

Voyage by example

tips and tricks on persisting object models

Esteban Lorenzano

Pharo core developer

INRIA - RMoD

<http://smallworks.eu>

Why?

- You already know about Voyage
- You already attended to a tutorial last year
- But there are some recurrent problems people find when trying to use it

Review: What is Voyage?

- Abstraction layer to map objects into a database
 - Very well suited for document databases, but **in theory**, the approach will work for other kind of repositories
 - ▶ There was (long time ago) a Voyage-GLORP backend
 - ▶ There was (even more time ago) a Voyage-ImageSegment backend
- Voyage-Memory
- Voyage-Mongo

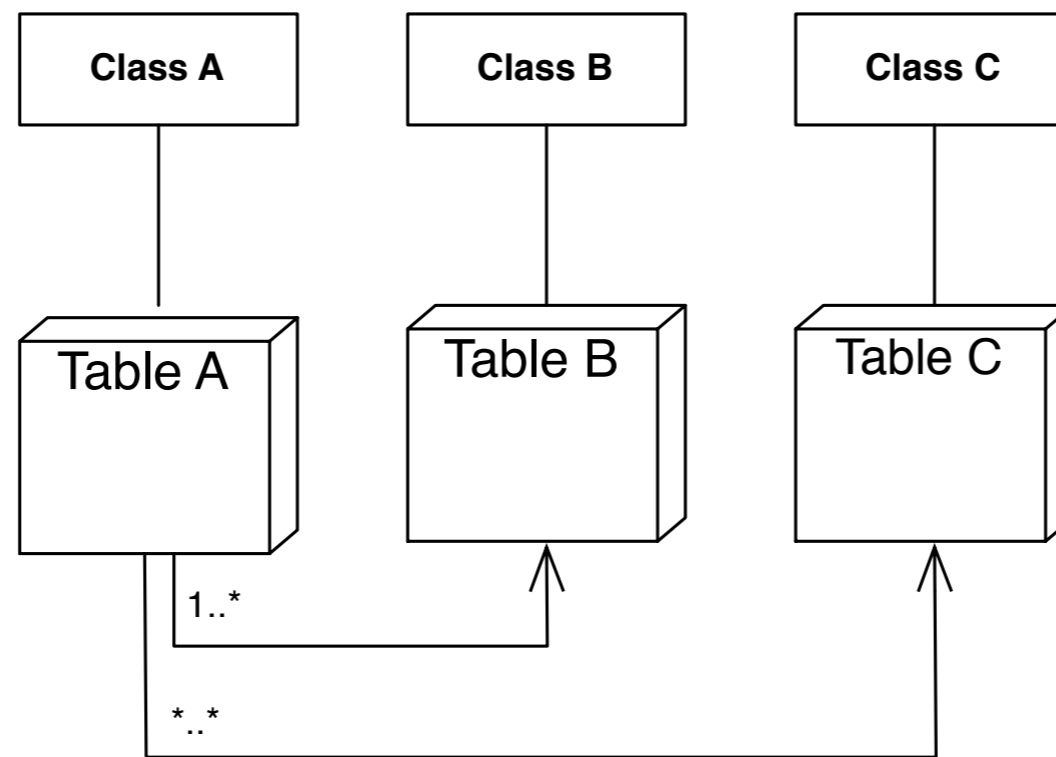
Review: Voyage principles

- Behavioural complete (for common usage), but decoupled approach also possible.
- Same design for different backends, but not a *common abstraction*
 - There is no such thing as a “voyage query language”, etc.
 - is a bit more work for users who want to switch, but a lot more happiness for the program itself

So, what about those tips?

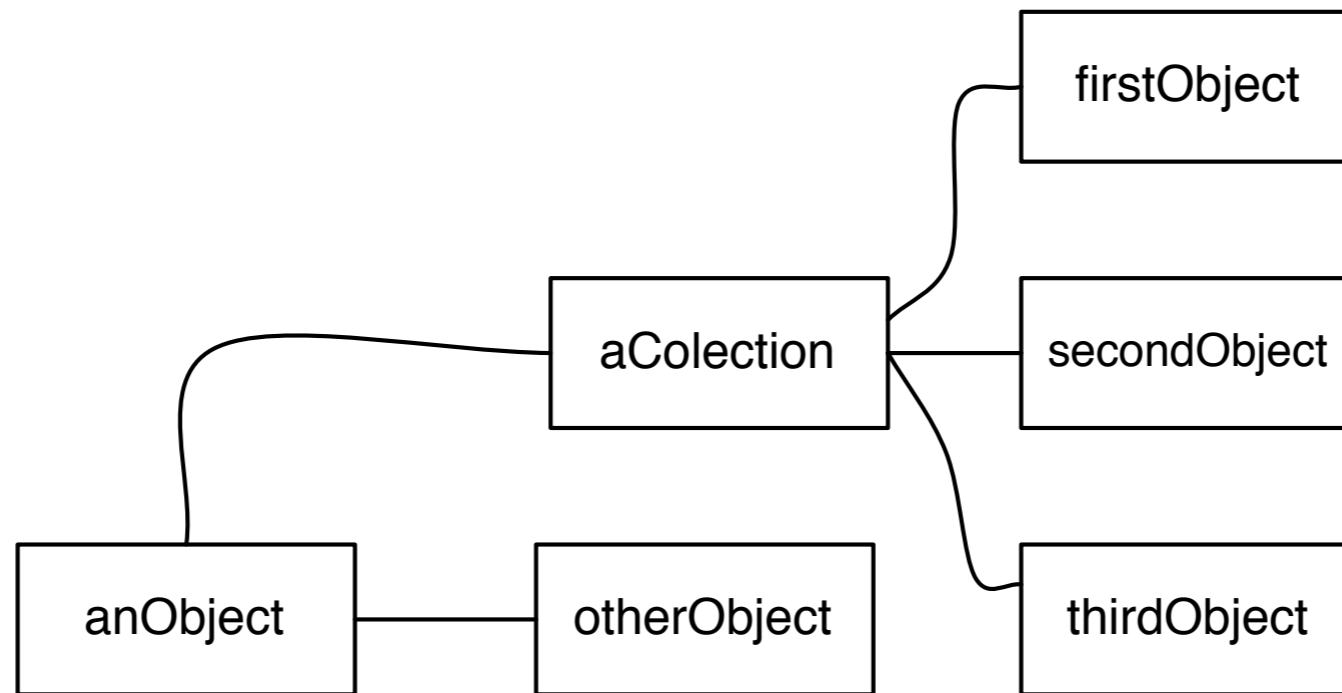
Think in objects

Impedance mismatch



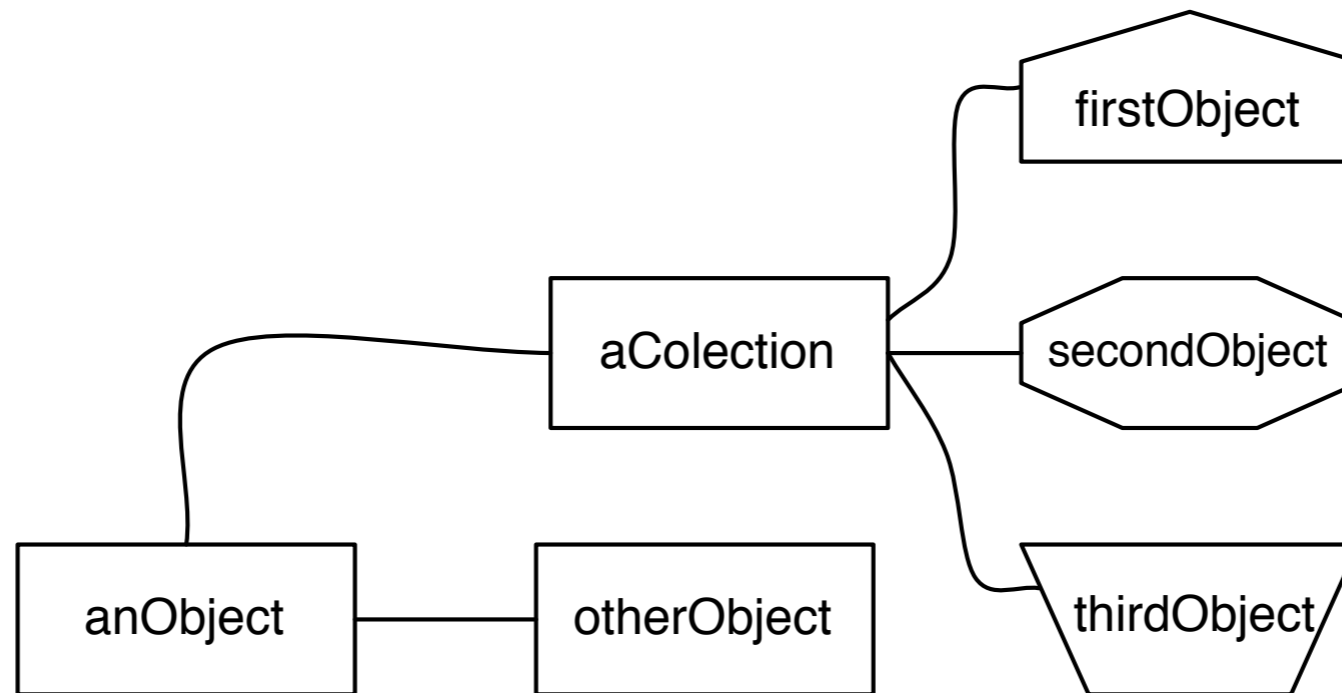
Ideal relational model

Impedance mismatch



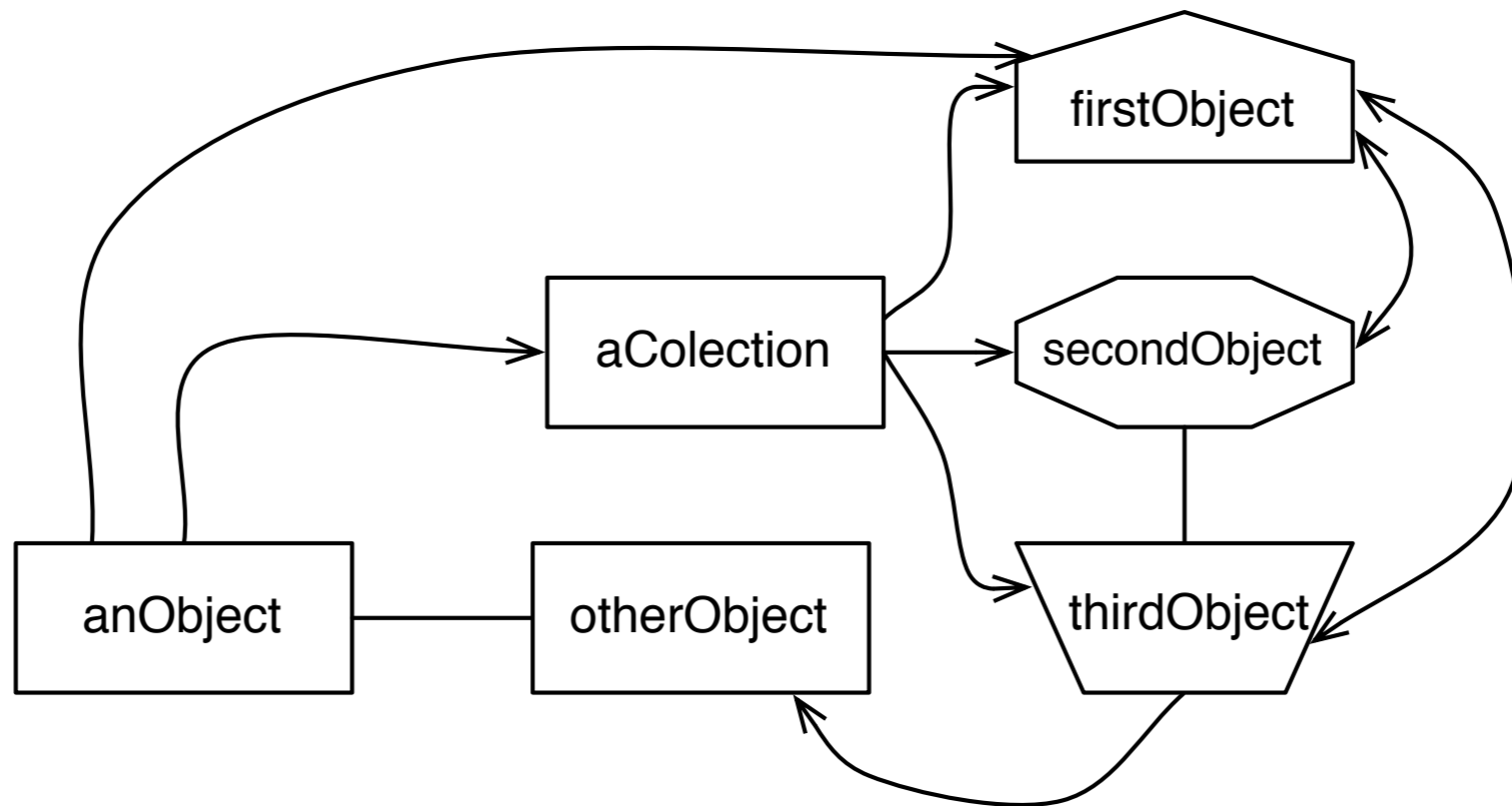
Reality object model

Impedance mismatch



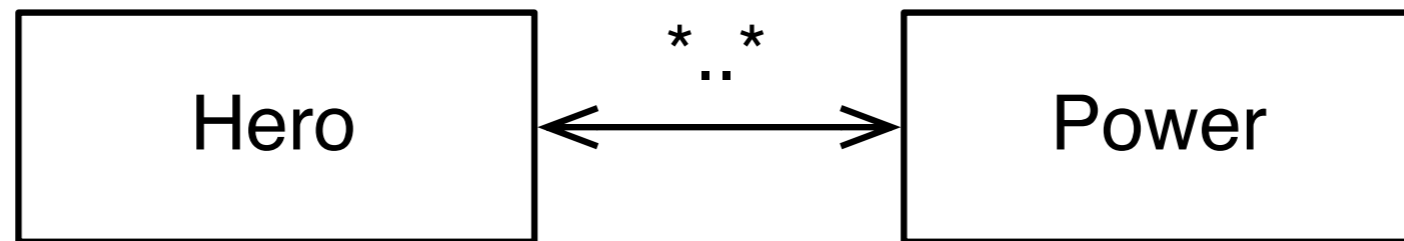
Reality object model

Impedance mismatch



Reality object model

A simple model



Persist

```
(Hero named: 'Groot')  
  addPower: ((Power named: 'Plant Control')  
    level: #epic;  
    yourself);  
save.
```

No need of more complications!

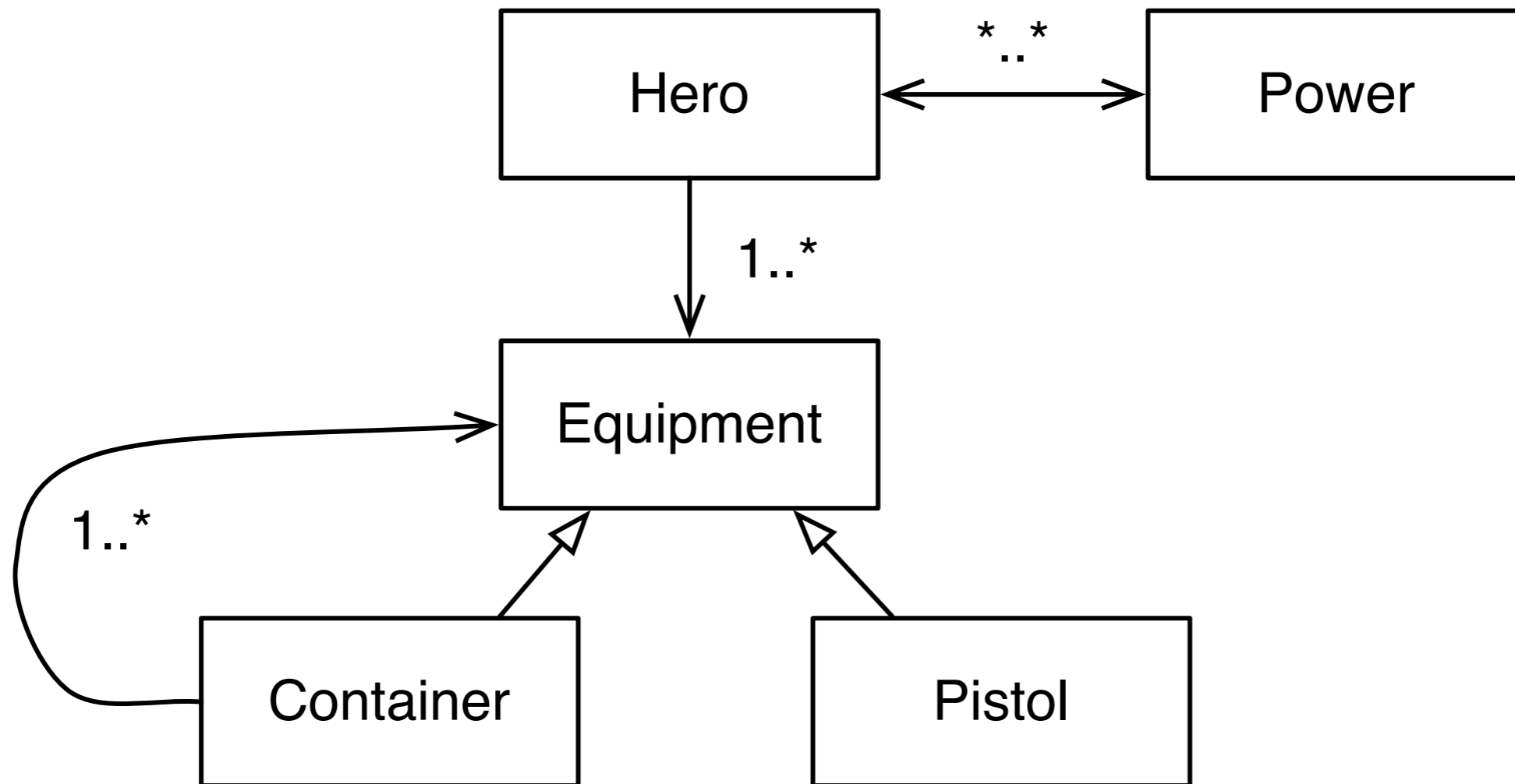
Persist

```
{
  _id: OID(...),
  #version: ...,
  #instanceOf: 'Hero',
  name: 'Groot',
  powers: [ { #collection: 'Power', __id: OID(...) } ]
}

{
  _id: OID(...),
  #version: ...,
  #instanceOf: 'Power',
  name: 'Plant Control',
  level: #epic,
  heroes: [ #collection: 'Hero', __id: OID(...) ]
}
```

Take control

A simple model (a bit more complete)



Persist

```
(Hero named: 'Star-lord')  
  addEquipment: (Container  
    addItem: Pistol new;  
    yourself);  
save.
```

No need of more complications!

Persist (1)

```
{
  _id: OID(...),
  #version: ...,
  #instanceOf: 'Hero',
  name: 'Star-lord',
  powers: [],
  equipment: [ {
    #instanceOf: 'Container',
    'items', [
      { #instanceOf: 'Pistol' } ] ] ]
}
```

Persist (2)

```
{
  _id: OID(1),
  #version: ...,
  #instanceOf: 'Hero',
  name: 'Star-lord',
  powers: [],
  equipment: [ { #collection: 'Equipment', __id: OID(2) } ]
}

{
  _id: OID(2),
  #version: ...,
  #instanceOf: 'Container',
  items: [ { #collection: 'Equipment', __id: OID(3) } ]
}

{
  _id: OID(3),
  #version: ...,
  #instanceOf: 'Pistol',
}
```

Integrity is a
consequence

Allowing missing content

- We do not have foreign keys
 - So we cannot do things like “ON DELETE CASCADE”
 - Even delete validations are difficult
 - ▶ Imagine “hero” has a “power”, and I remove the “power”. How can the hero notice it?

Persist

```
mongoContainer  
  <mongoContainer>  
  
  ^ VOMongoContainer new  
    collectionName: 'powers';  
    enableMissingContent;  
    yourself
```

Querying

Query (1)

Hero

```
selectMany: [ :each | ... ]  
sortBy: { #name -> V00order ascending } asDictionary  
limit: 100  
offset: 100
```


Query (2)

Hero

```
selectMany: {  
  'name' -> {  
    '$regex': '^G.*'.  
    '$options': 'i'  
  } asDictionary  
} asDictionary
```

Adapt schemes

The “scheme is not mine” problem

- Yes, it happens :(
- You can use the Magritte-Voyage metamodel to read it anyway
- You can change the versioning model too

Use it today!

Gofer it

```
smalltalkhubUser: 'Pharo' project: 'MetaRepoForPharo30';  
configurationOf: 'VoyageMongo';  
loadStable.
```