directement en mode production (notamment le drag and drop,...). De même, lorsque l'application doit charger une grande quantité d'information (grand nombre de cartons, professeurs,...) la page mais beaucoup de temps à s'ouvrir.
Autre inconvénient, le temps de compilation du logiciel. Celui-ci peut être très fastidieux en fonction des paramètres demandé.

Un autre inconvénient est la limitation des widgets fournis de base dans l'application. Pour certaines chose plus avancée nous avons du avoir recourt à des librairie externe, bien que celle-ci ne soit pas aussi performante (comboBox avec CheckBox,...).

4.1.2 GWT-Designer

GWT-Designer est un outils permettant de créer de manière simple les interfaces graphiques. Celle-ci est crée via un fichier XML [7] et est retranscrite en code java par le compilateur. Ce code XML est éditable "manuellement" ou peut être créer à l'aide d'une interface de drag and drop ou les composants (widgets) peuvent être sélectionné. L'avantage d'utiliser un tel outil est la bonne pratique que celui-ci apporte, permettant de faire une distinction entre les différentes partie du code.

4.1.3 GWT - Plateform

4.1.3.1 Présentation

GWT plateform est un framework basé sur le MVP [8] et permettant de mieux structurer notre projet en respectant une certaine méthode. Ce framework s'incorpore directement à l'IDE et est dépendant de GWT.

4.1.3.2 Le model View Presenter

Le MVP se base sur le MVC [9]. Celui-ci est un design pattern fournissant une manière d'élaborer des interfaces graphique. Il est séparé en trois partie, le modèle de données, les différentes vues de l'application (L'interface du client) et le présenteur (correspondant au controleur du MVC). La partie intéressante de ces modèles réside dans l'utilisation d'un controleur/présenteur, ou transite les informations. L'application est soumise à leur contrôle ce qui permet d'avoir une application plus solide. Dans le MVC le contrôleur s'occupe de gérer les évènements. La logique de contenu (Rendering logic) se passe directement entre le modèle et la vue. En MVP, cette logique est gérée

6. Java Virtual Machin
7. eXtensible Markup Language
8. Model View Presenter
9. Model View Controler

par le présenteur. Plus rien ne transite directement entre l'interface et le modèle mais est soumise au contrôle du présenteur.

4.1.4 Libraire externe

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 ou les smartphones et tablettes on une place prépondérante sur le marché. Cette librairie permet de rendre dragable n'importe quel widget ou un ensemble de ceux-ci.

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

4.1.4.2 Smart-GWT

Smart-GWT est un wrapper [10] de la librairie JS SmartClient. Elle propose un grand nombres de widget venant s'ajouter à ceux fournis par GWT. Puisque Smart-GWT n'est qu'un wrapper de la 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 proposé par cette librairie ne sont pas aussi réactif que ceux de GWT.

Ces deux libraires ont été spécialement conçue pour être utilisée avec GWT. Il ne s'agit pas de librairie JS comme Jquery, mais bien de librairie [11] 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 au librairie.

4.1.5 Les langages web

L'évolution des navigateurs, des langages web et des possibilités que ceux-ci permettent, l'évolution de la société visant à s'orienter de plus en plus vers le cloud computing, nous montre l'utilité d'utiliser ces langages dans leurs dernières normes. Afin de proposer une application web interactive, nous nous sommes orienté sur ces derniers langages web, même si nous n'utilisons pas tout l'étendu de ce qu'ils proposent.

4.1.5.1 HTML5

GWT nous permet l'utilisation du HTML5. l'arrivée du HTML5 permet d'accroitre les performances d'une application web, en proposant des fonctionnalités plus avancées telles que l'utilisation

10. Un adaptateur
11. Fournies sous la forme de .JAR

de canvas, de balise audio ou video. Une autre fonctionnalité est 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 (LS)

L'apparition du local storage avec le HTML5 ouvre une porte en plus au développeur d'application web. Ce dernier permet de stocker une grande quantité de données coté 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

	Local Storage	Cookie
Stockage :	5MB	80KB (4KB/cookie, 20cookies/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 – Légende

Tout comme les cookies, les informations contenu dans le local storage ne peuvent pas contenir d'informations sensibles [12]. Nous ne nous sommes pas attardé sur ce point car les informations contenu dans notre local storage ne sont pas utile pour les Hackers, inutile donc de crypter le nom d'un professeur.

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 enregistrer les informations de la partie cliente dans des tableaux ou listes, mais les informations ne seraient plus partager entre les onglets et fenêtres du navigateur, ce qui limiterait de manière drastique l'utilisation du programme. Le local storage à donc bon nombres d'avantages et l'utilisation et la mise en place de celui-ci, bien qu'étant une partie fastidieuse, était pour nous nécessaire au bon développement de l'application ainsi qu'à son développement futur.

12. Comme les numéros de carte de crédit

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 mise 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 ou la base de données ne serait plus accessible (ex : perte de connexion) nous avons implémenté une fonctionnalité [13] permettant de synchroniser l'état du local storage avec la base de données lors d'un retour de connexion.

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 fonctionnalité, en Alpha actuellement, permet, si l'option est activé ou si la bdd n'est pas accessible, de fonctionner sans cette dernière, et lors de la récupération de la connexion, les données sont alors synchronisé.

4.2 Développement du côté serveur

4.2.1 Java EE

La plateforme Java entreprise édition est un ensemble de spécifications portées par un consortium de sociétés internationales qui offrent, ensemble, des solutions pour le développement, le déploiement et la gestion des applications d'entreprises multi-niveaux, basées sur des composants.

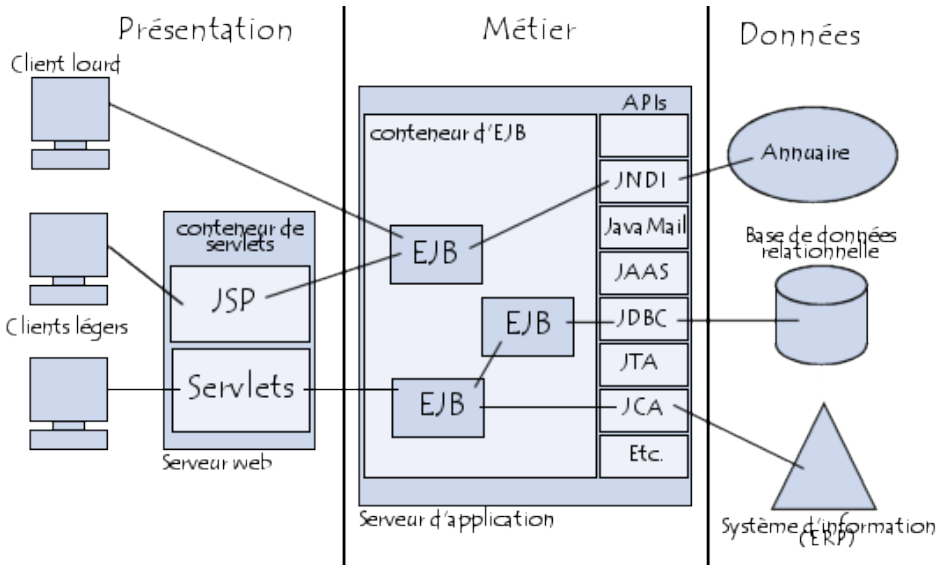

FIGURE II – Diagramme issu de`architecture_JEE.png`

13. version alpha

Et dans la figure II [14] ** 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'execution JavaEE est donc constitué de services (API) et spécifications tel 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)
- 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 on été utilisé, tel que les servelets, les JSP, JavaMail ou encore JDBC avec Hibernate, tel que décrit dans le point suivant.

De plus, 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. Il n'est plus utile de chercher un framework adapté à nos besoin et de devoir faire face à des implémentations, parfois fastidieuse, de ce dit framework.

Bien qu'ayant déjà utilisé et programmé en Java SE, nous ne nous étions jamais retrouver confronté au Java sous sa forme Entreprise Edition. La différence entre ceux - ci se trouve surtout dans la complexité de leur utilisation. Le java SE ayant une approche plus simpliste et plus orienté desktop, et le Java EE étant beaucoup plus puissant, proposant des outils avancés qui peuvent faire l'objet d'une longue étude avant de pouvoir être implémenté. Utilisé cette forme de Java nous a permis d'en apprendre plus sur ce langage fort demandé en entreprise. Connaitre ce langage constitue un atout majeur pour un développeur, nous sommes ravis d'avoir pu utiliser une

14. Diagramme issu de

partie des possibilités offert par ce langage.

Pour le log des données coté serveur, nous avons utilisé log4j. Cette librairie ne fait pas partie intégrante du Java EE mais nous l'avons utilisé conjointement afin de pouvoir logger ce qu'il se passe coté serveur.

4.2.2 Hibernate

Pour pouvoir communiquer avec notre base de données à partir de notre code java, nous avons tout d'abord utilisé JDBC. Cet outils n'était pas auto-sufisant c'est pour ces raisons que nous nous sommes orienté ver Hibernate.

Hibernate est un framework libre, appellé framework de « mapping objet-relationnel » ou encore de « persistance objet des données » (voir Figure III). Cela permet donc à la couche applicative

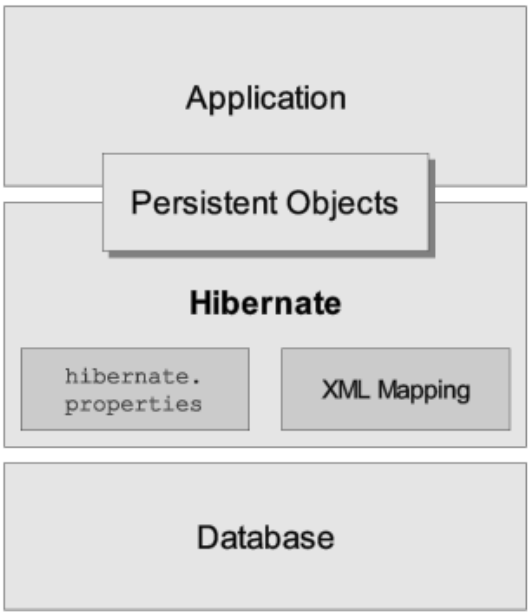

FIGURE III – Diagramme issu de

de notre programme de traiter les données vennant de la base de donnée comme des objects donc le contenu reste en mémoire même après la fin d'exécution du programme. D'où persistance objet des données. Le lien entre les classes exposées et la source physique des données (dans notre cas une base de données *relationnelle*) est définie par un fichier xml. D'où mapping objet-relationnel. Cela nous à permit de gérer notre base de données, comme le reste de l'application en orienté objet, et même si son apprentissage ne fut pas des plus aisée, la gestion générale du code s'en est retrouvé grandement amélioré.

Un autre avantage recherché par cette solution, est que son indépendance à la base de donnée le rend ultra portable, et, sans aucune modification du code, notre application peu tourner sur 221 [15] base de données différente. Notre application, après avoir configuré ses crédential dans le fichier de configuration d'Hibernate, se chargera de créer l'entièretée des tables et la structure de la base de donnée.

15. http ://developers.sun.com/product/jdbc/drivers

Théoriquement il aurait donc été possible d'envoyer directement un objet "hibernate" (représentant par exemple une table de notre bdd), à gwt, donc à l'utilisateur, malheureusement le monde n'est pas parfait, et comme stipulé dans la doc GWT [16], une SerializationException est levée à chaque fois un type transférée sur 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 bytecode au format JSON et vice-versa. Le problème vient du fait qu'Hibernate modifie les objects afin de les rendre persistant (pour être exact, c'est la librairie javassist qui se charge de réécrire le bytecode de ces objects pour les rendre persistant). Au moment du transfer de l'object, une serialisation est tantée, mais n'étant pas le même (car modifé par javassist), il ne peut être sérialisé par RPC-GWTP. Il a fallut donc rajouter des classes de type DTO [17], qui, eu sérialisable ont pu servir de communiquation entre la partie cliente et serveur.

Hibernate possedant sont propre language de requète HQL [18], d'apparence similaire au SQL, mais pleinement orienté Object et comprenant des notions comme l'inheritance ou le polymorphisme [19]. Mais ca nous à bien fais chier (xx), et on a pas su en tirer pleinement avantage.

16. https ://developers.google.com/web-toolkit/articles/using_gwt_with_hibernate
17. Data Transfer Object
18. Hibernate Query Language
19. 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 un bon déroulement de ce projet. 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 et partage des données

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, elle doit s'échanger l'état d'avancement du travail ainsi que les résultats obtenus. Pour ce faire, il faut de bons moyens de communication, ou dans notre cas, UN bon moyen de communication : GIT**?**.

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. Il garde également en mémoire toutes les versions du code. Il est libre, gratuit et particulièrement facile à utiliser. Le code se trouve sur un dépôt internet, nous avons choisi GitHub**?** pour ce dépôt.

5.2 Logiciel de suivi de problème

En plus de cette gestion de versions, GitHub offre un logiciel de suivi de problème, c'est-à-dire, un 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.

5.3 Méthodologie de conception

Pour ce travail, il a été important d'avoir une méthode de travail rigoureuse. Nous avons donc scindé le travail en plusieurs parties, sous forme de bloques à effectuer. Pour chacun de ces bloques, nous avons noté les problèmes de conceptions, si il y en a eu, que nous avons pu rencontrer. Nous avons analyser comment chaque bloque pouvait etre conceptualiser, nous pouvons par exemple

citer le filtre de cartons qui ne pouvais être conçu avec les outils de base proposé par GWT. Nous avons donc soigneusement analysé chaque partie avant de nous lancer tête baissé dans la conception d'un bloque pour ensuite se retrouver bloquer par les possibilités qui nous était offerte.

Nous avons suivit la méthode RUP [1] basé sur UP, définissant un moyen de travailler sur des applications de type orienté objet. elle fournis de nombreux avantages comme le rôle de chaque acteur du projet, le cycle de vie de l'application, etc. Nous nous sommes donc basé sur cette méthode pour élaborer notre travail dans de bonne condition.

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, mais nous pouvons dire qu'elle nous à été bénéfique pour l'élaboration de notre application. Certain aspect de cette méthode comme la gestion de projets on pu directement être utiliser via GitHub et sont bugtracker. Cet outils nous a été d'une grande aide.

Nous avons conceptualiser notre application dans une optique d'extension. Dans cette 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 effectué, nous rendant la charge de travail plus lourde mais possédant des avantages non négligeable pour la bonne évolution de l'application. L'étendu et le potentiel d'un tel outil-logiciel est énorme, surtout pour un établissement scolaire, nous nous sommes donc conforté dans cette optique.

1. Rational Unified Process

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 différentes acteur prenant par au développement de l'application de pouvoir travailler simultanément. Nous allons découper ces différentes partie.

6.1.1 Le modèle

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

6.1.2 La vue

La vue correspond au graphisme de l'application. Pour bien illustrer ce concept, nous pouvons reprendre nos cartons. Ceux contiennent des informations sur les attributions et ce vois attribuer, lorsqu'il est placé dans un horaire, un local. La vue n'a aucune connaissance de ces informations. Notre cartons est uniquement un widget contenant des labels, que ceux ci sont positionner d'une certaines façon, qu'il font un taille de 80 pixels, etc.

6.1.3 Le présenteur

Le présenteur représente la logique de l'application. Nous pouvons toujours prendre notre carton pour illustrer cette partie. Le modèle étant les données brute, 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. Ces dans cette partie qu'intervient le présenteur, en mettant dans chaque carton les informations relative au attribution.

6.1.4 Le controleur de l'application

GWT introduit un controleur d'application. Dans une application GWT, certaines logique qui ne sont pas assigné à un présenteur en particulier. Ce concept est propre à GWT et ne fait pas réellement parti de l'architecture du design pattern MVP.

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

6.2 Les packages java

nous allons dans cette section, décortiquer 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 celle ci comporte. Comme nous l'avons vu dans le point précédent, l'application est basé sur MVP. Nous pourrons distingué ce modèle au travers des différentes classe contenu dans nos packages.

Chapitre 7

La base de données

la base de données se veut, délibérément, extrêmement simple/minimal. Le but n'est pas de gérer l'information mais plutôt de la stocker.

Par exemple, on ne se soucie pas de donnée propre à l'année, la section, l'implantation, etc., Nous nous focalisons uniquement sur les Attributions. Les informations qui les composent se doivent d'être séparé en différente partie (par exemple, les professeurs, les différents groupes, etc.

L'objectif n'étant pas de faire une base de données relationnelle dans les règles de l'art (genre éviter les répétitions dans les tables, y a un mot pour ça mais je parviens pas à le retrouver, il est 5h du matin i, mais d'avoir les données rassemblées en un point fixe (comme les classes (Année, Section, groupe) (même si il y a des répétitions des données (genre section T)) et de pouvoir accéder aux données de la manière la plus rapide qu'il soit. Nous avons donc opté pour ce système de stockage, évitant les cout imputer au passage entre les différentes tables pour récupérer l'ensemble des informations dont nous avions besoins, et permettant de garder une logique sur ce que doit représenter une attribution.

une autre table très particulière est les activityState, qui correspond à l'états d'un carton (idéalement à un instant donné), cad, si ce carton est placé, où et qd. Le but est de pouvoir offrir des retour en arrière ou une annalyse des mouvements effectué, comme par exemple en selectionnant un carton et voir les états précédent de se dit cartons (mais cela n'est pas fait dans la version actuel du programme) ainsi que de priviligier l'ajout à lecture (car il faudrait faire une recherche et un update à chaque pose de cartons), et également permetre de régler des litiges pouvant survenir (utilisation synchrone du programme par exemple). Il rend également la gestion des différentes "Instances" bcp plus simples. L'inconvénient, c'est qu'il faut faire une recherche un peu plus longue lors du chargement du projet car il faut selectionner le nernier état du carton. Egalement, il est possible que cette table pourrait atteindre de grosse tailles, et il est necessaire, une fois le projet cloturé, de supprimer les données inutiles.

Les "instances" sont un set d'états, xxx

pour la gestions des contraintes, il est également necessaire de les sauvegarder, et seulement un set non exostif de contraintes est possibles d'etre enregistré (et traité dans notre programme). Néanmoins, ce set est relativement large et permet de "nommer" la plus part des contraintes. Il s'agit principalement de "préférences". Préférences qui pourront être "hard" ou "soft" (un indiquateur est possible pour y mettre plus de grénulosité, mais il n'est actuelement pas pris en compte par notre solveur). préférence applicable sur un Moment (jour/period) ou sur un(des) local(aux). préférences dont la sources peuvent être prof, cours, groupe, local.

Avec ca, il est possible de

Chapitre 8

Solveur

i

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

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

Pour ariver à ses résultats on s'est tourné vers des outils de programmation par contrainte 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.

Chapitre 9

Discussions

9.1 Choix concernant la performance

9.2 Choix concernant la sécurité

Du fait de nos choix de conception, tel que gwt pour générer le java-script, ou hibernate pour abstraire la base de donnée, ou le choix des rpc pour la commnuniquation client-serveur,.. Nous pensons avoir créé une application “de base”, très sécure, et qu'il serait tout à fait envisageable de faire tourner notre application sur un serveur externe. (xxx) xss, csrf, sha256, check (basique) de la complexité du mot de passe, ainsi qu'un catcha très basique aussi pour eviter des bots.

La secu du coté serveur à également été pris en compte, mais, logiquement, en moins poussé (quoi que). A savoir : l'application tourne sur une debian *stable* (à savoir Squeeze), n'ayant que peu de services installé. Tomcat, notre conteneur d'applet, est lui aussi à jours, et en version stable, il ne bénificie d'aucun droit root, ainsi donc, (contrairement à une autre application tournant sur Windows par exemple), il n'a pas le droit d'emmetre sur le port 80. Pour garder l'application safe et peformante, une redirection de son port d'origine est effectué par netfiller sur le port 80.

Nous avons pas poussé plus loins la sécu du coté serveur, étant donnée qu'elle est destiné à fonctionner sur d'autre machine, mais en situation réel, il serait jusdicieu de mettre en place un liens https. En situation réel, nous préconisons xxx (parfeu, strapping, vm,..)

N'étant pas infaillble, la mise à jour (très aisée pour notre type d'application), est nous semple -t-il, une part importante dans la sécurité du système. (cela probablement accentué par la mise de notre code source sous une license ouverte (gpl3))

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 collonnes représentant plusieur professeurs, loacux ou classe, tout en gardant, pour les lignes, les périodes. Malheureusement le manque de temps à eu raison de nous, ains qu'un problème fondamentale, à savoire qu'il n'es pas possible, de rendre une région ne fesant 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 fu également un point assez frustrant, étant donné nos recheches abondante sur le sujet, et surtout la trouvaille de cette librairire "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 parrait très performante et d'une rare adéquoicité. C'est donc avec bcp de tristesse dans l'ame (: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 continuer 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

actuelement il n'est pas possible de modifier toutes les données intervenant dans l'application (notament 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

Pour que cette partie puisse être éfectuée, il faudra d'abord s'assurer d'avoir un solveur très résistant. 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

– on conclut dans le foin :) –

-> point cool perso : utilisre des nouvelles technologies

-> ce projet nous tien à coeur
non, il n'est pas encore completement utilisable dans le sans que nous voudrions. Il est possible de créer horraire, mais les avantage qu'offrent l'outils informatique ne sont eux pas fonctionnel. Notre solution, actuelement offre donc moins d'avantage que le faire en version papier. Nous croyons cependant bcp dans notre programme et pensons sincèrement, que mme si c'était pas les les plus simple, les décisions que nous avons prise peuvent permettre énormément de choses.

-> nous avons été supris par la dificulté (dans le sans qu'un petit scout est surpris par la nuit :p), et il à été très frustrant de devoir faire face à d'autre problèmes qu'on avait pas imaginé et qui nous ont dévié de nos objectif premier. Heureusement ce fut des problème interessant qui nous ont bcp appris, que ca soit sur la gestion du code, gestion de notre "équipe" (fair ce tfe à deux était une exelente expérience. ca nous à permit de fournir du code plus propre, car relu, des idées plus aboutie, ...Notre gestion, et la grosseur de la tâche ainsi que sa "segmentation" nous aurais permi d'éfectuer ce tfe avec 2fois plus de participant, sans qu'ont ne se "marche sur les pieds",..)

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.