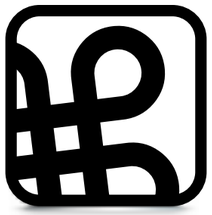


Macoun

Core Data - Clients syncen

Ingo Kasprzak und Peter Hauke

Ablauf

- Problemstellung
- Gewählter Lösungsansatz
- Code-Beispiele
- Demo

Einleitung

Problemstellung

- Abgleich von *NSManagedObjects* zwischen mehreren Geräten
- Datenunabhängigkeit
- Verzicht auf Server
- Minimalinvasive Implementation

Problemstellung
Datenabgleich

- Änderungen an *NSManagedObjects* instantan „verteilen“
- Anwendungen:
  - Mehrpersonenspiele (zugbasiert)
  - Datenaustausch zw. Geräten

Problemstellung
Datenunabhängigkeit

- ganz ohne Anpassungen geht‘s nicht:
  - Zeitstempel für jedes *NSManagedObject*
- zusätzliche Entity *Device*
  - uid
- Verwaltung gepairter Geräte
- Verwaltung gelöschter *NSManagedObjects*

Problemstellung

Verzicht auf Server

- Ad-hoc-Verbindung zwischen zwei Geräten
- Clients lösen Syncprozess automatisch aus
  - Auslöser sind Änderungen an den *NSManagedObjects*

Problemstellung

Minimalinvasive Implementation

- Einfache Integration in bestehende Projekte
- Möglichst wenige Änderungen an Controllern

Lösungsansatz

- Game Kit
- Ablaufgraph
- Konkrete Änderungen am *datamodel*

Lösungsansatz
Game Kit

- Game Kit:
  - Infrastruktur
  - Methoden zum Senden & Empfangen
- Demo: Local Game Kit

Lösungsansatz
Ablaufgraph

- Trigger: *NSManagedObjectContextDidSaveNotification*
- Sammeln der Änderungen
- Serialisieren der *NSManagedObjects* (via *NSDictionary*)
- Senden & Empfangen
- Update der *NSManagedObjects* auf Zielgerät
- ggfs. Empfangsbestätigung senden & auswerten

Lösungsansatz

Änderungen am *datamodel*

- Attributes für alle Entities
  - *creationDate*, *modificationDate*, *uid*
- Zusätzliche Entities
  - *Device*, *SynchronizationInfo*, *DeletionInfo*

Hauptteil

Hauptteil
Doch noch nicht ...

- Bequemes Fetchen
- Kategorien für Entities
- Verrührte Methoden

- *Matt Gallagher - http://cocoawithlove.com*

Convinient fetch methods

- Abstraktion von *NSFetchRequest*
- *CoreDatabase*-Klasse für Handling
  - *init, saveChanges, ...*
  - *convinient methods*
- *http://cocoawithlove.com/2008/03/core-data-one-line-fetch.html*

Convinient fetch methods

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];

NSEntityDescription *entity =
  [NSEntityDescription entityForName:@"Conference"
              inManagedObjectContext:managedObjectContext];
[request setEntity:entity];

NSPredicate *predicate =
  [NSPredicate predicateWithFormat:@"NAME == %@", @“Macoun“];
[request setPredicate:predicate];

NSError *error;
NSArray *array =
  [managedObjectContext executeFetchRequest:request error:&error];
```

Convinient fetch methods

```
NSSet *set =
  [self fetchObjectsForEntityClass:[Conference class]
                         withPredicate:@"NAME == %@", @“Macoun“];
```

Convinient fetch methods

```
-(id)fetchObjectForEntityClass:(Class)entityClass
                 withPredicate:(id)stringOrPredicate, ...;

-(NSSet *)fetchObjectsForEntityClass:(Class)entityClass
                       withPredicate:(id)stringOrPredicate, ...;

-(NSArray *)fetchObjectsForEntityClass:(Class)entityClass
                        sortDescriptor:(id)stringOrDescriptorOrArray
                         withPredicate:(id)stringOrPredicate, ...;

-(NSUInteger)countForEntityClass:(Class)entityClass
                   withPredicate:(id)stringOrPredicate, ...;

-(NSFetchRequest *)fetchRequestTemplateForName:(NSString *)name;
```

Category für Entities

Makes Core Data handling more simple

- *Create NSManagedObject Subclass* überschreibt .h/.m-Datei
- Typisches Vorgehen:
  - Manuelles Änderung der .h/.m-Datei (*attributes/relationships*)
  - Manuelles Sichern der eigenen Methoden
- Besser:
  - Categories

Category für Entities

makes core data handling simpler

- Nach *Create NSManagedObject Subclass*: Category erzeugen
  - Namensvorschlag: „*+Methods*“
- Eigenen Code nur in Category hinzufügen
- *#import <MyEntity+Methods.h>*

Category für Entities

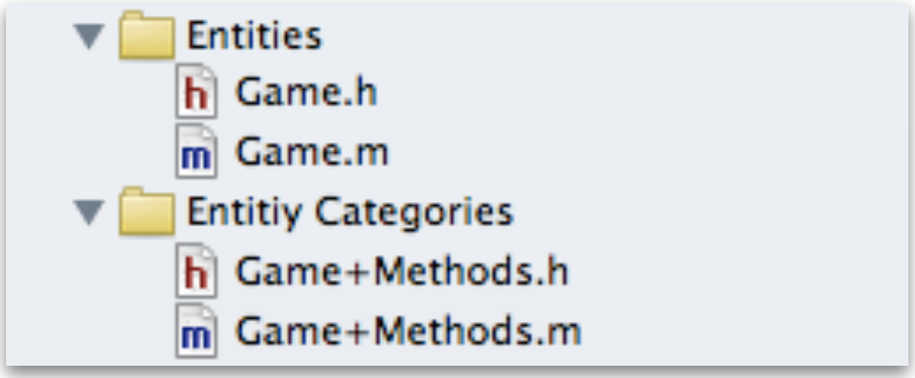
Game+Methods.m

```
//  Game+Methods.m
#import "Game+Methods.h"
#import "Move+Methods.h"

struct Combo {
    int field0; int field1; int field2;
};

@implementation Game (Methods)

- (NSSet*)checkForWinningCombo {
    ...
}
```



Method swizzling

- Problem: Datamodeler unterstützt nur Ableitung von *NSManagedObject*
  - z.B. eigener Code in *awakeFromInsert*
- Lösung: Griff in die obj-c run time Trickkiste

- Macoun 2011 *iOS 5 Appearance Customization* (Ortwin Gentz)

Method swizzling

```
+ (void) initialize {
   if ([self class] == [SyncDatabase class]) {

      Method awakeFromInsert_custom =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert_custom));

      Method awakeFromInsert_original =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert));

      method_exchangeImplementations
        (awakeFromInsert_original, awakeFromInsert_custom);
  }}
```

Method swizzling

```
+ (void) initialize {
   if ([self class] == [SyncDatabase class]) {

      Method awakeFromInsert_custom =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert_custom));

      Method awakeFromInsert_original =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert));

      method_exchangeImplementations
        (awakeFromInsert_original, awakeFromInsert_custom);
 }}
```

Method swizzling

```
+ (void) initialize {
   if ([self class] == [SyncDatabase class]) {

      Method awakeFromInsert_custom =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert_custom));

      Method awakeFromInsert_original =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert));

      method_exchangeImplementations
        (awakeFromInsert_original, awakeFromInsert_custom);
  }}
```

Method swizzling

```
+ (void) initialize {
   if ([self class] == [SyncDatabase class]) {

      Method awakeFromInsert_custom =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert_custom));

      Method awakeFromInsert_original =
        class_getInstanceMethod([NSManagedObject class],
        @selector(awakeFromInsert));

      method_exchangeImplementations
        (awakeFromInsert_original, awakeFromInsert_custom);
  }}
```

Hauptteil

Jetzt aber ...

- Codebeispiele
- Demo

Hauptteil
Ablaufgraph

- Trigger: *NSManagedObjectContextDidSaveNotification*
- Sammeln der Änderungen
- Serialisieren der *NSManagedObjects* (via *NSDictionary*)
- Senden & Empfangen
- Update der *NSManagedObjects* auf Zielgerät
- ggfs. Empfangsbestätigung senden & auswerten

Trigger: *NSManagedObjectContextDidSaveNotification*

```
[[NSNotificationCenter defaultCenter]
  addObserver:self
     selector:@selector(managedObjectContextDidSave:)
         name:NSManagedObjectContextDidSaveNotification
       object:[[TicTacToeDatabase sharedTicTacToeDatabase]
                  managedObjectContext]];
```

```
- (void)managedObjectContextDidSave:(NSNotification*)notification {
   [self synchronizeDeviceBidirectional:NO];
}
```

Sammeln der Änderungen I

```
for (NSEntityDescription *entity in [self.managedObjectModel entities]) {
  if (![entity.managedObjectClassName isEqualToString:
        NSStringFromClass([SynchronizationInfo class])])
  {
    NSSet *set = [self fetchObjectsForEntityClass:NSClassFromString
    (entity.managedObjectClassName)
                  withPredicate: @"%K>=%@ AND %K!=%@",
                        ManagedObjectModificationDateKey, lastSyncDate,
                        ManagedObjectModificationDeviceUid, self.remoteDeviceUID];

   [set enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
      NSDictionary *propDict = [obj propertyDictionary];
       [modifiedObjectsSet addObject:propDict];   }];
  }
}
```

Sammeln der Änderungen 2

```
- (NSDictionary *)propertyDictionary {
   // Alle Attribute der Entität holen
   NSDictionary *source = self.entity.attributesByName;

   // Attribute auslesen und in ein Dictionary schreiben
   NSMutableDictionary *destination =
      [NSMutableDictionary dictionaryWithCapacity:[source count]];

   [source enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
      id value = [self valueForKey:key];
      if (!value) { value = [NSNull null]; }
      [destination setObject:value forKey:key];
   }];

...
```

Sammeln der Änderungen 3

```
// Alle Relationships der Entität holen
 source = self.entity.relationshipsByName;

 [source enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if (![obj isToMany]) {
       id value = nil;
       if ([self valueForKey:key]) {
          value = [self valueForKeyPath:
           [NSString stringWithFormat:@"%@.%@", key, ManagedObjectUIDKey]];
       } else {
          value = [NSNull null];
       }
       [destination setObject:value forKey:key];
    }
 }];
```

Sammeln der Änderungen 4

```
    // Den Namen der Klasse dieses Objects ebenfalls ins Dictionary schreiben
    [destination setObject:NSStringFromClass([self class])
                    forKey:PropertyDictionaryEntityNameKey];
    return destination;
}
```

Serialisieren der *NSManagedObjects*

```
NSDictionary *dict = @{
   SyncDatabaseSyncCommandKey     : command,    // Data, SyncRequest, Receipt
   SyncDatabaseSyncSenderUIDKey   : self.thisDevice.uid,
   SyncDatabaseSyncDataUIDKey     : myUcid,     // eindeutige Datensatz-ID
   SyncDatabaseSyncDateKey        : [NSDate date],
   SyncDatabaseSyncWantsReceiptKey  : @(yesOrNo),
   SyncDatabaseSyncDataKey        : modifiedObjectsSet // eigentlicher Datensatz
};

NSData *dataArchive = [NSKeyedArchiver archivedDataWithRootObject:dict];
```

Senden

```
dispatch_after
  (dispatch_time(DISPATCH_TIME_NOW, GAMEKIT_EXECUTION_DELAY * NSEC_PER_SEC),
   dispatch_get_main_queue(),^(void){

    [self.gkSession sendData:dataArchive
                     toPeers:@[peerID]
                withDataMode:GKSendDataReliable
                       error:nil];

   });
```

Empfangen

```
NSDictionary *dict = [NSKeyedUnarchiver unarchiveObjectWithData:data];

NSString *command = [dict objectForKey:SyncDatabaseSyncCommandKey];
NSMutableSet *modifiedObjectsSet = [dict objectForKey:SyncDatabaseSyncDataKey];
```

Update der *NSManagedObjects* I

```
NSDictionary *dict = [NSKeyedUnarchiver unarchiveObjectWithData:data];

NSString *command = [dict objectForKey:SyncDatabaseSyncCommandKey];
NSMutableSet *modifiedObjectsSet = [dict objectForKey:SyncDatabaseSyncDataKey];

if ([command isEqualToString:SyncDatabaseSyncSynchronizeCommand]) {
    [self receivedModifiedObjects:modifiedObjectsSet];
}
```

Update der *NSManagedObjects* 2

```
- (void)receivedModifiedObjects:(NSSet*)objects {
  ...

  [objects enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
      [self updateDatabaseWithPropertyDictionary:obj];
   }];

  ...
}
```

Update der *NSManagedObjects* 3

```
-(NSManagedObject*)updateDatabaseWithPropertyDictionary:
    (NSDictionary*)propertyDictionary {

  NSString *entityName = [propertyDictionary
                            objectForKey:PropertyDictionaryEntityNameKey];
  NSString *uid        = [propertyDictionary
                            objectForKey:ManagedObjectUIDKey];

...
```

Update der *NSManagedObjects* 4

```
...
 // prüfen, ob ManagedObject bereits vorhanden ist
 NSSet *set = [self
                fetchObjectsForEntityClass:NSClassFromString(entityName)
                withPredicate:@"%K = %@", ManagedObjectUIDKey, uid];

 NSManagedObject *object = nil;
 if (![set count]) {
  object = [NSEntityDescription
                insertNewObjectForEntityForName:entityName
                inManagedObjectContext:self.managedObjectContext];
 } else {
   object = [set anyObject];
 }
...
```

Update der *NSManagedObjects* 5

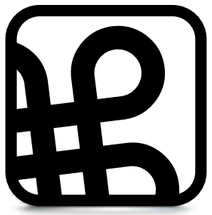
```
// don‘t mess with Voodoo
```

Demo

Abschluss

Fragen?

Vielen Dank



Macoun