Programmation sous GNUstep (2)
gestion des préférences utilisateur

Nicolas Roard et Fabien Vallon

11 mai 2003

Ce mois-ci, après une petite révision du Design Pattern Modèle/Vue/Contrôleur (MVC) et de Gorm, nous allons voir comment GNUstep gère les préférences utilisateurs.
Nous allons également introduire un nouveau Design Pattern géré nativement par l'API GNUstep : l'Observateur.

News

GNUstep-gui (AppKit) et GNustep-back (Backend) sont passés en version 0.8.5. Les principaux changements sont :
- Diverses optimisations sur l'affichage
- De nombreuses corrections dans le système d'impression
- NSToolBar est partiellemnt implémenté

GNUstep-Base (Foundation) et GNUstep-make passent en version 1.6.0
- Meilleur support pour Microsoft Windows
- Nouvelles méthodes compatibles MacOSX.
- Les extensions de Foundation de GNUstep compilent nativement sous MacOSX.
- Améliorations d'autogsdoc (l'outil de génération automatique de documentation)

Applications GNUstep

Cela sort un peu du cadre de cet article, mais on constate l'apparition de plus en plus d'applications intéressantes utilisant le framework, et un petit coup de projecteur ne fait pas de mal :
- Connect, un frontend a pppd `http://stepmaker.sourceforge.net/connect.html`
- Popup, un logiciel d'aide à la traduction `http://popup.sourceforge.net/`
- Camaeleon, un moteur de thème pour GNUstep `http://www.roard.com/camaelon`
- Poe, un éditeur ogg `http://www.eskimo.com/~pburns/rob/Poe/`
- SVGViewer, un visualisateur de fichiers SVG
- Zillion, un programme permettant d'effectuer des calculs répartis `http://zillion.sourceforge.net/`
- RuntimeBrowser, un browser de classes à partir du runtime Objective-C `http://www.people.virginia.edu/~yc2w/GNUstep/english/`
- Mplayer.app (une interface graphique à mplayer) `http://gnustep-apps.org/fabien/Mplayer/`

N'oublions pas les applications existantes mises à jour :
- CDPlayer (0.3) , un lecteur de CD
- CodeEditor (0.4) , un éditeur de texte
- InterfaceWM (0.2.4) , un WindowManager GNUstep
- StepTalk (0.7.1) , un frameWork permettant de scripter les applications GNUstep
- Pantomime (1.1.0pre2) , le JavaMail de GNUstep
- GNUMail (1.1.0pre2) , un lecteur de mail
- GWorkspace (0.5.1) , un gestionnaire de bureau
- Helpviewer (0.3) , un visualisateur d'aide en ligne

Enfin, un projet très intéressant vient d'être démarré par Marko Riedel, il s'agit de la rédaction d'un "livre de recettes de cuisine" concernant la programmation GNUstep[1] !

NSWindowController et Design Pattern MVC

GNUstep possède une classe NSWindowController qui va nous faciliter la mise en place du Design Pattern MVC.
NSWindowController gère notamment :
- l'instanciation de la fenêtre (généralement le .gorm) via la méthode initWithWindowNibName :
- la fermeture de la fenêtre close, shouldCloseDocument, setShouldCloseDocument :

[1] disponible sur `http://www.roard.com/docs/cookbook/`

– la sauvegarde des propriétés de la fenêtre (position/taille...) setWindowFrameAutosaveName :, windowFrameAutosaveName :

Remarques à propos de l'interface utilisateur

Sauvegarder automatiquement la taille et la position des fenêtres utilisées est un exemple de l'approche utilisateur sous GNUstep ; on peut citer le guide de création d'interface graphique de NeXT :
"La similarité entre les objets réels et leur représentation graphique est à considérer à un niveau fondamental, et non superficiel. Les objets graphiques n'ont pas besoin de ressembler à leurs équivalents physiques dans leurs moindres détails. Mais ils doivent se comporter de la façon dont notre expérience avec des objets réels nous y prépare. Par exemple, les objets du monde réel restent à l'endroit où on les a mis." [2]

Quelques classes et widgets...

Avant de mettre en place notre MVC, expliquons d'abord le principe des classes NSView,NSCell et NSMatrix du framework GNUstep.

NSView et NSCell

GNUstep dispose de deux classes permettant d'afficher quelque chose dans une interface graphique :
– NSView est la classe abstraite chargée de l'affichage, de la gestion des évènements (souris, clavier, etc.), et de l'impression (postscript). Tous les widgets disponibles dans l'ApplicationKit sont ainsi dérivés de cette classe.
– NSCell est une classe permettant juste d'afficher du texte et des images, mais qui du coup est plus légère que NSView.

En pratique, les widgets GNUstep utilisent la plupart du temps des NSCells pour leur affichage plutôt que de surcharger directement la méthode de dessin de la classe NSView, permettant un meilleur découpage.

[2] *" The similarity of graphical to real objects is at a fundamental rather than a superficial level. Graphical objects don't need to resemble physical objects in every detail. But they do need to behave in ways that our experience with real objects would lead us to expect.For example, objects in the real world stay where we put them"* (NeXT UI Guide)

NSMatrix

NSMatrix est un objet permettant de regrouper des objets NSCell et les gérer. Il permet de les grouper en lignes et en colonnes. Les NSCell utilisés sont en général de la même classe (comme nous le verrons dans notre cas), mais on peut éventuellement utiliser différentes classes dans un même NSMatrix ; la seule contrainte étant que les NSCells doivent nécessairement avoir la même taille.

Utilisation de NSWindowController

Revenons à notre MVC, et prenons comme exemple d'utilisation de NSWindowController le panneau des préférences de notre application "Todo" commencée dans le précédent article.
On choisira ici de sauvegarder "à la volée" (pas de bouton "appliquer") les préférences, donnant aussi plus d'interactivité.
Nous avons donc les composants suivants :
– Preferences.gorm (La Vue)
– PreferencesController (Le Contrôleur)
– NSUserDefaults (Le Modèle)

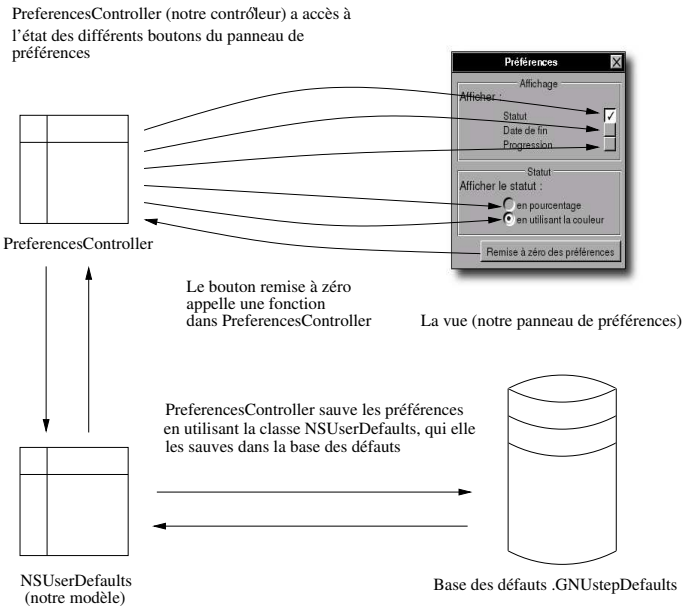

FIG. 1 – Schéma MVC utilisé

Le schéma 1 nous montre les relations entre les différents composants.

Il est à noter que pour toutes les applications basées sur GNUstep le panneau Préférences est (doit être) accessible via le menu Infos->Préférences .
Créons maintenant notre panneau de Préférences avec Gorm :
Document->New Module->New Empty
Dans le panneau Palettes choisissez la section fenêtre et drag & dropper (faites glisser) un panel sur le bureau.

FIG. 2 – Palette fenêtre

Note : Il est important de choisir un panneau plutôt qu'une fenêtre car dans GNUstep (et OPENSTEP), lorsqu'une application perd le focus, ces panneaux deviennent invisibles (cela permet de ne pas encombrer l'espace de travail).
Placez-y deux boites (Box). Dans la première nous y mettrons une "Matrice" (un objet NSMatrix) de boutons.
Pour cela drag & droppez un SwitchButton puis sélectionnez ce bouton dans la fenêtre, appuyez sur la touche Alt et tirez simultanément vers le bas pour obtenir une matrice de 3 boutons.
Appelez les :
– Statut
– Date de fin
– Avancement
Faites de même pour le status : drag & droppez un RadioButton et créez la matrice. Positionnez des tags en utilisant l'inspecteur (voir article précédent) pour chacun de vos éléments, par exemple :
– Pourcent tag : 0
– Couleur tag : 1
Ajoutez ensuite un bouton "remise à zéro" qui permettra de remettre les valeurs par défaut des préférences.
vous devez obtenir un panneau ressemblant à la figure 3.

FIG. 3 – Panneau des préférences de notre application

Créons maintenant notre sous classe de NSWindowController, PreferencesController (figure 4).

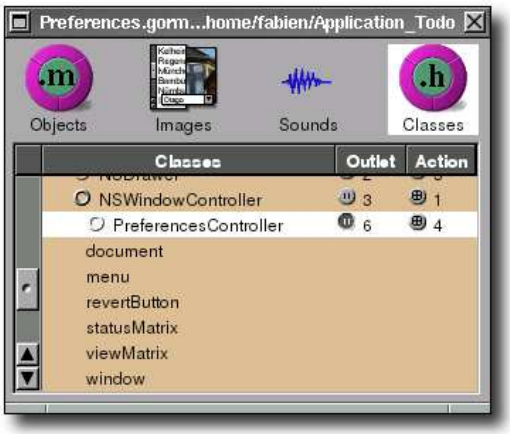

FIG. 4 – classe PreferencesController

Ajoutez les Outlets suivants :
– revertButton
– statusMatrix
– viewMatrix
et les Actions :
– statusAction :
– viewAction :
– revertAction :
Retournez ensuite dans "Objects", cliquez sur NSOw-

ner et choisissez PreferencesController. Cela indique que le propriétaire du fichier Gorm (représenté par Gorm par l'icone NSOwner), c'est à dire l'objet qui va charger le fichier, sera une instance de la classe PreferencesController.
On peut alors connecter directement les Outlets et Actions sur le NSOwner. Connectez les Outlets statusMatrix et viewMatrix aux deux NSMatrix que vous avez mis sur le panneau de préférences, et l'Outlet revertButton vers le bouton "remise à zéro".
Connectez ensuite les actions, des deux NSMatrix vers NSOwner (Actions viewAction : et statusAction :) et du bouton "défauts" vers l'action revertAction : du NSOwner.
Notre contrôleur est ainsi relié à la vue (le panneau préférences), le fichier Gorm contenant ainsi toutes les relations. Il nous reste à nous occuper du modèle.

NSUserDefaults : La gestion des préférences

Comme nous vous l'avions déjà dit, les spécifications OpenStep intègrent certaines classes qui aident à la gestion d'un environnement utilisateur (ce qu'on appelle plus communément sous linux un *desktop*).
La gestion des préférences utilisateur en fait partie.
Une base de préférences est maintenue dans `GNUSTEP_USER_ROOT/Defaults/.GNUstepDefaults` . (qui correspond en principe au répertoire `~/GNUstep/Defaults/.GNUstepDefaults`).
Contrairement à d'autres environnements, cette base de préférences est un simple fichier ascii, lisible en clair. Le format utilisé est une "propertylist" (si vous utilisez WindowMaker, il s'agit du même format que ses fichiers de configuration).
Par exemple, pour stocker un tableau d'objets, les délimiteurs sont des parenthèses, pour stocker une chaîne de caractères les guillemets sont utilisés, et pour stocker un dictionnaire, ce sont les accolades.
Les objets qui peuvent être stockés tel quel dans une propertylist sont NSData, NSString, NSNumber, NSDate, NSArray, et NSDictionary. Ils peuvent être lus et écrits directement.
exemple d'une base defaults :

```
{
   ImageViewer = {
      "NSWindow Frame Inspector" = "
          382 192 260 382 0 0 960 768 ";
      "NSWindow Frame Preferences" = "
          382 233 260 303 0 0 960 768 ";
   };

   NSGlobalDomain = {
      GSFontAntiAlias = YES;
      NSLanguages = (
         French,
         English
      );
      "back-art-subpixel-text" = 0;
      controlBackgroundColor = {
         Alpha = 1.000000;
         B = 0.83;
         ColorSpace =
             NSCalibratedRGBColorSpace;
         G = 0.85;
         R = 0.86;
      };
      windowBackgroundColor = {
         Alpha = 1.000000;
         B = 0.83;
         ColorSpace =
             NSCalibratedRGBColorSpace;
         G = 0.85;
         R = 0.86;
      };
   };
}
```

Ici deux clés sont présentes (ImageViewer et NSGlobalDomain). NSGlobalDomain est la clé où les préférences globales de l'utilisateur sont sauvegardées (langue, timezone ...). On peut se servir de l'application Preferences.app pour configurer ces défauts sous forme graphique [3].
ImageViewer est la clé sauvant les préférences de l'application ImageViewer ; elle contient la taille et la position de deux fenêtres de cette application, et ces préférences ont été automatiquement gérées et sauvegardées par GNUstep et l'utilisation de la classe NSWindowController.

Accès aux défauts par la classe NSUserDefaults

On accède aux préférences via la méthode de classe

```
+ (NSUserDefaults *)standardUserDefaults
```

de NSUserDefaults, qui nous retourne une instance de NSUserDefaults. NSUserDefaults est en fait un singleton, c'est à dire qu'il n'y a qu'une seule et

[3] Preferences.app : `http://prefsapp.sf.net`

même instance de NSUserDefaults par programme (la méthode standardUserDefaults crée une instance si elle n'existe pas déjà, et retourne sinon l'instance existante). Ce Design Pattern permet d'éviter d'éventuels problèmes liés au maintien de la cohérence.

On peut alors accéder aux préférences via les méthodes :

```
- (NSArray *)arrayForKey:(NSString *)
    defaultName
- (BOOL)boolForKey:(NSString *)defaultName
- (NSData *)dataForKey:(NSString *)
    defaultName
- (NSDictionary *)dictionaryForKey:(
    NSString *)defaultName
- (float)floatForKey:(NSString *)defaultName
- (int)integerForKey:(NSString *)
    defaultName
- (id)objectForKey:(NSString *)defaultName
- (NSArray *)stringArrayForKey:(NSString
    *)defaultName
- (NSString *)stringForKey:(NSString *)
    defaultName
```

Les modifier avec les méthodes :

```
- (void)setBool:(BOOL)value forKey:(
    NSString *)defaultName
- (void)setFloat:(float)value forKey:(
    NSString *)defaultName
- (void)setInteger:(int)value forKey:(
    NSString *)defaultName
- (void)setObject:(id)value forKey:(
    NSString *)defaultName
```

ou les supprimer avec

```
- (void)removeObjectForKey:(NSString *)
    defaultName
```

L'écriture (sur disque) se fait par la méthode

```
- (BOOL) synchronize
```

Les préférences accessibles ainsi sont celles de l'utilisateur courant et de l'application qui y accède.

Les préférences, en plus d'être manipulables de l'intérieur des programmes, peuvent être aussi modifiées avec un outil en ligne de commande : defaults.

```
defaults write NSGlobalDomain test "Bonjour tout le Monde"
```

rajoutera ainsi la clé test de valeur "Bonjour tout le Monde" dans le domaine NSGlobalDomain :

```
test = "Bonjour tout le Monde" ;
```

```
defaults delete NSGlobalDomain test
```

supprimera la clé test du domaine NSGlobalDomain.

Implémentation pour Todo.app

Nous avons décidé de changer "à la volée" les préférences (i.e. leur effet est immédiat) pour éviter à l'utilisateur les pénibles actions Appliquer/OK/Annuler .
Voici donc l'implémentation des méthodes viewAction et statusAction de la classe PreferencesController. Ce code est à rajouter au fichier PreferencesController.m que vous aurez généré avec Gorm (Menu Class->Create Class Files) .

```
static NSString *STATUS = @"STATUS";
static NSString *PERCENT = @"PERCENT";
static NSString *COLOR = @"COLOR";
```

et dans l'implémentation de la classe PreferencesController :

```
-(void) awakeFromNib
{
  defaults = [NSUserDefaults
      standardUserDefaults];

  //Nous gardons en mémoire les préférences
       utilisateur
  //Lors du lancement de Préférences
  _revertDict =[NSDictionary
      dictionaryWithDictionary : [defaults
      dictionaryRepresentation]];
  [_revertDict retain];

  //On affiche les préférences
  [self _displayPrefsWithDict:_revertDict];
}
```

```
-(IBAction) statusAction:(id) sender
{
  if ( sender != statusMatrix )
    return;
  {
    //On recupère le tag de la sélection
```

```
    int tag = [[sender selectedCell] tag];

    //On effectue les changements dans notre
        instance des préférences
    if ( tag == 0 )
      [defaults setObject:PERCENT forKey:
          STATUS];
    if ( tag == 1 )
      [defaults setObject:COLOR forKey:
          STATUS];

    //On synchronise pour sauver les défauts
    [defaults synchronize];

    [revertButton setEnabled: YES];
  }
~}
```

```
-(IBAction) revertAction:(id) sender
{
  if ( sender != revertButton )
    return;

  //on affiche les préférences avec les valeurs
      par défaut
  [self _displayPrefsWithDict:_revertDict];
  [revertButton setEnabled:NO];
}
```

Nous souhaiterions cependant que les changements effectués dans les préférences se répercutent sur notre application (par exemple ....). Une première solution serait de passer directement les messages à notre fenêtre principale.

Soyons plus élégant :-)
Il est possible que des préférences intéressent d'autres parties de notre application. Pour cela nous allons utiliser un autre grand classique des Designs Patterns : L'**Observateur** .

Le Design Pattern Observateur

Ce Design Pattern (figure 5) permet de simplifier le maintien de la cohérence quand on travaille avec des objets qui dépendent de l'état d'un autre objet.
Les objets "observateurs" s'enregistrent alors auprès de l'objet qu'ils veulent surveiller. Celui-ci envoie alors, à tous les observateurs, un message lors d'un changement d'état.

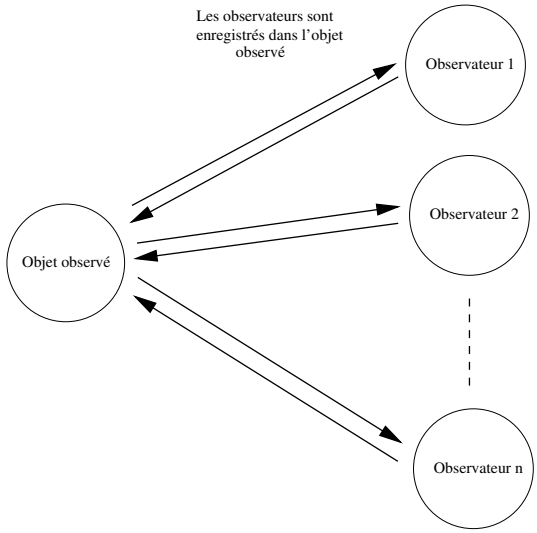

FIG. 5 – Design Pattern Observateur

Ce pattern permet également d'envoyer des messages à des objets qu'on ne connait pas forcément, voir même dont le nombre évolue dynamiquement pendant que l'application fonctionne.

Les notifications

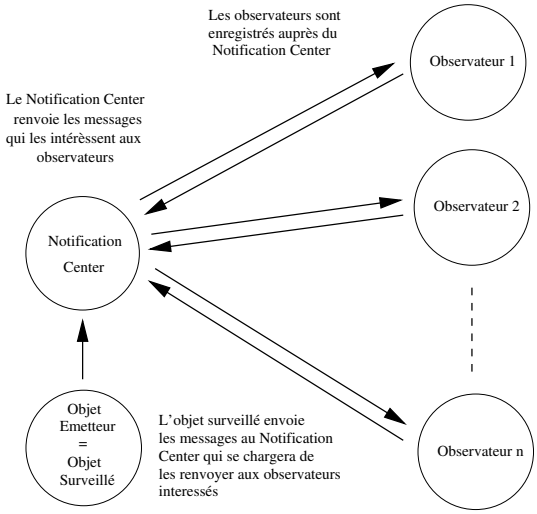

FIG. 6 – Les notifications sous GNUstep

GNUstep dispose en fait d'une implémentation pratique de ce Design Pattern, au travers des notifications. Une notification est un message qu'un ob-

jet quelconque peut envoyer à un objet NSNotificationCenter, qui va centraliser les notifications reçues. Si un ou plusieurs autres objets sont intéressés par un type de message donné, ou provenant d'un objet donné, ils s'inscrivent comme récepteurs chez le NSNotificationCenter. Celui-ci sert ainsi d'aiguilleur (voir figure 6 page précédente). Cette organisation permet du coup très simplement d'envoyer des notifications distribuées (à des objets distants, d'une autre application, sur la même machine ou carrément à travers le réseau), en changeant simplement d'objet "aiguilleur" . Il suffit ainsi de remplacer l'objet NSNotificationCenter par un objet NSDistributedNotificationCenter.
Les méthodes de NSNotificationCenter couramment utilisées sont :

```
- (void)postNotificationName:(NSString *)
    notificationName object:(id)anObject
    userInfo:(NSDictionary *)userInfo
```

pour poster une notification,

```
- (void)addObserver:(id)anObserver selector
    :(SEL)aSelector name:(NSString *)
    notificationName object:(id)anObject
```

pour ajouter un observateur, et

```
- (void)removeObserver:(id)anObserver name
    :(NSString *)notificationName object:(
    id)anObject
```

pour supprimer un observateur.

Utilisons les notifications

NSUserDefaults poste déjà une notification lorsque les préférences ont été synchronisées. Cette notification s'appelle `NSUserDefaultsDidChangeNotification`. Il nous suffit donc d'ajouter un Observateur dans les classes qui veulent être au courant du changement, ici dans TodoController.
Dans la méthode init de TodoController nous rajoutons donc :

```
[[NSNotificationCenter defaultCenter]
 addObserver:self
 selector:@selector(prefsChanged:)
 name: NSUserDefaultsDidChangeNotification
 object:nil];
```

Cet ajout indique au NSNotificationCenter que nous rajoutons l'objet TodoController (self) en tant qu'observateur de la notification `NSUserDefaultsDidChangeNotification`.

Lorsque l'observateur recevra la notification, la méthode prefsChanged : sera appelée avec pour paramètre la notification (un objet NSNotification).

La notification aura pour objet l'instance de NSUserdefaults. Il nous suffit alors d'appliquer les changements à notre interface.

```
-(void) prefsChanged: (NSNotification *)
    aNotification
{
 //Nous récupérons l'instance de UserDefaults
      preference
 NSUserDefaults *defaults = [aNotification
      object];

 //Nous appellons des méthodes (privées) qui vont
      traiter
 //les differents changements
 [self _reloadWithTableColumns: [defaults
    objectForKey:VIEW]];
 [self _changeStatusDisplay:[defaults
    objectForKey:DISPLAYSTATUS]];
}
```

Conclusion

Nous avons ainsi vu dans cet article une particularité importante de GNUstep, l'utilisation d'une base de données accessible en clair, contenant les défauts de toutes les applications et de l'environnement.

Cette gestion des défauts permet facilement et de façon standard au programmeur de sauvegarder l'état de son application et les préférences de l'utilisateur.

L'utilisation des notifications est un autre exemple d'un Design Pattern directement disponible dans le framework GNUstep, permettant de plus une communication inter-applications (avec les notifications distribuées).

Le canal irc #gnustep sur le réseau freenode (irc.debian.org par exemple) est un bon endroit où trouver de l'aide si besoin est. N'oubliez pas non plus les nombreux sites sur GNUstep, Cocoa ou OpenStep, comme ceux donnés en référence à la fin de cet article !

N'hésitez pas à nous contacter si vous avez des re-

marques ou des suggestions à nous faire.

Nicolas Roard, <nicolas@roard.com>
Fabien Vallon, <fabien@tuxfamily.org>

Quelques liens :

Le wiki GNUstep : http://wiki.gnustep.org/
La page tutoriels du wiki : http://wiki.gnustep.org/index.php/Tutorials
La page des news du wiki : http://wiki.gnustep.org/index.php/News%20Page
StepWise, un site contenant beaucoup d'articles intéressants sur la programmation OpenStep : http://www.stepwise.com
Un article sur les NSCell,NSControl,NSView : http://www.stepwise.com/Articles/Technical/NSCell.html
Un article sur les défauts : http://www.stepwise.com/Articles/Technical/ApplicationStorage.html