Projet MIM2 - Rapport de projet

Laure Danthony et Vivian Brégier

Pour le 22 Décembre 2001

Réalisation d'un logiciel d'emploi du temps

Table des matières

Introduction

L'objectif de ce projet était de réaliser un logiciel graphique et simple d'emploi dont le but est d'aider Simone Durand lors de la réalisation semestrielle des emplois du temps du MIM. Ce logiciel fonctionne sous Windows 2000 et a été testé la semaine dernière par la principale intéressée. Les contraintes imposées pour l'emploi du temps sont celles effectivement rencontrées par Simone.

Dans ce document, nous présentons comment nous avons traité le problème, en particulier comment nous avons segmenté l'implémentation. En annexe de ce document, on fournit les signatures des fonctions OCAML utilisées pour le calcul.

1 Cahier des charges

Les contraintes retenues sont du type :

1. créneaux autorisés (durée : 2h). On pourra ainsi décider que le jeudi après-midi ne possède pas de créneau pour cause de sport.
2. disponibilités des professeurs. Tel professeur faisant tel cours peut n'être disponible que le mercredi, par exemple.
3. intersection de cours des élèves. Deux cours ne peuvent être parallélisés (c'est-à-dire peuvent avoir lieu en même temps) si un seul élève suit ces deux cours ou si le même prof réalise le cours, évidemment !

On a fait le choix de ne pas tenir compte des salles disponibles, ni de la possibilité de dédoubler certains TD pour cause de sur-effectif.

2 Découpage des tâches et communication entre modules

Ce projet est composé de deux parties bien distinctes :
- La partie Interface Graphique Utilisateur (IGU) qui se présente sous forme d'application Java "clic and play". Cette partie contient aussi la partie d'affichage "esthétique" d'une éventuelle solution.
- La partie de Calcul d'une solution, appelée par l'utilisateur en cliquant sur un bouton "Calculer", qui se présente sous la forme d'un exécutable Ocaml.

Ces deux parties étant en grandes parties indépendantes, nous avons choisi d'en traiter chacun une indépendamment, en minimisant les communications nécessaires entre les parties. Cependant, nous avons dû initialement nous mettre d'accord sur le format des fichiers de communication :

- **En entrée**, l'IGU fournit à l'exécutable OCaml un fichier des contraintes sous la forme :

```
%DEBUT%

%PROFS%
        Yves Robert:algo_cours,algo_par_cours;4,5,6,7,8,9,12,13,14,15;
        Natacha Portier:algo_td;1,2,3,4,5,6,7,8,9,10,11,12,13,14,15;
```

```
        Arnaud Legrand:algo_par_td;1,2,3,4,5,6,7,8,9,10,11,12,13,14,15;
        Florent de Dinechin:compil_td,archi_td;1,2,3,4,11,12,13,14,15;
...

%MODULES%
        algo:Yves Robert;
        archi_cours:Jean Duprat;
        decidabilite:Jacques Mazoyer;
        logique:Pierre Lescanne;
...

%ELEVES%
        Bernard Nicolas:algo_cours,algo_td,archi_cours,archi_td;
        Bienvenu Laurent:deci_cours,deci_td,logique_cours,logique_td;
        Bui Xuan Binh Minh:algo_cours,algo_td,prog_cours,prog_td;
        Capern Simon:prog_cours,prog_td,unix_cours,unix_td;
...
%FIN%
```

- **En sortie**, l'éventuelle solution trouvée par l'exécutable est envoyée sous forme d'un fichier du type :

```
automates_td:12
calc_formel_td:11
unix_td:14
prog_td:13
prog_cours:12
...
```

Remarque : *Les informations données ici sont redondantes. Le programme OCaml les récupère tout de même toutes dans les structures dont il dispose. En revanche, il ne calcule une solution qu'à l'aide des informations "profs" et "élèves". Les informations "modules" auraient pû être utilisées pour voir si les informations sont cohérentes.*

3 Implémentation du calcul de solution

3.1 Vocabulaire

On utilisera le vocabulaire suivant dans la suite et pour l'implémentation :

- Un **cours** est indifféremment un cours proprement dit ou un td, sa seule caractéristique est qu'il occupe un créneau de 2 heures.
- Un **créneau** est une plage horaire du type $(jour, debut)$. Ainsi $(lundi, 10)$ est un créneau. Pour une journée donnée, une plage horaire appartient à l'ensemble $\{8, 10, 14, 16\}$
- Une **liste de cours** d'élèves est une liste d'intitulés de **cours** : pour un élève donné, une telle liste est de longueur deux fois le nombre de modules choisis (on compte les cours et les td).

3.2 Choix des structures de données

Les structures de données suivantes ont été utilisées

1. Un `cours` est un type "record" contenant les informations : le nom du cours, le nom du professeur, les élèves suivant le cours, la liste des créneaux possibles, et un champ "souhait" qui pourra éventuellement servir plus tard pour affiner "à la main" ;
2. Une structure de données `eleves` contiend pour chaque élève son nom, la liste des modules pris et un champ "souhait" qui pourra éventuellement servir plus tard ;
3. Un graphe d'interdictions où deux sommets de type $(cours, horairelist)$ sont reliés ssi il existe un élève qui prend ces deux modules ou un prof qui réalise les deux cours. Ce graphe est calculé une fois pour toute dans la phase d'initialisation.
4. Une `situation` (en cours) contiend à tout moment de l'algorithme d'allocation :
   - les modules pour lesquelles ont a déjà attribué un créneau,
   - les modules qui restent à placer,
   - pour chaque créneau de la semaine, les modules qui peuvent encore être mis dans ce créneau,
   - les plages encore possibles pour chaque module.

   Les deux dernières informations sont redondantes mais on ne peut se permettre de renverser très souvent des tables, étant donné la complexité de l'algorithme. Cette structure est la strucure clef de l'algorithme, celle qui est constamment utilisée.

3.3 Remarques sur la stratégie d'allocation

L'ordre dans lesquels les cours devront être alloués est très important, on doit commencer par les contraintes des professeurs les plus exigeants pour scier au plus tôt "les branches mortes" de l'arbre de parcours des possibilités. Il faut bien remarquer qu'à chaque étape, il faut recalculer ces priorités de traitement car par exemple YR qui n'était pas très exigeant au début (plages possibles : tout sauf le lundi et le vendredi), devra être placé prioritairement dès que toutes les plages du mardi, mercredi et jeudi seront allouées !

3.4 L'algorithme de recherche d'une solution

Notre algorithme réalise les étapes suivantes :

- **Initialisation :** initialisation des structures, calcul du graphe d'interdiction, calcul des priorités initiales pour les cours, calcul des cours pouvant être parallélisés ...En particulier, on initialise la structure `situation` bien soigneusement.
- **Boucle** : On désire traiter une situation s. On choisit pour cela un cours M le plus restrictif possible et non encore alloué. Pour chacun des horaires C possibles, on va essayer d'attribuer cet horaire au cours, en faisant une copie de la situation,dans laquelle on change la liste des cours à attribuer, les cours de chaque créneau ...et essayer d'appeler récursivement le programme sur cette nouvelle situation. Si cet appel récursif n'aboutit pas (c'est-à-dire lance une exception `Echec`), on essaie d'attribuer un autre horaire. Sinon, on reçoit de cet appel récursif un ensemble Γde couples (cours,horaire) et on renvoie l'ensemble $\Gamma \cup \{(M, C)\}$. Si aucun horaire

n'est possible, on lève l'exception `Echec`. Cela signifie que la situation s n'admet pas de solution.

Effectivement, on s'aperçoit qu'une stratégie d'allocation “contrainte maximale” est nécessaire à l'aboutissement de l'algorithme en moins de trois heures. L'implémentation de cette simple heuristique a été suffisante pour ramener le temps de calcul à une simension raisonnable : par exemple le fichier mim1/mim2 de cette année a été “résolu” en moins de $5s$ sur un Céléron 366Mz. Nous n'avons donc pas implémenté de stratégie supplémentaire.

3.5 Découpage en modules

Notre code se divise en 5 sections :

- Le fichier `type_defs.ml` contient les définitions des structures ainsi que des fonctions (très utiles) d'impression de ces diverses structures.
- Le fichier `recup_data.ml` contient notamment la fonction `recup_donnees` qui permet de récupérer toutes les informations à partir d'un fichier généré par l'interface graphique Java.
- Le fichier `init_situation.ml` contient l'initialisation du graphe des dépendances et de tous les champs de la structure `situation`.
- Le fichier `calcule_solution` contient le calcul d'une solution à partir des données précédentes. Cette fonction nécessite en particulier de nombreuses manipulations de la structure situation, et l'implémentation d'une heuristique de chois d'un cours à placer.
- Le fichier `interface.ml` permet de réaliser l'exécutable proprement-dit. Il contient la fonction `calc` qui récupère les arguments de la ligne de commande, récupère les données du fichier donné, initialise les structures, calcule éventuellement une solution et l'imprime dans un fichier résultat dont le nom lui a été passé en paramètre. C'est ce fichier résultat qu'utilise l'interface-utilisateur pour renvoyer le résultat sous forme de tableau à l'utilisateur.

4 Implémentation de l'interface graphique

Passons maintenant à l'interface graphique car une ligne de commande, ce n'est pas très ergonomique ...L'implémentation de l'interface graphique a été réalisée dans le langage Java.

4.1 Grandes lignes de l'implémentation

Le but du programme est, via une interface graphique intuitive, de récupérer les données nécéssaires à l'élaboration de l'emploi du temps, puis les communiquer à l'exécutable Ocaml, et enfin d'afficher de manière graphique la solution renvoyée par cet exécutable.

- Le programme contient donc en interne une structure de données adaptée à la mémorisation des contraintes. C'est la classe `EmplTpsData` qui contient cette structure de données.

  Lorsque l'utilisateur modifie les données à un endroit, les modifications doivent être répercutées sur l'ensemble de l'interface graphique (par exemple si l'utilisateur rajoute un professeur dans la liste des professeurs, dans la partie module, à l'endroit où on choisit quel professeur donne le module, on doit voir le nouveau professeur apparaître dans la liste des choix) Pour cela, cette classe contient des références des parties de l'interface graphique qu'il faut modifier lorsque l'on rajoute ou enlève un eleve, module ou professeur, et des méthodes pour effectuer ces opérations tout en faisant les modifications nécessaires sur l'interface graphique. Ce sont ces méthodes, et uniquement elles, qui seront appelées pour toute modification de la structure de données. Ainsi, l'interface graphique sera toujours dans un état cohérent.

- De plus, le programme doit être capable de sauvegarder son état et de le charger. Pour cela, on a utilisé un générateur de lexer pour faciliter l'implémentation de la lecture dans le fichier sauvegarde.
- Le reste du programme est occupé par l'interface graphique elle-même. l'API Java a été utilisée au maximum pour réaliser ce programme.

  L'interface est divisée en 4 panneaux, 3 panneaux pour entrer des données plus un panneau pour afficher le résultat. Ces panneaux de données sont :

  élèves : liste des élèves, pour chaque élève liste des modules qu'il suit (figure 1)

  professeurs : liste des professeurs, pour chaque professeur choix des créneaux horaires où il est disponible. (figure 2)

  modules : liste des modules, pour chaque module choix du professeur qui fait ce module (figure 3)

  Un seul des trois panneaux est affiché à la fois pour plus de clarté. On navigue entre les trois panneaux par des onglets (en fait des boutons).
  Le panneau de résultats est affiché dans une boîte de dialogue lorsque le calcul a été lancé et que le programme caml a renvoyé le résultat (figure 4).

4.2 Quelques screenshots

La phase d'entrée des contraintes est visible sur les figures 1, 2 et 3

La phase de visualisation du résultat est visible sur la figure 4

5 Pour aller plus loin

Pendant la semaine qui nous reste, nous pensons (si nous avons le temps !) :

- Fignoler encore et toujours l'interface graphique, afin qu'il soit le plus ergonomique possible pour simone.
- Compiler le programme sous Windows 2000.
- Modifier un petit peu l'algorithme afin qu'il puisse éventuellement donner plusieurs solutions [1].
- Permettre la réalisation automatique d'un fichier log par le programme `calc` qui pourra éventuellement permettre d'analyser les étapes de calcul en l'éditant "à la main".
- Evaluer le nombre d'étapes de récursion dans un fichier raisonnable.
- ...

[1] indispensable !

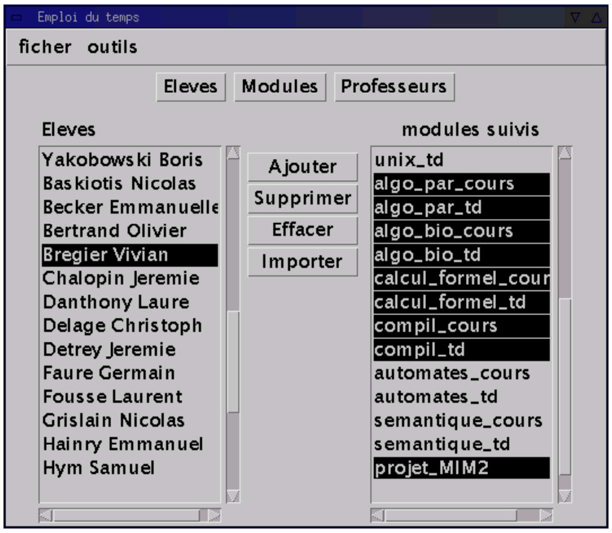

Fig. 1 – La partie Eleves

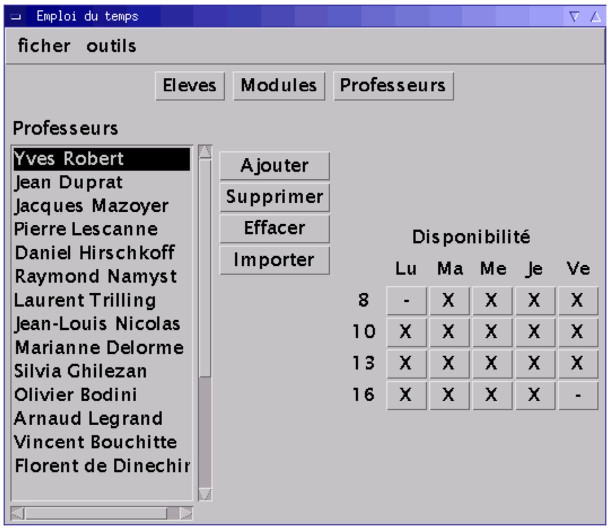

Fig. 2 – La partie Profs

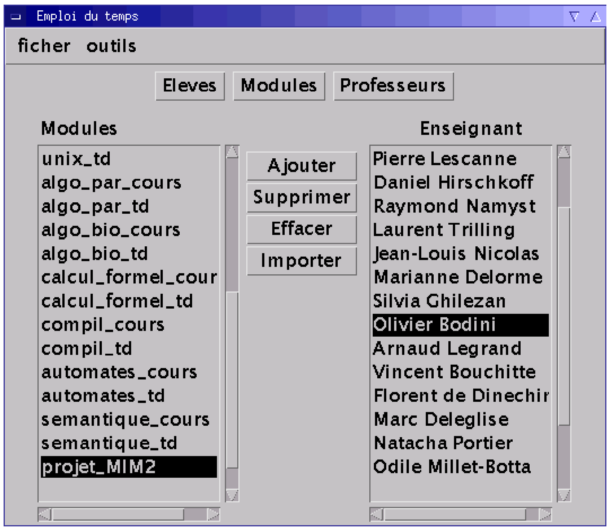

FIG. 3 – La partie Modules

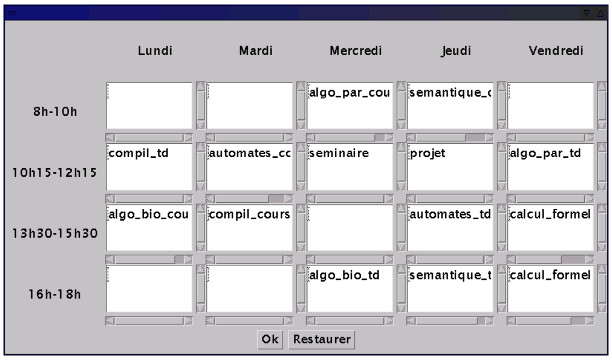

FIG. 4 – Le résultat sous forme d'un tableau !

A signature des fonctions OCAML

A.1 Fichier type_defs.ml

```
type horaire = int

type cours = {
  intitule : string;
  professeur : string;
  mutable participants : eleve list;
  mutable creneaux : horaire list;
  divers : string;
}

type eleve = {
  login : string;
  mutable modules : cours list;
  souhaits : string;
}

val affiche_eleve : eleve -> unit = <fun>
val affiche_cours : cours -> unit = <fun>

type situation = {
  mutable deja_places : (cours * horaire) list;
  mutable a_placer : cours list;
  mutable cours_poss_par_cren : (horaire * cours list) list;
  mutable cren_poss_par_cours : (cours * horaire list) list;
}

--pp de situations --
val imprime_deja_places : (cours * int) list -> unit = <fun>
val imprime_a_placer : cours list -> unit = <fun>
val imprime_cours_poss_par_cren : (int * cours list) list -> unit = <fun>
val imprime_cren_poss_par_cours : (cours * int list) list -> unit = <fun>
val imprime_situation : situation -> unit = <fun>
val affiche_dependances : (cours * cours list) list -> unit = <fun>
```

A.2 Fichier recup_data.ml

```
exception Fini;

val trouve_profs : in_channel -> unit = <fun>
val un_prof : in_channel -> string * string list * int list = <fun>
val donnees_profs : in_channel -> (string * string list * int list) list =
  <fun>
------- donnees_profs = liste de (login,modules,horaires) ------------

val trouve_modules : in_channel -> unit = <fun>
val un_module : in_channel -> string * string = <fun>
```

```
val donnees_modules : in_channel -> (string * string) list = <fun>
------- donnees_modules = liste de (nom,prof) ---------------------

val trouve_eleves : in_channel -> unit = <fun>
val un_eleve : in_channel -> string * string list = <fun>
val donnees_eleves : in_channel -> (string * string list) list = <fun>
------- donnees_eleves = liste de (login,modules) -----------------

val recup_donnees :
  string ->
  (string * string list * int list) list * (string * string) list *
  (string * string list) list = <fun>
-------- renvoie (donnees_profs,donnees_modules,donnees_eleves) ----
```

A.3 Fichier init_situation.ml

```
val creer_situation_vide : unit -> situation = <fun>
val creer_cours : string * string * horaire list -> cours = <fun>

val creer_list_cours_un_prof : string * string list * horaire list
                                     -> cours list = <fun>

val creer_list_cours :  (string * string list * horaire list) list
                                     -> cours list = <fun>

val couple_egal : string * string -> cours -> bool = <fun>
val est_dans_liste : cours list -> string * string -> bool = <fun>
val verifier_avec_liste_modules :
  cours list -> (string * string) list -> bool = <fun>

------------------ pour réaliser la liste des cours ----------
val creer_eleve : string -> eleve = <fun>
val est_ok : string -> cours -> bool = <fun>
val retrouve_module : cours list -> string -> cours = <fun>
val ajoute_module : cours list -> eleve -> string -> unit = <fun>
val ajoute_modules_pour_un : cours list -> string list -> eleve -> unit = <fun>
val creer_eleve_et_modifier_modules :
  cours list -> string * string list -> eleve = <fun>

val creer_tous_eleves_en_modifiant_modules :
  cours list -> (string * string list) list -> eleve list = <fun>

------------------ pour la liste cren_poss_par_cours------------
val init_cren_poss_par_cours :
  cours list -> (cours * horaire list)list = <fun>

------------------ pour la liste cours_poss_par_cren----------
val inserer : horaire ->
  cours -> (horaire * cours list) list -> (horaire * cours list) list = <fun>
```

```
val ajoute_cours : cours -> horaire list -> (horaire * cours list) list ->
                          (horaire * cours list) list = <fun>

val init_cours_poss_par_cren : cours list -> (horaire * cours list) list =  <fun>

------------------- pour le graphe -------------------------------
val init_graphe_vide : situation -> (cours * cours list) list = <fun>

val ajouter : 'a -> 'a list -> 'a list = <fun>

val ajouter_a_liste_associee :
  'a -> 'b -> ('a * 'b list) list -> ('a * 'b list) list = <fun>

val ajouter_crs_aux_modules :
  'a -> 'b list -> ('a * 'b list) list -> ('a * 'b list) list = <fun>

val ajouter_cours_communs_eleves :
  'a -> eleve list -> ('a * cours list) list -> ('a * cours list) list =
  <fun>

val ajouter_cours_communs_prof :
  cours -> string -> (cours * cours list) list -> (cours * cours list) list =
  <fun>

val traiter :
  cours -> (cours * cours list) list -> 'a -> (cours * cours list) list =
  <fun>

val augmenter_graphe :
  (cours * cours list) list -> cours list -> (cours * cours list) list =
  <fun>

val vrai_init_graphe : situation -> (cours * cours list) list = <fun>


-------------------FONCTIONS PRINCIPALES DU MODULE ---------------
val init_situation : bool -> string -> situation = <fun>
val init_graphe : situation -> (cours * cours list) list = <fun>
```

A.4 Fichier calcule_solution.ml

```
val enleve_dans_liste_cours : cours -> cours list -> cours list = <fun>
val nettoie_vide : (horaire * cours list) list -> (horaire * cours list) list = <fun>
val enleve_dans_couple : string -> horaire * cours list -> horaire * cours list = <fun>
val enleve_dans_liste_cren_cours : cours -> (horaire * cours list) list
                       -> (horaire * cours list) list = <fun>
val enleve_dans_liste_cours_cren : cours -> (cours * horaire list) list
                       -> (cours * horaire list) list = <fun>
val gros_enleve_dans_liste_cren_cours : cours list -> (horaire * cours list) list
```

```
                       -> (horaire * cours list) list = <fun>
val gros_enleve_dans_liste_cours_cren : cours list -> (cours * horaire list) list
                       -> (cours * horaire list) list = <fun>

val virer : 'a -> 'a list -> 'a list = <fun>
val virer_intrus : 'a list -> 'a list -> 'a list = <fun>

val enleve_dans_liste_crs_pp_crn :
  'a -> 'b list -> ('a * 'b list) list -> ('a * 'b list) list = <fun>

val dedans : 'a -> 'a list -> bool = <fun>

val purger_horaire :
  'a -> 'b -> 'b list -> ('b * 'a list) list -> ('b * 'a list) list = <fun>

val placer_cours : situation -> string -> horaire
             -> (cours * cours list) list -> unit = <fun>

val copie_situation : situation -> situation = <fun>

exception NeMarchePas
exception PlusDeCours

val selectionne_cours : situation -> 'a -> cours * horaire list = <fun>
val attribuer : situation -> cours -> horaire -> (cours * cours list) list
             -> unit = <fun>

--------------FONCTION PRINCIPALE DU MODULE et impression --------------------
val resolution :
      bool ->situation -> (cours * cours list) list -> (cours * horaire) list = <fun>

val imprime_res_dans_fichier : (cours * int) list -> out_channel -> unit = <fun>
```

A.5 Fichier interface.ml

```
 val calc : unit -> unit = <fun>
```

A.6 Makefile

On a la série de dépendances suivante :

$type\_defs.ml \rightarrow recup\_data.ml \rightarrow init\_situation.ml \rightarrow calcule\_solution.ml \rightarrow interface.ml \rightarrow calc$

D'où le makefile :

```
CAMLC = ocamlc
RM = -rm -f

.PHONY : all
all : calc
```

```
calc :
$(CAMLC) type_defs.ml;
$(CAMLC) type_defs.ml recup_data.ml;
$(CAMLC) type_defs.ml recup_data.ml init_situation.ml;
$(CAMLC) type_defs.ml recup_data.ml init_situation.ml\
calcule_solution.ml;
$(CAMLC) type_defs.ml recup_data.ml init_situation.ml\
calcule_solution.ml interface.ml -o calc;
$(RM) a.out;

.PHONY : clean
clean :
$(RM) *.cmi *.cmo *.dvi *~ calc
```

B Les principales classes JAVA

B.1 structure de données

L'ensemble des données mémorisées dans le programme java est maintenu par la classe `EmpTpsData`.

Elle contient toutes les méthodes nécéssaires pour maninpuler ces données tout en gardant des données cohérentes entre elles. (par exemple quand on supprime un module tous les élèves qui y étaient inscrits sont désinscrits) :

```java
public class EmpTpsData {
   static int listSize=15;
   ProfDataList profsList;
   ModuleDataList modulesList;
   EleveDataList elevesList;

   public EmpTpsData();
   public void init();
   public int addEleve(String nom);
   public void removeEleve(int index);
   public void removeAllEleves();
   public int currentEleve();
   public void removeCurrentEleve();
   public int getNumEleves();
   public String getEleveNom(int index);
   public int getEleveId(String nom);
   public boolean eleveSuitModule(int eleve, int module);
   public void setEleveToModule(int eleve, int module);
   public void unsetEleveToModule(int eleve, int module);
   public void setCurrentToModule(int module);
   public void unsetCurrentToModule(int module);
   public List getElevesList();
   public int addModule(String nom);
   public void removeModule(int module);
   public void removeAllModules();
   public int currentModule();
```

```
    public void removeCurrentModule();
    public int getNumModules();
    public String getModuleNom(int module);
    public int getModuleId(String nom);
    public int getModuleProf(int module);
    public List getModulesList();
    public List getModulesList4Eleves();
    public void setModuleProf(int module, int prof);
    public void setCurrentProf(int prof);
    public int addProf(String nom);
    public void removeProf(int prof);
    public void removeAllProfs();
    public int currentProf();
    public void removeCurrentProf();
    public int getNumProfs();
    public void setProfDispo(int prof, int plage);
    public void setProfNotDispo(int prof, int plage);
    public void setCurrentDispo(int plage);
    public void setCurrentNotDispo(int plage);
    public boolean getProfDispo(int prof, int plage);
    public boolean getCurrentDispo(int plage);
    public String getProfNom(int prof);
    public int getProfId(String nom);
    public List getProfsList();
    public List getProfsList4Modules();
}
```

B.2 affichage graphique

La classe principale (celle qui contient la méthode `main()` appelée au lancement par la machine virtuelle JAVA) est la classe `EmplTps` :

```
public class EmplTps{
    public static void main(String argv[]);
};
```

C'est la méthode `main()` qui crée un objet `Frame` (la fenetre graphique) et tous les composants de cette fenetre, en particulier les 3 panneaux principaux : `ElevesPanel`, `ModulesPanel`, `ProfsPanel`.