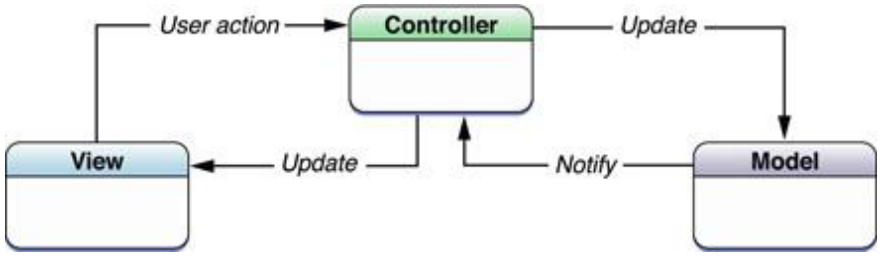
Manipulation 2 : Création d'une « horloge ».

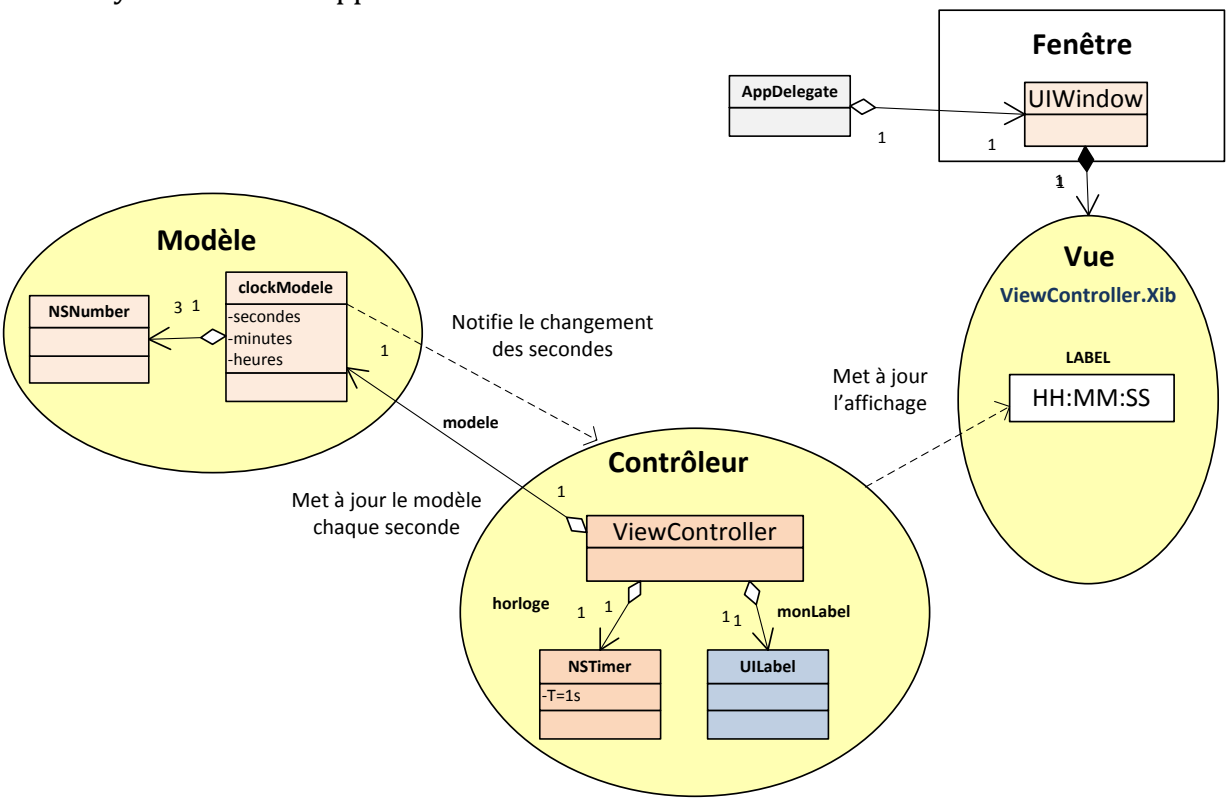
Cette manipulation a pour but de mettre en place « à la main » un modèle MVC « Modèle Vue Contrôleur » afin de se familiariser avec ce modèle de programmation.
Le programme consistera simplement à afficher l'heure au format **HH :MM :SS** dans une vue contenant un unique contrôle de type **label**.
Nous utiliserons pour cela un projet de type « **Single View Application** », modèle de projet consistant en une application graphique basée sur une unique fenêtre sans autre contrôle et utilisant le modèle MVC.



Le modèle MVC retenu répondra au schéma ci-dessous pour lequel :

- ✓ la vue est implémentée dans le fichier Interface Builder **ViewController.xib** contenant un objet graphique de type **Label**,
- ✓ le modèle est implémenté dans une classe **clockModele**,
- ✓ le contrôleur est implémenté dans la classe déléguée de la vue **ViewController** en charge du traitement des messages de la vue.

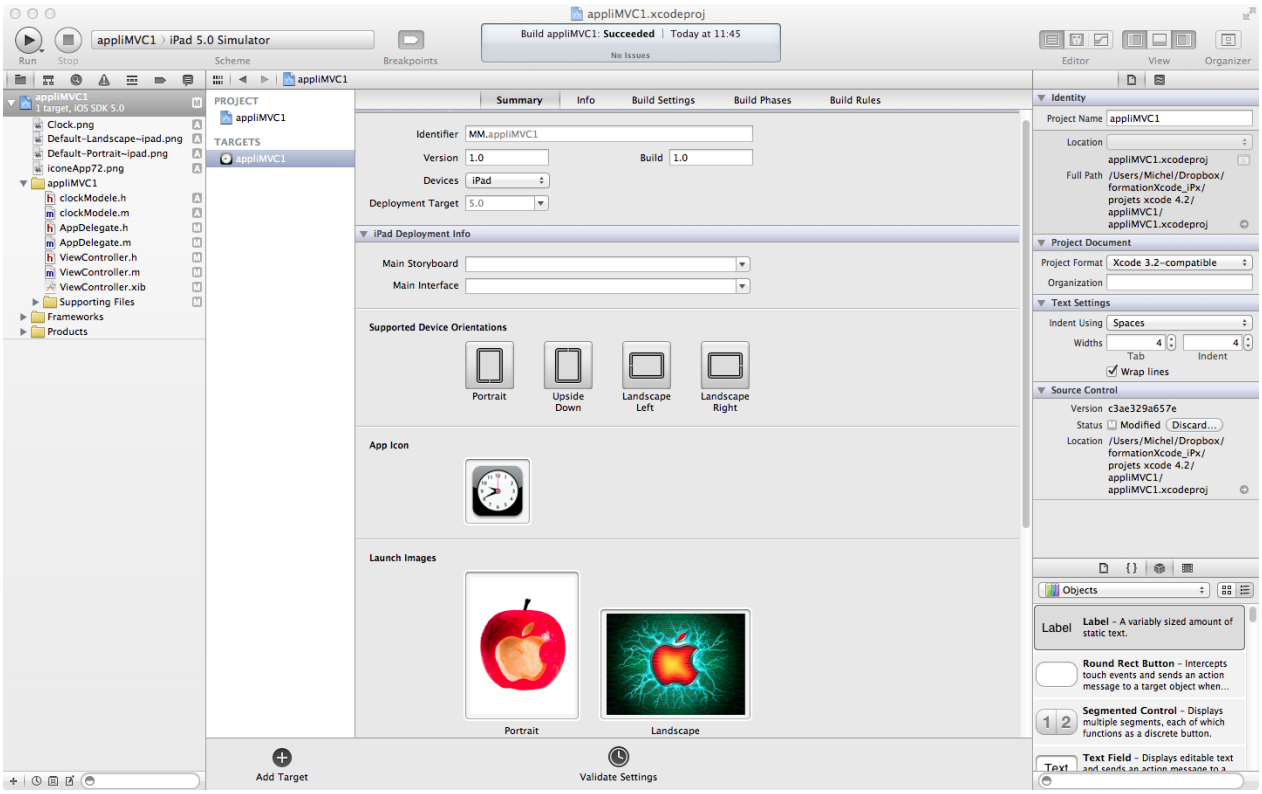
La classe **AppDelegate** quant à elle gère les messages de l'application et affiche la vue dans la fenêtre de l'application. Pour rappel, les applications ios ne disposent que d'un objet fenêtre, celui-ci ayant pour rôle de charger les différentes vues de l'application qui apparaîtront dans la fenêtre au cours du cycle de vie de l'application.



Création du projet

1. Créer un projet Xcode de type « **Single View Application** » et choisir une cible de type iPad. Par souci de cohérence avec le nommage utilisé dans la présentation, nommer le projet **appliMVC1**. Décocher les options « **Use Storyboard** » et « **use ARC** ».
2. Configurer la cible du projet avec :
    a. le fichier **Clock.png** comme icône associée à l'application, les icônes sur Ipad doivent avoir une résolution de 72x72 pixels,
    b. le fichier **feh_ipad.png** comme fond d'écran paysage de l'application, pour l'Ipad il doit avoir une résolution de 1024x748 pixels,
    c. le fichier **fev_ipad.png** comme fond d'écran portrait de l'application, pour l'Ipad il doit avoir une résolution de 768x1004 pixels.

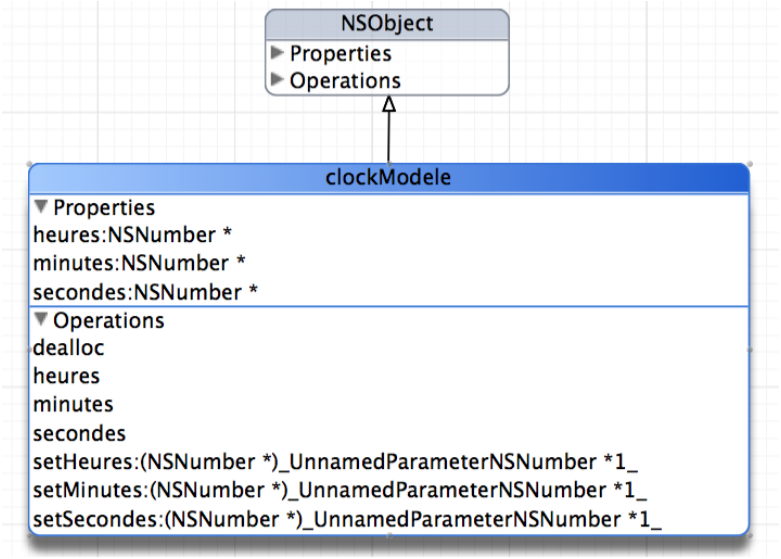
Attention, contrairement à ce que l'on pourrait croire ce ne sont pas des fonds d'écran affichés lorsque l'application est active, ils sont uniquement affichés pendant la phase de démarrage de l'application pour éviter la phase « écran noir ». En conséquence, si l'application n'est pas trop « lourde » on n'a pas le plaisir de les voir apparaître.
Associer ces ressources à l'aide de la fenêtre résumé de votre application comme ci-dessous :



Création du modèle

La classe modèle de notre application devra respecter le diagramme de classe ci-dessous, elle héritera donc de la classe de base du **framework Cocoa** puis sera constituée :

- ✓ de trois attributs heures, minutes et secondes de type **NSNumber***, type générique pour représenter un scalaire,
- ✓ de trois propriétés auto-générées pour accéder en lecture écriture aux attributs précédents,
- ✓ d'une méthode **dealloc** pour la libération des ressources de la classe.



3. Ajouter au projet une classe « **File->New->New File** » de type **Objective-C class** et vérifier quelle hérite bien de la classe **NSObject**.
4. Ajouter à cette classe les attributs du modèle pour l'heure, les minutes et les secondes ainsi que les propriétés auto-générées associées. Les propriétés seront du type « non atomiques » et utiliseront le comptage de référence.

   Déclaration pour la propriété « heures » :

```
@interface clockModele : NSObject {
      NSNumber* heures;
}
@property (nonatomic,retain) NSNumber* heures;
```

   Auto-génération de la propriété :

```
@synthesize heures;
```

5. Ajouter la méthode **dealloc** et la compléter par la libération des attributs.
6. Construire le projet et corriger les improbables erreurs !

Chacun aura remarqué la possibilité que présente ce code d'appeler une méthode sur une instance nulle « **nil** », ce qui arrivera si on instancie un objet **clockModele** puis le libère sans avoir entre deux initialisés les attributs… Ceci ne pose pas de problème en Objective-C qui ne considère pas comme une erreur l'envoi d'un message à un pointeur « **nil** ». Si la méthode appelée sur l'objet « **nil** » est sensée retourner une valeur, elle retournera 0 sur un type numérique, « **nil** » sur un type pointeur et une structure avec tous ses champs à 0 ou « **nil** » pour une valeur retournée de type structure.

Gestion du modèle par le contrôleur (première et courte partie)

Le contrôleur de la vue est implémenté au sein de la classe **ViewController**. Il utilisera un objet **NSTimer** pour déclencher un événement toutes les secondes, cet événement servant à mettre à jour les données du modèle, secondes, minutes et heures.

7. Ajouter au contrôleur un attribut référençant le modèle nommé « **modele** », un attribut **NSTimer*** nommé « **horloge** » et un attribut **IBOutlet UILabel*** nommé « **monLabel** ». Ajouter les propriétés auto-générées associées. Les propriétés seront du type « non atomiques » et utiliseront le comptage de référence.
8. Compléter la méthode **dealloc** avec la libération des attributs précédents.

C'est tout pour le moment, nous allons à présent créer la vue avant d'y revenir pour terminer le contrôleur.

Création de la vue

La vue de notre application se contentera d'inclure un label affichant l'heure courante, elle pourra avoir l'allure ci-dessous (vous remarquerez que je ne m'arrête pas de travailler durant la pause du déjeuner…) :

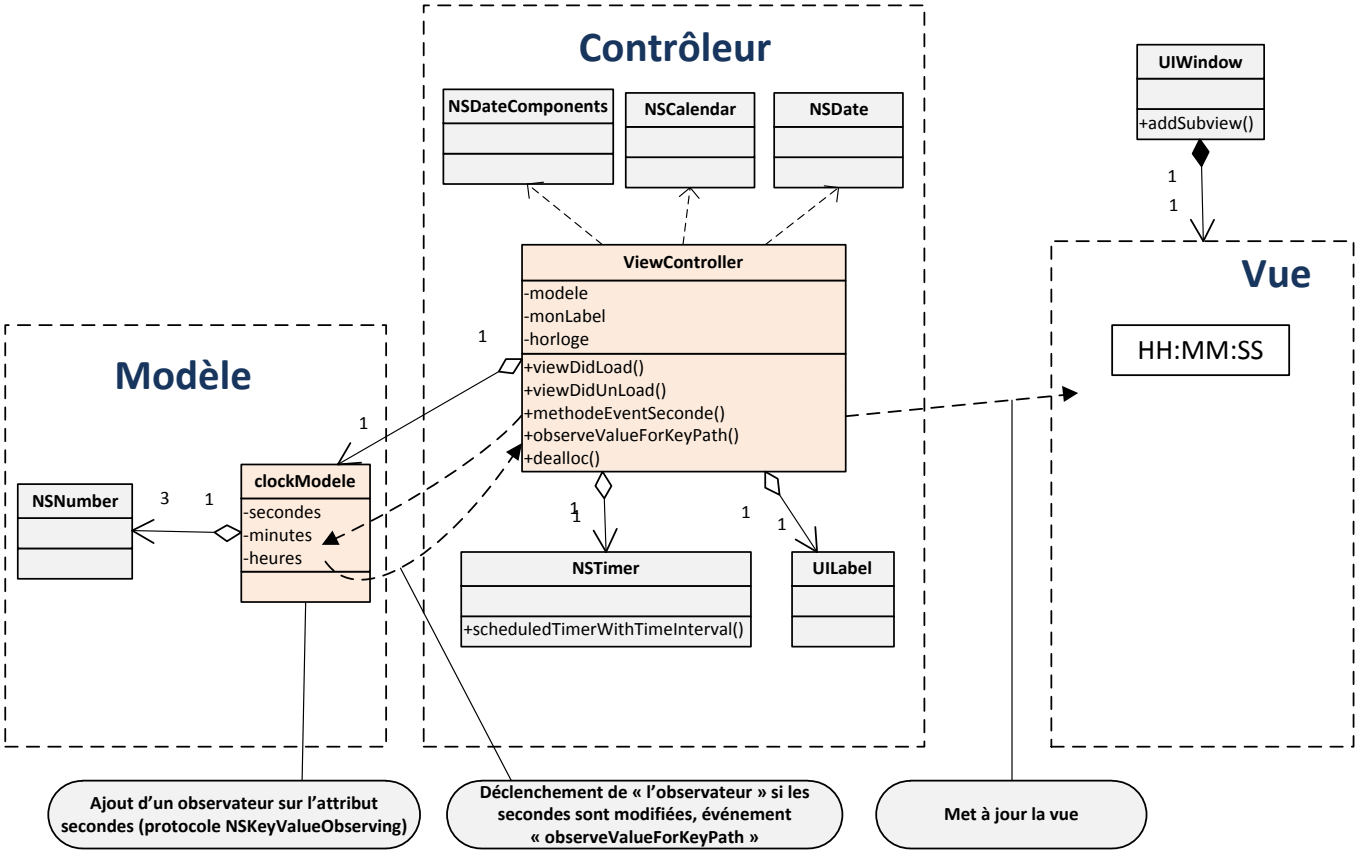
Gestion du modèle par le contrôleur (suite et fin)

Le principe retenu pour la mise à jour du modèle et de la vue associée est le suivant :

- ✓ le timer (horloge) déclenche un événement chaque seconde pour mettre à jour le modèle en fonction de l'heure courante,
- ✓ on abonne le contrôleur pour l'observation de la propriété secondes du modèle de données,
- ✓ dès qu'un changement survient sur cette donnée la méthode abonnée du contrôleur de vue est activée afin de mettre à jour la vue.

Pour l'abonnement d'une propriété à l'observation on dispose d'une méthode d'observation du protocole **NSKeyValueObserving** déclenchée lorsque qu'un changement survient sur un attribut ou sa propriété précédemment abonnée à l'observation à l'aide de la méthode **addObserver** du même protocole.

Le diagramme de classes final de l'application est donné page suivante afin de tenter de rendre plus claires les relations entre les différents éléments à mettre en place…



9. Compléter la méthode **viewDidLoad** de la classe **ViewController** afin d'instancier l'objet **NSTimer** via la propriété horloge pour qu'il déclenche une méthode nommée **methodeEventSeconde:** chaque seconde.
10. Compléter cette même méthode afin d'abonner l'objet courant à l'observation sur la propriété du modèle KVC **modele.secondes**.
11. Compléter la méthode **viewDidUnLoad** de la classe **ViewController** afin de stopper le timer et de supprimer l'objet courant de l'observation du modèle KVC **modele.secondes**.
12. Ajouter la méthode **dealloc** afin de libérer les ressources horloge, label et modele.

    Le code devrait ressembler à ceci :

```
- (void)viewDidLoad {
    [super viewDidLoad];
    modele = [clockModele new];
    horloge = [NSTimer scheduledTimerWithTimeInterval:1.0 target:self
    selector:@selector(methodeEventSeconde:) userInfo:nil repeats:YES];
    [self addObserver:self forKeyPath:@"modele.secondes"
    options:NSKeyValueObservingOptionNew  context:NULL];
}
- (void)viewDidUnload {
    [super viewDidUnload];
    [horloge invalidate];
    [self removeObserver:self forKeyPath:@"modele.secondes"];
}

-(void)dealloc {
    [horloge invalidate];
    [horloge release];
    monLabel = nil;
    [modele release];
}
```

Si ce n'est pas déjà fait, aller jeter un coup d'œil à la classe **NSTimer** dans l'aide, puis à la méthode **scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:**.

13. Copier le prototype de cette méthode depuis l'aide afin de l'ajouter au contrôleur puis la renommer en **methodeEventSeconde**.
14. Coder cette méthode afin quelle mette à jour les données heures minutes et secondes du modèle. Pour ceci on pourra utiliser :
    a. La classe **NSDate** qui permet d'obtenir la date courant à partir de la méthode **date**,
    a. la classe **NSCalendar** qui à partir de sa méthode **currentCalendar** retourne un objet calendrier formaté selon la localisation configurée dans les réglages du iDevice,
    b. la méthode **components** appliquée à l'objet calendrier pour la date courante qui permet d'extraire du calendrier les informations souhaitées désignées par une combinaison de constantes (heures, minutes et secondes dans notre cas),

    Le code devrait ressembler à ceci :

```
- (void)methodeEventSeconde:(NSTimer*)theTimer
  {
       // La date actuelle
       NSDate* dateNow = [NSDate date];
       // Un calendrier pour stocker la date et la "décortiquer"
       NSCalendar* calendrier =[NSCalendar currentCalendar];
       //On décortique les éléments du calendrier pour la date du jour
       NSDateComponents* composantsDeLaDateHMS = [calendrier components:
         (NSHourCalendarUnit|NSMinuteCalendarUnit|NSSecondCalendarUnit)
        fromDate:dateNow];
       // Utilisation du modèle KVC pour affecter le modèle
       [self setValue:[NSNumber numberWithInt:composantsDeLaDateHMS.hour]
               forKeyPath:@"modele.heures"];
       [self setValue:[NSNumber numberWithInt:composantsDeLaDateHMS.minute]
              forKeyPath:@"modele.minutes"];
       [self setValue:[NSNumber numberWithInt:composantsDeLaDateHMS.second]
              forKeyPath:@"modele.secondes"];}
```

Si ce n'est pas déjà fait, aller jeter un coup d'œil au protocole **NSKeyValueObserving** dans l'aide, puis à la méthode **observeValueForKeyPath**. Cette méthode (ou ce message) est déclenchée si un changement survient sur une KVC abonnée à l'observation.

15. Copier le prototype de cette méthode depuis l'aide afin de l'ajouter à votre contrôleur de vue.
16. Coder cette méthode afin :
    a. qu'elle vérifie que la KVC est bien **modele.secondes**, et si c'est le cas,
    b. qu'elle lise les données heures, minutes et secondes du modèle,
    c. qu'elle affiche le résultat au format **HH :MM :SS** dans le label de la vue.

    Le code devrait ressembler à ceci :

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {
        if ([keyPath isEqualToString:@"modele.secondes"]) {
              NSNumber* h = [object valueForKeyPath:@"modele.heures"];
              NSNumber* m = [object valueForKeyPath:@"modele.minutes"];
              NSNumber* s = [object valueForKeyPath:@"modele.secondes"];
              self.monLabel.text=[NSString stringWithFormat:@"%02d:%02d:%02d",
              [h intValue],[m intValue],[s intValue]];
        }
  }
```

Et voilà, il ne reste plus qu'à construire et éventuellement à déboguer le programme puis vérifier qu'il n'existe aucune fuite mémoire.