



**Macoun'10**

# Datenmigration mit Core Data

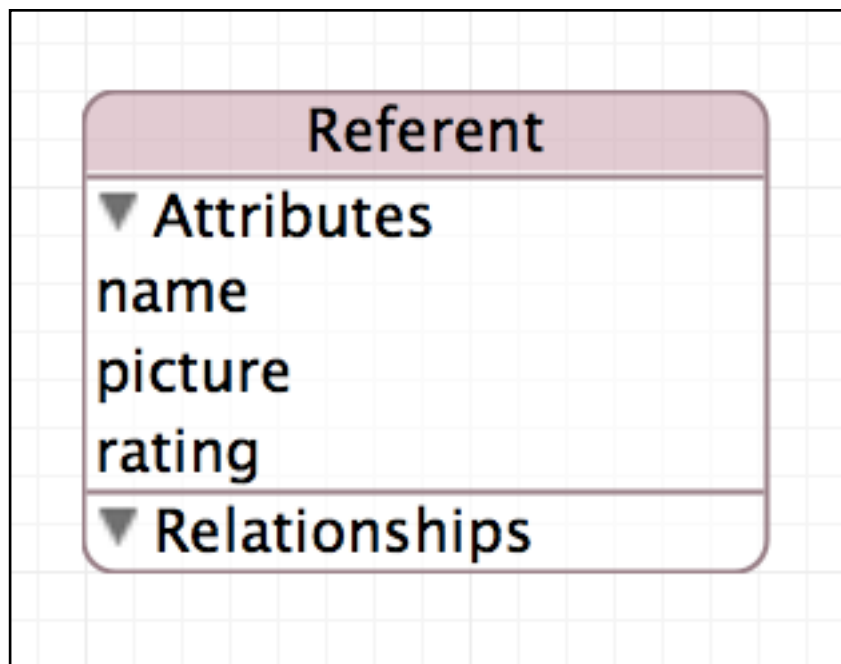
[co coa:ding]

Christian Kienle

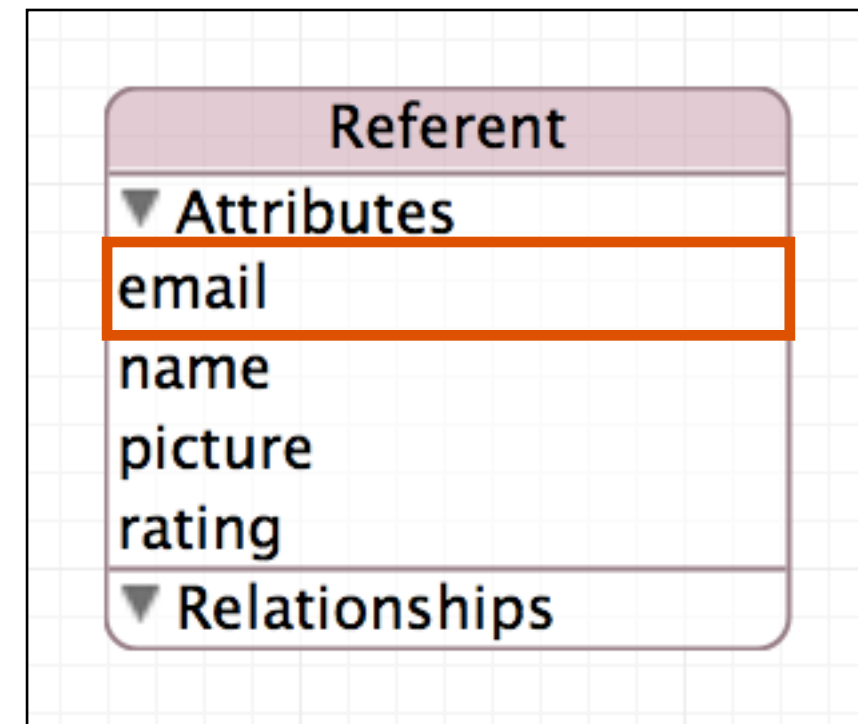
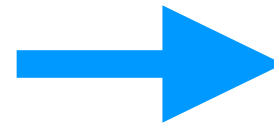
# Motivation

- Kompatibilität
- Unterschiedliche Formate: Nicht das Problem der Benutzer
- Viele Entwickler nutzen Core Data

# Problembeschreibung

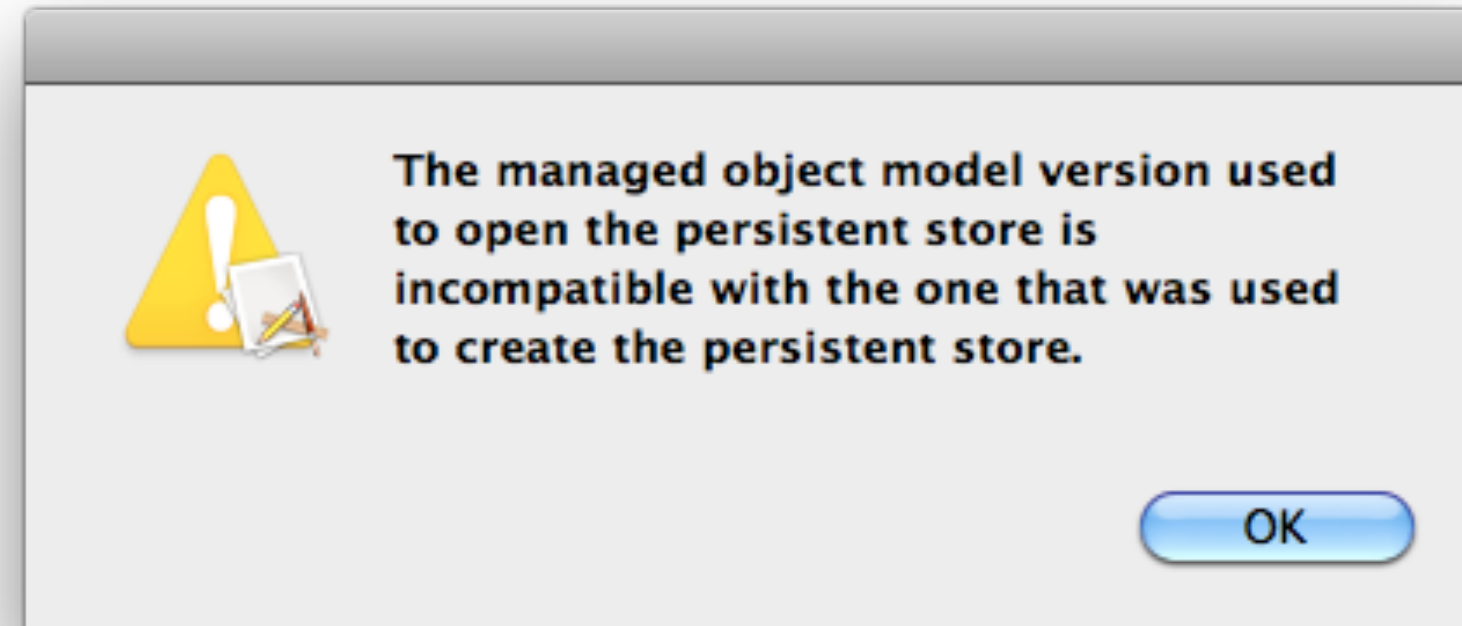


Version 1.0



Version 2.0

# Problembeschreibung



# Lösungen

- Migration komplett selbst durchführen: Mac OS X 10.4
- Core Data zu Hilfe nehmen:  $\geq$  Mac OS X 10.5, iOS 3.0

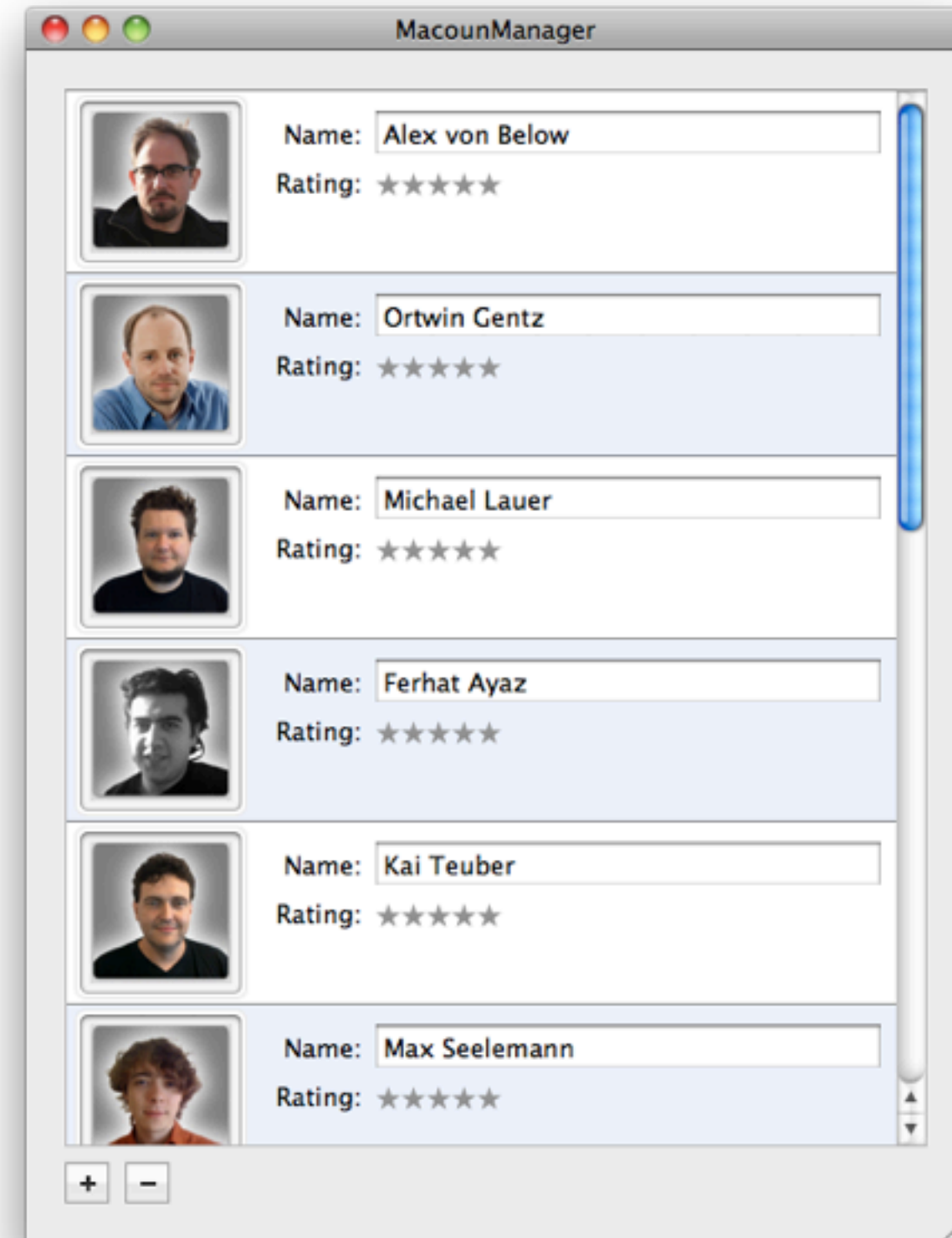
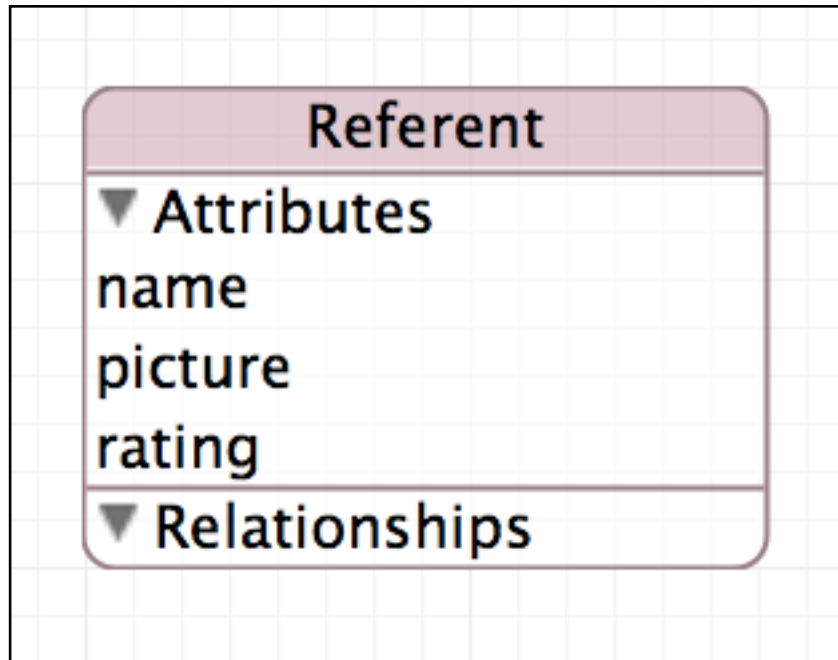
*Common things are easy, not so  
common things are possible.*

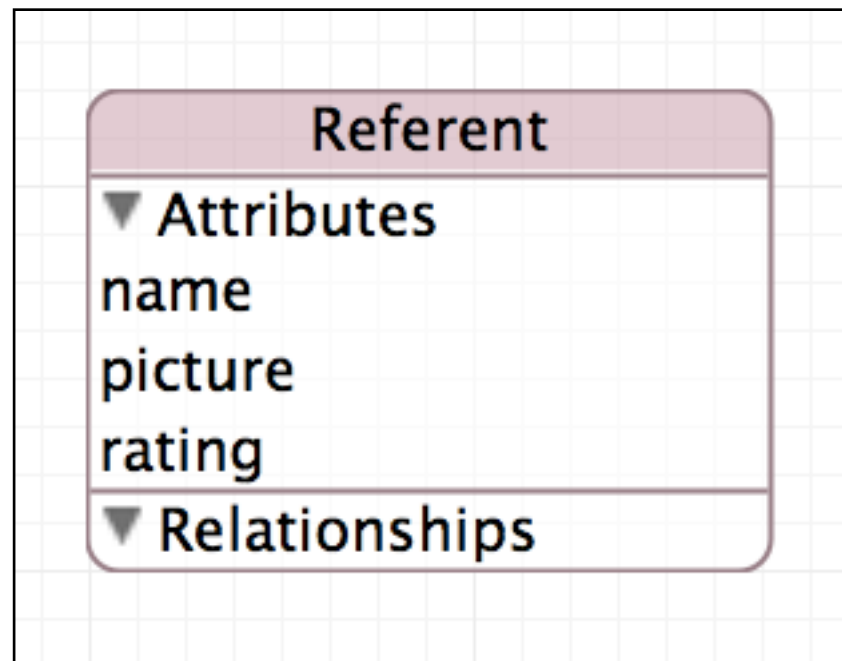
Aaron Hillegass

# Prinzip

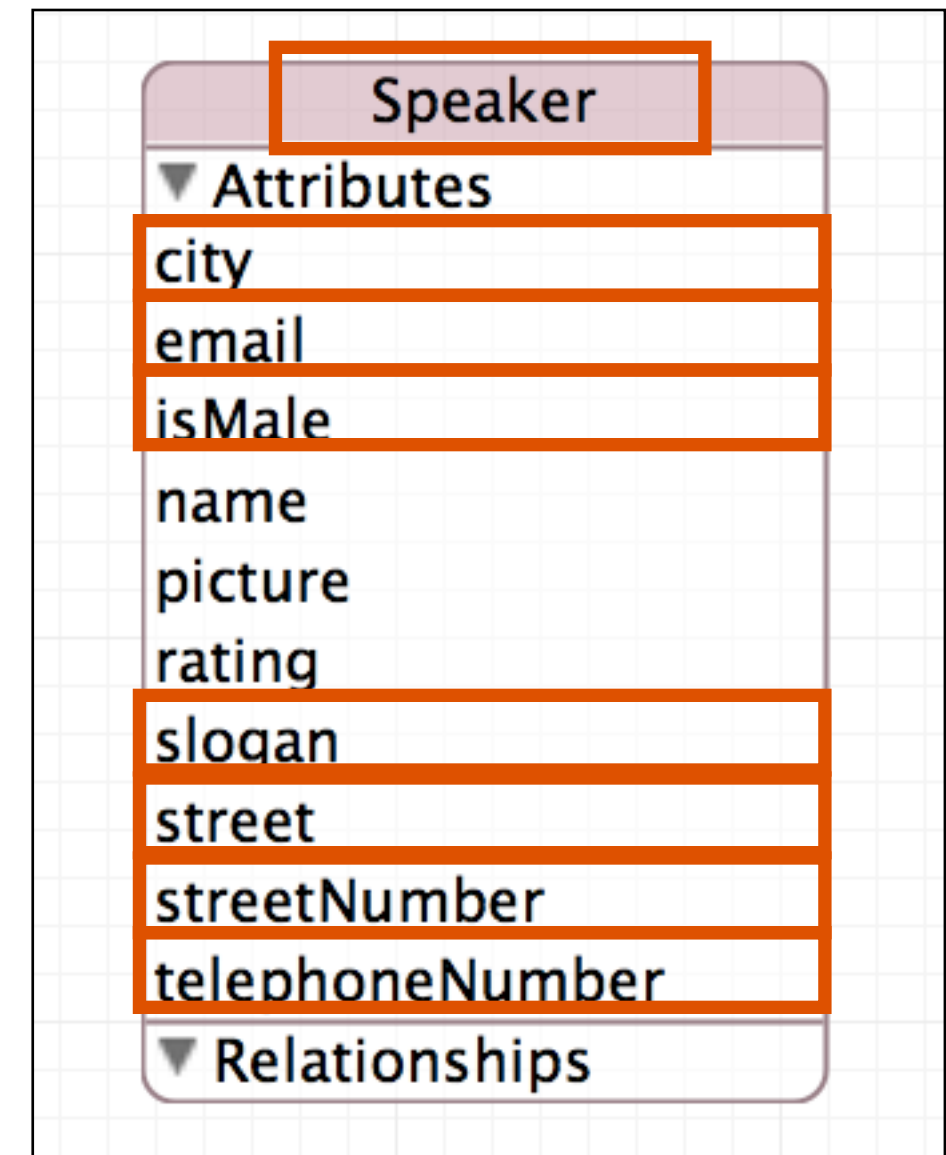
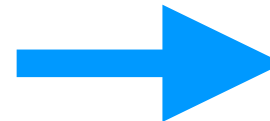
- mehrere Models
- ein “neustes” Model
- veralteter Store wird in die aktuelle Form gebracht







Version 1



Version 2

Demo

# Mögliche Änderungen

- Hinzufügen eines Attributes
- Umbenennen eines Attributes
- Optionales Attribute obligatorisch machen
- Obligatorisches Attribut optional machen
- Umbenennen einer Entität

# Ergebnis

- Versionskonflikt wird automatisch erkannt
- Notwendige Migrationsschritte werden abgeleitet
- Migration wird durchgeführt

# Vorbereitung

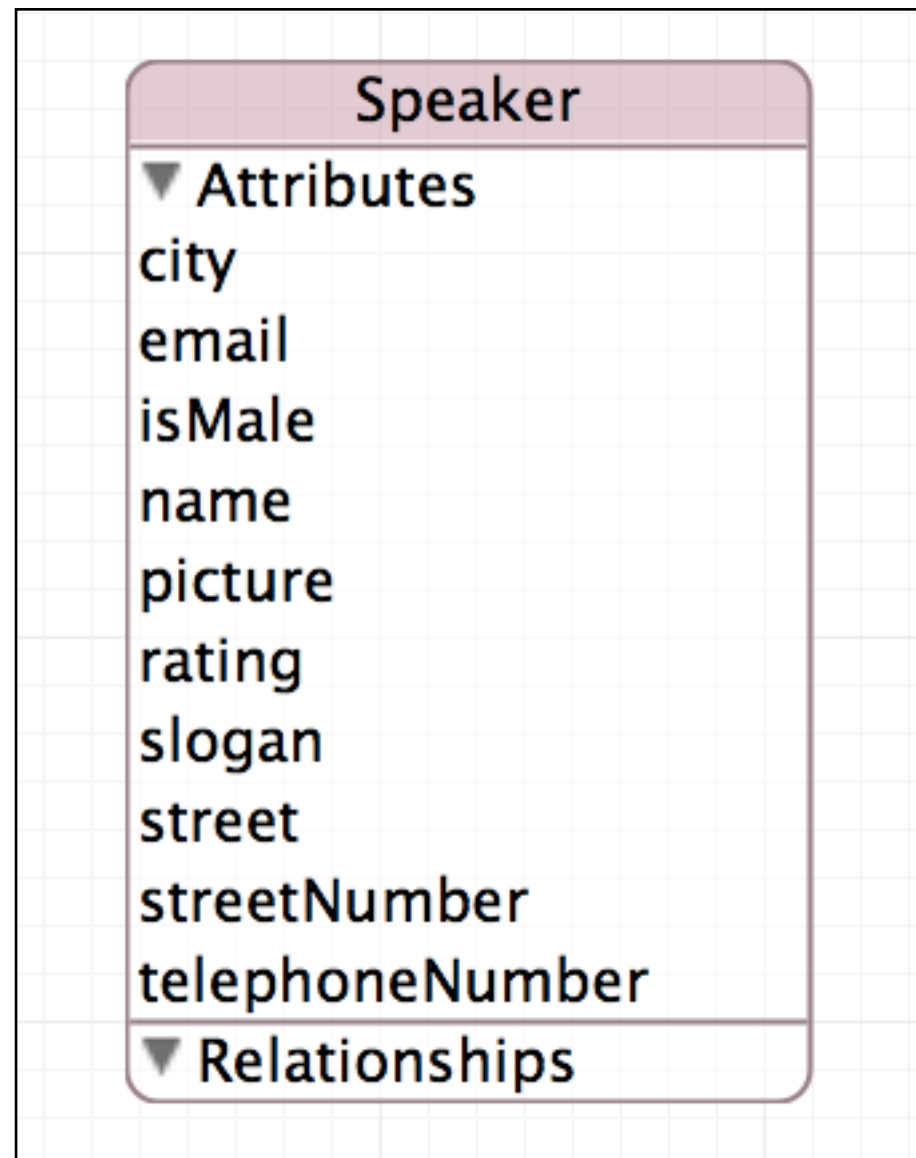
```
NSMutableDictionary *options = [NSMutableDictionary dictionary];

NSNumber *yes = [NSNumber numberWithBool:YES];

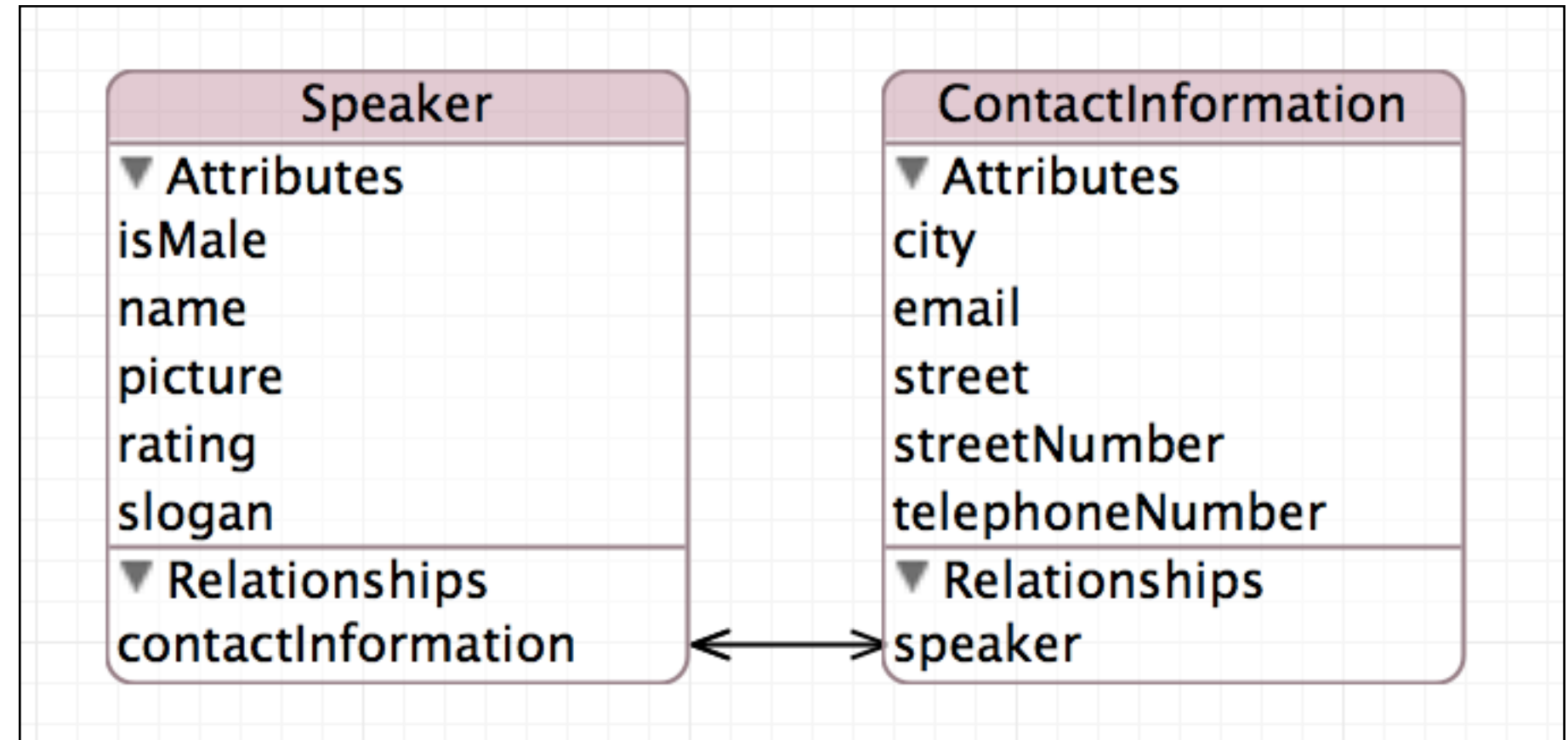
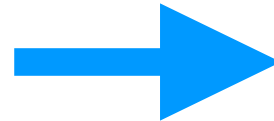
[options setObject:yes forKey:NSMigratePersistentStoresAutomaticallyOption];
[options setObject:yes forKey:NSInferMappingModelAutomaticallyOption];

[persistentStoreCoordinator addPersistentStoreWithType:NSXMLStoreType
                                   configuration:nil
                                   URL:url
                                   options:options
                                   error:&error];
```

Dies war die sog. *Lightweight  
Migration*

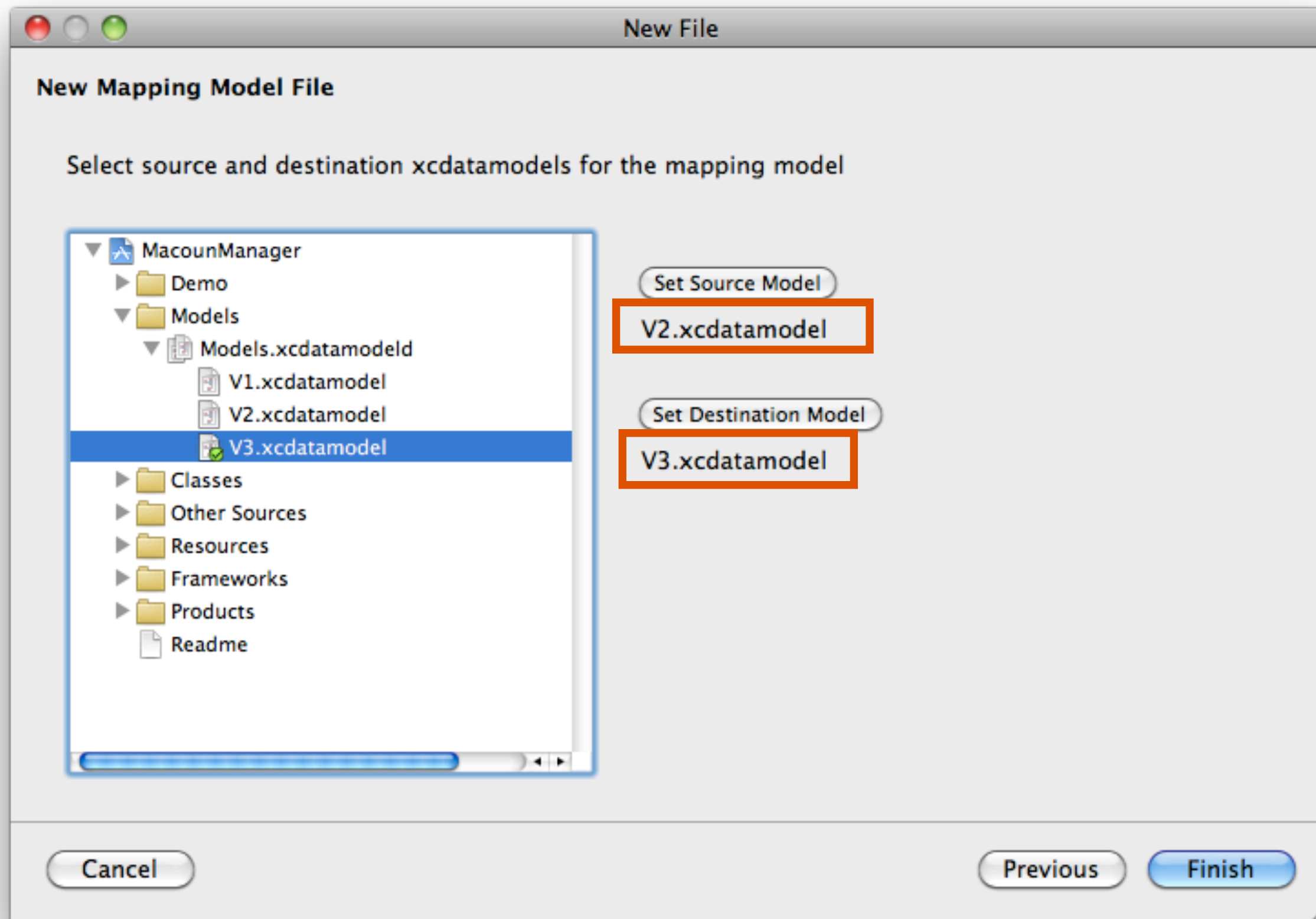


Version 2



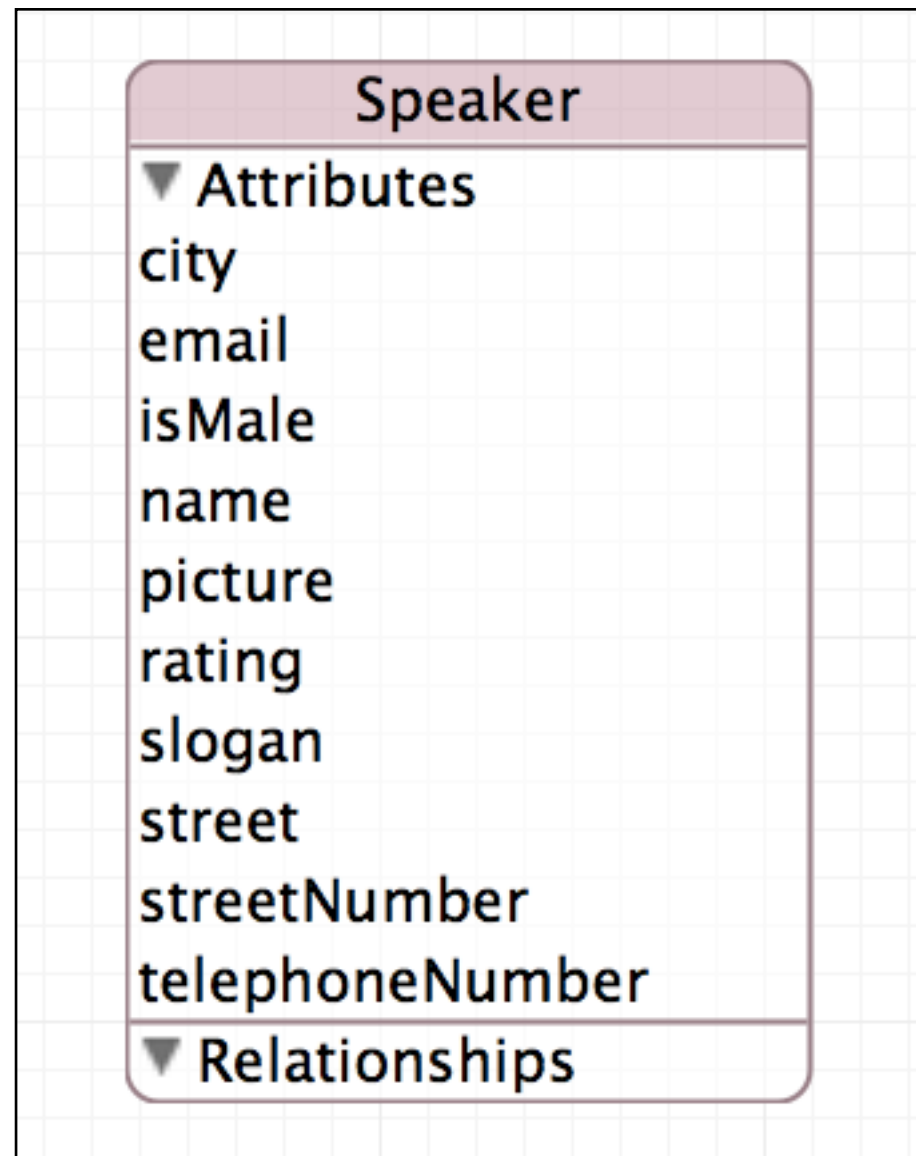
Version 3



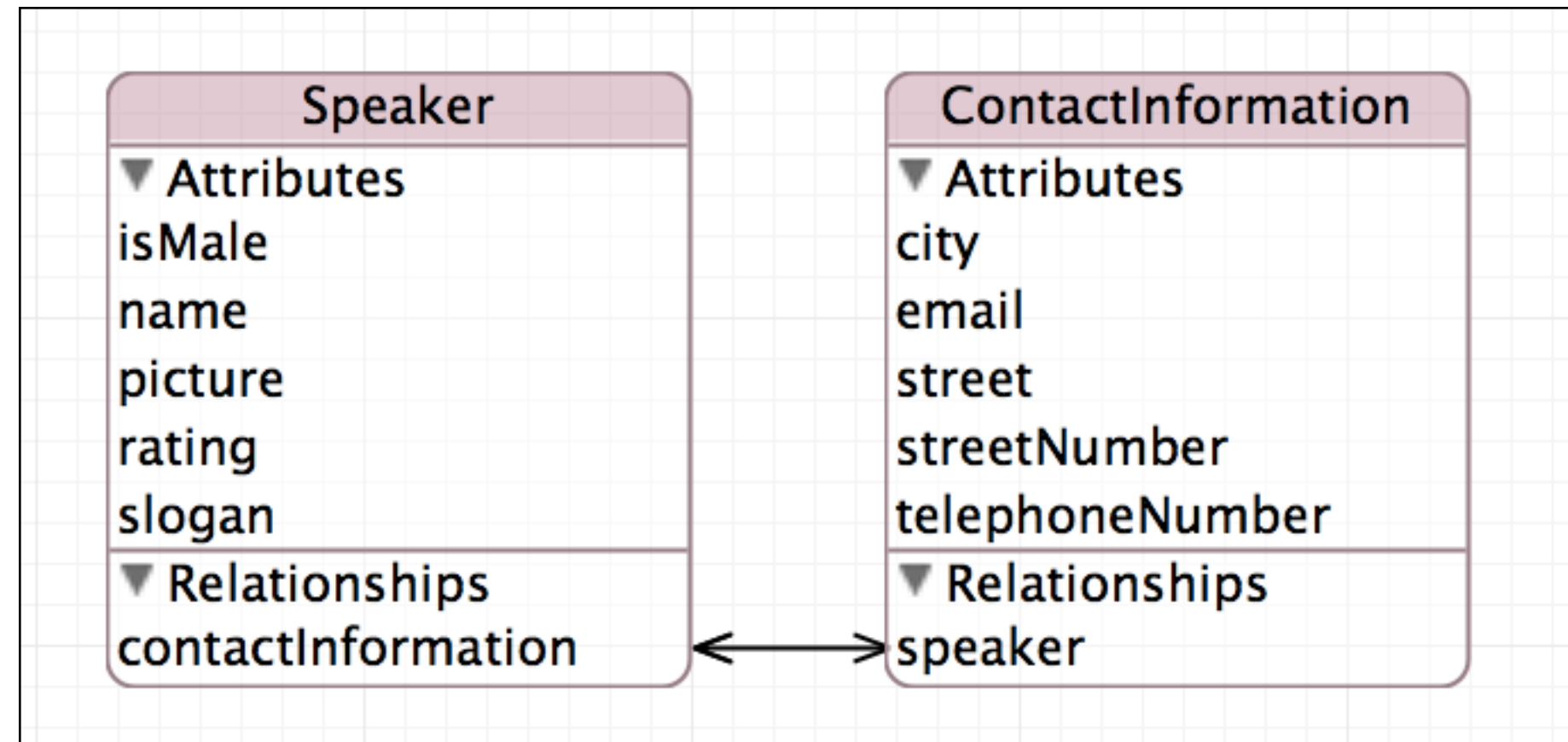
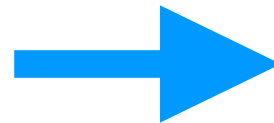


# Mapping Model

- Von Source Model nach Destination Model
- Editor eingebaut in Xcode
- Wichtig: Erst Zielmodel dann Mapping Model!



Version 2



Version 3

Demo

# Ablauf einer Migration

(Three-Stage Migration)

- Zielinstanzen werden erzeugt, Werte gesetzt
- Beziehungen werden hergestellt
- Validierung

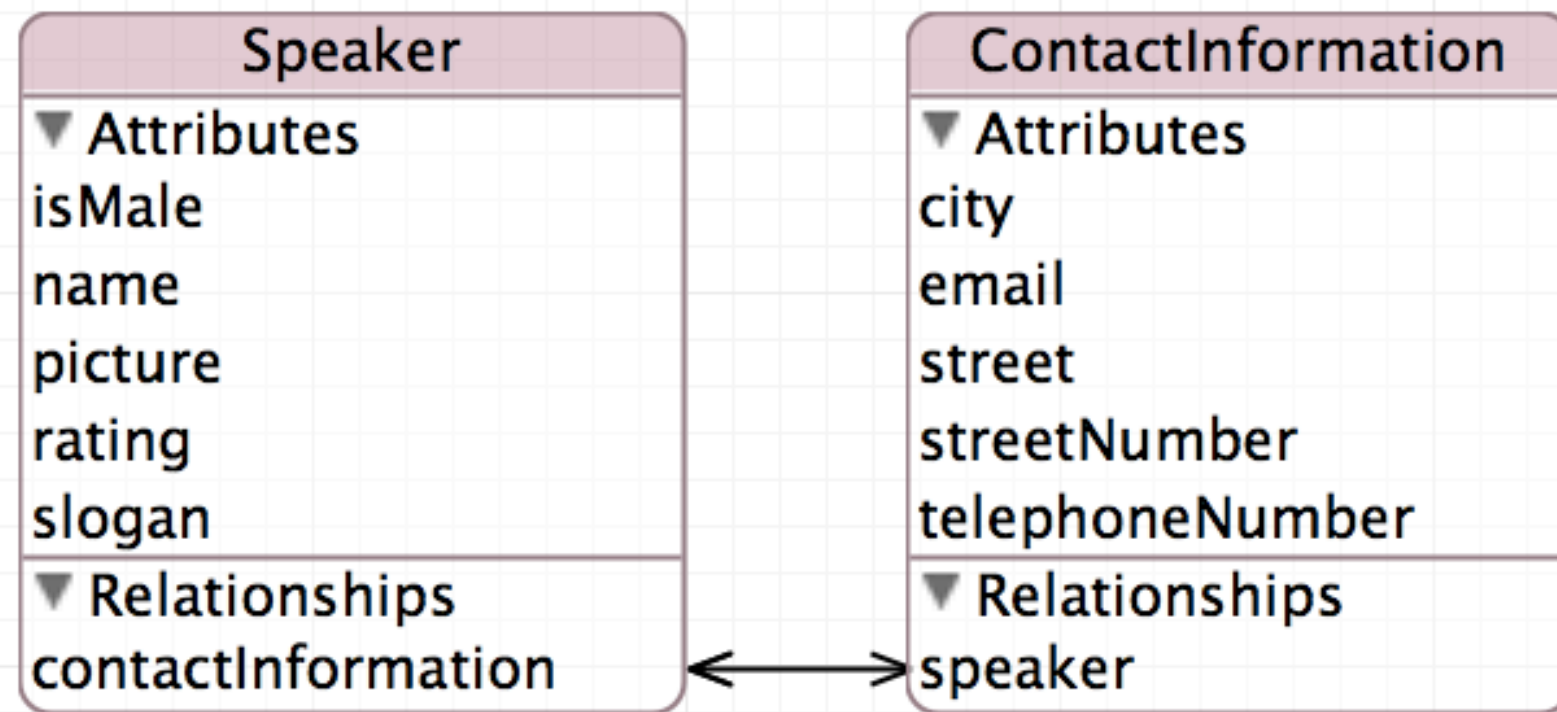
# Notwendigkeit einer Migration

- Pro Entität ein Hash
- Name der Entität, Parent Entität, Existenz der persistenten Eigenschaften, Namen der Eigenschaften, ...
- Passiert zum Glück automatisch

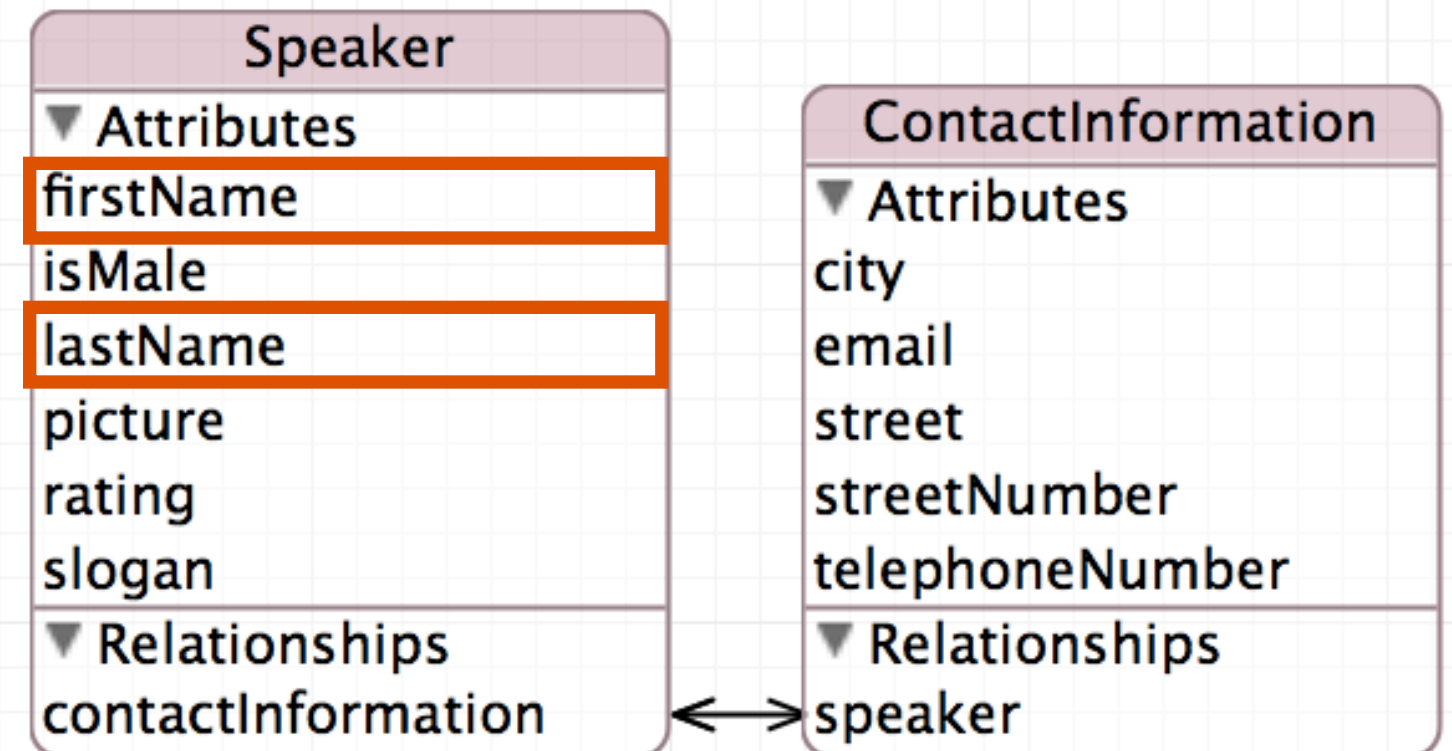
# Entity Mappings

NSEntityMigrationPolicy

Entity Mappings					
# ▲	Mapping Name	Source	H	Destination	
1	SpeakerToSpeaker	Speaker	*	Speaker	
2	SpeakerToContactInformation	Speaker	*	ContactInformation	



Version 4



Version 5



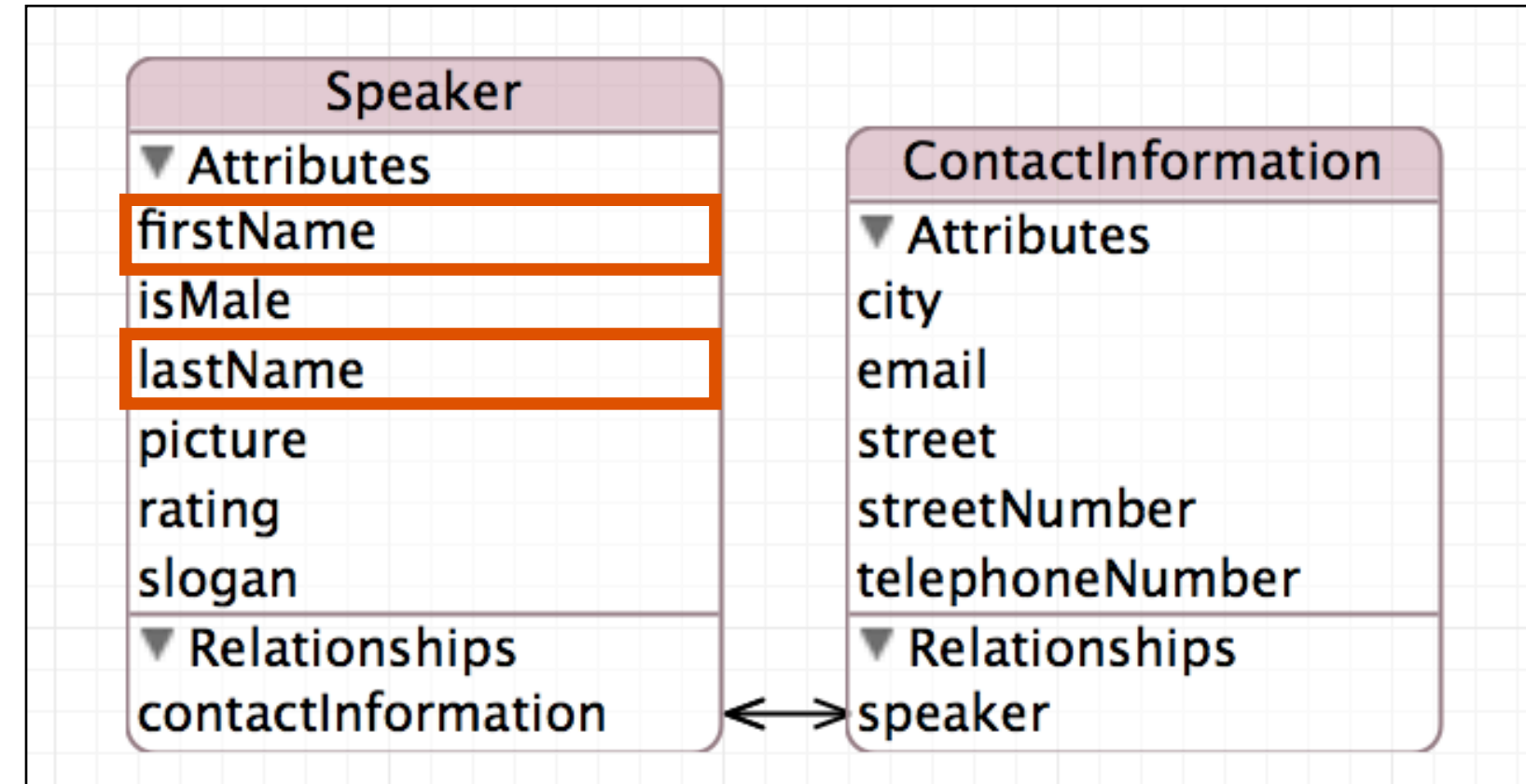
aus:

name = Christian Kienle

wird:

firstName = Christian

lastName = Kienle



Version 5

Demo

# Interface

```
#import <Cocoa/Cocoa.h>

@interface SpeakerToSpeakerPolicy : NSEntityMigrationPolicy

@end
```



# Implementation 2

```
BOOL s = [super createDestinationInstancesForSourceInstance:source
           entityMapping:mapping
           manager:manager
           error:error];

if(!s) {
    return NO;
}
```

# Implementation 3

```
NSArray *sourceArray = [NSArray arrayWithObject:source];
NSArray *d = [manager destinationInstancesForEntityMappingNamed:[mapping name]
                                                    sourceInstances:sourceArray];

NSManagedObject *destinationInstance = [d lastObject];

NSString *name = [source valueForKey:@"name"];
NSString *firstName = @"";
NSString *lastName = @"";
NSRange rangeOfSpace = [name rangeOfString:@" "];
```

# Implementation 4

```
if(rangeOfSpace.location == NSNotFound) {  
    firstName = name;  
}  
else {  
    firstName = [name substringToIndex:rangeOfSpace.location];  
    lastName = [name substringFromIndex:rangeOfSpace.location + 1];  
}  
  
[destinationInstance setValue:firstName forKey:@"firstName"];  
[destinationInstance setValue:lastName forKey:@"lastName"];  
  
return YES;  
}
```

# Zusammenfassung

- Lightweight Migration ist toll, sehr performant (Migration direkt auf der Datenbank)
- Lightweight Migration gut während der Entwicklung
- Komplexere Migrationen mit Mapping Models
- Flexibilität durch eigene Policies



Vielen Dank.  
Fragen?

*Und Piggeldy ging mit Frederick nach Hause.*

Show Differences

Source Model: V4.xcdatamodel

Destination Model: V5.xcdatamodel

### Entity Mappings

#	Mapping Name
1	SpeakerToSpe.
2	SpeakerToCon

### Property Mappings

T	Name	Value Expression
R	contactInformation	FUNCTION(\$manager, "destinationInstancesForEntityMappingNamed:sourceInstances:", "Spe.
A	isMale	\$source.isMale
A	name	\$source.name
A	picture	\$source.picture
A	rating	\$source.rating
A	slogan	\$source.slogan




# FUNCTION(...)?

```
FUNCTION($manager,  
"destinationInstancesForEntityMappingNamed:sourceInstances:" ,  
"SpeakerToContactInformation", $source)
```

Show Differences

Source Model:  V4.xcdatamodel



Destination Model:  V5.xcdatamodel



#### Entity Mappings

#	Mapping Name
1	SpeakerToSpe.
2	SpeakerToCon

#### Property Mappings

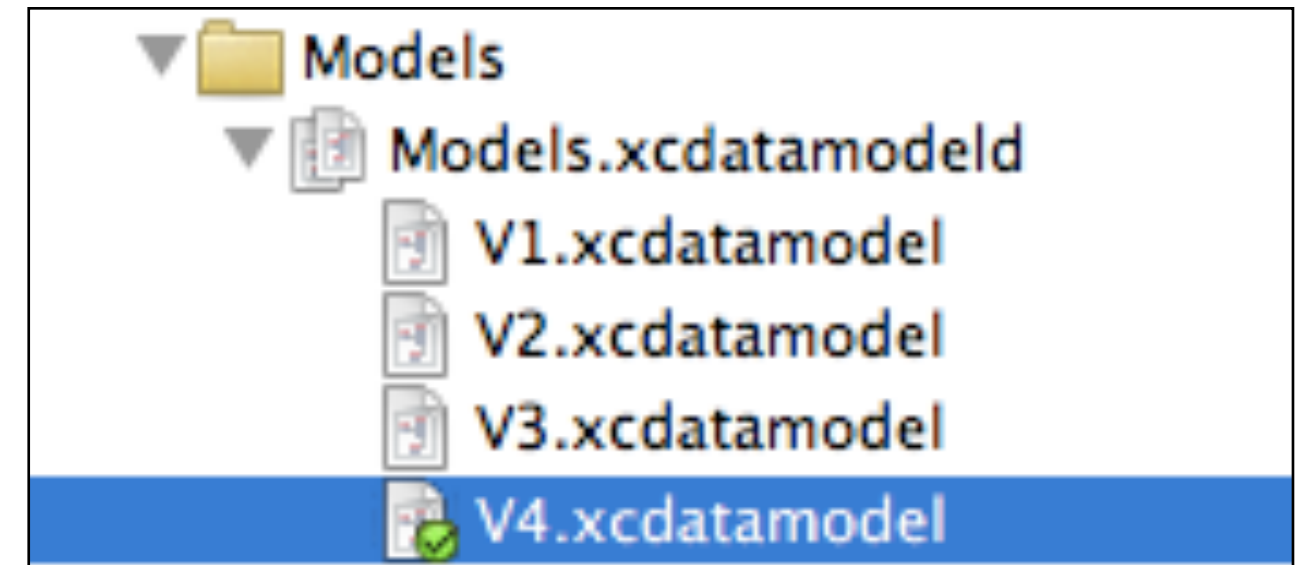
T	Name	Value Expression
R	contactInformation	FUNCTION(\$manager, "destinationInstancesForEntityMappingNamed:sourceInstances:", "Spe.
A	isMale	\$source.isMale
A	name	\$source.name
A	picture	\$source.picture
A	rating	\$source.rating
A	slogan	\$source.slogan

# Vorsicht!

- Während der Migration werden Klassen der Entitäten auf NSObject gesetzt.
- [super ...] aufrufen!

# Vorsicht!

- Modelversionen haben keine Reihenfolge
- V1, V2, ..., Vn ist für uns klar, für Core Data nicht
- M. Zarra: *Progressive Migration* (Core Data, Pragmatic Press)



Fragen?

**Vielen Dank**





**Macoun'10**