Core Data and Web Services

Sitting in a tree

Conrad Stoll
Architect
mutualmobile

Mutual Mobile

The mobile solutions company.

Agenda

What are we doing here?

- What is Data Integration?
- Using Core Data.
- Challenges
- Object Oriented Design for Web Requests.
- A Better Way to Write Web Requests!
- Demo!
- Questions?

What is Data Integration?

What is Data Integration

Where does my data come from?

foursquare

Google

salesforce

APP.NET

heroku

Parse

What is Data Integration

That thing you have to do before you can build great apps

It’s the reason our apps work, even though it’s not why people use our apps.

- Data has to come from somewhere.
- View Controllers shouldn’t care from where.
- Needs to be performant and maintainable.

Challenges with Data Integration

Using Web Services

It’s a scary world out there.

- We don’t control our web services.
- Web services aren’t always finished.
- The only constant is change.
- APIs are always different.
- Time consuming and repetitive.

Challenges with Data Integration

It takes way too much time.

Time we spend integrating with web services is time we can’t spend delighting our users.

Using Core Data

Why use Core Data

Unless you're using iCloud

The object graph is a key part of our applications and using a mature tool to maintain it is a good investment.

- Core Data is well built and well maintained.
- Model versioning is hard to build.
- Change control over each of our objects.
- Greater control over sets of objects.
- Several useful features.
- Faulting gives us a low memory footprint.

Challenges with Core Data

Other than using iCloud

Core Data is a different way of thinking from both relational databases and vanilla object graphing.

- Threading.
- Steep learning curve.
- Core Data is very verbose.
- Significant performance gotchas.

Why this problem is worth solving

Core Data and Data Integration: a match made in heaven?

Core Data and Data Integration are both hard.

- Everyone has to do this.
- Users don’t care how we solve this problem.
- Makes it easier to provide user value.
- Our apps need to be easy to maintain.
- We shouldn’t spend all our time here.

Object Oriented Design for Web Requests

Effectively using GET Requests

What the heck is JSON?

GET requests are usually the workhorse of data integration on most apps.

- Most APIs treat GET the same way.
- View Controllers shouldn’t care about GET.
- A response comes back with a JSON string.
- View Controllers shouldn’t care about JSON.

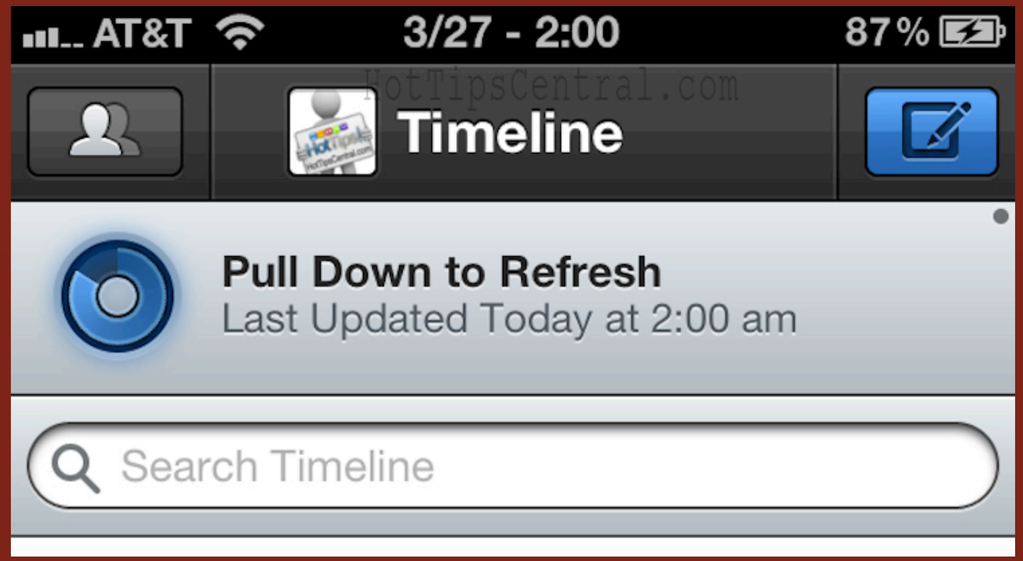
Effectively using GET Requests

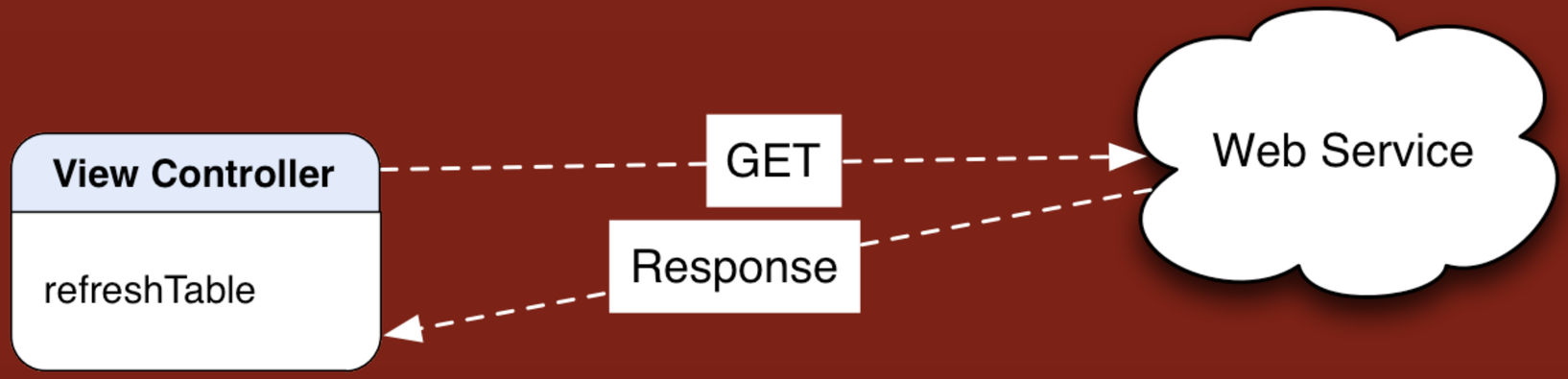
Can I just have my data please?

The goal is to get from response data to native objects as easily and as fast as possible.

NOT Effectively using GET Requests

Refreshing a table view





OOD for Web Requests

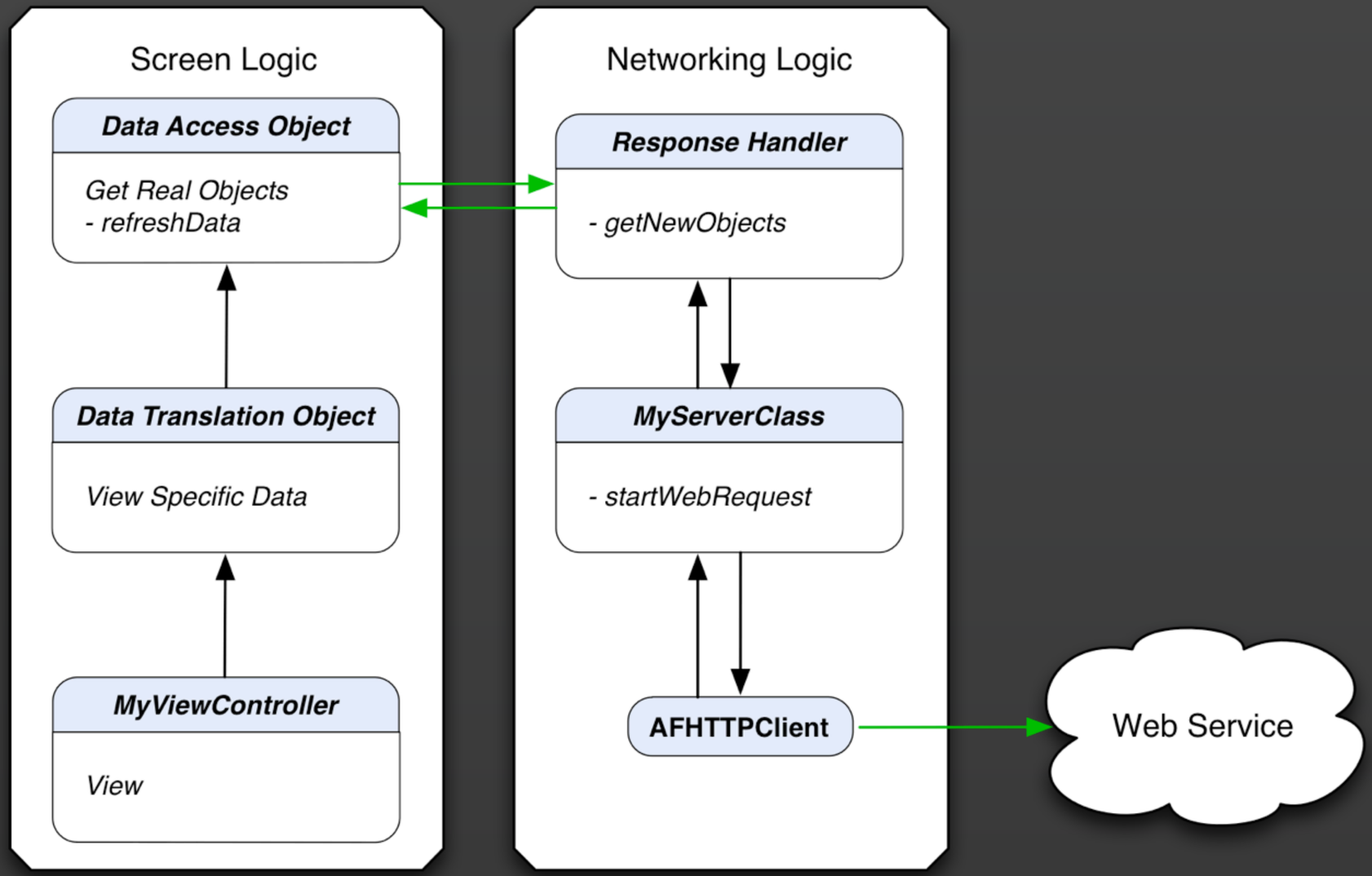
Wait, this doesn't just all go in the View Controller??!?

Let **SOLID** be our guide.

- Single Responsibility Principle.
- View Controllers **don't** do networking.
- A block based networking is very clean.
- Use Data Access Objects (DAOs).
- AFNetworking is a great tool to use.
- But use it wisely.

OOD for Web Requests

Ok so where does this all go?



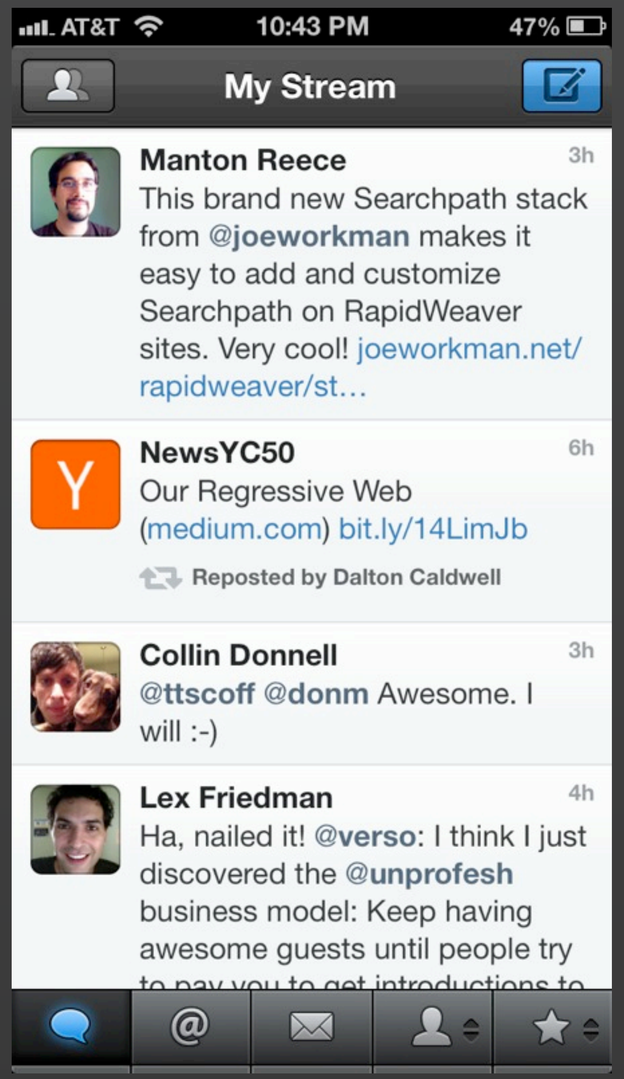
OOD for Web Requests

Why do it this way?

Well designed software is easier to change and easier to maintain.

OOD for Web Requests

For the users!



That’s great, but ...

... wouldn’t it be nice if something did this for us?

Introducing MMRecord

One way for making Data Integration simpler and easier.

MMRecord

https://github.com/mutualmobile/MMRecord

Introducing MMRecord

One way for making Data Integration simpler and easier.

MMRecord is a simple library for integrating with web services.

- Fully integrated with Core Data.
- No parsing code required.
- Block based API for data access.
- Networking library agnostic.

Design Goals

Design Goals

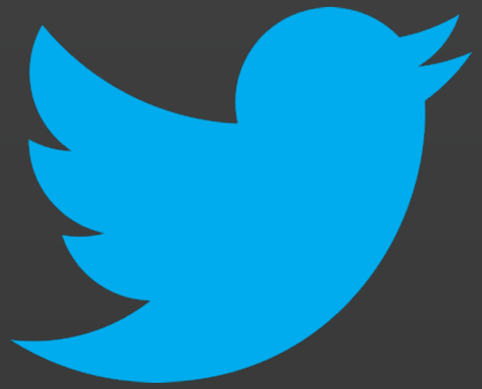
Make Data Integration simpler and easier.

Implementing new web requests should be as simple as possible.

- Easy local/live server switching.
- Automatic response handling.
- Integration with Core Data.
- Support for common web service paradigms.
- Simple and light weight.
- High Performance.

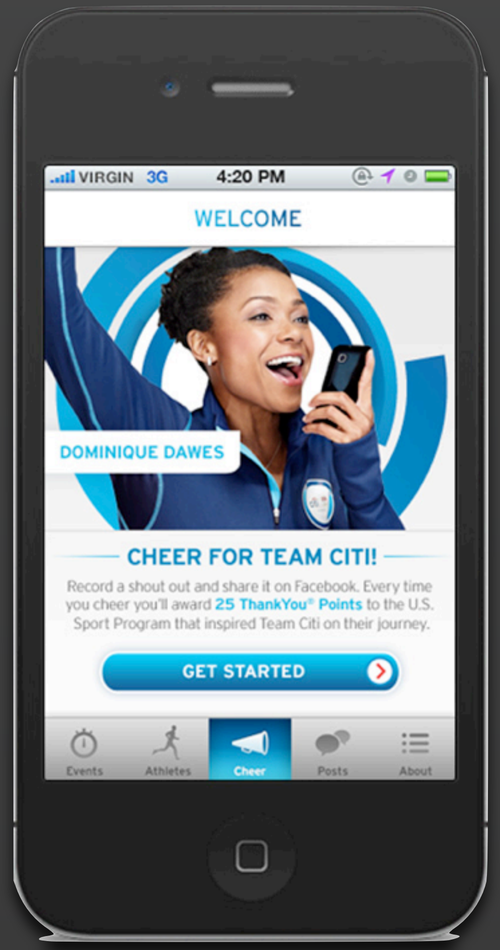
Tested Web Services

Compatibility with a wide range of response formats.



Easy Integration

Olympic turn around time.



Design Philosophy

MMRecord is NOT a REST framework!

Our goal was NOT to build a REST framework.

- Do one thing and do it well.
- MMRecord obtains data. That's it.
- MMRecord doesn't try to do it all.
- Data retrieval is a repeatable process.
- Syncing and remote update is not.

Features

Take Something and Make it Better

Useful features and functionality.

Let’s make various paradigms for performing web requests even easier!

- Easily switch network environments.
- Easy to use custom response handling.
- Clean and simple interface for pagination.
- Pre-fetching.
- Summary/detail requests.
- No-hassle block based API for batching.

Batching

Making a batched request with MMRecord

```
NSManagedObjectContext *context = [self managedObjectContext];

[Tweet startBatchedRequestsInExecutionBlock:^{
    [Tweet timelineTweetsWithContext:context
                              domain:self
                         resultBlock:resultBlockHandler
                        failureBlock:failureBlockHandler];

    [Tweet favoriteTweetsWithContext:context
                              domain:self
                         resultBlock:resultBlockHandler
                        failureBlock:failureBlockHandler];
} withCompletionBlock:^{
    NSLog(@"Finished retrieving timeline and favorite tweets");
}];
```

Pagination

Making a paged request with MMRecord

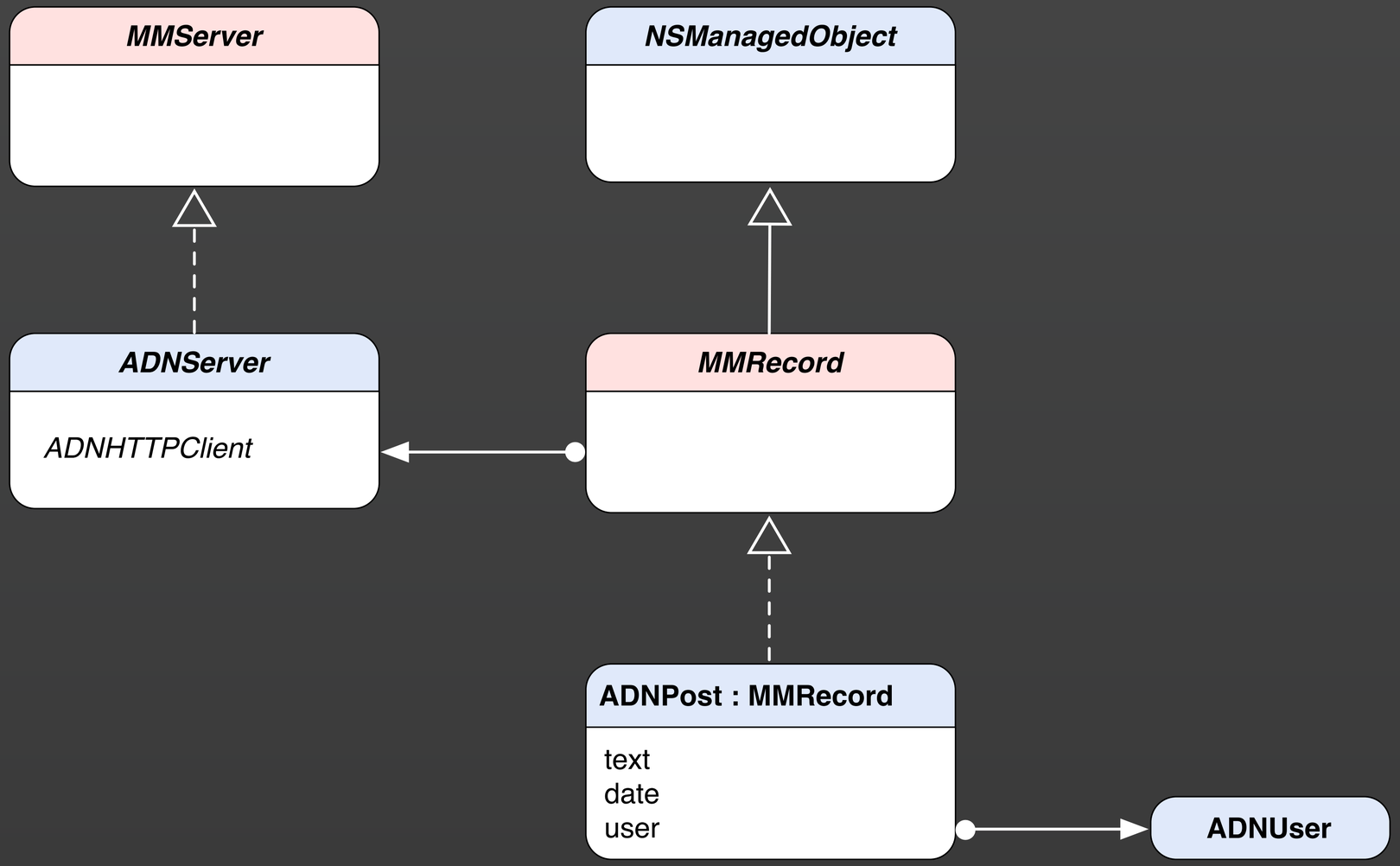
```
- (void)getPosts {
    [Post
     getStreamPostsWithContext:self.managedObjectContext
     domain:self
     resultBlock:^(NSArray *posts, ADNPageManager *pageManager, BOOL *) {
         [self populatePostsTableWithPosts:posts];
         self.pageManager = pageManager;
     }
     failureBlock:^(NSError *error) {
         [self endRequestingPosts];
     }];
}

- (void)getPreviousPosts {
    [self.pageManager getPreviousPosts:^(NSArray *posts) {
        [self populatePostsTableWithPosts:posts];
    }];
}
```

Class Structure

MMRecord Class Structure

Map of the Universe



MMServer

Common interface for making a request for data

MMServer is a plug and play provider of data from any source.

- Common interface for starting a request.
- Takes a URN and Parameters.
- Is **NOT** a networking framework!!!
- **NOT** responsible for parsing the response.
- Only needs to return an array or dictionary.

MMServer

Simple to implement, easy to use.

Servers can be swapped in and out for different web services and networking environments.

- MMServer is designed to be subclassed.
- Register a MMServer class with MMRecord.
- Plug and play network environments.
- Use any networking library you want!
- Use local or fake server.
- Helps keep MMRecord light weight.

MMServer

Example Implementations.

Reference server implementation that uses AFNetworking. Includes support for batching and request cancellation.

```
@interface MMAFJSONServer : MMServer

/**
 This method can be used to register an instance of an AFHTTPClient.

 @param client The client this server should use to make requests.
 @return YES if the client was successfully registered. No otherwise.
 */
+ (BOOL)registerAFHTTPClient:(AFHTTPClient *)client;

@end
```

MMRecord

The hub for accessing data and returning instances of MMRecord subclasses.

MMRecord is a design pattern for data integration.

- Entry point for making requests.
- Class methods for starting requests.
- Return objects of requested type.
- Requires an MMServer class.
- Parsing is handled in other classes.

MMRecord

That's cool, but how does this work?

The Managed Object Model is the center of the known universe.

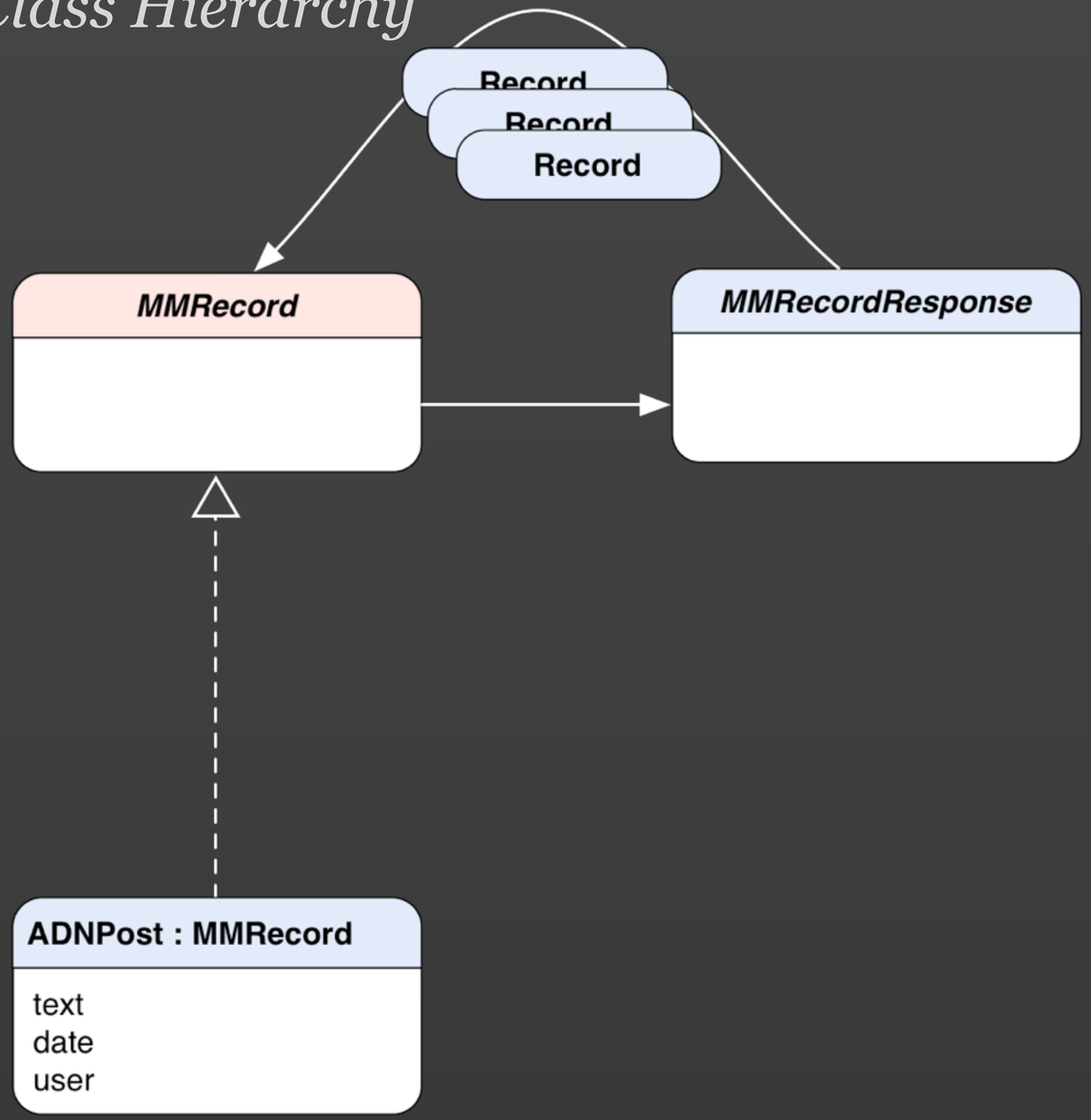
- Mapping data is contained in the model.
- Initial Entity.
- Looks at each attribute/relationship.
- Recursively populates relationship entities.
- Opportunities to customize behavior.

MMRecord Response Handling

What does our response handling pipeline look like?

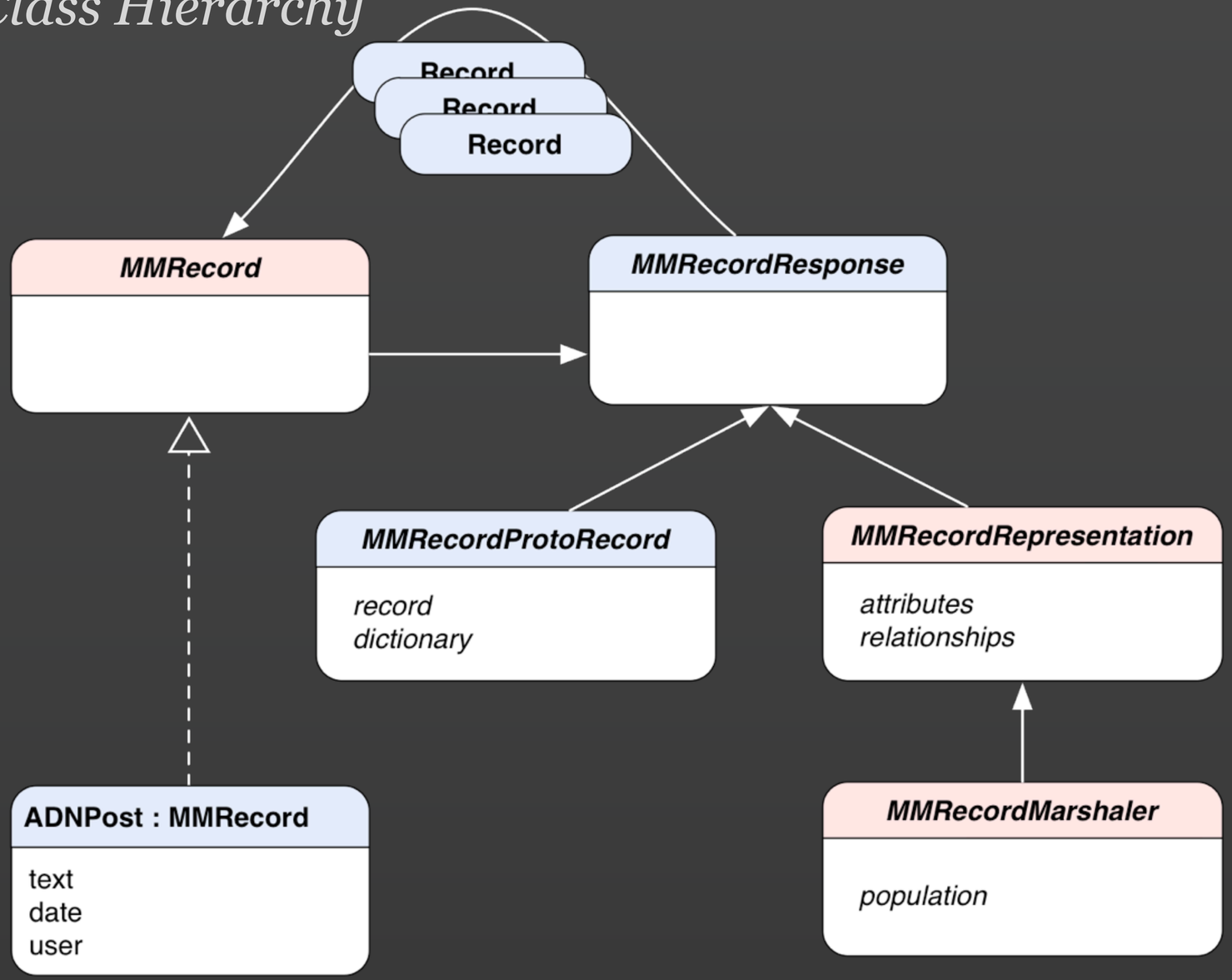
MMRecord Response Handling

Response Handling Class Hierarchy



MMRecord Response Handling

Response Handling Class Hierarchy



Configuration

MMRecord

Model Configuration

Some User Info dictionary setup required.

- Unique Identification.
- Use **MMRecordEntityPrimaryAttributeKey**.
- Populates attributes by name automatically.
- Supports alternate names as well.
- Use **MMRecordAttributeAlternateNameKey**.
- Supports date and transformable attributes.

MMRecord

Subclassing MMRecord

MMRecord is designed to be subclassed.

- One subclass per entity.
- Set the Managed Object Class name.
- One required method.
- Several optional methods.

MMRecord

Subclassing MMRecord

```objc
static NSDateFormatter *ADNRecordDateFormatter;

@implementation ADNRecord

+ (NSString *)keyPathForResponseObject {
    return @"data";
}

+ (NSDateFormatter *)dateFormatter {
    if (!ADNRecordDateFormatter) {
        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat:@"yyyy-MM-dd'T'HH:mm:ssZ"];
        ADNRecordDateFormatter = dateFormatter;
    }

    return ADNRecordDateFormatter;
}

@end
```

MMRecord

Making a request with MMRecord

```
+ (void)getPostsForUser:(User *)user
                context:(NSManagedObjectContext *)context
                 domain:(id)domain
            resultBlock:(void (^)(NSArray *, ADNPageManager *, BOOL *))resultBlock
           failureBlock:(void (^)(NSError *))failureBlock {
    NSString *URN = [NSString stringWithFormat:@"stream/0/users/%@/posts", user.id];

    [self startPagedRequestWithURN:URN
                              data:nil
                           context:context
                            domain:self
                       resultBlock:resultBlock
                      failureBlock:failureBlock];
}
```

Demo!

Real life coding!

Questions?

But what about AFIncrementalStore?!?!?

Thanks!!

Go forth and integrate!

Conrad Stoll

@conradstoll - cnstoll@me.com

mutualmobile