Projet MIM2

ENS Lyon - septembre 2001 - décembre 2001

Vivian Brégier & Laure Danthony

Réalisation d'un logiciel d'emploi du temps

PLAN :

- Présentation du projet et cahier des charges
- Le résultat en images !
- Quelques détails d'implémentation
- Conclusion.

Présentation du projet et cahier des charges

Le problème

PROJET : réaliser un logiciel Windows facile d'utilisation qui étant données des contraintes de type horaires, disponibilités des professeurs et non-collision des cours d'un élève donné, donne UNE solution possible au problème.

Les contraintes

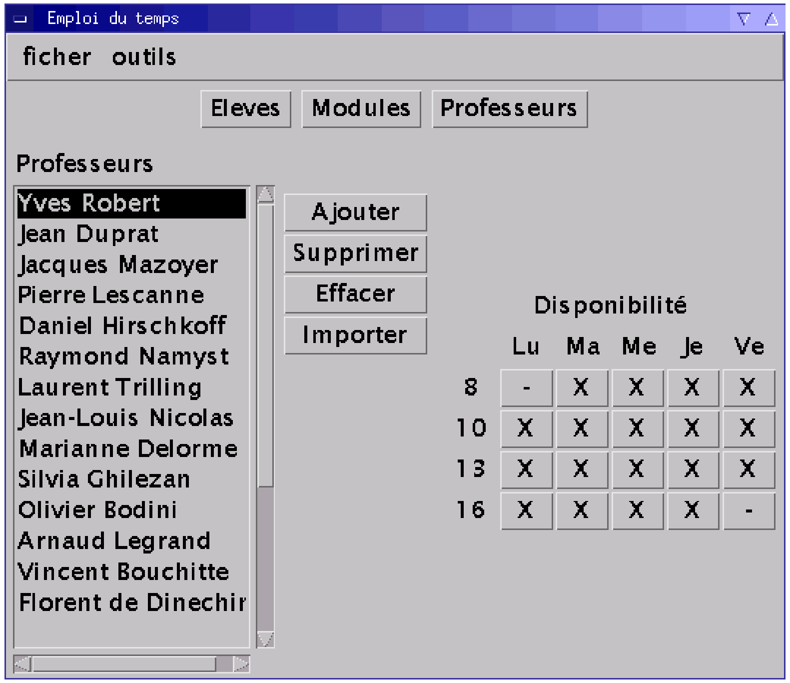
Elles sont de trois types :

- Un certain nombre de créneaux disponibles : 1, 2, ...
- Disponibilité des professeurs : `yrobert` peut être présent aux créneaux 4 et 7, ...
- Pas de collisions entre les cours des élèves : si `vbregier` va aux modules "algo_cours" et "compil_cours", alors ces deux modules ne peuvent avoir lieu en même temps !

Le résultat en images !

Rentrée des contraintes

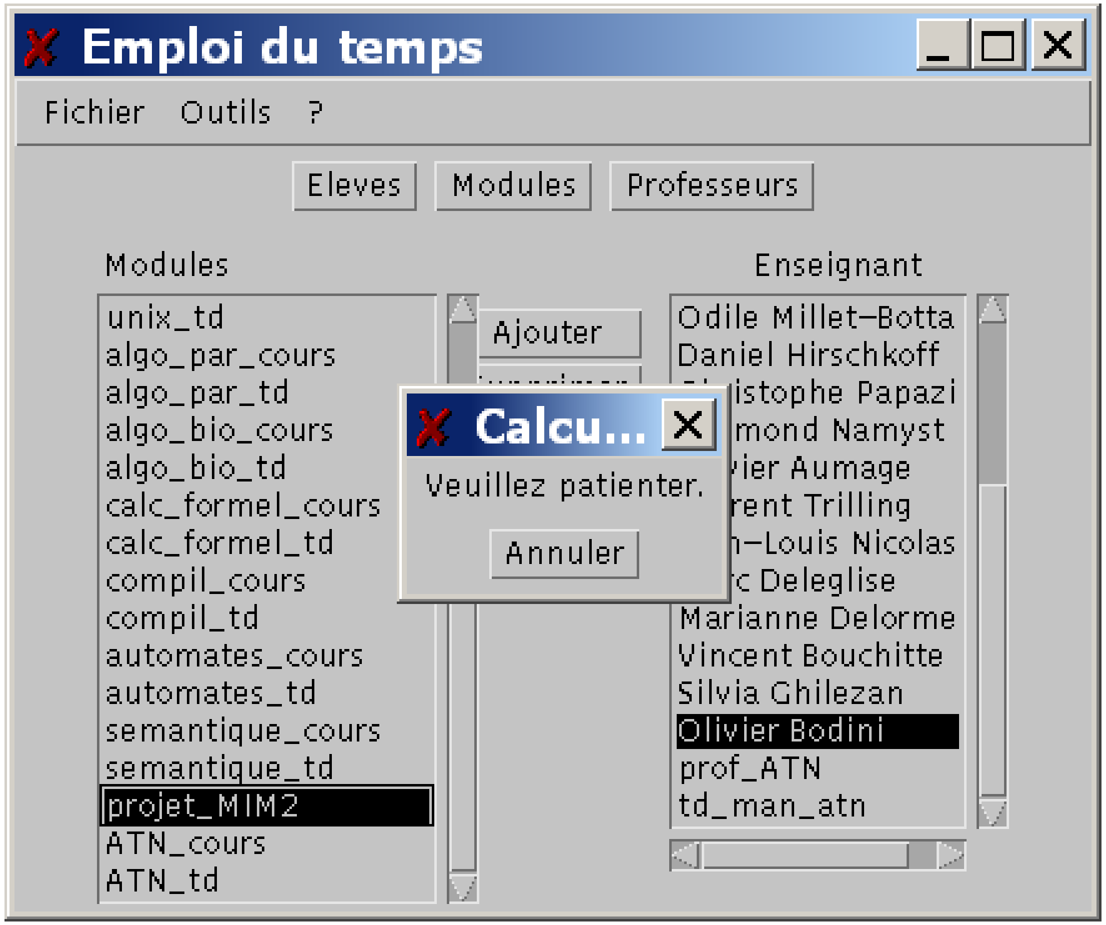
Rentrée des données à la souris, pour les professeurs, les modules et les élèves :



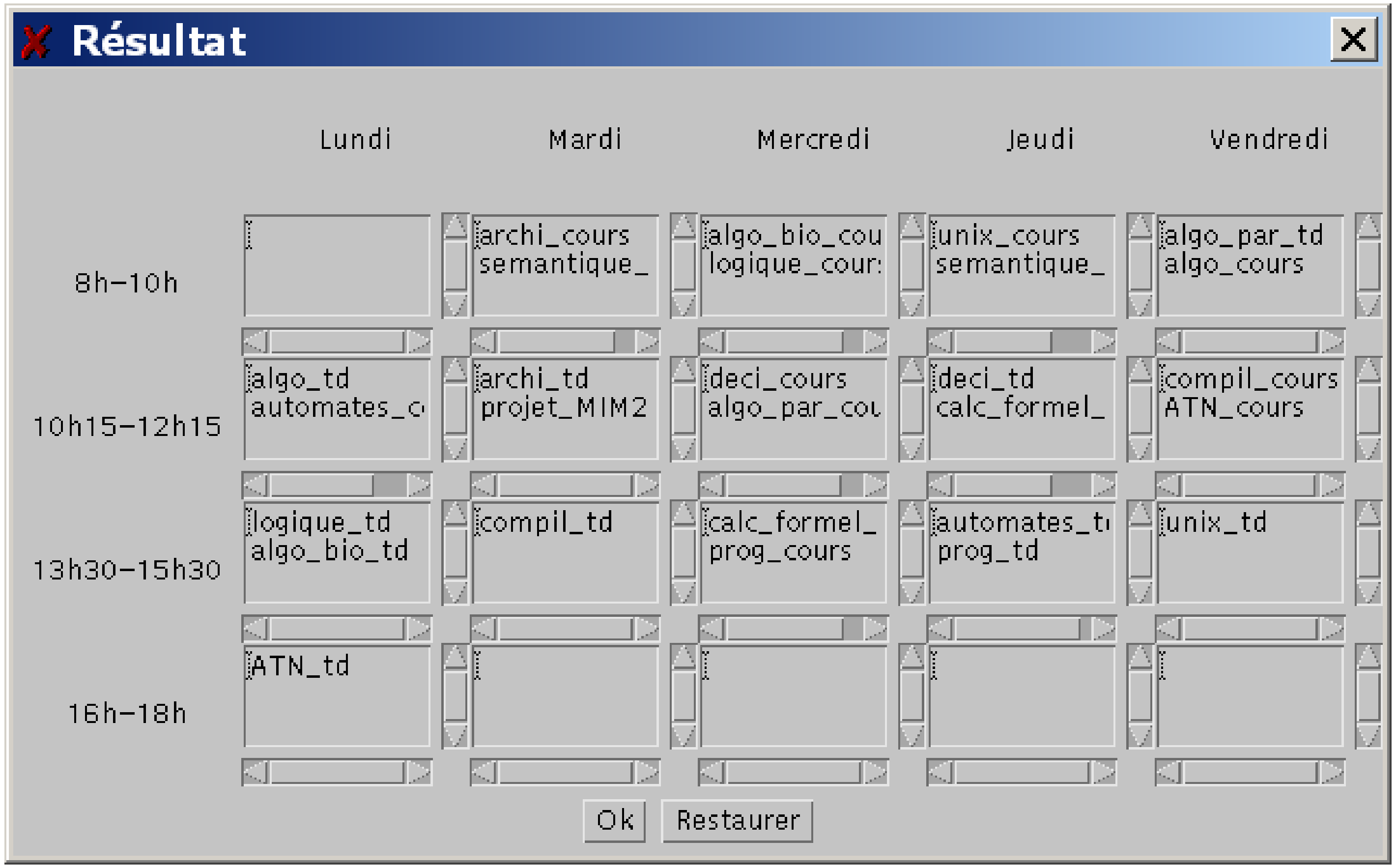
On peut sauver aussi ...

Calcul d'une solution

Un clic sur le bouton **`Calculer`** et 5 secondes d'attente. . .



Visualisation d'une solution



Quelques détails d'implémentation

Algorithme utilisé

```
let rec resolution sit_en_cours graphe =
  try (let (cours_a_placer,cren_poss)=
     selectionne_cours sit_en_cours graphe in begin
       let rec trouver_un_creneau=
         (function
          | [] -> raise NeMarchePas
          | cr::reste ->
               try
                 let nouvelle_sit=copie_situation sit_en_cours in
                   begin
                     attribuer nouvelle_sit cours_a_placer cr graphe;
                     resolution afficher nouvelle_sit graphe;
                   end
               with  | NeMarchePas-> trouver_un_creneau reste)
       in trouver_un_creneau cren_poss  end)
  with | PlusDeCours-> sit_en_cours.deja_places;;
```

Structures de données OCAML

```
type horaire=int;;

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

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};;
```

Etapes de l'implémentation OCAML

1. Récupération des données à partir d'un fichier généré par l'IGU Java. (fichier `recup_data.ml`)
2. Initialisation de la situation : liste des cours, horaires associés, graphe des dépendances ...(fichier `init_situation.ml`)
3. Calcul d'une solution. (fichier `calcule_solution.ml`)
4. Impression du fichier solution pour l'IGU Java. (fonction `imprime_res_dans_fichier`)
5. Le fichier `interface.ml` contient uniquement la fonction principale `calc : unit -> unit = <fun>`

Restrictions du logiciel

Le logiciel ne sait pas ...

- Récupérer d'(éventuelles) autres solutions Plus ou moins facile !
- Gérer les TD dédoublés Pas facile !
- S'arrêter au milieu d'un calcul si il s'éternise Facile !
- Imprimer son résultat ou l'exporter dans un autre format (type Excel) Plus ou moins facile !

Conclusion

Un travail d'équipe

- Vivian : partie Graphique Java
- Laure : partie Calcul OCaml

. . . et mise en commun rapide !

Un produit fini satisfaisant

Même si on pourrait encore effectuer quelques modifications, le produit est utilisable par Simone. Si elle dispose d'une solution, des permutations "à la main" peuvent lui donner une solution acceptable.